Leveraging Software Fault Tolerance for Longer Flash Hardware Lifespan

Aviad Zuck
Technion

Rob Johnson
Broadcom

Donald E. Porter
UNC — Chapel Hill

Dan Tsafrir
Technion

ABSTRACT

In the context of data center computing, the carbon emission incurred during the manufacturing of solid-state drives (SSDs), called embodied carbon, is increasingly overtaking the emissions caused by powering these drives throughout their operational lifetime. Out of an abundance of caution, current SSDs are designed to fail long before the internal flash components are completely unusable. Although this choice may be appropriate for single-disk systems, distributed storage systems can already tolerate individual SSD failures through redundancy.

We propose Salamander, a new design for SSD servers where SSDs expose multiple logical *minidisks*, which match the granularity of hardware failures, so that: (1) as minidisks fail, distributed storage systems can continue using the remaining good capacity; and (2) SSDs may recycle portions of failed minidisks that still have some usable life. Salamander increases the lifespan of SSDs, and therefore amortizes embodied carbon, by incrementally exposing hardware deterioration to the system and leveraging existing, end-to-end redundancy mechanisms to recover from this deterioration.

CCS CONCEPTS

• **Hardware → Impact on the environment**.

ACM Reference Format:
Aviad Zuck, Rob Johnson, Donald E. Porter, and Dan Tsafrir. 2025. Leveraging Software Fault Tolerance for Longer Flash Hardware Lifespan. In *Workshop on Hot Topics in Operating Systems (HOTOS '25), May 14–16, 2025, Banff, AB, Canada.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3713082.3730386

1 INTRODUCTION

As the prevalence of SSDs in datacenters increases, so does their role in the sustainability of these systems [1–4]. SSDs are a key storage medium in distributed systems [5–7], including cloud storage, due to their superior performance, power efficiency, and improving cost effectiveness (i.e., $/GB).

This paper argues that we should lower the carbon footprint of SSD servers by extending device lifetime, at the cost of tolerating partial failures. A recent estimate of SSD server sustainability [7] indicates that SSDs are responsible for up to 80% of embodied emissions (related to raw materials and manufacturing) and 38% of operational emissions. Renewable energy sources are expected to offset operational carbon in datacenters [8], leaving embodied emissions as the primary source of datacenter emissions. Previous approaches to decarbonize SSD storage proposed to increase densities [9] and extend SSD lifetime [8]. This work expands upon recent proposals [7, 10] to lower the embodied carbon footprint of SSDs through extended device lifetime.

The primary factor determining SSD lifetime is the endurance of underlying flash media. The raw bit-error rate (RBER) of pages grows proportionally to the number of prior writes [11]. To mitigate this deterioration, flash pages include additional, hidden space, called a **spare area**, that stores error-correction codes (ECC). The relative size of the data and spare areas determines the **code rate**, i.e., $\frac{data}{data+spare}$, and number of correctable bit errors [12], which in turn determines the maximum number of device writes that an SSD can endure. A typical flash page spare code rate is 88% [13]

In practice, SSDs are preemptively retired before the internal flash is completely worn out (§2). Datacenter operators usually retire SSDs after several years of operation due to a combination of: (1) averting unexpected data loss from worn-out SSDs; and (2) replacing SSDs with newer, denser models. Further, SSD firmware is typically designed to stop functioning when some minimal threshold of worn-out flash blocks (or erase units) is exceeded (e.g., 2.5% [14]), Consequently, when an SSD stops functioning, there is considerable lifetime potential left on many of the flash blocks [15].

Prior work established that SSDs can be used longer than current practice if one can tolerate a significant reduction in the free space of local file systems [16]. In the context of a single device, such as a phone or laptop, losing blocks storing data can be a show-stopper, rendering the entire system unusable [17]. Put differently, the efficacy of this idea is limited by free space in the device. Prior work also explored the potential for extending flash lifespan using lower code

HOTOS '25, May 14–16, 2025, Banff, AB, Canada

ACM ISBN 979-8-4007-1475-7/2025/05
https://doi.org/10.1145/3713082.3730386

rates [18–23]. This paper contributes the observation that a distributed file system can seamlessly handle device errors through redistribution of replicated data, obviating the need for sacrificial free space as the SSDs age.

Our first key observation is that SSDs should expose many logical **minidisks** to a distributed storage system, rather than a single, monolithic volume. Existing distributed systems handle SSDs as large failure units, as all underlying flash units fail at once. For example, suppose that within a 128 GiB SSD, 4 GiB of internal storage wears out, putting it above the 2.5% failure threshold. At this point, the device would stop working in practice. With current shrinking device proposals [16, 24], one could simulate a failure of the entire SSD, recreate a smaller version at, say, 125 GiB, and recover the full 125 GiB from other devices in the system. In contrast, if one assumes the granularity of failure within the SSD is 1 GiB, one could expose 128 × 1 GiB minidisks to the distributed storage system. If three minidisks wear out, the system need only recover 4 GiB of data by replicating these minidisks' contents elsewhere. By matching the device's failure granularity to what software expects, one can logically "shrink" the device with lower overheads.

