# Revive Bad Flash-Memory Pages by HLC Scheme

Han-Yi Lin, *Student Member, IEEE*, and Jen-Wei Hsieh, *Senior Member, IEEE*

***Abstract*—In recent years, flash memory has been widely used in embedded systems, portable devices, and high-performance storage products due to its nonvolatility, shock resistance, low power consumption, and high performance natures. To reduce the product cost, multi-level-cell (MLC) flash memory has been proposed; compared with the traditional single-level-cell (SLC) flash memory that only stores one bit of data per cell, each MLC cell can store two or more bits of data. Thus MLC can achieve a larger capacity and reduce the cost per unit. However, MLC also suffers from the degradation in both performance and reliability. In this paper, we try to enhance the reliability and reduce the product cost of flash-memory-based solid-state drive (SSD) from a totally different perspective. We propose the half-level-cell (HLC) scheme to manage and reuse the worn-out space in SSD; through our management scheme, the system can treat two bad pages as a normal page without sacrificing performance and reliability. The proposed scheme is purely on software/firmware-level, thus there is no need to change the hardware. The experiment results show that the lifetime of SSD with our proposed HLC scheme can be extended to 50.56% under the Windows workload and up to 65.45% under the multimedia workload. When we apply the HLC scheme to flash-memory cache of hybrid storage systems, the response time can be improved up to 20.57%.**

***Index Terms*—Endurance, NAND flash memory, reliability, solid-state drives (SSDs).**

## I. Introduction

SINCE flash memory has advantages in nonvolatility, shock resistance, low power consumption, and high performance, both academia and industry believe that flash-memory-based solid-state drive (SSD) has the potential to be a popular mass storage device in the near future. However, flash memory has the reliability problem. The lifetime of flash memory is limited. When a flash-memory block endured over 3000 program/erase (P/E) cycles, data access of pages in the block might suffer from frequent read/write errors. A flash-memory block is regarded as worn out if data cannot be correctly accessed from it.

Manuscript received September 27, 2017; revised January 28, 2018 and April 14, 2018; accepted May 1, 2018. Date of publication May 8, 2018; date of current version April 19, 2019. This work was supported by the Ministry of Science and Technology, Taiwan, under Grant MOST 104-2628-E-011-004-MY3. This paper is an extended version of the paper that appeared in the 2015 Design, Automation and Test in Europe. This paper was recommended by Associate Editor J. Xu. *(Corresponding author: Jen-Wei Hsieh.)*

H.-Y. Lin is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: d03922006@csie.ntu.edu.tw).

J.-W. Hsieh is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan (e-mail: jenwei@mail.ntust.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2834420

Excellent researches from various perspectives have been proposed to mitigate the reliability problem, which could be summarized into the following six categories.

1) *Enhanced ECC:* Error correction code (ECC) is used to correct error bits in flash memory. However, the capability of ECC is confined by the limited space of spare area. Jung *et al.* [1] proposed to preserve some data-area space in a page to store longer (or stronger) ECC. Chang and Kuo [2] traded more space for a higher reliability. In addition to the original ECC, their design supports enhanced ECC (in ECC-block area) to further protect user data (in data-block area). In case of burst errors, critical data are mirrored. Hsieh *et al.* [3] proposed an adaptive ECC scheme with four ECC levels. With adaptive management, ECC capability for a page would be upgraded whenever the current ECC cannot guarantee its reliability.
2) *Wear Leveling:* Previous researches of wear leveling focused on distributing P/E cycles evenly across all blocks of flash memory. However, only taking P/E cycles into consideration might not be enough. Yang *et al.* [4] noted that flash blocks endured the same P/E cycles usually have different numbers of error bits. Thus they proposed a reliability-aware wear-leveling scheme to distribute P/E cycles over blocks based on their bit error rates so as to even out the error rate among flash-memory blocks. As a result, the number of good blocks can be maximized, and the product lifetime of flash-memory storage devices can be prolonged. Woo and Kim [5] used a more sophisticated wear index to indicate the wear status of all the blocks. They found that the P/E latencies have a relation with the degree of page wear. In their method, both bit error rates and P/E latencies were taken into consideration to produce a more precise wear index, from which a better wear leveling can be achieved.
3) *Write Traffic Reduction:* Some researches exploited data de-duplication [6] or data compression to reduce the write traffic [7]–[9]. In [6], a content-aware flash translation layer (FTL) was proposed to reduce write traffic by removing unnecessary duplicate writes and extend available free flash memory space by coalescing redundant data in SSDs. In [7], zFTL was proposed to reduce the amount of data written to flash memory by supporting on-line transparent data compression based on the X-Match algorithm; a prediction scheme that identifies incompressible data in advance without going through full compression was designed to minimize compression overhead and power consumption. Wu and He [8] exploited the content locality between the write data and its corresponding old version in flash memory; by only

