Design Tradeoffs for SSD Reliability

Bryan S. Kim, ***Seoul National University;*** **Jongmoo Choi,** ***Dankook University;*** **Sang Lyul Min,** ***Seoul National University***

https://www.usenix.org/conference/fast19/presentation/kim-bryan

This paper is included in the Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19).

February 25–28, 2019 • Boston, MA, USA

978-1-939133-09-0

Open access to the Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19) is sponsored by

NetApp®

Design Tradeoffs for SSD Reliability

Bryan S. Kim
Seoul National University

Jongmoo Choi
Dankook University

Sang Lyul Min
Seoul National University

Abstract

Flash memory-based SSDs are popular across a wide range of data storage markets, while the underlying storage medium—flash memory—is becoming increasingly unreliable. As a result, modern SSDs employ a number of in-device reliability enhancement techniques, but none of them offers a *one size fits all* solution when considering the multi-dimensional requirements for SSDs: performance, reliability, and lifetime.

In this paper, we examine the design tradeoffs of existing reliability enhancement techniques such as data re-read, intra-SSD redundancy, and data scrubbing. We observe that an uncoordinated use of these techniques adversely affects the performance of the SSD, and careful management of the techniques is necessary for a graceful performance degradation while maintaining a high reliability standard. To that end, we propose a holistic reliability management scheme that selectively employs redundancy, conditionally re-reads, judiciously selects data to scrub. We demonstrate the effectiveness of our scheme by evaluating it across a set of I/O workloads and SSDs wear states.

1 Introduction

From small mobile devices to large-scale storage servers, flash memory-based SSDs have become a mainstream storage device thanks to flash memory's small size, energy efficiency, low latency, and collectively massive parallelism. The popularity of SSDs is fueled by the continued drop in cost per GB, which in turn is achieved by storing multiple bits in a memory cell [7, 42] and vertically stacking memory layers [41, 48].

However, the drive for high storage density has caused the flash memory to become less reliable and more error-prone [9, 20]. Raw bit error rate measurement of a single-level cell flash memory in 2009 was in the order of 10^{-8} [19], but this increased to 10^{-7}–10^{-4} in 2011 for a 3x-nm multi-level cell [52] and to 10^{-3}–10^{-2} in 2017 for a 1x-nm memory [10]. The high error rates in today's flash memory are caused by various reasons, from wear-and-tear [11, 19, 27], to gradual charge leakage [11, 14, 51] and data disturbance [13, 19].

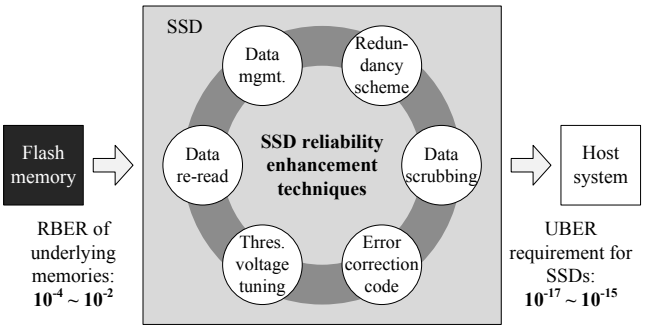


Figure 1: SSD reliability enhancement techniques.

In order to mask out the error-prone nature of flash memory, the state-of-the-art SSDs employ a number of in-device reliability enhancement techniques, as shown in Figure 1. These techniques originate from a wide range of domains, from device physics that tunes threshold voltage levels for sensing memory states [12, 14, 38], coding theory that corrects errors using computed parity information [18, 35], to system-level approaches such as scrubbing that preventively relocates data [22, 37]. This variety is caused by the fact that there is no *one size fits all* solution for data protection and recovery: each technique has a multi-dimensional design tradeoff that makes it necessary to compositionally combine complementary solutions. This is much easier said than done as reliability is only one of the many design goals for SSDs: a study spanning across multiple institutions reveals that these reliability enhancements, in fact, cause performance degradation in SSDs [21].

In this paper, we examine the design tradeoffs of existing techniques across multiple dimensions such as average and tail performance, write amplification, and reliability. Our investigation is inspired by studies in the HDD domain that evaluate the effectiveness of different reliability enhance-

ments [24, 40, 49], but our findings deviate from those work due to the difference in the underlying technology and the SSD's internal management. We make the following three observations from our experiments. First, the use of data re-read mechanisms should be managed, as the repeated re-reads further induce errors, especially for read disturbance-vulnerable cells. Second, in the absence of random and sporadic errors, the overheads of intra-SSD redundancy outweigh its benefits in terms of performance, write amplification, and reliability. Lastly, SSD-internal scrubbing reduces the error-induced long-tail latencies, but it increases the internal traffic that negates its benefits.

Based on our observation, we propose a holistic reliability management scheme that selectively employs intra-SSD redundancy depending on access characteristics of the data, conditionally uses data re-read mechanism to reduce the effects of read disturbance, and judiciously selects data to scrub so that the internal relocation traffic is managed. Redundancy is applied only to infrequently accessed *cold* data to reduce write amplification, and frequently read *read-hot* data are selected for scrubbing based on a cost-benefit analysis (overhead of internal traffic vs. reduction in re-reads). In this paper, we present the following:

- We construct and describe an SSD architecture that holistically incorporates complementary reliability enhancement techniques used in modern SSDs. (§ 3)
- We evaluate the state-of-the-art solutions across a wide range of SSD states based on a number of flash memory error models, and discuss their tradeoffs in terms of performance, reliability, and lifetime. (§ 4)
- We propose a holistic reliability management scheme that self-manages the use of multiple error-handling techniques, and we demonstrate its effectiveness across a set of real I/O workloads. (§ 5)

2 Background

In this section, we describe the causes of flash memory errors and their modeling, and the existing reliability enhancement techniques that correct and prevent errors. For more detailed and in-depth explanations, please refer to Mielke *et al.* [43] for error mechanisms and modeling, and Cai *et al.* [9] for error correction and prevention techniques.

2.1 Errors in Flash Memory

We focus on three major sources of flash memory errors: wear, retention loss, and disturbance.

Wear. Repeated programs and erases (also known as P/E cycling) wear out the flash memory cells that store electrons (data), and cause irreversible damage to them [9, 11, 19]. Flash memory manufacturers thus specify an *endurance limit*, a number of P/E cycles a flash memory block can withstand, and this limit has been steadily decreasing for every new generation of flash memory. However, the endurance limit is *not* a hard limit: not all blocks are created equally due to process variations, and a number of studies dynamically measure the lifetime of a block to extend its usage [27].

Retention loss. Electrons stored in flash memory cells gradually leak over time, making it difficult to correctly read the data stored, and errors caused by retention loss increase as cells wear [9, 14, 43]. While a number of studies indicate that retention loss is a dominant source of errors [11, 14], retention errors are fortunately transient: they reset once the block is erased [43].

Disturbance. Reading a wordline in a block *weakly* programs other wordlines in the block, unintentionally inserting more electrons into their memory cells [9, 19, 43]. Disturbance and retention errors are opposing error mechanisms, but they do not necessarily cancel each other out: disturbance mainly affects cells with fewer electrons (erased state), but charge leakage affects those with more (programmed state) [9, 43]. Similar to retention loss, errors caused by read disturbances increase as cells wear and reset once the block is erased [43].

These three sources of errors are used to model the raw bit error rate (RBER) of flash memory with the following additive power-law variant [36, 43]:

$$\begin{aligned} &RBER(cycles, time, reads) && (1)\\ &= \varepsilon + \alpha \cdot cycles^{k} && \text{(wear)}\\ &\quad + \beta \cdot cycles^{m} \cdot time^{n} && \text{(retention)}\\ &\quad + \gamma \cdot cycles^{p} \cdot reads^{q} && \text{(disturbance)} \end{aligned}$$

where ε, α, β, and γ are coefficients and k, m, n, p, and q are exponents particular to a flash memory. These *nine* parameters define the RBER of a flash memory chip, and Mielke *et al.* [43] and Liu *et al.* [36] further explain the validity for the additive power-law model in detail.

2.2 SSD Reliability Enhancement Techniques

Table 1 outlines the tradeoffs for the commonly used reliability enhancement techniques.

Error correction code (hard-decision). Hard-decision ECC such as BCH (code developed by Bose, Ray-Chaudhuri, and Hocquenghem) [35] is the first line of defense against flash memory errors. When writing, the ECC encoder computes additional parity information based on the data, which is typically stored together in the same page. Flash memory manufacturers conveniently provide additional spare bytes within a page for this purpose. When reading, the hard-decision ECC decoder returns the error-corrected data or reports a failure after a fixed number of cycles. With the continued decline in flash reliability, it is

Table 1: Comparison of SSD reliability enhancement techniques.

