An Analysis of Flash Page Reuse with WOM Codes

GALA YADGAR, Computer Science Department, Technion
EITAN YAAKOBI, Computer Science Department, Technion
FABIO MARGAGLIA, Pure Storage
YUE LI, California Institute of Technology
ALEXANDER YUCOVICH, Computer Science Department, Technion
NACHUM BUNDAK, Computer Science Department, Technion
LIOR GILON, Computer Science Department, Technion
NIR YAKOVI, Computer Science Department, Technion
ASSAF SCHUSTER, Computer Science Department, Technion
ANDRÉ BRINKMANN, Johannes Gutenberg-Universität Mainz

Flash memory is prevalent in modern servers and devices. Coupled with the scaling down of flash technology, the popularity of flash memory motivates the search for methods to increase flash reliability and lifetime. Erasures are the dominant cause of flash cell wear, but reducing them is challenging because flash is a *write-once* medium— memory cells must be erased prior to writing.

An approach that has recently received considerable attention relies on *write-once memory (WOM)* codes, designed to accommodate additional writes on write-once media. However, the techniques proposed for reusing flash pages with WOM codes are limited in their scope. Many focus on the coding theory alone, while others suggest FTL designs that are application specific, or not applicable due to their complexity, overheads, or specific constraints of MLC flash.

This work is the first that addresses all aspects of page reuse within an end-to-end analysis of a general-purpose FTL on MLC flash. We use a hardware evaluation setup to directly measure the short and long-term effects of page reuse on SSD durability and energy consumption, and show that FTL design must explicitly take them into account. We then provide a detailed analytical model for deriving the optimal garbage collection policy for such FTL designs, and for predicting the benefit from reuse on realistic hardware and workload characteristics.

CCS Concepts: •**Information systems** → **Flash memory;** •**Hardware** → **Aging of circuits and systems;** Memory and dense storage; •**Software and its engineering** → **Secondary storage;**

Additional Key Words and Phrases: WOM codes, NAND flash, flash translation layer, SSD, offline analysis

ACM Reference Format:
Gala Yadgar, Eitan Yaakobi, Fabio Margaglia, Yue Li, Alexander Yucovich, Nachum Bundak, Lior Gilon, Nir Yakovi, Assaf Schuster, André Brinkmann, 2017. An Analysis of Flash Page Reuse with WOM Codes. *ACM Trans. Storage* 0, 0, Article 0 (2017), 36 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Flash memories have special characteristics that make them especially useful for solid-state drives (SSD). Their short read and write latencies and increasing throughput provide a great performance improvement compared to traditional hard-disk based drives. However, once a flash cell is written upon, changing its value from 1 to 0, it must be erased before it can be rewritten. In addition to the

Author's addresses: G. Yadgar, Computer Science Department, Technion; E. Yaakobi, Computer Science Department, Technion; F. Margaglia, Pure Storage; Y. Li, California Institute Of Technology; A. Yucovich, Computer Science Department, Technion; N. Bundak, Computer Science Department, Technion; L. Gilon, Computer Science Department, Technion; N. Yakovi, Computer Science Department, Technion; A. Schuster, Computer Science Department, Technion; Brinkmann, Johannes Gutenberg-Universität Mainz;.

DOI: 0000001.0000001

latency they incur, these erasures wear the cells, degrading their reliability. Thus, flash cells have a limited lifetime, measured as the number of erasures a block can endure.

Multi-level flash cells (MLC), which support four voltage levels, increase available capacity but have especially short lifetimes, as low as several thousands of erasures. Many methods for reducing block erasures have been suggested for incorporation in the flash translation layer (FTL), the SSD management firmware. These include minimizing user and internal write traffic [Colgrove et al. 2015; Gupta et al. 2011; Huang et al. 2013; Kim and Ahn 2008; Oh et al. 2012; Park et al. 2015; Saxena et al. 2012; Soundararajan et al. 2010; Yang et al. 2013] and distributing erasure costs evenly across the drive's blocks [Agrawal et al. 2008; Im and Shin 2010; Jimenez et al. 2014; Kgil et al. 2008].

A promising technique for reducing block erasures is to use write-once memory (WOM) codes. WOM codes alter the logical data before it is physically written, thus allowing the reuse of cells for multiple writes. They ensure that, on every consecutive write, ones may be overwritten with zeros, but not vice versa. Reusing flash cells with this technique might make it possible to increase the amount of data written to the block before it must be erased.

Flash page reuse is appealing because it is orthogonal to other FTL optimizations. Indeed, the design of WOM codes and systems that use them has received much attention in recent years. While the coding theory community focuses on optimizing these codes to reduce their redundancy and complexity [Burshtein 2015; Burshtein and Strugatski 2013; Cohen et al. 1986; En Gad et al. 2015; Shpilka 2014; Yaakobi et al. 2012b], the storage community focuses on SSD designs that can offset these overheads and be applied to real systems [Jagmohan et al. 2010; Odeh and Cassuto 2014; Yadgar et al. 2015b].

However, the application of WOM codes to state-of-the-art flash chips is not straightforward. MLC chips impose additional constraints on modifying their voltage levels. Previous studies that examined page reuse on real hardware identified some limitations on reprogramming MLC flash, and thus resort to page reuse only on SLC flash [Jagmohan et al. 2010], outside an SSD framework [Grupp et al. 2009], or within a limited special-purpose FTL [Margaglia and Brinkmann 2015].

Thus, previous SSD designs that utilize WOM codes have not been implemented on real platforms, and their benefits were analyzed by simulation alone, raising the concern that they could not be achieved in real world storage systems. In particular, hardware aspects such as possible increase in cell wear and energy consumption due to the additional writes and higher resulting voltage levels have not been examined before, but may have dramatic implications on the applicability of this approach.

In this study, we extend the end-to-end evaluation and analysis of flash page reuse from our preliminary study [Margaglia et al. 2016]. The first part of our analysis consists of a low-level evaluation of five state-of-the-art MLC flash chips. We examine the possibility of several reprogramming schemes for MLC flash and their short and long-term effects on the chip's durability, as well as the difference in energy consumption compared to that of traditional use. We present two alternative FTL designs that take into account the limitations identified in the low-level analysis and can potentially be implemented on real hardware. Namely, LLH-FTL reuses 50% of each block, but requires reserving some physical capacity on about-to-be-reused blocks. $\overline{\text{LHH}}$-FTL does not require such reservation, but reuses only 25% of each block.

In the second part of our analysis, we present a theoretical framework for optimizing flash page reuse with WOM codes and for calculating the expected benefit from reprogramming. This theoretical model is a generalization of the basic model from our preliminary study [Yaakobi et al. 2015], and takes into account the physical limitations on page reuse as well as the (possibly non-uniform) distribution of write requests. From this model, we derive an optimal garbage collection policy, as well as the expected number of block erasures and its reduction due to reprogramming. Our validation on representative real-world and synthetic workloads shows that this model predicts the benefit from reprogramming with much higher accuracy than previous analyses [Odeh and Cassuto 2014; Yadgar et al. 2015b].

The rest of this paper is organized as follows. Section 2 describes the basic concepts that determine to what extent it is possible to benefit from flash page reuse. We identify the limitations on page reuse in MLC flash in Section 3, with the implications on FTL design in Section 4. We present our theoretical framework in Section 5, and present its validation in Section 6. We survey related work in Section 7, and conclude in Section 8.

2. PRELIMINARIES

In this section, we introduce the basic concepts that determine the potential benefit from flash page reuse: WOM codes, MLC flash, and SSD design.

2.1. Write-Once Memory Codes

Write-once memory (WOM) codes were first introduced in 1982 by Rivest and Shamir, for recording information multiple times on a write-once storage medium [Rivest and Shamir 1982]. They give a simple WOM code example, presented in Table I. This code enables the recording of two bits of information in three cells twice, ensuring that in both writes the cells change their value only from 1 to 0. For example, if the first message to be stored is 00, then 110 is written, programming only the last cell. If the second message is 10, then 010 is written, programming the first cell as well. Note that without special encoding, 00 cannot be overwritten by 10 without prior erasure. If the first and second messages are identical, then the cells do not change their value between the first and second writes. Thus, before performing a second write, the cell values must be *read* in order to determine the correct encoding.

Table I. WOM code example

Data bits	1st write	2nd write
11	111	000
01	011	100
10	101	010
00	110	001

WOM code instances, or *constructions*, differ in the number of achievable writes and in the manner in which each successive write is encoded. The applicability of a WOM code construction to storage depends on three characteristics: (a) the *capacity overhead* —the number of extra cells required to encode the original message, (b) the encoding and decoding *efficiency*, and (c) the *success rate*—the probability of producing an encoded output that can be used for overwriting the chosen cells. Any two of these characteristics can be optimized at the cost of compromising the third [Shpilka 2013; Yaakobi et al. 2012b; Burshtein and Strugatski 2013].

Consider, for example, the code depicted in Table I, where encoding and decoding are done by a simple table lookup, and therefore have complexity $O(1)$ and a success rate of 100%. However, this code incurs a capacity overhead of 50% on each write. This means that (1) only $\frac{2}{3}$ of the overall physical capacity can be utilized for logical data, and (2) every read and write must access 50% more cells than what is required by the logical data size.

The theoretical lower bound on capacity overhead for two writes is 29% [Rivest and Shamir 1982]. Codes that incur this minimal overhead (*capacity achieving*) are not suitable for real systems. They either have exponential and thus inapplicable complexity, or complexity of $n \log n$ (where n is the number of encoded bits) but a failure rate that approaches 1 [Burshtein and Strugatski 2013; En Gad et al. 2015]. Thus, early proposals for rewriting flash pages using WOM codes that were based on capacity achieving codes were impractical. In addition, they required partially programming additional pages on each write, modifying the physical page size [Berman and Birk 2013; Grupp et al. 2009; Jacobvitz et al. 2012; Luojie et al. 2012; Odeh and Cassuto 2014; Yaakobi et al. 2010], or compressing the logical data prior to encoding [Jagmohan et al. 2010].

Two recently suggested WOM code families, Polar [Burshtein 2015; Burshtein and Strugatski 2013] and LDPC [En Gad et al. 2015], have the same complexities as the error correction codes they are derived from. For these complexities, different constructions incur different capacity overheads, and the failure rate decreases as the capacity overhead increases. Of particular interest are constructions in which the overhead of the first write is 0, i.e., one logical page is written on one physical page. The data encoded for the second write requires two full physical pages for one logical

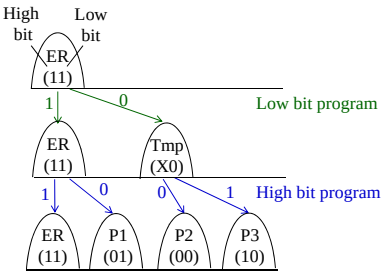

Fig. 1. Normal programming order and states of MLC flash. ER is the initial (erased) state.

page. Such a construction is used in the design of ReusableSSD [Yadgar et al. 2015b], where the second write is performed by programming pages containing invalid data on two different blocks in parallel.

2.2. Multi-Level Cell (MLC) Flash

A flash chip is built from floating-gate cells whose state depends on the number of electrons they retain. Writing is done by *programming* the cell, increasing the *threshold voltage (V_{th})* required to activate it. Cells are organized in blocks, which are the unit of erasure. Blocks are further divided into pages, which are the read and program units.

Single-level cells (SLC) support two voltage levels, mapped to either 1 (in the initial state) or 0. Thus, SLC flash is a classic write-once memory, where pages can be reused by programming some of their 1's to 0's. We refer to programming without prior erasure as *reprogramming*. Multi-level cells (MLC) support four voltage levels, mapped to 11 (in the initial state), 01, 00 or 10. This mapping, in which a single bit is flipped between successive states, minimizes bit errors if the cell's voltage level is disturbed. The least and most significant bits represented by the voltage levels of a multi-level cell are mapped to two separate pages, the *low page* and *high page*, respectively. These pages can be programmed and read independently. However, programming must be done in a certain order to ensure that all possible bit combinations can be read correctly. Triple-level cells (TLC) support eight voltage levels, and can thus store three bits. Their mapping schemes and programming constraints are similar to those of MLC flash. We focus our discussion on MLC flash, which is the most common technology in SSDs today.

Figure 1 depicts a normal programming order of the low and high bits in a multi-level cell. The cell's initial state is the erased (ER) state corresponding to 11. The low bit is programmed first: programming 1 leaves the cell in the erased state, while programming 0 raises its level and moves it to a temporary state.[1] Programming the high bit changes the cell's state according to the state it was in after the low bit was programmed, as shown in the bottom part of the figure. We discuss the implications of this mapping scheme on page reuse in the following section.

Bit errors occur when the state of the cell changes unintentionally, causing a bit value to flip. The reliability of a flash block is measured by its *bit error rate (BER)*—the average number of bit errors per page. The high voltage applied to flash cells during repeated program and erase operations gradually degrades their ability to retain the applied voltage level. This causes the BER to increase as the block approaches the end of its lifetime, which is measured in program/erase (P/E) cycles.

Bit errors in MLC flash are due mainly to *retention errors* and *program disturbance* [Cai et al. 2013]. Retention errors occur when the cell's voltage level gradually decreases below the boundaries of the state it was programmed to. Program disturbance occurs when a cell's state is altered during programming of cells in a neighboring page. In the following section, we discuss how program disturbance limits MLC page reuse, and evaluate the effects of reusing a block's pages on its BER.

[1] Partially programming the high bit in the temporary state is designed to reduce program disturbance.

Table II. Evaluated flash chip characteristics

	A16	A27	B16	B29	C19	D35
Feature size	16nm	27nm	16nm	29nm	19nm	35nm
Page size	16KB	8KB	16KB	4KB	16KB	8KB
Pages per block	256	256	512	256	256	128
Spare area (%)	10.15	7.81	11.42	5.47	7.81	3.12
Lifetime (T)	3K	5K	10K	10K	3K	NA

A, B, C and D represent different manufacturers. The D35 chip was examined in a previous study, and is included here for completeness.

Error correction codes (ECC) are used to correct some of the errors described above. The redundant bits of the ECC are stored in each page's *spare area*. The number of bit errors an ECC can correct increases with the number of redundant bits, chosen according to the expected BER at the end of a block's lifetime [Zhao et al. 2013].

Write requests cannot update the data in the same place it is stored, because the pages must first be erased. Thus, writes are performed *out-of-place*: the previous data location is marked as invalid, and the data is written again on a clean page. The *flash translation layer (FTL)* is the SSD firmware component responsible for mapping logical addresses to physical pages. We discuss relevant components of the FTL further in Section 4.

3. FLASH RELIABILITY

Flash chips do not support reprogramming via their standard interfaces. Thus, the implications of reprogramming on the cells' state transitions and durability cannot be derived from standard documentation, and require experimentation with specialized hardware. We performed a series of experiments with several state-of-the-art flash chips to evaluate the limitations on reprogramming MLC flash pages and the implications of reprogramming on the chip's lifetime, reliability, and energy consumption.

3.1. Flash Evaluation Setup

We used five NAND flash chips from three manufacturers and various feature sizes, detailed in Table II. We also include in our discussion the observations from a previous study on a chip from a fourth manufacturer [Margaglia and Brinkmann 2015]. Thus, our analysis covers the four existing flash vendors.

Chip datasheets include the expected lifetime of the chip, which is usually the maximal number of P/E cycles that can be performed before the average BER reaches 10^{-3}. However, cycling the chips in a lab setup usually wears the cells faster than normal operation because they program and erase the same block continuously. Thus, the threshold BER is reached after fewer P/E cycles than expected. In our evaluation, we consider the lifetime (T) of the chips as the minimum of the expected number of cycles, and the number required to reach a BER of 10^{-3}.

Our experiments were conducted using the SigNASII commercial NAND flash tester [Sig 2014]. The tester allows software control of the physically programmed flash blocks and pages within them. By disabling the ECC hardware we were able to examine the state of each cell, and to count the bit errors in each page.

