

Article

Wear Leveling in SSDs Considered Harmful: A Case for Capacity Variance[†]

Ziyang Jiao^{1,*}  and Biyuan Yang²¹ School of Computer Science and Technology, Huaibei Normal University, Huaibei 235000, China² Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244, USA; byang02@syr.edu

* Correspondence: jiaoziyang@chnu.edu.cn; Tel.: +86-131-4584-9450

[†] This article is a revised and expanded version of a paper entitled “Wear Leveling in SSDs Considered Harmful”, which was presented at The 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage, 2022, USA), Virtual, 27–28 June 2022. It received the Best Paper Award at the conference from Samsung Semiconductors.

Abstract

The trend of decreasing endurance of flash memory makes the overall lifetime of SSDs more sensitive to the effects of wear leveling. Under these circumstances, we observe that existing wear-leveling techniques exhibit anomalous behavior under workloads without clear access skew or under dynamic access patterns and produce high write amplification, as high as $5.4\times$, negating its intended benefits. We argue that wear leveling is an artifact for maintaining the fixed-capacity abstraction of a storage device, and it becomes unnecessary if the exported capacity of the SSD is to gracefully reduce. We show that this idea of capacity variance extends the lifetime of the SSD, allowing up to $2.94\times$ more writes under real workloads.

Keywords: SSD; wear leveling; endurance; device lifetime; write amplification; capacity variance



Academic Editors: Jie Zhou,
Shuoyang Qiu and Jiarui Gong

Received: 14 September 2025

Revised: 13 October 2025

Accepted: 21 October 2025

Published: 25 October 2025

Citation: Jiao, Z.; Yang, B. Wear Leveling in SSDs Considered Harmful: A Case for Capacity Variance. *Electronics* **2025**, *14*, 4169. <https://doi.org/10.3390/electronics14214169>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

NAND flash-based solid-state drives (SSDs) are an integral part of today’s computing systems, used in mobile and embedded devices to large-scale data centers. However, flash memories wear out as they are programmed and erased and progressively exhibit more errors [1,2]. Furthermore, once used beyond an endurance limit, memory cells may not behave correctly and are considered bad [3]. Thus, minimizing the total amount of writes that causes programs and erases in the SSD is paramount in managing the overall lifetime [4–6]. However, this is becoming increasingly challenging as the endurance limit of flash memory has been steadily decreasing, as shown in Figure 1. This figure plots the endurance limit for a wide variety of SSDs over the past several years, estimated by dividing the amount of writes the SSD can sustain (in the form of TBW, terabytes written) by the logical capacity of the device. This downward trend is a result of trading reliability for higher density [2,7,8], and a modern flash memory cell can only withstand up to a few hundred programs and erases (a portion of this research was conducted at Syracuse University and published as a conference paper [9]).

Wear leveling (WL) techniques seek to equalize the amount of wear within the SSD so that most, if not all, cells reach the endurance limit prior to the end of the SSD’s lifetime [10–13]. While there are different approaches to implementing WL (from

static [10–12] to dynamic [13]), the underlying goal is to use younger blocks with fewer erases more than the older blocks. Static wear-leveling techniques [10–12], in particular, proactively relocate data within an SSD, thereby incurring additional write amplification for the sake of equalizing the number of erases. In other words, WL techniques incur additional wear-out to maximize the total number of writes that the SSD can sustain.

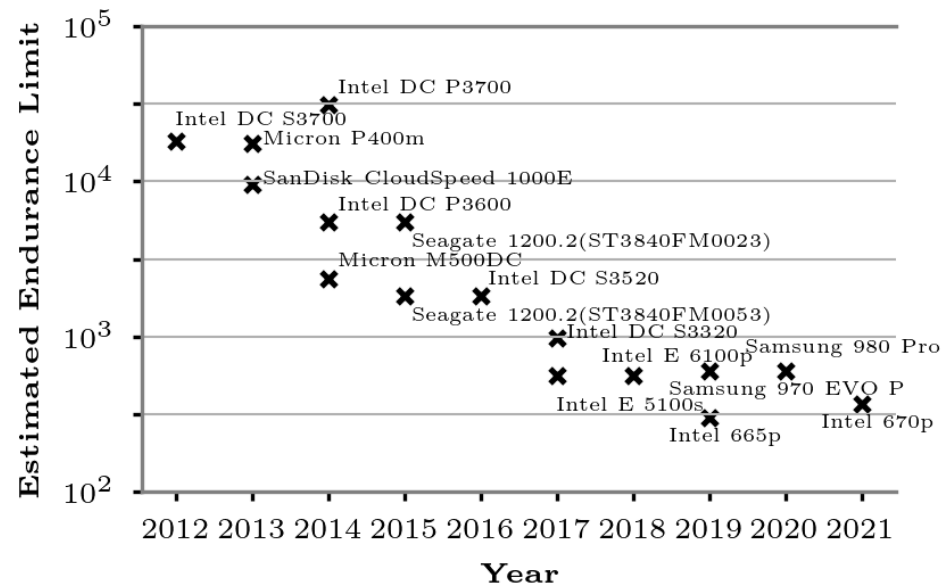


Figure 1. The estimated endurance limit of various SSDs released in recent years. The endurance limit is calculated by dividing each SSD’s manufacturer-specified TBW (terabytes written: the total number of writes the SSD manufacturer guarantees) by its logical capacity. The data are collected from the official specifications of well-known SSD manufacturers and include both client-grade and enterprise-class models to represent different NAND flash generations. The y -axis is shown in log scale.

We argue that the conventional approach of wear leveling is ill-suited for today’s SSDs for the following three reasons. First, as Figure 1 shows, the endurance of flash memory cells has been decreasing to a level where only a few hundred program–erase cycles would wear them out. Because each WL decision also contributes to wear-out due to data relocation, suboptimal decisions for WL cause more harm than good. Secondly, we find that WL algorithms produce a counter-productive result where the erase counts diverge, increasing the spread rather than reducing it. We observe this when the workload’s access footprint is large and its access pattern is skewed, as the WL attempts to proactively move data that it perceives to be cold into old blocks. Lastly, WL algorithms are a double-edged sword in which achieving a tight distribution of erase count comes the cost of high write amplification. Although adaptive WL approaches [12,14] exhibit a low write amplification while they are inactive during the early stages of the SSD’s life, they overcompensate toward the end of life and significantly accelerate the wear-out.

Instead of designing a new wear-leveling algorithm that patches these performance issues, we explore and quantify the benefits of capacity variance in SSDs. Unlike a traditional SSD that exports a fixed capacity, a capacity-variant SSD reduces its capacity gracefully as flash memories become bad. The fixed capacity interface, the current paradigm for all storage devices, is built around traditional hard disk drives that exhibit a *fail-stop* behavior (in this context, a fail-stop model means that the storage device as a whole is either working or not, and its failure state can immediately be made aware by other components of the system) when their mechanical parts crash [15]. However, SSDs fail *partially* as flash memory blocks are the basic unit of failure, and mapping out failed bad blocks is an inherent

responsibility of the SSD firmware as manufacture time bad blocks exist [3]. Wear leveling exists to maintain the fixed-capacity interface and emulate a fail-stop behavior when the underlying technology is actually fail-partial. Thus, it is necessary to revisit the efficacy of wear leveling and the validity of a fixed-capacity interface in a modern setting with a low endurance limit.

In this work, we evaluate three representative wear-leveling techniques [10–12] in the traditional fixed-capacity interface and show that their write amplification factors can reach up to 5.4. We also uncover that WL algorithms can produce opposite results where the variance of the erase count increases as WL becomes more active. We then evaluate the presence and absence of WL in both fixed-capacity and capacity-variant SSDs. Our experimental results show that capacity variance allows up to 84% more writes to the SSD with wear leveling. Interestingly, capacity variance *without* WL allows up to $2.94\times$ more writes, demonstrating that WL techniques can accelerate the overall wear state significantly.

The contributions of this work are as follows:

- We evaluate representative wear-leveling algorithms and show that they exhibit high write amplification and undesirable results in modern SSD configurations (Section 3).
- We qualitatively describe the benefits of capacity variance for SSDs and discuss the necessary modifications in the system and storage management (Section 4).
- We quantitatively demonstrate that a capacity-variant interface can significantly extend the SSD's lifetime by evaluating the presence and absence of wear-leveling algorithms in both fixed and variable capacity SSDs using a set of real I/O workloads (Section 5).

2. Motivation

In this section, we provide the challenges in managing lifetime in SSDs, describe some of the typical wear-leveling algorithms, and then qualitatively describe their shortcomings and side effects.

2.1. Managing the SSD Lifetime

The overall architecture of a typical SSD is shown in Figure 2. The flash translation layer (FTL) hides the peculiarities of the underlying flash memory and provides an illusion of a block storage device [16]. Some of these peculiarities are as follows: (1) Flash memory prohibits in-place updates. Because all updates must be out-of-place, data are written to a new location and the *mapping table* associates the logical address with the physical location. (2) The program operation ($1 \rightarrow 0$) and the erase operation ($0 \rightarrow 1$) have different granularity: a page and a block, respectively. Thus, to erase a block and reclaim space for new updates, all valid pages within the block must first be copied to another location through *garbage collection*. (3) Flash memory cells wear out and eventually become unusable. The *wear leveler* aims to equalize the wear for all blocks so that blocks do not prematurely wear out before the SSD's lifetime.

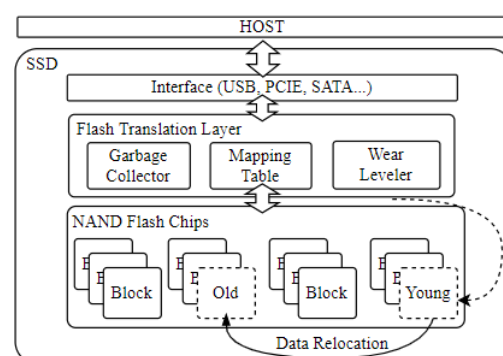


Figure 2. An architectural diagram of a typical SSD.

An SSD's lifetime is typically defined with a conditional warranty restriction under *DWPD* (*drive writes per day*), *TBW* (*terabytes written*), or *GB/day* (*gigabytes written per day*) [17]. This is because writing to an SSD induces programs (and eventually erases) that weaken the non-volatile memory, leading to a wear-out. Flash memory manufacturers specify how many times a flash memory block can be erased before turning bad [3]. Symptoms of these bad blocks include flash memory operation errors and high bit error rates, and an SSD typically maps out these blocks so that they will no longer be used [1,3]. When too many blocks become bad, the SSD cannot maintain the logical capacity, and the entire storage device becomes unusable. As such, managing the number of writes to the SSD is critical in guaranteeing the lifetime.

However, it is not only I/O writes that cause programs and erases. Any SSD-internal housekeeping such as garbage collection and wear leveling perform erases that wear out the device [8]. This additional amount of writes in relation to the amount external I/O write is referred to as *write amplification* (WA), and minimizing WA benefits not only the lifetime but also the performance [8]. However, achieving this is not as straightforward. For garbage collection, greedily selecting the minimum amount of data to copy leads to unexpected inefficiencies [18], and wear leveling directly opposes WA as it causes additional data relocation to equalize the erase count.