Techniques	Impact on average performance	Impact on tail performance	Write amplification	Management overhead	Related work
ECC (hard-decision)	Negligible	None	Negligible	None	BCH, LDPC [35]
ECC (soft-decision)	None	High	Negligible	Negligible	LDPC [18, 35]
Threshold voltage tuning	None	High	None	Voltage levels	Read retry [12] Voltage prediction [14, 38]
Intra-SSD redundancy	High for small stripes; low for large stripes	Low for small stripes; high for large stripes	High	Stripe group information	Dynamic striping [31, 33] Intra-block striping [46] Parity reduction [25, 34]
Background data scrubbing	Depends	Depends	Depends	Block metadata such as erase count or read count	Read reclaim [22] Read refresh [37]

becoming increasingly inefficient to rely solely on stronger ECC engines [15].

Error correction code (soft-decision). Soft-decision ECC such as LDPC (low-density parity-check) [18, 35] also encodes additional parity, but uses soft-information—the probability of each bit being 1 or 0—for decoding. This requires the data in flash memory to be read multiple times. The error correction strength of soft-decision decoding is orders of magnitude greater than its hard-decision counterpart, but this is achieved at an expense of multiple flash memory reads.

Threshold voltage tuning. The electrons stored in flash memory cells gradually shift over time due to charge leakage [11, 14] and disturbance [13, 19]. To counteract this drift, threshold voltages for detecting the charge levels are adjustable through special flash memory commands [12]. In SSD designs without soft-decision ECC, data are re-read after tuning the threshold voltages if the hard-decision ECC fails [9]. Although the underlying mechanisms are different, both soft-decision ECC and threshold voltage tuning share the same high-level design tradeoff: the greater the probability of correcting errors with repeated reads.

Intra-SSD redundancy. SSDs can internally add redundancy across multiple flash memory chips [31, 33, 46], similar to how RAID [47] protects data by adding redundancy across multiple physical storage devices. While both ECC and RAID-like redundancy enhance the SSD's reliability by adding extra parity information, striping data across multiple chips protects the SSD against chip and wordline failures that effectively renders the traditional ECC useless. In general, increasing the stripe size trades the overhead of parity writes for the penalty of reconstructing data. In the context of SSDs, employing redundancy amplifies the write traffic not only because of parity writes, but also because the effective over-provisioning factor is decreased.

Data scrubbing. We use this as an umbrella term in this paper for the variety of SSD's housekeeping tasks that enhance reliability. This includes read reclaim [22] that addresses read disturbance-induced errors, and read refresh [37] that handles retention errors. While ECC and voltage tuning *correct* errors, these tasks *prevent* errors: by monitoring SSD-internal information, data are preventively relocated before errors accumulate to a level beyond error correction capabilities. In effect, background data scrubbing reduces the overhead of error correction by creating additional internal traffic, but this traffic also affects the QoS performance and accelerates wear.

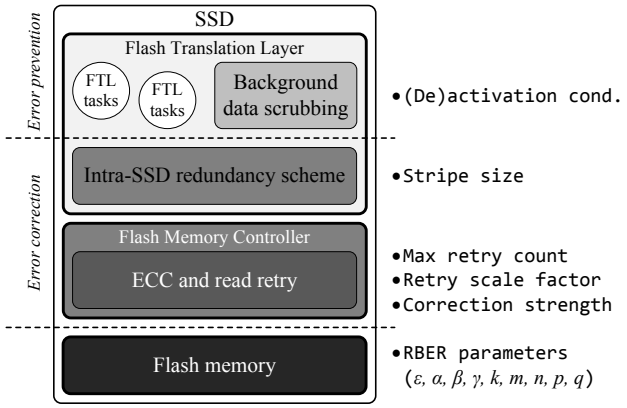


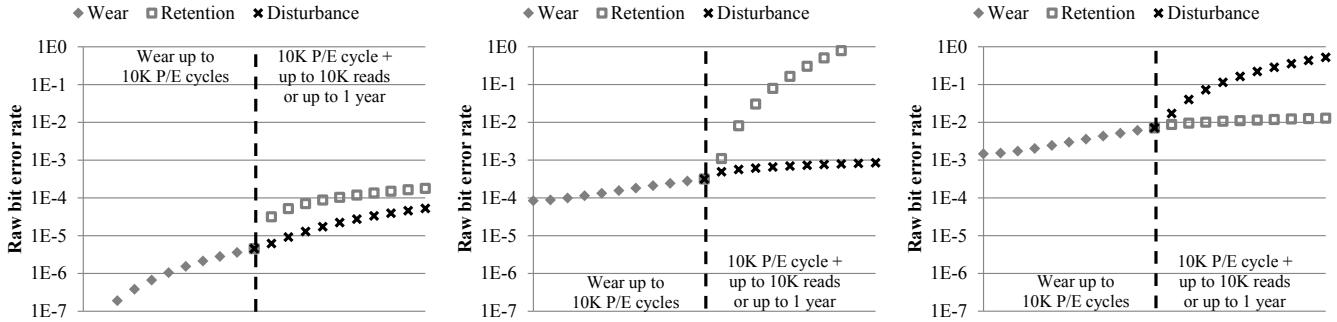
Figure 2: The overall error-handling architecture of an SSD, and its associated configuration parameters.

3 Design Tradeoffs for SSD Reliability

To understand how data protection and recovery schemes in modern SSDs ensure data integrity in the midst of flash memory errors, we construct an SSD model that holistically considers the existing reliability enhancement techniques. Figure 2 illustrates this SSD model with particular emphasis on error-related components and their configurable parameters.

Table 2: RBER model parameters. Parameters ε, α, β, γ, k, m, n, p, and q describe the RBER model in Equation 1. R^2 represents the goodness of fit and is computed using the log values of the data and model, and N is the sample size.

Flash memory	Year	ε	α	β	γ	k	m	n	p	q	R^2	N
3x-nm MLC [52]	2011	5.06E-08	1.05E-14	9.31E-14	4.17E-15	2.16	1.80	0.80	1.07	1.45	0.984	98
2y-nm MLC [13, 14]	2015	8.34E-05	3.30E-11	5.56E-19	6.26E-13	1.71	2.49	3.33	1.76	0.47	0.988	173
72-layer TLC	2018	1.48E-03	3.90E-10	6.28E-05	3.73E-09	2.05	0.14	0.54	0.33	1.71	0.969	54



(a) 3x-nm MLC. (b) 2y-nm MLC. (c) 72-layer TLC.

Figure 3: Projected RBER graphs based on model parameters in Table 2. Each graph shows the error rate caused by the three mechanisms: wear, retention loss, and disturbance. In the first half of the x-axis, RBER increases due to repeated programs and erases (up to 10K cycles). In the second half, the cells are kept at 10K P/E cycle, but the data are repeatedly read (up to 10K reads) to induce disturbance errors or are left unaccessed (up to 1 year) for retention errors.

3.1 Error-Prone Flash Memory

Flash memory is becoming increasingly unreliable in favor of high-density [20], and we observe this trend across error datasets we analyzed. Table 2 shows the *nine* `RBER parameters` (see Equation 1 in § 2.1) for three different flash memory chips: 3x-nm MLC, 2y-nm MLC, and 72-layer TLC. We curve-fit the parameters of the three chips through simulated annealing. The datasets for the 3x-nm MLC [52] and 2y-nm MLC [13, 14][1] are extracted from the figures in the publications using plot digitization [3], while the dataset for the 72-layer TLC is provided by a flash memory manufacturer.

Figure 3 illustrates the contributing factors of errors for these chips. The graphs are generated based on the RBER model parameters in Table 2, and show that the overall error rate increases with the newer and denser flash memories. The most interesting observation, however, is the dominance of disturbance errors in the 72-layer TLC. This is in stark contrast with the 2y-nm MLC whose dominant error is due to retention loss.

In this work, we neither argue the importance of one error type over the other, nor claim a shifting trend in dominant errors. In fact, the sample space and sample size of the three datasets for the flash memory chips are different, making it difficult to compare equally. For example, we do not claim that the 2y-nm MLC in Figure 3b will have a 100% error rate after 1 year: the projected retention loss error is computed based on a limited number of RBER data samples that cover a smaller subset of the sample space. Rather, the graphical representation of Figure 3 is only used to illustrate the wide variation in error characteristics, and that an error-handling technique tailored for one particular memory chip may fail to meet the reliability requirements in others.

3.2 Mechanism in Flash Memory Controller

The flash memory controller not only abstracts the operational details of flash memory, but also handles common-case error correction. In addition to hard-decision ECC, soft-decision ECC and threshold voltage tuning are implemented in the controller as their mechanisms simply iterate through a pre-defined set of threshold voltage levels for successive reads (although setting appropriate voltage levels may involve the firmware).

