# REFlash: Reducing Total Endurance Cost in Flash-Based Tiering via Cross-Tier Write Balancing

Sejun Kwon
Samsung Electronics
Hwaseong, Republic of Korea
Seoul National University
Seoul, Republic of Korea
sejun000@snu.ac.kr

Minwook Kim
Seoul National University
Seoul, Republic of Korea
ace@snu.ac.kr

Jin-Soo Kim
Seoul National University
Seoul, Republic of Korea
jinsoo.kim@snu.ac.kr

## Abstract

QLC SSDs offer high density and low cost per byte, but their limited write endurance remains a critical barrier to widespread deployment. Flash-based tiering mitigates this limitation by placing a higher-endurance SSD as a buffer tier in front of a QLC capacity tier. However, existing schemes fail to jointly optimize write-induced wear across tiers: in-place-update schemes incur high device-level write amplification, while log-structured schemes leave the trade-off between buffer-tier garbage collection (GC) and capacity-tier flush writes unoptimized.

This paper presents REFlash, a cost-aware log-structured buffering scheme for SSD-SSD tiered storage. REFlash introduces *Total Endurance Cost (TEC)*, a unified metric that captures the asymmetric replacement cost across the buffer and capacity tiers, and uses it to dynamically choose between host-level GC and flush during segment reclamation. REFlash further reduces GC overhead by exploiting the bounded lifetime of buffered data. Implemented as a block device driver on an FDP-enabled TLC SSD and a QLC SSD, REFlash reduces TEC by up to 31% over existing log-structured tiering schemes while maintaining write throughput comparable to the best-performing baseline.

***CCS Concepts:*** • **Information systems → Hierarchical storage management**; *Flash memory*.

***Keywords:*** flash-based tiering, SSD endurance, write amplification

**ACM Reference Format:**
Sejun Kwon, Minwook Kim, and Jin-Soo Kim. 2026. REFlash: Reducing Total Endurance Cost in Flash-Based Tiering via Cross-Tier Write Balancing. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026), November 15–18, 2026, Hong Kong, China.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3828523.3857073



This work is licensed under a Creative Commons Attribution 4.0 International License.

*SIGOPS ATC 2026, Hong Kong, China*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2818-1/2026/11
https://doi.org/10.1145/3828523.3857073

## 1 Introduction

The rise of the AI era has driven explosive growth across a wide range of data-intensive workloads, which now demand both high performance and large capacity from storage simultaneously. In response to these demands, QLC NAND SSDs [31, 45, 46] have emerged as an alternative, offering higher density and lower cost per byte than TLC SSDs, while also surpassing HDDs in both density and performance. Indeed, QLC SSDs are increasingly adopted as a primary storage medium in production environments [8, 32, 38, 57].

However, despite these advantages, QLC SSDs suffer from inherently low write endurance, making them vulnerable to write-intensive workloads [4, 27, 28, 61]. Under heavy write traffic, their lifespan can shorten rapidly, necessitating frequent device replacements [28, 41, 52]. As a result, although QLC drives provide excellent cost per byte, such shortened device lifetimes can offset these benefits from a total cost of ownership (TCO) perspective.

**Flash-based tiering**, traditionally used to absorb I/O destined for slower HDDs [3, 7, 19, 21, 24, 25, 50, 51, 54, 59], is increasingly being adopted to mitigate the write endurance challenges in QLC SSDs [8, 12, 37, 57, 66]. With QLC SSDs taking over the *capacity-tier* role from HDDs, a higher-endurance SSD (e.g., SLC, TLC) serves as a *buffer-tier* that absorbs write traffic, shielding the capacity tier from frequent writes. However, this protection comes at a non-negligible cost of writes to the buffer-tier SSD. Since both tiers are built on flash, writes to either tier consume finite endurance and ultimately incur device replacement. This makes it essential to consider **endurance costs** across the entire system rather than optimizing each tier in isolation.

Despite the need to jointly consider endurance costs across both tiers, existing flash-based tiering schemes either concentrate wear-out on a single tier or fail to account for how endurance cost is distributed between the two tiers. The **in-place-update scheme** [7, 19, 35, 36, 50, 59] manages buffer-tier space at block granularity. In-place overwrites and reuse of freed locations produce random write patterns, triggering device-level *garbage collection (GC)* and increasing buffer-tier endurance cost. The **log-structured scheme** [5, 10, 25, 26, 29, 30, 54, 63, 66] instead writes incoming data sequentially at segment granularity to reduce

device-level GC. However, reclaiming buffer-tier space requires relocating valid data, introducing *host-level GC* writes to the buffer-tier SSD and *flush* writes to the capacity-tier SSD. These two reclamation methods incur write costs on different tiers, and their ratio determines the endurance cost distribution across tiers. Yet no existing work jointly optimizes these competing write costs. We experimentally observe that representative schemes [36, 66] skew writes toward one tier, leaving the total endurance cost unoptimized.

We propose **REFlash** (**R**educing total **E**ndurance cost in **Flash**-based tiering), a new tiering architecture that jointly optimizes endurance costs across both tiers. REFlash adopts a log-structured approach that reclaims free space for incoming writes through host-level GC and flush. Because these two reclamation methods generate writes on tiers with different endurance characteristics and cost per capacity, REFlash introduces the Total Endurance Cost (TEC) to capture their combined cost. It then dynamically balances host-level GC and flush to minimize TEC. REFlash further reduces the buffer-tier endurance cost of host-level GC, by adopting a data placement strategy that exploits the finite lifetime of buffered data, bounded by flushing. This strategy is physically realized on an FDP SSD [33, 47], an emerging class of SSDs that enables the host to control data placement for physical separation within the device. Evaluation using both micro- and macro-benchmarks shows that REFlash reduces TEC by up to 31% over existing log-structured tiering schemes, without sacrificing write throughput.

This paper makes the following contributions:

- We propose a runtime TEC-aware decision model that chooses between GC and flush by weighing buffer-tier GC costs against expected flush savings. GC can delay flushing, giving host writes more time to invalidate buffered blocks and reducing the data that would otherwise be flushed.
- We design a bounded-lifetime data placement strategy that enables more accurate remaining-lifetime prediction of buffered data, reducing host-level GC overhead.
- We implement and evaluate REFlash on an FDP-enabled TLC SSD and a QLC SSD, demonstrating that REFlash effectively balances endurance cost across both tiers and mitigates host-level GC overhead.

## 2 Background

### 2.1 QLC SSDs

Quad-level cell (QLC) NAND [31, 45, 46] stores four bits per cell, the highest density commercially available in current NAND flash technology. However, increasing the number of bits per cell substantially reduces endurance. As the number of voltage states per cell grows, the threshold-voltage window allocated to each state becomes narrower, and cells require more program/verify iterations. These tighter margins make QLC more vulnerable to program/erase stress, read disturb, and retention errors [6, 39, 40]. Endurance is typically specified in drive writes per day (DWPD) [16], which represents how many times the entire drive capacity can be rewritten per day over a standard warranty period (usually five years). For example, a device rated at 1 DWPD guarantees that the drive can receive writes equivalent to its full capacity every day for five years. With increasing density, DWPD drops sharply across generations: SLC can provide 100 DWPD, TLC offers 1–3 DWPD, and QLC delivers 0.1–0.3 DWPD [13–15, 20, 23, 31, 46, 48, 53, 60]. Thus, although QLC SSDs are highly cost-efficient, their limited durability makes them unsuitable for write-intensive workloads without additional buffering.

Commercial QLC SSDs also integrate a small internal SLC region to improve write performance and endurance; however, density and cost constraints limit the size of this region, leaving the overall device-level DWPD still lower—typically one-third to one-thirtieth that of TLC SSDs [14, 20, 31, 46, 48, 60]. As a result, additional system-level techniques are required to effectively mitigate write pressure when QLC SSDs are used in modern storage hierarchies.

### 2.2 Flexible Data Placement (FDP) SSDs

Conventional SSDs manage host-written data internally, where blocks with different lifetimes are mixed within the same erase unit. When the device needs to reclaim an erase unit, valid blocks that have not yet been invalidated must be relocated to another erase unit, causing device-level GC and additional writes invisible to the host. Flexible Data Placement (FDP) [2, 33] is a feature ratified as part of the NVMe specification that allows the host to specify a Reclaim Unit Handle (RUH) on each write, controlling which Reclaim Unit (RU) the data is placed in. When the host directs data with similar lifetimes to the same RUH, blocks within each RU are invalidated at approximately the same time, allowing the entire RU to be reclaimed without relocating valid blocks, thereby eliminating device-level GC writes.

### 2.3 Flash-Based Tiering

Traditional flash-based tiering architectures place a flash device as a buffer tier in front of capacity-tier devices such as HDDs. More recently, QLC SSDs have increasingly been deployed as the capacity tier, paired with a higher-endurance SSD buffer tier to form an SSD–SSD hierarchy. QLC SSDs offer read performance comparable to higher-endurance SSDs, as both are bounded by the host interface. However, their write endurance is significantly lower. Therefore, the hierarchy has been primarily used to buffer writes rather than to cache reads [8, 12, 37, 57, 66].

Flash-based tiering can be broadly classified into **in-place-update** and **log-structured schemes**, depending on how new blocks are written and old blocks are replaced in the buffer tier. Figure 1 illustrates the difference between the

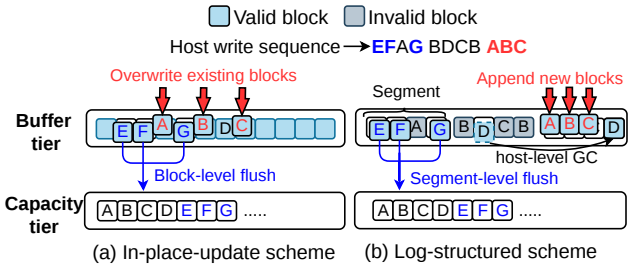



**Figure 1.** Host writes to the buffer tier and space reclamation in (a) in-place-update and (b) log-structured tiering schemes. Letters denote host addresses. Capacity-tier layouts are illustrative.

two approaches. We refer to the address requested by the application as the *host address*.

**In-place-update schemes** [7, 19, 35, 36, 50, 59] allocate free space on the buffer tier at block granularity for incoming writes (Figure 1(a)). When previously written host addresses (e.g., A, B, C) are written again, the scheme overwrites the same block, incurring random writes. If free space becomes insufficient, blocks are flushed to the capacity tier at block granularity to reclaim space, where the evicted blocks may be scattered (e.g., E, F, G) depending on their hotness. Subsequent incoming writes then reuse these scattered free blocks, producing random write patterns that can trigger device-level GC and accelerate wear-out of the buffer-tier SSD.