Some manufacturers employ *scrambling* within their chip, where a random vector is added to the logical data before it is programmed. Scrambling achieves uniform distribution of the flash cell levels, thus reducing various disturbance effects. In order to control the exact data that is programmed on each page, we bypass the scrambling mechanism on the chips that employ it.

Our evaluation excludes retention errors, which occur when considerable time passes between programming and reading a page. Reprogramming might increase the probability of retention errors because it increases the cell's V_{th}. However, since it is intended primarily for short-lived data, we believe it will not cause additional retention errors.

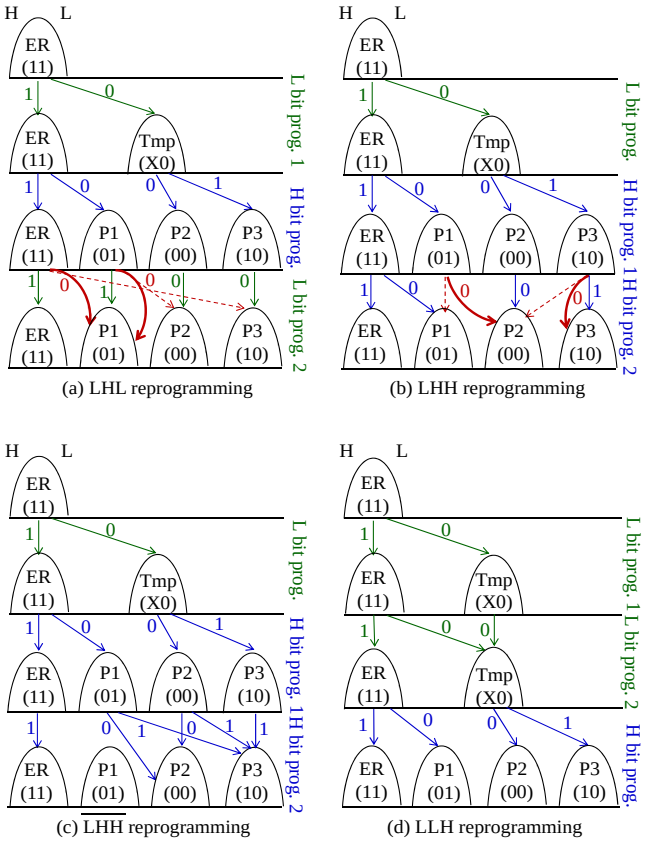

Fig. 2. State transitions in the three reprogramming schemes. A thin arrow represents an attempted transition. A dashed arrow represents a failed transition, with a bold arrow representing the erroneous transition that takes place instead. Only LLH reprogramming achieves all the required transitions for page reuse without program disturbance.

3.2. Limitations on reprogramming

Flash cell reprogramming is strictly limited by the constraint that V_{th} can only increase, unless the block is erased. At the same time, WOM encoding ensures that reprogramming only attempts to change the value of each bit from 1 to 0. However, additional limitations are imposed by the scheme used for mapping voltage levels to bit values, and by the need to avoid additional program disturbance. Thus, page reuse must follow a *reprogramming scheme* which ensures that all reprogrammed cells reach their desired state.

We use our evaluation setup to examine which state transitions are possible in practice. We include all "allowed" transitions, including those that are not expected to change the bit value. The reason is that flash cells are programmed in page granularity, so bits that should not be modified are still reprogrammed with their existing value. We consider four different reprogramming schemes. In the first three schemes, a block is fully programmed before being reprogrammed. In the fourth scheme, only the low pages of the block are initially programmed. Of these schemes, we identify the two that are applicable in a practical FTL design.

Let us assume that the entire block's pages have been programmed before they are reused. Thus, the states of the cells are as depicted in the bottom row of Figure 1. In the *low-high-low (LHL)* reprogramming scheme, depicted in Figure 2(a), we attempt to program the low bit from this state. The thin arrows depict possible desired transitions in this scheme. Two such transitions are impossible, resulting in an undesired state (depicted by the bold arrow). In the *low-high-high (LHH)* reprogram-

ming scheme, depicted in Figure 2(b), the high page is reprogrammed in a fully used block. Here, too, two state transitions fail.

A possible reason for the failed transitions in the LHL scheme is that the voltage applied by the command to program the low bit is not high enough to raise V_{th} from $P1$ to $P2$ and from ER to $P3$.[2] The transition from $P3$ to $P2$ in the LHH scheme is impossible, because it entails decreasing V_{th}. Another problem in the LHH scheme occurs in state $P1$ when we attempt to leave the already programmed high bit untouched. Due to an unknown disturbance, the cell transitions unintentionally to $P2$, corrupting the data on the corresponding low page.

Three of these problematic transitions can probably be made possible with proper manufacturer support—the transition from $P3$ to $P2$ in the LHH scheme would be possible with a different mapping of voltage levels to states, and the two transitions in the LHL scheme could succeed if a higher voltage was applied during reprogramming. While recent technology trends, such as one-shot programming and 3D V-NAND [Im et al. 2015], eliminate some constraints on page programming, applying such architectural changes to existing MLC flash might amplify program disturbance and increase the BER. Thus, they require careful investigation and optimization.

The mapping of voltage levels to two-bit values in multi-level cells implies that in some cases, 0's may be overwritten by 1's. Specifically, a transition from $P1$ or $P2$ to $P3$ changes the high bit from 0 to 1. Motivated by this observation, we examine a third reprogramming scheme that uses a modified WOM encoding, presented in Table III. The initial state of each bit is 1. In the first write, 1's are only overwritten by 0's, according to the original WOM requirement. However, in the second write, 0's are only overwritten by 1's. The resulting code is the complement of the code in Table I. For example, if the first message to be stored is 10, then 100 is written, programming the second and the last cell. If the second message is 01, then 101 is written by reprogramming the last cell to be 1.

Table III. Modified WOM Code

Data bits	1st write	2nd write
00	000	111
10	100	011
01	010	101
11	001	110

Figure 2(c) shows the resulting $\overline{\textit{low-high-high}}$ $\overline{\textit{(LHH)}}$ reprogramming scheme. Its first drawback is that it corrupts the low pages, so a high page can be reused only if the data on the low page is either invalid, or copied elsewhere prior to reprogramming. Such reprogramming also significantly increased the BER of the high pages adjacent to the reprogrammed one. This could be a side effect of the voltage change required for the transition from $P1$ to $P3$, which is higher than that required for transitioning between adjacent states. Thus, this scheme allows safe reprogramming of only one out of two high pages. We address this limitation in our FTL design in Section 4.3.

Interestingly, reprogramming the high bits in chips from manufacturer A returned an error code and did not change their state, regardless of the attempted transition. A possible explanation is that this manufacturer might block reprogramming of the high bit by some internal mechanism to prevent the corruption described above.

The problems with the LHL and LHH schemes motivated the introduction of the *low-low-high (LLH)* reprogramming scheme by Margaglia and Brinkmann [Margaglia and Brinkmann 2015]. Blocks in this scheme are programmed in two rounds. In the first round only the low pages are programmed. The second round takes place after most of the low pages have been invalidated. All the pages in the block are programmed in order, i.e., a low page is reprogrammed and then the corresponding high page is programmed for the first time, before moving on to the next pair of pages.

We validated the applicability of the LLH scheme on the chips of manufacturers A and B. Figure 2(d) depicts the corresponding state transitions of the cells. Since both programming and reprogramming of the low bit leave the cell in either the erased or temporary state, there are no limitations on the programming of the high page in the bottom row. This scheme works well in all the chips we

[2]The transition from ER to $P3$ actually succeeded in the older, D35 chip [Margaglia and Brinkmann 2015]. All other problematic transitions discussed in this section failed in all the chips in Table II.

Table IV. Expected reduction in lifetime due to increased V_{th}

Num. of P_{LLH} cycles	A16	A27	B16	B29	C19
T (= entire lifetime)	32%	29%	20%	30.5%	36%
$0.6 \times T$	8%	9%	8%	9%	15.7%
$0.4 \times T$	6%	6.5%	6%	6.5%	12.7%
$0.2 \times T$	2%	3%	3%	3.5%	14.2%

examined. However, it has the obvious drawback of leaving half of the block's capacity unused in the first round.

The LLH and $\overline{\text{LHH}}$ reprogramming schemes are both applicable on the chips we examined, demonstrating that page reuse in MLC flash is possible. At the same time, both schemes can utilize only half of the pages, each presenting a different tradeoff: the $\overline{\text{LHH}}$ scheme disturbs the high page adjacent to the reprogrammed one, limiting reuse even further, while the LLH scheme requires that some of the block's capacity be reserved in advance. We examine the long term effects of each of these schemes as well as the implications of their specific limitations on FTL design in the following sections.

3.3. Average V_{th} and BER

In analyzing the effects of reprogramming on a chip's durability, we distinguish between *short-term* effects on the BER due to modifications in the current P/E cycle, and *long-term* wear on the cell, which might increase the probability of errors in future cycles. With this distinction, we wish to identify a *safe* portion of the chip's lifetime, during which the resulting BER as well as the long term wear are kept at an acceptable level.

Reprogramming increases the probability that a cell's value is 0. Thus, the average V_{th} of reused pages is higher than that of pages that have only been programmed once. A higher V_{th} increases the probability of a bit error. The short-term effects of increased V_{th} include increased program disturbance and retention errors, which are a direct result of the current V_{th} of the cell and its neighboring cells. The long-term wear is due to the higher voltage applied during programming and erasure.

Our first set of experiments evaluated the short-term effects of increased V_{th} on a block's BER. In each chip, we performed T regular P/E cycles writing random data on one block, where T is the lifetime of the chip as detailed in Table II. We repeated this process with different distributions of 1 and 0. $P_{0.5}$, in which the probability of a bit to be 0 is 0.5, is our baseline. With P_{LLH} the probability of 0 was 0.75 and 0.5 in the low and high page, respectively. This corresponds to the expected probabilities after LLH reprogramming. We read the block's content and recorded the BER after every P/E cycle. We repeated each experiment on six blocks, and calculated the average. In all our experiments, we consider $P_{0.5}$ as our baseline.

The implication of an increase in BER depends on whether it remains within the error correction capabilities of the ECC. A small increase in BER at the end of a block's lifetime might deem it unusable, while a large increase in a 'young' block has little practical effect. For a chip with lifetime T, let BER_T be the BER of after T regular P/E cycles, and let T' be the number of cycles required to reach a BER_T in this experiment. Then $T - T'$ is the *lifetime reduction* caused by increasing V_{th}.

Our results, summarized in Table IV, were consistent in all the chips we examined.[3] Programming with P_{LLH}, which corresponds to a higher average V_{th}, shortened the chips' lifetime considerably, by 20–36%.

In the next set of experiments, we evaluated the long-term effects of V_{th}. Each experiment had two parts: we programmed the block with P_{LLH} in the first part, for a portion of its lifetime, and with $P_{0.5}$ in the second part, which consists of the remaining cycles. Thus, the BER in the second part represents the long-term effect of the biased programming in the first part. We varied the length of the first part between 20%, 40% and 60% of the block's lifetime. Figure 3 shows the BER of

[3]The complete set of graphs for chips from manufacturers A and B in all the experiments described in this section is available in our technical report [Yadgar et al. 2016].

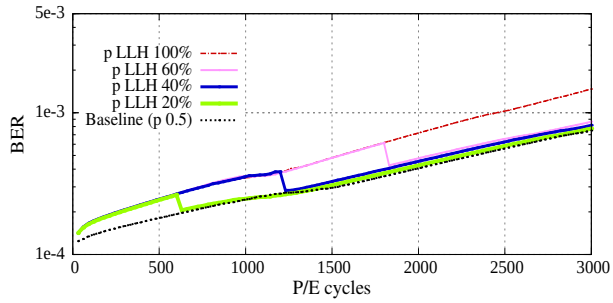

Fig. 3. Effects of increased V_{th} on the A16 chip.

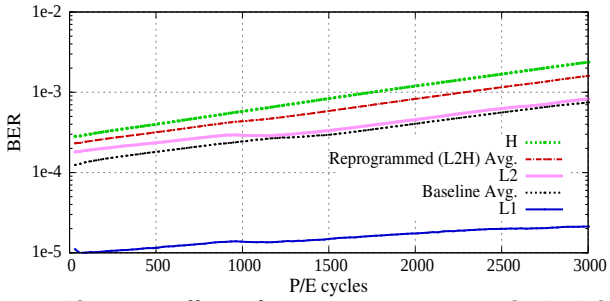

Fig. 4. Short-term effects of LLH reprogramming on the A16 chip.

Table V. Expected reduction in lifetime due to LLH reprogramming

Num. of LLH cycles	A16	A27	B16	B29	C19
T (= entire lifetime)	38%	59.5%	99%	31%	40%
$0.6 \times T$	8.5%	8%	7%	8.5%	22%
$0.4 \times T$	5.2%	6%	5%	5.5%	20%
$0.2 \times T$	1%	2.5%	3%	3%	1%

blocks in the A16 chip (the graphs for the different chips were similar), with the lifetime reduction of the rest of the chips in Table IV.

Our results show that the long-term effect of increasing V_{th} is modest, though nonnegligible—increasing V_{th} early in the block's lifetime shortened it by as much as 3.5%, 6.5% and 9%, with increased V_{th} during 20%, 40% and 60% of the block's lifetime, respectively, for the chips from manufacturers A and B. The lifetime of chip C19 was shortened considerably, by 12%–16%.

3.4. LLH Reprogramming and BER

Reprogramming schemes use blocks differently from regular programming. First, they increases the average V_{th} of reused pages, the effects of which were demonstrated above. Second, they program the block's pages in an order different from the one intended by the manufacturer. To evaluate the weight of each effect, we measure the effects of reprogramming and compare them to the effects of regular programming with increased V_{th}. In the third set of experiments, we measured the effects of reprogramming by performing T LLH reprogramming cycles on blocks in each chip. Figure 4 shows the BER results for the A16 chip, and Table V summarizes the expected lifetime reduction for the remaining chips.

In all the chips, the BER in the first round of programming the low pages (L_1) was extremely low, thanks to the lack of interference from the high pages. In the second round, however, the BER of both low (L_2) and high (H) pages was higher than the baseline, and resulted in a reduction of lifetime

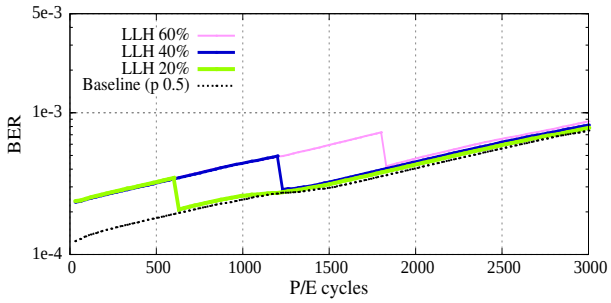

Fig. 5. Long-term effects of LLH reprogramming on the A16 chip.

greater than that caused by increasing V_{th}. We believe that a major cause of this difference are optimizations tailored for the regular LH programming order [Park et al. 2008]. These optimizations are more common in recent chips, such as the B16 chip.

In the fourth set of experiments, we evaluated the long-term effects of LLH reprogramming. Here, too, each experiment was composed of two parts: we programmed the block with LLH reprogramming in the first part, and with $P_{0.5}$ and regular programming in the second part. We varied the length of the first part between 20%, 40% and 60% of the block's lifetime. Figure 5 shows the BER results for the A16 chip, and Table V summarizes the expected lifetime reduction for the remaining chips.

We observe that the long-term effects of reprogramming are modest, and comparable to the long-term effects of increasing V_{th} on each chip. This supports our assumption that the additional short-term increase in BER observed in the previous set of experiments is not a result of the actual reprogramming process, but rather of the mismatch between the programming order the chips are optimized for and the LLH reprogramming scheme. This is especially evident in the B16 chip, in which the BER during the first part was high above the limit of 10^{-3}, but substantially smaller in the second part of the experiment.