The hard-decision ECC tolerates up to n-bit errors, defined by the `correction strength`. Increasing the ECC correction strength not only increases the logic complexity and power consumption, but also inflates the amount of parity data that needs to be stored in flash. The fixed number of bytes per page (including the spare) is thus the limiting factor for the ECC's capability. Errors beyond the hard-decision ECC's correction strength are subsequently handled by the flash memory controller with data re-reads. In these cases, the same data are accessed again, repeatedly if needed. If the data cannot be recovered after `max retry count`, the firmware is notified of a read error. We model threshold voltage tuning and soft-decision decoding in a way that each successive reads effectively reduces the RBER of the data by `retry scale factor`. This model is general enough to

cover both mechanisms, and they will be referred to as *data re-reads* henceforth.

3.3 Role of Flash Translation Layer

The flash translation layer (FTL) consists of a number of SSD-internal housekeeping tasks that collectively hide the quirks of flash memory and provide an illusion of a traditional block device. Mapping table management and garbage collection are the widely known FTL tasks, but these will not be discussed in detail. In the context of reliability, we focus on intra-SSD redundancy and data scrubbing.

Intra-SSD redundancy is used to reconstruct data when ECC (both hard-decision and data re-read) fails. In this work, we focus on constructing redundancy based on the physical addresses. Upon writes, *one* parity is added for every *s* data writes, defined by the `stripe size`. The *s* data and *one* parity create a stripe group, and the parity is distributed among the stripe group, akin to the workings of RAID 5. If the flash memory controller reports an uncorrectable error, the firmware handles the recovery mechanism by identifying the stripe group that dynamically formed when the data were written [31, 32]. If any of the other data in the same stripe group also fails in the ECC layer, the SSD reports an uncorrectable error to the host system. If the data are not protected by redundancy (`stripe size` of ∞), any ECC failure causes an uncorrectable error.

While the techniques discussed so far correct errors, data scrubbing prevent errors from accumulating by relocating them in the background. The scrubber `activates` and `deactivates` under certain conditions, which depends on the implementation: it can trigger periodically and scan the entire address space [6, 45], it can activate once the number of reads per block exceeds a threshold [22, 30], or it can relocate data based on the expected retention expiration [37].

These firmware-oriented reliability enhancements pay a cost in the *present* to reduce the penalty in the *future*. For intra-SSD redundancy, the increased frequency of writing parity data reduces the number of reads to reconstruct the data. For data scrubbing, proactive relocation of data prevents the ECC in the controller from failing. At the same time, both techniques increase the write amplification and accelerate wear, not only reducing the lifetime of the SSD, but also effectively increasing the chance of future errors. This cyclical dependence makes it difficult to quantify the exact benefits and overheads of these techniques.

4 Evaluation of SSD Reliability

We implement the flash memory error model and the reliability enhancements on top of the DiskSim environment [1] by extending its SSD extension [5]. We construct three SSDs, each with three different initial wear states: The 3x-nm MLC blocks are initialized with 10K, 30K, and 50K P/E cycles on average, the 2y-nm MLC blocks to 2K, 5K, and 10K P/E cycles, and the 72-layer TLC to 1K, 3K, and 5K. These *nine* SSD states have different error rates, but are otherwise identical in configuration. Realistically, these SSDs should have different capacities (page size, number of pages per block, number of blocks, etc.) and even operation latencies, but we use the same internal organization to isolate the effects of reliability enhancement techniques on the overall performance. Table 3 summarizes the SSD's internal configuration.

Table 3: System configuration.

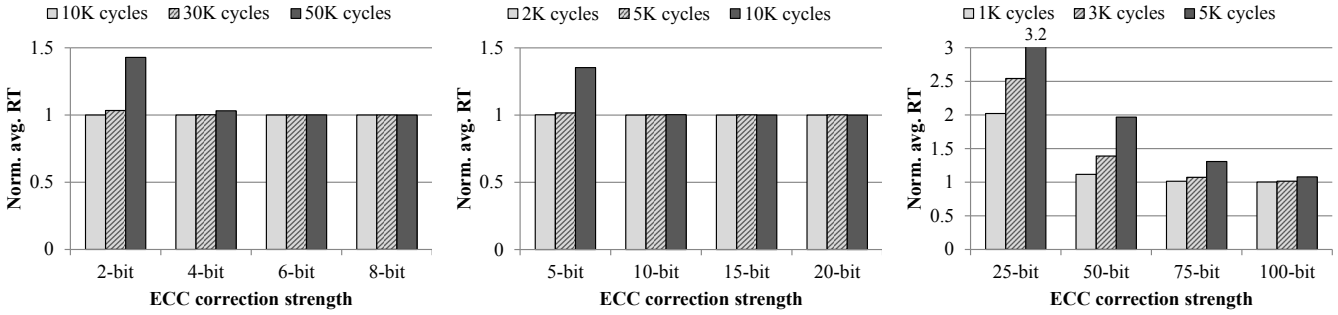
Parameter	Value	Parameter	Value
# of channels	8	Read latency	50μs
# of chips/channel	4	Program latency	500μs
# of planes/chip	2	Erase latency	5ms
# of blocks/plane	1024	Data transfer rate	667MB/s
# of pages/block	256	Physical capacity	256GiB
Page size	16KiB	Logical capacity	200GiB

We extend our prior work [30] that includes all essential FTL functionalities such as host request handling and garbage collection (GC) and implement the discussed reliability enhancement schemes. Host requests are handled in a non-blocking manner to fully utilize the underlying parallelism, and all FTL tasks run independently and concurrently. Flash memory requests from these tasks are generated with some delay to model the think time, and up to *eight* tasks can be active concurrently to model the limited number of embedded processors in SSDs. Host addresses are translated in 4KiB mapping granularity, and the entire map is resident in DRAM. Host data, GC data, and data from the scrubber are written to different sets of blocks to separate hot and cold data, and they are arbitrated by a prioritized scheduler: host requests have the highest priority, then garbage collection, and lastly scrubbing.

For evaluation, we use synthetic workloads of 4KiB read/write with a 70:30 mixture, and the access pattern has a skewed distribution to mimic the SSD endurance workload specification [2]. I/O requests arrive at the SSD every 0.5ms on average (2K IOPS): the I/O intensity is intentionally set so that the garbage collector's impact on the overall performance is properly reflected. The workload runs for an hour, executing 7.2 million I/Os in total. Prior to each experiment, data are randomly written to the entire physical space to emulate a pre-conditioned state.

4.1 Error Correction Code

We first investigate how the SSDs perform when relying solely on the flash memory controller. In this scenario, background scrubbing is disabled, and no intra-SSD redundancy is used. We test a number of `correction strength`, including ∞ that corrects all errors. Realistically, stronger ECC engines require larger ECC parity, but we assume that the



(a) 3x-nm MLC. (b) 2y-nm MLC. (c) 72-layer TLC.

Figure 4: Average read response time for the three SSDs at various wear states. For each graph, the *x*-axis shows the correction strength for the ECC, and the performance is normalized to that with ∞ error correction strength. The response time increases not only when the SSD is more worn out, but also when weaker ECC is used.

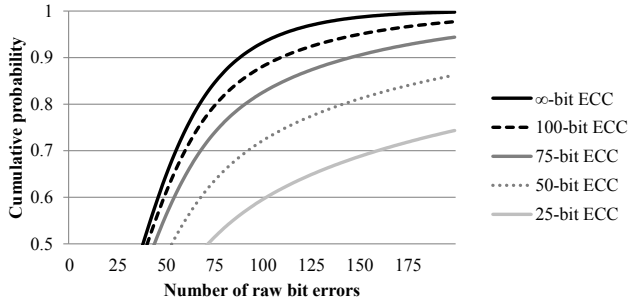


Figure 5: CDF of raw bit errors with varying ECC correction strength for the 72-layer TLC SSD of 5K initial wear state. With a ∞-bit ECC, 85% of data have less than 75-bit errors, but at 25-bit ECC, only 51% of data are. With more data being re-read, read disturbance increases the bit error rate.

flash memory page always has sufficient spare bytes. When ECC fails, data is re-read after adjusting the threshold voltage. Each data re-read reduces the RBER by 50% (`retry scale factor` of *2*), repeating until the error is corrected (`max retry count` of ∞).

Figure 4 shows the average response time for read requests for the three types of SSDs at various wear states. We chose the ECC correction strengths based on the relative RBER of the three flash memories. We observe that the performance degrades not only when the SSD is more worn out but also with weaker ECC correction strength. In the higher SSD wear states, errors are more frequent, and weaker ECC induces more data re-reads.