2.2. Wear-Leveling Algorithms

Figure 3 describes the relationship between different wear-leveling algorithms [10–12,14,19–22]. One naïve implementation called *hot-cold swap* simply swaps the data stored in the oldest block and the youngest block once the difference in erase count exceeds some predefined threshold [21]. For example, as shown in Figure 2, the data in the young block is relocated to the old block. This approach is based on the intuition that the youngest block likely holds infrequently updated *cold* data while the oldest block likely has frequently updated *hot* data. After the hot-cold swap, the youngest block will likely be erased because of the hot data, while the oldest block with cold data will not be erased. However, the oldest block may be selected for WL shortly after (even though it holds cold data) if the algorithm incorrectly judges it to hold hot data, causing unnecessary data swaps [11].

Old Block Protection (OBP) [10] addresses this problem by maintaining a recent history of WL activities. Once a block is erased, its erase count is compared with that of the youngest block in use: if the difference exceeds a predefined threshold, and if this just-erased block has not been involved in WL recently, the content of the youngest block will be copied into the just-erased block. OBP prevents the old block now containing a likely-cold data (that was copied from the youngest block) from being involved in wear leveling again.

Dual-Pool (DP) [11] implements a more sophisticated algorithm to prevent the same block from partaking in WL repeatedly. It maintains two pools of blocks, hot pool and cold pool, each intended for storing hot data and cold data, respectively. It also performs a hot-cold swap, but between the youngest block (lowest erase count) in the cold pool and the oldest block (highest erase count) in the hot pool. Furthermore, the pool associations for the two blocks are also changed so that the oldest block in the hot pool becomes part of the cold pool (since it now holds cold data). This prevents the same oldest block from getting selected by the wear leveler. DP also implements an adaptive pool resize where a hot pool block that contains data that became cold is moved to the cold pool and vice versa.

Unlike OBP and DP, **Progressive Wear Leveling (PWL)** [12] adaptively changes the trigger condition for wear leveling based on the overall wear state of the SSD. In the early stage of life for the SSD (when the average erase count is low), PWL lies dormant, avoiding unnecessary data copies. However, as the overall erase count increases, the activeness of

PWL proportionally increases as well. This algorithm requires a predefined initial threshold (THR_{init}) that dictates the behavior: lower THR_{init} causes WL to become active early in the SSD's life while a higher threshold delays the point in which WL becomes active.

OBP [10], DP [11], and PWL [12] represent different types of wear-leveling algorithms for SSDs. They are often described as static WL, distinct from dynamic WL [13,23] that is integrated with garbage collection and block allocation. We do not consider dynamic WL in our evaluation as it conflates the efficiency of space reclamation and effectiveness of wear leveling.

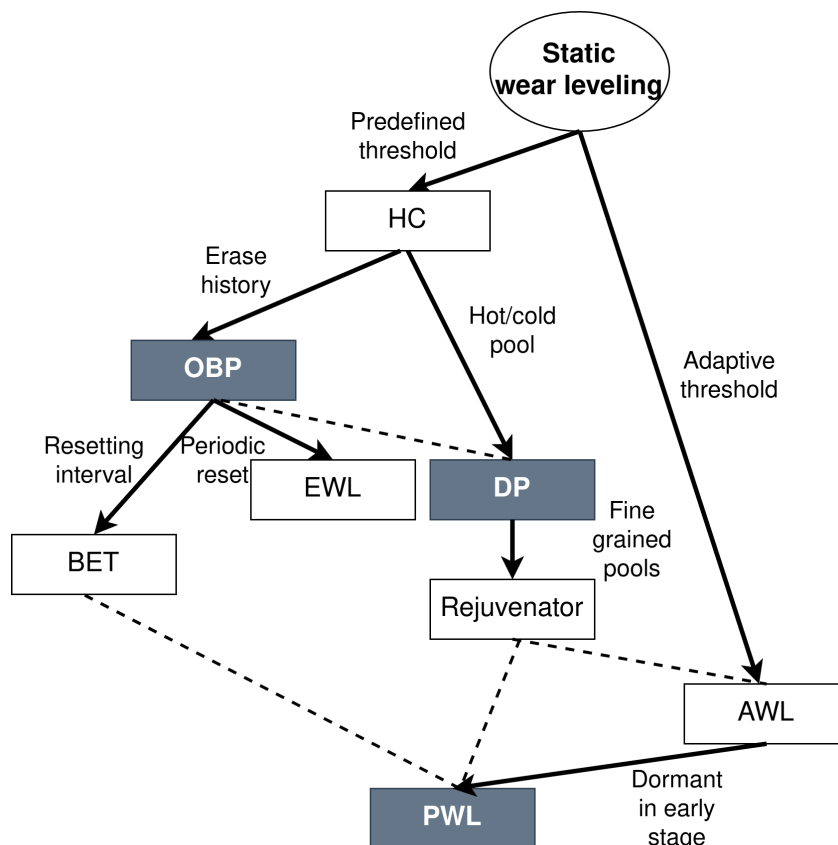


Figure 3. The relationship between various static wear-leveling algorithms [10–12,14,19–22]. Solid arrows indicate an enhancement from the previous algorithm (top-down implies the chronology), and the dotted lines indicate that they were evaluated in their respective paper.

2.3. Wear-Leveling Behaviors

Wear-leveling algorithms, in practice, often exhibit unintended odd behaviors that stem from misjudging the lifetime of data in a block. Figure 4 depicts the different possible erase count distributions, given without WL in Figure 4a. A moderately active WL (Figure 4b) would reduce the spread of erase count compared to that without, but an overly aggressive WL (Figure 4c) increases the overall erase count through significant write amplification. Hence, WL can be a double-edged sword where it is either ineffective reducing the variance in erase count, or overly active and causes the younger blocks to become as old as the others. A highly undesirable scenario is where the erase count diverges as shown in Figure 4d, and we describe the conditions in which this occurs in Section 3. Ideally, WL should reduce the spread as shown in Figure 4e.

In the next section, we demonstrate that the existing WL algorithms show behaviors in Figure 4, and argue that because of their shortcomings they may not be suitable for today's flash memory with a low endurance limit where each WL decision consumes an erase count.

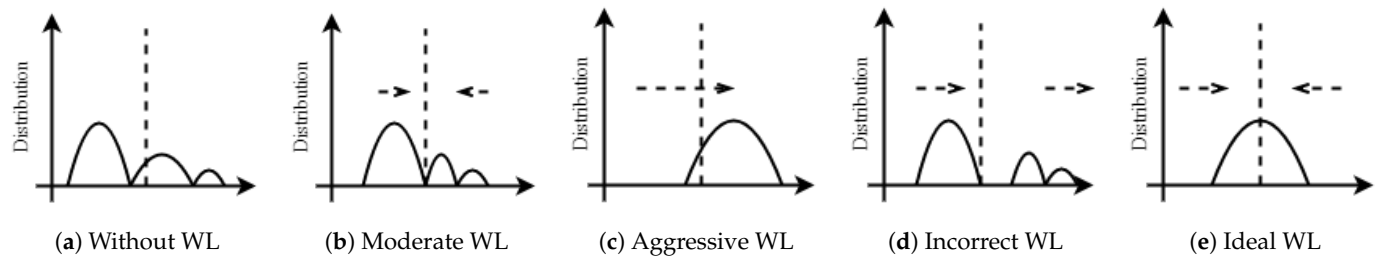


Figure 4. Pictorial description of wear-leveling behavior. Given a distribution of erase count without wear leveling (WL) in (a), (b) shows the spread given typical, moderately active WL. However, an overly aggressive WL accelerates the erase count as shown in (c), and an incorrectly behaving WL even causes an increased spread, shown in (d). Ideally, WL should achieve a distribution similar to (e).

3. Performance of Wear Leveling

We evaluate the performance of three representative wear-leveling (WL) algorithms: Old Block Protection (OBP) [10], Dual-Pool (DP) [11], and Progressive Wear Leveling (PWL) [12]. The $OBP(TH, PT)$ configuration copies the data of the youngest block to the just-erased block if the difference between the erase counts exceeds TH , unless the block has been involved in WL in the recent PT erases. $DP(TH)$ swaps the content of the oldest block in the hot pool and the youngest block in the cold pool if the erase count different exceeds TH . $PWL(TH)$ performs wear leveling on blocks that exceed a certain erase count threshold that scales proportionally with the average erase count, offset by the $TH\%$ of the endurance limit. We first describe our experimental setup, present the evaluation results, and summarize our findings.

3.1. Experimental Setup

We extend FTLSim [24] to include a WL model for our experiments. The original FTLSim validates the analytical model for garbage collection, and thus focuses on accurately modeling SSD-internal statistics such as write amplification rather than SSD-external performance such as latency and throughput. This simplicity makes FTLSim fast, allowing us to simulate the entire lifetime of an SSD (a few hundreds of TiB written). The wear-leveling (WL) models in our simulator were implemented faithfully following the algorithms described in their respective original papers (OBP, DP, and PWL). The parameter configurations, such as block selection thresholds and trigger conditions, are identical to those reported in the original studies to ensure behavioral consistency [10–12]. Table 1 summarizes the SSD configuration and policies for our experiments. We adopt an 11% over-provisioning (OP) ratio, typical of commercial SSDs that reserve 6.7%–28% of capacity for internal management [25]. While OP affects the absolute write amplification and higher OP can reduce garbage collection overhead, it does not change the relative trends of WL behaviors.

Table 1. SSD configuration and policies. Only the parameters relevant to understanding the wear-leveling behavior are shown.

Parameter	Value	Parameter	Value
Page size	4 KiB	Physical capacity	284 GiB
Pages per block	256	Logical capacity	256 GiB
Block size	1 MiB	Over-provisioning	11%
Block allocation	FIFO	Garbage collection	Greedy

To understand the behavior of WL techniques, we synthetically generate workload to control the I/O pattern better. All I/Os are small random writes, but the distribution is controlled by two parameters r and h ($0 < r < 1$ and $0 < h < 1$): r fraction of writes go to the h fraction of the footprint (hot addresses) [26]. The r fraction of accesses are uniformly and randomly distributed within the hot addresses, and conversely, the $1 - r$ fraction of accesses are uniformly and randomly distributed within the $1 - h$ cold addresses. We use r/h to indicate that the r fraction of writes occurs on the h fraction of the logical space. Unless otherwise noted, we generate the I/O workload for the entire logical address space. Prior to each experiment, we pre-condition the SSD with one sequential full-drive write, followed by three random full-drive writes (256 GiB sequential + 768 GiB random write) to put the drive into a steady state [27].

3.2. Evaluation of Wear Leveling Under Synthetic Workloads

We investigate the performance of wear leveling in the following three aspects: (1) write amplification, (2) effectiveness in equalizing the erase count, and (3) behavior under different access footprint.

3.2.1. Write Amplification

We measure the WL-induced write amplification (WA) by using a synthetic workload of $r/h = 0.9/0.1$ for up to 100 full-drive writes (25 TiB). We examine PWL and two different configurations for OBP and DP wear leveling each. The WL parameter values we experiment with are similar to those used in the prior work [11,12].

