Diversifying Wear Index for MLC NAND Flash Memory to Extend the Lifetime of SSDs

Yeong-Jae Woo
Samsung Electronics Co.
Hwaseong, South Korea
yeongjae.woo@samsung.com

Jin-Soo Kim
Sungkyunkwan University
Suwon, South Korea
jinsookim@skku.edu

ABSTRACT

NAND flash-based solid state drives (SSDs) are replacing magnetic disks because of their fast random access performance, shock resistance, and low power consumption. However, the number of program and erase cycles that can be performed on NAND flash is limited. To overcome this limitation, SSDs require a sophisticated wear-leveling algorithm which distributes program/erase cycles evenly across all flash blocks. While most of existing wear-leveling algorithms are only based on the erase counts of flash blocks, our empirical study indicates that the erase count alone is not a good *wear index* which tells us how much a flash block is worn out.

This paper proposes a new wear index for MLC NAND flash memory which takes into account more diverse properties of NAND flash memory. To show the effectiveness of the proposed wear index, we also develop a wear-leveling algorithm, named *Equalizer*. In our evaluation with three realistic workloads, Equalizer based on the proposed wear index improves the effective lifetime of SSDs by up to 145% compared to the existing wear-leveling technique based on the erase count.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Memory technologies*

General Terms

Design, Reliability, Measurement

Keywords

NAND flash memory, wear-leveling, lifetime, reliability, storage system

1. INTRODUCTION

NAND flash-based storage devices are being widely used from MP3 players, digital cameras, and mobile phones to laptops, desktops, and large-scale server systems due to their small form factor, fast random access performance, shock resistance, noiselessness, and low power consumption. A NAND flash memory chip consists of an array of flash blocks, each of which is comprised of a number of pages. Within a flash block, the program operation should be preceded by the erase operation as in-place update is not allowed in NAND flash memory. In order to overcome this *erase-before-write* limitation, flash memory requires a software layer called flash translation layer (FTL) to maximize performance and lifetime [6, 8, 10, 11].

To reduce the cost of flash memory, NAND manufacturers are aggressively shrinking down the cell size and advancing their technologies from single-level cell (SLC) to multi-level cell (MLC) which stores more than one bit per cell. Unfortunately, shrinking the cell size and storing multiple bits per cell lead to undesirable situations where the bit error rate is severely increased and a lesser number of program and erase operations can be performed on each flash block. In modern flash-based storage devices, bit errors are corrected by powerful error correction codes (ECCs) such as Reed-Solomon and BCH up to a certain number of bits.

NAND manufacturers usually specify the number of program/erase cycles that can be performed on a flash block without error in the data sheet. We call this number the *guaranteed program/erase (P/E) cycle* of the NAND flash chip. If a flash block is used beyond the guaranteed P/E cycle, NAND manufacturers do not ensure the reliability of the flash block. Unreliable flash blocks may result in error during later read, program, or erase operations. The actual value of the guaranteed P/E cycle varies from NAND chip to NAND chip, depending on the cell size, the type of flash memory, the manufacturing process, etc.

As defined in [11], we use the term "*lifetime*" of flash memory to refer to the amount of total bytes that can be written to flash memory until the first error occurs among flash blocks. The worst-case lifetime can be achieved when program and erase operations are only performed on a single flash block. We can extend the lifetime of flash memory by using all flash blocks uniformly. Hence, existing FTLs adopt a wear-leveling algorithm which attempts to erase all flash blocks evenly to prolong the time we encounter the first error in one of flash blocks. Usually, traditional wear-leveling algorithms are based on the following assumptions: (1) A flash block becomes unreliable suddenly if it is used more than the guaranteed P/E cycle. (2) All flash blocks have the same lifetime, i.e., they all die at the same time.

In our experiments with real flash memory chips, how-

.

978-1-4799-1443-2/13/$31.00

ever, all the flash blocks could perform program and erase operations reliably even beyond the guaranteed P/E cycle. Moreover, each flash block experiences the first error during flash operations at different P/E cycles. This suggests that the erase count is insufficient to accurately measure how much a flash block is worn out and thus, the traditional wear-leveling algorithms based on the erase count alone are meaningless and hard to maximize the lifetime of flash memory.

In this paper, we strive to discover a new measure that predicts the remaining lifetime of a flash block more accurately than the erase count to extend the lifetime of flash-based solid state drives (SSDs). We call this measure the "*wear index*" and diversify it to include not only the erase count but also other intrinsic characteristics of MLC NAND flash memory. In order to demonstrate the effectiveness of the proposed wear index, we develop a new wear-leveling algorithm, called *Equalizer*, that evens out the wear index (instead of the erase count) over the entire flash blocks. In this way, Equalizer defers the time when the first flash block dies. The contributions of this paper can be summarized as follows:

- We reveal the various error characteristics of MLC NAND flash memory according to repeated program, read, and erase operations through in-depth experiments with real flash memory chips.
- We examine several different wear indices based on the intrinsic characteristics of MLC NAND flash memory. As a result of this study, we propose a new wear index which considers the programming latency as well as the conventional erase count.
- We design and implement a dynamic wear-leveling algorithm called Equalizer. Although Equalizer is based on *Rejuvenator* [11], Equalizer outperforms Rejuvenator by up to 36% in three realistic workloads when we use just the erase count for wear-leveling.
- Compared to the erase count-based Rejuvenator, we show that Equalizer extends the effective lifetime of SSDs by up to 145% in three realistic workloads by applying the proposed wear index.

The rest of this paper is organized as follows. Section 2 gives background and motivation. The previous studies on NAND flash memory characteristics and wear-leveling algorithms are described in Section 3. Section 4 provides the various error characteristics of MLC NAND flash memory we found empirically from real flash memory chips. In Section 5, we present the proposed wear index and the Equalizer algorithm. The experimental results are discussed in Section 6. Finally, Section 7 concludes the paper.

2. BACKGROUND

2.1 NAND Flash Memory

NAND flash memory is comprised of a large array of transistors with floating gates [14]. The basic operations of NAND flash memory are read, program, and erase. The program operation is achieved by supplying a high voltage to the control gate so that the electrons trapped into the floating gate prohibit current flow. On the contrary, the erase

Specification		Value
Guaranteed P/E Cycle		3,000 cycles
Blocks Per Chip		4,152 blocks
Unit Size	Page	8KB (+ spare 640 bytes)
	Block	1MB (128 pages/block)
Latency	Page Read	250 (μs)
	Page Program	1,300 (μs)
	Block Erase	1,500 (μs)

Table 1: Specification of a 35nm 2-bit MLC NAND flash memory chip

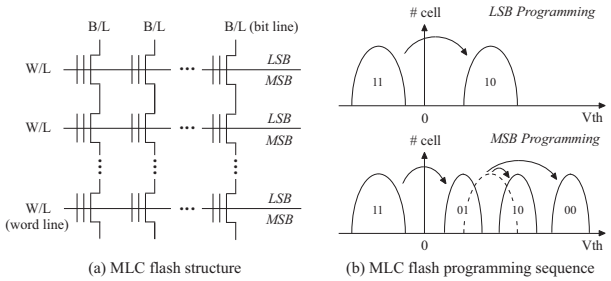

Figure 1: MLC NAND flash memory architecture

operation removes the electrons from the floating gate by applying a high voltage across source and drain. The floating gate which is insulated by oxide gives NAND flash memory the non-volatile characteristic where the electrons trapped into the floating gate do not leak after power off. However, bit errors can occur due to the deterioration of the tunnel oxide over repeated program and erase cycles [12].