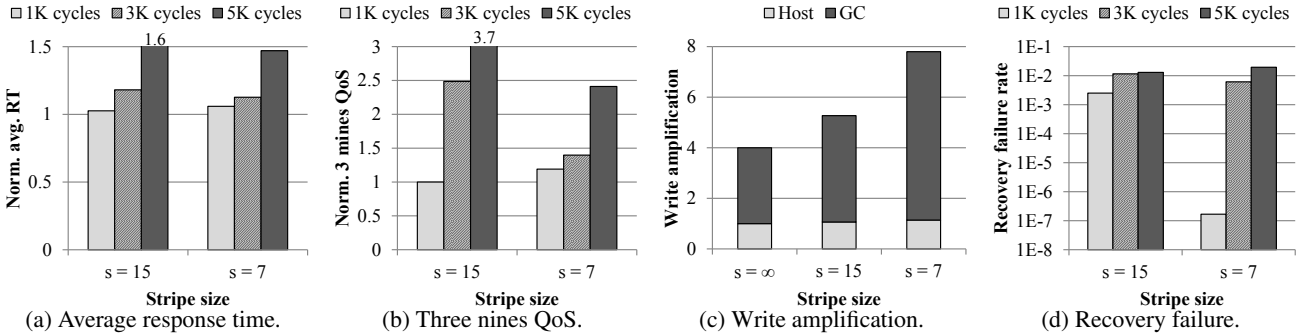
Compared to the 3x-nm (Figure 4a) and 2y-nm MLC (Figure 4b), the 72-layer TLC in Figure 4c shows a greater performance degradation. This is not only because of the higher overall RBER, but also because of relatively higher vulnerability to read disturbances (cf. Figure 3c). Figure 5 illustrates this case: it shows the cumulative distribution of measured raw bit errors with different ECC strengths for the 72-layer TLC at 5K P/E cycle wear state. When no data re-reads occur (∞-bit ECC), 85% of the ECC checks have less than 75-bit errors. With weaker ECC, however, the subsequent data re-reads induce additional read disturbance, lowering the CDF curve. Only 74% of the data have less than 75-bit errors when using a 75-bit ECC, but this further drops to 51% with a 25-bit ECC correction strength.

Thus, for read disturbance-sensitive memories, avoiding frequent data re-reads is critical for improving the performance. However, as illustrated in the upper portion of Figure 5, increasing the ECC strength has diminishing returns. This necessitates the use of ECC-complementary schemes that read pages in other blocks to reconstruct data (intra-SSD redundancy) or reduce the probability of data re-reads through preventive data relocations (data scrubbing).

4.2 Intra-SSD Redundancy

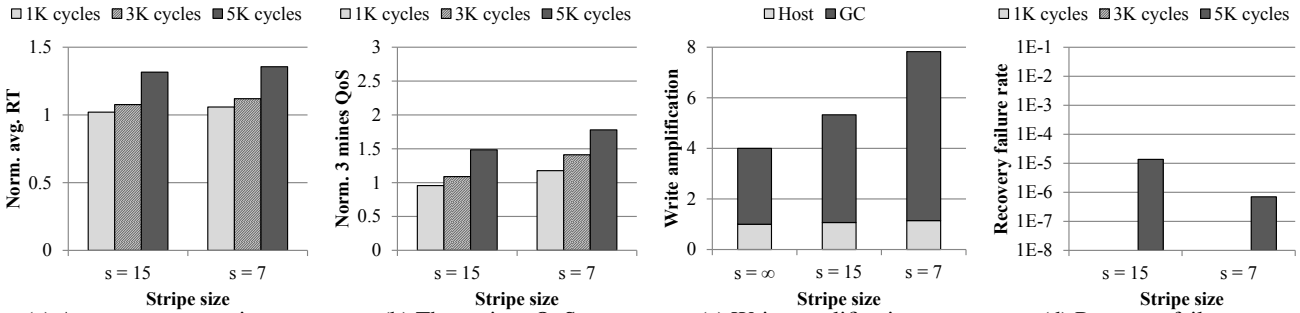
If data cannot be corrected after a given number of data re-read attempts, the data are reconstructed using other data within the stripe group. In this experiment, we examine the performance, reliability, and write amplification aspect of intra-SSD redundancy. We present the results from the 72-layer TLC using 75-bit ECC.

Figure 6 shows the results when `max retry count` is *one*: the flash memory controller attempts a data re-read scheme once before notifying the firmware. As shown in Figure 6a and Figure 6b, attempting frequent data reconstruction degrades performance especially in terms of long-tail latency because of increased internal traffic. The degradation is more severe for higher wear states (more errors), and greater stripe size (more pages accessed). This performance penalty is in addition to the increase in write amplification illustrated in Figure 6c. Even though host data programs are amplified due to parity writes, GC data programs are amplified at a greater rate because of the reduced effective capacity: without redundancy, the effective over-provisioning factor is 28%, but this drops to 20% when s=15, and to 12% when s=7. Furthermore, using intra-SSD redundancy does not guarantee full data recovery: accessing other pages in the stripe group can be uncorrectable through



(a) Average response time. (b) Three nines QoS. (c) Write amplification. (d) Recovery failure.

Figure 6: Performance, write amplification, and reliability for the 72-layer TLC SSD when `max retry count` is *one*. The performances in Figure 6a and Figure 6b are normalized to a system with ∞ correction strength. Using intra-SSD redundancy increases write amplification (Figure 6c), but moreover does not warrant full data recovery (Figure 6d).



(a) Average response time. (b) Three nines QoS. (c) Write amplification. (d) Recovery failure.

Figure 7: Performance, write amplification, and reliability for the 72-layer TLC SSD when `max retry count` is *three*. The performance degradation is not as severe as shown in Figure 6, but the write amplification (Figure 7c) remains similar. Reliability improves, but not all data can be reconstructed fully in the 5K wear state (Figure 7d).

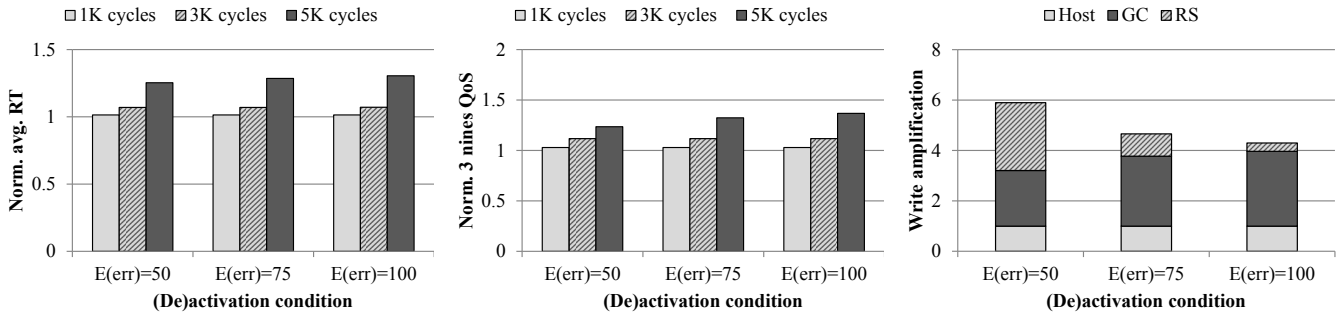
ECC, causing data recovery to fail. Given identical error rates, s=15 should have a higher failure rate as only one error is tolerated within a stripe group. While this is observed in Figure 6d in the lower 1K wear state, s=7 exhibits failure rates as high as those for s=15 in the higher wear states due to higher RBER.

Setting the `max retry count` to *one* reveals more weakness of intra-SSD redundancy than its strength. In Figure 7, we increase this parameter to *three* and observe the differences. In this setting, the average performance (Figure 7a) show a negligible difference to the scheme relying solely on ECC (cf. Figure 4c, 75-bit ECC), and only the 3 nines QoS (Figure 7b) show a more pronounced difference between s=15 and s=7. In this scenario, the performance changes are due to the increase in traffic for writing parity data, and, as expected, the write amplification measurements in Figure 7c are similar to that of Figure 6c. Most data recovery attempts succeed, but still does not warrant full data reconstruction (Figure 7d): While data are fully recovered in the lower wear states, recovery failures are observed in the higher 5K wear state. Further increasing the `max retry count` suppresses the use of data reconstruction through redundancy. In such cases, the benefits of using redundancy scheme are eliminated while the penalty of accelerated wear and increased write amplification remain.

Unlike the proven-effectiveness in the HDD environment, redundancy in SSDs falls short in our experiments. Compared to the scheme that relies on data re-read mechanisms in § 4.1, it performs no better, accelerates wear through increased write amplification, and, what's worse, may not fully recover data due to correlated failures. We expect correlated failures in SSDs to be more prevalent than in HDDs because of flash memory's history-dependence: the error rate in flash memory is a function of its prior history of operations such as the number of erases, number of reads, and time since its last program, and these values are likely to be similar across blocks within a stripe group. With that said, however, the data re-read mechanism is modeled optimistically in our setting, and in the event of a complete chip or wordline failures, SSDs have no other way to recover data aside from device-internal redundancy.