storing the difference between the new and old data, the number of writes to flash memory can be reduced. Zhang *et al.* [9] presented a set of design techniques at the filesystem and computer architecture levels to realize mobile device OS/Apps transparent compression without read latency overhead. Lee *et al.* [10] combined several lifetime-enhancement schemes, including de-duplication, lossless compression, and performance throttling, in an integrated fashion so that the lifetime of SSDs can be maximally extended.

4) *Mellow Write:* In addition to the notion of wear leveling and write traffic reduction, Zhang *et al.* [11] proposed *mellow writes* to reduce the wearout of individual writes at the expense of long write latency. By selectively perform slow writes, the performance impact could be minimized while a longer lifetime can be achieved. Strukov [12] derived a phenomenological model to realize the endurance-write-speed tradeoffs for nonvolatile memories. Observing the goals of boosting performance and prolonging lifetime are often in opposition, Deng *et al.* [13] proposed a learning-based framework, referred to as *memory cocktail therapy*, that manages to optimize combined techniques which utilize multiple dynamic tradeoffs in non-volatile memory.

5) *Downgraded Flash Memory Reclamation:* MLC requires the controller to be able to identify four different states to store 2-bit data per cell. Jimenez *et al.* [14] proposed to revive those flash-memory blocks that are becoming unreliable to store multiple bits per cell by storing only one single bit per cell, referred to as Phoenix. Thus the controller only needs to identify two different states per cell. Although the capacity of the storage device is halved, its lifetime is extended. Hsieh *et al.* [15] presented a set-based mapping strategy with low hardware resource requirements for making downgraded flash-memory chips useable in products. The basic idea is to use $m$ physical blocks to serve $n$ logical blocks, where $m \geq n$. Observing that many pages in a bad block are still functional, Wang and Wong [16] modified the bad block management scheme so that good pages from bad blocks can be reclaimed.

6) *Healing:* In a few years ago, researchers at Macronix found that heating worn-out flash-memory blocks can make them reusable. Based on their report, a self-healing flash memory can survive more than 100 million cycles [17], [18]. However, the heating process consumes a substantial amount of power. Chang *et al.* [19] introduced the notion of virtual erase counts to evenly disperse erasures among blocks in a controlled interval. In this way, blocks will undergo healing at different times. By scheduling write and erase operations on a small number of flash memory cells at a time, Chen *et al.* [20] avoided quick energy depletion caused by concentrated heating. Chang *et al.* [21] proposed a heal-leveling design that evenly distributes healing cycles to flash blocks without introducing a large amount of live-data copying overheads.