A NAND flash memory chip consists of thousands of flash blocks and a single flash block is in turn composed of 64∼128 pages. Each page has extra hundreds of bytes called spare area, which is used for storing ECCs, bad block indicator, etc. The unit of the read and program operation is a page, while the erase operation erases the whole flash block at once. Once a page is programmed, the page cannot be overwritten unless the entire flash block containing the particular page is erased in advance. This characteristic is called the *erase-before-write* limitation of flash memory. There are two flavors of NAND flash memory: single-level cell (SLC) and multi-level cell (MLC). MLC NAND allows to store multiple bits per cell whereas SLC NAND stores only a single bit per cell. Recent flash-based storage devices tend to adopt MLC NAND flash memory, as it provides higher capacity with lower cost. Table 1 presents the specification of a 35nm 2-bit MLC NAND flash chip used in this paper.

As shown in Figure 1(a), MLC NAND flash memory is organized into two types of pages: least significant bit (LSB) page and most significant bit (MSB) page. A single cell logically represents two distinct bits, one in the LSB page and the other in the MSB page, and those two pages that share the same word line (W/L) are called *paired pages*. Within paired pages, the LSB page must be programmed before the MSB page, as illustrated in Figure 1(b). While the LSB page programming only produces two different cell distributions, cells are distributed in four different levels after programming the MSB page that logically correspond to 11, 01, 10, and 00. Since reading or programming MSB pages requires more sophisticated sensing and voltage manipulation, LSB pages exhibit shorter latencies than MSB pages in both read and program operations [5].

2.2 Solid State Drives

Solid state drives (SSDs) are storage devices that support the block I/O interface compatible with hard disk drives (HDDs). Unlike HDDs that are constructed from rotating magnetic disks, SSDs use NAND flash memory as a storage medium. SSDs require a software layer called flash translation layer (FTL) that converts read and write requests from the host into read, program, and erase operations of flash memory. Typically, FTL stores new data into the unwritten page and maintains its mapping information internally. This out-of-place update scheme produces many invalid pages which contain the obsolete data. Recycling these invalid pages is achieved by the *garbage collection (GC)* algorithm in FTL. The GC algorithm first selects a victim flash block to be reclaimed and copies live pages in the victim block into other unwritten pages. When this is done, the victim flash block is erased and then reused by FTL.

The dedicated hardware ECC engine recovers original data from bit errors that may occur in flash memory. During the program operation, the ECC engine generates an ECC code and stores it in the spare area of the page. Whenever a page is read from flash memory, the ECC engine detects and corrects bit errors, if any, using the ECC code located in the spare area. The data still remains corrupted in case of uncorrectable ECC errors, i.e., when the number of bit errors exceeds the maximum number of bits that can be corrected by the ECC engine.

2.3 Guaranteed P/E Cycle and Lifetime

NAND flash memory allows only a limited number of program/erase cycles for a flash block. In the data sheet, NAND flash manufacturers specify the guaranteed P/E cycle, i.e., the minimum number of program/erase cycles that can be performed without any error. For example, the guaranteed P/E cycle is specified as 3,000 for the MLC NAND flash chip used in this paper (cf. Table 1).

We measure the lifetime of flash memory as *total bytes written (TBW)*, the amount of written data until the first error occurs in one of flash blocks [11]. The worst-case lifetime can be obtained when only a single block is programmed and erased repeatedly. Instead, many FTLs adopt wear-leveling algorithms based on the erase count which try to program and erase all flash blocks evenly. In this case, the guaranteed lifetime, $TBW_{guaranteed}$, can be defined as follows under the assumption that no flash blocks are used beyond the guaranteed P/E cycle:

$$TBW_{guaranteed} = BytesPerBlock * NumBlocks * G\_PEcycle, \quad (1)$$

where *BytesPerBlock*, *NumBlocks*, and *G_PEcycle* represent the flash block size, the number of flash blocks, and the guaranteed P/E cycle, respectively. In reality, there is no reason to stop using flash memory just because we reach the guaranteed P/E cycle. Since most FTLs have the facility that can detect the failure of a flash block at run time, the wear-leveling algorithms work fine until one of flash blocks fails. Therefore, the actual lifetime given by the erase count-based wear-leveling algorithm, TBW_{ec_based}, is:

$$TBW_{ec\_based} = BytesPerBlock * NumBlocks * MinPEcycle, \quad (2)$$

where *MinPEcycle* denotes the erase count of the flash block which fails first. In our experiments, however, each flash block has failed at different P/E cycles. For a flash block b, let *PEcycle(b)* be the P/E cycle in which the flash block experiences the first error. If the wear-leveling algorithm knows *PEcycle(b)* exactly, the best-case lifetime of flash memory, TBW_{best}, is calculated as follows:

$$TBW_{best} = \sum_{b=1}^{NumBlocks} BytesPerBlock * PEcycle(b). \quad (3)$$

In Section 4, we show that there is a significant gap between TBW_{ec_based} and TBW_{best}. Therefore, the questions are how we can estimate *PEcycle(b)* accurately for the individual flash block and use them in the wear-leveling algorithm. These are issues tackled in this paper to maximize the lifetime of SSDs.

2.4 The Jasmine OpenSSD Platform

Investigating the error characteristics of NAND flash memory requires a hardware environment that provides an accurate means to measure various parameters such as read latency, program latency, erase latency, bit error count, etc. For this purpose, we use the Jasmine OpenSSD platform available from the OpenSSD project [16]. The OpenSSD project is an open, community-driven project in collaboration with Indilinx, which aims to promote research and education on the SSD technology by providing open SSD hardware platforms.

The Jasmine OpenSSD platform consists of an ARM7TDMI embedded processor, 96KB internal SRAM, and 64MB external mobile SDRAM. The ECC engine supports both Reed-Solomon and BCH with the ability to correct up to 16-bit errors per 512-byte sector. When reading a page, the number of bits corrected by the ECC engine can be reported to the firmware. Although the controller supports various NAND flash memory chips from Samsung, Hynix, Toshiba, Micron, etc., the Jasmine OpenSSD platform currently supports only a single type of NAND flash memory, whose specification is summarized in Table 1.