Figure 5 shows the write amplification caused by wear leveling over time, and we make the following four observations. First, the wear leveling-induced write amplification can be as high as $5.4\times$. This means that for each 256 GiB user data written, wear leveling alone will create an additional 1.35 TiB of data writes internally. Secondly, the WL threshold parameter TH dictates the WA for both OBP and DP. Changing the TH from 8 to 4 for the OBP algorithm will amplify the amount of data written by $1.2\times$, while changing from 10 to 5 for the DP, by $1.4\times$. Thirdly, as expected, PWL is inactive during the early stages. However, beyond about 80 full-drive writes, it exhibits an aggressive behavior, causing significant WA, as high as $5.4\times$. The late-life surge in WA observed for PWL results from its threshold-based activation design. PWL remains dormant while the average erase count stays below a predefined threshold, minimizing data movement during early SSD life. Once this threshold is exceeded, PWL activates and performs extensive data redistribution, causing the abrupt rise in write amplification near the end of life. Lastly, WA steadily increases over time as the SSD ages, indicating that SSD aging will accelerate as more data are written.

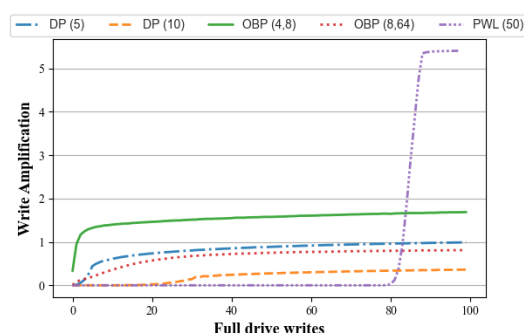


Figure 5. The write amplification caused by wear leveling under a $r/h = 0.9/0.1$ synthetic workload. These values exclude garbage collection copies to isolate the WL-induced overhead. *PWL(50)* that aggressively performs wear leveling at the late stage causes its write amplification to be as high as $5.4\times$.

3.2.2. Wear Leveling Effectiveness

We measure the distribution of erase count of different configurations under a synthetic workload as shown in Figure 6. We exercise a workload with $r/h = 0.9/0.1$ for 100 full-drive writes (25 TiB).

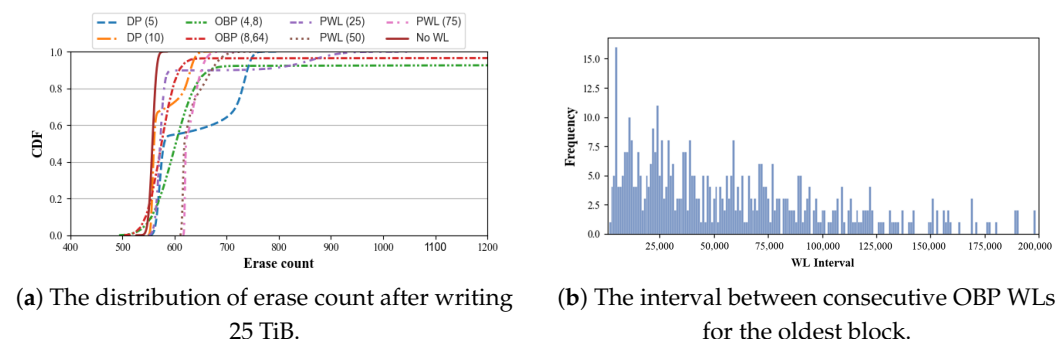


Figure 6. The performance anomaly of wear leveling under $r/h = 0.9/0.1$ workload. In (a), we observe that wear leveling makes the erase count more *uneven*, as evident by the flat plateaus in the CDF curve. This is because a small set of blocks is heavily involved in wear leveling, as shown in (b): most of the erases happen soon after the protection period ends.

Figure 6a shows that under $r/h = 0.9/0.1$, all three WL show uneven distribution of erase counts, in fact, worse than not running WL (*NoWL*). For *OBP*(4,8) and *OBP*(8,64), we observe that the CDF plateaus before reaching 1.0, indicating that there is a small group (of 7% for *OBP*(4,8) and 3% for *OBP*(8,64)) blocks that exceed the 1200 erase count. Similarly, *DP* and *PWL* also show a concave dip in the CDF curve, indicating a bimodal distribution of erase counts. *NoWL*, on the other hand, shows a nearly vertical line, meaning that the erase counts are more tightly distributed. We consider this to be a *performance anomaly* of wear leveling because it behaves the opposite of what is expected.

Through analysis, we find that the block with the highest erase count in the *OBP* algorithm is overly involved in wear leveling despite the fact that the *PT* parameter is designed to protect the block from being erased again. Specifically, one old block can become extraordinarily worn because each time it participates in hot-cold swapping, it is erased and ages further. In subsequent activation periods, this block is likely chosen again, as it remains the oldest block, creating a feedback loop of repeated erasures. We run another experiment using the same $r/h = 0.9/0.1$ workload, but for *OBP*(5,1000). Figure 6b shows the interval between two consecutive WL for the most worn-out block. Even with a long *PT* = 1000, the block is involved in WL again soon after its protection period expires.

We also evaluate *OBP*, *DP*, and *PWL* with $r/h = 0.8/0.2$ and $r/h = 0.5/0.5$, as shown in Figures 7 and 8. Even though *OBP*(4,8) and *OBP*(8,64) show nearly vertical CDF curves, the average erase count is much higher, indicating that it amplifies the amount of written data significantly. *PWL*(25), *PWL*(50), and *PWL*(75), however, show a similar result with *NoWL*. On the other hand, both *DP*(5) and *DP*(10) show a performance anomaly with a skewed workload (Figure 7a), as shown in the wider CDF curves compared to *NoWL*. We examine the bimodal distribution of *DP*(5) and find that the erase count distribution is bimodal with blocks associated with the cold pool older than those in the hot pool. The *DP* algorithm's underlying assumption is that blocks containing hot data are older than blocks with cold data, and it compares the erase count of the oldest block in the hot pool and the youngest block in the cold pool. If the youngest block in the cold pool happens to be older than the oldest block in the hot pool, however, it will still trigger the swap between the two blocks, causing this inversion.

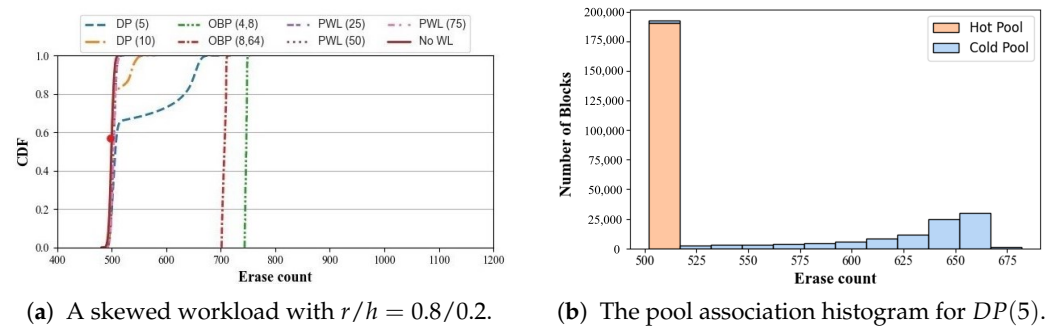


Figure 7. The distribution of erase count under $r/h = 0.8/0.2$ (skewed). The red dot indicates the average erase count for NoWL. For the skewed workload in (a), wear leveling comes at a high cost of write amplification, as evident by the fact that the lines are on the right side of the red dot. DP causes the erase count to diverge, and examining the pool association reveals that the blocks associated with cold data are much older than the blocks associated with hot data, as shown in (b). This inversion occurs because the algorithm assumes that blocks with hot data are old and those with cold data are young and swaps the oldest block in the hot pool with the youngest block in the cold pool. If the youngest block in the cold pool becomes older than the oldest block in the hot pool, an unnecessary WL takes place, moving the hot data into the older block.

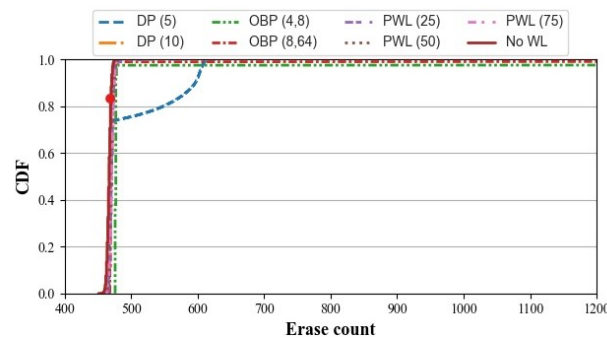


Figure 8. The distribution of erase count under $r/h = 0.5/0.5$ (uniform). We observe that the benefit from wear leveling is negligible compared to not running at all.

On the other hand, with a uniformly random workload (Figure 8), there is a negligible difference between running WL and not running WL at all. This is because with a uniform workload, all blocks are used equally, and there is little room for wear leveling. We do observe, however, that DP(5) still exhibits a performance anomaly though at a smaller degree than under $r/h = 0.9/0.1$ (cf. Figure 6a).

These experiments show that WL algorithms are a double-edged sword. As shown in Figure 6a, it can make the distribution of wear worse than not running WL at all. On the other hand, it can achieve good wear leveling but at a high cost of accelerated overall wear state.

3.2.3. Small Access Footprint

Previously, in Sections 3.2.1 and 3.2.2, the experiments were conducted by writing to the entire logical address space. Here we explore the performance of wear leveling when the accesses are restricted to a small address space using two synthetic workloads, $r/h = 0.9/0.1$ and $r/h = 0.5/0.5$, as shown in Figure 9.

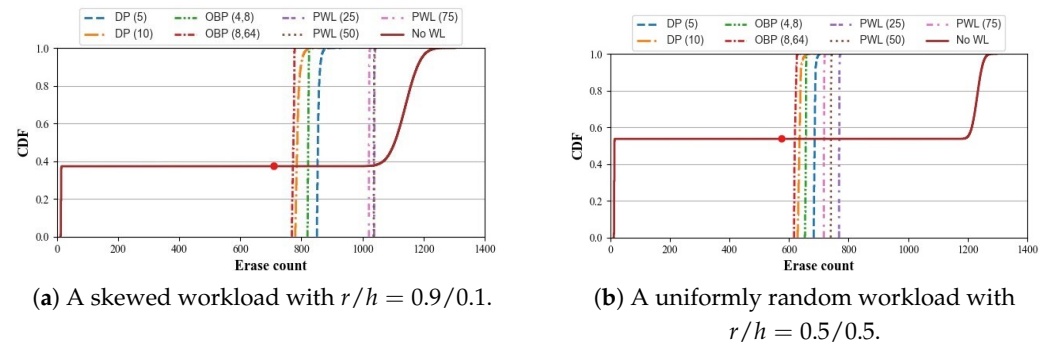


Figure 9. The distribution of erase count when only 5% of the logical address space is used. The red dot indicates the average erase count for NoWL. With a small footprint, wear leveling achieves good evenness in erase count, while not running it causes a bimodal distribution. However, with a skewed workload in (a), we observe that the erase counts for WL are further to the right of the red dot compared to those in (b), indicating that WL amplified the number of data writes.

Overall, we observe that most WL techniques are effective in equalizing the erase count, as shown by the near-vertical CDF curve in both Figure 9a,b. NoWL, on the other hand, shows a bimodal distribution between used blocks and unused blocks in both workloads. We also observe that when the workload is skewed (Figure 9a), the WL techniques achieve this evenness by amplifying the amount of data writes, as shown by the rightward shift in the CDF curves. For a uniform workload, on the other hand (Figure 9b), the overall write amplification from wear leveling is much lower as data in used blocks are equally likely to be invalidated.