In this paper, we propose a software-based half-level-cell (HLC) scheme to improve the reliability and extend the product lifetime of SSD and SSD/hard disk drive (HDD) hybrid storage system. We focus on how to manage and reuse the already worn-out space. Since the HLC scheme can revive bad pages, additional space can be obtained to apportion write burden of the storage system to prolong its lifetime. Different from Phoenix [14] that requires hardware modification to downgrade a corrupted MLC block into an single-level-cell (SLC) block, the proposed scheme is purely on software/firmware-level, thus there is no need to change the hardware. In addition, the HLC scheme is more flexible than Phoenix, since the HLC scheme reclaims worn-out space in page level while Phoenix must reclaim worn-out space in block level due to hardware constraint. Under our scheme, the capability of ECC can be enhanced without changing the hardware design of ECC encoder/decoder. The built-in parallel commands are properly adopted to minimized the incurred overheads. Thus the endurance and reliability of a storage system can be improved without sacrificing the performance. By reusing those corrupted space and enhancing the ECC capability, we could obtain more available space to reduce the burden on other normal space.

The rest of this paper is organized as follows. Section II provides background of flash-memory storage systems. We present the detail of the proposed HLC design in Section III and provide implementation remark in Section IV. We evaluate the performance of our scheme in Section V and conclude this paper in Section VI.

## II. Background

### A. Basic of Flash Memory

Flash memory is a kind of electrically erasable programmable read-only memory that must conduct an erase operation over a block before a program (i.e., write) operation can be performed on its page. The asymmetric P/E units make in-place-update impractical; thus out-place-update is usually adopted in the management. As a result, a mapping table must be maintained to keep track of the current version of data, and garbage collection is required to reclaim free space from old version of data.

When a block is erased, all of its bits would be in “1” state. When a page is programmed, some of its bits might be changed to “0” state. As indicated in [22], the state change of a cell will wear the cell. After sustaining a large amount of P/E cycles, some cells of a page might be damaged. If these damaged bits cannot be corrected by the associated ECC, the resided block is regarded as a *bad block* and no longer available. Thus vendors usually provide extra blocks, referred to as the *overprovision space*, for the replacement of bad blocks. After the overprovision space is exhausted, product lifetime of the storage device is considered as the end if any of its remaining blocks turns into a bad block.

### B. SLC and MLC

Flash memory can be roughly categorized into SLC flash memory or MLC flash memory depending on the number of bits that each cell can store: SLC can store one bit per cell, while MLC can store two or more bits per cell. Those flash memory that can store three or four bits per cell are also referred to as triple-level-cell (TLC) flash memory or quadruple-level-cell (QLC) flash memory, respectively. Since we use two bad pages to serve one logical page, which could be regarded as storing half bit per cell, we named our scheme the HLC scheme.

Although MLC provides higher storage density than SLC, it suffers from poorer performance and endurance. An SLC block can tolerate about 10000 P/E cycles, while an MLC block can only endure about 3000 P/E cycles for 30–40 nm (i.e., 3*x*-nm) technology generations [23]. The endurance of a TLC block is even worse, which is about 1000 P/E cycles. In addition, MLC requires pages in a block to be programmed sequentially and does not support partial programming as SLC does.

### *C. Two-Plane Commands*

A flash-memory storage device (e.g., [24]–[26]) usually consists of a fixed number of channels. Each channel has its own data/control bus connected to the host interface and contains a number of packages. Through the host interface, the storage device can communicate with the host. Each package is composed of chips. All chips in a package share the same I/O bus of the package. Each chip consists of two or more dies. Each die can be further divided into multiple planes. Each plane contains thousands of blocks and one or two data/cache registers as its I/O buffer.

To boost the throughput, most flash memory supports two-plane commands, including two-plane read, two-plane write, and two-plane erase. Two-plane read can read two pages simultaneously. However, the two pages must be in different planes of the same die with the same page offset. Similarly, two-plane write can write two pages simultaneously with the same constraint of two-plane read. Since MLC requires sequential page-programming in a block, two-plane write sometimes would incur dummy-page writes due to the requirement of the same page offset. Two-plane erase can erase two blocks simultaneously. It requires the two blocks to be in different planes of the same die with the same block offset. Although the proposed HLC scheme uses two bad pages to serve one logical page, the performance overhead can be hidden by two-plane commands.