3. RELATED WORK

There have been many empirical studies to reveal the inherent characteristics of NAND flash memory [4, 5, 7]. Grupp et al. [7] investigate the characteristics of flash memory such as power consumption, reliability, and performance of basic operations, using various flash chips across manufacturers, cell types, and technology nodes. They show that program latencies are shortened, but bit errors are increased exponentially as their flash chips are worn out. Boboila et al. [4] find that the erase latency is increased over program/erase cycles contrary to the program latency. In addition, they perform program and erase cycles over and over again until either of flash operations fails in order to verify the guaranteed P/E cycle. The experimental results show that flash chips survive significantly longer than their guaranteed P/E cycles, even almost hundred times. All of these studies have observed the changes in performance and reliability over repeated program and erase cycles. Our study goes one step further and tries to utilize these properties to predict the actual wear of each flash block more accurately than the traditional measure based on the erase count.

The lifetime of flash memory can be prolonged by enhancing the wear-leveling algorithm. The wear-leveling algorithms are classified into two categories: dynamic wear-leveling and static wear-leveling. Dynamic wear-leveling is

usually performed when new data is written to flash memory, by allocating the clean flash block with the lowest erase count for storing frequently-updated hot data and the clean flash block with the highest erase count for cold data. While dynamic wear-leveling can be achieved without any additional overhead, the flash block containing cold data can remain for a long time even if this block has the lowest erase count. Static wear-leveling compensates this by periodically moving cold data to the clean flash block with the higher erase count at the expense of copying the entire block.

Most wear-leveling algorithms perform both dynamic and static wear-leveling. The dual-pool algorithm [6] groups flash blocks into two pools: hot pool and cold pool. Initially, flash blocks are randomly assigned to one of pools. For dynamic wear-leveling, hot data is stored into the hot pool, and cold data into the cold pool. Whenever the gap between the maximum erase count of the hot pool and the minimum erase count of the cold pool becomes larger than a certain threshold, the static wear-leveling algorithm swaps the corresponding flash blocks between two pools after exchanging their valid data.

Rejuvenator [11] partitions flash blocks into higher-numbered lists (HNLs) and lower-numbered lists (LNLs) based on the erase count. Dynamic wear-leveling is achieved by writing hot data into LNLs and cold data into HNLs. In Rejuvenator, the difference between the maximum erase count and the minimum erase count is kept under the threshold value τ. If the difference becomes larger than τ, Rejuvenator performs static wear-leveling by performing garbage collection on the flash blocks with the minimum erase count in LNLs. The value of τ is decreased as flash blocks get worn out to minimize the wear-leveling overhead in the early stage of the lifetime. Moreover, the sizes of HNLs and LNLs are adaptively expanded or shrunk according to the given workload.

These wear-leveling algorithms, which try to keep the difference in the erase counts of flash blocks below a certain threshold, assume that all the flash blocks fail at the same time. However, our experimental results with real flash memory chips show that each flash block faces an error at different P/E cycles. Thus, we propose a new wear-leveling algorithm which evens out the degree of worn-out among flash blocks instead of the erase count. This enables that long-lived flash blocks receive a relatively larger amount of data than short-lived flash blocks. Consequently, the lifetime of flash memory can be extended much longer than the one the traditional wear-leveling algorithms used to achieve.

4. ERROR CHARACTERISTICS OF MLC NAND FLASH MEMORY

4.1 Methodology

To examine the intrinsic characteristics of flash memory, we have performed a series of experiments with MLC NAND flash chips from a major manufacturer on the Jasmine OpenSSD platform. As shown in Table 1, each chip has 4,152 flash blocks and the guaranteed P/E cycle is specified as 3,000. We measure the performance and reliability of flash blocks over repeated test cycles. Each test cycle for a flash block consists of programming all pages sequentially, verifying them by read operations, and then erasing the block for the next iteration. The data patterns written to flash memory are randomly generated for each program operation.

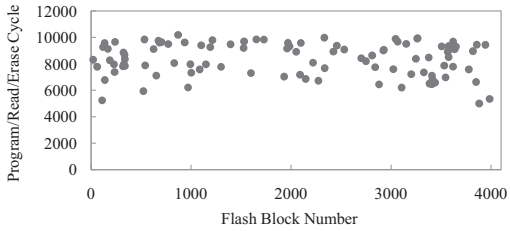

Figure 2: The distribution of the maximum number of P/E cycles for one hundred flash blocks

The operation latencies are measured using the 10ns resolution timer available from the Jasmine OpenSSD platform. For each read operation, we also keep track of the number of bit errors corrected by the ECC engine using the facility provided by the platform. The hardware ECC engine is configured to correct the maximum 16-bit errors per 512-byte sector using the BCH algorithm.

4.2 Lifetime of Flash Blocks

The first experiment we have performed is to obtain the distribution of *PEcycle(b)*, the maximum number of P/E cycles that can be used without any error for each flash block b. We choose one hundred flash blocks randomly within a flash memory chip and run the program, read, and erase cycles on those flash blocks repeatedly until we encounter the first error. As illustrated in Figure 2, each block has failed far beyond the guaranteed P/E cycle of 3,000. From Figure 2, we can see that *PEcycle(b)* is not affected by the location of the flash block, and the actual values of *PEcycle(b)* differ from flash block to flash block. Among the tested flash blocks, the first error has occurred in the flash block #3,882 at 4,999 cycle. The average of *PEcycle(b)* is 8,524 cycles with the standard deviation of 1,318.

Recalling the equations shown in Section 2.3, the guaranteed lifetime of this flash memory chip is as follows.

$$TBW_{guaranteed} = 1MB * 4,152 * 3,000 = 11.88TB. \quad (4)$$

If we generalize the results obtained from one hundred flash blocks into the whole flash memory chip, the actual lifetime achievable by the wear-leveling algorithm based on the erase count is:

$$TBW_{ec\_based} = 1MB * 4,152 * 4,999 = 19.79TB. \quad (5)$$

The lifetime is maximized if the wear-leveling algorithm exhausts each flash block b up to its *PEcycle(b)*. In this case, the lifetime of flash memory is given by

$$TBW_{best} = 1MB * \Sigma_{b=1}^{4,152} PEcycle(b) = 34.42TB. \quad (6)$$

Compared to TBW_{ec_based}, we can see that the lifetime can be improved further by a factor of 1.74 if we utilize all the flash blocks to the maximum.