Thus, schemes that reuse flash pages only at the beginning of the block's lifetime can increase its utilization without degrading its long-term reliability. Moreover, in all but the B16 chips, LLH reprogramming in the first 40% of the block's lifetime resulted in BER that was well within the error correction capabilities of the ECC. We rely on this observation in our FTL design in Section 4.

We note, however, that the variance between the chips we examined is high, and that short and long-term effects do not depend only on the feature size. For example, the A16 chip is "better" than the A27 chip, but the B16 chip is "worse" than the B29 chip. Thus, the portion of the block's lifetime in which its pages can be reused safely depends on the characteristics of its chip. The FTL must take into account the long-term implications of reuse on the chips it is designed for.

3.5. $\overline{\text{LHH}}$ Reprogramming and BER

We conducted a similar evaluation of the short and long term effects of $\overline{\text{LHH}}$ reprogramming. To fully understand the implications of the disturbance caused by reprogramming the high page, we define the following variations of the $\overline{\text{LHH}}$ scheme. In $\overline{\text{LHH}}$-Skip-X reprogramming, the high pages are reprogrammed, skipping every X high pages. Thus, $\overline{\text{LHH}}$-Skip-0 is the basic $\overline{\text{LHH}}$ scheme examined in Figure 2(c). In $\overline{\text{LHH}}$-Skip-1 and $\overline{\text{LHH}}$-Skip-2, reprogramming is applied to one of every two or three high pages, respectively. Wear leveling within each block is obtained by alternating the skipping pattern in subsequent P/E cycles. For example, in $\overline{\text{LHH}}$-Skip-1, the even and odd high pages are reprogrammed in even and odd cycles, respectively.

In the fifth set of experiments, we measure the short-term effects of three variations of the $\overline{\text{LHH}}$ reprogramming scheme. The BER of the high pages in all our experiments was higher than that of the low pages. In this specific set of experiments, the BER of the high pages was more indicative of the wear of the block's cells than the overall BER (of low and high pages combined). The reason is

Table VI. Expected reduction in lifetime due to $\overline{\text{LHH}}$ reprogramming

$\overline{\text{LHH}}$ cycles	$\overline{\text{LHH}}$-Skip-1 cycles	$\overline{\text{LHH}}$-Skip-2 cycles	Baseline	B29	C19
T				99.99%	99.96%
	T			12.5%	62.7%
		T		5%	52.54%
	$0.8 \times T$		$0.2 \times T$		12.7%
	$0.6 \times T$		$0.4 \times T$	15%	8.5%
	$0.4 \times T$		$0.6 \times T$	7.5%	10.2%
	$0.2 \times T$		$0.8 \times T$	6%	
		$0.8 \times T$	$0.2 \times T$	5%	9.3%
		$0.6 \times T$	$0.4 \times T$	4.5%	4.2%
		$0.4 \times T$	$0.6 \times T$	4%	5.1%
		$0.2 \times T$	$0.8 \times T$		3.4%
	$0.6 \times T$	$0.2 \times T$	$0.2 \times T$	12%	
	$0.4 \times T$	$0.4 \times T$	$0.2 \times T$	13.5%	
	$0.2 \times T$	$0.6 \times T$	$0.2 \times T$	10.5%	
	$0.2 \times T$	$0.4 \times T$	$0.4 \times T$	6%	
	$0.2 \times T$	$0.2 \times T$	$0.6 \times T$	4%	

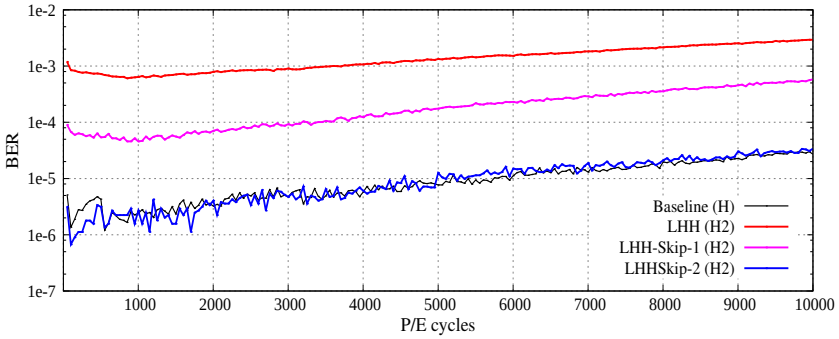

Fig. 6. Effects of $\overline{\text{LHH}}$ reprogramming variations on the B29 chip.

that during reprogramming, only the high pages hold valid data, while the low pages are inevitably corrupted. Thus, in the following discussion, the BER of the baseline is represented by the BER of its high pages (H), and the BER of $\overline{\text{LHH}}$ is represented by the BER of the high pages when they are reprogrammed (H_2). The BER of H_2 in each cycle is computed only for the high pages that were actually reprogrammed in this cycle.

Figure 6 shows the BER of $\overline{\text{LHH}}$-Skip-0, $\overline{\text{LHH}}$-Skip-1, and $\overline{\text{LHH}}$-Skip-2 on the B29 chip. The expected lifetime reduction for the B29 and C19 chips are summarized in Table VI. As we expected, the BER of $\overline{\text{LHH}}$-Skip-0 is always higher than 10^{-3}—reprogramming each high page significantly increased the BER of the high page adjacent to it. However, the BER of $\overline{\text{LHH}}$-Skip-1 remains below this threshold for a significant amount of the block's lifetime—as much as 87% in the B29 chip. $\overline{\text{LHH}}$-Skip-2 can be used safely even longer, for up to 95% of the block's lifetime, but has the obvious disadvantage of reusing less of the page's capacity. The BER of $\overline{\text{LHH}}$-Skip-2 is lower than that of the baseline in the beginning of the chip's lifetime, because it is measured here only for the reprogrammed high pages, which are far enough apart to avoid program disturbance.

Our sixth set of experiments measures the long term effects of $\overline{\text{LHH}}$ reprogramming. As before, each experiment was composed of two parts: we programmed the block with $\overline{\text{LHH}}$-Skip-1 or $\overline{\text{LHH}}$-Skip-2 in the first part, and with regular programming in the second part. We varied the length of the first part between 20% and 80%, depending on the results of each scheme on each chip in the previous set of experiments. Figure 7 shows the BER results for the B29 chip with $\overline{\text{LHH}}$-Skip-1 in the first part, and Table VI summarizes the expected lifetime reduction in the remaining experiments. These results demonstrate the advantage of the $\overline{\text{LHH}}$-Skip-X schemes: although at most half of the

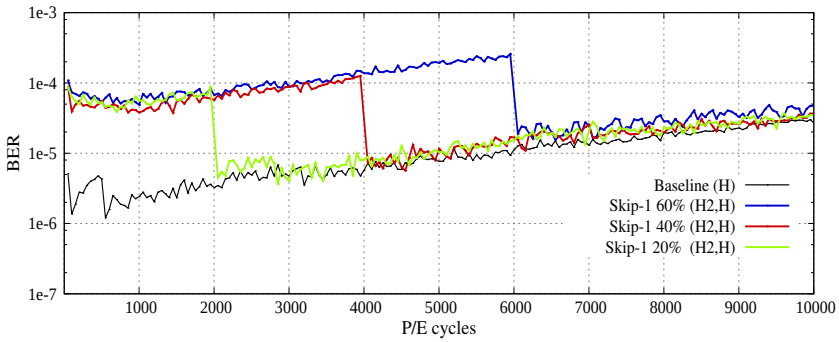

Fig. 7. Long term effects of $\overline{\text{LHH}}$-Skip-1 reprogramming on the B29 chip.

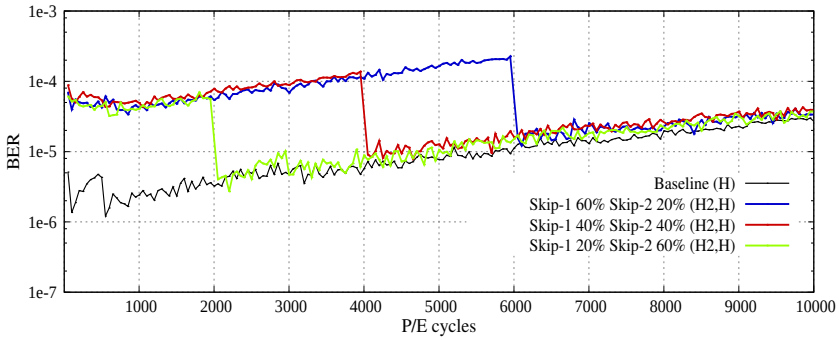

Fig. 8. Long term effects of combined $\overline{\text{LHH}}$-Skip-1 and $\overline{\text{LHH}}$-Skip-2 reprogramming on the B29 chip.

high pages can be reused in each cycle, skipping the remaining pages alleviates the long term wear caused by reprogramming. As a result, these schemes can be applied for a considerably longer portion of the block's lifetime compared to LLH reprogramming.

In the final set of experiments we attempted to maximize the reuse potential of the B29 chip, in which the difference between $\overline{\text{LHH}}$-Skip-1 and $\overline{\text{LHH}}$-Skip-2 was substantial. Each experiment was composed of three parts: we programmed the block with $\overline{\text{LHH}}$-Skip-1 and $\overline{\text{LHH}}$-Skip-2 in the first and second parts, respectively, and with regular programming in the last part. Figure 8 shows the BER results where the length of the first and second parts was varied and the length of the last part was fixed at 20%. The expected lifetime reduction for additional combinations is summarized in Table VI. Comparing figures 7 and 8 demonstrates that the additional $\overline{\text{LHH}}$-Skip-2 in the second part of the experiment has only a minor effect on the lifetime reduction.

3.6. Energy consumption

Flash read, write and erase operations consume different amounts of energy, which also depend on whether the operation is performed on the high page or on the low one, and on its data pattern. We examined the effect of LLH reprogramming on energy consumption by connecting an oscilloscope to the SigNAS tester. We calculated the energy consumed by each of the following operations on the A16 chip: an erasure of a block programmed with P_{LLH} and p=0.5, reading and writing a high and a low page, reprogramming a low page, and programming a high page on a partially-used block. This part of our evaluation is restricted to LLH reprogramming because reprogramming the high page is disabled on chips from manufacturer A.

To account for the transfer overhead of WOM encoded data, our measurements of read, program and reprogram operations included the I/O transfer to/from the registers. Our results, averaged over

Table VII. Energy consumed by flash operations on chip A16

Operation	Baseline (μJ)	LLH (μJ)
Erase	192.79	186.49
Read (L)	50.37	50.37
Read (H)	51.25	51.25
Program (L_1)	68.18	68.55
Reprogram (L_2)	NA	63.04
Program (H)	195.65	180.85
Average logical read	50.81	60.79
Average logical write	132.64	145.71

three independent measurements, are summarized in Table VII. We also present the average energy consumption per read or write operation with baseline and with LLH reprogramming, taking into account the size of the programmed data, the reading of used pages for supplying the invalid data as input to the WOM encoder, and the number of pages that can be written before each erasure.

These results show that page reuse consumes more overall energy than the baseline. This is in contrast to previous studies showing possible energy savings. These studies assumed that the energy is proportional to the number of programmed *cells*, which is equivalent in a first and in a second write [Grupp et al. 2009; Yadgar et al. 2015b]. However, our hardware evaluation shows that the number of reprogrammed *pages* is the dominant factor in energy consumption. While reprogramming a lower page consumes less energy than the average logical write in the baseline, the use of WOM encoding entails an extra read and page reprogram for each logical write. The low energy consumption of the saved erasures does not offset the additional energy consumed by those operations. We note, however, that when page reuse reduces the internal writes by the FTL, some energy savings may result.

The possibility of energy savings thus depends strongly on the way reprogramming is applied within the FTL. The evaluation of an FTL based on LLH reprogramming in our preliminary study showed that the need to reserve half of the block's capacity prevented it from reducing the amount of internal writes [Margaglia et al. 2016]. Examining the energy savings possible with $\overline{\text{LHH}}$ reprogramming remains part of our future work. Nevertheless, energy savings could be achieved more easily by FTLs that do not rely on WOM codes for the correctness of the reprogrammed data [Kaiser et al. 2013; Margaglia and Brinkmann 2015]. Such special-purpose designs are outside the scope of this study.

4. FTL DESIGN

Following our lessons from Section 3, we describe the general design principles for a *reprogramming FTL*—an FTL that reuses flash pages using a specified reprogramming scheme. We assume such an FTL would run on the SSD controller, and utilize the physical page and block operations supported by the flash controller. Thus, it shares the following basic concepts with the standard FTL and SSD.

To accommodate out-of-place writes, the physical storage capacity of the drive is larger than its exported logical capacity. The drive's *overprovisioning* is defined as $\frac{T-U}{U}$, where T and U represent the number of physical and logical blocks, respectively [Desnoyers 2013]. Typical values of overprovisioning are 7% and 28% for consumer and enterprise class SSDs, respectively [Smith 2013].

Whenever the number of clean blocks drops below a certain threshold, *garbage collection* is invoked. Garbage collection is typically performed *greedily*, picking the block with the minimum *valid count* (the lowest number of valid pages) as the victim for *cleaning*. The valid pages are *moved*—read and copied to another available block, and then the block is erased. These additional internal writes, referred to as *write amplification*, delay the cleaning process, and require, eventually, additional erasures. Write amplification does not accurately represent the utilization of drives that reuse pages for WOM encoded data, since some redundancy must always be added to the logical

data to enable second writes [Yaakobi et al. 2015; Yadgar et al. 2015a]. Thus, instead of deriving the number of erasures performed by the FTL from its write amplification, we measure them directly.

4.1. Reprogramming FTL

The design of a reprogramming FTL must address three key issues: (1) the specific reprogramming scheme used, including the resulting block states and the transitions between them, (2) the encoding scheme used for reprogrammed pages, and (3) the decision of which logical pages are written as second writes, i.e., reprogrammed on previously written physical pages. We elaborate on the application of the LLH and $\overline{\text{LHH}}$ schemes to FTL design in the following subsections.

WOM encoding. When WOM codes are employed for reusing flash pages, the FTL is responsible for determining whether a logical page is written in a first or a second write, and for recording the required metadata. The choice of WOM code determines the data written on clean physical pages, and the data written on them when they are reprogrammed. The encoding scheme in the reprogramming FTLs we present below is similar to that of ReusableSSD [Yadgar et al. 2015b]. Data on clean pages is written as is, without storage or encoding overheads. Data written as a second write on reprogrammed pages is encoded with a Polar WOM code that requires two physical pages to store the encoded data of one logical page [Burshtein 2015; Burshtein and Strugatski 2013]. This WOM implementation has a 0.25% encoding failure rate.

We note that the mathematical properties of WOM codes ensure they can be applied to any data pattern, including data that was previously scrambled or compressed. In fact, WOM encoding also ensures an even distribution of zeroes throughout the page, which is one of the objective of scrambling.

While manufacturers have increased the flash page size (see Table II), the most common size used by file systems remains 4KB. Our reprogramming FTL design distinguishes between the logical page used by the host and some larger physical page size. Thus, the FTL maps several logical pages onto each physical page. This allows it to program the encoded data for a second write on one physical page. In the rest of this section we assume that the physical page size is exactly twice the logical page size. We note that the changes required in the design if the physical pages are even larger are straightforward.

If the physical and logical page sizes are equal, a reprogramming FTL can utilize the multi-plane command that allows programming two physical pages in parallel on two different blocks, as in the ReusableSSD design. In both approaches, the latency required for reading or writing an encoded logical page on a second write is equal to the latency of one flash page write.

As in the design of ReusableSSD [Yadgar et al. 2015b], our reprogramming FTLs address the 0.25% probability of encoding failure by writing the respective logical page as a first write on a clean block, and prefetch the content of physical pages that are about to be rewritten to avoid the latency of an additional read. Pages are reprogrammed only in the safe portion of their block's lifetime (the first 40% in all but one of the chips we examined), thus limiting the long-term effect of reprogramming to an acceptable level.