## III. Design of Half-Level-Cell Scheme

### *A. Overview of the HLC Scheme*

The design goal of the HLC scheme is to extend the lifespan of flash-memory storage devices by reclaiming bad pages. The basic idea is to reuse the strong segments of two bad pages to serve one logical page.[1] The design goal can be achieved from two perspectives.

1) By reusing bad pages, the maximum P/E cycles of blocks could be further pushed. Thus the lifespan of flash-memory storage devices can be extended explicitly.
2) Since extra free space can be provided from bad pages, garbage collection would be triggered less frequently. By postponing garbage collection, pages with frequently updated data would have a chance to become invalid before the resided block is erased. Thus the efficiency of garbage collection is improved, and the lifespan of flash-memory storage devices can be extended implicitly.

[1]We logically partition each page into two segments.

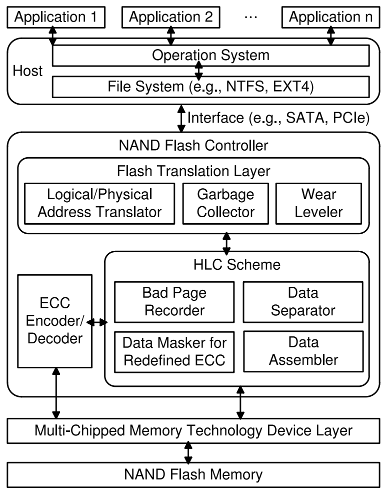


Fig. 1. System architecture.

Fig. 1 illustrates the overall system architecture. The proposed HLC scheme is introduced between FTL and multi-chipped memory technology device (MTD) layer. FTL realizes high-level management algorithms, including logical/physical address translator, garbage collector, and wear leveler. Low-level functionalities of flash memory, including read, write, and erase, are implemented in MTD layer. With the layered design, hardware manufacturers can integrate the HLC scheme into their products easily without significant modifications. Since the scheme we proposed does not depend on a particular hardware and is only relevant to the manipulation of data, we can easily add our scheme through firmware updates without modifying the hardware.

The HLC scheme consists of four major components, including *bad page recorder*, *data separator*, *data masker for redefined ECC*, and *data assembler*. *Bad page recorder* keeps track of bad pages. Since the HLC scheme reclaims bad pages for reuse, we must know which page was already worn-out. When a read or write operation is performed, bad page recorder will monitor errors reported from ECC decoder and record the physical page address of bad pages. In our design, two bad pages are combined (referred to as a *bad-page set*) and treated as a normal page to serve one-page data. The key issue is how to put the data into two bad pages. It is handled by *data separator*, which will be discussed in Section III-C1. When we allocate a bad-page set to serve one-page data, data separator will transform the one-page data into two-page data. A two-plane write is then issued to write two bad pages in parallel. While data separator transforms one-page data into two-page data, *data assembler* is in charge of recovering the data from two bad pages. *Data masker* transforms half of the input data into dummy data during the process of ECC encode/decode, which means half of the content that generates the ECC is known in advance. As a result, the ECC is redefined to protect the other half of the content. In other words, the ECC capability is implicitly doubled.

Since we combine two bad pages as a bad-page set to represent one logical page, we might suffer from a performance degradation due to accessing two pages for one-page data. However, this performance degradation can be alleviated by utilizing the *two-plane commands* that read/write two pages in parallel. All planes on the same die can carry out the same type of operation at the same time. In order to speed up the request processing time, manufacturers, e.g., [27]–[29], have built-in two-plane commands in their products. There is a restraint when we conduct two-plane commands on two pages/blocks: offsets of the two pages/blocks must be the identical in two different planes of the same die. Taking this restraint into consideration, we adopt two-plane commands to serve requests [26]. As a result, the wearing statuses of pages in a page set are almost the same. Thus we can easily find two bad pages in the same page set. When the proposed HLC scheme transforms one-page data into two-page data and writes them into flash memory, two-plane write is used to deal with the two-page data. With two-plane commands, bad pages can be reclaimed to extend lifespan of flash-memory storage devices with minimized performance overhead.