In contrast, **log-structured schemes** [5, 10, 25, 26, 29, 30, 54, 63, 66] allocate free space for incoming writes at segment granularity (Figure 1(b)), ensuring sequential host writes within each segment. When host addresses such as A, B, and C are overwritten, new blocks are allocated from the current segment, while old copies are invalidated. Because segment sizes are typically aligned with the SSD's internal erase unit size, host-level segment reclamation invalidates a physically contiguous region at once, thereby mitigating device-level GC [10, 22, 64–66]. However, as log-structured schemes write data out of place, invalidated copies remain in the device's address space until their segments are reclaimed.

Existing log-structured schemes differ in their segment reclamation policies. CSAL [66] reclaims segments solely by flushing valid data at segment granularity, as illustrated in Figure 1(b). Bcache [56] evicts dirty blocks individually, but performs host-level GC at segment granularity, selecting the segment with the fewest valid blocks. FairyWREN [30] combines segment-granularity eviction with host-level GC to obtain free segments.

# 3 Motivation

In this section, we examine how write overhead is distributed across tiers and motivate the need for a unified cost-aware approach to minimizing system-wide endurance cost.

We experimentally measure the write overhead imposed on the buffer-tier and capacity-tier SSDs under both in-place-update and log-structured schemes. As representative state-of-the-art implementations, we use OpenCAS [36] for the in-place-update scheme and CSAL [66] for the log-structured scheme. CSAL reclaims free space only through flush. We also evaluate CSAL+GC, an extended version of CSAL that performs host-level GC by selecting the segment with the fewest valid blocks as the victim. Using a real Alibaba Cloud block I/O trace [27, 58], we measure host-level writes and device-level writes (i.e., physical writes to the underlying flash) on the SSDs in both tiers, all on real hardware. The detailed experimental setup is described in §7.1.

## 3.1 Problems of Existing Schemes

**Device-level write amplification of in-place-update scheme.** Figure 2(a) compares the device-level WA (Write Amplification)—the ratio of physical NAND writes to host writes—of OpenCAS, CSAL, CSAL+GC, and REFlash on the buffer-tier SSD. OpenCAS manages free space at 4KB block granularity for incoming writes. This fine-grained allocation and reclamation lead to frequent device-level GC, resulting in a device-level WA of 3.35. Consequently, OpenCAS incurs substantial write-induced wear on the buffer tier. By contrast, log-structured variants (CSAL, CSAL+GC, and REFlash) reclaim free space at segment granularity, which preserves sequential writes and mitigates device-level WA.

**Cross-tier write overhead in log-structured schemes.** However, log-structured schemes add two sources of write overhead: (1) flushing data to the capacity tier and (2) migrating valid data during host-level GC on the buffer tier.

Figure 2(b) shows the write volume to the capacity-tier SSD for each tiering approach. OpenCAS keeps all buffer-tier blocks valid, allowing the buffer tier to absorb more writes and reducing writes to the capacity tier. In contrast, CSAL reclaims segments only by flushing, leaving invalidated blocks in their segments until reclamation. This reduces the number of valid blocks retained in the buffer tier, lowers its write-absorption capability, and increases writes to the capacity tier by 3.3× compared with OpenCAS.

Although CSAL+GC reduces some flush overhead through host-level GC, Figure 2(c) shows that it lowers *total endurance cost (TEC)* (§4.1) by only 7% relative to CSAL, because the additional buffer-tier writes from host-level GC offset much of the flush savings. TEC captures the SSD replacement cost due to write-induced wear across both tiers (§4.1). REFlash achieves the lowest TEC by balancing host-level GC and flush volumes at runtime (Figure 2(c)). These results demonstrate that minimizing writes on a single tier is insufficient; endurance cost must be considered jointly across both tiers.

## 3.2 Challenges of Cross-Tier Write Balancing

**Competing writes: host-level GC vs. flush.** In log-structured schemes, host-level GC writes to the buffer-tier SSD and flush

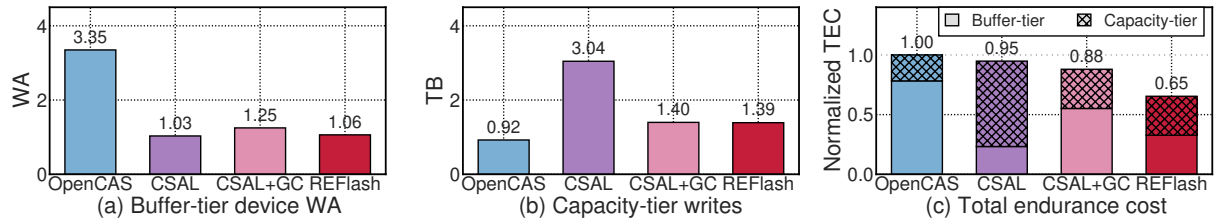



**Figure 2.** Device write amplification, capacity-tier writes, and total endurance cost (TEC) of in-place update (OpenCAS), log-structured flush-only (CSAL), log-structured flush+GC (CSAL+GC), and REFlash.

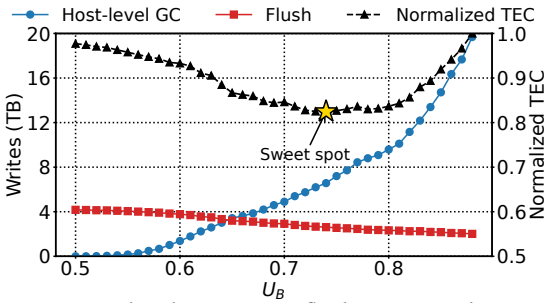



**Figure 3.** Host-level GC writes, flush writes and normalized TEC (normalized to the maximum) as a function of buffer-tier utilization ($U_B$), defined as the ratio of valid blocks to total buffer-tier capacity, under CSAL+GC.

writes to the capacity-tier SSD exhibit a trade-off relationship. Figure 3 shows the volume of host-level GC rewrites and flush writes as a function of $U_B$, defined as the ratio of valid blocks to the total capacity of the buffer-tier SSD. We ran CSAL+GC at each target $U_B$, performing GC when the current $U_B$ was below the target and flushing otherwise. As $U_B$ increases, more host-level GC is required, which increases buffer-tier writes but reduces flush writes to the capacity tier. Conversely, a lower $U_B$ reduces host-level GC writes but increases flush writes. This observation shows that the two types of writes move in opposite directions as $U_B$ varies, suggesting the existence of a sweet spot that minimizes the total endurance cost across both tiers.

**Asymmetry in host-level GC and flush costs.** Notably, host-level GC rewrites are significantly more expensive than flush writes. Comparing the 70% and 80% utilization points in Figure 3, flush writes differ by only 0.6 TB, while host-level GC writes differ by 4.7 TB—approximately 7.8× larger. This asymmetry implies that increasing $U_B$ incurs a relatively high endurance cost on the buffer tier, while the corresponding reduction in capacity-tier endurance cost is comparatively small. Therefore, reducing host-level GC overhead is critical to minimizing the overall endurance cost.

Prior data placement techniques for reducing host-level WA in log-structured storage [18, 34, 55, 58] can help mitigate this imbalance. However, these techniques were designed for single-device storage and do not consider that flushed blocks are invalidated upon eviction, which bounds their lifetime in the buffer tier. This bounded-lifetime property creates an opportunity to further optimize data placement, reduce host-level WA, and ultimately lower the total endurance cost.

**Table 1.** Endurance cost ratio ($r$) for representative buffer/capacity tier configurations. DWPD values reflect commercially available enterprise SSDs [13–15, 20, 23, 46, 48, 53, 60].

| Buffer | Capacity | $n_b$ | $n_c$ | $\text{DWPD}_b$ | $\text{DWPD}_c$ | $r$ |
|---|---|---|---|---|---|---|
| SLC | QLC | 1 | 4 | 100 | 0.26 | 96 |
| MLC | QLC | 2 | 4 | 3~10 | 0.26 | 5.76~19.2 |
| TLC | QLC | 3 | 4 | 1~3 | 0.26 | 2.88~8.64 |

## 4 Cross-Tier Cost-Aware Reclamation

In this section, we define an endurance cost model that accounts for write overhead across both tiers. Based on this model, we develop a cost-aware segment reclamation decision policy that selects either host-level GC or flush to minimize the overall endurance cost.

### 4.1 Endurance Cost Model

We define an endurance cost model that captures the per-byte write cost of each tier. Because the two tiers differ in both endurance and cost per GB, we normalize their endurance cost on a per-unit-write basis and use their sum as the optimization criterion.

We build on an established SSD cost model [30] and interpret it as the device replacement cost incurred by writes. We extend this model to a two-tier setting by aggregating the per-tier costs into a single metric. The total write endurance of an SSD is characterized by its Total Bytes Written (TBW):

$$TBW = DWPD \times Cap \times Y \times 365 \quad (1)$$

where $DWPD$ is the drive writes per day, $Cap$ is the device capacity, and $Y$ is the guaranteed lifetime in years.

When an SSD wears out, it must be replaced at a cost proportional to its price per capacity. We normalize the price per gigabyte of each tier based on bit density: a flash cell storing $n$ bits provides $n$ times the density of a single-level cell. Thus, we model the normalized cost per byte as:

$$P = \frac{\alpha}{n} \quad (2)$$

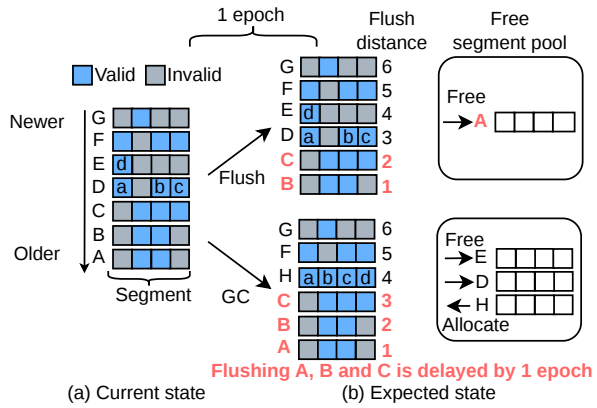



**Figure 4. Expected segment layouts under flush and host-level GC in the buffer tier.** Compared with flush, host-level GC delays the flushing of segments A, B, and C by one epoch, allowing host writes to invalidate more blocks in these segments and thereby reducing the flush volume.

where $n$ is the number of bits per cell (e.g., 3 for TLC, 4 for QLC) and $\alpha$ is the per-byte cost of a single-level cell.

Given these definitions, the per-byte endurance cost for each tier—the replacement cost incurred per byte written—is:

$$c_i = \frac{P_i \times Cap_i}{TBW_i} = \frac{P_i}{DWPD_i \times Y_i \times 365} \tag{3}$$

where the subscript $i \in \{b, c\}$ denotes the buffer tier and the capacity tier, respectively.