The traditional wear-leveling algorithms based on the erase count treat all the flash blocks equally, as if they had the same health at the same age (or erase count). As a result, the lifetime of flash memory is bounded by the lifetime of the weakest flash block. Instead, our approach is to give more data to long-lived flash blocks so that all the flash blocks with different *PEcycle(b)* can end their lives almost at the same time. The challenge in this approach is how to estimate the *health* of individual flash block accurately. In the following subsections, we investigate several parameters of NAND

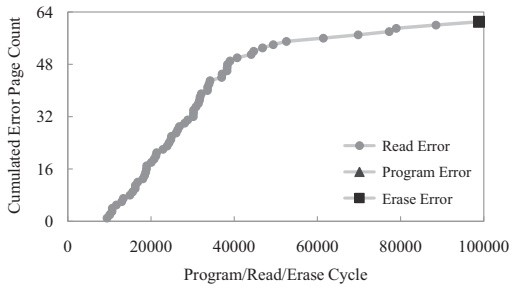

Figure 3: Errors occurred in the flash block #1,102.

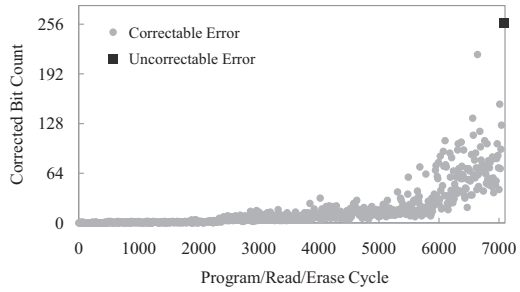

Figure 4: Corrected bit counts in the first MSB page (page #2) within the flash block #1,598.

flash memory that can be monitored at run time and discuss whether they can be used to differentiate flash blocks according to their health levels.

4.3 Error Patterns Within a Flash Block

During the experiment in the previous subsection, we notice that the first error within a flash block has occurred always in read operations. To observe the error patterns after the first error has occurred, we have performed extra program, read, and erase cycles on flash blocks. When a read or program error occurs on a page, we have immediately excluded the page for the remaining test cycles so that it is not programmed or read again. We run the test cycles until the flash block is completely failed, i.e., until the flash block experiences an erase error or all the pages within the flash block have either program errors or read errors. In this way, we have collected error patterns for randomly-chosen one hundred flash blocks.

Figure 3 plots the cumulated number of error pages due to read errors or program errors until the flash block is completely failed by an erase error. We only show the results for the flash block #1,102 as the error patterns in other flash blocks are very similar. Ever since the flash block encounters the first read error at 9,416 cycle, the number of error pages increases steadily until 40,000 cycle from which the increase slows down. Finally, the flash block has failed around 100,000 cycle due to an erase error. Before the flash block failure, almost half of the pages within the block resulted in read errors and most of them were MSB pages.

It is worth noting that no program errors occurred until the flash block dies. We observed the same results across all the flash blocks tested. Considering that the program operation has its own verification steps [15], read errors are caused by the data corrupted *after* they are written, possibly due to program interference from other pages [13].

4.4 Bit Error Count

To better understand read errors, we have collected the number of bit errors on each page as the test cycle proceeds. Figure 4 presents the changes in the number of bits corrected by the ECC engine in the first MSB page (page #2) of the flash block #1,598. Similar to the previous studies [5, 7], the bit error count increases rapidly beyond the guaranteed P/E cycle. Note that each page is composed of 16 sectors and the ECC engine can correct errors up to 16 bits per sector. In the tested flash block, the read error has occurred around 7,000 cycle, which means that more than 16 bits become corrupted in one of the sectors at this point. The same trend was observed in other pages as well.

From Figure 4, it seems that the occurrence of a read error can be predicted by monitoring the increasing number of corrected bits after each read operation. However, the main problem of this approach is that monitoring the bit error count of a page is possible only when there is a read request on the page. An unexpected read error may occur if a page is written continuously with its contents being read very rarely.

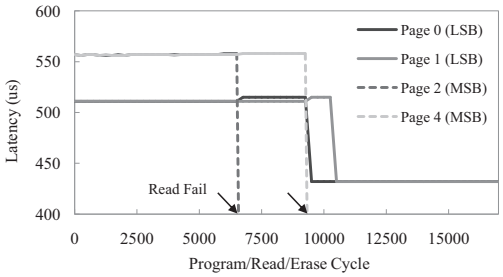

Figure 5: Latency changes in read operations in the flash block #1,598

4.5 Read Latency

Figure 5 shows the latency changes during read operations. We denote the read latency as zero when a read operation terminates with the uncorrectable ECC error. As described in Section 4.3, the first MSB page (page #2) dies first within the flash block and MSB pages are more fragile than LSB pages. Also we can see that reading MSB pages takes longer than LSB pages by almost 10%.

It is somewhat surprising that the read latency remains stable during the lifetime of each page. There is one exception; soon after an MSB page fails, the read latency of the paired LSB page suddenly drops by almost 15%[1]. This is because programming MSB pages narrows the cell margin of the associated LSB pages due to the program interference [13]. This makes the read operation require additional sensing steps to determine the logical value. Conversely, LSB pages can be read faster if the paired MSB pages are no longer programmed.

4.6 Program Latency

Next, we examine the program latency as a function of program, read, and erase cycles. As illustrated in Figure 6(a), the latency of programming an MSB page constantly decreases to around 2,400μs, at which point a read error is encountered during the read operation on the same page. The electrons become easier to be trapped into the floating gate as the cell ages by repeated program and erase cycles [4,

[1]In the NAND flash memory chip used in this paper, page #0 (LSB) and #2 (MSB) form a paired page and so do page #1 (LSB) and #4 (MSB).

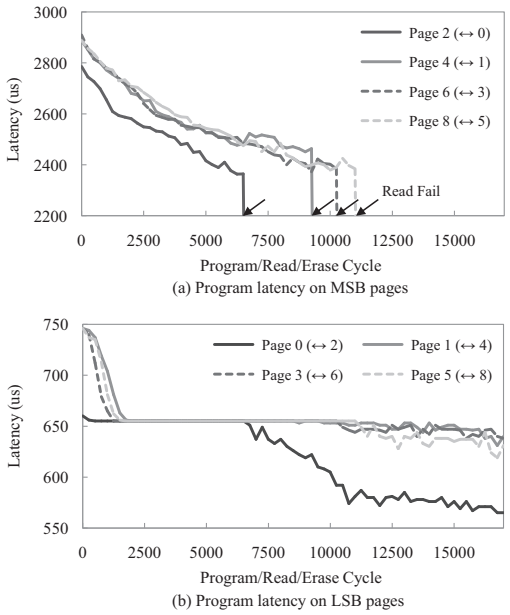

Figure 6: Latency changes in program operations in the flash block #1,598

7]. As a consequence, programming the aged pages requires lesser number of iterations to complete the operation than before. The same phenomenon was observed in other MSB pages across all flash blocks.