4.3 Background Scrubbing

We perform a set of experiments that measure the effectiveness of data scrubbing, and for this purpose, we assume an *oracle* data scrubber that knows the expected number of errors[2] for each data. This is possible in simulation (though not feasible in practice) as all the error-related parameters for each physical location in the SSD can be tracked to com-



(a) Average response time. (b) Three nines QoS. (c) Write amplification at the 5K wear state.

Figure 8: Performance and write amplification for the 72-layer TLC SSD using oracle scrubbing. The performances are normalized to an SSD with ∞ ECC strength. The oracle scrubber's (de)activation condition uses the expected number of errors per block. The ECC engine corrects up to 75-bit errors, so the E(err)=50 represents an aggressive scrubber.

pute the RBER at any given point in time. The all-knowing scrubber `activates` once the expected number of errors for any data exceeds a threshold, relocating that data to another location, and `deactivates` when the expected number for all data drops below that threshold. We use the oracle scrubber to illustrate the upper-bound benefits. Similar to § 4.2, we show the results from the 72-layer TLC using 75-bit ECC.

Figure 8 illustrates the average response time, 3 nines QoS, and write amplification of the oracle scrubber with three different trigger conditions. E(err)=50 relocates data most aggressively, while E(err)=100 does so lazily. There is little performance loss for the lower wear states, but for the 5K wear state, the difference between the aggressive and the lazy scrubber can be observed in the 3 nines QoS (Figure 8b). By proactively relocating data, the scrubber avoids the long-tail latencies caused by data re-reads. However, this comes at an increase in write amplification in the high wear states, as illustrated in Figure 8c. This shows the relative amount of write amplification per source, including that caused by the read scrubber. The aggressive scrubber in (E(err)=50) moves more data than garbage collection, resulting in a much higher write amplification; this increases the SSD's internal traffic, adding back some of the long-tail latency it reduced. The lazy counterpart, on the other hand, minimally relocates data.

Scrubbing is not a panacea, but it is more suitable than intra-SSD redundancy for complementing the underlying ECC. The scrubber's performance overhead is less than the redundancy scheme, and the increase in write amplification only occurs towards the end-of-life phase. There are several factors that contribute to our results. First, the out-of-place update for flash amplifies the overhead of garbage collection when using intra-SSD redundancy. Second, the history-dependent error patterns of flash memory work against redundancy because of correlated failures, but they make preventive mechanisms more effective because of error predictability.

4.4 Retention Test

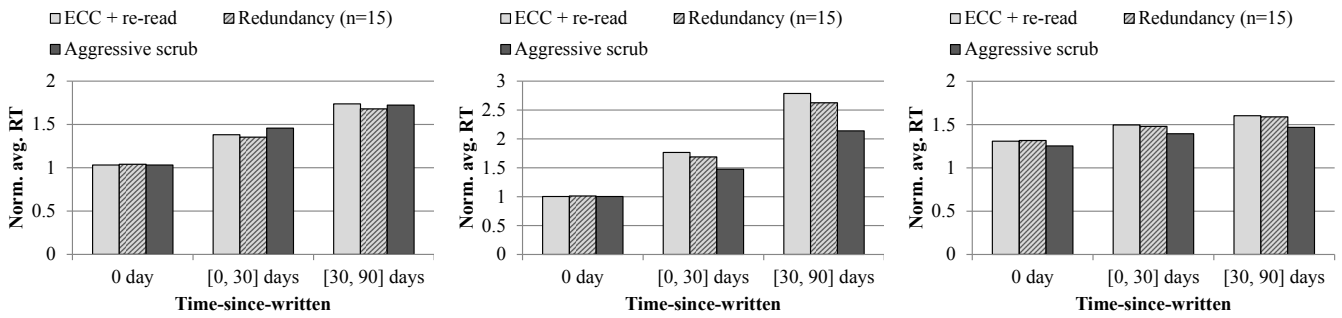
While the experiments so far considered a range of wear states (erase count) and the dynamicity of internal data accesses (read count), the 1 hour experiment is too short to exercise scenarios where data are lost due to retention errors: that is, all the pre-conditioned data are assumed to be written just prior to starting each workload. In this subsection, we explore the effects of data loss due to charge leakage by initializing a non-zero time-since-written value for each data.

Figure 9 shows how the representative error-handling approaches (ECC+re-read of § 4.1, s=15 redundancy of § 4.2, and aggressive scrub of § 4.3) perform when emulating non-zero time-since-written values. SSDs are all at the end-of-life state (50K cycles for the 3x-nm MLC, 10K cycles for the 2y-nm MLC, and 5K cycles for the 72-layer TLC), and they have an ECC correction strength of 4-bits for the 3x-nm, 10-bits for the 2y-nm, and 75-bits for the 72-layer (cf. Figure 4). All performances are normalized to that with an SSD with ∞-bit ECC. We observe that the performance difference between the background scrub approach and others becomes more noticeable. The scrubber proactively relocates data to fresh blocks to prevent upcoming reads from experiencing long-tail latencies. This is particularly more effectively for the 2y-nm MLC that exhibits vulnerability to retention errors (cf. Figure 3b). Compared to the scheme that relies on data re-reads, the aggressive scrubber reduces the performance degradation by 23% for the [30,90] days setting.

4.5 Discussion

We briefly summarize our findings:

- In the high wear states, data re-reads (§ 4.1) severely degrade the performance, increasing the average response time by up to 3.2× when a weak ECC engine is used. Each data re-read further increases the bit error rate that, in turn, cause subsequent accesses to perform more data re-reads.



(a) 3x-nm MLC at the 50K wear state. (b) 2y-nm MLC at the 10K wear state. (c) 72-layer TLC at the 5K wear state.

Figure 9: Average read response time for the three SSDs (all at end-of-life wear state) with various initial time-since-written states. For 0 days, all blocks starts with no retention loss penalty. For [0,30] days, each block starts with an initial time-since-written between 0 and 30 days. Similarly, [30,90] days initializes blocks with values between 30 and 90 days. Performance is normalized to ∞-bit ECC.

- Intra-SSD redundancy (§ 4.2) shows more disadvantages than its merits, in terms of performance, write amplification, and reliability. However, when encountering a random chip and wordline failure, it is the only mechanism to recover data.

- Background scrubbing (§ 4.3) is not a cure-for-all, but is more robust, reducing the performance degradation to as low as 1.25× compared to the ideal no-error scenario even at the end-of-life states. The effectiveness of scrubbing depends on the accuracy of error prediction and internal traffic management. The oracle scrubber circumvents the first issue and reduces the probability of data re-reads, but the created internal traffic degrades performance.

Our experiments, however, are not without limitations. First, the data re-read mechanism we modeled is too optimistic, as it eventually corrects errors given enough re-reads. Because of this, uncorrectable data errors are only observed in the intra-SSD redundancy experiments in the form of recovery failures, while the other experiments are not able to produce such scenarios. A more accurate approach requires an analog model for each flash memory cell, integrated with the SSD-level details such as FTL tasks and flash memory scheduling. Second, the short 1 hour experiments are insufficient to show UBER $< 10^{-15}$. I/Os in the order of petascale are required to experimentally show this level of reliability. Lastly, while Equation 1 models flash memory errors as a function of history-dependent parameters, real flash memories nevertheless exhibit random and sporadic faults. These manifest as not only as chip and wordline failures, but also as runtime bad blocks.

5 Holistic Reliability Management

Our experiments on the effectiveness of existing reliability enhancements across a wide range of SSD states show that there is no *one size fits all* solution. Data re-read mechanism, even though optimistic in our model, not only causes long-tail latencies for that data, but also increase the error rate for other data in the same block. Intra-SSD redundancy, while relevant against random errors, does not offer significant advantages due to its high write amplification. Background data scrubbing, though relatively more robust than other techniques, accelerates wear, and the internal traffic generated by it negates the benefits of error prevention.

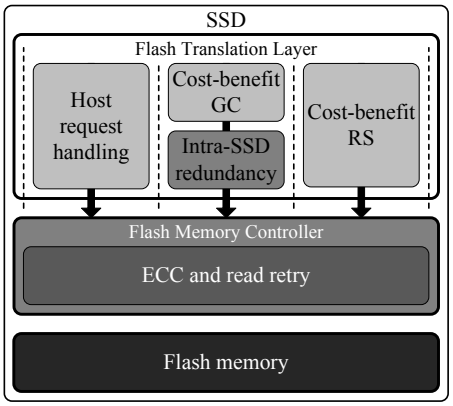


Figure 10: The overall SSD architecture for holistic reliability management. Data written by the garbage collector are protected through redundancy, and read scrubber selects data based on its cost (number of valid data) and benefit (re-read count).