Unlike the results from Figures 6 and 7, where the entire logical address space is written, WL is effective only when a small fraction of the address space is used, restricting its overall usefulness.

3.3. Summary of Findings

Based on the experimental results using synthetic workloads, we qualitatively summarize the effectiveness of wear leveling in Table 2.

- **Uniform access with a small access footprint:** Overall, WL evens out the erase count with low write amplification (Figure 9b).
- **Skewed access with a small access footprint:** WL achieves good performance but at the cost of high write amplification (Figure 9a).
- **Uniform access with a large access footprint:** The performance of wear leveling has a negligible difference with not running it at all. There may be cases where performance anomaly occurs (Figure 8).
- **Skewed access with a large access footprint:** Wear leveling not only amplifies writes but also exhibits an anomaly by significantly accelerating the erase count for a group of blocks (Figure 6a).

Table 2. Qualitative effectiveness of wear leveling.

	Uniform Access	Skewed Access
Small footprint	Effective	Write amplified
Large footprint	Negligible	Anomaly

To summarize, WL can be effective when the workload exhibits clear hot/cold data separation, where a small subset of data is frequently updated while most data remains static. In such cases, WL helps prevent uneven wear and extends device lifetime by

correctly identifying and redistributing hot data across blocks. However, in workloads lacking a distinct access skew, or under dynamic capacity conditions, WL may interfere with data placement strategies and introduce unnecessary data movement. Instead of proposing a new wear-leveling algorithm that solves both the write amplification overhead and performance anomaly, we question the circumstances that require wear leveling and examine its necessity in the next section.

4. A Capacity-Variant SSD

Our experiments on the effectiveness of wear leveling show that it is difficult to achieve good WL performance. There is only a limited number of scenarios where WL achieves its intended goal; in other cases, it either results in high write amplification or further unevenness.

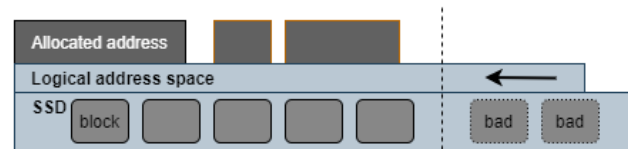
We argue that WL is an artifact only designed to maintain an illusion of a fixed-capacity device wherein its underlying storage components (i.e., flash memory blocks) either all work or fail. This, however, is far from the truth where blocks wear out individually (often at different rates) and are mapped out by the SSD once they become unusable. In other words, it is natural for the SSD's physical capacity to reduce over time, and it is only the fixed-capacity abstraction that mandates the SSD's lifetime to be defined as when the physical capacity becomes less than the logical capacity.

Instead, a capacity-variant SSD would gracefully reduce its exported capacity. We define capacity variance as the property of an SSD whose exported logical capacity dynamically changes over time as a function of the underlying flash block health. Unlike conventional SSDs that maintain a fixed logical capacity through wear leveling and internal remapping, a capacity-variant SSD allows its logical space to shrink gracefully when worn-out or unreliable blocks increase. This design exposes the natural wear behavior of flash devices to the system, enabling simplified management and reduced write amplification. This has three benefits over the traditional fixed-capacity interface. First, wear leveling becomes unnecessary as it does not need to ensure that all blocks wear out evenly. This would free the SSD from needing the computational resources for the WL algorithm and incur no write amplification overhead. Secondly, the lifetime of an SSD would extend significantly, and it would be defined by the amount of data stored in the SSD, not by the initial logical capacity. Lastly, it would be easier to determine when to replace an SSD. An SSD whose capacity is reduced, close to the amount of data stored and in use, would be a clear indication of an ailing SSD.

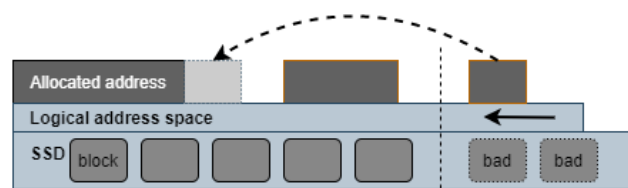
However, a capacity-variant interface would need support from the file system. Thankfully, the current system design can make this transition seamless for the following three reasons. First, the TRIM command, widely supported by interface standards such as NVMe (TRIM is called `Deallocate` in NVMe) [28], allows the file system to explicitly declare that the data (at the specified addresses) are no longer in use. This allows the SSD to discard the data safely and would help determine if the exported capacity can be gracefully reduced. Second, modern file systems can safely compact their content so that the data in use are contiguous in the logical address space. Log-structure file systems such as F2FS [29] support this more readily, but file system defragmentation can also achieve the same effect in in-place update file systems such as ext4 [30]. Lastly, the file abstraction to the applications thankfully hides the remaining space left on storage. A file is simply a sequence of bytes, and managing storage capacity is the responsibility of the system administrator. We expected the system support for a capacity-variant SSD to be no more complex than the *zoned namespace* [31] command set that shifts the responsibility of garbage collection to the host side.

This file system support potentially incurs overhead for the file system to relocate data from one logical space to another, as illustrated in Figure 10. In the case for Figure 10a,

the high logical address space is unused by the file system, and the capacity-variant SSD gracefully reduces this space with no data relocation overhead for the file system. However, in the case of Figure 10b, the SSD and the file system must handshake prior to reducing the capacity. Naïvely, the file system would not only relocate the data at the high address space but also update any metadata for the block allocation and inode. More advanced commands such as SHARE [32] can be used to reduce the relocation overhead.



(a) Capacity reduction without relocation overhead.



(b) Capacity reduction with relocation overhead.

Figure 10. Illustration of the data relocation overhead for reducing capacity. In (a), the capacity can be reduced without the file system relocating data at the high address. In (b), however, data in the allocated space at the high address must be moved before reducing capacity.

File System Design for Capacity Variance

The higher-level storage interfaces, such as the POSIX file system interface, allow multiple applications to access storage using common file semantics. However, to support capacity variance, the file system needs to be modified. This section focuses on the design of the file system to support capacity-variant SSDs. We consider F2FS [29], a log-structured file system (LFS), as our base file system due to its out-of-place update nature.

Data consistency. Maintaining data consistency is crucial when reducing the recommended space below the currently declared space of a logical storage partition. Shrinking the logical capacity of a file system can be a complex procedure that may result in inconsistency and data loss if not performed carefully [33]. To ensure users do not need to unmount and remount the device, the logical capacity should be reduced in an online manner. Additionally, the time it takes to reduce capacity should be minimal with low overhead.

Figure 11 depicts three approaches to performing online address space reduction: (1) capacity variance through decoupling the address space from LBA; (2) capacity variance through data re-mapping; and (3) capacity variance through data relocation. We describe each approach and our rationale for choosing the data relocation (Figure 11c).

- **Address space decoupling** (Figure 11a). The file system internally decouples the space exported to users from the LBA and avoids using the space near the end of the logical partition as much as possible to minimize potential data relocation overhead. However, this increases the file system cleaning overhead and fragments the file system address space. Due to the negative effect of address fragmentation [34–37], we avoid this approach despite the lowest upfront cost.
- **Data remapping** (Figure 11b). Data is relocated logically, either at the file system level or the device level, to maintain data consistency while avoiding address space fragmentation. File system level remapping does not need to update the inodes for data relocation, but it creates yet another level of indirection. Device-level remapping, on the other hand, takes advantage of the already existing SSD-internal mapping table.

However, it requires a new SSD command for associating data with a new LBA without relocation and additional non-volatile data structures to ensure correctness [32,36,38]. We do not explore this approach due to its inherent concerns for crash consistency.

- **Data relocation** (Figure 11c). The file system relocates valid data within the to-be-shrunk area to a lower LBA region before reducing the capacity from the higher end of the logical partition. This approach maintains the continuity of the entire address space. Additionally, the cumulative overhead of data relocation is bounded by the original logical capacity of the device throughout its lifetime.

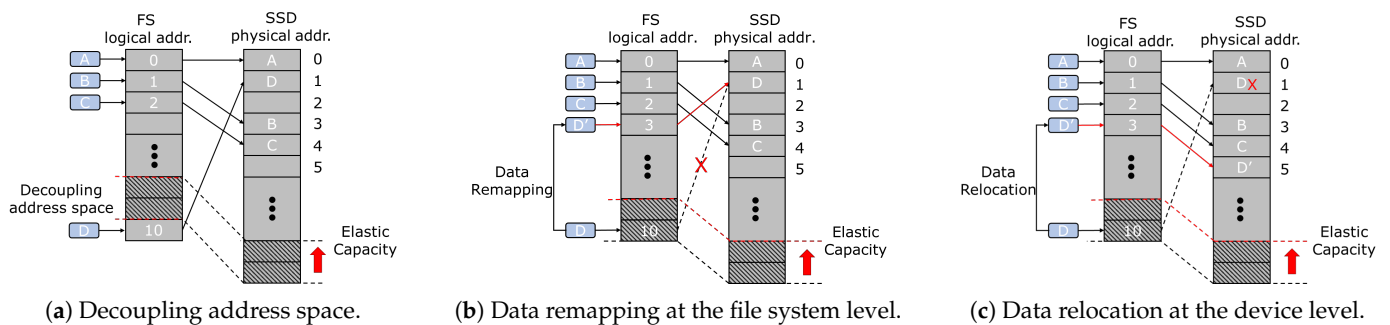


Figure 11. Approaches to maintain data consistency for capacity variance. In (a), the FS internally maps out a range of free LBA from the user, causing address space fragmentation. In (b), data block D can be logically remapped to lower LBA. This approach requires a special SSD command and introduces additional crash consistency concerns. Lastly, in (c), data block D can be physically relocated to lower LBA. This approach takes advantage of the out-of-place update nature of an LFS.

To ensure data integrity during the shrinking of a file system, Figure 12 illustrates the data relocation process, which consists of three phases: (1) state validation, (2) relocation, and (3) metadata update.

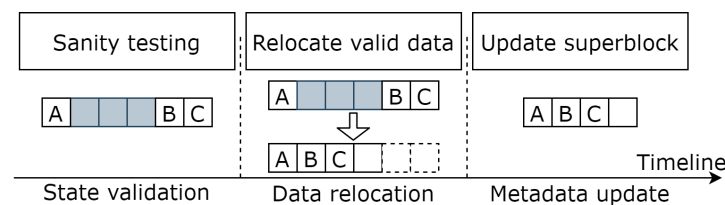


Figure 12. Illustration of the data relocation during the shrinking of a file system. Data in the allocated space at the high address must be moved before reducing capacity.

- **Step 1: State validation.** The file system performs sanity testing and checkpoints the current file system layout. If the file system checkpoint functionality is disabled or the file system is not ready to shrink (i.e., frozen or read-only), the reduction will not continue. This stage also initializes and checks new file system parameters, which ensures that the remaining free space is sufficient to accommodate data within the to-be-shrunk logical blocks.
- **Step 2: Relocation.** The write frontiers of each logging area are moved out of the target range, and any valid data in the to-be-shrunk blocks is relocated while restricting the available space to conform to the new file system layout. This is similar to segment cleaning or defragmentation. The block bitmap, reverse mapping, and file index are updated during this stage.
- **Step 3: Metadata update.** Once the to-be-shrunk area is free, the file system states are validated and the file system superblock is updated. In case of a system crash or error, the file system is rolled back to its latest consistent state.