Figure 6(b) presents the program latencies for LSB pages. Basically, the program speed of LSB pages is four times faster than that of MSB pages. In contrast with MSB pages, the program latencies of LSB pages only decrease to around 650μs. As soon as MSB pages die due to read errors, the program latencies of the paired LSB pages begin to decrease again with the same reason described in Section 4.5.

Figure 6(a) suggests that the occurrence of a read error on an MSB page can be predicted by monitoring whether the program latency on the page reaches a certain threshold. Since the failure of a flash block is caused by the read error on the first MSB page, this can be a very effective means to measure the current health of a flash block.

4.7 Erase Latency

Finally, Figure 7 depicts the latency changes in erase operations until the flash block #1,598 becomes useless due to an erase error. The erase latency increases from 1,500μs to 7,000μs over the cycles and the overall trend is very similar to the change in the cumulated number of error pages within the flash block. This result is exactly the opposite phenomenon of what we have observed in Section 4.6; as the cell ages, the erase operation requires more iterations to remove the trapped electrons from the floating gate [4].

4.8 Summary

We summarize our findings on the error characteristics of MLC NAND flash memory. First, flash blocks in the same chip have different lifetimes. Second, the main cause of death for flash blocks is the read error on their first MSB pages. Program errors have not been observed, and erase failure occurs long after the first read error. Third, it is likely that the death of a flash block due to the read error can be predicted by monitoring the bit error count during read operations or

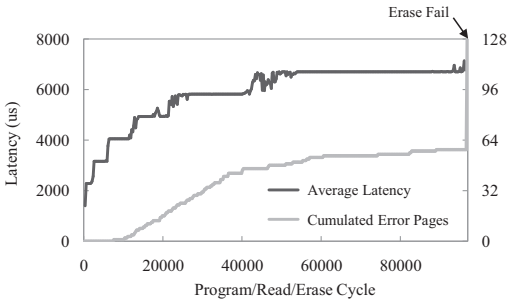

Figure 7: Latency changes in erase operations in the flash block #1,598

the latency changes during program operations. Other alternatives, such as the read latency, the erase latency, and the conventional erase count, are difficult to be used for this purpose. Based on these observations, we define several wear indices in Section 5 that can predict how long each flash block will remain alive.

Due to the restriction of the Jasmine OpenSSD platform, our study is performed only on one type of MLC NAND flash memory chips. Referring to the previous study [7], however, the trends of the flash memory characteristics are similar across various flash memory chips regardless of manufacturers, cell types, etc. We believe that the characteristics of flash memory presented in this section also hold for other MLC NAND flash memory chips, as those characteristics all originate from the basic structure of flash memory (i.e., the transistor with the floating gate insulated by oxide) and the common internal programming algorithm such as incremental step pulse programming (ISPP) [15].

5. EXTENDING THE LIFETIME OF SSDS

5.1 Diversifying Wear Index

We use the term *wear index* to indicate the relative health of each flash block quantitatively. Formally, the wear index $W(b,t)$ is defined as a value in [0, 1], which estimates the degree of wear for the flash block b at t-th P/E cycle. The value of 1.0 indicates it is very likely that the flash block results in error during any of read/program/erase operations. The conventional wisdom was to define the wear index as a function of the erase count alone. However, as described in Section 4, it is too inaccurate to fully utilize the potential lifetime of flash memory. In this subsection, we introduce several wear indices that may be used to predict the lifetime of a flash block more accurately.

Wear index based on the erase count The traditional scheme based on the erase count can be emulated using the notion of wear index. The traditional wear-leveling algorithms assume that the flash block wear is proportional to the erase count (or the P/E cycle) and the flash block becomes fully worn out if the erase count reaches the guaranteed P/E cycle ($G\_PEcycle$). The corresponding wear index can be defined as follows:

$$W_{EC}(b,t) = min(\frac{t}{G\_PEcycle}, 1). \quad (7)$$

This wear index treats all flash blocks with the same erase count equally, and more importantly, does not differentiate one from another once the erase count reaches the guaranteed P/E cycle.

Wear index based on the bit error count As described in Section 4.3, the first error encountered by each flash block was all read errors. A read error occurs when the number of bit errors exceeds the maximum number of bits correctable by the ECC engine. Because the corrected bit count increases during read operations as the flash block wears, we can define the wear index based on the bit error count as follows:

$$W_{Err}(b,t) = max(\frac{N_{CorBit}(b,t)}{N_{MaxCorBit}}, W_{Err}(b,t-1)), \quad (8)$$

where $N_{MaxCorBit}$ is the maximum bit count that can be correctable by the ECC engine and $N_{CorBit}(b,t)$ denotes the maximum number of corrected bits during read operations for the pages in the flash block b. Note that W_{Err} can be updated only when the host reads the data belonging to the flash block b. This weakness can be compensated by adjusting the wear index whenever the valid pages in b are copied to another block during garbage collection. Nonetheless, some flash blocks may still have no opportunity to determine the up-to-date wear index value.

Wear index based on the program latency As shown in Section 4.6, a read error on a page can be also predicted by observing how much the program latency to the page approaches to a certain threshold value. Thus, the wear index based on the program latency can be modeled as follows:

$$W_P(b,t) = max(\frac{T_{MaxProg} - T_{Prog}(b,t)}{T_{MaxProg} - T_{MinProg}}, W_P(b,t-1)), \quad (9)$$

where $T_{MaxProg}$ and $T_{MinProg}$ represent the initial program latency measured at the very beginning and the threshold latency when the pages suffer from read errors, respectively. $T_{Prog}(b,t)$ means the shortest latency during program operations for the pages in the flash block b.

In contrast with W_{Err}, W_P can be updated whenever FTL writes data to the flash block either by the write request from the host or during garbage collection. Hence, W_P can forecast the occurrence of a read error more accurately than W_{Err}. Note that W_P requires the preliminary profiling step for the given flash memory chip to determine the parameters such as $T_{MaxProg}$ and $T_{MinProg}$.

5.2 Proposed Wear Index

To understand the behavior of wear indices defined in the previous subsection, we simulate them using the real data collected from the flash memory chip. Figure 8 plots the changes in their values for the flash block #236 over the repeated program/read/erase cycles. The constant values are set as follows: $G\_PEcycle = 3{,}000$ cycles, $N_{MaxCorBit} = 256$ bits, $T_{MaxProg} = 2{,}894\mu$s, and $T_{MinProg} = 2{,}417\mu$s. In this flash block, a read error has occurred at 7,393 cycle.

W_{EC} increases linearly until the erase count reaches the guaranteed P/E cycle. Afterwards, it is fixed to 1.0. Even if the flash block lives less than half of its life, W_{EC} overestimates that the flash block is completely worn out. On the contrary, W_{Err} tends to underestimate the lifetime; it reaches at most 0.6 even when the flash block dies due to the read error.

