Incremental Redundancy to Reduce Data Retention Errors in Flash-based SSDs

Heejin Park
University of Seoul
bakhi@uos.ac.kr

Jaeho Kim
Hongik University
kjhnet@gmail.com

Jongmoo Choi
Dankook University
choijm@dankook.ac.kr

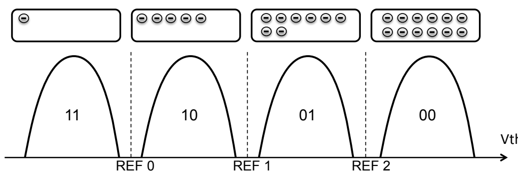
Donghee Lee
University of Seoul
dhl_express@uos.ac.kr

Sam H. Noh
Hongik University
http://next.hongik.ac.kr

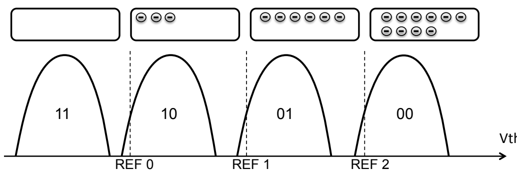
Abstract*—As the market becomes competitive, SSD manufacturers are making use of multi-bit cell flash memory such as MLC and TLC chips in their SSDs. However, these chips have lower data retention period and endurance than SLC chips. With the reduced data retention period and endurance level, *retention errors* occur more frequently. One solution for these retention errors is to employ strong ECC to increase error correction strength. However, employing strong ECC may result in waste of resources during the early stages of flash memory lifetime as it has high reliability and data retention errors are rare during this period. The other solution is to employ *data scrubbing* that periodically refreshes data by reading and then writing the data to new locations after correcting errors through ECC. Though it is a viable solution for the retention error problem, data scrubbing hurts performance and lifetime of SSDs as it incurs extra read and write requests. Targeting data retention errors, we propose incremental redundancy (IR) that incrementally reinforces error correction capabilities when the data retention error rate exceeds a certain threshold. This extends the time before data scrubbing should occur, providing a grace period in which the block may be garbage collected. We develop mathematical analyses that project the lifetime and performance of IR as well as when using conventional data scrubbing. Through mathematical analyses and experiments with both synthetic and real workloads, we compare the lifetime and performance of the two schemes. Results suggest that IR can be a promising solution to overcome data retention errors of contemporary multi-bit cell flash memory. In particular, our study shows that IR can extend the maximum data retention period by 5 to 10 times. Additionally, we show that IR can reduce the write amplification factor by half under real workloads.

I. INTRODUCTION

Solid State Drives (SSDs) that use flash memory as storage media are now popular in computer systems as well as mobile devices due to advantages such as superior performance, low power consumption, and shock resistance. In consumer products, MLC/TLC (Multiple/Triple Level Cell) flash memory chips that store two or three bits per cell are employed, while SLC (Single Level Cell) based SSDs prevail in the enterprise market [1]. Compared to SLC flash memory, MLC/TLC flash memory can significantly reduce cost, but it comes with sacrificed performance, lifetime, and reliability. In particular, the bit error rate increases as feature size decreases, density increases, and more bits are condensed into a cell. More importantly, it is well known that the bit error rate is aggravated 1) as the P/E cycle increases with extensive use and 2) as the data retention period, that is, the time that has passed since the data has been written to, increases [2]. Errors due to long data retention periods are generally referred to as data retention errors, and henceforth, we will refer to this problem as the data retention problem. One of reasons of the retention error is the leak of injected electrons from floating gates. Fig. 1(a) shows the ideal case of threshold voltages and the number of electrons in a flash memory cell right after programming. As time goes on, some injected electrons are leaked and threshold voltages shift across boundaries as seen in Fig. 1(b), eventually causing retention errors. This retention error occurs more frequently in aged flash memory blocks with higher P/E cycles.



(a) Threshold voltages and # of electrons



(b) Leaking of injected electrons

Fig. 1: Data retention error

To cope with bit errors, redundancy information such as ECC (Error Correction Code) is adopted in the OOB (Out Of Band) area of each page in flash memory. However, current flash memory such as TLC flash chips with 2xnm technology has high error rates that require considerable OOB space for strong ECC. Moreover, enormous ECC space may be needed to cope with worst case scenarios where the P/E cycles is high (after substantial use) and the retention period for the data has grown large. In particular, the worst case scenario where only a small fraction of cold data that sits still for an extended time period on an aged flash memory device with high P/E cycles may be rare and little. Unfortunately, providing large and strong ECC for the entire SSD to insure itself from such corner cases not only wastes a significant portion of OOB space, but also hurts performance and energy efficiency because stronger ECC requires longer decoding time and more energy.

Another solution, specifically targeted for the data retention problem, is to employ *data scrubbing*, which is a scheme that periodically refreshes data by reading and then writing the data to a different location when the bit error rate exceeds a certain safe level. Though data scrubbing is a viable solution for retention errors, it has serious drawbacks in that it hurts performance and lifetime of SSDs as it incurs extra reads, writes, and eventually erase operations.

The main goal of this study is to provide an alternative, efficient solution to the data retention problem. To this end, we propose, what we call, incremental redundancy (IR), a scheme that reinforces error correction strength when the error rate exceeds a safe level. By so doing, data scrubbing is postponed for an extended period and the flash memory block earns a grace period in which data in the block may be naturally refreshed by write requests and through garbage collection. We emphasize that additional space and operations for IR are required only when the error rate exceeds a safe level while strong ECC always occupies space.

There are various ways in which IR can be implemented. In this paper, we first present vertical striping, a simple, low overhead IR scheme[1]. In vertical striping, a stripe is composed of pages residing within the same block, with each page being a strip. The name comes from the fact that a block in flash can be visualized a being composed of pages that are numbered from top to bottom. Then, we derive mathematical analyses that estimates the performance and lifetime improvements that are possible with IR compared to conventional data scrubbing. We validate our mathematical analyses through experiments with realistic workloads. Both mathematical and experimental results show that IR brings about performance and lifetime benefits.

The rest of the paper is organized as follows. In the next section, we describe work related to our study including flash memory and data scrubbing. In Section III, we present the vertical striping technique that is used to implement IR and introduce the notion of a safe level and a safe period. Then, in Section IV, we discuss error rate modelling. We derive the write amplification factor (WAF) under random workload in Section V and WAF under Hot/Cold workload in Section VI. We present evaluation results in Section VII and, finally, we conclude with Section VIII.

II. Background and Related Work

A. Flash Memory Basics

Flash memory, which comes in the form of flash memory chips, is a non-volatile storage medium that can be erased and reprogrammed electrically. The now popular solid state drives (SSDs) are composed of numerous flash memory chips (denoted Chip 0, Chip 1, etc.) as shown in Fig. 2. A flash memory chip consists of multiple blocks and each block has multiple pages. PBN (Physical Block Number) and PPN (Physical Page Number) in Fig. 2, respectively, refers to the block number within the chip and the page number within the block. Read and write (also referred to as program in flash terminology) operations are the most basic operations, and these are performed in page units. A unique characteristic of flash memory is that data cannot be overwritten on a used page. Hence, when data are updated, it needs to be written to a different location. This results in blocks that contain pages that no longer contain valid data. These are referred to as invalid pages. To make invalid pages writeable once again, an erase operation, which is performed in block units, must be performed. The process of programming and erasing is called the P/E (program/erasure) cycle and the number of P/E cycles is limited. The P/E cycle limit determines the lifespan of the flash memory chip, and the actual number depends on the manufacturer and the technology used. As the number of invalid pages increases, the chip may run out of space. To retain writeable pages, garbage collection (GC) that goes about moving valid pages and erasing blocks is performed. A software layer called the flash translation layer (FTL) [3], [4], [5] is generally provided to hide these unique flash memory characteristics from the user so that the same HDD interface may be used.

B. Reliability of Flash Memory Device

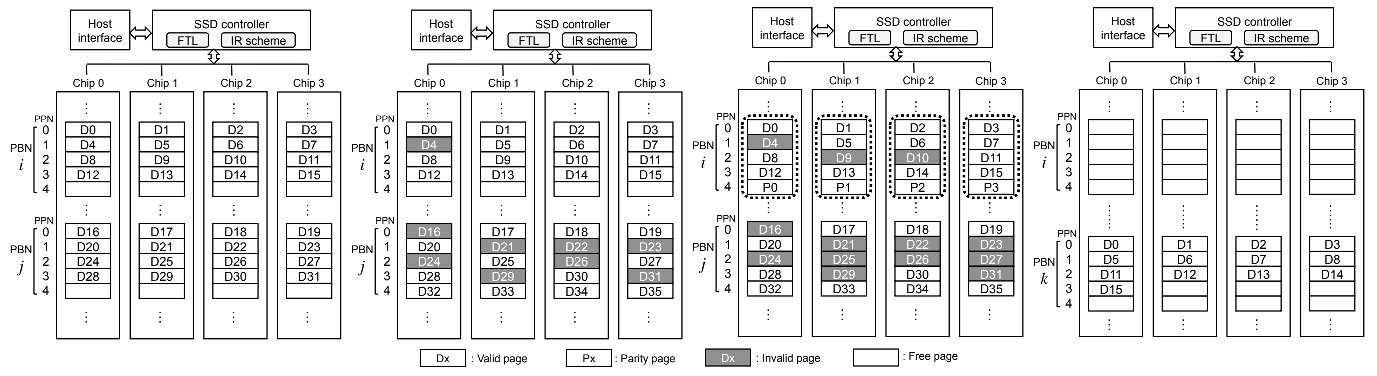
SSDs now use MLC (Multi-Level Cell) and TLC (Triple-Level Cell) flash memory that have higher density and lower price than SLC flash memory. However, their advantages come with sacrifice in reliability. For example, the number of P/E cycles permitted for MLC is around 10,000, but they drop to a few thousands for TLC, whereas for SLC it is around 100,000 [1], [6]. Besides P/E cycles, MLC/TLC flash memory suffers from high bit error rates, and the reliability issue is exacerbated as P/E cycles and the data retention period[2] increases [2], [7], [8].