## B. Bad Page, Strong/Weak Segment, and Bad-Page Set

A bad page is the page that has more errors than that can be corrected by its ECC; it is no longer available for normal use. However, the proposed HLC scheme could reclaim bad pages for further reuse. Suppose the number of errors that the ECC can correct is no more than $n$ bits. As long as there is no error in ECC and the number of error bits in the data area of a bad page does not exceed $2n$, we can always reclaim two such pages to store one-page data. The space ratio of data area to all area (including data area and spare area) in a page is defined as *code rate*. The code rate of a typical flash-memory page is 32/33 [30], [31]. To estimate the probability $p$ of a bad page with no error in ECC, the hypergeometric distribution function is adopted. The probability $p$ can be derived by

$$p = \left[ \frac{\binom{D+S-E}{x}\binom{E}{R-x}}{\binom{D+S}{R}} \right]^B$$

where $B$ is the number of chunks per page, $D$ and $S$ are the number of bits in data area and spare area of a chunk, $E$ is the number of ECC bits for a chunk, $R$ is the number of error bits in a chunk, and $x$ is the number of error bits in data area of a chunk. Note that spare area of a page is not dedicated to ECC; it is also used to store other metadata, e.g., the associated logical block address (LBA). To illustrate, we take Micron 2112-byte SLC pages as an example [31]. For the 2112-byte page, it is divided into four 512-byte (+ 16-byte spare) chunks; each chunk is protected by 3-byte ECC to correct 1-bit error. Thus we have $B = 4$, $D = 512 \times 8 = 4096$, $S = 16 \times 8 = 128$, and $E = 3 \times 8 = 24$. Suppose the number of errors per chunk is $2n$ and there is no error in ECC. Thus we have $R = 2$ and $x = 2$. As a result, we have the probability $p \approx 0.95$.[2]

We divide such a bad page into two segments with equal size. If $2n$ errors occurred in the page, we can still find a segment with the number of error bits in the range between 0 and $n$; the number of error bits in the other segment is from $2n$ to $n$, thus the total number of error bits is equal to $2n$. The segment with less error bits is regarded as the *strong segment*, while the segment with more error bits is regarded as the *weak segment*. By storing data in the strong segment, reliability of the data can be guaranteed.

[2]Due to the demanding requirement of ECC capability, current products have various code rates. For example, Micron MT29F32G08ABAAA 8-KB SLC is 8192/8640, Toshiba TH58TFxxDFLBA 15-nm 16-KB MLC is 16 384/17 760, and Samsung K9ACGD8U0D is 16 384/18 432.

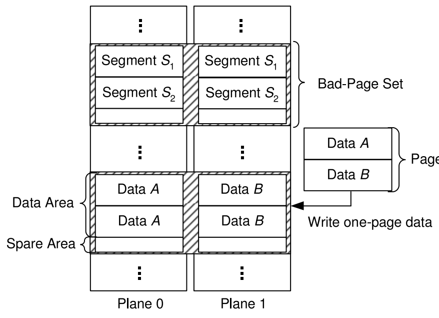


Fig. 2. Layout of bad-page sets.

Since the available space of a bad page is now halved, the HLC scheme groups two bad pages as a *bad-page set* to serve one-page data. Fig. 2 illustrates the layout of bad-page sets. Two bad pages which have the same offset from different planes in the same die are combined together as a bad-page set to serve one-page data. As mentioned earlier, each bad page is divided into two segments, in which the strong segment can be used to store data. The technical issue is how to identify the strong segment in each bad page, which is discussed in the next section.

## C. Write and Read in Bad-Page Set

*1) Write Flow of Bad-Page Set:* The solid line in Fig. 3 illustrates the process of writing data in a bad-page set. In our design, *data separator* is in charge of transforming one-page data into two-page data and arranging the transformed data in two bad pages of a bad-page set. When the one-page data is submitted to data separator, data separator divides the data into two equal-sized parts; the size of each part is equal to the size of a segment. To avoid the overhead of identifying the strong segment of each bad page in advance, data separator duplicates each part to fit the size of a page and writes them to the bad-page set directly. Since the data is written to both segments of a bad page, it can be guaranteed that the data is protected by the strong segment of the bad page as long as there is no error in ECC and the total number of error bits in the data area does not exceed $2n$.