The total endurance cost (TEC) of the system over a given period is then:

$$TEC = c_b \times W_b + c_c \times W_c \tag{4}$$

where $W_b$ and $W_c$ denote the total physical NAND bytes written to the buffer-tier and capacity-tier SSDs, respectively, including writes caused by device-internal GC. TEC thus estimates the SSD replacement cost associated with the wear caused by these writes. To reason about the relative cost of writing to each tier, we define the *endurance cost ratio*. Because the two tiers typically have the same warranty period ($Y_b = Y_c$) [13–15, 20, 31, 48, 53], the lifetime terms cancel out:

$$r = \frac{c_c}{c_b} = \frac{P_c}{P_b} \times \frac{DWPD_b}{DWPD_c} = \frac{n_b}{n_c} \times \frac{DWPD_b}{DWPD_c} \tag{5}$$

Table 1 lists the endurance cost ratios for representative tier configurations. This ratio, $r$, is the key parameter REFlash uses to balance host-level GC and flush (§4.2).

### 4.2 Run-time TEC-Aware Decision Model

REFlash adopts a log-structured buffer tier to keep device-level WA close to one and control both GC and flush at the host level. Its runtime decision model balances these reclamation methods by comparing their estimated endurance costs using the TEC model. The key insight is that host-level GC can delay flushing, giving host writes more time to invalidate buffered blocks and thereby reducing future capacity-tier writes. REFlash chooses GC when these expected savings exceed the endurance cost of GC writes to the buffer tier.

**Separate victim-selection policies.** REFlash reclaims buffer-tier space at segment granularity. A segment close to being flushed can be a poor GC victim because its copied blocks may be flushed before subsequent host writes invalidate them. REFlash therefore selects victims separately for GC and flush.

For flush, we retain CSAL's segment-level FIFO policy [66], which evicts the least recently written segments first and retains more recently written data to absorb subsequent overwrites. This ordering approximates a block-level least recently used (LRU) policy based on host-write recency [36]. REFlash builds on log-structured tiering that uses this flush policy. It adds host-level GC as an additional reclamation method and uses a runtime decision model to choose between GC and flush. For GC, REFlash uses the warm-aware policy to avoid selecting segments that will soon be flushed. The rationale for these different victim-selection policies is detailed in §5.2.

**How GC reduces flush writes.** When free space runs low, REFlash must reclaim space for incoming host writes. REFlash compares the effects of GC and flush from the same current state in Figure 4(a). We define an *epoch* as the interval during which the buffer tier absorbs one segment's worth of host writes. GC and flush each reclaim one net free segment to accommodate these writes. After one epoch, the system reaches the corresponding expected state in Figure 4(b).

Under flush, shown in the upper half of Figure 4(b), the valid blocks in segment A are written to the capacity tier, freeing A in the buffer tier. Under GC, shown in the lower half, suppose the GC policy selects D and E as victims. Their valid blocks are copied into H, which is allocated from the free segment pool. D and E are then returned to the pool. GC thus allocates one segment and releases two, yielding one net free segment while retaining A.

We use *flush distance* to denote the number of epochs before a segment is flushed in this comparison. In the flush case, B and C have flush distances of 1 and 2. Under GC, H replaces D and E between C and F in flush order. A remains in the buffer, and A, B, and C now have flush distances of 1, 2, and 3. These segments precede the GC victims in flush order and gain one extra epoch before flushing. F and G retain the same flush distances. Keeping A adds one segment ahead of them, while replacing D and E with H reduces that count by one. The delay therefore applies to A, B, and C rather than to every segment remaining in the buffer. During this extra epoch, host overwrites can invalidate blocks in the delayed segments that would otherwise be flushed to the capacity tier. These additional invalidations constitute the flush savings that REFlash weighs against GC's buffer-tier write cost.

**Decision model.** We translate the above effect into a TEC-based comparison between the cost of performing host-level GC and the capacity-tier writes that GC is expected to save.

The cost of GC is the amount of valid data copied to reclaim one net free segment. To estimate this cost, let $S_v$ denote the set obtained by traversing the GC victim list in its existing order and including segments until their combined invalid blocks total at least one segment's worth (e.g., segments D and E in Figure 4): $\sum_{i \in S_v} \big(B - v(i)\big) \geq B$, where $B$ is the number of blocks per segment, and $v(i)$ is the number of valid blocks in segment $i$. The valid blocks copied during this process are rewritten to the buffer-tier device, whose device-level WA is denoted $A_b$. The GC cost is then:

$$Cost_{gc} = A_b \sum_{i \in S_v} v(i) \tag{6}$$

The benefit of GC comes from the segments whose flushes are delayed by choosing GC instead of flush. During the delayed epoch, blocks that become invalid in these delayed segments no longer need to be flushed to the capacity tier. Let $S_{old}$ denote the set of segments whose flush distance increases due to the GC decision, namely the segments older than the GC victim set in flush order (e.g., segments A, B, and C in Figure 4). Let $\Delta_{inv}(i)$ be the number of blocks in segment $i$ expected to be invalidated during the additional epoch for which flushing is deferred. These blocks would otherwise be flushed to the capacity tier, whose device-level WA is denoted $A_c$. Recall that $r = c_c/c_b$ denotes the ratio of capacity-tier to buffer-tier per-NAND-write cost. The benefit, expressed in buffer-tier cost units, is then:

$$Benefit_{gc} = A_c\, r \sum_{i \in S_{old}} \Delta_{inv}(i) \tag{7}$$

Given these definitions, REFlash performs GC when the expected benefit exceeds the cost,

$$Cost_{gc} < Benefit_{gc} \tag{8}$$

Otherwise, REFlash performs flush.

In our implementation, $\sum_{i \in S_v} v(i)$ is measured by traversing the current GC victim list and counting the valid blocks that must be copied until the reclaimed space reaches one segment. The term $\sum_{i \in S_{old}} \Delta_{inv}(i)$ is a future quantity, so it cannot be known at decision time. We therefore approximate it with the number of blocks invalidated in the delayed segments over the previous epoch. Both quantities are measured per epoch and smoothed across epochs using an exponentially weighted moving average (EWMA) [42], reducing the impact of transient fluctuations on the reclamation decision. At each epoch, REFlash updates each estimate as $\hat{x}_t = (1-\alpha)\hat{x}_{t-1} + \alpha x_t$, where $x_t$ is the current epoch's observation and $\hat{x}_t$ is the smoothed estimate. We fix $\alpha = 0.5$, so the weight of each past observation halves every epoch. The device-level WA of each tier ($A_b$ and $A_c$) is obtained from recently reported values: the FDP log on the buffer-tier SSD and the SMART log on the capacity-tier SSD [2, 33].

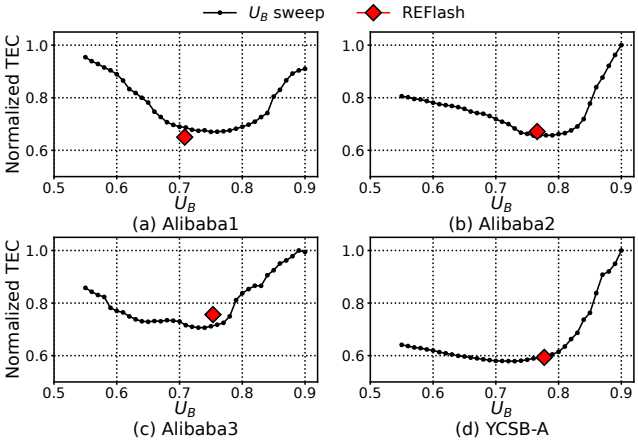



**Figure 5.** Normalized TEC over the $U_B$ sweep and REFlash.

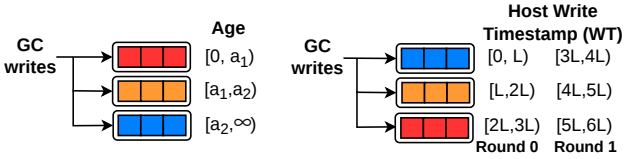



**Figure 6.** Comparison of GC data placement strategies.

Figure 5 compares REFlash with an exhaustive sweep over $U_B$ across four representative workloads, which are summarized in Table 2 and described in detail in §7.1. In Figure 5, each point is evaluated in the simulator with a fixed target $U_B$, defined as the ratio of valid blocks to the total buffer-tier capacity; $U_B$ is swept in 1% increments. At each point, the scheme performs GC when the current $U_B$ is below the target and flushes otherwise. REFlash's runtime decision model converges to the low-TEC region just before the curve rises sharply, achieving a TEC within 5% of the sweep minimum. This validates the decision model: without any prior offline analysis, REFlash approaches the offline-swept minimum using only runtime estimates of recent values. This shows that the model captures the key trade-off between buffer-tier GC cost and capacity-tier flush savings.

## 5 Buffer-Tier GC Optimization

REFlash improves the efficiency of host-level GC within the buffer tier itself. As shown in Figure 3, reclamation through host-level GC is substantially more expensive than flush, motivating dedicated optimization. REFlash mitigates this overhead with two techniques: bounded-lifetime grouping (§5.1) and warm-aware cost-benefit victim selection (§5.2).

### 5.1 Bounded-Lifetime Grouping

REFlash reduces buffer-tier endurance cost by lowering host-level GC overhead through improved data placement. Existing data placement techniques for single log-structured storage [34, 58] route blocks likely to be *overwritten* together into the same active segment, so that GC later finds few valid blocks to relocate. REFlash additionally exploits a key property of the buffer tier: flushing invalidates any valid blocks remaining in a segment, placing an upper bound on each block's lifetime from its host write to invalidation.

Prior approaches assign GC-migrated blocks to *GC groups* based on the hotness or age of host-written blocks [34, 58]. A GC group is a set of GC-migrated blocks managed with one dedicated active segment. These policies commonly group blocks according to each block's expected remaining lifetime. In single-device storage, however, remaining lifetimes can span a broad range. For example, a block written once and never overwritten may remain valid indefinitely. A limited number of GC active segments must therefore accommodate a potentially unbounded range of remaining lifetimes, so an individual group can cover a broad range. Figure 6(a) shows an age-based example in which the oldest group has an open-ended age range.

In contrast, flushing bounds the residence of data in the buffer tier. REFlash exploits this property to place GC-migrated blocks into active segments with narrower lifetime ranges. As older data is flushed, REFlash reuses the GC groups for successive ranges, as shown in Figure 6(b). This enables finer lifetime classification with a limited number of GC active segments. For host writes, REFlash uses two active segments to separate hot and cold blocks, following prior work [34, 58].