Hot and cold data separation. Workloads typically exhibit a certain amount of skew, combining frequently updated *hot* data with infrequently written *cold* data. Separating hot and cold pages has been demonstrated as beneficial in several studies [Desnoyers 2014; Im and Shin 2010; Jimenez et al. 2014; Stoica and Ailamaki 2013]. Previous studies also showed that second writes are most beneficial for hot pages, minimizing the time in which the capacity of reused blocks is not fully utilized [Margaglia and Brinkmann 2015; Odeh and Cassuto 2014; Yadgar et al. 2015a; Yadgar et al. 2015b]. Thus, our reprogramming FTLs separate hot and cold pages into different logical partitions, as described below.

The classification of hot and cold pages is orthogonal to the design of the FTL, and can be done using a variety of approaches [Chiao and Chang 2011; Im and Shin 2010; Min et al. 2012; Stoica and Ailamaki 2013]. We describe the classification schemes used in our evaluations in Section 6.

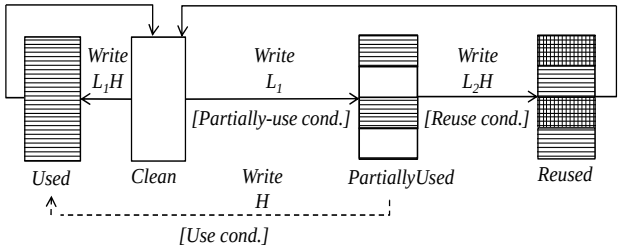

Fig. 9. Block life cycle in a Low-Low-High FTL.

4.2. Low-Low-High FTL

Blocks in a Low-Low-High FTL cycle between four states, as depicted in Figure 9. In the initial, *clean* state all the cells are in the erased state, *ER*. If all the pages are programmed ($write\ L_1H$), the block reaches the *used* state. Alternatively, if only the low pages are used ($write\ L_1$), the block reaches the *partially-used* state. A partially-used block can be reused, in which case the FTL will reprogram all or some of the low pages and all the high pages ($write\ L_2H$), transitioning the block to the *reused* state. Alternatively, the FTL can program the high pages and leave the low pages untouched ($write\ H$), thus transitioning the block to the used state. Used and reused blocks return to the clean state when they are erased.

The choice of state transition is determined by the conditions depicted in Figure 9. The conditions that determine when to *partially use*, *use* or *reuse* a block, as well as the encoding scheme used for reprogrammed pages, are in turn determined by the specific FTL design. We next describe *LLH-FTL*—our Low-Low-High FTL design.

In LLH-FTL, we write hot data on partially-used and reused blocks, and cold data on used blocks. Hot data on partially-used blocks is invalidated quickly, maximizing the benefit from reusing the low pages they are written on. Reused blocks store pages in first as well as in second writes. Nevertheless, we use them only for hot data, in order to maintain the separation of hot pages from cold ones.

Partially-use, use and reuse conditions. The number of partially-used blocks greatly affects the performance of a Low-Low-High FTL. Too few mean that the blocks will be reused too soon, while they still contain too many valid low pages, thus limiting the benefit from reprogramming. Too many mean that too many high pages will remain unused, reducing the available overprovisioned space, which might increase internal page moves. The three conditions in Figure 9 control the number of partially-used blocks: if the partially-use condition does not hold, a clean block is used with regular LH programming. In addition, the FTL may define a use condition, which specifies the circumstances in which a partially-used block is reclaimed, and its high pages will be written without rewriting the low pages. Finally, the reuse condition ensures efficient reuse of the low pages. The FTL allows partially-used blocks to accumulate until the reuse condition is met.

LLH-FTL allows accumulation of at most $threshold_{pu}$ partially-used blocks. This threshold is updated in each garbage collection invocation. An increase in the valid count of the victim block compared to previous garbage collections indicates that the effective overprovisioned space is too low. In this case the threshold is *decreased*. Similarly, a decrease in the valid count indicates that page reuse is effective in reducing garbage collections, in which case the threshold is *increased* to allow more reuse. Thus, the partially-use and reuse conditions simply compare the number of partially-used blocks to the threshold. To maintain the separation between hot and cold pages, LLH-FTL does not utilize the use condition.

In our preliminary study [Margaglia et al. 2016], we implemented and evaluated LLH-FTL on the OpenSSD Jasmine board [Ope 2015] with several synthetic and real world workloads. The results of this evaluation are tightly coupled with some additional limitations imposed by the high level SSD design. Our analysis in this work is focused on the low-level flash characteristics, and thus we

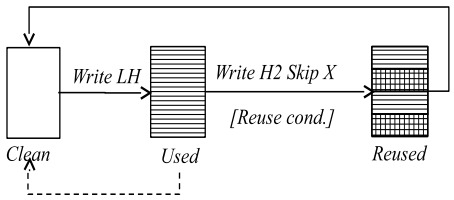

Fig. 10. Block life cycle in a $\overline{\text{Low-High-High}}$ FTL.

refer the reader to our earler publication for the detailed results. For completeness, we summarize below our results and conclusions from the system-level evaluation.

With 28% overprovisioning, LLH-FTL successfully reduced the number of erasures according to the amount of hot data in the workload. However, with low (7%) overprovisioning, reserving partially-used blocks for additional writes was not as efficient for reducing erasures, and in some cases it even increased the number of erasures instead. This limitation motivated the design of our $\overline{\text{LHH}}$-FTL which does not require any reservation, and is described in the following section.

Our performance measurements showed that even with high overprovisioning, and despite the considerable reduction in erasures (and thus, garbage collection invocations), the average I/O response time was almost unchanged. The reason is the high memory consumption of WOM encoded logical pages, which reduced the efficiency of the write buffers on the OpenSSD board, and increased the frequency of destaging to flash. When we applied the energy measurements described in Section 3.6, we observed an increase in energy consumption which was proportional to the reduction in erasures. The reduction in erasures did not reduce the amount of internal data copying in most of the workloads, and thus this reduction was not translated to time or energy savings.

4.3. $\overline{\text{Low-High-High}}$ FTL

Blocks in a $\overline{\text{Low-High-High}}$ FTL cycle between three states, as depicted in Figure 10. The clean and the used states are equivalent to those states in a Low-Low-High FTL: all the cells of a clean block are erased, and all the pages in a used page have been programmed once. A used block can be reused, in which case the FTL will reprogram some of its high pages ($write\ H_2, skip\ X$), transitioning the block to the reused state. Alternatively, the FTL can erase the block, in which case it returns to the clean state without being reused. Reused blocks return to the clean state when they are erased.

The most important difference compared to a Low-Low-High FTL is that in a $\overline{\text{Low-High-High}}$ FTL, blocks *always* transition from the clean to the used state, without going through the partially-used state. The design is thus simplified to a single condition that determines the time in which a block is reused. We assume that the skip value, X, is determined by the physical characteristics of the underlying flash chip, as demonstrated in the previous section. Nevertheless, future advances in flash technology may justify more complicated FTL designs that optimize the reprogramming scheme dynamically according to the condition of the chip and current as well as expected workload.

$\overline{\text{LHH}}$-FTL—our $\overline{\text{Low-High-High}}$ FTL design—uses a greedy scheme to determine which pages are reprogrammed in a reused block. Like previous designs, $\overline{\text{LHH}}$-FTL does not move valid pages from blocks before they are reused [Yadgar et al. 2015b; Margaglia and Brinkmann 2015]. It first reprograms the first invalid high page whose corresponding low page is also invalid. This will corrupt the low page, and may increase the BER in the adjacent high pages. It then skips X high pages, and continues to search for the next available high page, until the last page in the block is reached. By skipping X pages after each reprogrammed page, and by reprogramming the block only at the beginning of its lifetime, we ensure that the BER in the reprogrammed and in the skipped pages is within the error-correction capabilities of the ECC.

Without the need to reserve partially-used blocks, $\overline{\text{LHH}}$ reprogramming is almost orthogonal to other FTL design choices. $\overline{\text{LHH}}$-FTL takes advantage of this property. Logical pages in $\overline{\text{LHH}}$-FTL are partitioned according to their temperature and written into the hot partition or the cold one. Garbage collection and block state transitions take place separately within each partition. The

size of the hot partition (and the cold one, respectively) can be dynamically adjusted by picking a victim block from one partition and allocating it to the other partitions after it is erased. We study the optimal allocation of blocks to partitions within our theoretical framework in Section 5. Approximating the optimal partitioning online can be done by tracking the update frequency in each partition and adjusting the partition sizes to one of several sets of predetermined optimal values.

The reuse condition should strike a balance between two conflicting objectives. Reusing as many blocks as possible reduces the number of erasures. However, reusing a block that has little reprogramming potential might harm performance. Consider the illustrative example in Table VIII, where a block of 16 pages is reused when 5 of them are still valid. There is only one pair of high pages, P_2 and P_{12}, that can be reprogrammed with $\overline{\text{LHH}}$-Skip-1, increasing the valid count of the block from 5 to 6. Such a small increase means that the block is likely to be chosen again as victim before any of its valid pages is invalidated. Before this block is erased, its valid pages, including the one that was just reprogrammed, will be copied to a clean block. This cancels any benefit from reuse, and possibly incurrs additional latency due to copying. Alternatively, if the block is erased rather than reused, it can accommodate 11 (16-5) writes before the next garbage collection.

Table VIII. Example

High	Low
P_0: valid	P_1: —
P_2: $\Longleftrightarrow$	P_3: —
P_4: —	P_5: —
P_6: —	P_7: valid
P_8: valid	P_9: —
P_{10}: valid	P_{11}: —
P_{12}: $\Longleftrightarrow$	P_{13}: —
P_{14}: —	P_{15}: valid

Our theoretical framework provides the means to calculate the benefit, in terms of reduction in erasures, from reusing blocks in each partition. Thus, the reuse condition should hold only if this benefit is high enough to justify the potential increase in latency. $\overline{\text{LHH}}$-FTL applies a simple heuristic in which the reuse condition holds only in the hot partition, where the valid count of victim pages is expected to be low. $\overline{\text{LHH}}$-FTL relies on the theoretical framework also for choosing one of two options in each garbage collection invocation: either to erase a reused block, or to reuse a fully used block. This choice is done based on the minimum valid count of blocks in each of the two states, as explained in detail in Section 5.2. $\overline{\text{LHH}}$-FTL applies the theoretical results by periodically adjusting a threshold value to one of several predetermined optimal values.

5. THEORETICAL ANALYSIS

In this section we describe a theoretical framework for evaluating the benefit of a general reprogramming FTL. Our analysis compares the expected number of erasures with and without reprogramming, and is composed of two steps. In the first step, we calculate the number of erasures for a given garbage collection policy. In the next step, which is formulated as an optimization problem, we identify the policy that will result in the minimum number of erasures with and without reprogramming.

Our analysis, like those in previous studies, is based on the connection between the overprovisioningand the number of block erasures. This connection depends on the garbage collection algorithm used by the FTL and on the probability distribution of the page write requests in the workload. It may also depend on the number of pages in each block, but we assume that this number is large enough to avoid such dependency.

In the first part of our analysis, we consider uniform workloads: we assume that requests are uniformly distributed over all the logical pages. We follow the observation from [Hu and Haas 2010] that *greedy garbage collection*—the policy that always chooses for cleaning the block with the minimum number of valid pages—is optimal for uniform distribution. We also assume that greedy garbage collection is invoked whenever there are no more clean blocks, without requiring a minimum fraction of available blocks. The analysis with and without this requirement is similar [Desnoyers 2014]. In the second part of our analysis we relax the uniformity assumption, and show how to apply our results to non-uniform workloads.

We use the following notations throughout this section.

(1) Every block has Z pages. There are T physical pages in the SSD, and U logical pages, where both T and U are a multiple of Z.

(2) The overprovisioning is $\rho = (T - U)/U$. $\alpha = U/T = 1/(\rho + 1)$ is the *storage rate*—the ratio between logical data and physical storage. Our analysis is presented in terms of the storage rate (α) rather the more commonly used overprovisioning (ρ), to allow for more natural formulation. Note that the two are easily interchangeable using the definitions above.
(3) The write amplification is $WA = P/L$, where L is the number of write requests of logical pages and P is the number of resulting physical page writes. We also define L/Z to be the number of *logical block writes*.

Reprogramming writes a block multiple times before it is erased. Thus, WA is not the right figure of merit for a reprogramming FTL—it is possible to write more pages and yet erase less. Hence, we introduce a new measure that better characterizes this behavior.

Definition 5.1. The **erasure factor** EF in a flash memory system is the ratio between the number of block erasures E and the number of logical block writes L/Z. That is,

$$EF = \frac{E}{L/Z}.$$

Note that without reprogramming, $EF = WA$. In the rest of the section, we present an analysis of the optimal erasure factor (EF) without reprogramming and with various reprogramming schemes. The erasure factor is closely related to *writes per erase* (*WE*), an alternative measure proposed with similar motivation [Yadgar et al. 2015a]. WE is defined as the ratio between the number of logical page writes and the number of block erasures. Thus, we get $WE = L/E = Z/EF$. In other words, writes per erase are the inverse of the erasure factor, multiplied by the number of physical pages per block.

5.1. Optimal Erasure Factor without Reprogramming

The relation between the write amplification and overprovisioning has received a significant attention in recent years due to its implication for the lifetime of flash memories, see e.g. [Desnoyers 2014; Hu and Haas 2010; Stoica and Ailamaki 2013]. Of the numerous works in this area, we consider two recent studies which we believe give an accurate model of this analysis [Desnoyers 2014; Stoica and Ailamaki 2013]. The proof given here is based upon the analysis in these two studies and we give it in completeness since its understanding is crucial to the results in the rest of this section. We refer to the FTL that uses greedy garbage collection and no reprogramming as the ***baseline system***.

THEOREM 5.2. *The number of block erasures E and the erasure factor $EF_1(\alpha)$ of the baseline system are given by*

$$E = \frac{P}{Z} = \frac{L}{Z(1-\alpha')}, \quad EF_B(\alpha) = \frac{1}{1-\alpha'}, \tag{1}$$

where $\alpha = \frac{\alpha'-1}{\ln(\alpha')}$, or $\alpha' = -\alpha \cdot W\left(-\frac{1}{\alpha}e^{-1/\alpha}\right)$, and $W(x)$ is the Lambert W function.

PROOF. For $0 \le i \le Z$, let $N(i)$ be a random variable corresponding to the number of blocks with i valid pages, so $\sum_{i=0}^{Z} N(i) = T/Z$. If we denote by Y the expected number of valid pages when a block is erased, then for $0 \le i \le Y-1$, $N(i) = 0$, and $N(Y)$ is relatively small enough. We assume that the system is in a steady state and thus the expected value of $N(i)$ doesn't change over time[4]. According to this assumption, we also get that for $Y+1 \le i \le Z$,

$$iN(i) = C,$$

[4]These properties are taken from [Bux and Iliadis 2010; Desnoyers 2014] where this process is modeled as a Markov chain and the number of blocks with a given number of valid pages is fixed for analysis purposes.

for some constant C, or $N(i) = (Y+1)N(Y+1)/i$. Therefore, we get[5]

$$\begin{aligned} T/Z &= \sum_{i=0}^{Z} N(i) = \sum_{i=Y+1}^{Z} N(i) = \sum_{i=Y+1}^{Z} (Y+1)N(Y+1)/i \\ &= (Y+1)N(Y+1) \sum_{i=Y+1}^{Z} \frac{1}{i} \\ &\approx (Y+1)N(Y+1)(\ln(Z) - \ln(Y)) \\ &= (Y+1)N(Y+1)\ln(Z/Y). \end{aligned}$$

We also have that

$$U = \sum_{i=0}^{Z} iN(i) = \sum_{i=Y+1}^{Z} iN(i) = (Z-Y)(Y+1)N(Y+1).$$

Together, we get that

$$(Y+1)N(Y+1) = \frac{T/Z}{\ln(Z/Y)} = \frac{U}{Z-Y},$$

or

$$\alpha = \frac{U}{T} = \frac{Z-Y}{Z\ln(Z/Y)} = \frac{Y/Z-1}{\ln(Y/Z)} = \frac{\alpha'-1}{\ln(\alpha')}.$$

where $\alpha' = Y/Z$, and is given by $\alpha' = -\alpha \cdot W\left(-\frac{1}{\alpha}e^{-1/\alpha}\right)$.