Notably, the HLC scheme does not send the data generated by data separator to ECC encoder directly. Instead, *data masker* generates a pair of inputs to ECC encoder by separately putting each part of data in segment $S_1$ and padding segment $S_2$ with 0s. Thus the strong segments in the bad-page set can be fully protected by the ECC, which would be further discussed in Section III-D. The parity generated by ECC encoder will be merged with the data generated by data separator as the output data. After the output data is written to a bad-page set, a two-plane read is conducted to read out the data just stored in the bad-page set. Since we still have the

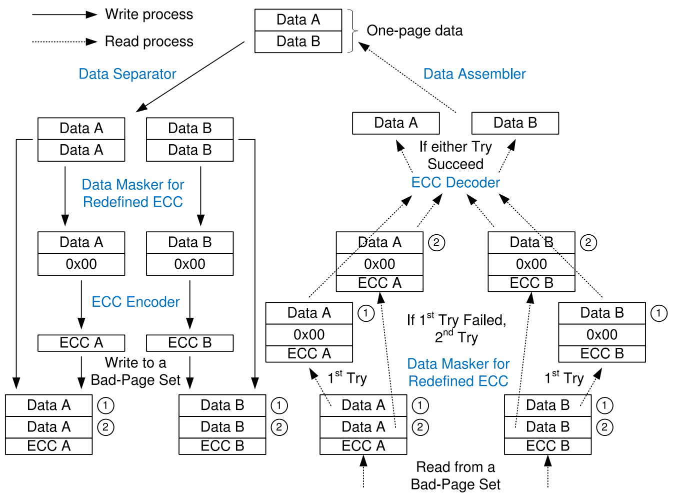


Fig. 3. Process of data read/write in bad-page set.

original data in the buffer, we can easily detect the number of error bits by XORing the original data with the read-out data.[3] Based on the result, we can identify the strong segment and the weak segment of each page in the bad-page set and record this information in the page-set status table. It can help us quickly find the strong segment during a read operation. The detail of the page-set status table would be discussed in Section III-E. If the number of error bits in data area of either page of the bad-page set exceeds $2n$, the bad-page set is no longer available for storing data; another bad-page set would be allocated for the write request.

*2) Read Flow of Bad-Page Set:* When we want to read data saved in a bad-page set, we must read both pages in the set and then extract the required data from bad pages. This work is handled by *data assembler*. The dotted line in Fig. 3 illustrates the process of data read in a bad-page set. We need to obtain the two parts of data, i.e., data $A$ and data $B$, from strong segments of the two bad pages. The two parts of data are those separated from one-page data by *data separator*, and strong segments can be identified by looking up the corresponding entry in the page-set status table that are set during the write operation. With data assembler, data from the two strong segments can be combined to recover the required data.

However, if the page-set status table is not maintained in NVRAM or battery-backed DRAM and the storage device has just suffered a power failure or a system crash, we would lose the information about the strong segments. In this case, the strong segment in a bad page can still be identified by ECC decoder, data assembler, and data masker as follows.

1) Data assembler requests data masker to mask out one segment of the bad page, either $S_1$ or $S_2$, to extract the other segment.
2) No matter the extracted segment is $S_1$ or $S_2$, data masker treats it as $S_1$ in a page with $S_2$ filled with all 0s.
3) The assembled page would be send to ECC decoder to determine its correctness.
4) If there are too many error bits in the assembled page that cannot be corrected by the associated ECC, the extracted segment would be regarded as a weak segment. Data assembler would request data masker to mask out the other segment and continued from steps 2 to 3.[4]
5) After strong segment of the page is identified and the part of data is extracted, data assembler continues to extract the other part of data from the other bad page in a similar way.
6) After both parts of data are extracted, data assembler assembles two parts of data and returns the data to the host.