Compared to W_{EC} and W_{Err}, W_P is relatively accurate, approaching to 1.0 at the end of the lifetime of this flash block. However, one undesirable property of W_P is that the wear index value grows rapidly at the very early stage of the lifetime. In Figure 8, W_P already shows the value of 0.4 when the flash block is used only for 1,000 cycles. The rapidly increased wear index value may incur unnecessary static wear-leveling overhead at the beginning of the lifetime.

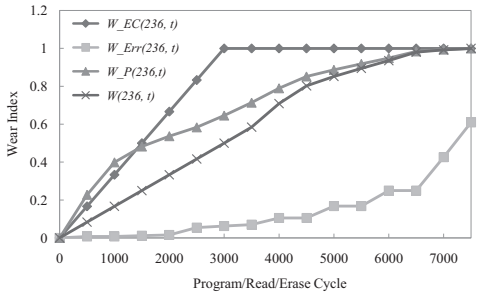

Figure 8: The behavior of diverse wear indices for the flash block #236.

The ideal wear index is the one whose value is linearly increased during the entire lifetime of a flash block, reaching 1.0 when the flash block dies. To model the ideal wear index, this paper proposes the following wear index W:

$$W(b,t) = \beta \cdot W_{EC}(b,t) + (1-\beta) \cdot W_{LogP}(b,t), \quad (10)$$
$$\text{where } W_{LogP}(b,t) = max(0, Log_\alpha W_P(b,t) + 1).$$

The proposed wear index first takes the log of W_P (with the log base α) and then calculates the weighted sum with W_{EC}. W_{LogP} has been introduced for two reasons. First, W_{LogP} forces the wear index to increase more smoothly where α regulates the gradient of W_{LogP}. Second, W_{LogP} is set to zero in the region that $Log_\alpha W_P(b,t) + 1$ has negative values. This corresponds to the region of the lifetime that W_P rapidly grows. Therefore, W_{LogP} makes the value derived from W_P not contribute to W in the early stage. Instead, by taking the weighted sum with W_{EC}, the values of W during this early stage of the lifetime are determined solely by W_{EC}. Figure 8 also shows the graph for the proposed wear index W. We use the values of $\alpha = 1.5$ and $\beta = 0.5$ for W. We can see that the proposed wear index W closely matches the ideal wear index.

5.3 Equalizer

We develop a new wear-leveling algorithm named *Equalizer* that evens out the wear index of flash blocks. Equalizer is largely based on Rejuvenator, the latest wear-leveling algorithm proposed by Murugan and Du [11]. The overall architecture of Equalizer is depicted in Figure 9. First, the hot/cold data detector separates the incoming data into hot data and cold data, using the LRU window with the write address and the access count. Then, hot data is stored into the hot partition, while cold data into the cold partition. Each flash block belongs to either the hot partition or the cold partition according to its wear index. The flash blocks in the hot partition have lower wear indices than those in the cold partition. In each partition, the new data is written to its active log block sequentially. To achieve better garbage collection performance, the hot partition and the cold partition are managed by the page-level mapping [8] and by the hybrid mapping [10], respectively.

Equalizer enhances Rejuvenator in two ways by adopting the *global free list* and the *wear index-aware merge*. The global free list manages free flash blocks that are generated by garbage collection from both partitions. If a partition lacks a free block, the global free list allocates a free block

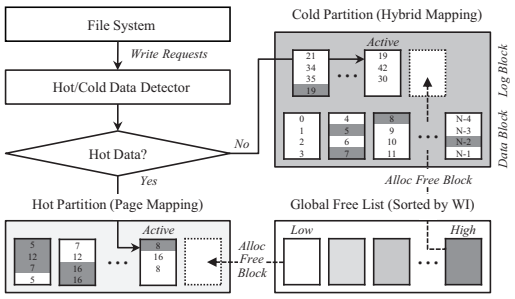

Figure 9: Overall architecture of Equalizer

to the partition. The global free list supplies the free block with the minimum wear index to the hot partition, since the wear index of the flash block in the hot partition is likely to be increased due to its hot data. Similarly, the global free list provides the free block with the maximum wear index for the cold partition. To facilitate this allocation policy, free blocks in the global free list are sorted by the wear index.

After a flash block is allocated from the global free list, each partition triggers garbage collection if the number of remaining free blocks falls below a certain threshold. The victim block to be reclaimed is chosen from the partition who initiates garbage collection. For the hot partition, the greedy policy is used, i.e., the flash block having the minimum number of valid pages is chosen as the victim block. In case of the cold partition, the full merge operation is performed on the victim block, which is chosen among log blocks in a round-robin manner [10].

During this full merge operation, the wear index-aware merge aims to recycle the flash block with the lowest wear index among data blocks. In hybrid mapping, a log block can be associated with a number of data blocks. To merge the log block, the existing algorithm copies the valid pages from one of the associated data blocks into a free block. Then, the data block is used as a free block for the next merge operation. This is repeated until all the valid pages in the log block are merged with the corresponding data blocks. The full merge operation finally produces two free blocks: the victim log block and the data block merged last with the victim log block. If we generate free blocks with lower wear indices during the full merge operation, those blocks can be allocated to the hot partition later, which is helpful for dynamic wear-leveling. Thus, the wear index-aware merge performs the full merge operation with the associated data blocks in a descending order of wear index in order to convert the data block with the lowest wear index into the free block.

For static wear-leveling, Equalizer maintains the gap between the maximum and the minimum wear index among flash blocks below the threshold τ. As the average wear index of flash blocks increases, Equalizer decreases τ to tightly control the gap. If the gap exceeds τ, Equalizer performs garbage collection for the flash blocks that have the minimum wear index. When the wear index is based on the erase count such as W_{EC}, this increases the minimum wear index immediately. However, there is no guarantee that the minimum wear index is increased as a result of garbage collection if other kinds of wear indices are used. For example, the wear index based on the bit error count, W_{Err}, can be unchanged while performing garbage collection. Similarly, the wear index based on the program latency, W_P, is not affected since no program operations are performed on the victim block

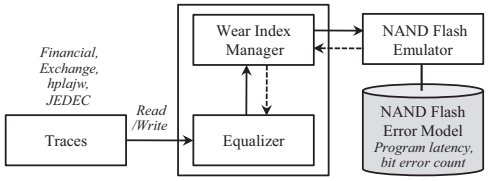

Figure 10: Overview of the SSD Simulator

during garbage collection. In these cases, the gap is not reduced and garbage collection will be performed over and over again. To remedy this problem, we temporarily increase the wear index of the victim block during static wear-leveling. The original wear index is restored and is newly updated naturally when the actual wear index of the flash block is measured later.