Based on our observation, we propose that these existing techniques should be applied in a coordinated manner as shown in Figure 10. Redundancy should be selectively applied only to infrequently accessed *cold* data to reduce write amplification while providing protection against retention errors. Frequently read *read-hot* data should be relocated through scrubbing to reduce the data re-reads, but the benefit of scrubbing should be compared against the cost of data relocation. Update-frequent *write-hot* data require less attention as it is likely written to a fresh block due to the out-

Table 4: Trace workload characteristics. `Access footprint` is the size of the logical address space accessed, and `Data accessed` is the total amount of data transferred. `Hotness` is the percentage of data transferred in the top 20% of the frequently accessed address.

Workload	Application description	Duration (hrs)	Access footprint (GiB)		Data accessed (GiB)		Hotness (%)	
			Write	Read	Write	Read	Write	Read
DAP-DS	Advertisement caching tier	23.5	0.2	3.5	1.0	40.5	77.8	35.3
DAP-PS	Advertisement payload	23.5	35.1	35.1	42.9	35.2	34.6	20.3
LM-TBE	Map service backend	23.0	192.7	195.5	543.7	1760.0	34.4	45.1
MSN-BEFS	Storage backend file	5.9	30.8	45.8	102.3	193.7	56.9	58.7
MSN-CFS	Storage metadata	5.9	5.7	14.6	14.0	27.0	58.5	56.6
RAD-AS	Remote access authentication	15.3	4.8	1.2	18.7	2.4	63.3	53.1
RAD-BE	Remote access backend	17.0	14.7	8.3	53.3	97.0	49.0	32.7

of-place updates in SSDs. For data classification, we take advantage of SSD's existing mechanisms: data gathered by the read scrubber (RS) is read-hot, while the leftover data selected by the garbage collector (GC) is cold. Write-hot data will be naturally invalided in GC and RS's blocks, and will be re-written to new blocks allocated for host data. This approach for data classification is reactive and conservative as it relies on GC and RS's selection algorithm *after* the data is first written by the host request handler.

The background data scrubbing in § 4.3 used an oracle scrubber that knows the expected error rate for all data. This is impractical in implementation and was only used to illustrate the best-case usage of scrubbing. In the cost-benefit analysis for selecting victims to scrub, the number of valid data is used to represent the cost of relocation. The benefit is the reduction in re-reads after scrubbing, and we use the number of past re-reads for each block since its last erasure as a proxy. That is, if the number of re-reads for a block is large, the potential benefit of scrubbing that block is also large.

5.1 Workload and Test Settings

We use real-world I/O traces from Microsoft production servers [29] to evaluate the representative error-handling approaches and our proposed holistic reliability management (HRM). The traces are modified to fit into the 200GiB range, and all the accesses the aligned to 4KiB boundaries, the same as the mapping granularity for the SSD. Similarly to the synthetic workload evaluation, the logical address range is randomly written to pre-condition the SSD. Table 4 summarizes the trace workload characteristics with particular emphasis on data access pattern. `Access footprint` is the size of the logical address space accessed (of the total 200GiB), and `Data accessed` is the total amount of data transferred. `Hotness` is the percentage of data transferred in the top 20% of the frequently accessed addresses.

We evaluate the following four schemes. Intra-SSD redundancy is omitted as it performed badly due to high write amplification.

∞-bit ECC corrects all errors, and any performance degradation is caused by queueing delays and garbage collection. This represents the baseline performance.

ECC + re-read (§ 4.1) relies on the ECC engine of the flash memory controller, and repeatedly re-reads the data until the error is corrected. 4-bit ECC is used for the `3x`-nm SSD, 10-bit ECC for the `2y`-nm SSD, and 75-bit for the 72-layer.

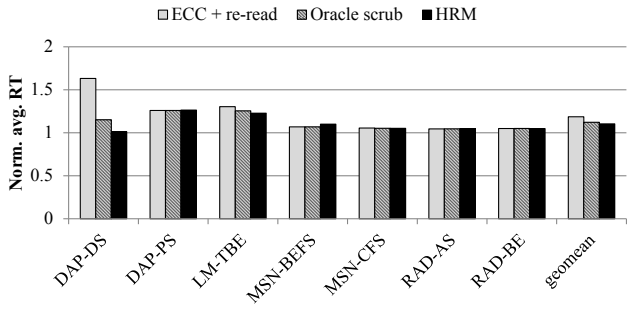
Oracle scrub (§ 4.3) knows the expected number of errors for all data and preventively relocates them before errors accumulate. In an unfortunate event of an ECC failure, it falls back to the `ECC + re-read` approach.

HRM (§ 5) selectively employs redundancy to data gathered by the garbage collector, conditionally re-reads data depending on its redundancy level, and judiciously manages data scrubbing through a cost-benefit analysis.

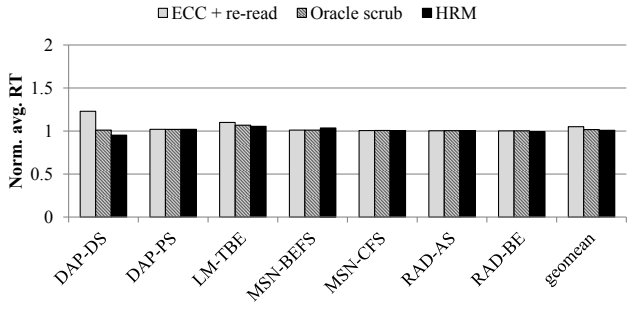
5.2 Experimental Results

Figure 11 shows the performance of ECC+re-read, Oracle scrub, and the proposed HRM, normalized to the performance of ∞-bit ECC on the three SSDs, each at its end-of-life phase. One of the most noticeable results is the performance under `DAP-DS`, which shows that repeated data re-reads severely degrade the performance. `DAP-DS` has a small write footprint (0.2GiB accessed), a high read/write ratio (40.5GiB read vs. 1.0GiB write), and a high write-hotness (77.8% of write data are to 20% of the address). This means that without preventive data relocation, only a small write-hot region will be frequently relocated during garbage collection, and a large region of read-only data suffers from read disturbance. In some cases, HRM even performs better than the baseline, as shown in `DAP-DS` of Figure 11b. This is due to data relocation (unexpectedly) improving parallelism.

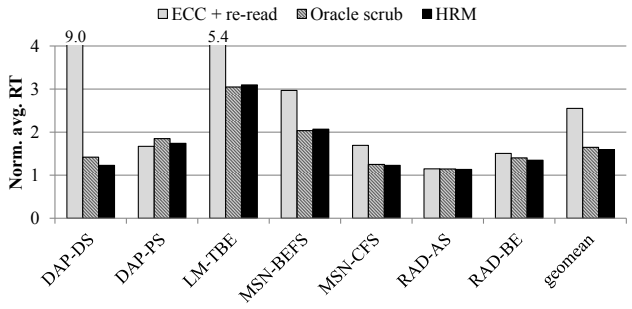
Though marginal, ECC+re-read achieves better performance under `DAP-PS` in Figure 11c. The lack of read skew in the workload (only 20.3% of reads from the top 20% of the address) reduces the effects of read disturbance, as accesses



(a) 3x-nm MLC at the 50K wear state.



(b) 2y-nm MLC at the 10K wear state.



(c) 72-layer TLC at the 5K wear state.

Figure 11: Average read response time on three SSDs, all at their end-of-life phase. In order to ensure data integrity, ECC+re-read adds 18.7%, 5.0%, and 155.1% overhead on average for the 3x-nm MLC, 2y-nm MLC, and 72-layer TLC, respectively. Preventive measures reduce this overhead in general: Oracle scrub adds 12.3%, 1.7%, and 64.5% overhead, and HRM adds 10.4%, 0.9%, and 59.0% overhead, respectively.

are not concentrated to a particular block. On the other hand, workloads with high read skew (LM-TBE, MSN-BEFS, and MSN-CFS) show that the performance degradation can be reduced by preventively relocating data when the read accesses are concentrated. In particular, Oracle scrub outperforms HRM under MSN-BEFS for all three SSDs. This performance gap between Oracle scrub and HRM, however, is very small: 3% at most.

Overall, ECC+re-read adds 18.7%, 5.0%, and 155.1% overhead on average for reliability to the 3x-nm MLC (Figure 11a), 2y-nm MLC (Figure 11b), and 72-layer TLC (Figure 11c), respectively. Oracle scrub that knows the error rate for all data reduces this overhead to 12.3%, 1.7%, and 64.5%, respectively. By judiciously selecting data to relocate, HRM further reduces the overhead to 10.4%, 0.9%, and 59.0%, respectively. In HRM, data not relocated by the scrubber are protected by selective redundancy. Even though Oracle scrub represents the upper-bound benefit of scrubbing, HRM achieves better performance overall by reducing the relocation traffic and delegating the responsibility of data protection for unaccessed data to redundancy.