Mielke et al. measure the raw bit error rates as well as the uncorrectable bit error rates after ECC correction with flash memory chips of various vendors [8]. Also, Sun et al. measure the raw bit error rates of 5xnm, 4xnm, and 3xnm MLC flash memory chips, showing that the bit error rate increases as the cell size becomes smaller [9]. These studies showed that the bit error rate increases exponentially with the increase in P/E cycles.

To cope with bit errors, redundancy information such as ECC (Error Correction Code) is stored in the OOB (Out of Band) area [10], [11], [12]. However, the OOB area is limited in size and is shared with other meta-data such as the logical block number. RAID has been used to supplement the ECC limitations. It has been applied both with the chips within the SSDs or with the SSDs themselves [13], [14], [15], [16], [17]. Jimenez et al. propose switching from MLC to SLC mode to extend the lifetime of SSDs [18] and Liu et al. employ incrementally stronger ECC to cope with increased BER [19].

[1]Hereafter, IR will be meant to refer to vertical striping IR unless otherwise mentioned.

[2]Hereafter, we will use the shorthand terms "P/E cycles" and "data retention period" to mean the P/E cycles that have been executed and the time period that data is retained in a cell after a write, respectively, throughout our discussion.



(a) Before observation period (b) After observation period (c) After safe period (d) After scrubbing or GC

Fig. 2: Dynamic vertical striping

These studies are similar to ours in that striping is employed or in that they incrementally reinforce error correction strength. However, our study differs in that we target the data retention problem working in concert with data scrubbing and that we provide mathematical analyses for the methods.

Hybrid use of flash memory has also been suggested to overcome the inferior performance and endurance problem of MLC/TLC flash memory. Chang employs SLC cache inside MLC-based SSDs [20], while Im and Shin propose storing hot data in the SLC region and cold data in the MLC region [21]. Also, Park et al. propose using both SLC and MLC chips inside SSDs [22]. The focus of these studies, however, is on performance and not on data retention errors.

Deriving analytic models has been a topic of research for storage devices. Wang et al. develop performance models for LFS (Log-structured File System) and their results have been used in modeling garbage collection costs of flash memory based storage [23]. Hu et al. analyse write amplification of flash memory storage [24] and Desnoyers derives complicated cost models including the WAF of flash memory storage under random and hot/cold workloads [25]. Also, Oh et al. propose cost models of hybrid storage with SSD cache and HDD-based back-end storage [26]. Recently, Kim et al. propose cost models for SSDs employing the RAID architecture [27]. Though these studies focus on deriving analytic models for flash memory storage, they differ from ours in that our goal is in modeling the overhead for data protection while they focus on performance and/or lifetime without considering data errors.

C. Data Scrubbing

Unless strong ECC with sufficient length enough to cope with the worst case errors is provided, ECC may not be able to correct errors of cold data in aged flash memory. In reality, some cold data do not change for long periods, accumulating, what is generally called, data retention errors [28]. To resolve these data retention errors, the FTL inside an SSD periodically refreshes data by performing data scrubbing operations [29]. Though this is a simple and feasible solution, data scrubbing has a serious drawback in that it hurts performance and lifetime of SSDs.

Cai et al. observe that retention errors are dominant in flash memory, which can be mitigated by FCR (Flash Correct-and-Refresh) [30]. In addition, to reduce the overhead of refresh, they propose two techniques; one is hybrid FCR that applies remapping and reprogramming selectively and the other is adaptive FCR that has a low refresh rate for the block with smaller retention errors. Pan et al. design a new SSD, called quasi-nonvolatile SSD, which trades data retention time for endurance and performance, and develop an SSD scheduling scheme to minimize the impact of internal refresh on normal I/O requests [31]. Mohan et al. devise an EDF (Earliest-Deadline First) based refresh policy to improve the endurance of SSDs [32]. Liu et al. utilize refresh for data that are not overwritten in the guaranteed retention time when they optimize SSDs via retention relaxation [33]. Our proposal differs from these in that they are trying to reduce the refresh overhead while ours extends the refresh cycle with incremental redundancy.

PCM (Phase Change Memory), another type of new emerging non-volatile memory, also suffers from retention errors due to the phenomenon called resistance drift, which is similar to charge loss in flash memory. To overcome this problem, several studies also make use of data scrubbing. For instance, Awasthi et al. discuss how to extend scrub mechanisms for MLC PCM with stronger ECC codes, headroom schemes, and adaptive rates [34]. Liu et al. propose a smart refreshing scheme that eliminates unnecessary refreshing operations for their NVM-duet architecture [35].

III. Incremental redundancy in SSDs

Our solution to data retention errors due to lengthy data retention periods is to incrementally reinforce the data recovery capability as needed. This is done by incrementally increasing redundancy. Simply, redundancy can be incremented in SSDs upon need, for example, when the error rate starts to exceed a certain level, by dynamically constructing a stripe with data and writing parities for them. Incrementally increasing redundancy with such dynamic striping requires extra effort such as managing the stripe map and reconstructing stripes during garbage collection. Hence, a naive approach may negate the benefits brought about through incremental redundancy.

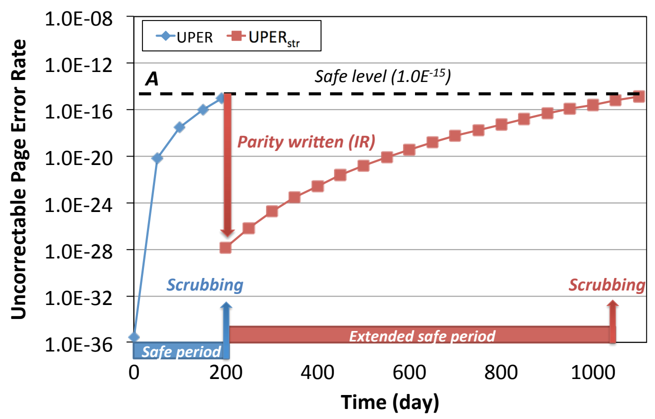


Fig. 3: Error rate variation with traditional data scrubbing and IR when P/E cycles = 3K

Symbol	Description
$RBER$	Raw Bit Error Rate
$RBER_{th}$	Threshold RBER
$RBER_{th\_str}$	Threshold RBER with striping applied
$CPER$	Correctable Page Error Rate
$DPER$	Detectable Page Error Rate
$UPER$	Uncorrectable Page Error Rate
$UPER_{str}$	UPER with stripe constructed
$UPER_{th}$	Threshold UPER equal to HDD (10^{-15})
T_{safe}	Safe Period
T_{esafe}	Extended Safe Period
U	User Space
u	Utilization of Block
a	Over-Provisioning Factor $a = \frac{ExtraSpace}{UserSpace}$
p_u	Update probability ($\frac{\#\ of\ writes}{data\ space}$)
D_{write}	Amount of write requests per day $(U \cdot p_u)$
$D_{scrub(n)}$	Amount of scrubbed data

TABLE I: Symbols used in modeling

Thus, for IR to become a promising solution for data retention errors, it should be implementable in existing SSDs with little overhead. To this end, in this section, we present *vertical striping*, a low overhead IR scheme. Then in the next section, we derive the mathematical analyses of vertical striping IR.

Let us start with Fig. 2 to illustrate vertical striping. Assume initially that data D0∼D15 are striped over PBN (Physical Block Number) i and data D16∼D31 are striped over PBN j of Chips 0∼3. This is typically what is done in today's SSDs. Updating data in PBN i and j results in pages that are occupied by invalid data, which are shown as shaded pages in Fig. 2. Note that the last pages of PBN i's and j's are reserved for other purposes to be explained later. Some of these turn out to be space overhead required of vertical striping. However, we later show that this overhead is minimal compared to the gains possible through IR.

After a certain observation period, SSD firmware determines that PBN i's are cold data blocks and PBN j's are hot data blocks. In general, this decision can be made quickly because hot data tend to be updated quickly due to temporal locality [3], [36]. Once the hot and cold decision is made, different actions are taken based on this decision. For hot data blocks, the firmware uses the reserved pages for write requests. In particular, data D32∼D35 are written to the reserved pages as shown in Fig. 2(b). The basic premise behind this choice is that a hot data block will be reclaimed soon and thus, data in the block will be refreshed through garbage collection (GC).