Algorithm 1 presents REFlash's data placement policy on the buffer-tier SSD. Each block maintains a host-write timestamp, $WT$, assigned from a monotonically increasing counter, $CT$, that tracks cumulative host writes. REFlash classifies a block as hot or cold based on its inter-write distance, defined as the gap between the current $CT$ and the block's previous $WT$, and directs the block to one of two host active segments accordingly. Each GC active segment is responsible for a specific range of $WT$ values. During GC, REFlash directs a migrated block to the GC active segment whose range contains the block's $WT$ (Figure 6(b)). The remainder of this section describes each path in detail.

**Host writes.** On a host write (OnHostWrite), REFlash classifies the block based on its inter-write distance: the block is classified as hot if the distance is below *hot_thr*, and cold otherwise (line 3). This classification and the threshold-update rule follow SepBIT [58]. After every 16 hot-segment reclamations, REFlash updates *hot_thr* to the mean age of these segments. Each segment's age is computed as the difference between $CT$ at reclamation and the oldest $WT$ in that segment. The threshold is therefore measured in host-block writes and requires no per-configuration tuning.

After writing the block to the buffer-tier device, REFlash records its write timestamp using $CT$ (line 14). When a block is flushed from the buffer tier to the capacity tier (OnFlush), its $WT$ entry is removed (line 19). This removal bounds the in-device lifetime of buffered blocks.

**GC writes.** On a GC write (OnGCWrite), REFlash determines the destination GC active segment based on the $WT$ of the block migrated from the victim segment. The host-write timestamp is preserved across GC migrations rather than reset, so that the block's classification reflects when its data was originally written by the host. Each GC active segment covers a lifetime range of width $L$, computed as the age range of valid blocks currently stored in the buffer-tier SSD divided by $N_{gc}$ (line 22). Because $L$ changes as the reclamation controller balances GC and flush, UpdateLifetimeRange is invoked once per epoch to recompute $L$. ClassifyGC then maps a block to a GC active segment as $idx = \lfloor WT[lba]/L \rfloor \bmod N_{gc}$ (line 7).

Because $L$ is updated only at epoch boundaries, the live $WT$ range may temporarily exceed $N_{gc} \cdot L$. In this case, the modular mapping may direct blocks from different rounds to the same GC active segment. When a GC group is reused for a new round, REFlash dummy-fills its current active segment, opens a fresh one, and excludes the previous-round segment from future GC victim selection. This mechanism reduces mixing across lifetime rounds. In our evaluation, dummy-fill writes account for 0.27% of host writes, and dummy-filled space occupies at most 0.35% of buffer-tier capacity.

By confining each segment to a narrow age range, REFlash groups blocks that are likely to be invalidated around the same time. As a result, segments tend to contain fewer valid blocks when selected for GC. We experimentally confirm this reduction in host-level GC overhead in §7.6.

### 5.2 Warm-Aware Cost-Benefit Victim Selection

REFlash further exploits the bounded-lifetime property of buffer-tier data when selecting GC victims. In conventional log-structured storage, blocks that have survived longer are likely to remain valid in the future. Therefore, the cost–benefit (CB) policy selects segments with the highest score, defined as $\frac{1-u}{u} \times \text{age}$ [43] or $\frac{1-u}{u} \times \sqrt{\text{age}}$ [9, 11, 34, 58], where $u$ is the fraction of valid blocks in the segment.

However, this assumption does not hold for a buffer-tier SSD. Because buffered data has a bounded lifetime, longer-lived blocks have less remaining lifetime and are more likely to be evicted, while the blocks in very young segments are likely to be overwritten soon.

REFlash therefore replaces the conventional CB score with a simple heuristic: score $= \frac{1-u}{u} \times \sqrt{\min(\text{age},\ T - \text{age})}$, where $T$ is the maximum age of any live block in the buffer tier. A segment's age is defined as the age of its oldest block. REFlash computes this score per segment and selects the highest-scoring segments as GC victims first. This policy

**Algorithm 1** REFlash data placement on the buffer-tier SSD.

$N_{gc}$: number of GC groups
$WT[lba] \leftarrow INVALID$
$gc_round[N_{gc}] \leftarrow 0$
1: **function** ClassifyHost($lba$)
2: **if** $WT[lba] = INVALID$ **then return** $COLD$
3: **if** $CT - WT[lba] < hot_thr$ **then return** $HOT$
4: **else return** $COLD$
5: **function** ClassifyGC($lba$)
6: $b \leftarrow \lfloor WT[lba] / L \rfloor$
7: $idx \leftarrow b \bmod N_{gc}$; $r \leftarrow \lfloor b / N_{gc} \rfloor$
8: **if** $gc_round[idx] < r$ **then**
9: DummyFill($idx$); $gc_round[idx] \leftarrow r$
10: **return** $idx$
11: **procedure** OnHostWrite($lba$)
12: $idx \leftarrow$ ClassifyHost($lba$)
13: WriteToHostSeg($idx$, $lba$)
14: $WT[lba] \leftarrow CT$; $CT \leftarrow CT + 1$
15: **procedure** OnGCWrite($lba$)
16: $idx \leftarrow$ ClassifyGC($lba$)
17: WriteToGCSeg($idx$, $lba$)
18: **procedure** OnFlush($lba$)
19: $WT[lba] \leftarrow INVALID$
20: **procedure** UpdateLifetimeRange( )
21: $m \leftarrow \min\{WT[lba] : WT[lba] \neq INVALID\}$
22: $L \leftarrow (CT - m) / N_{gc}$

avoids segments with short ages, whose blocks are likely to be rewritten soon after GC. It also deprioritizes segments with long ages, whose small $T$ − age indicates they are close to being flushed. As a result, REFlash preferentially selects warm segments—segments that are neither too young nor too close to eviction—as GC victims.

# 6 Design of REFlash

## 6.1 Architecture Overview

REFlash is a host-level block device driver that manages writes to both the buffer-tier SSD and the capacity-tier SSD. Figure 7 illustrates the main components of REFlash. REFlash follows the address-translation model commonly used in log-structured schemes [5, 10, 29, 30]. It manages buffer-tier placement through a hash map, while addressing the capacity-tier SSD directly with the host address, so that capacity-tier data is read and written in place.

REFlash maintains three metadata structures. First, a **hash map** translates host addresses to device addresses at 4KB block granularity; each entry also stores the block's write timestamp ($WT$). Second, a **block map** provides the reverse mapping from device addresses to host addresses. Third, a **segment map** tracks the $WT$ of the oldest block in each segment. Each segment is configured to align with the device's reclaim-unit granularity, which is 6 GB in our prototype. Metadata persistence and crash recovery follow standard log-structured techniques [66] and are orthogonal to REFlash's endurance optimization.

REFlash consists of three main modules that operate on these metadata structures. The **allocator** assigns free blocks and segments for host writes and GC writes. The **segment reclaimer** obtains free segments through either GC or flush. For GC, the victim list is implemented as a priority queue sorted by the warm-aware cost-benefit score. For flush, the victim list is implemented as a FIFO queue, so the oldest segment is evicted first. The **reclamation controller** estimates GC overhead from the GC victim list and tracks the number of blocks invalidated in each segment during every epoch to choose the reclamation method (§4.2). The segment reclaimer then executes the selected method.

REFlash directs host and GC writes to separate active segments (§5.1) and uses an FDP buffer-tier SSD to place these segments in physically separate regions, mitigating device-level GC.

## 6.2 Host I/O Paths

**Read path.** On a read, REFlash first looks up the host address in the hash map to determine whether the requested data resides in the buffer-tier SSD (1). On a hit, the hash map returns the corresponding device address, and REFlash reads the data from the buffer-tier SSD (2). On a miss, REFlash reads the data directly from the capacity-tier SSD using the host LBA, without address translation (3).

**Write path.** On a host write, the allocator allocates a new block from the active segment according to the data placement policy (§5.1). The policy looks up the host LBA in the hash map and uses the previous $WT$ to estimate the block's update frequency and determine its placement (1). If the allocation fills the active segment, the segment reclaimer inserts the segment into both victim lists, and the allocator obtains a new active segment from the free segment pool (2). REFlash then writes the data to the device address of the allocated block on the buffer-tier SSD (3).

At this point, each active segment is assigned a distinct RUH on the FDP SSD, ensuring that blocks of different segments are physically placed in separate reclaim units [33, 49]. The same mechanism can be realized equivalently by allocating a separate zone on a ZNS SSD, or by assigning a distinct stream ID on a multi-streamed SSD [2, 17, 60].

REFlash then updates the segment map, block map, and hash map (4). The $WT$ in the hash map is set to the cumulative number of host writes at the current moment (§5.1). The $WT$ in the segment map is updated only when the first block is written to the segment, so that it always reflects the $WT$ of the oldest block in the segment.

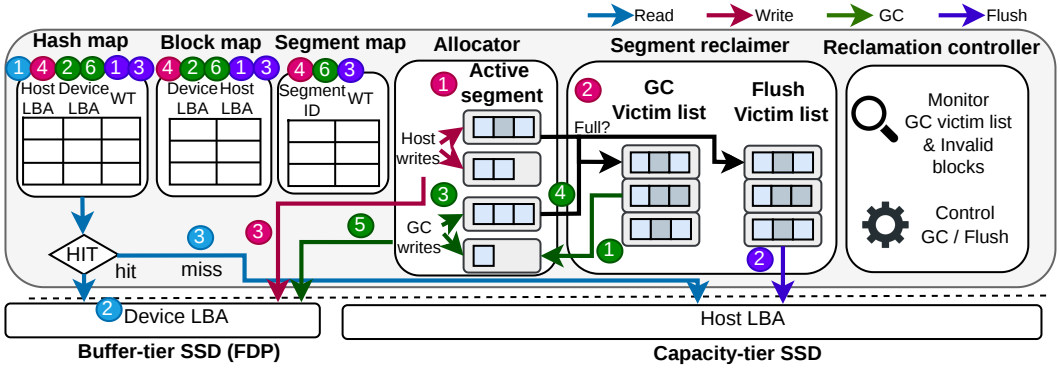



**Figure 7.** Overview of the REFlash architecture.

### 6.3 Reclamation Paths

**Garbage collection path.** When the free segment pool runs low, the segment reclaimer initiates reclamation. The reclamation controller determines whether to perform GC or flush. For GC, the segment reclaimer selects the segment with the highest warm-aware cost-benefit score from the GC victim list (1), and removes the corresponding pointer from the flush victim list.

For each block in the victim segment, REFlash determines whether the block is still valid by looking up its host LBA in the block map and checking whether the hash map entry for that host LBA still points to the block (2). Each valid block is then relocated to a block allocated by the allocator. The data placement policy (§5.1) consults the block's *WT* to select the appropriate active segment (3). If the active segment becomes full during GC, it is inserted into both victim lists (4). REFlash then writes each valid block from the victim segment to the device address of the allocated block in the selected active segment (5). As in the write path, each active segment is assigned its corresponding RUH so that its data is placed in a separate reclaim unit.