[3]Note that retention error is not considered since it is out of scope of this paper, and a 100% accuracy is not required to make our idea work.

## D. Redefined ECC

In this section, we will discuss how to extend the ECC capability for the one-page data stored in a bad-page set. For the same amount of data, the more parity space can be supported, the more error bits can be corrected. Since we use two bad pages to serve one logical page, there are two spare areas from physical pages in the bad-page set that can be utilized. In other words, we could save more ECC parity in the extra spare area to serve one-page data and enhance its reliability. However, most manufacturers usually implement ECC encoder/decoder in hardware to accelerate the processing time of ECC encoding/decoding; if we want to produce more ECC parity, we might need to change the existing hardware of ECC encoder/decoder.

We propose an approach to extend the ECC capability for one-page data stored in a bad-page set without changing the hardware of ECC encoder/decoder. Rather than using double-sized ECC parity to protect one-page data, we use two normal-sized ECC parities to protect two half-page data. We add dummy bits with all 0s to the half of one-page data to generate a temporary data to fit the input format of the ECC encoder. Notably, the temporary data is generated for ECC decoding, not for storing in bad pages. With this approach, we can know half content of the data in advance, thus the ECC parity only needs to protect the other half of the real data.

We shall illustrate how the ECC capability can be extended with an example. We use Hamming code, which is adopted by Micron Technology, Inc., as ECC for their SLC flash memory [31], in the example to simplify the explanation. It uses 24-bit ECC per 512-byte chunk to perform a 2-bit errors detection and a 1-bit error correction. For the 2-KB page, the calculation can be done per 512-byte chunk, as shown in Fig. 4. For every data byte in each chunk, 18 bits for line parity (LP0–LP17) and 6 bits for column parity (CP0–CP5) are generated according to the scheme shown in Fig. 5. For example, LP0 is generated by XORing all the bits of even bytes, while CP1 is generated by XORing odd bits of all bytes. When the main area has a 1 bit error, each parity pair (e.g., LP0 and LP1) is 1 and 0 or 0 and 1. The fail bit address offset can be obtained by retrieving the following bits from the result of XOR: byte address = (LP17, LP15, LP13, LP11, LP9, LP7, LP5, LP3, LP1) and bit address = (CP5, CP3, CP1). For example, suppose an error occurred at bit 3 of byte 2, we would have LP0 = LP3 = LP4 = LP6 = LP8 = LP10 = LP12 =

[4]If both segments of the bad page are regarded as weak segments, the total number of bit errors must exceed $2n$. Data in the page cannot be correctly accessed.

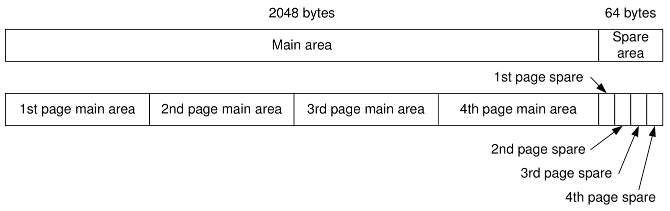


Fig. 4. 2-KB page divided into chunks [31].

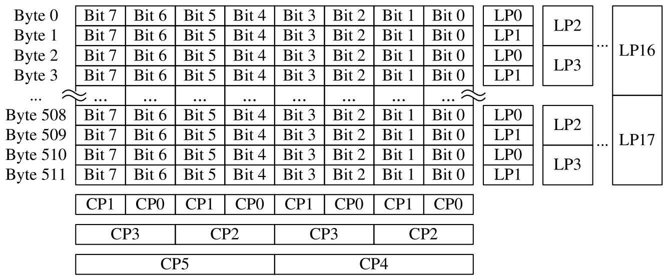


Fig. 5. Parity generation [31].