For cold data, though, a different action is taken. To explain the actions, we first define two terms. The first is *safe level*, which is the data error rate level at which the firmware considers the data in the block to be unsafe. The other is *safe period*, which to be the time period from write of data to the time when the block reaches its safe level. Now, assume that PBN i's are rarely updated until the safe period expires as they are holding cold data. Then, the firmware writes parities to the reserved pages and constructs stripes to protect data in those blocks as shown in Fig. 2(c). In particular, parity for data D0, D4, D8, and D12 is calculated and written to the reserved page of PBN i in Chip 0. Likewise, parities for data in PBN i of Chips 1∼3 are calculated and written to the reserved pages in PBN i. In Fig. 2(c), the firmware constructs four stripes, each of them consisting of four data pages and one parity page within the same block. The number of parities per stripe can be increased by preserving more parity pages in a block to provide stronger data recoverability.

We refer to this as *dynamic vertical striping* as stripes are constructed dynamically and vertically within the same block. In practice, writing parities is orthogonal to processing read and write requests and can be done in background between the observation period and the safe period, minimizing impacts on performance. Notice that IR can be implemented by reinforcing recovery capabilities of blocks in SSDs through vertical striping as the safe level expires.

The data in PBN i may be updated after parities are written. Recall that flash memory uses out-of-place updates and that invalid data is retained until garbage collected. Likewise, if data in a vertical stripe is updated, new data is written to new locations while old data and parities are retained, in PBN i in this example, until they are garbage collected. Hence, the old data and parities, though invalid, may still contribute in recovering the remaining valid data within the vertical stripe if ECC-uncorrectable bit errors happen to occur among the valid pages.

Though the safe period is extended by virtue of parities, the data error rate will reach the safe level again after long retention period. Then data scrubbing is employed to rewrite the data or garbage collection may be performed for any reason before it. Anyway, in this case, valid data in PBN i's are copied to other locations, PBN k's in Fig. 2(d), and PBN i's become empty blocks that will be used later for data scrubbing or GC operations. Note that parities in PBN i are not copied and just disappear.

Let us now consider the relation between the error rate and dynamic striping. Assume that when the safe period expires data scrubbing or IR is required to protect data from errors. Fig. 3 shows the error rate variations from data writing time to data scrubbing. In the figure, x- and y-axes denote time and error rate, respectively.

When data is written to SSD at time 0, it has the the lowest error rate. As time progresses, the error rate increases due

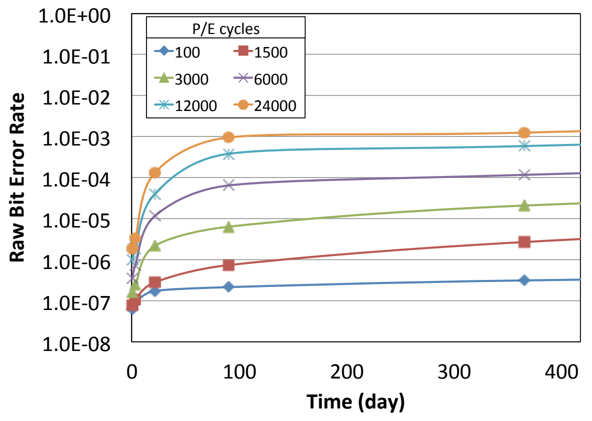


Fig. 4: RBER vs. P/E cycle and retention period

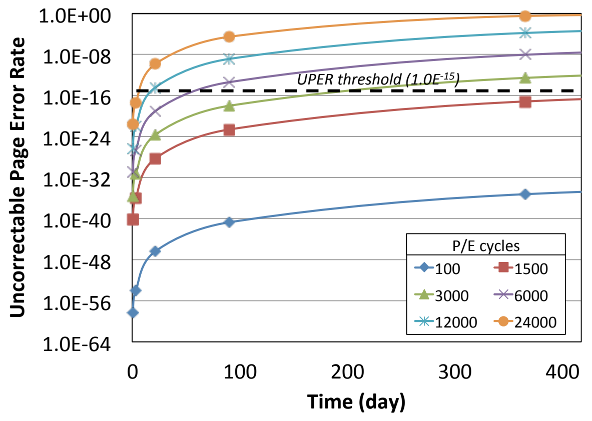


Fig. 5: UPER vs. various P/E cycles and retention period

to data retention errors. We assume that the specific error rate denoted 'A' in Fig. 3 is the safe level, and this level is reached after 200 days. At this moment, the conventional data scrubbing scheme rewrites the data to new locations and the newly written data has again the lowest error rate at time 0. (The lowest error rate at time 0 becomes higher as flash memory chips undergo P/E cycles, but we ignore this effect in this example.) In contrast, the red line denotes the error rate after using IR with parities written at time 200 (See PBN i in Fig. 2(c)). By virtue of the parities, the error rate decreases and the safe period is extended to time 1000. At time 1000, the error rate reaches the safe level again and this time data scrubbing is employed to rewrite the data (See Fig. 2(d)). Naturally, the newly written data has the lowest error rate as at time 0.

Obviously, incremental redundancy incurs parity write overhead. However, this additional parity write is needed only when the error rate reaches the safe level. We will see later that this overhead is a small price to pay for attaining the performance and lifetime benefits through IR. It should also be noted that though parities in stripes can greatly extend the safe period even when the stripe size is large, additional parities or data scrubbing is needed to protect data from errors when even the extended safe period expires.

IV. Error Modeling and Safe Period

We now derive the mathematical analyses for performance and lifetime of data scrubbing and IR. Symbols used in our analyses are listed in TABLE I. To implement vertical striping only one bit per block is needed to keep track of whether each block is a constructed stripe or not. For example, if SSD capacity is 128GB and the block size is 512KB, then the number of blocks is about 262K; thus, additional map management overhead for vertical striping is only 32KB. In our mathematical analyses, we ignore map management overhead as it is relatively small.

Also, reserving pages for vertical striping may increase GC cost as the unused pages need to be reserved until the hot/cold decision is made. However, as our analyses and experimental results show, the space loss can be compensated for by reduced overhead through delayed data scrubbing. As with previous studies, we consider only write requests in this study as read requests have negligible effects on long-term performance and lifetime of SSDs. We start off with modeling errors and deriving the safe period.

Assume that ECC can correct k errors and detect $2k$ bit errors as coding theory suggests. Then, error rate less than $k+1$ bits, that is, the Correctable Page Error Rate (CPER) can be define as follows [8], [12], [14], and with it, the rate of more than k bits failing, that is, the Uncorrectable Page Error Rate (UPER) naturally follows.

$$CPER(n,k) = \sum_{i=0}^{k} \binom{n}{i} \cdot RBER^i \cdot (1-RBER)^{n-i} \quad (1)$$

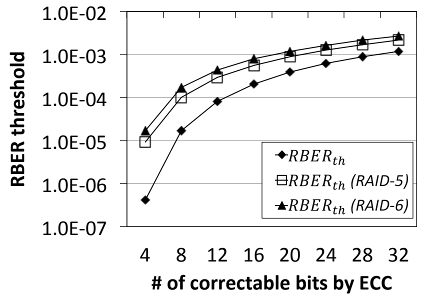
$$UPER(n,k) = 1 - CPER(n,k) \quad (2)$$

Note that RBER (Raw Bit Error Rate) of a page is not a fixed value but a function of two parameters, the block P/E cycle, that is, the P/E cycle state at which the data was written and the data retention period, that is, the length of time since data was written. The actual relation between RBER and the two parameters are inherent to the characteristics of the individual flash memory chips. Hence, the exact relation can only be obtained through observations of these characteristics. For example, we can make use of data such as shown in Fig. 4, which is for a 2-bit MLC NAND flash device manufactured in 3xnm technology, obtained from a previous study that investigated data error rates in relation to P/E cycles and retention period [28].[3] In general, the RBER function $RBER(c,d)$ can be formulated through curve fitting of these data resulting in a formula of the form

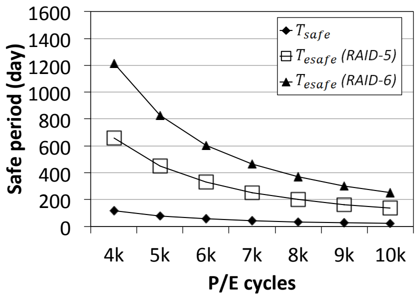
$$RBER(c,d) = d_r(c) \cdot d \quad (3)$$

where $d_r(c)$ is the deterioration rate, that is, the rate at which RBER deteriorates per d (Day), which is a function of parameter c (P/E cycles). In the particular case of Fig. 4, we obtain $d_r(c) = 10^{-13} \cdot c^{1.71}$ through curve fitting. Then, with $RBER(c,d)$, Eqs. 1 and 2 simply become $CPER(n,k,c,d)$ and $UPER(n,k,c,d)$ with the added c and d parameters, assuming that one bit error correction code is employed.

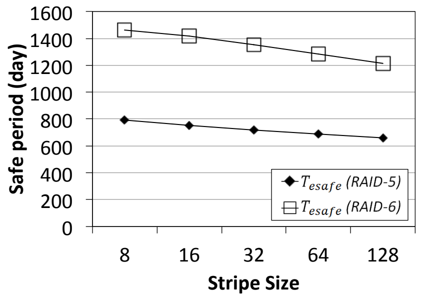
[3]Note that, unless otherwise mentioned, we make use of these data for all examples and formulations for the rest of the paper.



(a) RBER threshold



(b) Safe period and extended safe period



(c) Extended safe period for different stripe size (P/E cycles=4K)

Fig. 6: Result when using different stripe size: 1 (RAID-5) or 2 (RAID-6) parities

Fig. 5 presents UPER functions when P/E cycles are 100, 1,500, 3,000, 6,000, 12,000, and 24,000. In the figure, the dotted horizontal line denotes the error rate of HDD and the x and y axes denote data retention time in days and UPER, respectively. Like RBER in Fig. 4, UPER increases as data retention period and P/E cycles increase.