After the GC writes complete, REFlash updates the metadata (6). The destination segment's entry in the segment map is set to the minimum *WT* among the copied blocks, i.e., the *WT* of the oldest block now stored in that segment. The *WT* entries in the hash map remain unchanged because GC only relocates existing data and does not represent a new host write. In the block map, the entries for the original copies are invalidated, and the segment map entry for the victim segment is invalidated as well.

**Flush path.** For flush, the segment reclaimer selects the oldest segment from the flush victim list, and removes the corresponding pointer from the GC victim list. Valid blocks in the victim segment are identified in the same way as in the GC path (1). REFlash then writes these valid blocks to the capacity-tier SSD at the host LBAs obtained from the block map (2). Once the flush writes complete, the corresponding entries are removed from the hash map because the data no longer resides in the buffer tier. As in the GC path, the block map entries for the flushed blocks and the segment map entry for the victim segment are invalidated (3).

**Table 2.** Workload characteristics used in evaluation.

| Trace | DWPD range | WSS | Scale up |
|---|---|---|---|
| FIO | – | 3.2 TB | 1x |
| YCSB-A | – | 3.2 TB | 8x |
| Alibaba1 | >2 | 2.4 TB | 10x |
| Alibaba2 | 1–2 | 2.6 TB | 8x |
| Alibaba3 | 0.1–1 | 4.8 TB | 2x |
| Varmail | – | 2.7 TB | 1x |

### 6.4 Implementation

We implement REFlash as a user-space block device driver based on SPDK 25.09. The driver implements the full data path, including the allocator, segment reclaimer, reclamation controller, and bounded-lifetime grouping mechanisms described in the preceding sections. The segment size is set to 6 GB, matching the reclaim-unit size of the FDP SSD used in our evaluation. REFlash uses seven active segments, each mapped to a dedicated Reclaim Unit Handle (RUH) provided by the FDP SSD: two for host writes and five for GC.

We also develop a trace-driven flash-based tiering simulator in C++ to perform the $U_B$ sweep shown in Figure 5 and to collect segment age distributions, lifetime statistics, and GC statistics in Figure 12. The simulator takes an I/O trace and device configurations including capacity, over-provisioning ratio, page size, and block size, as input, and models block-level GC for both the buffer and capacity tiers.

We compare REFlash against CSAL+GC, SepBIT [58], and MiDAS [34] to isolate the effect of the host-level GC policy. CSAL+GC selects the segment with the fewest valid blocks as the GC victim. SepBIT and MiDAS are state-of-the-art non-machine-learning data placement policies for log-structured storage. We exclude machine-learning-based approaches [18, 55] to isolate the effect of bounded-lifetime grouping itself, without confounding the results with the accuracy of a learned predictor. All schemes use the same

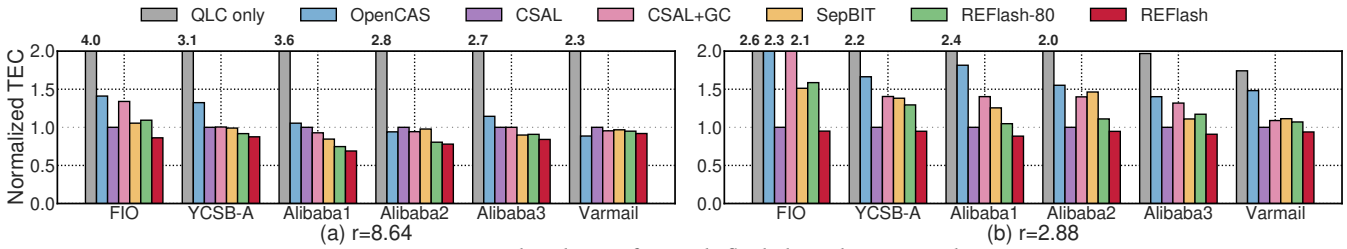



**Figure 8.** Normalized TEC for each flash-based tiering scheme.

log-structured tiering with segment-level FIFO flush, differing only in host-level GC. The baselines are implemented on the same platform, following the original papers and the authors' reference implementations [9, 11].

SepBIT and MiDAS use their default configurations of six and five active segments, respectively. Prior work [18] shows that increasing the number of groups without sufficient prediction accuracy can increase WA due to misprediction penalties; therefore, we retain each baseline's default number of active segments.

# 7 Evaluation

## 7.1 Experimental Setup

We conduct experiments on Ubuntu 22.04 with Linux kernel 6.12.0. The tiering hierarchy consists of a Samsung PM9D3 1.88 TB TLC SSD with FDP support as the buffer tier and a Samsung BM1743 15.36 TB QLC SSD as the capacity tier. The host machine is equipped with a 12th Gen Intel Core i7-12700 CPU and 64 GB of memory. We compare REFlash against CSAL and OpenCAS, the representative log-structured and in-place-update tiering schemes introduced in §3, respectively. All schemes are implemented on SPDK 25.09 and use a single dedicated I/O core for both SSDs across all configurations. We also evaluate CSAL+GC, SepBIT, and REFlash-80, each operating at a fixed target $U_B$ of 80%. REFlash-80 uses bounded-lifetime grouping and warm-aware victim selection, but disables the runtime reclamation decision model.

We evaluate REFlash using FIO, YCSB workload A, Alibaba Cloud block traces, and Varmail, configured as shown in Table 2. We replay only writes to measure cross-tier endurance cost. For FIO, we generate 4 KB random writes with a queue depth of 64, four threads, and a Zipf distribution with $\theta = 0.9$. For YCSB-A, we use a publicly available block trace [1, 62] collected from RocksDB on ext4. The Alibaba Cloud trace [4, 27, 58] consists of multiple volumes. To focus on write-intensive volumes that stress the buffer tier, we select volumes whose working set exceeds 30% of the volume capacity and whose daily write volume exceeds 10% of the drive capacity. We group these volumes into three configurations by the ratio of daily write volume to drive capacity (DWPD). Each configuration partitions the buffer-tier LBA space into contiguous regions, one per volume. This setup reflects environments where multiple workloads share a single physical device. For Varmail, we use a block trace generated by Filebench on ext4. We replicate each trace across disjoint LBA regions so that its working set exceeds the 1.88 TB buffer-tier capacity.

Both SSDs are prefilled to full capacity before each experiment. Each experiment begins with a warm-up phase that replays 6 TB of host writes to bring the buffer tier to a steady state. We then reset all metrics and report results over the subsequent 8 TB run phase. We select two endurance cost ratios, $r = 8.64$ and $r = 2.88$, computed from the TEC model in Table 1 using the specifications of two commercial TLC SSDs [44, 47] paired with the BM1743 QLC SSD. These two $r$ values affect both the TEC computation and the runtime reclamation decision. A higher $r$ makes the controller favor host-level GC more aggressively to reduce capacity-tier writes. Unless otherwise noted, all results are obtained from real hardware experiments. Simulator-based experiments are explicitly indicated where applicable.

## 7.2 Total Endurance Cost Comparison

Figure 8 compares TEC normalized to CSAL (= 1.00). REFlash achieves the lowest average TEC among all evaluated schemes, reducing TEC by up to 31% compared with CSAL. Its average reduction is 17.2% at $r = 8.64$ and 7.0% at $r = 2.88$.

QLC-only incurs 2.6× the TEC of CSAL on average, demonstrating the endurance-cost benefit of tiering. OpenCAS incurs high device-level WA on the buffer tier. Although CSAL+GC and SepBIT mitigate device-level WA through log-structured writes, they perform host-level GC at a fixed $U_B = 0.80$. They improve over CSAL for some workloads at $r = 8.64$. At $r = 2.88$, however, buffer-tier writes contribute more to TEC, and their GC costs outweigh the flush savings. Consequently, both schemes incur higher TEC than CSAL across all evaluated workloads.

At the same fixed $U_B$, REFlash-80 reduces GC overhead through bounded-lifetime grouping and warm-aware victim selection, but its TEC still ranges from 0.75–1.59× CSAL. Adding the runtime decision model narrows this range to 0.69–0.95× CSAL. This comparison demonstrates the benefit of runtime balancing beyond GC optimization alone. The decision model limits GC when its buffer-tier endurance cost outweighs the expected capacity-tier savings.

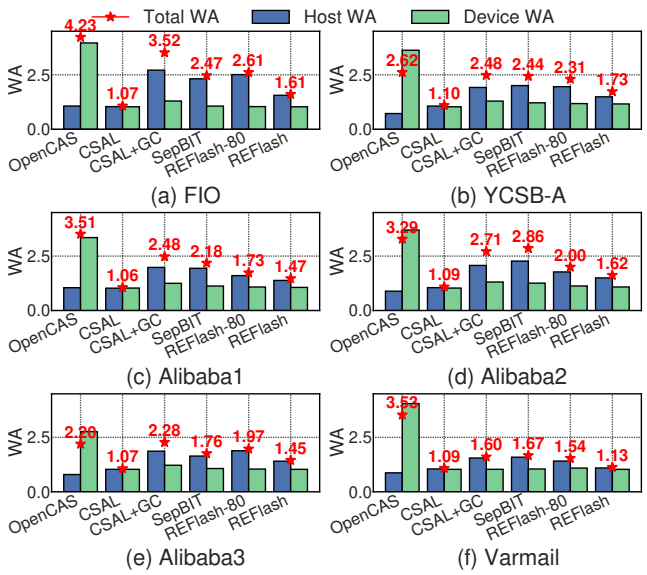



**Figure 9.** Write amplification on the buffer-tier SSD for each flash-based tiering scheme.

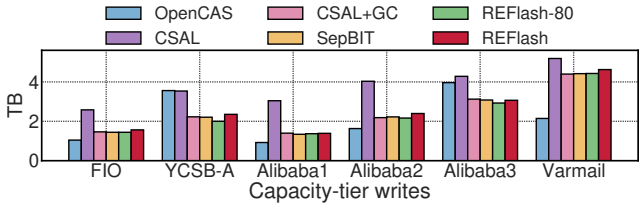



**Figure 10.** Writes to the capacity-tier SSD for each flash-based tiering scheme.

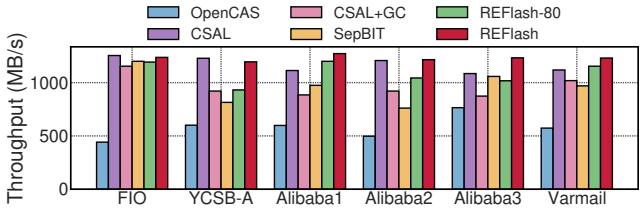



**Figure 11.** Write throughput of each flash-based tiering scheme.

### 7.3 Write Amplification on Buffer-Tier SSD

Figure 9 shows host- and device-level WA on the buffer-tier SSD. Total WA is their product. Host-level WA is the ratio of bytes issued to the buffer-tier SSD to workload write bytes. Device-level WA is the ratio of media bytes written to bytes issued to the SSD, with both counters obtained from the NVMe FDP Statistics log [2, 33].