6. EVALUATION

6.1 Evaluation Methodology

Since it is estimated that more than 70 days are needed to get completely worn out a 4GB flash memory chip, we have constructed a trace-driven SSD simulator to evaluate the proposed approach. Figure 10 depicts the overall architecture of the simulator.

The NAND Flash Error Model indicates the data we have collected from a real NAND flash chip in Section 4. It includes the information on the program latency and the bit error count for each P/E cycle from one hundred of *real* flash blocks. Based on this model, the NAND Flash Emulator mimics the total 4GB of *virtual* flash memory chip consisting of 4,192 *virtual* flash blocks. It makes the k-th virtual flash block follow the performance and error characteristics of (k%100)-th real flash block stored in the NAND Flash Error Model. Those characteristics are used by the Wear Index Manager to compute the wear index value for each virtual flash block. The Equalizer module performs address mapping, garbage collection, and wear-leveling according to the algorithm described in Section 5.3. Besides Equalizer, we have also implemented Rejuvenator for comparison.

We use disk access traces from three realistic workloads and one synthetic workload. FINANCIAL is the OLTP (On Line Transaction Processing) trace collected from a large financial institution [3]. The EXCHANGE trace was collected for a month from Microsoft Exchange Server which was used for mailing purpose [1]. The HPLAJW trace from HP Lab. includes the personal workstation workload such as sending e-mails and editing documents [2]. Finally, JEDEC is the synthetic workload trace we have generated based on the JESD218 document [9]. This workload is proposed by JEDEC for evaluating the endurance of SSDs, modeling the behavior of enterprise servers.

We use the *effective* lifetime as a metric to evaluate different wear-leveling policies. The effective lifetime is measured as the *total user bytes written (TUBW)*, i.e., the amount of the total *user* data written to flash memory excluding the amount of data written due to garbage collection and wear-leveling. When measuring the effective lifetime, the traces are repeated until one of flash blocks dies.

The SSD simulator exposes only 93% of the total flash memory capacity to the outside world and uses the remaining 7% of flash blocks to enhance the garbage collection efficiency. However, we reduce the total flash memory capacity

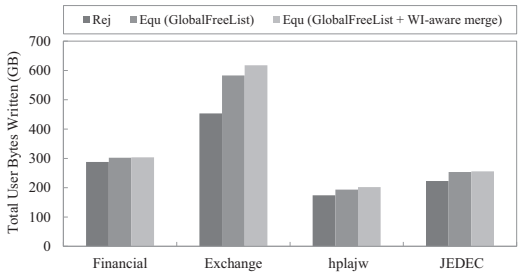

Figure 11: Comparison of Rejuvenator and Equalizer with W_{EC}.

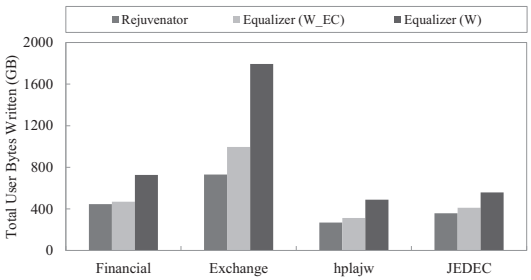

Figure 12: Comparison of Rejuvenator, Equalizer with W_{EC}, and Equalizer with the proposed wear index (W).

to 700MB for the HPLAJW trace since it requires at most 634MB. The LRU window size in Equalizer is set to 1,024. Although the wear index is defined as a real value between 0 and 1, we represent it with an integer value after scaling to the range between 0 and 3,000 to avoid floating-point arithmetic. Initially, τ is set to 300 and it is linearly decreased down to 30 as the maximum wear index value approaches to 3,000. The wear index of the victim flash block is temporarily increased by one during static wear-leveling unless the wear index is based on the erase count (cf. Section 5.3).

6.2 Rejuvenator vs. Equalizer

Before evaluating the effectiveness of the proposed wear index, we first compare the performance of Equalizer with the baseline algorithm, Rejuvenator, in the same condition. This can be done by using the erase count-based wear index (W_{EC}) for Equalizer. For fair comparison, we also use the same parameters for both Rejuvenator and Equalizer. Figure 11 compares the effective lifetimes of four workloads under three different wear-leveling schemes. The leftmost bar is the effective lifetime achieved by Rejuvenator. The middle bar shows the result of Equalizer with the global free list only. Finally, the rightmost bar is the result of Equalizer which implements both the global free list and the wear index-aware merge. In Figure 11, the effective lifetime is measured only for 3,000 cycles – the guaranteed P/E cycle of flash memory.

Since we represent the wear index as an integer value between 0 and 3,000, Equalizer with W_{EC} works exactly the same as Rejuvenator. The difference only comes from the enhancements made by Equalizer: the global free list and the wear index-aware merge. We find that the global free list is quite effective in improving the effective lifetime. On average, Equalizer with the global free list improves the effective lifetime by 15% compared to Rejuvenator. Adding the wear index-aware merge further extends the effective lifetime by 3%. This is because both the global free list and the wear index-aware merge are helpful for dynamic wear-leveling, which tries to allocate a free block with higher wear index to cold data and one with lower wear index to hot data.

In this experiment, the amount of total bytes written by each workload is the same as $TBW_{guaranteed} = 11.88$TB (cf. Section 4.2), since the simulation runs only for the guaranteed P/E cycle. Hence, the difference between $TBW_{guaranteed}$ and the effective lifetime shown in Figure 11 is due to the overhead incurred by garbage collection and wear-leveling.

6.3 Equalizer with the proposed wear index

In this subsection, we examine how much Equalizer with the proposed wear index (W) extends the effective lifetime compared to the conventional wear-leveling algorithms based on the erase count. Figure 12 displays the effective lifetimes for Rejuvenator, Equalizer based on the erase count (W_{EC}), and Equalizer based on the proposed wear index (W). Unlike the previous experiment, we run the simulation until one of flash blocks dies and this is why the effective lifetimes of Rejuvenator and Equalizer with W_{EC} are increased compared to the results shown in Figure 11.

Equalizer with the proposed wear index (W) outperforms Rejuvenator and Equalizer with W_{EC} by 95% and by 63% on average, respectively. The greatest improvement has been made in the EXCHANGE trace; the effective lifetime is extended by 145% compared to Rejuvenator. The total bytes written to flash memory by the proposed approach was 33.71 TB on average, which corresponds to 98% of TBW_{best}, the best-case lifetime described in Section 4.2. This suggests that every flash block is almost fully utilized until near the end of its lifetime, with no one being dead early. Two techniques make this possible. First, the proposed wear index based on the erase count and the program latency estimates the health of each flash block accurately. Second, Equalizer manages flash blocks effectively using the proposed wear index; it tries to write more data to long-lived flash blocks, but less data to short-lived flash blocks to postpone the time they are worn out as much as possible.