Now let $UPER_{th}$ denote the threshold UPER, that is, the target safe level. Then, the safe period is simply the number of days RBER stays below this threshold RBER given the deterioration rate $d_r(c)$. Hence, the safe period becomes

$$T_{safe} = \frac{RBER_{th}}{d_r(c)} \tag{4}$$

where $RBER_{th}$ is obtained from Eqs. 1$\sim$3 given $UPER(n,k) = UPER_{th}$.

For the remainder of the paper we will assume the safe level to be the UPER of an HDD. Thus, the *safe period* is the data retention period when UPER is less than or equal to that of the HDD. Then, with the RBER function in Eqs. 3 and 4 with $d_r(c) = 10^{-13} \cdot e^{1.71}$, the safe period is calculated to 192 days when P/E cycles are 3000 and 18 days when P/E cycles are 12,000. These are points where the UPER graph crosses the HDD UPER in Fig. 5 for the respective P/E cycle values. These results suggest that cold data in aged SSDs may become volatile in a couple of months without appropriate data protection measures even though ECC may be employed. In contrast, we observe that when the P/E cycle is relatively small, say 1500, UPER does not reach the threshold level until 629 days, and hence, one can argue that for all practical purposes, the data written will be safe without being concerned with retention errors.

The derivation so far determined the safe period. Typically, at the end of the safe period, that is, at every T_{safe} interval, the action taken is to perform data scrubbing of the cold data. In this paper, we proposed the use of incremental redundancy as an alternative action. We now analyze how much we gain through incremental redundancy. We denote this extra gains in the safe period as the *extended safe period*.

To derive the extended safe period, we need to calculate the UPER after a stripe is formed with the associated parity, which we will denote as $UPER_{str}$. To do this, we start with results from previous studies that derive $UPER_{str}$ of SSDs that employ the RAID architecture [14], [27].

$$UPER_{str}(N) = \frac{1 - CSER_{str}(N)}{N} \tag{5}$$

In the following, we show how we extend this to take into account the c and d parameters. Based on the assumption that ECC can tolerate up to k bit errors and detect $2k$ bit errors, the Detectable Page Error Rate (DPER), that is, the probability that the number of errors is more than k and less than or equal to $2k$ among n bits is as follows.

$$\begin{aligned} DPER(n,k,c,d) = \sum_{j=k+1}^{2k} \binom{n}{j} RBER(c,d)^j \\ \cdot(1 - RBER(c,d))^{n-j} \end{aligned} \tag{6}$$

With Eq. 6, we can derive the Correctable Stripe Error Rate (CSER) as follows, where N is the stripe size and P is the number of parities.

$$\begin{aligned} CSER_{str}(N,P,c,d) = \sum_{j=0}^{P} \binom{N}{j} CPER(n,k,c,d)^{N-j} \\ \cdot DPER(n,k,c,d)^j \end{aligned} \tag{7}$$

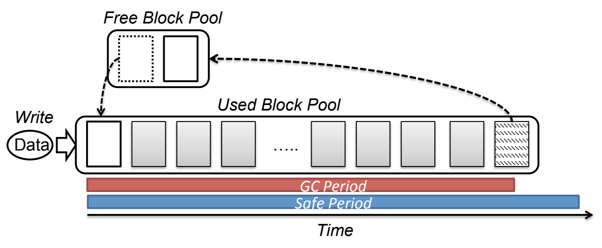
Then, we derive $UPER_{str}(N,P,c,d)$, the Uncorrectable Page Error Rate when striping is constructed, as follows.

$$UPER_{str}(N,P,c,d) = \frac{1 - CSER_{str}(N,P,c,d)}{N} \tag{8}$$

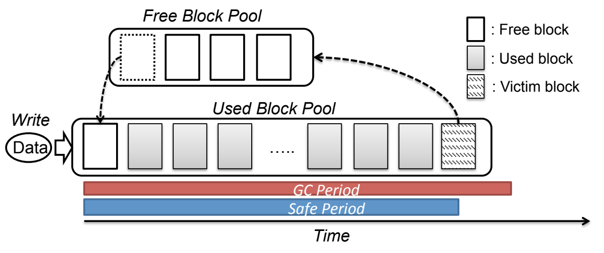
Let $RBER_{th\_str}$ be the raw bit error rate that makes $UPER_{str}$ equal to that of HDD through Eqs. 1$\sim$3 and Eqs. 6$\sim$8. Then, with the given RBER function in Eq. 3, we can calculate the extended safe period, T_{esafe}, as follows.

$$T_{esafe} = \frac{RBER_{th\_str}}{d_r(c)} \tag{9}$$

With the given RBER function, we calculate both $RBER_{th}$ and $RBER_{th\_str}$ for various numbers of correctable bits by ECC, and Fig. 6(a) shows the results. In this figure, $RBER_{th}(RAID-5)$ and $RBER_{th}(RAID-6)$ refers to



(a) GC Period $<$ Safe Period



(b) GC Period $>$ Safe Period

Fig. 7: Two scenarios according to relative size of GC period and safe period

the raw bit error rate that makes $UPER_{str}$ equal to the HDD error rate when RAID-5 and RAID-6 is employed inside the SSD, respectively. As the number of correctable bits by ECC increases, higher raw bit error rate can be tolerated. However, a much higher raw bit error rate can be tolerated if RAID-5 or RAID-6 is employed. On the other hand, without parities as denoted $RBER_{th}$, the raw bit error rate must be much lower than RAID-5 or RAID-6 to keep $UPER_{str}$ below the HDD error rate. This result shows that the RAID scheme effectively lowers the error rate and for ECC only deployment, a much stronger ECC is required to provide reliability comparable to RAID schemes.

Fig. 6(b) shows examples of T_{safe} and T_{esafe} when ECC can correct 8 bit errors. For T_{esafe}, we assume stripe size 128 and one or two parity page(s) for each stripe. (Note that in Fig. 2, we showed the case with only one reserved page which would be needed for RAID-5. We can extend this to two reserved pages for use with RAID-6.) In the figure, we observe that dynamic striping extends the safe period even with one parity per stripe, and even more with two parities per stripe. Also, in Fig. 6(c), we calculate the implication of the stripe size when P/E cycle is 4K. The results show that using RAID-6 compared to RAID-5 can extend the safe period much more than decreasing the stripe size. The conclusion that we make from these results is that incremental redundancy through dynamic striping has the potential to reduce data scrubbing overheads in aged SSDs and this can be done with relatively small parity write overhead.

V. WAF UNDER RANDOM WORKLOAD

As WAF (Write Amplification Factor) determines performance and lifetime of flash memory storage, many studies focus on deriving and reducing WAF of flash memory storage. From now, we derive WAFs of data scrubbing and IR under random workload.

A. Garbage collection period

Before deriving WAF, we derive the GC period, that is, the time span from data write to block reclamation under random workload. In particular, we assume that data are updated randomly with the same probability and that the LRW (Least Recently Written) policy is used to select victim blocks for GC [25]. For further derivations of GC period and effective OPS (Over Provisioning Space) size in the next section, we briefly present results of a previous study that derives the utilization of the victim block for GC [37].

We assume that a is the over-provision ratio such that $a = \frac{E}{U}$, where E is the number of over-provisioned extra pages for GC and U is the number of pages holding valid user data. Then, for each update request, the update probability of data is $\frac{1}{U}$ and the survival (not-update) probability is $1 - \frac{1}{U}$. Then, a block is reclaimed (selected as a victim block for GC) after $U(1+a)(1-u)$ update requests. With the above equations, utilization, u, of the victim block selected for GC can be derived as follows:

$$u = (1 - \frac{1}{U})^{U(1+a)(1-u)} \tag{10}$$

Assuming that U is extremely large, u can be transformed by applying the Euler's limit and the Lambert-W function as follows [37], [38]:

$$u = e^{-(1+a)(1-u)} = -\frac{W(-(1+a)e^{-(1+a)})}{1+a} \tag{11}$$

Now that we have utilization u of the victim block selected for GC, let us now derive the GC period. Assume that, under random workload, the same amount of data is updated everyday with update probability p_u. Then, the utilization of a block after t days, u', is such that $u' = (1-p_u)^t$. As the block is reclaimed when u' becomes u in Eq. 11, we replace u' with u and transform the above formula to obtain the GC period T_{GC} in day units as follows:

$$T_{GC} = \frac{log(u)}{log(1-p_u)} \tag{12}$$

Under random workload, GC operations are performed after t days since data were written unless data scrubbing is performed before GC. During the GC operation, $u \cdot n_p$ pages are copied to a newly erased block, where $(1-u) \cdot n_p$ clean pages are used for the write requests that follow.

B. Relation between safe period and GC period

Data scrubbing may be performed before GC operations if the safe period is smaller than the GC period. Technically, the GC operation and data scrubbing are the same in that they copy valid data to new blocks and generate some clean pages. Therefore, GCs are not needed if data scrubbing generates clean pages before it. Likewise, if GCs are performed before data scrubbing, data scrubbing is not needed as data are refreshed during GC operations. Derivation of WAF must consider these situations.