At $r = 8.64$, REFlash achieves the second-lowest total buffer-tier WA after flush-only CSAL, with a maximum of 1.73 across all workloads. OpenCAS incurs high device-level WA from random writes, while its host-level WA remains near or below 1 because it performs no host-level GC and directs some host writes straight to the capacity tier. Log-structured schemes keep device-level WA near 1, but CSAL+GC and SepBIT incur high host-level WA from GC. REFlash-80 reduces host-level WA on average relative to CSAL+GC and SepBIT through improved grouping and victim selection. REFlash's runtime reclamation decision further reduces GC writes.

### 7.4 Capacity-Tier Writes

Figure 10 compares capacity-tier writes. REFlash writes less data to the capacity tier than CSAL, but up to 1.18× as much as REFlash-80. Despite these additional capacity-tier writes, REFlash achieves lower TEC than REFlash-80 by reducing buffer-tier GC writes (§7.2). This trade-off arises because REFlash limits GC when its buffer-tier endurance cost outweighs the expected flush savings.

CSAL's flush-only policy provides fewer opportunities for in-buffer invalidation, resulting in the highest capacity-tier write volume. SepBIT and REFlash-80 retain data through GC, reducing flush writes by 15–56% relative to CSAL. OpenCAS keeps capacity-tier writes low on most workloads, but bypassing write-once blocks in Alibaba3 and YCSB-A increases its capacity-tier writes relative to some schemes.

### 7.5 Write Throughput

Figure 11 reports write throughput measured by replaying each workload on real hardware. REFlash achieves throughput comparable to CSAL, the best-performing baseline on average, while adaptively limiting host-level GC overhead through its runtime decision model.

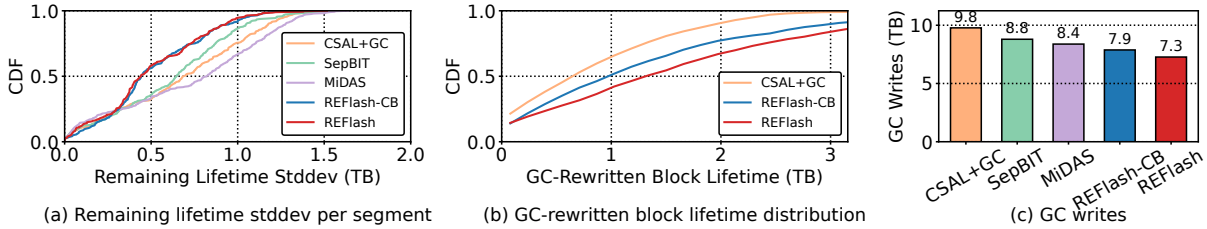
OpenCAS has the lowest throughput, primarily due to its high device-level WA. Consistent with Figure 9, workloads with higher device-level WA under OpenCAS also exhibit lower write throughput.

CSAL+GC, SepBIT, and REFlash-80 perform host-level GC at a fixed target of $U_B = 0.80$. The additional GC writes reduce throughput for CSAL+GC and SepBIT by 17.6% and 17.2% on average, respectively, relative to CSAL. REFlash-80 reduces GC overhead through bounded-lifetime grouping and warm-aware victim selection, improving average throughput over SepBIT. However, its throughput remains 6.3% below CSAL on average, showing the performance cost of fixed-target GC even with these optimizations.

### 7.6 Host-level GC Comparison on Buffer Tier

We compare host-level GC efficiency using Alibaba2 in our simulator, with 6 TB of warm-up followed by 8 TB of host writes. To isolate data placement and victim selection, all schemes use a fixed $U_B = 0.80$ and FIFO flushing, evicting the least recently written segment.

**Remaining lifetime distribution.** Figure 12(a) plots the CDF of per-segment remaining-lifetime standard deviations.




(a) Remaining lifetime stddev per segment (b) GC-rewritten block lifetime distribution (c) GC writes

**Figure 12.** Impact of data placement strategies on the buffer tier: (a) remaining lifetime distribution, (b) lifetime distribution of GC-rewritten blocks, and (c) GC write volume.

We measure each valid block's remaining lifetime from a given point in time to its invalidation by overwrite or flush, in cumulative host-write bytes. Grouping blocks with similar lifetimes aligns their invalidation times, reducing GC overhead [17, 43].

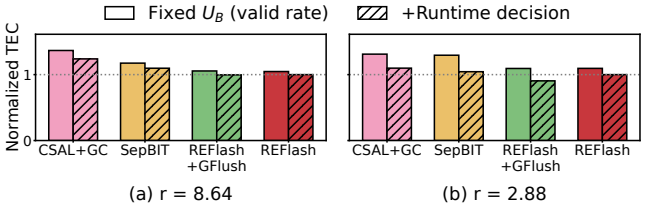
Most segments under REFlash and REFlash-CB have a standard deviation below 1.0 TB. REFlash-CB uses bounded-lifetime grouping with conventional cost-benefit selection, while REFlash adds warm-aware selection (§5.2). Both account for FIFO-flush-driven invalidation through bounded-lifetime grouping. SepBIT and MiDAS instead group GC-migrated blocks more coarsely using overwrite-based lifetimes alone, showing little improvement over CSAL+GC.

**Post-migration lifetime.** Figure 12(b) compares post-migration block lifetimes. CSAL+GC's greedy selection favors hot segments [43], whose migrated blocks are frequently overwritten, yielding the shortest lifetimes. Conventional cost-benefit selection instead favors older segments to avoid imminent overwrites [43], yielding longer lifetimes than CSAL+GC. However, these older blocks are closer to FIFO flushing, limiting the benefit of their migration in the buffer tier. REFlash's warm-aware selection avoids victims whose relocated blocks are likely to be invalidated soon, either by overwrites or by FIFO flushing. Consequently, its relocated blocks remain valid longer, yielding the longest post-migration lifetimes.

**GC write volume.** Figure 12(c) compares the resulting GC write volume. CSAL+GC incurs the most GC writes because it lacks dedicated data placement and frequently migrates blocks that are invalidated soon afterward. SepBIT and MiDAS reduce this overhead through their data-placement and victim-selection policies. REFlash combines bounded-lifetime grouping with warm-aware selection and achieves the lowest GC write volume, 26% below CSAL+GC.

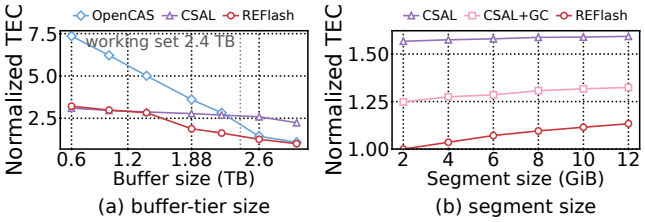
### 7.7 Runtime Decision Model under Alternative Reclamation Policies

We examine whether the runtime decision model remains effective under different GC and flush policies. On the GC path, we replace REFlash's default GC selection policy with greedy selection (CSAL+GC) and with SepBIT. On the flush path, we replace the FIFO flush order with greedy flush (REFlash+GFlush), which flushes the segment with the fewest valid blocks among the oldest 20% of segments. For each




(a) r = 8.64 (b) r = 2.88

**Figure 13.** Fixed-target $U_B$ (0.80) versus our runtime decision model (hatched) with different reclamation policies on Alibaba1. TEC is normalized to REFlash with the runtime decision model in each panel.




(a) buffer-tier size (b) segment size

**Figure 14.** TEC sensitivity to buffer-tier size (a) and segment size (b) on Alibaba1, normalized to the lowest TEC per panel.

configuration, including REFlash's default policies, we compare a fixed target of $U_B = 0.80$ against the runtime decision model. Figure 13 shows that the runtime model reduces TEC in every configuration, by 4.5–9.4% at $r = 8.64$ and by 8.7–19.3% at $r = 2.88$. These results indicate that the benefit of the runtime decision model persists across alternative GC and flush policies.

### 7.8 Configuration Sensitivity

We evaluate sensitivity to buffer-tier size, segment size, and the number of GC groups using Alibaba1, whose working-set size is 2.4 TB. We use trace-driven simulation for the buffer-tier and segment-size sweeps, allowing us to explore buffer capacities beyond the fixed 1.88 TB device in our testbed. We evaluate the number of GC groups on real hardware.

**Buffer-tier size.** Figure 14(a) shows that REFlash achieves lower TEC than CSAL at larger buffer-tier sizes. When the buffer tier is smaller than 1.4 TB, however, REFlash converges toward CSAL because host-level GC provides little reduction

**Table 3.** DRAM footprint of REFlash metadata for the 1.88 TB buffer tier. Hash map (LBA) includes host LBAs and device LBAs.

| Metadata type | Memory (GB) | Share (%) |
|---|---|---|
| Hash map (LBA) | 4.05 | 41.07 |
| Hash map (WT) | 4.05 | 41.07 |
| Block map | 1.76 | 17.84 |
| Segment map | < 0.01 | 0.01 |
| Other metadata | < 0.01 | 0.01 |
| **Total** | **9.86** | **100.00** |

in backend flushes. REFlash detects this condition at runtime and limits GC-induced writes to 13% of host writes, applying GC only when the expected reduction in flush traffic outweighs its relocation cost.

**Segment size.** Figure 14(b) varies the segment size from 2 to 12 GiB. TEC increases with segment size for CSAL, CSAL+GC, and REFlash because larger segments coarsen the migration granularity, moving more blocks to the capacity tier before they would otherwise be invalidated. Nevertheless, REFlash maintains lower TEC than both CSAL and CSAL+GC across the entire range, demonstrating that its benefit is robust to segment granularity.

**Number of GC groups.** Using Alibaba1 on real hardware, we observe little variation in TEC with $N_{gc}$ = 3–5. With $N_{gc}$ = 2, coarser lifetime grouping increases host-level GC writes by 9.4% relative to $N_{gc}$ = 5. Our prototype uses five GC groups, with the remaining two of the SSD's seven RUHs assigned to hot and cold host writes.

### 7.9 CPU and DRAM Overheads

**CPU overhead.** In our implementation, REFlash's buffer-management logic runs on a single dedicated core, which remains 40.6% idle. Its runtime decision consumes only 0.35% of a core because it is invoked once per segment rather than per I/O. Most of the additional CPU cost relative to CSAL comes from mapping lookups (6.0% versus 0.9% of a core). REFlash uses hash-based lookups, whereas CSAL maps both tiers at 4 KB granularity and uses array-based lookups.