We measure the time it takes to reduce capacity by 1 GiB. As illustrated in Figure 13, the elapsed time for shrinking 1 GiB logical space takes 0.075 s on a relatively young file system and 4.5 s on an aged file system that exhibits fail-slow symptoms, demonstrating the low overhead of keeping data consistent during capacity reduction.

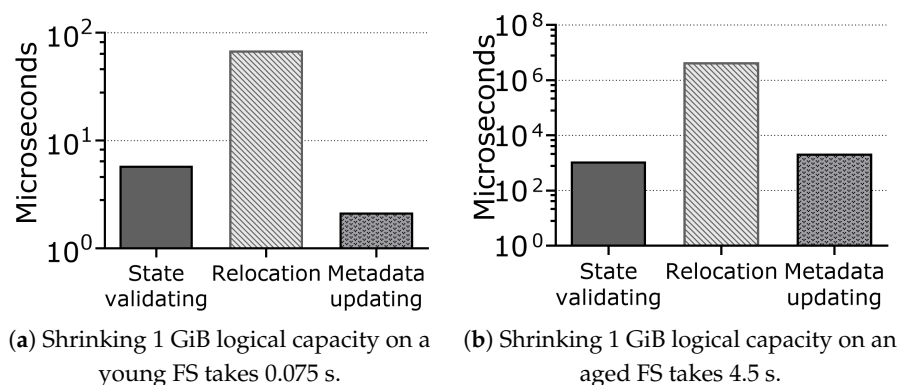


Figure 13. Shrinking 1 GiB logical capacity for a relatively young file system (**left**) and an aged file system (**right**). The y -axis is log-scale, so most of the time is spent copying data.

File system cleaning. For log-structured file systems, the append-only nature requires cleaning processes to reclaim scattered and invalidated logical blocks for further logging. Reducing the size of a logical partition may cause log-structured file systems to reclaim free space more frequently when the utilization keeps at the same level, increasing the write amplification of the file system. To mitigate this issue, dynamic logging policies are necessary to adapt capacity variance for LFS [18]. Normal logging always writes data to clean space in a strictly sequential manner and thus yields desirable performance, which should be used when the utilization is relatively low. In the threaded logging, data are written to the invalidated portion of the address space. Consequently, this policy could introduce random writes but requires no cleaning operations. However, as the utilization increases, threaded logging should be considered to avoid high file system cleaning overhead.

5. Evaluation

We evaluate both the presence and absence of wear leveling (OBP [10], DP [11], and PWL [12]) on both a fixed-capacity SSD and a capacity-variant SSD using real-world I/O traces. We design the experiments with the following questions in mind:

- Does capacity variance extend the lifetime of the SSD? How does wear leveling (WL) interact with capacity variance? (Section 5.1)
- How sensitive is the effectiveness of capacity variance to different garbage collection (GC) policies? Do GC policies alter WL results? (Section 5.2)
- What is the file system-level overhead with capacity-variant SSDs? Is there a tunable knob for capacity variance? (Section 5.3)

We use the same extended FTLSim [24] from Section 3 and the SSD configuration in Table 1 for our evaluation. However, we set the endurance limit to 500 erases, a typical level for QLC [39,40], and once a block reaches this, it will be mapped out and no longer used in the SSD. This mapping out will effectively reduce the SSD's internal physical capacity. For the fixed-capacity SSD, the SSD is considered to reach its end of life once the physical capacity becomes smaller than its logical capacity. On the other hand, the capacity-variant SSD can reduce its capacity below the initial logical space, to as low as the access footprint for the workload. We assume that the file system will TRIM unnecessary data as the exported

capacity shrinks, and that the overhead for maintaining a contiguous block address space is negligible.

For the workload, we use eleven real-world I/O traces that were collected from running YCSB [41], a virtual desktop infrastructure (VDI) [42], and Microsoft production servers [43]. The traces are modified into a 256GiB range (the logical capacity of the SSD), and all the requests are aligned to 4KiB boundaries. Similar to the synthetic workload evaluation, the SSD is pre-conditioned with one sequential full-drive write and three random full-drive writes on the entire logical space. The traces run in a loop indefinitely, continuously generating I/O until the SSD becomes unusable at its end of life. Table 3 summarizes the trace workload characteristics, only showing the information relevant to understanding the behavior of wear leveling (WL).

Table 3. Trace workload characteristics. YCSB-A is from running YCSB [41], VDI is from a virtual desktop infrastructure [42], and the remaining 9 (from WBS to RAD-BE) are from Microsoft production servers [43]. Footprint is the size of the logical address space that has write accesses, and Hotness is the skewness, where $r\%$ of data are written to the top $h\%$ of the frequently accessed address. Sequentiality is the fraction of write I/Os that are sequential.

Workload	Description	Footprint (GiB)	Avg. Write Size (KiB)	Hotness (r/h)	Sequentiality
YCSB-A	User session recording	89.99	50.48	64.69/35.31	0.49
VDI	Virtual desktop infrastructure	255.99	17.99	64.45/35.55	0.14
WBS	Windows build server	56.05	27.82	60.34/39.66	0.02
DTRS	Developer tools release	150.63	31.85	54.20/45.80	0.12
DAP-DS	Advertisement caching tier	0.17	7.21	78.62/21.38	0.03
DAP-PS	Advertisement payload	36.06	97.20	55.02/44.98	0.16
LM-TBE	Map service backend	239.49	61.90	60.29/39.71	0.94
MSN-CFS	Storage metadata	5.58	12.92	69.28/30.72	0.25
MSN-BEFS	Storage backend file	31.42	11.62	70.18/29.82	0.03
RAD-AS	Remote access authentication	4.53	9.87	70.90/29.10	0.45
RAD-BE	Remote access backend	14.73	13.02	65.51/34.49	0.33

We evaluate the following eight designs.

Fix_NoWL does not run any WL on a fixed-capacity SSD.

Fix_OBP runs *OBP*(4,8) on a fixed-capacity SSD.

Fix_DP runs *DP*(5) on a fixed-capacity SSD.

Fix_PWL runs *PWL*(50) on a fixed-capacity SSD.

Var_NoWL does not run any WL on a capacity-variant SSD.

Var_OBP runs *OBP*(4,8) on a capacity-variant SSD.

Var_DP runs *DP*(5) on a capacity-variant SSD.

Var_PWL runs *PWL*(50) on a capacity-variant SSD.

5.1. Effectiveness of Capacity Variance

Figure 14 shows the amount of data written to the SSD before failure for the 11 I/O traces. The y -axis is in terms of the number of drive writes. For example, for 100 drive writes, 25 TiB of data have been written. Overall, we observe that with fixed-capacity SSDs, running WL is better than not running WL, but only by a small margin: *Fix_DP* extends the lifetime by only 18% on average compared to *Fix_NoWL*, and with workloads such as VDI and DTRS, *Fix_OBP* and *Fix_DP* perform worse than *Fix_NoWL*. However, with capacity variance, not running WL is better than running WL by a large margin: *Var_NoWL* extends the lifetime by $1.33\times$ on average, and as much as $2.94\times$ for RAD-BE compared to *Fix_PWL*. We explain this result by the measurement of write amplification caused by wear leveling. Figure 15 illustrates the WL overhead across different workloads, WL algorithms, and SSD

configurations. Specifically, the write amplification overhead introduced by WL is generally larger in conventional fixed-capacity SSDs (Fix_OBP, Fix_DP, and Fix_PWL), whereas capacity-variant SSDs (Var_OBP, Var_DP, and Var_PWL) mitigate it to some extent, resulting in lower WL-induced write amplification. Consequently, to maximize performance, it is preferable to adopt the capacity-variant SSD without WL (Var_NoWL), as it incurs no WL overhead.

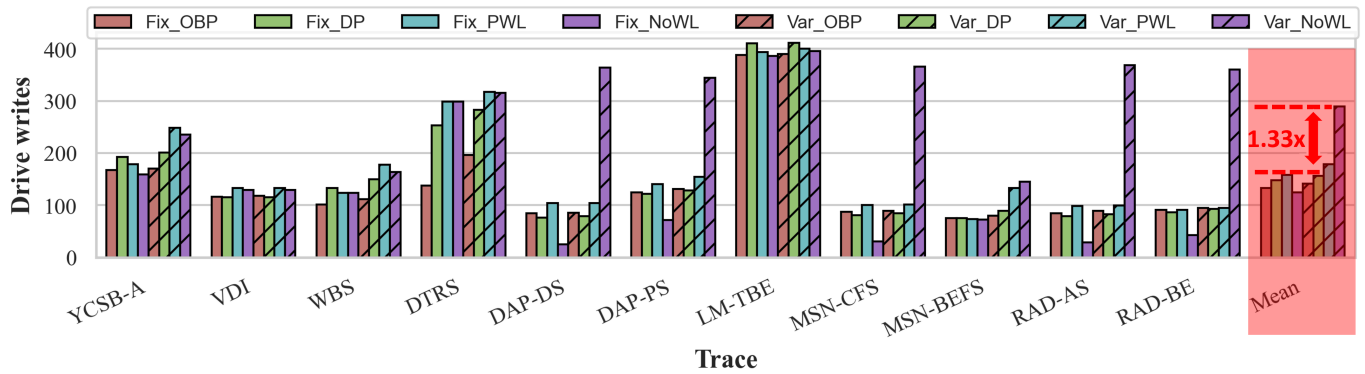


Figure 14. Evaluation of the presence and absence of wear leveling in both a fixed-capacity and a capacity-variant SSD. Compared to the best case with WL (Fix_PWL), the Var_NoWL configuration extends the lifetime by $1.33\times$ on average, and as high as $2.94\times$ in the case of RAD-BE.

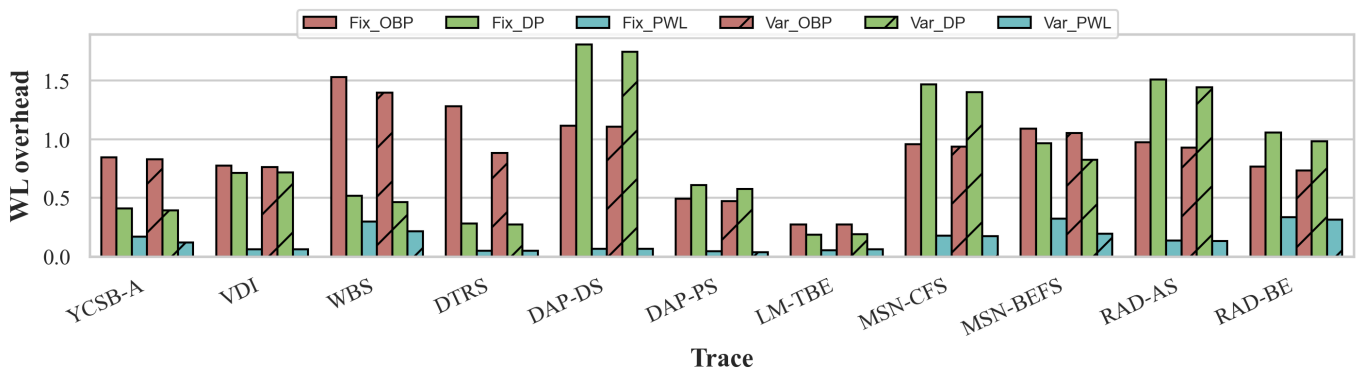


Figure 15. Write amplification caused by WL. While a large sequential workload such as LM-TBE only has a low write amplification overhead of 0.18, most other workloads exhibit high wear-leveling overhead for OBP and DP, reaching as high as 1.8. PWL is aggressive at the late stage but dormant initially, causing the overall overhead to be relatively low.