There are two cases that need to be considered according to the relative size of the safe period (T_{safe}) and GC period (T_{GC}). In the first case where the safe period is larger than the GC period as seen in Fig. 7(a), GC operations refresh data before the safe period expires and, naturally, data scrubbing is never performed. Then, WAF in this case is calculated with u in Eq. 11 as follows:

$$WAF_{GC} = \frac{1}{1-u} \tag{13}$$

The other case occurs when the safe period is shorter than the GC period. In this case, data scrubbing refreshes data while generating clean pages and thus, the GC operation is never performed as we see in Fig. 7(b). We should note that, in this case, data scrubbing is performed because the safe period expires, not because there is no clean page for write requests. In other words, the OPS may not be fully utilized because data scrubbing generates free space before free space runs out. Hence, in this case, WAF is irrelevant to u and is determined only by the amount of write requests, D_{write}, and the scrubbed data size, D_{scrub}, as follows:

$$WAF_{scrub} = \frac{D_{write} + D_{scrub}}{D_{write}} \tag{14}$$

In the above equation, the amount of write requests per day, D_{write} is calculated from the update probability per day, p_u, and amount of user data, U, such that $D_{write} = U \cdot p_u$. To calculate D_{scrub}, let $s = 1 - p_u$ and $t = T_{safe}$ for brevity. In the first period, D_{write} data are initially written and $D_{write} \cdot s^t$ data survive after t days and are refreshed by data scrubbing. Consequently, in the next period, D_{write} and $D_{write} \cdot s^t$ are written and, after t days again, $(D_{write} + D_{write}s^t)s^t$ data survive and are refreshed. Continuing in this manner, after n safe period iterations, we can derive $D_{scrub}(n)$, the size of refreshed data after n safe periods, as follows:

$$\begin{aligned} D_{scrub}(1) &= D_{write} \cdot s^t \\ D_{scrub}(2) &= (D_{write} + D_{write}s^t)s^t = D_{write}s^t + D_{write}s^{2t} \\ &\dots \\ D_{scrub}(n) &= D_{write} \cdot s^t + D_{write} \cdot s^{2t} + ... + D_{write} \cdot s^{nt} \end{aligned} \tag{15}$$

As $D_{scrub}(n)$ is a form of geometric series, we can obtain the sum of an infinite geometric series as follows:

$$\begin{aligned} D_{scrub} &= \lim_{n\to\infty} \sum_{j=0}^{n} D_{write}s^t * s^{jt} \\ &= \frac{D_{write} \cdot s^t}{1-s^t} \end{aligned} \tag{16}$$

Through Eqs. 14 and 16, we can calculate WAF_{scrub}. As we will see later, the difference between WAF_{GC} and WAF_{scrub} can make a big difference in terms of performance and lifetime. We emphasize again that, if the safe period is shorter than the GC period, some OPS is not fully utilized and is wasted. In the next section, we will calculate the *effective OPS size* for the data scrubbing case. Our analyses and experimental results show that OPS larger than the effective OPS size is wasted.

C. Relation between extended safe period and GC period

Now we investigate the benefits of IR that extends the safe period. Again, we have two cases according to the relative size of the GC period and the extended safe period. If the extended safe period is larger than the GC period, data scrubbing is never performed as GC operations refresh data before the extended safe period expires. Specifically, the probability of this case occurring is much higher than conventional data scrubbing without IR because the safe period can be significantly extended with parities. However, some extra space is consumed to write parities and thus, WAF in Eq. 13 has to be calculated with the modified u that is again calculated with $a = \frac{E-P}{U}$, where P is the size of the parity space.

Let us now calculate P. If the stripe size is s_{str} and n_{parity} parity page(s) is written for $(s_{str} - n_{parity})$ data pages, then $P = U/(s_{str} - n_{parity})$ if we assume that all data have associated parities. Otherwise, if only some cold data have parities, we can calculate P with the size of cold data having associated parities such that $P = C/(s_{str} - n_{parity})$, where C is the amount of cold data.

Now we discuss the other case where the GC period is larger than the extended safe period. In this case, WAF is as follows, where D_{parity} is the amount of parity writes made for striping.

$$WAF_{scrub} = \frac{D_{write} + D_{scrub} + D_{parity}}{D_{write}} \tag{17}$$

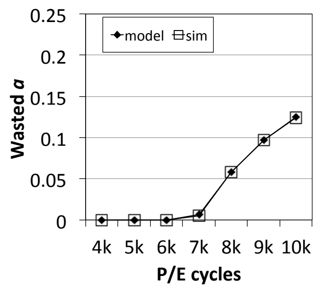
In the above equation, D_{scrub} can be obtained from Eq. 16 by setting $t = T_{esafe}$. Also, D_{parity} is calculated from stripe size, s_{str}, and the number of parities in a stripe, n_{parity}, such that $D_{parity} = D_{write}/(s_{str} - n_{parity})$. Later, through experiments, we show that IR can improve the lifetime of SSDs significantly by extending the safe period with little overhead for parity writes for the workloads that we consider.

VI. Effective OPS size and Hot/Cold separation

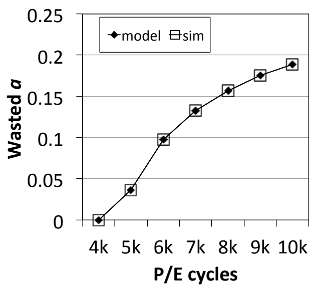
In this section, we extend our analyses to calculate the effective OPS size and to separate hot and cold data. As we mentioned above, OPS is not fully utilized if data scrubbing is performed before GC operations and we can calculate the effective OPS size as well as the space wasted in this case.

A. Effective OPS size

To calculate the effective OPS size, we first calculate the utilization of the block selected for data scrubbing. Let u_{scrub} be the utilization of the block selected for data scrubbing. For each update request, data is updated with probability p_u and, thus, $(1-p_u)^{T_{safe}}$ data survive after T_{safe} under random workload. Consequently, $u_{scrub} = (1-p_u)^{T_{safe}}$ if data scrubbing is performed after T_{safe} and $u_{scrub} = (1-p_u)^{T_{esafe}}$ if it is performed after T_{esafe}. With u_{scrub}, we can derive the



(a) W_{day} = 1%



(b) W_{day}=0.5%

Fig. 8: Wasted over-provision ratio a

effective over-provisioned ratio, $a_{effective}$, through Eq. 11 as follows:

$$a_{effective} = \frac{ln(u_{scrub})}{(1 - u_{scrub})} - 1 \tag{18}$$

Recall that the initial over-provision ratio a is $\frac{E}{U}$. Now, we define the wasted over-provisioned ratio, a_{wasted} as follows:

$$a_{wasted} = a - a_{effective} \tag{19}$$

From Eqs. 18 and 19, we find that the number of effective extra pages is $U \cdot a_{effective}$, and the rest are being wasted. Fig. 8 shows the fraction of wasted OPS when the safe period is shorter than the GC period. In the figure, more OPS area is wasted as P/E cycles increase because the safe period becomes shorter as flash memory ages and thus, the error rate increases. If some OPS is wasted for the given workload, using the wasted OPS for other purposes may be beneficial and we will discuss this in the next section. We also note that extending the safe period may reap performance benefits by allowing control over when GC operations are performed [25].

B. *Hot and cold separation*

We consider a more realistic workload pattern, that is, the mixture of hot and cold references. In particular, real workloads have locality such that a small fraction of data occupies a majority of references. Hence, we divide user space into two areas, namely *hot* and *cold* areas [39]. We assume that the hot area takes s fraction of user space and occupies r ratio of total write requests, while the cold area takes $1 - s$ fraction of the user space and occupies $1 - r$ ratio of the writes. Also, assume that the OPS is divided into two spaces that are allocated separately to the hot and cold areas. Then we consider the two areas to be two independent storage devices, each one serving either hot or cold references. In particular, the hot area has $U \cdot s$ user data pages and $E \cdot o$ extra pages and the cold area has $U \cdot (1 - s)$ user data pages and $E \cdot (1 - o)$ extra pages, where o is the fraction of OPS allocated to the hot area and $1 - o$ is the fraction allocated to the cold area.

Now, for the given hot references bound for the hot area, we can calculate T_{hot_GC}, GC period, and WAF_{hot_GC}, WAF, of the hot area using Eqs. 12 and 13 when OPS is reclaimed by GC operations. We assume that IR is applied to extend the safe period. (In this discussion, we omit the case without IR for brevity and also as it is trivial.) Then, we can calculate T_{hot_scrub}, the extended safe period, and WAF_{hot_scrub}, the WAF in the hot area, using Eqs. 9 and 17 if data scrubbing generates clean pages before GC. Similarly, for the given cold references bound for the cold area, we can calculate T_{cold_GC}, WAF_{cold_GC}, T_{cold_scrub}, and WAF_{cold_scrub} in the cold area.

In a previous study, Desnoyers derived the optimal o that maximizes performance by minimizing the overall WAF of flash memory storage as follows [25]:

$$WAF = r \times WAF_{hot\_GC} + (1 - r) \times WAF_{cold\_GC} \tag{20}$$

Let o_{opt} be the optimal over-provision ratio determined by Desnoyers' solution. Then, the hot and cold area has $E \cdot o_{opt}$ and $E \cdot (1 - o_{opt})$ extra pages, respectively. Let $T^{opt}_{hot\_GC}$ and $T^{opt}_{cold\_GC}$ be the GC period for the hot and cold areas, respectively, calculated with the optimal OPS size for each area. Then, we have to consider four cases according to the relative size between $T^{opt}_{hot\_GC}$ and T_{hot_scrub} and between $T^{opt}_{cold\_GC}$ and T_{cold_scrub}. In the first case where $T^{opt}_{hot\_GC} \leq T_{hot\_scrub}$ and $T^{opt}_{cold\_GC} \leq T_{cold\_scrub}$, data scrubbing is never performed. In other words, the OPS of both the hot and cold areas is reclaimed by GC operations and the overall WAF can be calculated from Eq. 20.