A second key observation of this paper is that one can dynamically decrease the code rate (i.e., increase redundancy) as flash wears, extending lifespan at the cost of space and latency. Current SSDs have a fixed ratio of spare area to data area, reflecting the abstraction of a fixed-sized disk. After a minidisk logically fails, some space within that disk could still be useful at a lower code rate. As a simple example, if two minidisks fail with a bit error rate above what the default ECC can handle, one could combine two failed minidisks into a new one that internally uses a lower code rate. The reconstructed minidisk uses more space and may have higher latency than a minidisk on a new device; the alternative, however, is not using the space at all. We acknowledge that some datacenter applications are latency critical and would prefer to lose storage rather than slow it down; we also note that there are many users and applications that are more sensitive to cost or environmental concerns than latency [25].

This paper presents Salamander[1], a new design for SSD-based distributed systems. The design of Salamander includes two modes (§3). Shrinking Salamander (ShrinkS) reduces SSD usable capacity by discarding flash pages that become too worn out to reliably store data using pre-configured ECC capabilities, and triggering a recovery event in the distributed file system. A regenerating Salamander (RegenS) further utilizes worn-out flash pages by re-purposing data bits to store additional ECC bits. RegenS more gracefully degrades capacity in order to further extend lifetime, though at some performance overhead. Salamander further minimizes changes to storage systems by exposing the same SSD abstraction, but with finer-grain failure units, thereby utilizing existing failure recovery recovery logic.

Our analysis (§4) indicates that Salamander can extend flash lifetime by up to 1.5x. Furthermore, our analysis shows a potential 8% reduction in embodied emissions of distributed systems, even when accounting for the efficiency gains of replacing older SSDs with newer SSDs. By having devices fail more gradually, there is less risk of unexpected data loss from a device failure. This change in turn alleviates the need for premature, preemptive device retirement, as well as paves the way for the use of less endurant, cheaper flash [8].

2 FLASH RELIABILITY IN PRACTICE

Flash devices access data at the granularity of **flash pages**—groups of cells that are typically 4–16KB in size. A **flash block** is a group of several hundred pages. Flash endurance is measured in the number of program/erase cycles (PEC) that pages can endure before encountering uncorrectable bit errors. Over the course of repeated P/E cycles, charge becomes trapped in the flash cells, skewing subsequent voltage measurements and ultimately flipping bits. Bit errors can also be caused by other factors such as read disturbances from neighboring pages [26].

Flash vendors take several measures to mitigate flash errors [12, 26]. Primarily, each page is augmented with an additional *spare* area (e.g., 12%) dedicated for ECC. Another important mechanism is iterative voltage adjustment, which attempts to compensate for voltage skew as the device ages, at the cost of increased read latency.

Blocks that are too worn-out to reliably store data are replaced with spare ones from the drive's over-provisioned space until reaching some threshold of failed blocks (e.g., 2.5% [14]) beyond which SSDs either fail entirely (i.e., brick) or become read-only.

SSD vendors provide a concrete estimate for the expected lifetime of SSDs in drive writes per day (DWPD) during their warranty period, derived from the PEC limit. SSD warranties tend to be highly conservative [17, 27].

2.1 SSD Life Cycle in Datacenters

SSD failures due to underlying flash wear are perceived as a major concern. Multiple works have analyzed SSD failures over long periods in large, distributed deployments [14, 28–32]. Perhaps counter-intuitively, reported SSD annual failure rates (AFR) are relatively low (e.g., ~1%) and do not necessarily increase with age and use. To wit, studies by NetApp [14, 32] indicate that 60-98% of SSDs "do not even use up 1% of their PEC limit", concluding that "SSDs will last for more than 100 years in production without wearing out".

[1]Salamanders have the ability to regenerate lost organs.

Moreover, our discussions with industry experts indicate that datacenter operators regularly and proactively replace SSDs after several years — long before they fail, mostly due to: (1) preemptive failure mitigation to avoid costly unscheduled replacements; and (2) introducing newer, high-capacity, more power-efficient models. This is consistent with another recent observation that discarded datacenter SSDs are used "well under their ratings" [33]. Recent results further suggest that when a hardware component within a drive fails, this typically happens before the SSD reaches 50-70% of the SSD's expected lifetime [34, 35].

Discussion. Modern datacenters rarely use all of the write capacity in SSDs before they are replaced, which adversely affects system sustainability due to the embodied emissions of new replacement SSDs [2, 3]. Replacing drives with newer, denser models can potentially reduce embodied emissions by requiring fewer SSDs for the same capacity. However, several recent studies indicate that carbon intensity likely scales with density for 3D stacking, the process with which modern FABs increase flash densities [2, 3, 7, 36]. Prior work shows increased embodied carbon outweigh the gains from improved power efficiency [25]. Because storage device failures are already handled well by distributed storage systems, preemptive replacement of aged SSDs is of marginal value.

Prior work on extending SSD lifespan gains additional space for additional ECC/parity by sacrificing some data storage capacity [19–22, 37, 38], which this paper extends. Another, orthogonal approach to lifespan extension changes how the voltage within the cell is discretized into bits, e.g., switching from a TLC (3 bits per cell) to MLC (2 bits per cell) [39, 40].