Now, we deduce that for every $Z-Y$ logical page writes, Z physical pages are written. Hence, $P = L \cdot \frac{Z}{Z-Y} = \frac{L}{1-\alpha'}$, and

$$E = \frac{P}{Z} = \frac{L}{Z(1-\alpha')}, EF_B(\alpha) = \frac{E}{L/Z} = \frac{1}{1-\alpha'}.$$

□

The number of block erasures is simply given by $E = \frac{L}{Z} \cdot EF$. Thus, for brevity, we will only discuss the erasure factor.

5.2. Optimal Erasure Factor with Ideal Reprogramming

We begin our analysis of the erasure factor with reprogramming by considering an ideal reprogramming scheme in which all the pages of a used block can be reprogrammed. We assume that the FTL employs the WOM codes described in Section 4.1 for reusing flash pages. Thus, reprogramming still requires two used physical pages for writing one logical page, to accommodate the overhead of the WOM encoding.

This ideal FTL implements the following greedy garbage collection policy. Blocks are managed in two queues according to their states: used or reused. The FTL is characterized by a parameter $0 \leq \gamma_1 \leq 1$. Let B_1, B_2 be the used and reused blocks with the minimum number of valid logical pages, respectively. If the number of valid pages in B_1 is at most $Y_1 = \gamma_1 \cdot Z$ then this block is reused. Otherwise the block B_2 is physically erased and its valid pages are copied and written to an available block. In other words, the reuse condition in this FTL is always true, and γ_1 determines whether to pick a victim block for erasure or for reuse.

The value of γ_1 in the greedy policy which optimizes the number of erasures and the corresponding erasure factor is found in the next theorem.

[5]There are better approximations of the differences between two Harmonic series. However, the one we use here is preferable for our analysis because it provides expressions that do not depend on the number of pages per block.

THEOREM 5.3. *For any storage ratio α and greedy garbage collection with parameter γ_1, the erasure factor of the ideal FTL is given by*

$$EF_1'(\alpha,\gamma_1) = \frac{1}{3/2-\gamma_1/2-\gamma_2}, \tag{2}$$

where γ_2 satisfies the relation

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{1+\gamma_1}{2\gamma_1}\right)+\frac{\gamma_1-3}{2\alpha}}\right). \tag{3}$$

The optimal erasure factor is given by

$$EF_1(\alpha) = \min_{0\le\gamma_1\le 1}\left\{EF_1'(\alpha,\gamma_1)\right\}. \tag{4}$$

PROOF. Let $Y_1 = \gamma_1 \cdot Z$ and let us denote by Y_2 the expected number of valid pages when a block on a second write is physically erased and let $\gamma_2 = Y_2/Z$. We will determine the relation between the values of Y_1 and Y_2. For $0 \le i \le Z$, we denote by $N_1(i), N_2(i)$ the number of blocks with i valid pages on a first, second write, respectively. Notice first that for $i \le Y_1$, $N_1(i) = 0$ and for $i \le Y_2$, $N_2(i) = 0$. Furthermore, when a block is moved from first to second write, it already contains Y_1 valid pages. Since every logical page is written into 2 available pages, the total number of logical pages this block can accommodate is at most $Y_1 + (Z - Y_1)/2 = (Z + Y_1)/2$ and thus $N_2(i) = 0$ for $i > (Z + Y_1)/2$.

We follow the same steps of the proof of Theorem 5.2 to have the following equations:

$$\begin{aligned}&(Y_1+1)N_1(Y_1+1) = \cdots = ZN_1(Z)\\ =&(Y_2+1)N_2(Y_2+1) = \cdots = \frac{Z+Y_1}{2}N_2(\frac{Z+Y_1}{2}).\end{aligned}$$

According to these definitions, a block can accommodate Z page writes on the first write and $(Z - Y_1)/2$ more page writes on the second write. Furthermore, on every block erasure, Y_2 pages were moved from a previously erased block, so the number of erasures is given by

$$E = \frac{L + EY_2}{Z + (Z - Y_1)/2}$$

or

$$E = \frac{L}{3Z/2 - Y_2 - Y_1/2} = \frac{L}{Z}\cdot\frac{1}{3/2-\gamma_1/2-\gamma_2}.$$

Hence, the erasure factor, as a function of both γ_1 and γ_2 is $1/(3/2 - \gamma_1/2 - \gamma_2)$.

Following the rest of the steps from Theorem 5.2 we get

$$\begin{aligned}T/Z &= \sum_{i=0}^{Z}(N_1(i)+N_2(i)) = \sum_{i=Y_1+1}^{Z}N_1(i) + \sum_{i=Y_2+1}^{\frac{Z+Y_1}{2}}N_2(i)\\ &= \sum_{i=Y_1+1}^{Z}\frac{(Y_1+1)N_1(Y_1+1)}{i} + \sum_{i=Y_2+1}^{\frac{Z+Y_1}{2}}\frac{(Y_1+1)N_1(Y_1+1)}{i}\\ &= (Y_1+1)N_1(Y_1+1)\left(\sum_{i=Y_1+1}^{Z}\frac{1}{i} + \sum_{i=Y_2+1}^{\frac{Z+Y_1}{2}}\frac{1}{i}\right)\\ &\approx (Y_1+1)N_1(Y_1+1)\left(\ln\left(\frac{Z}{Y_1}\right)+\ln\left(\frac{Z+Y_1}{2Y_2}\right)\right)\\ &= (Y_1+1)N_1(Y_1+1)\ln\left(\frac{1+\gamma_1}{2\gamma_1\gamma_2}\right).\end{aligned}$$

As before, we also have

$$\begin{aligned} U &= \sum_{i=0}^{Z} iN_1(i) + \sum_{i=0}^{Z} iN_2(i) \\ &= \sum_{i=Y_1+1}^{Z} iN_1(i) + \sum_{i=Y_2+1}^{\frac{Z+Y_1}{2}} iN_2(i) \\ &= (3Z/2 - Y_1/2 - Y_2)(Y_1+1)N_1(Y_1+1). \end{aligned}$$

Thus we get

$$(Y_1+1)N_1(Y_1+1) = \frac{U}{3Z/2 - Y_1/2 - Y_2} = \frac{T/Z}{\ln\left(\frac{1+\gamma_1}{2\gamma_1\gamma_2}\right)},$$

or

$$\alpha = \frac{3/2 - \gamma_1/2 - \gamma_2}{\ln\left(\frac{1+\gamma_1}{2\gamma_1\gamma_2}\right)},$$

that is

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{1+\gamma_1}{2\gamma_1}\right)+\frac{\gamma_1-3}{2\alpha}}\right). \tag{5}$$

Hence, the erasure factor, as a function of α and γ_1 is

$$EF_1'(\alpha,\gamma_1) = \frac{1}{3/2 - \gamma_1/2 - \gamma_2},$$

where γ_2 is given by (5). Lastly, since we can choose the threshold γ_1, the value $EF_1(\alpha)$ is achieved by minimizing the value of $EF_1'(\alpha,\gamma_1)$ under the condition in (5). □

5.3. Optimal Erasure Factor with Partial Reprogramming

Next, we generalize the analysis of the ideal reprogramming FTL to realistic flash characteristics, where only a subset of the pages in a block can be reprogrammed. We start by considering $\overline{\text{LHH}}$ reprogramming, assuming that all the high pages can be reprogrammed safely, and then consider the general case of partial reprogramming.

THEOREM 5.4. *For any storage ratio α and greedy garbage collection with parameter γ_1, the erasure factor of a reprogramming FTL based on the $\overline{LHH}$ scheme is given by*

$$EF_2'(\alpha,\gamma_1) = \frac{1}{5/4 - \gamma_1/4 - \gamma_2}, \tag{6}$$

where γ_2 satisfies the relation

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{1+3\gamma_1}{4\gamma_1}\right)+\frac{\gamma_1-5}{4\alpha}}\right). \tag{7}$$

The optimal erasure factor is given by

$$EF_2(\alpha) = \min_{0\le\gamma_1\le 1}\left\{EF_2'(\alpha,\gamma_1)\right\}. \tag{8}$$

The proof is based on the following observation. Y_1 out of the Z pages of a used block are still valid when it is chosen for reuse. Due to the uniformity of the updates, half of these pages are high. Thus, $(Z - Y_1)/2$ high pages are available for reprogramming, resulting in a total of

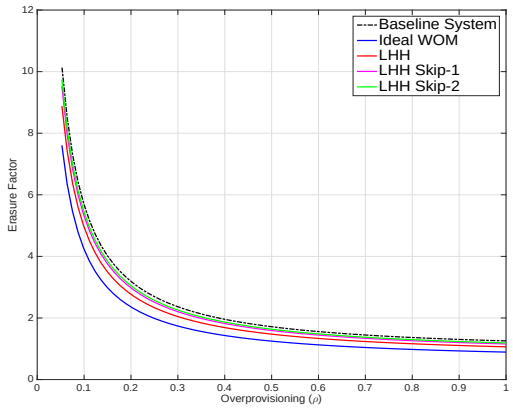

Fig. 11. The expected erasure factor (EF) for uniform workloads with and without reprogramming.

$Y_1 + (Z - Y_1)/4 = (Z + 3Y_1)/4$ valid pages on the reused block. The proof of Theorem 5.4 is a special case of Theorem 5.5 below and thus we skip its presentation.

We now consider the general case of partial reprogramming, where only one out of every S pages can be reprogrammed. In other words, we require a gap of $S - 1$ pages between every two reused pages, to avoid disturbance and allow safe reprogramming. Thus, for the $\overline{\text{LHH}}$, $\overline{\text{LHH}}$-Skip-1, and $\overline{\text{LHH}}$-Skip-2 schemes described in Section 3, we get $S = 2$, $S = 4$, and $S = 6$, respectively. We generalize the results in Theorem 5.4 as follows.

THEOREM 5.5. *For any storage ratio α and greedy garbage collection with parameter γ_1, the erasure factor of a reprogramming FTL with parameter S is given by*

$$EF'_S(\alpha, \gamma_1) = \frac{1}{1 + \frac{1}{2S} - \frac{\gamma_1}{2S} - \gamma_2}, \tag{9}$$

where γ_2 satisfies the relation

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha} e^{\ln\left(\frac{1+(2S-1)\gamma_1}{2S\gamma_1}\right) + \frac{\gamma_1 - (2S+1)}{2S\alpha}}\right). \tag{10}$$

The optimal erasure factor is given by

$$EF_S(\alpha) = \min_{0 \le \gamma_1 \le 1} \left\{ EF'_S(\alpha, \gamma_1) \right\}. \tag{11}$$

The proof is based on the same observation, that the $(Z - Y_1)$ valid pages on the used block are distributed uniformly. In the general case, we utilize only $1/S$ of them, resulting in a total of $Y_1 + (Z - Y_1)/2S = \frac{Z + (2S-1)Y_1}{2S}$ valid pages on the reused block. We give the full proof of Theorem 5.5 in the Appendix.

Figure 11 illustrates the benefit, in terms of the erasure factor, from the reprogramming schemes examined in Section 3. These results demonstrate the gap between the theoretical benefit of the ideal reprogramming scheme and the modest benefit achievable in practical designs that take into account the limitations of current flash chips. For example, with an overprovisioning of 28%, the erasure factor of the baseline, the ideal reprogramming and the $\overline{\text{LHH}}$-Skip-1 scheme is 2.5, 1.83, and 2.3, respectively. This corresponds to a theoretical reduction in erasures of 27%, but an expected practical reduction of only 8%.

Remark 5.6. The analysis of the reprogramming schemes can be extended further, to apply to even more general restictions on reprogramming. Assume that A pages out of the Z pages in a used block are available for reuse. Then, following the analysis in the proofs of Theorems 5.3– 5.5, the erasure factor is given by

$$EF = \frac{1}{1 + \frac{A}{2Z} - \gamma_2},$$

and

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{\gamma_1+\frac{A}{2Z}}{\gamma_1}\right)-\frac{1+\frac{A}{2Z}}{\alpha}}\right).$$

One can verify that by setting $A = Z - Y_1$ we get the result from Theorem 5.3 and for $A = \frac{Z-Y_1}{2S}$ we get the result from Theorem 5.5.

This formulation allows us to address complex limitations like those implied by the $\overline{\text{LHH}}$ scheme. With $\overline{\text{LHH}}$, we can safely reprogram a high page only if its corresponding low page is also invalid. In this case, the value of A will be given by $A = \frac{Z}{2} - Y_1 + \left(\frac{Y_1}{Z}\right)^2$, and we get $\frac{A}{2Z} \approx \frac{1}{4} - \frac{\gamma_1}{2}$. The resulting erasure factor is slightly higher than the one obtained by Theorem 5.4, and is given by

$$EF \approx \frac{1}{5/4 - \gamma_1/2 - \gamma_2},$$

and

$$\gamma_2 = -\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{1+2\gamma_1}{4\gamma_1}\right)+\frac{2\gamma_1-3}{4\alpha}}\right).$$

5.4. Optimal Erasure Factor with Non-Uniform Workloads

Our assumption in Sections 5.1 and 5.2, that all the logical pages are updated with uniform distribution, does not hold in many real workloads. We apply our theoretical framework to non-uniform workloads following the approach used in previous studies: pages are grouped into several partitions according to their *temperatures*, which represent their update frequency [Desnoyers 2014; Stoica and Ailamaki 2013]. Namely, *hot* pages are updated frequently, while *cold* pages are rarely updated. We first consider workloads with only two temperatures, hot and cold, and then generalize our results for any number of temperatures.

We now assume that the U physical pages are distributed into hot and cold pages, where the number of hot and cold pages is $H = fU$ and $C = (1-f)U$ for some $0 < f < 1$, respectively. Update requests access hot or cold pages with probability p and $1-p$, respectively. We assume, as in previous studies, that within each temperature pages are updated uniformly. Thus, within each partition, the FTL will use the optimal greedy garbage collection policy obtained from our analysis above, with the goal of minimizing the total number of block erasures.

The T physical pages are divided into two partitions, one for the hot pages and the other for the cold pages. Assume that the number of physical pages allocated for the hot pages is βT, so the number of physical pages allocated for the cold pages is $(1-\beta)T$. To ensure that in each partition the number of physical pages is greater than the number of logical pages, we require that

$$\beta T > fU, (1-\beta)T > (1-f)U,$$

which implies that

$$f\alpha < \beta < 1 - (1-f)\alpha.$$

Our analysis provides the means to calculate the optimal β for a given workload and overprovisioning value.

Table IX. Parameters for the baseline system with two temperatures

Parameter	Hot Partition	Cold Partition
# physical pages	$T_h = \beta T$	$T_c = (1-\beta)T$
# logical pages	$U_h = fU$	$U_c = (1-f)U$
# logical page writes	$L_h = pL$	$L_c = (1-p)L$
storage rate	$\alpha_h = \frac{fU}{\beta T} = \frac{f\alpha}{\beta}$	$\alpha_c = \frac{(1-f)\alpha}{1-\beta}$
overprovisioning	$\rho_h = \frac{\beta T - fU}{fU}$	$\rho_c = \frac{(1-\beta)T-(1-f)U}{(1-f)U}$
erasure factor	$EF_{B,h} = EF_B(\alpha_h)$	$EF_{B,c} = EF_B(\alpha_c)$
# block erasures	$E_{B,h} = \frac{L_h}{Z} \cdot EF_{B,h}$	$E_{B,c} = \frac{L_c}{Z} \cdot EF_{B,c}$

5.4.1. Analysis for The Baseline System. Desnoyers [Desnoyers 2014] provides a thorough analysis of the optimal partitioning for a workload with two temperatures and no reprogramming. Our analysis for this special case is similar, and we formulate it here in its completeness in order to extend it to the general case of multiple partitions and reprogramming. We summarize the parameters of each partition in Table IX. Our results are formulated as an explicit optimization problem, as in the previous cases.