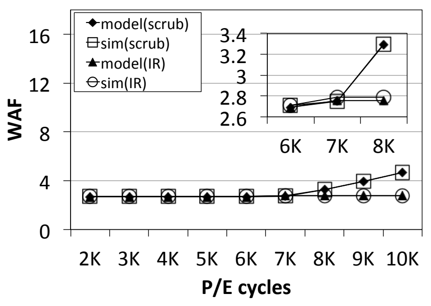
Now, let us consider the second case where $T^{opt}_{hot\_GC} \leq T_{hot\_scrub}$ and $T^{opt}_{cold\_GC} > T_{cold\_scrub}$. In this case, GC operations reclaims the OPS in the hot area, but in the cold area, data scrubbing generates clean pages during data refresh after the extended safe period. Recall that we can calculate the effective over-provision ratio, $a_{effective}$, of the cold area through Eq. 18 and, in turn, the effective OPS size, that is, $U \cdot a_{effective}$. Also, note that extra pages more than the effective size is wasted, but GC operation efficiency generally increases as more OPS is given. Therefore, if the OPS size of the cold area is larger than the effective size, reassigning the excess OPS to the hot area, which we refer to as *OPS tuning* hereafter, would benefit performance. In this case, WAF^{tune}, the WAF obtained after OPS tuning becomes as follows:

$$WAF = r \times WAF^{tune}_{hot\_GC} + (1 - r) \times WAF_{cold\_scrub} \tag{21}$$

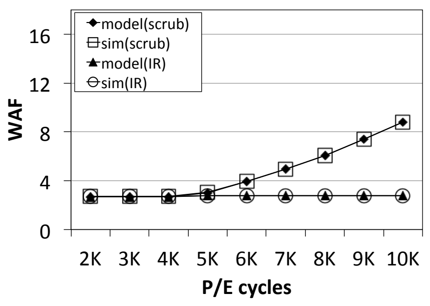
Similarly, we can minimize overall WAF for the third case where $T^{opt}_{hot\_GC} > T_{hot\_scrub}$ and $T^{opt}_{cold\_GC} \leq T_{cold\_scrub}$ by OPS tuning, which reassigns the excess OPS in the cold area to the hot area. This case seems to be unrealistic as, in real systems, hot data tend to be updated faster than cold data. Even so, in this case, some OPS in the hot area is being wasted and thus, reassigning the excess OPS in the hot area to the cold area benefits performance. WAF, in this case, becomes

$$WAF = r \times WAF_{hot\_scrub} + (1 - r) \times WAF^{tune}_{cold\_GC} \tag{22}$$

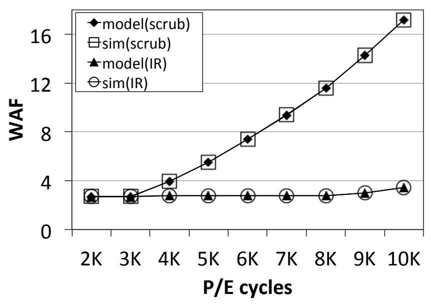
In the last case where $T^{opt}_{hot\_GC} > T_{hot\_scrub}$ and $T^{opt}_{cold\_GC} > T_{cold\_scrub}$, data scrubbing generates clean pages during data refresh in both the hot and cold areas. Therefore,



(a) W_{day} = 1%



(b) W_{day} = 0.5%



(c) W_{day} = 0.25%

Fig. 9: WAF under Random Workload

some OPS is wasted in both areas and the excess OPS may be used for other purposes such as duplicating crucial data in flash memory storage without compromising performance. (Utilizing wasted OPS for other purposes is beyond the scope of this paper.) For this case, WAF is irrelevant to OPS size and becomes

$$WAF = r \times WAF_{hot\_scrub} + (1-r) \times WAF_{cold\_scrub} \quad (23)$$

VII. Evaluation

In this section, through experiments, we validate our mathematical analyses and compare IR with conventional data scrubbing. For the experiments, we use a random workload, denoted Random Workload, and a mixture of hot and cold references that we denote Hot/Cold Workload.

A. Experiment Setup

For the experiments, we implement a simulator that emulates the behaviour of a page-mapping FTL including mapping, GC, and data scrubbing. To validate our simulator, we compare the simulation results with those of a previous study [25] and confirmed that they are almost the same. The total flash memory space is 128GB and we assign 80% of total space to user data space and 20% to OPS. In the experimental results, Write Ratio per Day (W_{day}) refers to the ratio of updated data per day in user data space. For example, if $W_{day} = 1\%$, then 1% of user data is updated every day. Also, we assume that ECC can correct up to 8 bit errors and detect up to 16 bit errors in codeword unit and a page consists of 8 codewords. Both PPB (Pages per Block) and stripe size are set to 128. A stripe comprises 127 data pages and one parity page. In the experiments under Random Workload, the FTL has one write frontier and applies the LRW (Least Recently Written) policy to select the block for GC while, under the Hot/Cold Workload, each of the hot and cold areas has its own write frontier and applies the LRW policy independently. Initially, the optimal OPS size is calculated through Desnoyers' solution and the optimal portion of OPS is given to hot and cold areas under the Hot/Cold Workload.

B. Evaluation with Random Workload

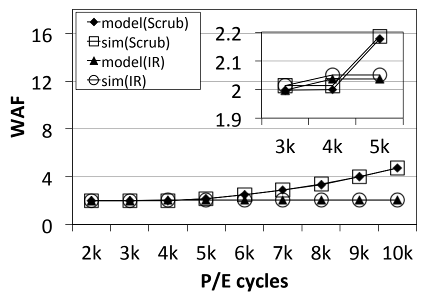
Under Random Workload, we measure WAFs of conventional data scrubbing and IR referred to as "Scrub" and "IR", respectively. In both schemes, OPS is tuned if there is wasted OPS in one area. In Fig. 9, to validate our mathematical analyses, we present values obtained from the analyses and experiments with $W_{day} = 1\%$, 0.5%, and 0.25%. We observe that the results of the analyses and the experiments are very close showing that our mathematical analyses quite accurately estimate WAFs. Now we investigate the effect of W_{day}. With smaller W_{day}, less data are updated every day such that the P/E cycle increases more slowly. Hence, compared to $W_{day} = 1\%$, more days elapse for $W_{day} = 0.25\%$ to reach the same P/E cycle, and ultimately, the safe period expires on lower P/E cycles. For this reason, WAF starts to increase with lower P/E cycles with smaller W_{day} as shown in Fig. 9.

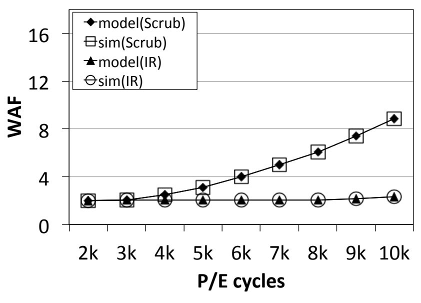
Now, let us compare the results of "Scrub" and "IR". In the early stages of the experiments with small P/E cycles, "IR" and "Scrub" have the same WAF as data scrubbing is not performed. Then, WAFs of both schemes diverges at P/E cycles 7K, 5K, and 3K for $W_{day} = 1\%$, 0.5%, and 0.25%, respectively. From these points on, WAF sharply increases for "Scrub" as data scrubbing occurs. It should be noted that WAF continuously increases as the P/E cycle increases. The reason behind this is that as the P/E cycle increases RBER increases and the safe period becomes shorter, ultimately scrubbing more data every day. Note also that WAF is almost constant for "IR" except when the P/E cycle is 9K$\sim$10K with $W_{day} = 0.25\%$ as data scrubbing is rarely performed due to the extended safe period.

C. Evaluation with Hot/Cold Workload

Figs. 10 and 11 show WAF under the Hot/Cold Workload. We set $s = 0.2$ and $r = 0.8$ for the results of Fig. 10 and $s = 0.1$ and $r = 0.9$ for the results of Fig. 11. The results in the figures show that the mathematical analyses and experiments are very close, again confirming the accuracy of the mathematical analyses. Also, we observe trends similar to the previous results in that, with smaller W_{day}, the safe period expires with smaller P/E cycles resulting in higher WAF.