5.1.1. Workloads with a Relatively Small Footprint

We observe that capacity variance is most effective on workloads such as DAP-DS, DAP-PS, MSN-CFS, RAD-AS, and RAD-BE. These workloads are characterized by a small access footprint where gracefully reducing the capacity allows for the SSD to be continuously used. The lifetime extension gained by capacity variance with wear leveling is minimal. At most, 9.2% is gained from Fix_PWL to Var_PWL, and only 7.7% from Fix_DP to Var_DP, 4.9% from Fix_OBP to Var_OBP. The underlying reason for this is WL causes write amplification (especially on skewed workload), aging all blocks in the SSD. In particular, under a workload such as DAP-DS, the write amplification for Fix_DP is as high as 1.80. A capacity-variant SSD running WL thus has to reduce its capacity abruptly as all blocks are similarly worn out. On the other hand, a capacity variance without wear leveling allows $2.94\times$ more data to be written to the SSD for RAD-BE, and $13.9\times$ for DAP-DS with most gain compared to Fix_NoWL.

MSN-BEFS also has a small footprint, but we observe a comparatively lower lifetime extension of $0.91\times$. In fact, the lifetime of Fix_NoWL is not too far off from that of Fix_DP,

only 5% less. The reason for this is due to garbage collection: This workload contains a lot of small random writes, causing garbage collection to be active, dwarfing the write amplification from WL. Because of this, MSN-BEFS only allows 145 full-drive writes (36.27 TiB) even for the capacity-variant SSD.

5.1.2. Workloads with a Relatively Large Footprint

LM-TBE and VDI are two workloads with the largest footprint, and the benefit of capacity variance is diminished in such workloads. For VDI, *Var_NoWL* reduces the lifetime by 3.1% compared to *Fix_PWL*, and for LM-TBE, *Var_NoWL* reduces it by 3.6% compared to *Fix_DP*. A large footprint means that there is little to gain from reducing the capacity as data are still in use. For LM-TBE, the large sequential write with relatively high uniformity causes the write amplification for WL to be small, as low as 0.18. This allows wear leveling to squeeze more lifetime out of the SSD. This is the only workload where *Var_DP* has the longest lifetime.

DTRS is one of the rare occasions where not running WL is better in a fixed-capacity SSD. *Fix_NoWL* allows 117% more writes compared to *Fix_OBP* and 18% more compared to *Fix_DP*. This is due to the high write amplification of wear leveling. Although the write access pattern of DTRS is fairly uniform, we suspect that a wear leveling anomaly occurred, causing a subset of blocks to age rapidly. Introducing capacity variance extends the lifetime for all three cases, however, with *Var_NoWL* extending the lifetime by 24% compared to *Fix_DP*. *Var_PWL* outperforms *Var_NoWL*, but the difference is only 3%.

5.2. Sensitivity to Garbage Collection

In this experiment, we investigate the effect that garbage collection (GC) policies have on the lifetime of the SSD. GC has a profound impact not only on the overall performance of the drive but also on the lifetime because of its write amplification [44–48]. We compare two GC policies: greedy [49] and cost-benefit [18]. Greedy policy selects the block with the lowest number of valid pages as the victim to clean with the rationale that it would incur the least overhead. However, LFS [18] shows that greedy can lead to a suboptimal performance, especially on skewed workloads because hot, soon-to-be-updated data gets cleaned when it would have been invalidated on its own.

Figure 16 shows the lifetime of a capacity-variant SSD when changing the GC policy, with each subfigure representing a different WL policy. Overall, we observe a negligible difference in the SSD's lifetime when changing GC: on average, the difference is 2.03% for *OBP* in Figure 16a, 0.43% for *DP* in Figure 16b, 0.6% for *PWL* (Figure 16c), and 0.67% without wear leveling (Figure 16d). In fact, the largest difference is for YCSB-A under *Var_NoWL*, and this is only a 6.6% difference. The experiment results here are also consistent with the data presented in Figure 14, where *Var_NoWL* extends the lifetime more than other designs.

We attribute the lack of sensitivity to GC to the following two reasons. First, we experiment on the entire lifetime of the SSD and the WA of WL steadily increases with age. This reduces the overall effect of the GC's WA on the lifetime. Secondly, capacity variance mitigates the WA from GC. In a fixed-capacity SSD, the overhead of GC increases as the effective over-provisioning decreases when physical blocks map out. However, a capacity-variant SSD also reduces the logical space and thereby does not exacerbate the WA from GC. On average, across all workloads, GC-induced WA was $1.66\times$ for the cost-benefit policy and $1.72\times$ for the greedy policy. In comparison, WL-induced WA in capacity-variant SSDs was $1.18\times$ for greedy GC and $1.03\times$ for cost-benefit GC.

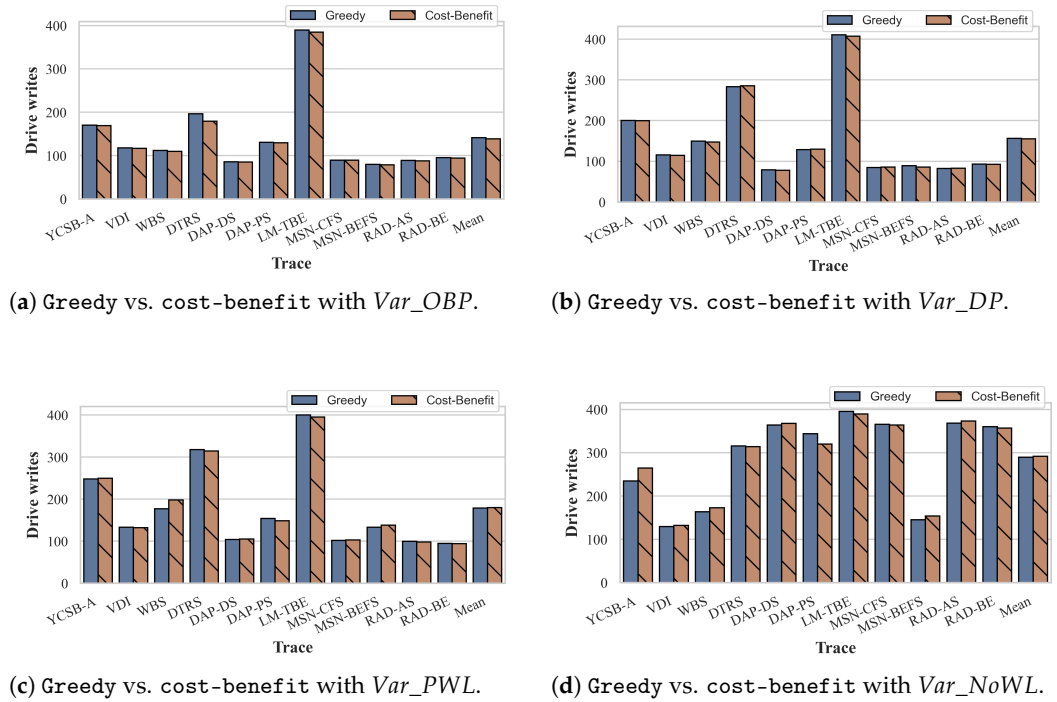


Figure 16. The sensitivity testing of capacity variance to garbage collection (GC) policies. GC has negligible effect on the overall lifetime of the capacity-variant SSD, with at most 6.6% difference between greedy and cost-benefit in the case of YCSB-A without wear leveling. The average difference between the two GC policies across all workloads are 2.03%, 0.43%, 0.6%, and 0.67% for *Var_OBP*, *Var_DP*, *Var_PWL*, and *Var_NoWL*, respectively.

5.3. Limiting File System Overhead

Figure 17 shows the relationship between different file system (FS)-overhead bounds and the maximum amount of data that can be written to the system. The *x*-axis is the FS-level overhead bound we set for the capacity variance and the *y*-axis is the amount of external data written to the SSD under that bound in terms of the number of drive writes. For example, the bound equal to 0 means that the system does not allow any FS-level overhead caused by capacity variance. On the other hand, setting the bounds to be unlimited represents an idealized configuration where the system can dynamically adjust its usable capacity without restriction. Specifically, the available capacity is allowed to shrink down to the workload’s footprint under this configuration (i.e., the minimum amount of capacity that must remain sustain the workload’s operations). No wear leveling is active in this experiment.

We observe that for the workload with a small footprint (DAP-DS, DAP-PS, MSN-CFS, RAD-AS, and RAD-BE), a small amount of capacity reduction allows a significant gain for the SSD lifetime. For example, with a 1GiB allowed overhead for data relocation at the file system, *Var_NoWL* under RAD-AS gains $11.64\times$ lifetime compared to *Fix_NoWL*. On the other hand, for workloads with a large footprint (VDI, DTRS, LM-TBE) we do not see a noticeable increase in the lifetime as we increase the amount of allowed data relocation. However, these workloads typically achieve a long lifetime even without capacity variance. Interestingly, DAP-PS shows a sudden increase in its lifetime when allowing the 18th GiB to relocate. This is due to the large capacity of allocated space in the high address range for DAP-PS.

Overall, Figure 17 demonstrates that capacity variance can be effective even when limiting the file system-level overhead. *Var_NoWL* with only a 1GiB allowance achieves a $3.21\times$ increase in lifetime across all workloads compared to *Fix_NoWL*, $4.07\times$ at 5 GiB,

and $4.10\times$ at 10 GiB. The steep lifetime improvement observed even with a small relocation allowance arises from the data distribution within the file system's address space. In small-footprint workloads, data are typically concentrated at the lower end of the logical block address (LBA) space. Because capacity reduction proceeds from the higher end, a small relocation allowance (e.g., 1 GiB) can enable the system to reclaim several GiBs of logical space with minimal or no data movement. This efficient compaction leads to more lifetime gain relative to the relocation budget.

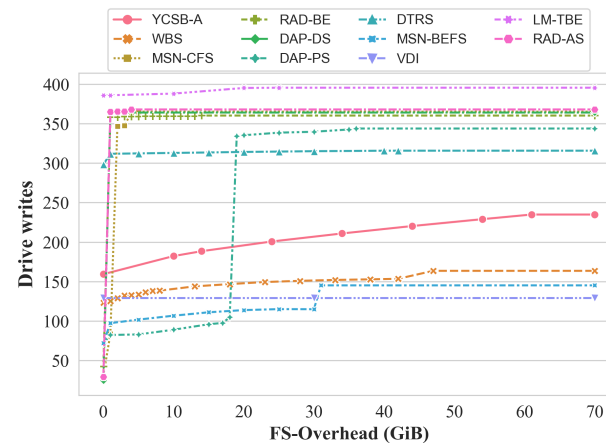


Figure 17. Lifetime of a capacity-variant SSD as a function of maximum allowed file system-level data relocation when no wear leveling is active. With the overhead of relocation set to zero, the capacity-variant SSD behaves identical to that of a fixed-capacity SSD. We observe that with most workloads gain significant lifetime even with a small allowance of data relocation.

6. Discussion and Related Works

Wear leveling is a mature and well-studied topic in both academia and industry [16]. However, to the best of our knowledge, our work is the first that presents a comprehensive evaluation of existing wear-leveling algorithms in the context of today's low endurance limit and quantitatively examines the benefits of capacity variance first introduced in [33]. This work builds upon a number of prior works, and we discuss how it relates to them below.