THEOREM 5.7. *The erasure factor for the baseline system and non-uniform updates with parameters f and p is given by*

$$EF_B(\alpha, f, p) = \min_{f\alpha<\beta<1-(1-f)\alpha} \{E_B(\beta)\},$$

where

$$E_B(\beta) = p \cdot EF_B\left(\frac{f\alpha}{\beta}\right) + (1-p) \cdot EF_B\left(\frac{(1-f)\alpha}{1-\beta}\right).$$

PROOF. Our goal is to minimize the total number of erasures

$$\begin{aligned} E_B =& E_{B,h} + E_{B,c} = \frac{L_h}{Z} \cdot EF_{B,h} + \frac{L_c}{Z} \cdot EF_{B,c} \\ =& \frac{L}{Z}\left(pEF_{B,h} + (1-p)EF_{B,c}\right) = \frac{L}{Z}\left(pEF_B(\alpha_h) + (1-p)EF_B(\alpha_c)\right). \end{aligned}$$

This is equivalent to finding the minimum value of

$$E_B(\beta) = p \cdot EF_B\left(\frac{f\alpha}{\beta}\right) + (1-p) \cdot EF_B\left(\frac{(1-f)\alpha}{1-\beta}\right),$$

in the range $f\alpha < \beta < 1-(1-f)\alpha$. □

Thus, the erasure factor within each partition is optimized in order to reduce the combined erasure factor. The optimal β is the one that optimizes the average of the two values, weighted by the frequency of updates in each partition.

5.4.2. Analysis for Reprogramming FTLs. For the generalization of Theorem 5.7 to reprogramming FTLs we assume, as before, that all the used blocks are reused before they are physically erased. This assumption contradicts practical experience, that reprogramming should be applied only to hot data [Margaglia and Brinkmann 2015; Odeh and Cassuto 2014; Yadgar et al. 2015a; Yadgar et al. 2015b]. However, it does hold when the number of erasures is the only objective, disregarding the time spent on moving pages from victim blocks. We discuss this limitation of our analysis further in Section 5.5 below. We summarize the parameters of each partition in Table X.

As in the previous case we get the following result.

Table X. Parameters for a reprogramming FTL with parameter S with two temperatures

Parameter	Hot Partition	Cold Partition
# physical pages	$T_h = \beta T$	$T_c = (1-\beta)T$
# logical pages	$U_h = fU$	$U_c = (1-f)U$
# logical page writes	$L_h = pL$	$L_c = (1-p)L$
storage rate	$\alpha_h = \frac{f\alpha}{\beta}$	$\alpha_c = \frac{(1-f)\alpha}{1-\beta}$
overprovisioning	$\rho_h = \frac{\beta T - fU}{fU}$	$\rho_c = \frac{(1-\beta)T-(1-f)U}{(1-f)U}$
erasure factor	$EF_{S,h}(\alpha_h) = EF_S(\alpha_h)$	$EF_{S,c}(\alpha_c) = EF_S(\alpha_c)$
# block erasures	$E_{S,h} = \frac{L_h}{Z} \cdot EF_{S,h}(\alpha_h)$	$E_{S,c} = \frac{L_c}{Z} \cdot EF_{S,c}(\alpha_c)$

THEOREM 5.8. *The erasure factor for a reprogramming FTL with parameter S and non-uniform updates with parameters f and p is given by*

$$EF_S(\alpha, f, p) = \min_{f\alpha<\beta<1-(1-f)\alpha} \{E_S(\beta)\},$$

where

$$E_S(\beta) = p \cdot EF_S\left(\frac{f\alpha}{\beta}\right) + (1-p) \cdot EF_S\left(\frac{(1-f)\alpha}{1-\beta}\right).$$

PROOF. As in the previous case, we seek to find the value of β which minimizes the total number of block erasures

$$\begin{aligned} E_S =& E_{S,h} + E_{S,c} = \frac{L_h \cdot EF_{S,h}(\alpha_h)}{Z} + \frac{L_c \cdot EF_{S,c}(\alpha_c)}{Z} \\ =& \frac{L}{Z}\left(pEF_{S,h} + (1-p)EF_{S,c}\right). \end{aligned}$$

Hence now we minimize the function

$$E_S(\beta) = pEF_S\left(\frac{f\alpha}{\beta}\right) + (1-p)EF_S\left(\frac{(1-f)\alpha}{1-\beta}\right),$$

in the range $f\alpha < \beta < 1-(1-f)\alpha$. □

5.4.3. Analysis for Multiple Temperatures. Characterizing non-uniform workloads by a pair of parameters, f and p, is an appealing simplification. However, several recent studies showed the potential benefit of a finer classification of page access frequencies to more than two temperatures [Stoica and Ailamaki 2013; Yadgar and Gabel 2016]. To address this general case, we now assume that the logical pages are classified into k groups, each characterized by a pair of parameters, f_i and p_i, and stored in a separate partition.

The parameters for the ith partition are as follows:

(1) The number of physical pages is $T_i = \beta_i T$,
(2) The number of logical pages is $U_i = f_i U$,
(3) The number of logical page writes is $L_i = p_i L$,
(4) The storage rate is $\alpha_i = \frac{f_i U_i}{\beta_i T} = \frac{f_i \alpha}{\beta_i}$,
(5) The overprovisioning is $\rho_i = \frac{\beta_i T - f_i U}{f_i U} = \frac{1}{\alpha_i} - 1$,
(6) The erasure factor is $EF_i = EF_S(\alpha_i)$,

while the values of $f_1, \ldots, f_k$ and $p_1, \ldots, p_k$ are given and $\sum_{i=1}^{k} f_i = \sum_{i=1}^{k} p_i = 1$. Then, the goal is to find the values of $\beta_1, \ldots, \beta_k$, such that $\sum_{i=1}^{k} \beta_i = 1$ and the total erasure factor is minimized. This optimization problem is formulated in the next theorem.

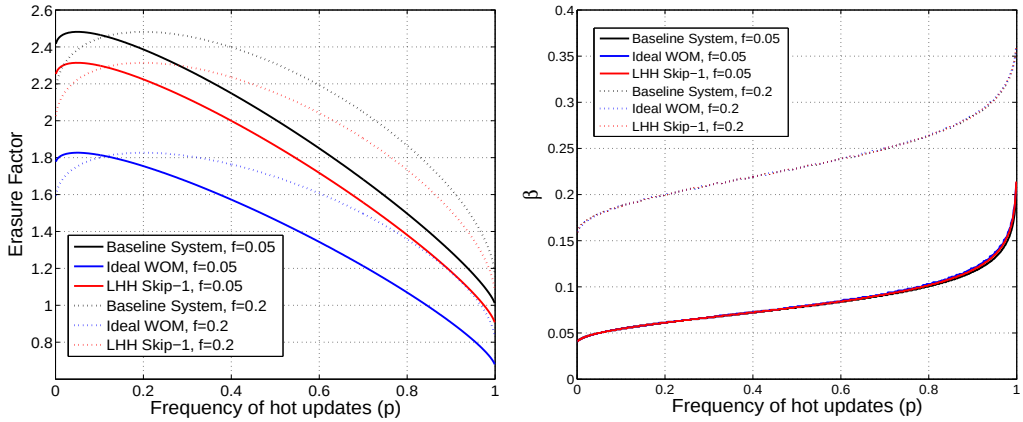

Fig. 12. The expected erasure factor (left) and the size of the hot partition (right) for non-uniform workloads with and without reprogramming. We fix the overprovisioning value at 28% and vary p for two values of f.

THEOREM 5.9. *The erasure factor for a reprogramming FTL with parameter S and non-uniform updates with parameters $f_1, \ldots, f_k$ and $p_1, \ldots, p_k$ is given by*

$$EF_{multi}(\alpha, f_1, \ldots, f_k, p_1, \ldots, p_k) = \min_{\beta_1,\ldots,\beta_k} \{E_{multi}(\beta_1, \ldots, \beta_k)\},$$

where

$$E_{multi}(\beta_1, \ldots, \beta_k) = \sum_{i=1}^{k} p_i EF_S(\alpha_i),$$

and $\beta_1, \ldots, \beta_k$ satisfy $\sum_{i=1}^{k} \beta_i = 1$ and for $1 \leq i \leq k-1$

$$f_i\alpha < \beta_i < 1 - \sum_{j=1}^{i-1} \beta_j - \sum_{j=i+1}^{k} f_j\alpha.$$

Figure 12 illustrates the effect of the workload parameters on the erasure factor. These results further demonstrate the gap between the theoretical benefit of the ideal reprogramming scheme and the achievable benefit in practical designs. Note that the maximal erasure factor is reached when the frequency of updated to hot pages is equal to their portion of the data ($p = f$), which corresponds to a uniform workload. We also note that the benefit from reprogramming increases with the portion of hot data.

Interestingly, the optimal size of the hot partition (β) depends only on the workload characteristics, p and f, and not on the reprogramming scheme. This property is especially desirable for FTL designs that dynamically adapt the reprogramming scheme to the workload characteristics or to the condition of the underlying flash hardware.

5.5. Discussion

The theoretical framework presented in this section is limited in several aspects. First, it does not provide an analytical solution to the optimization problems formulated in our theorems. In order to obtain the optimal γ_1 and the corresponding erasure factor we performed an exhaustive search for $0 \leq \gamma_1 \leq 1$. We implemented this search with a simple Matlab program that ran for several minutes.

Our framework is also limited in that the number of erasures is its only objective. Additional costs of reprogramming, such as possible increase in cell wear, latency and energy consumption,

Table XI. Trace characteristics of MSR and synthetic workloads. In the MSR workloads, pages can be accessed as hot and cold in different requests.

Volume	Unique pages (MB)	Hot page ratio ($\sim f$)	Cold page ratio ($\sim f-1$)	Hot write ratio (p)	Total writes (GB)
rsrch_0	300	0.63	0.62	0.95	11
ts_0	550	0.57	0.59	0.94	12
src2_0	510	0.61	0.66	0.91	10
web_0	730	0.35	0.82	0.87	17
usr_0	660	0.66	0.52	0.86	14
wdev_0	350	0.38	0.77	0.85	7
stg_0	400	0.59	0.77	0.85	16
prxy_0	330	0.81	0.59	0.78	21
hm_0	1700	0.53	0.9	0.7	23
src1_2	670	0.25	0.9	0.22	45
proj_0	1700	0.31	0.84	0.14	145
zipf(0.9,0.95,1)	1024	0.001	0.999	0.2	10

are not taken into account. This model implies that reprogramming is always beneficial, and blocks might be reused even if they will only accommodate a single logical page write, and this page will be immediately moved and written to a clean block in order to erase the reused block. This is illustrated in the example in Section 4. To avoid such scenarios, the FTL may disable reprogramming in cold partitions, or whenever the reduction in the erasure factor is too small to justify the possible increase in additional costs.

6. MODEL VALIDATION

In the following section we validate our theoretical model by answering the following questions: (1) how useful is the garbage collection policy derived for reprogramming FTLs in this model? (2) how accurate is the number of erasures derived from the model? and (3) how well does the model predict the possible benefit from reprogramming?

6.1. Reprogramming FTL Simulator

We build a special purpose simulator that implements the basic FTL functionality and measures the number of erasures for each workload, system setting, and theoretical parameters. We implement the baseline system with greedy garbage collection, as well as the $\overline{\text{LHH}}$-FTL and its variation with $\overline{\text{LHH}}$-Skip-1 and $\overline{\text{LHH}}$-Skip-2. We compare the number of erasures of each FTL and each workload, with and without partitioning the pages according to their temperatures. In other words, when the FTL uses only one partition, we calculate γ_1 according to Theorem 5.5. When the FTL uses two or more partitions, we calculate β_i and γ_1 in the i-th partition according to Theorem 5.9. We always compare the reprogramming FTLs to a baseline system with the same number of partitions. We examine two values of overprovisioning (ρ), 7% and 28%, and set the number of pages per block (Z) to 256. We align the requests in the real workloads to a page size of 4KB. We show only results for ρ=28%, where the benefit from reprogramming and the properties of our model could be easily observed.

6.2. Workloads

We used three synthetic workloads with a Zipf distribution with exponential parameter $\alpha = 0.9, 0.95$ and 1. In these traces, the frequency of access to page n is proportional to $\frac{1}{\alpha^n}$. Thus, we could mark each write request with the temperature of the page it is about to update. We marked the pages with five different temperatures as follows.

For each Zipf trace we extracted five thresholds $n_i, 0 < i \leq 5$, such that $n_0 = 0$, and pages with logical address between n_{i-1} and n_i were accessed 20% of the time. This divides the logical address space into five temperatures, such that pages with temperature i are always colder than pages with temperature $< i$. While this classification is impossible in real world settings, it serves

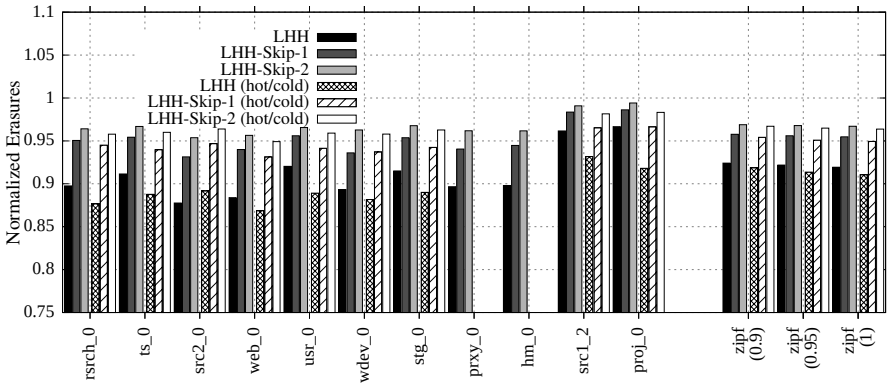

Fig. 13. Number of erasures with $\overline{\text{LHH}}$-FTL normalized to that of the baseline system.

for the validation of our theoretical model, and for demonstrating the benefit from page reuse under optimal conditions.

We also use real world traces from the MSR Cambridge workload [SNI 2014; Narayanan et al. 2008], which contains week-long traces from 36 volumes on 13 servers. We used the 16 traces with the smallest address space, and that include enough write requests for a meaningful analysis. These traces vary in a wide range of parameters, summarized in Table XI. In these traces, pages were classified as cold if they were written in a request of size 64KB or larger. This simple online heuristic was shown to perform well in several previous studies [Chiao and Chang 2011; Im and Shin 2010; Yadgar et al. 2015b].

The limitation of this heuristic is that pages can be classified as cold and hot simultaneously in different requests. This can be observed in Table XI, where the sum of the hot page ratio (column 3) and the cold page ratio (column 4) of the MSR workloads is higher than 1. Similarly, real workloads often exhibit a dynamic working set, where hot pages gradually become cold and vice versa. Practical FTL implementations address this behavior by moving logical pages to a different partition when they are updated and a change in their temperature is detected [Chiao and Chang 2011; Im and Shin 2010; Stoica and Ailamaki 2013]. In our evaluation, we adhere to the static partitioning determined by our theoretical framework. Thus, we use the approximate frequency ($\sim f$) of access to hot pages in order to calculate β, and store in the hot partition all the pages that are marked as hot in at least one request. We discuss the limitations of this approach below.

The workloads were collected on volumes of different sizes, and access logical pages from a wide range of address space sizes. However, most of this address space is never updated within the duration of the trace. The logical pages that are never updated affect the number of erasures because they inevitably occupy physical pages on blocks that also store hot pages. We chose to omit those pages from our evaluation, and set the logical capacity, U, as the number of unique pages in each workload. We note that setting U to the size of the full address space would have forced us to also use a very large physical device, resulting in a very large overprovisioning space compared to the size of the working set.