In Figs. 10 and 11, WAF of "Scrub" becomes higher than that of "IR" as the P/E cycle increases though they are the same at the beginning of the experiments. In particular, WAF diverges at P/E cycles 5K, 3K, and 2K in Fig. 10(a), (b), and (c), respectively, while it diverges at 3K, 2K, and 2K in Fig. 11(a), (b), and (c), respectively. From these points on,


(a) W_{day} = 1%


(b) W_{day} = 0.5%

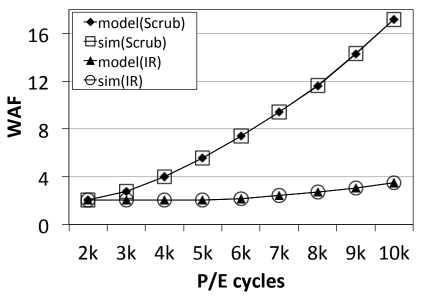
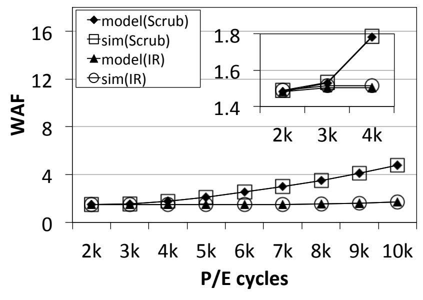
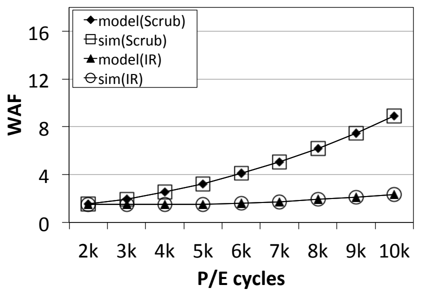

(c) W_{day} = 0.25%

Fig. 10: WAF under Hot/Cold Workload (s=0.2, r=0.8)


(a) W_{day} = 1%


(b) W_{day} = 0.5%

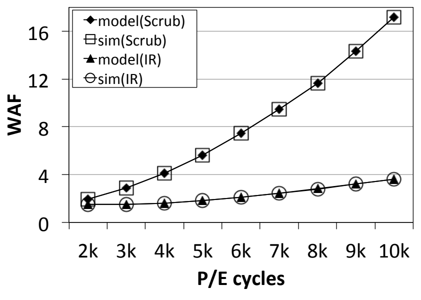

(c) W_{day} = 0.25%

Fig. 11: WAF under Hot/Cold Workload (s=0.2, r=0.8)

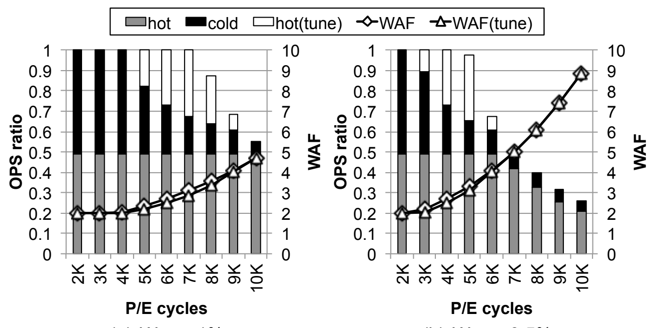

(a) W_{day} = 1% (b) W_{day} = 0.5%

Fig. 12: Effect of OPS tuning (Scrub)

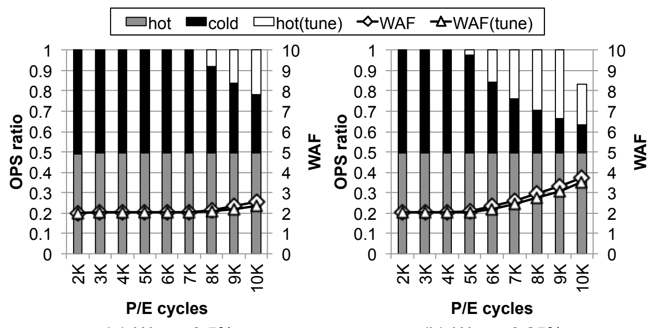

(a) W_{day} = 0.5% (b) W_{day} = 0.25%

Fig. 13: Effect of OPS tuning (IR)

WAF starts to soar for "Scrub", but for "IR" we see only a slight increase as P/E cycle increases. These results show that IR is highly efficient in decreasing WAF while paying only a small price for parity writes.

D. *Effect of OPS tuning*

To investigate the effects of OPS tuning, we measure WAFs of "Scrub" and "IR", each with and without OPS tuning for the Hot/Cold Workload ($s = 0.2$, $r = 0.8$). For the experiments, we set W_{day} to 1% and 0.5% for "Scrub" and 0.5% and 0.25% for "IR". (Note that the safe period of "Scrub" is much shorter than "IR". Hence, we set different W_{day} values for "Scrub" and "IR" as our goal is not to compare the results, but is to observe the effect of OPS tuning.) In the experiments, data scrubbing occurs earlier in the cold area compared to the hot area. Recall that we initially assign the optimal portion of OPS to the hot and cold areas. In Figs. 12 and 13, the vertical bars on the leftmost side consists of shaded and black boxes, which denotes the hot and cold OPS sizes that were initially equally divided to their optimal size. As the P/E cycle increases, some of the black boxes divides into black and white boxes, where the white box is the ratio of OPS reassigned to the hot area by OPS tuning.

In Figs. 12 and 13, the triangles and diamonds linked with a solid line denote the WAFs with and without OPS tuning, respectively. As we can see, WAF falls slightly for both schemes with OPS tuning. Specifically, in the case of "Scrub", WAF is lowered by OPS tuning when the P/E cycles are 5K$\sim$9K with $W_{day} = 1\%$ and 3K$\sim$6K with $W_{day} = 0.5\%$. In the case of "IR", OPS tuning lowers WAF when P/E cycles are 8K$\sim$10K with $W_{day} = 0.5\%$ and 6K$\sim$10K with $W_{day} = 0.25\%$. These experimental results confirm what the mathematical analyses predict, that is, optimally assigned OPS may be wasted when data scrubbing occurs and WAF can be lowered further by reassigning wasted OPS.

Interestingly, the total OPS ratio is less than 1 when P/E

Workload	Function	Read (GB)		Write (GB)		s	r	Avg. W_{day}
		Total	Working set	Total	Working set			
$hm0$	Hardware monitoring	10.99	1.86	22.86	1.63	0.014	0.990	3.19%
$mds0$	Media server	3.30	2.94	7.79	0.33	0.002	0.986	1.09%
$mds1$	Media server	87.20	83.84	1.55	0.60	0.002	0.824	0.22%
$rsrch0$	Research projects	1.39	0.08	11.02	0.29	0.002	0.991	1.54%
$stg0$	Web staging	7.39	6.14	15.78	0.39	0.004	0.991	2.20%
$stg1$	Web staging	79.70	79.60	6.00	0.40	0.002	0.979	0.84%
$wdev0$	Test web server	2.76	0.20	7.36	0.34	0.003	0.991	1.03%
$web0$	Web/SQL server	17.45	7.19	12.16	0.70	0.006	0.992	1.70%

TABLE II: Parameters of real workload

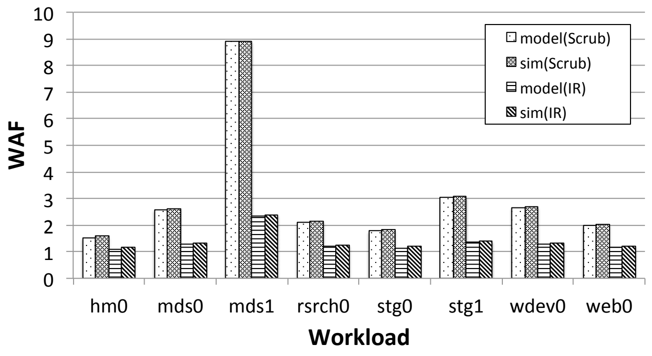


Fig. 14: WAF in Real Workload under Hot/Cold separation

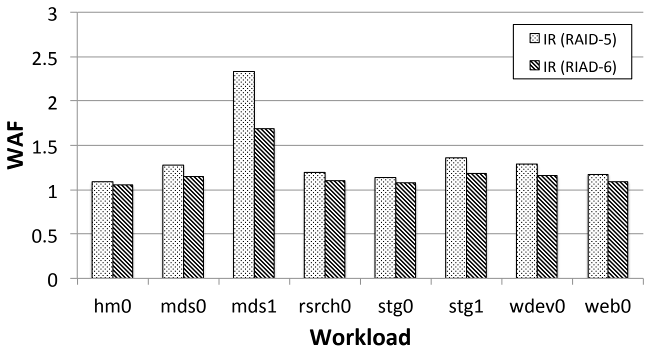


Fig. 15: WAF of IR with one parity per stripe (RAID-5) and two parties per stripe (RAID-6)

cycles are 8K$\sim$10K and 5K$\sim$10K in Fig. 12(a) and (b), respectively, and 10K in Fig 13(b). Our in-depth analyses reveal that, with more OPS reassigned to the hot area, the GC period becomes longer in the hot area and, eventually, exceeds the safe period. Then, data scrubbing starts to occur and some OPS in the hot area are wasted, resulting in the total OPS ratio being smaller than 1. Moreover, with P/E cycles 7K$\sim$10K in Fig. 12(b), even the initial hot area assigned OPS is not fully utilized and is wasted as the gray box shorter than 0.5 reveals. In this situation, utilizing wasted OPS for other purposes may be considered.

E. Evaluation with Real Workloads

In this section, we evaluate IR and data scrubbing with real workload traces, namely *hm, mds, rsrch, stg, wdev* and *web* [40]. As the time span of requests in these traces is less than one week, we run the traces a few hundred times to observe data scrubbing for cold data. Data referenced more than once in each trace iteration are regarded as hot while the others are cold. To expedite the experiments, we assume that initially flash memory has 6K P/E cycles. Through preprocessing, we extract parameters r, s, and W_{day} for each trace and the results of mathematical analyses are obtained with these parameters as listed in Table II. Fig. 14 shows WAFs of "Scrub" and "IR", each with results of experiments and analyses.