Wear leveling and write amplification. There exists a large body of work on garbage collection and its associated write amplification (WA) for SSDs, from analytical approaches [24,49–51] to designs supported by experimental results [44,52,53]. However, there is surprisingly limited work that measures the WA caused by wear leveling (WL), and it often relies on a *back-of-the-envelope calculation* for estimating the overhead and lifetime [54]. Even those that perform a more rigorous study evaluate the efficacy of WL by measuring the number of writes the SSD can endure [4,12,55,56] or the distribution of erase count [11,57,58]; only the Dual-Pool algorithm [11] presents the overhead of WL. Our faithful implementation of the Dual-Pool algorithm yields different WA from the original work, however, due to the differences in system configuration and workload.

Lifetime extension without wear leveling. Prior works such as READY [4], Zombi-eNAND [55], and wear unleveling [56] extend SSD lifetime without wear leveling by exploiting self-recovery, reusing expired blocks, or selectively skipping writes. READY [4] dynamically throttles writes to exploit the self-recovery effect of flash memory, and Zombi-eNAND [55] re-uses expired flash memory blocks after switching them into SLC mode. Wear unleveling [56] identifies strong and weak pages and skips writing to weaker ones to prolong the lifetime of the block. Unlike these block- or page-level techniques, our approach introduces a new SSD interface that flexibly adjust usable capacity according to workload characteristics. Capacity-variant SSD is compatible with these existing methods,

and integrating it with them can further enhance SSD lifetime. For example, reusing expired blocks and skipping weak pages can reduce the rate of capacity shrinkage in a variant SSDs. **Zoned namespace.** Zoned namespace (ZNS) [31] is a new abstraction for storage devices that has gained significant interest in the research community [59–61]. ZNS SSDs export large fixed-sized zones (typically in the GiB range) that can be written sequentially or reset (deleted) entirely. The approach shifts the responsibility of managing space via garbage collection from the SSD to the host-side file system. Unlike its predecessor (Open Channel SSD [62]), ZNS does not manage the physical flash memory block and leaves the lifetime management to the underlying SSD, allowing for the integration of capacity variance with ZNS.

Specifically, due to wear-out, a ZNS-SSD may (1) take a zone offline or (2) report a smaller zone size after a reset, both effectively resulting in a shrinking-capacity device. However, current software does not automate capacity reduction for ZNS-SSDs. The offline command simply makes a zone inaccessible, and data relocation must be performed manually. Most file systems are unaware of these changes, except for ZoneFS [63]. In contrast, a capacity-variant SSD interface provides a more streamlined solution by allowing hardware and software to cooperate, automatically handling capacity reduction without manual intervention. Since ZNS-SSDs represent a different class of devices, we plan to integrate and evaluate them in future work.

File systems and workload shaping. It is widely understood that SSDs perform poorly under random writes [64,65], and file systems optimized for SSDs take a log-structured approach so that writes will be sequential [10,29,66–68]. Because of the log-structured nature, these file systems create data of similar lifetime, leading to a reduction in wear-leveling overhead [64]. Similarly, aligning data writes to flash memory’s page size reduces the wear on the device [69].

Flash arrays and clusters. Most existing works at the intersection of RAID and SSDs address poor performance due to the compounded susceptibility to small random writes [70–74]. Of these, log-structured RAID [73,74] naturally achieves good inter- and intra-SSD wear leveling, but at the cost of another level of indirection. On the other hand, Diff-RAID [75] takes a counter-intuitive approach and intentionally skews the wear across the SSDs by distributing the parity blocks unevenly to avoid correlated failures in the SSD array. While we do not advocate the intentional wear out of flash memory blocks, willful neglect of wear leveling can be beneficial in the context of capacity variance.

7. Conclusions

In this work, we examine the performance of wear leveling (WL) in the context of modern flash memory with reduced endurance. Unfortunately, our findings show that existing WL is increasingly ineffective for modern low-endurance SSDs under large, skewed workloads while still providing benefits in small-footprint scenarios. To explore an alternative strategy, we quantify the benefits of capacity variance in SSDs and find that the lifetime can be extended significantly. We hope our findings here can shed light on future SSD designs and the tool we have built can benefit future SSD analysis work.

Author Contributions: Conceptualization, Z.J.; methodology, Z.J.; software, Z.J.; validation, Z.J.; formal analysis, Z.J.; investigation, Z.J.; resources, Z.J.; data curation, Z.J. and B.Y.; writing—original draft preparation, Z.J.; writing—review and editing, Z.J. and B.Y.; visualization, Z.J. and B.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author (privacy and ethical restrictions).

Acknowledgments: We sincerely appreciate the invaluable guidance and support provided by Bryan S. Kim throughout this research. The insightful feedback and help greatly contributed to the development of this study.

Conflicts of Interest: Author Ziyang Jiao is affiliated with Huaibei Normal University. A portion of this research was conducted during the author's study at Syracuse University.

Abbreviations

The following abbreviations are used in this manuscript:

WL	Wear Leveling
GC	Garbage Collection
FS	File System
SSD	Solid-State Drive

References

1. Mielke, N.R.; Frickey, R.E.; Kalastirsky, I.; Quan, M.; Ustinov, D.; Vasudevan, V.J. Reliability of Solid-State Drives Based on NAND Flash Memory. *Proc. IEEE* **2017**, *105*, 1725–1750. [CrossRef]
2. Cai, Y.; Ghose, S.; Haratsch, E.F.; Luo, Y.; Mutlu, O. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proc. IEEE* **2017**, *105*, 1666–1704. [CrossRef]
3. Open NAND Flash Interface. ONFI 5.0 Spec. 2021. Available online: <http://www.onfi.org/specifications/> (accessed on 12 December 2021).
4. Lee, S.; Kim, T.; Kim, K.; Kim, J. Lifetime management of flash-based SSDs using recovery-aware dynamic throttling. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), San Jose, CA, USA, 14–17 February 2012.
5. Jaffer, S.; Mahdavian, K.; Schroeder, B. Rethinking WOM Codes to Enhance the Lifetime in New SSD Generations. In Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage), Online, 13–14 July 2020.
6. Luo, Y.; Cai, Y.; Ghose, S.; Choi, J.; Mutlu, O. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 30 May–5 June 2015; pp. 1–14.
7. Grupp, L.M.; Davis, J.D.; Swanson, S. The bleak future of NAND flash memory. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), San Jose, CA, USA, 14–17 February 2012.
8. Kim, B.S.; Choi, J.; Min, S.L. Design Tradeoffs for SSD Reliability. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Boston, MA, USA, 25–28 February 2019.
9. Jiao, Z.; Bhimani, J.; Kim, B.S. Wear leveling in SSDs considered harmful. In Proceedings of the Workshop on Hot Topics in Storage and File Systems (HotStorage), Online, 27–28 June 2022; pp. 72–78.
10. Gleixner, T.; Haverkamp, F.; Bityutskiy, A. UBI—Unsorted Block Images. 2006. Available online: <http://linux-mtd.infradead.org/doc/ubidesign/ubidesign.pdf> (accessed on 12 December 2021).
11. Chang, L. On efficient wear leveling for large-scale flash-memory storage systems. In Proceedings of the ACM Symposium on Applied Computing (SAC), Seoul, Republic of Korea, 11–15 March 2007.
12. Chen, F.; Yang, M.; Chang, Y.; Kuo, T. PWL: A progressive wear leveling to minimize data migration overheads for NAND flash devices. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, (DATE), Grenoble, France, 9–13 March 2015.
13. Chiang, M.L.; Lee, P.C.; Chang, R.C. Using data clustering to improve cleaning performance for flash memory. *Softw. Pract. Exp.* **1999**, *29*, 267–290. [CrossRef]
14. Murugan, M.; Du, D.H. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST), Denver, CO, USA, 23–27 May 2011.
15. Prabhakaran, V.; Bairavasundaram, L.N.; Agrawal, N.; Gunawi, H.S.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. IRON file systems. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Brighton, UK, 23–26 October 2005.
16. Gal, E.; Toledo, S. Algorithms and data structures for flash memories. *ACM Comput. Surv.* **2005**, *37*, 138–163. [CrossRef]
17. The SSD Guy. Comparing Wear Figures on SSDs. 2017. Available online: <https://thesdsguy.com/comparing-wear-figures-on-ssds/> (accessed on 12 December 2021).
18. Rosenblum, M.; Ousterhout, J.K. The Design and Implementation of a Log-Structured File System. In Proceedings of the ACM Symposium on Operating System Principles (SOSP), Pacific Grove, CA, USA, 13–16 October 1991.
19. Liao, J.; Zhang, F.; Li, L.; Xiao, G. Adaptive wear leveling in Flash-Based Memory. *IEEE Comput. Archit. Lett.* **2015**, *14*, 1–4. [CrossRef]