3 SSD SHRINKING

This section describes the design of Salamander, a new class of SSDs for distributed systems, which aim to: (1) extend flash lifetime utilization beyond current, artificial limits; and (2) integrate seamlessly into a distributed storage system.

Salamander drives expose their logical block address (LBA) space as multiple **minidisks** (***mDisks***), or small-capacity, logical units that appear to the system as independent, tiny drives. We describe two modes of operation: **ShrinkS** and **RegenS**. In ShrinkS, once an *mDisk*-worth of flash pages become too worn out, the flash pages are preemptively retired, along with a logical *mDisk*. RegenS expands ShrinkS and further extends flash page endurance. When pre-configured ECC is insufficient for reliable storage, RegenS converts some data capacity from failed *mDisks* to extra ECC, thereby prolonging the lifetime of the remaining data capacity. We envision Salamander being implemented primarily in SSD firmware.

term	**definition**
diFS	distributed file system
LBA	host logical block address
oPage	logical data page in an *fPage* (e.g., 4KB)
fPage	flash physical page containing *oPages*
mDisk	minidisk
mSize	size of *mDisk* (e.g., 1MB)
L(*fPage*)	*fPage* tiredness level
$limbo[L_j]$	# of *fPages* with tiredness level $j \in 1 \ldots 4$
$CO_2e(X)$	Carbon footprint of server deployment X
f_{op}	Fraction of operational emissions of total
f_{opex}	Fraction of operational costs of total
$PE_{A\|B}$	Power effectiveness of SSD A relative to B
$Ru_{A\|B}$	Upgrade rate of SSDs in A relative to B
$CRu_{A\|B}$	Cost upgrade rate of SSDs in A relative to B

Table 1: Terms used to describe and analyze Salamander.

Terminology. When a new SSD drive is introduced into a distributed filesystem, it is logically partitioned into equally-sized access units (e.g., an HDFS 128MB block) which are stored redundantly. We use *diFS* to represent a distributed file system. Let *oPage* denote a regular 4KB (OS) page and *fPage* denote an SSD-internal flash page (capable of housing several *oPages*).

We assume that *fPage* is the IO granularity for internal SSD parallelism and that the SSD allows access at *oPage* granularity. For concreteness, in this section we assume an *fPage* size of 16KB, housing four *oPages* (§4.2 discusses other sizes). Notably, modern flash chips exhibit high variance in the error rate across pages even within the same block [41, 42]. Therefore, Salamander retires flash pages individually. Table 1 summarizes the main terms used in this section.

3.1 Page Tiredness

The PEC count of *fPages* determines their wear level and is regularly tracked by a Salamander SSD using a compact bit array. Over time when an *fPage* becomes "tired", i.e., too worn-out for its ECC to reliably hide its bit errors, the *fPage*'s valid *oPages* are relocated to a fresh *fPage*. However, unlike standard SSDs, in Salamander a tired page may still be used to store data. Specifically, each *fPage* in Salamander has a "tiredness level" $L \in \{0, 1, 2, 3, 4\}$, where $L(fPage)$ is the number of *oPages* in the *fPage* that are re-purposed for *extra* ECC to support reliable data storage in the rest of the *oPages*.

Page tiredness level L is related to erase cycles experienced so far. Thus, L_0 is associated with young *fPages* that store regular data in all of their four *oPages*; L_1 is associated with

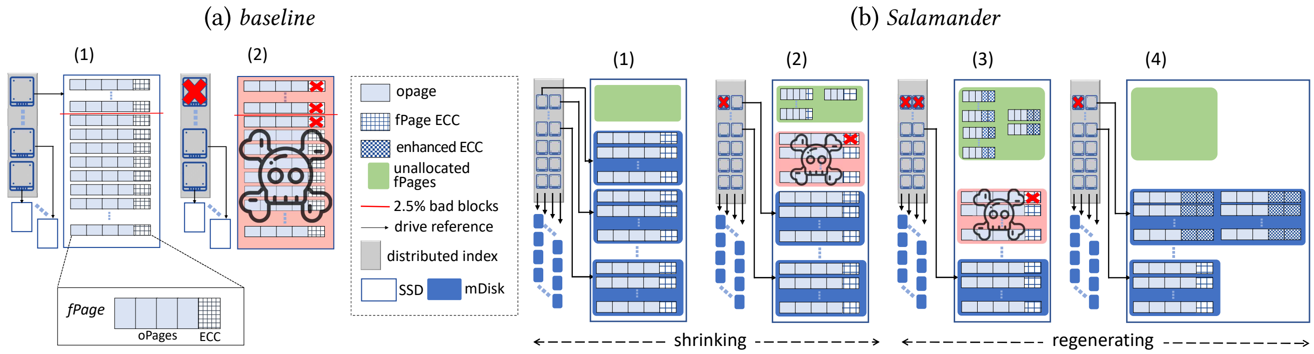

Figure 1: *Flash units in a baseline distributed system (a1) gradually fail. The drive fails when a small internal threshold of bad flash units is reached (a2). In Salamander data is replicated at finer "mini" disk granularity (b1). As mDisks fail the drive shrinks but continues functioning (b2). To further extend lifetime the drive can regenerate mDisks by deleting oPages in failed fPages for extra ECC (b3) until an mDisk-worth of oPages are available and the device creates a new mDisk (b4).*

fPages that must sacrifice one *oPage* for additional ECC; and L_4 means the *fPage* can no longer reliably store any data.