6.3. Simulation Results

We first examine the reduction in erasures achieved by reprogramming. Figure 13 shows the number of erasures performed by each of the $\overline{\text{LHH}}$ FTLs with 28% overprovisioning, with and without hot and cold data separation, normalized to that of the baseline system. The traces are ordered by the ratio of requests to hot pages (p), descending, although we note that this is not the dominant factor in determining the benefit from reprogramming. Reprogramming reduces the number of erasures by up to 13%, 7%, and 5%, with $\overline{\text{LHH}}$-FTL, $\overline{\text{LHH}}$-Skip-1, and $\overline{\text{LHH}}$-Skip-2, respectively.

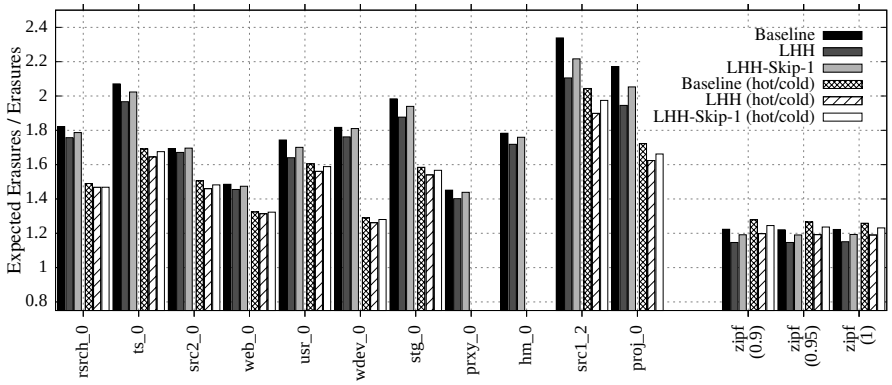

Fig. 14. Ratio between expected number of erasures and measured erasures.

This reduction is naturally smaller than that achieved by previous FTL designs that did not take into account the specific limitations of MLC reprogramming. For example, Reusable SSD [Yadgar et al. 2015b] reduces the number of erasures by up to 33% by reusing all the pages in each reprogrammed block, which is impossible with MLC flash. The Seal FTL [Margaglia and Brinkmann 2015] eliminates up to 80% of erasures on MLC flash by using LLH reprogramming. However, Seal FTL is not a general-purpose FTL, as it requires that applications write "overwrite compatible" data, which allows up to eight reprogramming cycles for each page without any WOM encoding overhead.

Our results show that separating hot and cold data into two partitions always increases the benefit from reprogramming. They also show that this benefit decreases as the skip value increases. The results with ρ=7% were similar, though the difference between the different schemes was much smaller. One exception is the proj_0 trace, which has a very low ratio of hot requests. With low overprovisioning, reprogramming increased the number of erasures by 1%-14%, because the high amount of WOM encoded cold pages reduced the availability of clean blocks.

The experiments for two of the MSR traces, prxy_0 and hm_0, with hot and cold data separation did not complete successfully. This demonstrates the limitation of the static allocation of blocks to partitions in our simulation. This allocation was based on the β value obtained from Theorems 5.7 and 5.9 for the baseline and $\overline{\text{LHH}}$-FTL experiments, respectively. For a given ratio of hot pages in a workload, f, and assuming that pages are either hot or cold, the given β ensures that the size of each partition can accommodate all the pages that belong to it. However, in these two traces, a large portion of the pages appeared in both hot and cold requests. Many hot pages were initially classified as cold and written in the cold partition, which overflowed as a result.

To validate the accuracy of our model, we compare the expected number of erasures in each experiment with the number measured in the simulation. We calculate the expected number of erasures using the erasure factor given by our theoretical model, so that $E = EF \cdot \frac{L}{Z}$. Figure 14 shows the ratio between this expected number and the number of erasures performed by our simulator, for each FTL and workload. Our model assumes that within each partition pages are updated uniformly. Thus, as we expected, without hot and cold data separation the difference between the expected and measured number of erasures is quite high. Adding p and f to the model, reflecting its non-uniform distribution, increases its accuracy considerably. We observe the highest accuracy in the synthetic Zipf workloads, where the hot and cold classification was ideal. Even in these cases, the expected number of erasures was up to 25% higher than the measured one, due to the uniformity assumption in the large cold partition.

The uniformity assumption affects the accuracy of our model for both the baseline and the reprogramming FTLs. We next examine how accurate it is in predicting the benefit from reprogramming,

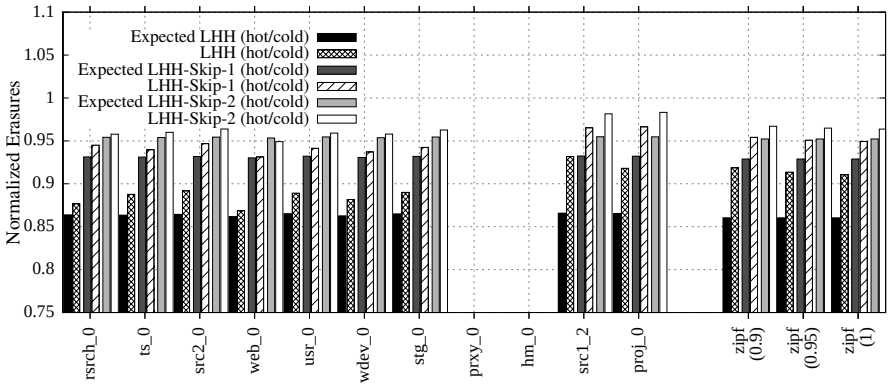

Fig. 15. Expected reduction in erasures and measured reduction. The number of erasures is normalized to that of the baseline system.

in terms of reduction in erasures. Figure 15 depicts this benefit as the number of erasures with reprogramming normalized to that of the baseline. We show the expected benefit, calculated with the erasure factor obtained from our model, and the observed one. The difference between the expected and observed benefits was around of 2%, and no more than 5% in all but the two "coldest" traces, src1_2 and proj_0. The difference with ρ=7% was slightly higher, but was below 10% for all but the src1_2 and proj_0 traces. The measured benefit was lower than expected because of the restriction on invalid low pages corresponding to the reprogrammed high pages. More accurate prediction can be obtained by refining our calculation according to Remark 5.6., which is part of our future work.

The accuracy of this model is considerably higher than the best case analysis of ReusableSSD [Yadgar et al. 2015b]. There, an expected reduction of 33% was based on the assumption that all the pages on a reused block have been invalidated and can be reused without additional hardware limitations.

6.4. Implications for System Design

The smaller number of garbage collection cycles when using WOM codes and the higher stress induced by rewriting pages lead to a tradeoff in FTL design. The following model summarizes the interplay between the physical chip characteristics, the FTL scheme, and the workload. Let E be the number of erasures incurred by M logical page writes in the baseline FTL, and let E' be the number of erasures in a reprogramming FTL for the same M logical writes. We define $R_{erasures}$ as $\frac{E'}{E}$, and use it to derive the amount of logical data that would incur E erasures in the reprogramming FTL: $M' = \frac{M}{R_{erasures}}$. Let $R_{reduction}$ be the reduction in a chip's lifetime due to reprogramming.

In realistic scenarios, a reprogramming FTL will reuse blocks only at the beginning of their lifetime. Let P_{reuse} be the portion of a block's lifetime in which it is reprogrammed, $0 \leq P_{reuse} \leq 1$. We calculate the lifetime extension achievable by reprogramming as follows. We assume that the baseline FTL writes a total of M_{total} logical pages in its entire lifetime. The total number of logical pages that can be written by a reprogramming FTL is given by

$$\begin{aligned} M_{reuse} &= P_{reuse} \cdot M'_{total} + (1 - P_{reuse} - R_{reduction}) \cdot M_{total} \\ &= M_{total} \cdot (\frac{P_{reuse}}{R_{erasures}} + 1 - P_{reuse} - R_{reduction}), \end{aligned}$$

and the lifetime extension is

$$\frac{M_{reuse}}{M_{total}} - 1 = P_{reuse} \cdot (\frac{1}{R_{erasures}} - 1) - R_{reduction}.$$

Table XII. Lifetime extension in sample reprogramming scenarios.

Reprogramming mode	P_{reuse}	$R_{erasures}$	$R_{reduction}$	$P_{reuse} \cdot (\frac{1}{R_{erasures}} - 1) - R_{reduction}$
Reusable SSD on a theoretical device	0.4	0.67	0	$0.2 \rightarrow$ 20% lifetime increase
LLH-FTL on a B29 chip	0.6	0.84	0.085	$0.03 \rightarrow$ 3% lifetime increase
$\overline{\text{LHH}}$-Skip-2 on a C19 chip	0.6	0.95	0.042	-0.01 $\rightarrow$ 1% lifetime reduction

Table XII summarizes the details of three reprogramming examples. In the first example, we consider the Reusable SSD, which achieves a reduction of 33% in erasures by reprogramming all the block's pages [Yadgar et al. 2015b]. In this example we assume that the increased wear due to reprogramming does not reduce the chip's lifetime, and that reprogramming is limited to 40% of this lifetime due to ECC constraints. The lifetime extension in this case is 20%.

In the second example, we consider the LLH-FTL on a device using B29 chips, which reuses blocks in the first 60% of their lifetime. The reduction in erasures is the maximal observed in the evaluation of LLH-FTL in simulation [Margaglia et al. 2016], and the lifetime reduction is taken from Table V. The lifetime extension in this case is 3%. In the third example, we consider a $\overline{\text{LHH}}$-Skip-2 FTL on a device using C19 chips. The reduction in erasures is the maximal observed in our simulations in Figure 13, and the lifetime reduction is taken from Table VI. This use case results in a lifetime *reduction* of 1%: the reduction in lifetime due to the increased wear outweighs the lifetime extension achieved by reprogramming in the first part of the chip's lifetime.

This analysis demonstrates the sensitivity of the reprogramming approach to the physical characteristics of flash chips and to the characteristics of the workloads. When the workload has a low percentage of hot writes (like proj_0), or when the chip is highly sensitive to increased V_{th} (like C19), reprogramming should likely be avoided. The combination of our flash evaluation and theoretical model can provide a good indication which reprogramming scheme will be most useful in increasing a given device's lifetime.

7. RELATED WORK

Several studies proposed FTL designs that reuse pages to extend SSD lifetime. Some are based on capacity achieving codes, and bound the resulting capacity loss by limiting second writes to several blocks [Odeh and Cassuto 2014] or by assuming the logical data has been compressed by the upper level [Jagmohan et al. 2010]. The overheads and complexities in these designs are addressed in the design of ReusableSSD [Yadgar et al. 2015b]. However, none of these studies addressed the limitations of reprogramming MLC flash pages. Some of these limitations were addressed in the design of an overwrite compatible B^+-tree data structure, assuming the mapping of V_{th} to bits can be modified [Kaiser et al. 2013]. Like the previous approaches, it has been implemented only in simulation. Extended P/E cycles [Margaglia and Brinkmann 2015] were implemented on real hardware, but the FTL that uses them relies on the host to supply data that is overwrite compatible. LLH-FTL and $\overline{\text{LHH}}$-FTL are the first general-purpose FTLs that addresses all practical limitations of WOM codes as well as MLC flash. Thus, we were able to demonstrate their strengths and weaknesses on real hardware and workloads.

BER characterization is important for understanding the limitations of flash and the properties of noise, which lead to more efficient algorithms to optimize NAND flash performance. Numerous studies explored the contributors to BER in flash, on a wide variety of chip technologies and manufacturers. They show the effects of erasures, retention, program disturbance and scaling down technology on the BER [Cai et al. 2013; Grupp et al. 2009; Mielke et al. 2008; Yaakobi et al. 2012a]. These studies demonstrate a trend of increased BER as flash feature sizes scale down, and the need for specialized optimizations employed by manufacturers as a result. Thus, we believe that some of the interference effects observed in our experiments are a result of optimizing the chips for regular LH programming. Adjusting these optimizations to LLH or $\overline{\text{LHH}}$ reprogramming is a potential approach to increase the benefit from page reuse.

Several studies examined the possibility of reprogramming flash cells. Most used either SLC chips [Jagmohan et al. 2010], or MLC chips as if they were SLC [En Gad et al. 2015]. A thorough

study on 50nm and 72nm MLC chips demonstrated that after a full use of the block (LH programming), half of the pages are "WOM-safe" [Grupp et al. 2009]. However, they do not present the exact reprogramming scheme, nor the problems encountered when using other schemes. A recent study [Margaglia and Brinkmann 2015] mapped all possible state transitions with reprogramming on a 35nm MLC chip, and proposed the LLH reprogramming scheme. Our results in Section 3 show that smaller feature sizes impose additional restrictions on reprogramming, but that LLH and $\overline{\text{LHH}}$ reprogramming are still possible.

Numerous methods have been suggested for improving the applicability of existing WOM codes. Recent studies have focused on improving their computation efficiency [En Gad et al. 2015], reducing their overhead [Shpilka 2013; 2014; Yaakobi et al. 2012b; Burshtein and Strugatski 2013; En Gad et al. 2015], and reducing their failure probability [Burshtein and Strugatski 2013; En Gad et al. 2015]. Most of these codes target the basic model of one bit per cell with the only restriction that zeroes cannot be overwritten by ones. The additional limitations on reprogramming that were demonstrated in our analysis lay the ground for investigating WOM models that reflect these limitations.

Previous studies examined the energy consumption of flash chips as a factor of the programmed pattern and page [Mohan et al. 2013], and suggested methods for reducing the energy consumption of the flash device [Salajegheh et al. 2011]. To the best of our knowledge, this study is the first to measure the effect of reprogramming on the energy consumption of a real flash chip and incorporate it into the evaluation of the FTL.

Recent trends in flash technologies, such as one-shot programming and 3D V-NAND [Im et al. 2015], eliminate the constraints on the programming order of pages in each block. This may allow reprogramming pages on a fully used block, and maybe even allow reprogramming of the low and high pages alike. To understand their implications, these technologies should be examined in an evaluation similar to ours.

Our theoretical analysis for non-uniform workloads adopts the approach of previous studies that assume pages are partitioned according to their temperature. Desnoyers [Desnoyers 2014] gives and exact analytical solution for the least recently written (LRW) garbage collection policy as well as an approximate solution for greedy garbage collection for two temperatures, and presents an extension to multiple partitions. Van Houdt [Hou 2014] studies the reduction in write amplification achieved by hot and cold data separation, taking into account false identification and dynamically changing workloads. Stoica and Ailamaki [Stoica and Ailamaki 2013] study the classification accuracy required for minimizing write amplification. Our work complement these results by providing an analytical model for optimal partitioning and garbage collection for reprogramming FTLs, that can also be used to predict the benefit from reprogramming under realistic hardware and system limitations.

8. CONCLUSIONS

Our study is the first to evaluate the possible benefit from reusing flash pages with WOM codes on real flash chips combined with an end-to-end FTL analysis. We showed that page reuse in MLC flash is possible, but can utilize at most half of the pages in each block, and achieves this maximum only if some of its capacity has been reserved in advance. While reprogramming is safe for at least 40% of the lifetime of the chips we examined, it incurs additional *long-term* wear on their blocks. Thus, even with an impressive 20% reduction in *erasures*, the increase in *lifetime* strongly depends on chip physical characteristics, and is fairly modest.

Our hardware evaluation exposed a considerable gap between the previously shown benefits of page reuse, which were based on theoretical analysis and simulations, and those that can be achieved on current state-of-the-art hardware. Our detailed theoretical model bridges this gap by providing a framework for maximizing and estimating the benefits of page reuse with validated accuracy.

While our study demonstrates restrictive limitations on page reuse, it also highlights the potential benefits and several approaches to achieve them. First, we believe that many limitations can be addressed with manufacturer support by reevaluating current MLC programming constraints. Sec-

ond, we expect that WOM codes that are specifically designed for the reprogramming limitations of NAND flash chips will allow the reuse of more pages in each blocks. Finally, special-purpose FTLs and data structures that are overwrite compatible will not be limited by WOM encoding overhead and can thus realize the full potential of page reuse.