6.4 Equalizer based on diverse wear indices

Now we evaluate the impact of diverse wear indices introduced in Section 5.1 on the effective lifetime. Figure 13 illustrates the changes in the effective lifetime under the various wear indices. These results are obtained by measuring the effective lifetime whenever the average erase count of all flash blocks is advanced by one thousand cycles.

With the wear index based on the erase count (W_{EC}), Equalizer always terminates near 4,999 cycle where one or more flash blocks fail according to our NAND flash model. Among the wear indices evaluated in Figure 13, W_{EC} shows the worst effective lifetime for the given average erase count.

The wear index based on the bit error count (W_{Err}) performs even worse than W_{EC}; Equalizer with W_{Err} reaches at most 4,000 cycle, and sometimes only 3,000 cycle as can be seen in the FINANCIAL trace. We find that the FINANCIAL trace contains only 15.4% of read requests, which is the lowest ratio among the workloads. Since W_{Err} can only be updated by read operations, the accuracy of W_{Err} will be impaired if there are not enough read requests in the workload.

In contrast, the wear indices W_P and W, which rely at least in part on the program latency, show relatively good

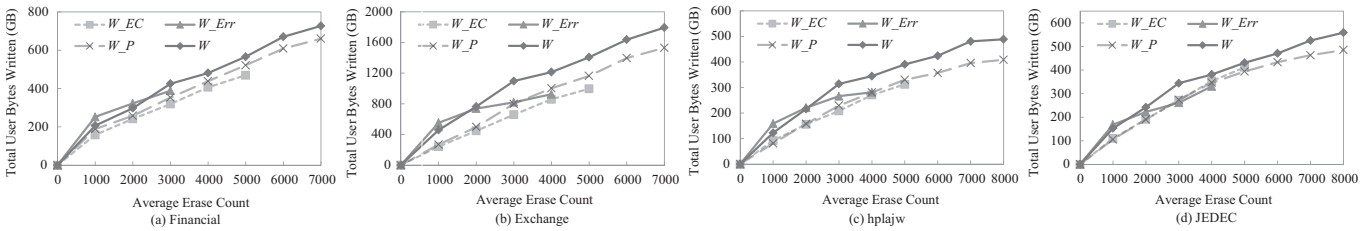

Figure 13: The effective lifetime with diverse wear indices

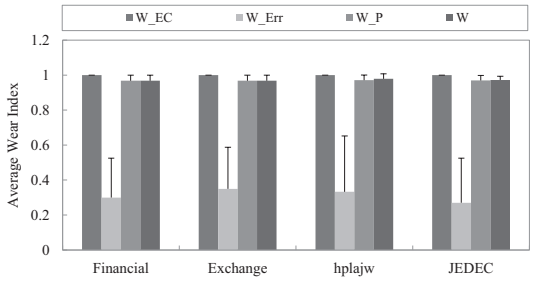

Figure 14: Average and maximum wear index values

performance. This is because the failure of a flash block can be predicted accurately by monitoring the program latency. The use of the proposed index W results in 13% higher effective lifetime on average than the case with W_P, the wear index solely based on the program latency. Especially, W_P achieves the lower effective lifetime compared to W during the first 1,000 cycles. As we discussed in Section 5, this is related to the fact that W_P increases rapidly during this period, incurring unnecessary wear-leveling overhead. On the other hand, the effective lifetime under the proposed wear index W increases steadily during the entire lifetime of flash memory.

Figure 14 plots the average wear index values at the end of the lifetime. The maximum wear index values among all the flash blocks are also displayed with the upper error bar. It is a good wear index if its value reaches 1.0 when one of the flash blocks ends its life. Although the value of W_{EC} always ends up with 1.0, W_{EC} can hardly be a good wear index as its value becomes 1.0 as soon as each flash block is used over the guaranteed P/E cycle. However, we can see that both W_P and W are relatively good wear indices, approaching to 1.0 when the lifetime of flash memory ends. On the other hand, W_{Err} is not accurate at all. W_{Err} remains near 0.3 on average, and becomes 0.6 at most.

7. CONCLUSION

The traditional wear-leveling algorithms try to distribute program/erase cycles evenly across all flash blocks. This is based on the assumption that all flash blocks become unreliable at the same time. However, our analysis with a real MLC NAND flash memory chip shows that the assumption is not true; flash blocks even in the same chip had different lifetimes.

To extend the lifetime of SSDs, this paper presents two techniques. First, we propose a new wear index that can predict how much a flash block is worn out. Second, we develop a new wear-leveling algorithm called Equalizer, which manages flash blocks effectively according to their wear indices. In this way, we can make every flash block be fully utilized until near the end of its lifetime, with no one being dead early. Our evaluations with three realistic workloads show that the proposed approach extends the effective lifetime of SSDs by up to 145% compared to the existing wear-leveling algorithm based on the erase count. In future work, we are interested in investigating the error characteristics of MLC NAND flash memory using more diverse chips from different manufacturers.

8. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIP) (No. 2013R1A2A1A01016441). This work was also supported by the IT R&D program of MKE/KEIT (No. 10041244, SmartTV 2.0 Software Platform).

9. REFERENCES

[1] Exchange server traces from SNIA IOTTA repository. http://iotta.snia.org/traces/130.

[2] hplajw traces from open source software. https://tesla.hpl.hp.com/opensource.

[3] OLTP application I/O traces from umass trace repository. http://traces.cs.umass.edu/index.php/storage/storage.

[4] S. Boboila and P. Desnoyers. Write endurance in flash drives: measurements and analysis. In *Proc. of FAST*, 2010.

[5] Y. Cai et al. Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In *Proc. of DATE*, 2012.

[6] L. Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *Proc. of SAC*, 2007.

[7] L. Grupp et al. Characterizing flash memory: anomalies, observations, and applications. In *Proc. of MICRO*, 2009.

[8] A. Gupta, Y. Kim, and B. Urgaonkar. *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*, volume 44. ACM, 2009.

[9] JEDEC solid state technology association. JESD219: solid-state drive (SSD) endurance workloads. 2010.

[10] S. Lee et al. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.*, 6, July 2007.

[11] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead. In *Proc. of MSST*, 2011.

[12] A. Olson and D. Langlois. Solid state drives data reliability and lifetime. *Imation White Paper*, 2008.

[13] K. Park et al. 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):919–928, 2008.

[14] M. Sanvido et al. Nand flash memory and its role in storage architectures. *Proceedings of the IEEE*, 96(11):1864 –1874, nov. 2008.

[15] K. Suh et al. A 3.3 v 32 mb nand flash memory with incremental step pulse programming scheme. *IEEE Journal of Solid-State Circuits*, 30(11):1149–1156, 1995.

[16] The OpenSSD project. http://www.openssd-project.org/wiki/the_openssd_project.