3.2 Minidisks

When a new Salamander SSD is introduced into the *diFS*, the device appears as N equally-sized *mDisks*, as illustrated in Fig. 1. The *diFS* treats these as separate failure domains and issues I/O operations to these *mDisks* separately.

Let *mSize* denote the *mDisk* size, which is set to match the granularity of SSD-internal hardware failures. To achieve granular failure and recovery processes, we currently assume *mSize* is small, e.g., 1MB. Each *mDisk* exposes an independent logical block address (**LBA**) space. Each LBA is internally mapped to a different *oPage* containing its data. For example, a 1MB *mDisk* maps LBAs 0-255 onto 256 different *oPages*, i.e., 1 MB of flash internal storage.

LBAs and Reads. An *mDisk* is only a logical abstraction. Each *mDisk* M_i ($i \in 1 \ldots N-1$) has a separate logical address space (i.e., LBAs), implemented as a set of consecutive indices in the in the SSD's internal logical-to-physical mapping. LBAs in an *mDisk* may be mapped to any *oPage* within the SSD. Concretely, a read request to an LBA j in *mDisk* M_i is translated into an internal index $< i, j >$ in the mapping array, which refers to a specific *fPage*.

Writes. oPage writes for any *mDisk* are performed to the next available *fPage*. Whenever a write to LBA j in *mDisk* M_i is issued, the SSD buffers the written data in a small non-volatile buffer until enough data is cached to fill all *oPages* in the next available *fPage*. For example, for a 16KB *fPage* if the next available *fPage* is of L_0, then whenever the equivalent of four *oPages* are buffered they are evicted and written to the *fPage*. The relevant mapping entries are updated accordingly so that the mapping entry in index $< M_i, j >$ points to the relevant *oPage* and *fPage* (e.g., 1st *oPage* in *fPage* 100).

An open design question for future work is how to navigate the trade-off between flexibility in mapping *mDisks* onto *fPages* and the potential for correlated failures in *mDisks*. It may be that simple placement rules within the SSD suffice, or that the best place to manage this risk is in the *diFS*.

3.3 Minidisk Decommissioning

Assuming every *fPage* contains four *oPages*, then the number of *oPages* that can be stored in a page with tiredness level L_j is $4 - j$. Let $limbo[L_j]$ denote the overall number of *fPages* with a tiredness level of L_j. The number of *oPages* that can be stored in limbo pages of type j is

$$valid[limbo[L_j]] = (4 - j) \cdot limbo[L_j] \quad (1)$$

Whenever an *fPage* accumulates more writes (i.e., wear) and transitions to some L_{j+1} the SSD checks whether the volume of available physical space is insufficient to store the device's logical capacity, i.e., if

$$\sum_{j=0}^{3} |valid[limbo[L_j]]| < |LBAs| \quad (2)$$

If so, the SSD proactively triggers decommissioning for a victim *mDisk* by retiring $\frac{mSize}{4KB}$ *oPages*, e.g., 256 for a 1MB *mDisk*.

To this end, the SSD preemptively retires the most worn-out *fPages* in L_j, regardless of their related *mDisk*, and relocates their data to less worn-out *fPages* until enough *oPages* are relocated. In this process only the *oPages* of the victim minidisk are invalidated, their associated *fPages* are placed in limbo according to their tiredness level L_j, and relevant $limbo[L_j]$ counts are updated accordingly. Once re-location is completed, the victim minidisk is decommissioned and the *diFS* is also notified so that it can recover data as usual

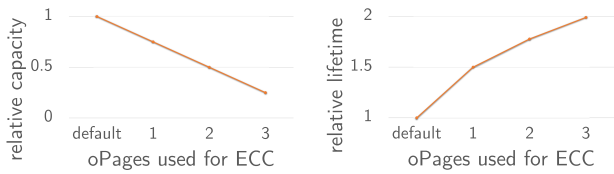

Figure 2: *Switching oPages to additional ECC trades capacity for increasingly diminishing lifetime benefits.*

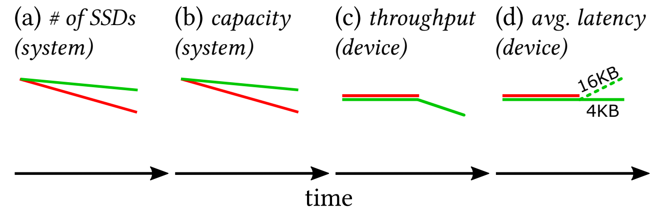

Figure 3: *Baseline SSDs (red) gradually fail and reduce system capacity (3b). For RegenS (green) worn-out devices can shrink and regenerate and reduce the rate of device failures (3a). RegenS degrades device performance for large accesses as more fPages transition to L_1 (3c and 3d).*

on other drives (e.g., re-replicate). Fig. 1b (left) illustrates an *mDisk* decommissioning.