Acknowledgments

We thank Hila Arobas for her help with the low-level experiments, and the anonymous reviewers for their suggestions that helped improve this manuscript. This work was supported in part by United States-Israel Binational Science Foundation (BSF) grant 2010075, NSF grant CCF-1218005, Israel Science Foundation (ISF) Grant 1624/14, EU Marie Curie Initial Training Network SCALUS grant 238808, and German-Israeli Foundation for Scientific Research and Development (GIF) grant I-1356-407.6/2016.

REFERENCES

2014. NAND Flash Memory Tester (SigNASII). http://www.siglead.com/eng/innovation_signas2.html. (2014).

2014. On the necessity of hot and cold data identification to reduce the write amplification in flash-based {SSDs}. *Performance Evaluation* 82 (2014), 1 – 14.

2014. SNIA IOTTA. http://iotta.snia.org/traces/388. (2014). Retrieved: 2014.

2015. Jasmine OpenSSD Platform. http://www.openssd-project.org/. (2015).

Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design Tradeoffs for SSD Performance. In *USENIX Annual Technical Conference (ATC)*.

A Berman and Y. Birk. 2013. Retired-page utilization in write-once memory – A coding perspective. In *IEEE International Symposium on Information Theory (ISIT)*.

David Burshtein. 2015. Coding for asymmetric side information channels with applications to polar codes. In *IEEE International Symposium on Information Theory (ISIT)*.

David Burshtein and Alona Strugatski. 2013. Polar Write Once Memory Codes. *IEEE Transactions on Information Theory* 59, 8 (2013), 5088–5101. http://dblp.uni-trier.de/db/journals/tit/tit59.html#BurshteinS13

Werner Bux and Ilias Iliadis. 2010. Performance of Greedy Garbage Collection in Flash-based Solid-state Drives. *Perform. Eval.* 67, 11 (Nov. 2010), 1172–1186.

Yu Cai, O. Mutlu, E.F. Haratsch, and Ken Mai. 2013. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *31st IEEE International Conference onComputer Design (ICCD)*.

Mong-Ling Chiao and Da-Wei Chang. 2011. ROSE: A Novel flash Translation Layer for NAND flash Memory Based on Hybrid Address Translation. *IEEE Trans. Comput.* 60, 6 (2011), 753–766. DOI:http://dx.doi.org/10.1109/TC.2011.67

Grard D. Cohen, Philippe Godlewski, and Frans Merkx. 1986. Linear binary code for write-once memories. *IEEE Transactions on Information Theory* 32, 5 (1986), 697–700. http://dblp.uni-trier.de/db/journals/tit/tit32.html#CohenGM86

John Colgrove, John D. Davis, John Hayes, Ethan L. Miller, Cary Sandvig, Russell Sears, Ari Tamches, Neil Vachharajani, and Feng Wang. 2015. Purity: Building Fast, Highly-Available Enterprise Flash Storage from Commodity Components. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*.

Peter Desnoyers. 2013. What Systems Researchers Need to Know about NAND Flash. In *5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*.

Peter Desnoyers. 2014. Analytic Models of SSD Write Performance. *Trans. Storage* 10, 2, Article 8 (March 2014), 25 pages. DOI:http://dx.doi.org/10.1145/2577384

E. En Gad, Huang W., Y. Li, and J. Bruck. 2015. Rewriting flash memories by message passing. In *IEEE International Symposium on Information Theory (ISIT)*.

Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. 2009. Characterizing flash Memory: Anomalies, Observations, and Applications. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

Aayush Gupta, Raghav Pisolkar, Bhuvan Urgaonkar, and Anand Sivasubramaniam. 2011. Leveraging Value Locality in Optimizing NAND flash-based SSDs. In *9th USENIX Conference on File and Storage Technologies (FAST)*.

X.-Y. Hu and R. Haas. 2010. The fundamental limit of flash random write performance: Understanding, analysis and performance modelling. *IBM Res. rep. RZ 3771, IBM Research - Zurich* (2010), 1–19.

Sai Huang, Qingsong Wei, Jianxi Chen, Cheng Chen, and Dan Feng. 2013. Improving flash-based disk cache with Lazy Adaptive Replacement. In *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*.

Jae-Woo Im and others. 2015. A 128Gb 3b/cell V-NAND Flash Memory with 1Gb/s I/O Rate. In *IEEE International Solid-State Circuits Conference (ISSCC)*.

Soojun Im and Dongkun Shin. 2010. ComboFTL: Improving Performance and Lifespan of MLC flash Memory Using SLC flash Buffer. *J. Syst. Archit.* 56, 12 (Dec. 2010), 641–653.

A. N. Jacobvitz, R. Calderbank, and D. J. Sorin. 2012. Writing cosets of a convolutional code to increase the Lifetime of flash memory. In *50th Annual Allerton Conference on Communication, Control, and Computing*.

A. Jagmohan, M. Franceschini, and L. Lastras. 2010. Write amplification reduction in NAND Flash through multi-write coding. In *26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*.

Xavier Jimenez, David Novo, and Paolo Ienne. 2014. Wear Unleveling: Improving NAND flash Lifetime by Balancing Page Endurance. In *12th USENIX Conference on File and Storage Technologies (FAST)*.

Jürgen Kaiser, Fabio Margaglia, and André Brinkmann. 2013. Extending SSD Lifetime in Database Applications with Page Overwrites. In *6th International Systems and Storage Conference (SYSTOR)*.

Taeho Kgil, David Roberts, and Trevor Mudge. 2008. Improving NAND flash Based Disk Caches. In *35th Annual International Symposium on Computer Architecture (ISCA)*.

Hyojun Kim and Seongjun Ahn. 2008. BPLRU: A Buffer Management Scheme for Improving Random Writes in flash Storage. In *6th USENIX Conference on File and Storage Technologies (FAST)*.

Xiang Luojie, B. M. Kurkoski, and E. Yaakobi. 2012. WOM codes reduce write amplification in NAND Flash memory. In *IEEE Global Communications Conference (GLOBECOM)*.

Fabio Margaglia and André Brinkmann. 2015. Improving MLC flash performance and endurance with extended P/E cycles. In *IEEE 31st Symposium on Mass Storage Systems and Technologies (MSST)*.

Fabio Margaglia, Gala Yadgar, Eitan Yaakobi, Yue Li, Assaf Schuster, and André Brinkmann. 2016. The Devil is in the Details: Implementing Flash Page Reuse with WOM Codes. In *14th Usenix Conference on File and Storage Technologies (FAST)*.

N. Mielke, T. Marquart, Ning Wu, J. Kessenich, H. Belgal, Eric Schares, F. Trivedi, E. Goodness, and L.R. Nevill. 2008. Bit error rate in NAND Flash memories. In *Reliability Physics Symposium (IRPS). IEEE International*.

Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random write considered harmful in solid state drives. In *10th USENIX Conference on File and Storage Technologies (FAST)*.

V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M.R. Stan, and S. Swanson. 2013. Modeling Power Consumption of NAND flash Memories Using FlashPower. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 7 (July 2013), 1031–1044. DOI:http://dx.doi.org/10.1109/TCAD.2013.2249557

Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write Off-loading: Practical Power Management for Enterprise Storage. *Trans. Storage* 4, 3, Article 10 (Nov. 2008), 23 pages. DOI:http://dx.doi.org/10.1145/1416944.1416949

S. Odeh and Y. Cassuto. 2014. NAND flash architectures reducing write amplification through multi-write codes. In *IEEE 30th Symposium on Mass Storage Systems and Technologies (MSST)*. DOI:http://dx.doi.org/10.1109/MSST.2014.6855549

Yongseok Oh, Jongmoo Choi, Donghee Lee, and Sam H. Noh. 2012. Caching Less for Better Performance: Balancing Cache Size and Update Cost of flash Memory Cache in Hybrid Storage Systems. In *10th USENIX Conference on File and Storage Technologies (FAST)*.

Heejin Park, Jaeho Kim, Jongmoo Choi, Donghee Lee, and S.H. Noh. 2015. Incremental redundancy to reduce data retention errors in flash-based SSDs. In *IEEE 31st Symposium on Mass Storage Systems and Technologies (MSST)*.

Ki-Tae Park, Myounggon Kang, Doogon Kim, Soon-Wook Hwang, Byung Yong Choi, Yeong-Taek Lee, Changhyun Kim, and Kinam Kim. 2008. A Zeroing Cell-to-Cell Interference Page Architecture With Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories. *IEEE Journal of Solid-State Circuits* 43, 4 (April 2008), 919–928.

R. L. Rivest and A. Shamir. 1982. How to Reuse a Write-Once Memory. *Inform. and Contr.* 55, 1-3 (Dec. 1982), 1–19.

Mastooreh Salajegheh, Yue Wang, Kevin Fu, Anxiao Jiang, and Erik Learned-Miller. 2011. Exploiting Half-wits: Smarter Storage for Low-power Devices. In *9th USENIX Conference on File and Stroage Technologies (FAST)*.

Mohit Saxena, Michael M. Swift, and Yiying Zhang. 2012. FlashTier: A Lightweight, Consistent and Durable Storage Cache. In *7th ACM European Conference on Computer Systems (EuroSys)*.

Amir Shpilka. 2013. New Constructions of WOM Codes Using the Wozencraft Ensemble. *IEEE Transactions on Information Theory* 59, 7 (2013), 4520–4529. http://dblp.uni-trier.de/db/journals/tit/tit59.html#Shpilka13

Amir Shpilka. 2014. Capacity achieving multiwrite WOM codes. *IEEE Transactions on Information Theory* 60, 3 (2014), 1481–1487.

Kent Smith. 2013. Understanding SSD over-provisioning. *EDN Network* (January 2013). http://www.edn.com/design/systems-design/4404566/1/Understanding-SSD-over-provisioning

Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. 2010. Extending SSD Lifetimes with Disk-based Write Caches. In *8th USENIX Conference on File and Storage Technologies (FAST)*.

Radu Stoica and Anastasia Ailamaki. 2013. Improving Flash Write Performance by Using Update Frequency. *Proc. VLDB Endow.* 6, 9 (July 2013), 733–744.

E. Yaakobi, L. Grupp, P.H. Siegel, S. Swanson, and J.K. Wolf. 2012a. Characterization and error-correcting codes for TLC flash memories. In *International Conference on Computing, Networking and Communications (ICNC)*.

Eitan Yaakobi, Scott Kayser, Paul H. Siegel, Alexander Vardy, and Jack K. Wolf. 2012b. Codes for Write-Once Memories. *IEEE Transactions on Information Theory* 58, 9 (2012), 5985–5999. http://dblp.uni-trier.de/db/journals/tit/tit58.html#YaakobiKSVW12

E. Yaakobi, Jing Ma, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf. 2010. Error characterization and coding schemes for flash memories. In *IEEE GLOBECOM Workshops (GC Wkshps)*.

Eitan Yaakobi, Alexander Yucovich, Gal Maor, and Gala Yadgar. 2015. When Do WOM Codes Improve the Erasure Factor in Flash Memories?. In *IEEE International Symposium on Information Theory (ISIT)*.

Gala Yadgar and Moshe Gabel. 2016. Avoiding the Streetlight Effect: I/O Workload Analysis with SSDs in Mind. In *8th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*.

Gala Yadgar, Roman Shor, Eitan Yaakobi, and Assaf Schuster. 2015a. It's Not Where Your Data is, It's How It Got There. In *7th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*.

Gala Yadgar, Eitan Yaakobi, and Assaf Schuster. 2015b. Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes. In *13th USENIX Conference on File and Storage Technologies (FAST)*.

Gala Yadgar, Alexander Yucovich, Hila Arobas, Eitan Yaakobi, Yue Li, Fabio Margaglia, André Brinkmann, and Assaf Schuster. 2016. *Limitations on MLC Flash Page Reuse and its Effects on Durability*. Technical Report CS-2016-02. Computer Science Department, Technion.

Jingpei Yang, Ned Plasson, Greg Gillis, and Nisha Talagala. 2013. HEC: Improving Endurance of High Performance Flash-based Cache Devices. In *6th International Systems and Storage Conference (SYSTOR)*.

Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *11th USENIX Conference on File and Storage Technologies (FAST)*.

APPENDIX

PROOF OF THEOREM 5.5. As before, we assign $Y_1 = \gamma_1 \cdot Z$ and $Y_2 = \gamma_1 \cdot Z$ is the expected number of valid pages when a reused block is physically erased. We use the same notation from the previous theorem. In this system, when a block is reused, it has Y_1 valid pages and under the uniformity we assume that half of them are high and the other half are low, and therefore there are $(Z - Y_1)$ high pages which are available for reprogramming, but we reuse only $1/S$ of them. Thus, the total number of valid pages that the block will have after it is rewritten is $Y_1 + (Z - Y_1)/2S = \frac{Z+(2S-1)Y_1}{2S}$. We get the following equations:

$$\begin{aligned}&(Y_1+1)N_1(Y_1+1) = \cdots = ZN_1(Z)\\ =&(Y_2+1)N_2(Y_2+1) = \cdots = \frac{Z+(2S-1)Y_1}{2S}N_2(\frac{Z+(2S-1)Y_1}{2S}).\end{aligned}$$

The number of block erasures is

$$E = \frac{L + EY_2}{Z + (Z - Y_1)/(2S)}$$

or

$$E = \frac{L}{(1+\frac{1}{2S})Z - Y_2 - \frac{Y_1}{2S}} = \frac{L}{Z} \cdot \frac{1}{1 + \frac{1}{2S} - \frac{\gamma_1}{2S} - \gamma_2}.$$

Hence, the erasure factor, as a function of both γ_1 and γ_2 is $\frac{1}{1+\frac{1}{2S}-\frac{\gamma_1}{2S}-\gamma_2}$.

Again as before we have

$$\begin{aligned} T/Z = &= (Y_1+1)N_1(Y_1+1)\left(\sum_{i=Y_1+1}^{Z}\frac{1}{i}+\sum_{i=Y_2+1}^{\frac{Z+(2S-1)Y_1}{2S}}\frac{1}{i}\right) \\ &\approx (Y_1+1)N_1(Y_1+1)\left(\ln\left(\frac{Z}{Y_1}\right)+\ln\left(\frac{Z+(2S-1)Y_1}{2SY_2}\right)\right) \\ &= (Y_1+1)N_1(Y_1+1)\ln\left(\frac{1+(2S-1)\gamma_1}{2S\gamma_1\gamma_2}\right), \end{aligned}$$

and

$$\begin{aligned} U &= \sum_{i=0}^{Z} iN_1(i)+\sum_{i=0}^{Z} iN_2(i) \\ &= \sum_{i=Y_1+1}^{Z} iN_1(i)+\sum_{i=Y_2+1}^{\frac{Z+(2S-1)Y_1}{2S}} iN_2(i) \\ &= \left((1+\frac{1}{2S})Z-\frac{Y_1}{2S}-Y_2\right)(Y_1+1)N_1(Y_1+1). \end{aligned}$$

Hence

$$\frac{U}{(1+\frac{1}{2S})Z-\frac{Y_1}{2S}-Y_2}=\frac{T/Z}{\ln\left(\frac{1+(2S-1)\gamma_1}{2S\gamma_1\gamma_2}\right)},$$

or

$$\alpha=\frac{(1+\frac{1}{2S})-\frac{\gamma_1}{2S}-\gamma_2}{\ln\left(\frac{1+(2S-1)\gamma_1}{2S\gamma_1\gamma_2}\right)},$$

that is

$$\gamma_2=-\alpha W\left(-\frac{1}{\alpha}e^{\ln\left(\frac{1+(2S-1)\gamma_1}{2S\gamma_1}\right)+\frac{\gamma_1-(2S+1)}{2S\alpha}}\right). \tag{12}$$

Hence, the erasure factor, as a function of α and γ_1 is

$$EF'_S(\alpha,\gamma_1)=\frac{1}{1+\frac{1}{2S}-\frac{\gamma_1}{2S}-\gamma_2},$$

where γ_2 is given by (12). Lastly, since we can choose the threshold γ_1, the value $EF_S(\alpha)$ is achieved by minimizing the value of $EF'_S(\alpha,\gamma_1)$ under the condition in (12). □