6 Related Work

To the best of our knowledge, our work first presents a holistic study on the interactions between multiple reliability enhancement techniques and their overall impact on performance in modern SSDs. Our work builds upon a number of prior work from the reliability enhancement techniques to QoS-conscious SSD designs for large-scale deployments.

6.1 Reliability Enhancement

LDPC is widely used in the communications domain and is slowly gaining attention for storage due to its error correction capability. In the context of flash memory-based storages, LDPC-in-SSD [54] reduces the LDPC's long response time by speculatively starting the soft-decision decoding and progressively increasing the iterative memory-sensing level. However, its interaction with other reliability enhancement techniques is not examined.

A series of work exists on threshold voltage prediction for flash memory reads. HeatWatch [38] predicts the change in threshold voltage level caused by the self-recovery effect of flash memory, RDR [13] predicts the changes caused by read disturbance, and ROR [14], by retention error. While these techniques reduce the raw bit error rate by 36%–93.5%, the system-level implications (particularly for QoS performance) are not extensively covered.

Aside from threshold voltage tuning, other tunable voltages exist in flash memory, and several prior work study the performance and reliability tradeoff for these settings. Reducing the read pass-through voltage mitigates the effects of read disturbance [13, 22], and tuning the program voltages tradeoff the flash memory's wear and SSD's write performance [26, 36, 51]. These approaches complement our study and further diversify the system parameters in the performance-reliability spectrum.

Prior work on intra-SSD redundancy techniques focus on reducing the overhead of parity updates [25,34], dynamically managing the stripe size [31,33], and intra-block incremental redundancy [46]. While these approaches are relevant for enhancing the SSD's reliability, we focus on the interaction of these techniques with error correction and error prevention schemes.

In addition to considering multiple reliability enhancement techniques, our work borrows ideas from prior work

on judicious data placement that supplements data protection. LDPC-in-SSD [54] splits a single ECC codeword across multiple memory chips to guard against asymmetric wears; WARM [37] groups write-hot data together and relaxes the management overhead for preventing retention-induced errors; RedFTL [22] identifies frequently read pages and places them in reliable blocks to reduce the overhead of read reclaim. In our work, we holistically cover all causes of errors and study the interactions among multiple reliability enhancements.

Our investigation of SSD's multiple reliability enhancement schemes is inspired by HDD's sector error studies that evaluate intra-HDD redundancy schemes and disk scrubbing techniques [24, 40, 49]. However, we re-assess the effectiveness of these techniques in the context of SSDs for the following three reasons. First, the out-of-place update in SSDs makes intra-SSD redundancy techniques [31,33,46] to be fundamentally different from intra-HDD techniques [17] that allow in-place update of parity data. Second, the existing need for SSD-internal data relocations amortizes the overhead of implementing read reclaim/refresh inside the SSD, while disk scrubbing for HDDs requires external management [6, 45]. Lastly, error patterns in flash memory are history-dependent (number of erase, time-since-written, and number of reads), and can be monitored and controlled to manage the error rate; this is in contrast to errors in HDD that are mostly random events (though temporally and spatially bursty) [8, 45, 49].

6.2 QoS Performance

Improving the QoS performance of SSDs is of great interest in large-scale systems [16, 21, 23], and few recent work suggest several methods in designing SSDs with short tail latencies. AutoSSD [30] dynamically manages the intra-SSD housekeeping tasks (such as garbage collection and read scrubbing) using a control theoretic-approach for a stable performance state, and RLGC [28] schedules garbage collection by predicting the host's idle time using reinforcement learning. ttFlash [53] exploits the existing intra-SSD redundancy scheme and reconstructs data when blocked by SSD-internal tasks to improve QoS performance. We expect QoS-aware scheduling to become increasingly important as more flash memory quirks are introduced, and reliability management in conjunction with scheduling is a central design decision for reducing tail latencies.

Despite the various efforts at the SSD device-level for QoS performance, large-scale systems nevertheless replicate data across devices, servers, data centers for responsiveness and fault-tolerance [16]. Thus, *fail-fast* SSDs are desirable over *fail-slow* ones under such circumstances so that replicated data are instead retrieved. This has culminated in the proposal of *read recovery level* [4] that allows a configurable tradeoff between QoS performance and device error rate. Such tunable service-level agreement between the system and the device further necessitates an comprehensive reliability management.

The increasingly unreliable trend of flash memory incites large production environments to independently study the failure patterns of SSDs [39, 44, 50]. While these studies provide valuable insight on correlating SSD failures and monitored information (such as erase counts, number of reads, amount of data written), they do not directly address how SSD reliability enhancement techniques should be constructed internally.

7 Conclusion

In this work, we examine the design tradeoffs of the existing reliability enhancement techniques in SSDs across multiple dimensions such as performance, write amplification, and reliability. Our findings show that existing solutions exhibit both strengths and weaknesses, and based on our observations, we propose a reliability management scheme that selectively applies appropriate techniques to different data.

There are several research directions that need further investigation. First, the limitation of our study reveals the necessity to integrate the SSD-level design framework (FTL and flash controller) and memory cell-level models that accurately describe electron distributions. Second, there exists a need to mathematically model the effectiveness of data rereads and data scrubbing, so that device reliability can be demonstrated without petascale I/O workloads.

Acknowledgment

We thank our shepherd, Peter Desnoyers, and the anonymous reviewers for their constructive and insightful comments. This work was supported in part by SK Hynix and the Basic Research Laboratory Program through the National Research Foundation of Korea (NRF-2017R1A4A1015498). The Institute of Computer Technology at Seoul National University provided the research facilities for this study.

Notes

[1] The dataset for the 2y-nm MLC comes from two different papers by the same author. The author informed us that the two papers used *different* chips of the *same* manufacturer. Due to lack of publically available RBER data of similar generation, however, we assume that the two chips are similar enough to create a representative 2y-nm MLC. One paper provided error rate as a function of wear and time-since-written, while the other, as a function of wear and read disturbance.

[2] During ECC checks, errors are generated randomly using the binomial probability distribution, but the expected number of errors for the oracle scrubber is deterministically computed using the RBER at that moment. This means that ECC may fail even if the expected number of errors is below the `correction strength`, and *vice versa*.

References

[1] The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). `http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf`, 2008. Parallel Data Laboratory.

[2] Jedec ssd endurance workloads. `https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2011/20110810_T1B_Cox.pdf`, 2011. Flash Memory Summit.

[3] Webplotdigitizer. `https://automeris.io/WebPlotDigitizer/`, 2011. Ankit Rohatgi.

[4] Solving latency challenges with NVM express SSDs at scale. `https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2017/20170809_SIT6_Petersen.pdf`, 2017. Flash Memory Summit.

[5] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M. S., AND PANIGRAHY, R. Design tradeoffs for SSD performance. In *2008 USENIX Annual Technical Conference, USENIX ATC 2008, Boston, MA, USA, June 22-27, 2008.* (2008), pp. 57–70.

[6] AMVROSIADIS, G., OPREA, A., AND SCHROEDER, B. Practical scrubbing: Getting to the bad sector at the right time. In *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012* (2012), pp. 1–12.

[7] ARITOME, S. *NAND flash memory technologies*. John Wiley & Sons, 2015.

[8] BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2007, San Diego, California, USA, June 12-16, 2007* (2007), pp. 289–300.

[9] CAI, Y., GHOSE, S., HARATSCH, E. F., LUO, Y., AND MUTLU, O. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE 105*, 9 (2017), 1666–1704.

[10] CAI, Y., GHOSE, S., LUO, Y., MAI, K., MUTLU, O., AND HARATSCH, E. F. Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017* (2017), pp. 49–60.

[11] CAI, Y., HARATSCH, E. F., MUTLU, O., AND MAI, K. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012* (2012), pp. 521–526.

[12] CAI, Y., HARATSCH, E. F., MUTLU, O., AND MAI, K. Threshold voltage distribution in MLC NAND flash memory: characterization, analysis, and modeling. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013* (2013), pp. 1285–1290.

[13] CAI, Y., LUO, Y., GHOSE, S., AND MUTLU, O. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015, Rio de Janeiro, Brazil, June 22-25, 2015* (2015), pp. 438–449.

[14] CAI, Y., LUO, Y., HARATSCH, E. F., MAI, K., AND MUTLU, O. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015* (2015), pp. 551–563.

[15] CAI, Y., YALCIN, G., MUTLU, O., HARATSCH, E. F., CRISTAL, A., ÜNSAL, O. S., AND MAI, K. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *30th International IEEE Conference on Computer Design, ICCD 2012, Montreal, QC, Canada, September 30 - Oct. 3, 2012* (2012), pp. 94–101.

[16] DEAN, J., AND BARROSO, L. A. The tail at scale. *Commun. ACM 56*, 2 (2013), 74–80.

[17] DHOLAKIA, A., ELEFTHERIOU, E., HU, X., ILIADIS, I., MENON, J., AND RAO, K. K. A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. *TOS 4*, 1 (2008), 1:1–1:42.