**DRAM footprint.** Table 3 breaks down REFlash's metadata footprint by metadata type. REFlash uses 9.86 GB of metadata for a 1.88 TB buffer tier, equivalent to 0.52% of the buffer-tier size. The footprint grows linearly with buffer-tier size. The 8-byte $WT$ values account for 41.07% of the footprint, and their storage cost could be reduced through timestamp quantization [18, 34].

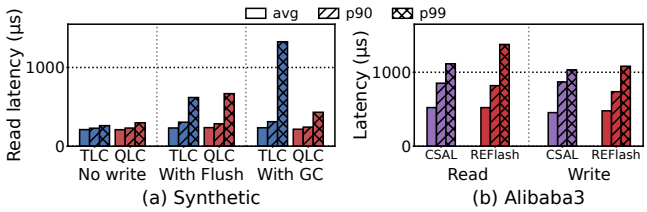



**Figure 15.** (a) Per-tier read latency without concurrent writes and during flush or GC. (b) Read and write latency on Alibaba3.

### 7.10 Read Performance

Separately from the preceding experiments, we evaluate read performance using a synthetic workload and mixed-I/O replay of Alibaba3. Figure 15(a) compares read latency on the TLC and QLC tiers. A controlled write pattern keeps one LBA range in the TLC buffer tier and flushes another to the QLC capacity tier. We use FIO to issue 4 KB reads at a queue depth of 32 to each range separately under three conditions: no concurrent writes, concurrent sequential writes that trigger flush, and concurrent 4 KB overwrites to the buffer-tier range that trigger host-level GC. Without concurrent writes, average and p90 latencies differ by less than 1% between tiers, while QLC p99 is 14% higher than TLC p99. Flush increases read tail latency on both tiers, whereas GC causes particularly pronounced interference on TLC.

We use Alibaba3 for mixed-I/O evaluation, as reads account for 30% of its requests, compared with less than 2% in Alibaba1 and Alibaba2. We replay both reads and writes. Figure 15(b) shows that REFlash matches CSAL's average read latency but increases read p99 by 23.5%. Average and p99 write latencies are approximately 5% higher, while p90 is lower for both reads and writes. The higher read p99 is consistent with the buffer-tier GC interference observed in the synthetic experiment.

## 8 Related Work

**Flash-Based Tiering.** OpenCAS [36] is an in-place-update caching scheme that performs block-level (4 KB) replacement and overwrites the same device location on cache hits, resulting in high device-level WA. It evicts blocks using an LRU policy; when configured as a write-back cache, it flushes the least recently written blocks. CSAL [66] is a log-structured caching scheme that reclaims segments solely through flush. It uses FIFO eviction, flushing the least recently written segments first. From the perspective of write ordering, this behavior is similar to OpenCAS in that the least recently written data is evicted first. REFlash builds on the log-structured approach but additionally performs cost-aware host-level GC to reduce flush writes.

Oh et al. [35] dynamically partition an SSD cache into read cache, write cache, and overprovisioning regions to optimize

flash-cache performance in an SSD-HDD tiering. REFlash targets an SSD-SSD tiering and instead dynamically adjusts the balance between flush and host-level GC as segment reclamation methods. This enables REFlash to minimize the total endurance cost across both flash tiers at runtime.

FairyWREN [30] co-designs the caching policy and host-level GC on a ZNS SSD. It separates hot and cold objects into distinct zones and performs GC at a fixed utilization using FIFO victim selection. REFlash similarly separates data groups physically through host-directed placement, using FDP in our implementation. However, REFlash differs in three key ways. First, it dynamically chooses the segment reclamation method based on endurance cost rather than maintaining a fixed utilization. Second, it uses separate victim-selection criteria for host-level GC and flush. Third, it employs warm-aware cost-benefit victim selection that accounts for the bounded lifetime of buffer-tier data, rather than relying on FIFO victim selection.

**Data Placement in Log-Structured Storage.** Data placement techniques for log-structured storage estimate the remaining lifetime of host-written and GC-migrated blocks, and assign them to appropriate active segments to reduce GC-induced write amplification. Here, we focus on how prior work places GC-migrated blocks, where REFlash fundamentally differs.

SepBIT [58] heuristically assigns GC-migrated blocks to progressively colder groups based on age, assuming that blocks surviving more GC rounds are likely to live longer. MiDAS [34] and PHFTL [55] follow a similar approach, using migration count as a proxy for age and placing blocks with the same GC count into the same active segment. DOGI [18] takes an ML-based approach, using historical prediction records to estimate the remaining invalidation time of GC-migrated blocks and relocating them to matching groups.

These techniques target single-device storage, where the remaining lifetime of GC-migrated blocks spans a wide and fluctuating range, making fine-grained prediction inherently difficult. In contrast, REFlash exploits a cross-tier eviction property of the buffer tier: because blocks are flushed from the buffer tier in FIFO order, each segment's maximum lifetime is naturally bounded. REFlash divides this bounded lifetime across GC groups to guide placement, enabling tighter and more predictable data separation.

Prior schemes also differ in victim selection. SepBIT uses cost-benefit selection, while MiDAS combines FIFO selection for earlier groups with cost-benefit for the last group. PHFTL employs an adjusted greedy policy that discounts short-lived segments to compensate for mispredictions, and DOGI prioritizes expired segments whose residence time exceeds the upper bound of their assigned group. REFlash instead introduces a warm-aware cost-benefit policy that uses the minimum of the current age and the remaining bounded lifetime as the weighting factor, adapting victim selection to the eviction characteristics of the buffer tier.

# 9 Conclusion

REFlash coordinates buffer-tier operations in flash-based tiering systems using TEC, a metric that captures both GC-induced write amplification and capacity-tier endurance cost. REFlash combines run-time reclamation decisions, bounded-lifetime grouping, and warm-aware victim selection to minimize TEC. By balancing host-level GC and flush across tiers, rather than optimizing either operation in isolation, REFlash consistently reduces total endurance cost compared with existing schemes across diverse workloads, enabling more cost-effective use of QLC SSDs.

# Acknowledgments

We acknowledge the experimental equipment support provided by Samsung Electronics' MAX Lab. This work was also supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. RS-2021-II211363 and No. RS-2024-00460762).

# References

[1] 2020. YCSB RocksDB SSD. SNIA IOTTA Repository. https://iotta.snia.org/traces/block-io/28568

[2] 2025. *NVM Express® Base Specification.* Specification (Ratified) Revision 2.3. NVM Express, Inc. https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-Revision-2.3-2025.08.01-Ratified.pdf

[3] Saba Ahmadian, Onur Mutlu, and Hossein Asadi. 2018. ECI-Cache: A high-endurance and cost-efficient I/O caching scheme for virtualized platforms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1, Article 9 (2018), 34 pages. doi:10.1145/3179412

[4] Alibaba. [n. d.]. Alibaba Block Traces. GitHub. https://github.com/alibaba/block-traces

[5] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. 2020. The CacheLib caching engine: Design and experiences at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association, 753–768.

[6] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. 2017. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proc. IEEE* 105, 9 (2017), 1666–1704. doi:10.1109/jproc.2017.2713127

[7] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2011. Hystor: Making the best use of solid state drives in high performance storage systems. In *Proceedings of the International Conference on Supercomputing (ICS '11)*. Association for Computing Machinery, 22–32. doi:10.1145/1995896.1995902

[8] Dell Technologies. 2024. Dell PowerStore: Best Practices Guide. https://infohub.delltechnologies.com/static/media/client/7phukh/DAM_2e2b9bcb-a15e-4831-8a60-2e4f2b4c3161.pdf

[9] dgist-datalab. [n. d.]. MiDAS. https://github.com/dgist-datalab/MiDAS

[10] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*. USENIX Association, 65–78.

[11] fallfish. [n. d.]. sepbit. https://github.com/fallfish/sepbit

[12] Sumit Gupta. 2025. QLC storage: Balancing power, cost and performance. https://files.futurememorystorage.com/proceedings/2025/20250806_QLCP-201-1_Gupta.pdf

[13] Intel Corporation. 2014. Intel Solid-State Drive Data Center Family for PCIe Provides Breakthrough Performance to Modernize Data Center Storage. News Fact Sheet, June 3, 2014. https://download.intel.com/newsroom/kits/ssd/pdfs/Intel_SSD-Data_Center_PCIe_Family-Factsheet.pdf

[14] Intel Corporation. 2019. Intel SSD DC P4610 Series. Product Brief, document no. 337735-003US, April 2019. https://www.solidigm.com/content/dam/solidigm/en/site/products/data-center/d7/p4610/documents/dc-p4610-series-product-brief.pdf

[15] Intel Corporation. 2020. Intel Optane SSD DC P5800X Series — Product Specifications. https://www.intel.com/content/www/us/en/products/sku/201860/intel-optane-ssd-dc-p5800x-series-800gb-2-5in-pcie-x4-3d-xpoint/specifications.html

[16] JEDEC. 2016. JESD218B.01: Solid-State Drive (SSD) Requirements and Endurance Test Method. JEDEC Standard.

[17] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. 2014. The multi-streamed solid-state drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '14)*. USENIX Association.

[18] Jeeyun Kim, Seonggyun Oh, Jungwoo Kim, Jisung Park, Jaeho Kim, Sungjin Lee, and Sam H. Noh. 2026. DOGI: Data placement with oracle-guided insights for log-structured systems. In *24th USENIX Conference on File and Storage Technologies (FAST '26)*. USENIX Association, 543–559.

[19] 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 (MASCOTS '11)*. IEEE, 227–236. doi:10.1109/mascots.2011.64

[20] KIOXIA America, Inc. 2025. KIOXIA Introduces LC9 Series NVMe SSDs for the Biggest AI, Object Storage and Data Lake Requirements. Product Overview, KIOXIA LC9 Series, November 2025, version 1.1. https://americas.kioxia.com/content/dam/kioxia/en-us/business/ssd/enterprise-ssd/asset/KIOXIA_LC9_Product_Overview.pdf

[21] 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 (ICAC '15)*. IEEE, 51–60. doi:10.1109/icac.2015.44

[22] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. F2FS: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST '15)*. USENIX Association, 273–286.

[23] LenovoPress. 2015. S3700 SATA MLC Enterprise SSDs. Product Guide, TIPS1014. https://lenovopress.lenovo.com/tips1014

[24] Chu Li, Dan Feng, Yu Hua, and Fang Wang. 2019. A high-performance and endurable SSD cache for parity-based RAID. *Frontiers of Computer Science* 13, 1 (2019), 16–34. doi:10.1007/s11704-017-6523-9

[25] Cheng Li, Philip Shilane, Fred Douglis, Hyong Shim, Stephen Smaldone, and Grant Wallace. 2014. Nitro: A capacity-optimized SSD cache for primary storage. In *2014 USENIX Annual Technical Conference (USENIX ATC '14)*. USENIX Association, 501–512.