The results in the figure show that the WAFs of the experiments and analyses are similar, though the WAFs of the experiments are slightly higher than the ones from the analyses. The reason behind this is that blocks with many valid data are sometimes garbage collected in real traces as we apply LRW, and not the Greedy policy, to select blocks for GC. In contrast, the mathematical analyses assumes that data in the blocks are invalidated at a constant rate with time and thus, it expects only a small number of valid data in the LRW block selected for GC. For all traces, "IR" has much smaller WAFs than "Scrub" as "IR" extends the safe period, ultimately reducing the data scrubbing cost. In particular, for *mds1*, which has a relatively large number of requests to cold data, "IR" has a much smaller WAF than "Scrub". This is because, through extending safe period, "IR" provides cold data more opportunities to be refreshed by write requests, while "Scrub" refreshes many cold data through data scrubbing when the safe period expires.

We now compare the WAFs of "IR" with one parity and two parities per stripe denoted as "IR (RAID-5)" and "IR (RAID-6)", respectively, in Fig. 15. For brevity, we only present the results of the analyses, but the experimental results are also similar. The results show that "IR (RAID-6)" has lower WAF than "IR (RAID-5)" as two parities per stripe extends the safe period further than the single parity case, resulting in lower data scrubbing overhead. This is despite the fact that "IR (RAID-6)" reserves one more page for parity and has one more parity write overhead than "IR (RAID-5)". This shows that even space reserved for the two parities and the extra write overhead is clearly compensated for by the reduced data scrubbing overhead, confirming that IR is a promising solution to overcome the data retention problem.

VIII. Conclusions

As multi-bit cell flash memory are widely used in SSDs, reducing retention errors is a serious issue that must be handled with urgency. To handle this issue, we propose and analyse the use of incremental redundancy (IR) that extends the data scrubbing period by paying a small overhead for striping. In this paper, we developed and presented mathematical analyses to compare and estimate the performance of conventional data scrubbing and IR. We validated the accuracy of our mathematical analyses through experiments and also compared the performance and lifetime of IR with conventional data scrubbing. Also, from the mathematical analyses and experimental results, we observed significant performance and lifetime benefits with IR. This study suggests that IR can be a promising solution to overcome data retention errors of contemporary multi-bit cell flash memory.

Acknowledgment

We would like to thank our shepherd Xiaosong Ma and the anonymous reviewers for their detailed comments that helped to improve the presentation of the paper. We also thank Yongseok Oh for providing useful comments. This research was supported in part by Seoul Creative Human Development Program funded by Seoul Metropolitan Government(No. HM120006), by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2012R1A2A2A01045733), and by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0025282).

References

[1] "Samsung Releases TLC NAND Based 840 SSD," http://www.anandtech.com/show/6329/samsung-releases-tlc-nand-based-840-ssd.

[2] L. M. Grupp, J. D. Davis, and S. Swanson, "The Bleak Future of NAND Flash Memory," in *Proc. USENIX FAST '12*, 2012.

[3] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-based Storage Systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 36–42, 2008.

[4] H. Kwon, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Janus-FTL: Finding the Optimal Point on the Spectrum between Page and Block Mapping Schemes," in *Proc. EMSOFT '10*, 2010.

[5] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," in *Proc. ASPLOS '09*, 2009.

[6] "OCZ TECHNOLOGY LAUNCHES NEXT GENERATION INDILINX EVEREST SSD CONTROLLER PLATFORM," http://www.ocztechnology.com/aboutocz/press/2012/491.

[7] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing Flash Memory: Anomalies, Observations, and Applications," in *Proc. MICRO 42*, 2009.

[8] N. Mielke, T. Marquar, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill, "Bit Error Rate in NAND Flash Memories," in *Proc. IEEE Int'l Reliability Physics Symp.*, 2008.

[9] H. Sun, P. Grayson, and B. Wood, "Quantifying Reliability of Solid-State Storage from Multiple Aspects," in *Proc. SNAPI '11*, 2011.

[10] S. Chen, "What Types of ECC Should be Used on Flash Memory?" http://www.spansion.com/Support/AppNotes/, 2007.

[11] E. Deal, "Trends in NAND Flash Memory Error Correction," http://www.cyclicdesign.com/whitepapers/Cyclic_Design_NAND_ECC.pdf, Cyclic Design, White Paper, 2009.

[12] Y. Wang, L. A. D. Bathen, N. D. Dutt, and Z. Shao, "Meta-Cure: a Reliability Enhancement Strategy for Metadata in NAND Flash Memory Storage Systems," in *Proc. DAC '12*, 2012.

[13] S. Im and D. Shin, "Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 80–92, 2011.

[14] S. Lee, B. Lee, K. Koh, and H. Bahn, "A Lifespan-Aware Reliability Scheme for RAID-based Flash Storage," in *Proc. SAC '11*, 2011.

[15] Y. Lee, S. Jung, and Y. H. Song, "FRA: A Flash-Aware Redundancy Array of Flash Storage Devices," in *Proc. CODES+ISSS '09*, 2009.

[16] M. Blaum, J. L. Hafner, and S. Hetzler, "Partial-MDS Codes and their Application to RAID type of Architectures," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4510–4519, 2013.

[17] K. M. Greenan, D. D. Long, E. L. Miller, T. J. Schwarz, and A. Wildani, "Building flexible, fault-tolerant flash-based storage systems," in *Proc. HotDep '09*, 2009.

[18] X. Jimenez, D. Novo, and P. Ienne, "Phoenix: Reviving MLC Blocks as SLC to Extend NAND Flash Devices Lifetime," in *Proc. DATE '13*, 2013.

[19] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen, "DuraCache: A Durable SSD Cache Using MLC NAND Flash," in *Proc. DAC '13*, 2013.

[20] L.-P. Chang, "Hybrid Solid-State Disks: Combining Heterogeneous NAND Flash in Large SSDs," in *Proc. ASP-DAC '08*, 2008.

[21] S. Im and D. Shin, "ComboFTL: Improving Performance and Lifespan of MLC Flash Memory using SLC Flash Buffer," *Journal of Systems Architecture*, vol. 56, no. 12, pp. 641–653, 2010.

[22] J.-W. Park, S.-H. Park, C. C. Weems, and S.-D. Kim, "A Hybrid Flash Translation Layer Design for SLC-MLC Flash Memory Based Multi-bank Solid State Disk," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 48–59, 2011.

[23] W. Wang, Y. Zhao, and R. Bunt, "HyLog: A High Performance Approach to Managing Disk Layout," in *Proc. USENIX FAST '04*, 2004.

[24] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write Amplification Analysis in Flash-based Solid State Drives," in *Proc. SYSTOR '09*, 2009.

[25] P. Desnoyers, "Analytic Modeling of SSD Write Performance," in *Proc. SYSTOR '12*, 2012.

[26] Y. Oh, J. Choi, D. Lee, and S. H. Noh, "Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems," in *Proc. USENIX FAST '12*, 2012.

[27] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Improving SSD Reliability with RAID via Elastic Striping and Anywhere Parity," in *Proc. DSN '13*, 2013.

[28] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *Proc. DATE '12*, 2012.

[29] T. J. E. Schwarz, Q. Xin, E. L. Miller, D. D. E. Long, A. Hospodor, and S. Ng, "Disk Scrubbing in Large Archival Storage Systems," in *Proc. MASCOTS '04*, 2004.

[30] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash Correct-and-Refresh: Retention-aware Error Management for Increased Flash Memory Lifetime," in *Proc. ICCD '12*, 2012.

[31] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile ssd: Trading flash memory nonvolatility to improve storage system performance for enterprise applications," in *Proc. HPCA '12*, 2012.

[32] V. Mohan, S. Sankar, S. Gurumurthi, and W. Redmond, "reFresh SSDs: Enabling High Endurance, Low Cost Flash in Datacenters," *University of Virginia, Technical Report, CS-2012-05*, 2012.

[33] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing NAND Flash-based SSDs via Retention Relaxation," in *Proc. USENIX FAST '12*, 2012.

[34] M. Awasthi, M. Shevgoor, K. Sudan, B. Rajendran, R. Balasubramonian, and V. Srinivasan, "Efficient Scrub Mechanisms for Error-prone Emerging Memories," in *Proc. HPCA '12*, 2012.

[35] R.-S. Liu, D.-Y. Shen, C.-L. Yang, S.-C. Yu, and C.-Y. M. Wang, "NVM Duet: Unified Working Memory and Persistent Store Architecture," in *Proc. ASPLOS '14*, 2014.

[36] J.-W. Hsieh, L.-P. Chang, and T.-W. Kuo, "Efficient On-line Identification of Hot Data for Flash-memory Management," in *Proc. SAC '05*, 2005.

[37] R. Stoica and A. Ailamaki, "Improving Flash Write Performance by Using Update Frequency," *VLDB Endow.*, vol. 6, no. 9, pp. 733–744, 2013.

[38] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the LambertW function," *Advances in Computational Mathematics*, vol. 5, no. 1, pp. 329–359, 1996.

[39] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26–52, 1992.

[40] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage*, vol. 4, no. 3, p. 10, 2008.