[18] GALLAGER, R. G. Low-density parity-check codes. *IRE Trans. Information Theory 8*, 1 (1962), 21–28.

[19] GRUPP, L. M., CAULFIELD, A. M., COBURN, J., SWANSON, S., YAAKOBI, E., SIEGEL, P. H., AND WOLF, J. K. Characterizing flash memory: anomalies, observations, and applications. In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42) 2009, December 12-16, 2009, New York, New York, USA* (2009), pp. 24–33.

[20] GRUPP, L. M., DAVIS, J. D., AND SWANSON, S. The bleak future of NAND flash memory. In *10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012* (2012), p. 2.

[21] GUNAWI, H. S., SUMINTO, R. O., SEARS, R., GOLLIHER, C., SUNDARARAMAN, S., LIN, X., EMAMI, T., SHENG, W., BIDOKHTI, N., MCCAFFREY, C., GRIDER, G., FIELDS, P. M., HARMS, K., ROSS, R. B., JACOBSON, A., RICCI, R., WEBB, K., ALVARO, P., RUNESHA, H. B., HAO, M., AND LI, H. Fail-slow at scale: Evidence of hardware performance faults in large production systems. In *16th USENIX Conference on File and Storage Technologies, FAST 2018, Oakland, CA, USA, February 12-15, 2018.* (2018), pp. 1–14.

[22] HA, K., JEONG, J., AND KIM, J. An integrated approach for managing read disturbs in high-density NAND flash memory. *IEEE Trans. on CAD of Integrated Circuits and Systems 35*, 7 (2016), 1079–1091.

[23] HAO, M., SOUNDARARAJAN, G., KENCHAMMANA-HOSEKOTE, D. R., CHIEN, A. A., AND GUNAWI, H. S. The tail at store: A revelation from millions of hours of disk and SSD deployments. In *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016.* (2016), pp. 263–276.

[24] ILIADIS, I., HAAS, R., HU, X., AND ELEFTHERIOU, E. Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems. In *2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2008, Annapolis, MD, USA, June 2-6, 2008* (2008), pp. 241–252.

[25] IM, S., AND SHIN, D. Flash-aware RAID techniques for dependable and high-performance flash memory SSD. *IEEE Trans. Computers 60*, 1 (2011), 80–92.

[26] JEONG, J., HAHN, S. S., LEE, S., AND KIM, J. Lifetime improvement of NAND flash-based storage systems using dynamic program and erase scaling. In *12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014* (2014), pp. 61–74.

[27] JIMENEZ, X., NOVO, D., AND IENNE, P. Wear unleveling: improving NAND flash lifetime by balancing page endurance. In *12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014* (2014), pp. 47–59.

[28] KANG, W., SHIN, D., AND YOO, S. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Trans. Embedded Comput. Syst. 16*, 5 (2017), 134:1–134:20.

[29] KAVALANEKAR, S., WORTHINGTON, B. L., ZHANG, Q., AND SHARDA, V. Characterization of storage workload traces from production Windows servers. In *4th International Symposium on Workload Characterization (IISWC) 2008, Seattle, Washington, USA, September 14-16, 2008* (2008), pp. 119–128.

[30] KIM, B. S., YANG, H. S., AND MIN, S. L. AutoSSD: an autonomic SSD architecture. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018.* (2018), pp. 677–690.

[31] KIM, J., LEE, E., CHOI, J., LEE, D., AND NOH, S. H. Chip-level RAID with flexible stripe size and parity placement for enhanced SSD reliability. *IEEE Trans. Computers 65*, 4 (2016), 1116–1130.

[32] KIM, J., LEE, J., CHOI, J., LEE, D., AND NOH, S. H. Improving SSD reliability with RAID via elastic striping and anywhere parity. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013* (2013), pp. 1–12.

[33] LEE, S., LEE, B., KOH, K., AND BAHN, H. A lifespan-aware reliability scheme for RAID-based flash storage. In *2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011* (2011), pp. 374–379.

[34] LEE, Y., JUNG, S., AND SONG, Y. H. FRA: a flash-aware redundancy array of flash storage devices. In *7th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2009, Grenoble, France, October 11-16, 2009* (2009), pp. 163–172.

[35] LIN, S., AND COSTELLO, D. J. *Error Control Coding*. Prentice-Hall, Inc., 2004.

[36] LIU, R., YANG, C., AND WU, W. Optimizing NAND flash-based SSDs via retention relaxation. In *10th USENIX conference on File and Storage Technologies, FAST 2012, San Jose, CA, USA, February 14-17, 2012* (2012), p. 11.

[37] LUO, Y., CAI, Y., GHOSE, S., CHOI, J., AND MUTLU, O. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In *IEEE 31st Symposium on Mass Storage Systems and Technologies, MSST 2015, Santa Clara, CA, USA, May 30 - June 5, 2015* (2015), pp. 1–14.

[38] LUO, Y., GHOSE, S., CAI, Y., HARATSCH, E. F., AND MUTLU, O. HeatWatch: Improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018* (2018), pp. 504–517.

[39] MEZA, J., WU, Q., KUMAR, S., AND MUTLU, O. A large-scale study of flash memory failures in the field. In *2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Portland, OR, USA, June 15-19, 2015* (2015), pp. 177–190.

[40] MI, N., RISKA, A., SMIRNI, E., AND RIEDEL, E. Enhancing data availability in disk drives through background activities. In *38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2008, June 24-27, 2008, Anchorage, Alaska, USA, Proceedings* (2008), pp. 492–501.

[41] MICHELONI, R. *3D Flash Memories*. Springer, 2016.

[42] MICHELONI, R., CRIPPA, L., AND MARELLI, A. *Inside NAND Flash Memories*. Springer, 2010.

[43] MIELKE, N. R., FRICKEY, R. E., KALASTIRSKY, I., QUAN, M., USTINOV, D., AND VASUDEVAN, V. J. Reliability of solid-state drives based on NAND flash memory. *Proceedings of the IEEE 105*, 9 (2017), 1725–1750.

[44] NARAYANAN, I., WANG, D., JEON, M., SHARMA, B., CAULFIELD, L., SIVASUBRAMANIAM, A., CUTLER, B., LIU, J., KHESSIB, B. M., AND VAID, K. SSD failures in datacenters: What? when? and why? In *9th ACM International on Systems and Storage Conference, SYSTOR 2016, Haifa, Israel, June 6-8, 2016* (2016), pp. 7:1–7:11.

[45] OPREA, A., AND JUELS, A. A clean-slate look at disk scrubbing. In *8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 23-26, 2010* (2010), pp. 57–70.

[46] PARK, H., KIM, J., CHOI, J., LEE, D., AND NOH, S. H. Incremental redundancy to reduce data retention errors in flash-based SSDs. In *IEEE 31st Symposium on Mass Storage Systems and Technologies, MSST 2015, Santa Clara, CA, USA, May 30 - June 5, 2015* (2015), pp. 1–13.

[47] PATTERSON, D. A., GIBSON, G. A., AND KATZ, R. H. A case for redundant arrays of inexpensive disks (RAID). In *1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 1-3, 1988.* (1988), pp. 109–116.

[48] PRINCE, B. *Vertical 3D memory technologies*. John Wiley & Sons, 2014.

[49] SCHROEDER, B., DAMOURAS, S., AND GILL, P. Understanding latent sector errors and how to protect against them. In *8th USENIX Conference on File and Storage Technologies, FAST 2010, San Jose, CA, USA, February 23-26, 2010* (2010), pp. 71–84.

[50] SCHROEDER, B., LAGISETTY, R., AND MERCHANT, A. Flash reliability in production: The expected and the unexpected. In *14th USENIX Conference on File and Storage Technologies, FAST 2016, Santa Clara, CA, USA, February 22-25, 2016.* (2016), pp. 67–80.

[51] SHI, L., QIU, K., ZHAO, M., AND XUE, C. J. Error model guided joint performance and endurance optimization for flash memory. *IEEE Trans. on CAD of Integrated Circuits and Systems 33*, 3 (2014), 343–355.

[52] SUN, H., GRAYSON, P., AND WOOD, B. Quantifying reliability of solid-state storage from multiple aspects. In *7th IEEE Storage Networking Architecture and Parallel I/O, SNAPI 2011, Denver, CO, USA, May 23-27, 2011.* (2011).

[53] YAN, S., LI, H., HAO, M., TONG, M. H., SUNDARARAMAN, S., CHIEN, A. A., AND GUNAWI, H. S. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017* (2017), pp. 15–28.

[54] ZHAO, K., ZHAO, W., SUN, H., ZHANG, T., ZHANG, X., AND ZHENG, N. LDPC-in-SSD: making advanced error correction codes work effectively in solid state drives. In *11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013* (2013), pp. 243–256.