[26] Cheng Li, Philip Shilane, Fred Douglis, and Grant Wallace. 2017. Pannier: Design and analysis of a container-based flash cache for compound objects. *ACM Transactions on Storage* 13, 3, Article 24 (2017), 34 pages. doi:10.1145/3094785

[27] Jinhong Li, Qiuping Wang, Patrick P. C. Lee, and Chao Shi. 2020. An in-depth analysis of cloud block storage workloads in large-scale production. In *2020 IEEE International Symposium on Workload Characterization (IISWC '20)*. IEEE, 37–47. doi:10.1109/iiswc50251.2020.00013

[28] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. 2022. Operational characteristics of SSDs in enterprise storage systems: A large-scale field study. In *20th USENIX Conference on File and Storage Technologies (FAST '22)*. USENIX Association, 165–180.

[29] Sara McAllister, Benjamin Berg, Julian Tutuncu-Macias, Juncheng Yang, Sathya Gunasekar, Jimmy Lu, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. 2021. Kangaroo: Caching billions of tiny objects on flash. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. Association for Computing Machinery, 243–262. doi:10.1145/3477132.3483568

[30] Sara McAllister, Yucong Wang, Benjamin Berg, Daniel S. Berger, Nathan Beckmann, George Amvrosiadis, and Gregory R. Ganger. 2025. FairyWREN: A sustainable cache for emerging write-read-erase flash interfaces. *ACM Transactions on Storage* 21, 4, Article 26 (2025), 34 pages. doi:10.1145/3718390

[31] Micron Technology, Inc. 2020. Micron 5210 ION Data Center SATA QLC SSD. Product Brief, Rev. B, June 2020, document no. CCM004-676576390-11154. https://www.simms.co.uk/PDFS/Products/SSD/5210_ion_ssd_product_brief.pdf

[32] NetApp. 2023. QLC all-flash arrays for the data center. https://www.netapp.com/blog/qlc-all-flash-arrays-data-center/

[33] NVM Express. 2022. TP4146 Flexible Data Placement. NVM Express Technical Proposal. https://nvmexpress.org/wp-content/uploads/NVM-Express-2.0-Ratified-TPs_20230111.zip

[34] Seonggyun Oh, Jeeyun Kim, Soyoung Han, Jaeho Kim, Sungjin Lee, and Sam H. Noh. 2024. MiDAS: Minimizing write amplification in log-structured systems through adaptive group number and size configuration. In *22nd USENIX Conference on File and Storage Technologies (FAST '24)*. USENIX Association, 259–275.

[35] Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2012. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *10th USENIX Conference on File and Storage Technologies (FAST '12)*. USENIX Association, 313–326.

[36] Open CAS. 2026. *Open Cache Acceleration Software | Open CAS*. Project homepage. Retrieved March 3, 2026 from https://open-cas.com/

[37] Oracle. 2026. Oracle Exadata Database Machine X11M. https://www.oracle.com/a/ocom/docs/engineered-systems/exadata/exadata-x11m-ds.pdf

[38] Chris Panos. 2024. QLC in the datacenter: Making the optimal choices. Presentation at FMS '24. https://files.futurememorystorage.com/proceedings/2024/20240808_QLCP-304-1_Panos.pdf

[39] Nikolaos Papandreou, Haralampos Pozidis, Nikolas Ioannou, Thomas Parnell, Roman Pletka, Milos Stanisavljevic, Radu Stoica, Sasa Tomic, Patrick Breen, Gary Tressler, Aaron Fry, Tim Fisher, and Andrew Walls. 2020. Open block characterization and read voltage calibration of 3D QLC NAND flash. In *2020 IEEE International Reliability Physics Symposium (IRPS '20)*. IEEE, 1–6.

[40] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. 2016. Improving performance and lifetime of NAND storage systems using relaxed program sequence. In *Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*. Association for Computing Machinery, 1–6. doi:10.1145/2897937.2898032

[41] QSAN Technology, Inc. 2022. QSLife Technology: From SSD Endurance to SSD Life Cycle. https://www.qsan.com/data/dl_files/qs_wp_QSLife_%28en%29.pdf

[42] S. W. Roberts. 1959. Control chart tests based on geometric moving averages. *Technometrics* 1, 3 (1959), 239–250. doi:10.1080/00401706.1959.10489860

[43] Mendel Rosenblum and John K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems* 10, 1 (1992), 26–52. doi:10.1145/146941.146943

[44] Samsung Electronics. 2018. Samsung PM1725b NVMe SSD — Product Brief. https://image-us.samsung.com/SamsungUS/PIM/Samsung_

1725b_Product.pdf

[45] Samsung Electronics. 2019. Samsung SSD 860 QVO white paper: QLC SSD – High density, high performance and affordable. https://download.semiconductor.samsung.com/resources/white-paper/Samsung_SSD_860_QVO_White_paper.pdf

[46] Samsung Electronics. 2024. Samsung BM1743 QLC V-NAND SSD. https://semiconductor.samsung.com/news-events/tech-blog/next-generation-qlc-v-nand-increases-data-center-profitability/

[47] Samsung Electronics. [n. d.]. PM9D3a — Product Specifications. https://semiconductor.samsung.com/ssd/datacenter-ssd/

[48] Samsung Electronics. [n. d.]. Samsung PM9A3 NVMe PCIe SSD. Product Brief. https://download.semiconductor.samsung.com/resources/brochure/Samsung%20PM9A3%20NVMe%20PCIe%20SSD.pdf

[49] Samsung Semiconductor. 2024. NVMe FDP: A promising new SSD data placement approach. Samsung Tech Blog. https://semiconductor.samsung.com/news-events/tech-blog/nvme-fdp-a-promising-new-ssd-data-placement-approach/

[50] Mohit Saxena, Michael M. Swift, and Yiying Zhang. 2012. FlashTier: A lightweight, consistent and durable storage cache. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12)*. Association for Computing Machinery, 267–280. doi:10.1145/2168836.2168863

[51] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. 2018. DIDACache: An integration of device and application for flash-based key-value caching. *ACM Transactions on Storage* 14, 3, Article 26 (2018), 32 pages. doi:10.1145/3203410

[52] Solidigm. [n. d.]. Efficiently accelerate access to more warm data with Solidigm QLC 3D NAND technology. https://www.solidigm.com/products/technology/efficiently-access-more-warm-data-solidigm-tech-brief.html

[53] Solidigm. [n. d.]. Solidigm D3-S4510 Series Product Brief. https://www.solidigm.com/content/dam/solidigm/en/site/products/data-center/d3/s4510/documents/d3-s4510-series-product-brief.pdf

[54] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD lifetimes with disk-based write caches. In *8th USENIX Conference on File and Storage Technologies (FAST '10)*. USENIX Association, 101–114.

[55] Penghao Sun, Litong You, Shengan Zheng, Wanru Zhang, Ruoyan Ma, Jie Yang, Guanzhong Wang, Feng Zhu, Shu Li, and Linpeng Huang. 2023. Learning-based data separation for write amplification reduction in solid state drives. In *2023 60th ACM/IEEE Design Automation Conference (DAC '23)*. IEEE, 1–6. doi:10.1109/dac56929.2023.10247795

[56] The Linux Kernel Documentation. [n. d.]. A block layer cache (bcache). https://docs.kernel.org/admin-guide/bcache.html

[57] VAST Data. 2025. VAST AI Operating System: White Paper. https://www.vastdata.com/whitepaper

[58] Qiuping Wang, Jinhong Li, Patrick P. C. Lee, Tao Ouyang, Chao Shi, and Lilong Huang. 2022. Separating data via block invalidation time inference for write amplification reduction in log-structured storage. In *20th USENIX Conference on File and Storage Technologies (FAST '22)*. USENIX Association, 429–444.

[59] Shucheng Wang, Ziyi Lu, Qiang Cao, Hong Jiang, Jie Yao, Yuanyuan Dong, and Puyuan Yang. 2020. BCW: Buffer-controlled writes to HDDs for SSD-HDD hybrid storage server. In *18th USENIX Conference on File and Storage Technologies (FAST '20)*. USENIX Association, 253–266.

[60] Western Digital. 2021. Ultrastar DC ZN540 — Data Sheet. Document no. 02-05-WW-04-00323-AA01. https://hypertecsp.com/wp-content/uploads/data-sheet-ultrastar-dc-zn540-1.pdf

[61] Haonan Wu, Erci Xu, Ligang Wang, Yuandong Hong, Changsheng Niu, Bo Shi, Lingjun Zhu, Jinnian He, Dong Wu, Weidong Zhang, Qiuping Wang, Changhong Wang, Xinqi Chen, Guangtao Xue, Yi-Chao Chen, and Dian Ding. 2025. Hey Hey, My My, skewness is here to stay: Challenges and opportunities in cloud block store traffic. In *Proceedings of the 20th European Conference on Computer Systems (EuroSys '25)*. Association for Computing Machinery, 736–752. doi:10.1145/3689031.3696068

[62] Gala Yadgar, Moshe Gabel, Shehbaz Jaffer, and Bianca Schroeder. 2021. SSD-based workload characteristics and their performance implications. *ACM Transactions on Storage* 17, 1, Article 8 (2021), 26 pages. doi:10.1145/3423137

[63] Chongzhuo Yang, Zhang Cao, Chang Guo, Ming Zhao, and Zhichao Cao. 2024. Can ZNS SSDs be better storage devices for persistent cache?. In *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '24)*. Association for Computing Machinery, 55–62.

[64] Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman. 2014. Don't stack your log on my log. In *2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW '14)*. USENIX Association.

[65] Jinsoo Yoo, Joontaek Oh, Seongjin Lee, Youjip Won, Jin-Yong Ha, Jongsung Lee, and Junseok Shim. 2018. OrcFS: Orchestrated file system for flash storage. *ACM Transactions on Storage* 14, 2, Article 17 (2018), 26 pages. doi:10.1145/3162614

[66] Yanbo Zhou, Erci Xu, Li Zhang, Kapil Karkra, Mariusz Barczak, Wayne Gao, Wojciech Malikowski, Mateusz Kozlowski, Łukasz Łasek, Ruiming Lu, Feng Yang, Lilong Huang, Xiaolu Zhang, Keqiang Niu, Jiaji Zhu, and Jiesheng Wu. 2024. CSAL: The next-gen local disks for the cloud. In *Proceedings of the 19th European Conference on Computer Systems (EuroSys '24)*. Association for Computing Machinery, 608–623. doi:10.1145/3627703.3629566