3.4 Minidisk Regeneration

By reviving aging flash pages, we can further improve upon the lifetime gains of ShrinkS, with potential performance overheads (§4). For applications that can tolerate some performance degradation, we propose RegenS, which re-uses partially worn-out pages to regenerate new *mDisks*. The key idea is to augment a retired *fPage* with additional retired *oPages* that are repurposed to store additional ECC.

In order to implement regeneration, when an *fPage* is retired and transitions from tiredness level j to $j + 1$, the SSD firmware must track whether enough *oPages* are available to form a new *mDisk* at tiredness level $j + 1$. For simplicity, we assume all *oPages* in a *mDisk* have the same tiredness level, and leave analysis of mixed tiredness for future work. If enough *oPages* are available, but not used, a new *mDisk* is created, N incremented accordingly, and the host is notified to introduce the new *mDisk* to *diFS*. Fig. 1b (right) illustrates an *mDisk* resurrection.

4 IMPLICATIONS

ShrinkS extends SSD lifetime by slowly reducing capacity, similarly to CVSS [16]. Unlike CVSS, the potential for lifetime extension in ShrinkS does not hinge on available free space in the host file system. Moreover, CVSS retires entire flash blocks whose average RBER is too high, potentially missing much of the remaining lifetime of stronger pages within blocks. We therefore conservatively assume that ShrinkS can extend device lifetime at least as well as CVSS, which reports a ~20% improvement in lifetime, given only 50% space utilization.

RegenS further trades device capacity for additional ECC capability and increased flash lifetime. To illustrate, we use known models for flash RBER [11] and the relationship between code rate and ECC capability [12]. For simplicity we only consider RBER due to aging. Fig. 2 illustrates the resulting relation between page tiredness levels (i.e., code rate) and PEC benefits for an example with a standard 16KB *fPage* (four *oPages*) and a 2KB spare [13], which has a 50% potential lifetime benefit for L_1. Due to the marginal utility of reusing very worn *oPages*, we conclude that, realistically, RegenS should limit itself to $L < 2$, noting that a sparsely populated *diFS* may continue shrinking for $L \geq 2$.

Figures 3a and 3b illustrate the number of functioning SSDs and available capacity over time for a batch of SSDs deployed in a distributed system. Because RegenS devices can shrink, this slows the rate of wear-related device failures and capacity reduction, flattening the slope (red) compared to the baseline (green).

4.1 Sustainability

To estimate the potential of Salamander for decarbonizing datacenters, including the effects of changing hardware upgrade rates, we apply a similar methodology to [43], using terms defined in Table 1. Therefore, the carbon footprint of a server deployment using Salamander drives (S), relative to baseline (B) is defined as:

$$f_{op} \cdot PE_{S|B} \cdot CO_2e(B) + (1 - f_{op}) \cdot Ru_{S|B} \cdot CO_2e(B) \quad (3)$$

According to a recent estimate [25] operational emissions account for 58% of datacenter-related emissions ($f_{op} = 0.58$). However, in addition to SSD storage servers, datacenters also include HDD and compute servers whose emissions are more dominated by operational emissions [3, 25]. We therefore apply a conservative factor of 20% for SSD servers only, i.e., $f_{op} = 0.46$. Notably, this value may effectively be even lower since the carbon model in [25] appears to be based on an SSD carbon intensity estimate (17.3 kgCO2e/TB) that is significantly lower than other recent estimates [3, 7].

Prior work estimates the increase in operational emissions for the same workloads by 6% according to [25] when not replacing drives with newer, more power-efficient models. We set the power effectiveness, $PE_{S|B} = 1.06$.

We model the relative upgrade rates of SSDs based on the estimated lifetime benefits of at least 20% for ShrinkS and 50% for RegenS. From here, $Ru_{S|B} = \frac{1}{1.2} = 0.83$ and $Ru_{S|B} = 0.66$.

We now consider the impact of extending SSD use at the system level. First, we consider overall system capacity. For L_0, Salamander and baseline SSDs have the same capacity.

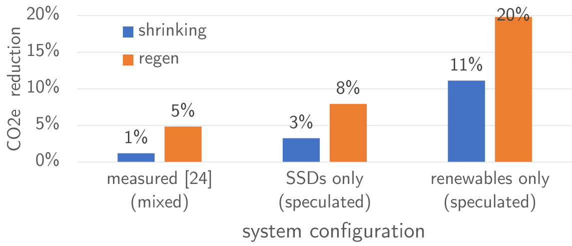

Figure 4: *CO2e reduction in different system configurations.*

For L_1 we assume Salamander drives shrink up to 20% of original capacity, so average SSD capacity is 60% of baseline, or 86% for a RegenS device lifetime. In both cases system operators may add new SSDs to offset missing capacity. However, baseline SSDs fail more frequently with reported AFR of 1-3% [28] (Fig. 3a, 3b) which further requires additional SSDs. These two behaviors partially cancel out in terms of emissions. Nevertheless, we conservatively fix $Ru_{S|A}$ gains by 40% to 0.9 for ShrinkS and 0.8 for RegenS.