20. Chang, Y.; Hsieh, J.; Kuo, T. Endurance Enhancement of Flash-Memory Storage, Systems: An Efficient Static Wear Leveling Design. In Proceedings of the Design Automation Conference (DAC), San Diego, CA, USA, 4–8 June 2007; pp. 212–217.
21. Kim, H.; Lee, S. A New Flash Memory Management for Flash Storage System. In Proceedings of the International Computer Software and Applications Conference (COMPSAC), Phoenix, AZ, USA, 25–26 October 1999.
22. Chang, Y.; Hsieh, J.; Kuo, T. Improving Flash Wear-Leveling by Proactively Moving Static Data. *IEEE Trans. Comput.* **2010**, *59*, 53–65. [\[CrossRef\]](#)
23. Chen, Z.; Zhao, Y. DA-GC: A Dynamic Adjustment Garbage Collection Method Considering Wear-leveling for SSD. In Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), Beijing, China, 8–11 September 2020; pp. 475–480.
24. Desnoyers, P. Analytic modeling of SSD write performance. In Proceedings of the International Systems and Storage Conference (SYSTOR), Haifa, Israel, 4–6 June 2012.
25. amsung Electronics Co. Samsung SSD Application Note for Data Centers. 2014. Available online: https://download.semiconductor.samsung.com/resources/others/Samsung_SSD_845DC_04_Over-provisioning.pdf (accessed on 12 December 2021).
26. Rosenblum, M.; Ousterhout, J.K. The Design and Implementation of a Log-Structured File System. Ph.D. Thesis, University of California at Berkeley, Berkeley, CA, USA, 1992.
27. Spanjer, E.; Ho, E. The Why and How of SSD Performance Benchmarking—SNIA. 2011. Available online: https://www.snia.org/sites/default/education/tutorials/2011/fall/SolidState/EstherSpanjer_The_Why_How_SSD_Performance_Benchmarking.pdf (accessed on 12 December 2021).
28. NVM Express. NVM Express Base Specification 2.0. 2021. Available online: <https://nvmexpress.org/developers/nvme-specification/> (accessed on 12 December 2021).
29. Lee, C.; Sim, D.; Hwang, J.Y.; Cho, S. F2FS: A New File System for Flash Storage. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 16–19 February 2015.
30. Mathur, A.; Cao, M.; Bhattacharya, S.; Dilger, A.; Tomas, A.; Vivier, L. The new ext4 filesystem: Current status and future plans. In Proceedings of the Ottawa Linux symposium, Ottawa, ON, Canada, 27–30 June 2007.
31. Western Digital. Zoned Namespaces (ZNS) SSDs. 2020. Available online: <https://zonedstorage.io/introduction/zns/> (accessed on 12 December 2021).
32. Oh, G.; Seo, C.; Mayuram, R.; Kee, Y.; Lee, S. SHARE Interface in Flash Storage for Relational and NoSQL Databases. In Proceedings of the International Conference on Management of Data (SIGMOD), San Francisco, CA, USA, 26 June–1 July 2016; pp. 343–354.
33. Kim, B.S.; Lee, E.; Lee, S.; Min, S.L. CPR for SSDs. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS), Bertinoro, Italy, 12–15 May 2019.
34. Conway, A.; Bakshi, A.; Jiao, Y.; Jannen, W.; Zhan, Y.; Yuan, J.; Bender, M.A.; Johnson, R.; Kuszmaul, B.C.; Porter, D.E.; et al. File Systems Fated for Senescence? Nonsense, Says Science! In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 27 February–March 2017; pp. 45–58.
35. Conway, A.; Knorr, E.; Jiao, Y.; Bender, M.A.; Jannen, W.; Johnson, R.; Porter, D.E.; Farach-Colton, M. Filesystem Aging: It’s more Usage than Fullness. In Proceedings of the Workshop on Hot Topics in Storage and File Systems (HotStorage), Renton, WA, USA, 8–9 July 2019; p. 15.
36. Hahn, S.S.; Lee, S.; Ji, C.; Chang, L.; Yee, I.; Shi, L.; Xue, C.J.; Kim, J. Improving File System Performance of Mobile Storage Systems Using a Decoupled Defragmenter. In Proceedings of the USENIX Annual Technical Conference (ATC), Santa Clara, CA, USA, 12–14 July 2017; pp. 759–771.
37. Ji, C.; Chang, L.; Hahn, S.S.; Lee, S.; Pan, R.; Shi, L.; Kim, J.; Xue, C.J. File Fragmentation in Mobile Devices: Measurement, Evaluation, and Treatment. *IEEE Trans. Mob. Comput.* **2019**, *18*, 2062–2076. [\[CrossRef\]](#)
38. Zhou, Y.; Wu, Q.; Wu, F.; Jiang, H.; Zhou, J.; Xie, C. Remap-SSD: Safely and Efficiently Exploiting SSD Address Remapping to Eliminate Duplicate Writes. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Online, 23–25 February 2021; pp. 187–202.
39. Liang, S.; Qiao, Z.; Tang, S.; Hochstetler, J.; Fu, S.; Shi, W.; Chen, H. An Empirical Study of Quad-Level Cell (QLC) NAND Flash SSDs for Big Data Applications. In Proceedings of the IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 3676–3685.
40. Liu, S.; Zou, X. QLC NAND study and enhanced Gray coding methods for sixteen-level-based program algorithms. *Microelectron. J.* **2017**, *66*, 58–66. [\[CrossRef\]](#)
41. Yadgar, G.; Gabel, M.; Jaffer, S.; Schroeder, B. SSD-based Workload Characteristics and Their Performance Implications. *ACM Trans. Storage* **2021**, *17*, 8:1–8:26. [\[CrossRef\]](#)
42. Lee, C.; Kumano, T.; Matsuki, T.; Endo, H.; Fukumoto, N.; Sugawara, M. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In Proceedings of the ACM International Systems and Storage Conference (SYSTOR), Haifa, Israel, 22–24 May 2017.

43. Kavalanekar, S.; Worthington, B.L.; Zhang, Q.; Sharda, V. Characterization of storage workload traces from production Windows Servers. In Proceedings of the International Symposium on Workload Characterization (IISWC), Seattle, WA, USA, 14–16 September 2008.
44. Kang, W.; Shin, D.; Yoo, S. Reinforcement Learning-Assisted Garbage Collection to Mitigate Long-Tail Latency in SSD. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 134:1–134:20. [[CrossRef](#)]
45. Kim, B.S.; Yang, H.S.; Min, S.L. AutoSSD: An Autonomic SSD Architecture. In Proceedings of the USENIX Annual Technical Conference (ATC), Boston, MA, USA, 11–13 July 2018; pp. 677–690.
46. Dean, J.; Barroso, L.A. The tail at scale. *Commun. ACM* **2013**, *56*, 74–80. [[CrossRef](#)]
47. Hao, M.; Soundararajan, G.; Kenchammana-Hosekote, D.R.; Chien, A.A.; Gunawi, H.S. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 22–25 February 2016; pp. 263–276.
48. Yan, S.; Li, H.; Hao, M.; Tong, M.H.; Sundararaman, S.; Chien, A.A.; Gunawi, H.S. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 27 February–March 2017; pp. 15–28.
49. Yang, Y.; Misra, V.; Rubenstein, D. On the Optimality of Greedy Garbage Collection for SSDs. *SIGMETRICS Perform. Eval. Rev.* **2015**, *43*, 63–65. [[CrossRef](#)]
50. Hu, X.; Eleftheriou, E.; Haas, R.; Iliadis, I.; Pletka, R.A. Write amplification analysis in flash-based solid state drives. In Proceedings of the Israeli Experimental Systems Conference (SYSTOR), Haifa, Israel, 4–6 May 2009; p. 10.
51. Park, C.; Lee, S.; Won, Y.; Ahn, S. Practical Implication of Analytical Models for SSD Write Amplification. In Proceedings of the ACM/SPEC on International Conference on Performance Engineering (ICPE), L'Aquila, Italy, 22–26 April 2017; pp. 257–262.
52. Chang, L.; Kuo, T.; Lo, S. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. Embed. Comput. Syst.* **2004**, *3*, 837–863. [[CrossRef](#)]
53. Zhang, Q.; Li, X.; Wang, L.; Zhang, T.; Wang, Y.; Shao, Z. Optimizing deterministic garbage collection in NAND flash storage systems. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Seattle, WA, USA, 13–16 April 2015; pp. 14–23.
54. Zhang, T.; Zuck, A.; Porter, D.E.; Tsafir, D. Flash Drive Lifespan *is* a Problem. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS), Whistler, BC, USA, 7–10 May 2017; pp. 42–49.
55. Wilson, E.H.; Jung, M.; Kandemir, M.T. ZombieNAND: Resurrecting Dead NAND Flash for Improved SSD Longevity. In Proceedings of the IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), Paris, France, 9–11 September 2014; pp. 229–238.
56. Jimenez, X.; Novo, D.; lenne, P. Wear unleveling: Improving NAND flash lifetime by balancing page endurance. In Proceedings of the USENIX conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 17–20 February 2014; pp. 47–59.
57. Agrawal, N.; Prabhakaran, V.; Wobber, T.; Davis, J.D.; Manasse, M.S.; Panigrahy, R. Design Tradeoffs for SSD Performance. In Proceedings of the USENIX Annual Technical Conference (ATC), Boston, MA, USA, 22–27 June 2008; pp. 57–70.
58. Huang, J.; Badam, A.; Caulfield, L.; Nath, S.; Sengupta, S.; Sharma, B.; Qureshi, M.K. FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 27 February–March 2017; pp. 375–390.
59. Bjørling, M.; Aghayev, A.; Holmberg, H.; Ramesh, A.; Moal, D.L.; Ganger, G.R.; Amvrosiadis, G. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In Proceedings of the USENIX Annual Technical Conference (ATC), Online, 14–16 July 2021; pp. 689–703.
60. Han, K.; Gwak, H.; Shin, D.; Hwang, J. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), Online, 14–16 July 2021; pp. 147–162.
61. Stavrinou, T.; Berger, D.S.; Katz-Bassett, E.; Lloyd, W. Don't be a blockhead: Zoned namespaces make work on conventional SSDs obsolete. In Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS), Ann Arbor, MI, USA, 1–3 June 2021; pp. 144–151.
62. Bjørling, M.; Gonzalez, J.; Bonnet, P. LightNVM: The Linux Open-Channel SSD Subsystem. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST), Santa Clara, CA, USA, 27 February–March 2017; pp. 359–374.
63. Le Moal, D.; Yao, T. Zonefs: Mapping POSIX File System Interface to Raw Zoned Block Device Accesses. In Proceedings of the Linux Storage and Filesystems Conference (VAULT). USENIX, Santa Clara, CA, USA, 24–25 February 2020; p. 19.
64. He, J.; Kannan, S.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. The Unwritten Contract of Solid State Drives. In Proceedings of the European Conference on Computer Systems (EuroSys), Belgrade, Serbia, 23–6 April 2017; pp. 127–144.
65. Chen, F.; Koufaty, D.A.; Zhang, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Seattle, WA, USA, 15–19 June 2009; pp. 181–192.

66. Min, C.; Kim, K.; Cho, H.; Lee, S.; Eom, Y.I. SFS: Random write considered harmful in solid state drives. In Proceedings of the USENIX conference on File and Storage Technologies (FAST), San Jose, CA, USA, 14–17 February 2012; p. 12.
67. Zhang, J.; Shu, J.; Lu, Y. ParaFS: A Log-Structured File System to Exploit the Internal Parallelism of Flash Devices. In Proceedings of the USENIX Annual Technical Conference (ATC), Denver, CO, USA, 22–24 June 2016; pp. 87–100.
68. One, A. Yet Another Flash File System. Available online: <http://www.yaffs.net> (accessed on 12 December 2021).
69. Kakaraparthi, A.; Patel, J.M.; Park, K.; Kroth, B. Optimizing Databases by Learning Hidden Parameters of Solid State Drives. *Proc. VLDB Endow.* **2019**, *13*, 519–532. [[CrossRef](#)]
70. Jeremic, N.; Mühl, G.; Busse, A.; Richling, J. The pitfalls of deploying solid-state drive RAIDs. In Proceedings of the Annual Haifa Experimental Systems Conference (SYSTOR), Haifa, Israel, 30 May–1 June 2011; p. 14.
71. Moon, S.; Reddy, A.L.N. Does RAID Improve Lifetime of SSD Arrays? *ACM Trans. Storage* **2016**, *12*, 11:1–11:29. [[CrossRef](#)]
72. Kim, Y.; Oral, S.; Shipman, G.M.; Lee, J.; Dillow, D.; Wang, F. Harmonia: A globally coordinated garbage collector for arrays of Solid-State Drives. In Proceedings of the IEEE Symposium on Mass Storage Systems and Technologies (MSST), Denver, CO, USA, 23–27 May 2011; pp. 1–12.
73. Kim, J.; Lim, K.; Jung, Y.; Lee, S.; Min, C.; Noh, S.H. Alleviating Garbage Collection Interference Through Spatial Separation in All Flash Arrays. In Proceedings of the USENIX Annual Technical Conference (ATC), Renton, WA, USA, 10–12 July 2019; pp. 799–812.
74. Chiueh, T.; Tsao, W.; Sun, H.; Chien, T.; Chang, A.; Chen, C. Software Orchestrated Flash Array. In Proceedings of the International Conference on Systems and Storage (SYSTOR), Haifa, Israel, 10–12 June 2014; pp. 14:1–14:11.
75. Balakrishnan, M.; Kadav, A.; Prabhakaran, V.; Malkhi, D. Differential RAID: Rethinking RAID for SSD reliability. In Proceedings of the European Conference on Computer Systems (EuroSys), Paris, France, 13–16 April 2010; pp. 15–26.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.