Overall, using Eq. 3, we conclude that Salamander achieves 3–8% CO2e savings in current designs. One easy carbon reduction on the horizon for datacenters is renewable energy sources; if one considers the reduction in carbon of Salamander relative to an estimated total carbon footprint when using only renewables, these gains increase to 11–20%, illustrated in the rightmost set of bars in Fig. 4.

4.2 Performance

ShrinkS performance is similar to a baseline SSD's, since *fPages* are retired at similar RBER levels in both setups. In RegenS, L_1 *fPages* require more IO operations to access the same amount of data as in L_0. Therefore, sequential access throughput and large random access latency (e.g., 16KB) degrades by a factor of $\frac{4}{4-L}$ for a given L (Fig. 3c, 3d), e.g., 25% reduction for L_1. To mitigate, RegenS may store more ECC in dedicated pages, which may also fit SSDs with $fPage < 16KB$.

We expect that small, random accesses (i.e., 4 KiB pages) will likely have the same latency in baseline and RegenS. Although L_1 *fPages* have a higher RBER, potentially incurring overheads for ECC computation and additional read retries, this is likely mitigated the lower code rate of these pages [44–46], provided all of the ECC is within the same *fPage*.

4.3 Recovery

We estimate that the volume of recovery traffic using *mDisks* will be comparable to the baseline, at least without regeneration, because the same total number of LBAs fail over time. Baseline SSD failure is logically equivalent to retiring all flash blocks simultaneously. A ShrinkS SSD similarly transitions flash blocks from tiredness level L_0 to L_1, but over a longer period, and at an increasing rate as the device ages. Recovery traffic with regeneration is more complicated, since the regenerated *mDisks* increase the total data that will fail, and are shorter lived than the original *mDisks*. As future work we will explore including a short grace period for *mDisk* decommissioning in RegenS during which *mDisk* data is maintained internally until the *diFS* system has safely re-distributed it.

4.4 Cost analysis

Datacenter SSD total cost of ownership (TCO) is composed of acquisition, surrounding hardware, and running costs over time, i.e., electricity, cooling and various maintenance costs. The exact composition of TCO for real-life datacenters is proprietary and varies for different server configurations. We use a similar methodology to how we estimated emissions overheads to estimate the TCO of an SSD server deployment with Salamander SSDs (S) relative to that of a deployment with baseline SSDs (B) as

$$TCO(S) = f_{opex} \cdot TCO(B) + (1 - f_{opex}) \cdot CRu_{S|B} \cdot TCO(B) \quad (4)$$

where f_{opex} is the fraction of operational costs. $CRu_{S|B}$ is the relative cost upgrade rate of SSDs in S, i.e., the cost of Salamander drives combined with that of newer, more cost-effective, baseline SSDs to compensate for the reduced capacity of Salamander drives during their L_1 phase.

We define $CRu_{S|B} = Ru_{S|B} + (1 - Ru_{S|B}) \cdot CE_{B_{new}} \cdot Cap(B_{new})$ as the cost effectiveness of SSDs in S relative to B, where $Ru_{S|B}$ is the previously defined upgrade rate, $Cap(B_{new})$ is the fraction of reduced capacity that requires new baseline SSDs, and $CE_{B_{new}}$ is the cost effectiveness of new baseline SSDs, i.e., \$/TB/year. Previously, we determined an average shrunk drive capacity of 60% of baseline capacity, i.e., $Cap(B_{new}) = 0.4$. An historical analysis of SSD \$/TB costs shows a ~4x improvement every five years [47], a conservative datacenter hardware replacement period [48]. Accordingly, we conservatively assume Salamander drives start shrinking (L_1) after five years and set $CE_{B_{new}} = 0.25$.

A recent analysis by Seagate [49] shows that "device acquisition cost is by far the dominant component" for datacenter devices with ~86% of TCO. We therefore set $f_{opex} = 0.14$ accordingly.

Overall, using Eq. 4, we conclude that Salamander achieves 13% and 25% cost savings for ShrinkS and RegenS accordingly. Even when we model operating costs as a higher percentage of the budget there is still a cost benefit for Salamander; for instance, if we assume half the cost is operational costs, Salamander lowers costs by 6–14%.

ACKNOWLEDGMENTS

This work was supported in part by NSF/BSF grant 2154771/2021736, NSF grant 2244937, and a gift from Broadcom. Porter has a significant financial interest in Fortanix, Inc. We also thank Ethan Miller, Nadav Amit, and Michael Factor for their insightful comments and assistance.

REFERENCES

[1] Lawrence Berkeley National Laboratory. 2024 united states data center energy usage report. https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report.pdf. Accessed: Jan 2025.

[2] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. ACT: Designing sustainable computer systems with an architectural carbon modeling tool. In *Annual International Symposium on Computer Architecture*, ISCA, 2022.

[3] Swamit Tannu and Prashant J Nair. The dirty secret of SSDs: Embodied carbon. In *Workshop on Sustainable Computer Systems Design and Implementation*, HotCarbon, 2022.

[4] Jialun Lyu, Jaylen Wang, Kali Frost, Chaojie Zhang, Celine Irvene, Esha Choukse, Rodrigo Fonseca, Ricardo Bianchini, Fiodar Kazhamiaka, and Daniel S Berger. Myths and misconceptions around reducing carbon embedded in cloud platforms. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, HotCarbon, 2023.

[5] Azure premium storage: Design for high performance. https://learn.microsoft.com/en-us/azure/virtual-machines/premium-storage-performance%20, 2024. Accessed: Jan 2025.

[6] What's the difference between an SSD and a hard drive? https://aws.amazon.com/compare/the-difference-between-ssd-hard-drive/. Accessed: Jan 2025.

[7] Sara McAllister, Fiodar Kazhamiaka, Daniel S Berger, Rodrigo Fonseca, Kali Frost, Aaron Ogus, Maneesh Sah, Ricardo Bianchini, George Amvrosiadis, Nathan Beckmann, et al. A call for research on storage emissions. In *Workshop on Sustainable Computer Systems*, HotCarbon, 2024.

[8] Sara McAllister, Benjamin Berg, Daniel S Berger, George Amvrosiadis, Nathan Beckmann, Gregory R Ganger, et al. FairyWREN: A sustainable cache for emerging {Write-Read-Erase} flash interfaces. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI, 2024.

[9] Aviad Zuck, Donald E. Porter, and Dan Tsafrir. Degrading data to save the planet. In *ACM Workshop on Hot Topics in Operating Systems*, HotOS, 2023.

[10] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing carbon: The elusive environmental footprint of computing. *IEEE Micro*, 42(4):37–47, 2022.

[11] Bryan S Kim, Jongmoo Choi, and Sang Lyul Min. Design tradeoffs for SSD reliability. In *USENIX Conference on File and Storage Technologies*, FAST, 2019.

[12] Alessia Marelli and Rino Micheloni. BCH and LDPC error correction codes for NAND flash memories. *3D Flash Memories*, pages 281–320, 2016.

[13] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Reducing solid-state drive read latency by optimizing read-retry. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2021.

[14] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. A study of {SSD} reliability in large scale enterprise storage deployments. In *USENIX Conference on File and Storage Technologies*, 2020.

[15] Jui-Nan Yen, Yao-Ching Hsieh, Cheng-Yu Chen, Tseng-Yi Chen, Chia-Lin Yang, Hsiang-Yun Cheng, and Yixin Luo. Efficient bad block management with cluster similarity. In *International Symposium on High-Performance Computer Architecture*, HPCA. IEEE, 2022.

[16] Ziyang Jiao, Xiangqun Zhang, Hojin Shin, Jongmoo Choi, and Bryan S Kim. The design and implementation of a fcapacity-variant storage system. In *USENIX Conference on File and Storage Technologies*, FAST, 2024.

[17] Tao Zhang, Aviad Zuck, Donald E Porter, and Dan Tsafrir. Apps can quickly destroy your mobile's flash: why they don't, and how to keep it that way. In *Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys, 2019.

[18] Ming-Chang Yang, Yuan-Hao Chang, Yuan-Hung Kuan, and Che-Wei Tsao. Graceful space degradation: An uneven space management for flash storage devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1425–1434, 2015.

[19] Chanha Kim, Chanik Park, Sungjoo Yoo, and Sunggu Lee. Extending lifetime of flash memory using strong error correction coding. *IEEE Transactions on Consumer Electronics*, 61(2):206–214, 2015.

[20] Shuhei Tanakamaru, Mayumi Fukuda, Kazuhide Higuchi, Atsushi Esumi, Mitsuyoshi Ito, Kai Li, and Ken Takeuchi. Post-manufacturing, 17-times acceptable raw bit error rate enhancement, dynamic codeword transition ECC scheme for highly reliable solid-state drives, SSDs. *Solid-State Electronics*, 58(1):2–10, 2011.

[21] Yuan-Hao Chang and Tei-Wei Kuo. A reliable MTD design for MLC flash-memory storage systems. In *ACM international conference on Embedded software*, EMSOFT, 2010.

[22] Shunzhuo Wang, Fei Wu, Zhonghai Lu, You Zhou, Qin Xiong, Meng Zhang, and Changsheng Xie. Lifetime adaptive ECC in NAND flash page management. In *Design, Automation & Test in Europe Conference & Exhibition*, DATE. IEEE, 2017.

[23] You Zhou, Fei Wu, Zhonghai Lu, Xubin He, Ping Huang, and Changsheng Xie. SCORE: A novel scheme to efficiently cache overlong ECCs in NAND flash memory. *ACM Transactions on Architecture and Code Optimization*, 15(4):1–25, 2018.

[24] ScaleFlux. https://scaleflux.com.

[25] Jaylen Wang, Daniel S Berger, Fiodar Kazhamiaka, Celine Irvene, Chaojie Zhang, Esha Choukse, Kali Frost, Rodrigo Fonseca, Brijesh Warrier, Chetan Bansal, et al. Designing cloud servers for lower carbon. In *Annual International Symposium on Computer Architecture*. IEEE, 2024.

[26] Yu Cai, Saugata Ghose, Erich F Haratsch, Yixin Luo, and Onur Mutlu. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE*, 105(9):1666–1704, 2017.

[27] The Tech Report. The SSD endurance experiment: They're all dead. http://techreport.com/review/27909/the-ssd-endurance-experiment-theyre-all-dead, 2015.

[28] Fan Xu, Shujie Han, Patrick PC Lee, Yi Liu, Cheng He, and Jiongzhou Liu. General feature selection for failure prediction in large-scale SSD deployment. In *International Conference on Dependable Systems and Networks*, DSN. IEEE, 2021.

[29] Wenwen Hao, Ben Niu, Yin Luo, Kangkang Liu, and Na Liu. Improving accuracy and adaptability of SSD failure prediction in hyper-scale data centers. *ACM SIGMETRICS Performance Evaluation Review*, 49(4):99–104, 2022.

[30] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. Proactive error prediction to improve storage system reliability. In *USENIX Annual Technical Conference*, USENIX ATC, 2017.

[31] Peng Li, Wei Dang, Congmin Lyu, Min Xie, Quanyang Bao, Xiaofeng Ji, and Jianhua Zhou. Reliability characterization and failure prediction of 3D TLC SSDs in large-scale storage systems. *IEEE Transactions on Device and Materials Reliability*, 21(2):224–235, 2021.

[32] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. Operational characteristics of SSDs in enterprise storage systems: A large-scale field study. In *USENIX Conference on File and Storage Technologies*, FAST, 2022.

[33] ServeTheHome. We bought 1347 used data center SSDs to look at SSD endurance. https://www.servethehome.com/we-bought-1347-used-data-center-ssds-to-look-at-ssd-endurance-solidigm/, 2022. Accessed: Nov. 2024.

[34] Jacob Alter, Ji Xue, Alma Dimnaku, and Evgenia Smirni. SSD failures in the field: symptoms, causes, and prediction models. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[35] Shujie Han, Patrick PC Lee, Fan Xu, Yi Liu, Cheng He, and Jiongzhou Liu. An in-depth study of correlated failures in production SSD-based data centers. In *USENIX Conference on File and Storage Technologies*, FAST, 2021.

[36] Lieven Eeckhout. Focal: A first-order carbon model to assess processor sustainability. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2024.

[37] Ren-Shuo Liu, Chia-Lin Yang, Cheng-Hsuan Li, and Geng-You Chen. Duracache: A durable SSD cache using MLC NAND flash. In *Annual Design Automation Conference*, DAC, 2013.

[38] Jaeho Kim, Eunjae Lee, Jongmoo Choi, Donghee Lee, and Sam H Noh. Chip-level RAID with flexible stripe size and parity placement for enhanced SSD reliability. *IEEE Transactions on Computers*, 65(4):1116–1130, 2014.

[39] Ellis H Wilson, Myoungsoo Jung, and Mahmut T Kandemir. ZombieNAND: Resurrecting dead nand flash for improved SSD longevity. In *International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, MASCOTS. IEEE, 2014.

[40] Xavier Jimenez, David Novo, and Paolo Ienne. Phoenix: Reviving MLC blocks as SLC to extend NAND flash devices lifetime. In *Design, Automation & Test in Europe Conference & Exhibition*, DATE. IEEE, 2013.

[41] Youngseop Shim, Myungsuk Kim, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim. Exploiting process similarity of 3D flash memory for high performance SSDs. In *Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO, 2019.

[42] Md Raquibuzzaman, Md Mehedi Hasan, Aleksandar Milenkovic, and Biswajit Ray. Layer-to-layer endurance variation of 3D NAND flash memory. In *IEEE International Reliability Physics Symposium*, IRPS. IEEE, 2022.

[43] Jennifer Switzer, Gabriel Marcano, Ryan Kastner, and Pat Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS, 2023.

[44] Meng Zhang, Fei Wu, Qin Yu, Weihua Liu, Lanlan Cui, Yahui Zhao, and Changsheng Xie. BeLDPC: Bit errors aware adaptive rate LDPC codes for 3D TLC NAND flash memory. In *2020 Design, Automation & Test in Europe Conference & Exhibition*, DATE. IEEE, 2020.

[45] Yunpeng Song, Yina Lv, and Liang Shi. DECC: Differential ECC for read performance optimization on high-density NAND flash memory. In *Asia and South Pacific Design Automation Conference*, ASPDAC, 2023.

[46] Myoungjun Chun, Myungsuk Kim, Dusol Lee, Jisung Park, and Jihong Kim. Readguard: Integrated SSD management for priority-aware read performance differentiation. *ACM Transactions on Storage*, 20(4):1–39, 2024.

[47] Our World in Data. Historical price of computer memory and storage. https://ourworldindata.org/grapher/historical-cost-of-computer-memory-and-storage, 2024. Accessed: April 2025.

[48] Supermicro. Green computing: Top ten best practices for a green data center. https://www.supermicro.com/white_paper/eGuide_Data_Center_Refresh.pdf, 2022. Accessed: April 2025.

[49] Seagate. Three truths about hard drives and ssds. https://www.seagate.com/blog/three-truths-about-hard-drives-and-ssds/, 2024. Accessed: April 2025.