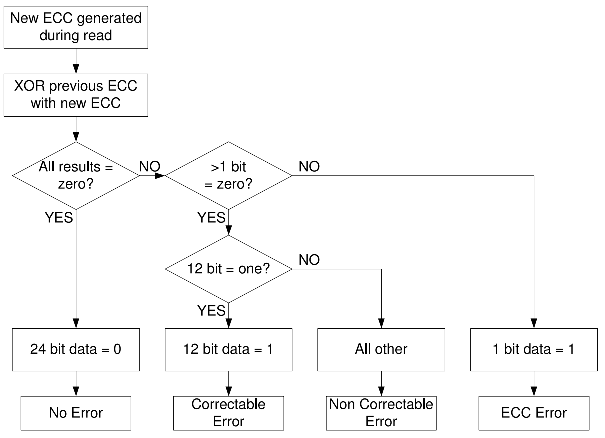


Fig. 6. ECC detection flowchart [31].

LP14 = LP16 = 1 and CP1 = CP3 = CP4 = 1, while all the other parity bits = 0. According to the above result, the byte address $(00000010)_2 = 2$ and the bit address $(011)_2 = 3$ of the fail bit can be correctly obtained. When more than two error bits occur, the data cannot be corrected. Fig. 6 shows the ECC detection flowchart.

Suppose a 512-byte page has two error bits occurred, which cannot be corrected by the ECC. Since the ECC is incapable of correcting all errors, the page is regarded as a bad page and cannot be used normally. Our HLC scheme divides the page into two equal-sized segments $S_1$ and $S_2$, where $S_1$ covers from bytes 0 to 255 and $S_2$ covers from bytes 256 to 511. In the best case, all the error bits might occur in one segment, leaving the strong segment with no error. During the decoding process, data masker would put the strong segment in $S_1$ and filled $S_2$ with all 0s, as mentioned in Section III-C2. Since there is no error bit in this combination, the stored data can be correctly extracted. In the worst case, all the error bits occur evenly in two segments, i.e., each segment has one error bit. By putting either segment in $S_1$ and filled $S_2$ with all 0s, we have only one error bit in the combined page, which can be corrected by the ECC to extract the required data.

TABLE I
PARAMETERS OF BINOMIAL PROBABILITY DENSITY FUNCTION

| Symbol | Definition | Setting |
|---|---|---|
| $p$ | Raw bit error rate (RBER) | $10^{-6}$ |
| $B$ | Number of sector in a flash memory page | 8 |
| $t$ | ECC capability (Unit: bits) | 4 |
| $k$ | Size of data (Unit: bits) | 4,096 |
| $r$ | Size of ECC parity (Unit: bits) | 64 |
| $N$ | Size of received codeword (Unit: bits) | 4,160 |

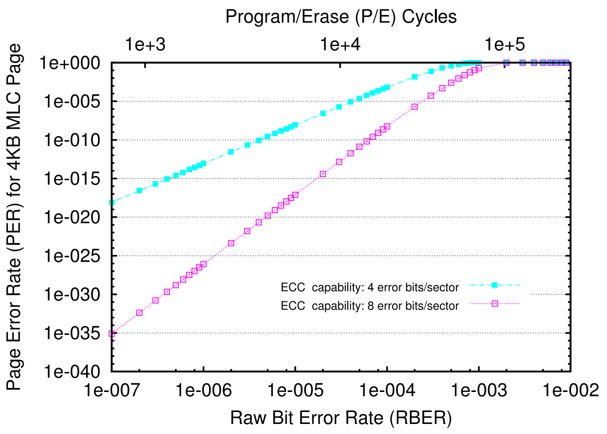


Fig. 7. Impact of different ECC capabilities over PER.

To estimate and observe the improvement of page error rate (PER) under the HLC scheme, the model of binomial probability density function [30] is adopted. Table I summarizes the definitions and the settings of its parameters. The PER can be calculated as follows:

$$\text{PER} = 1 - \left[\sum_{i=0}^{t} \binom{N}{i} \times p^i \times (1-p)^{(N-i)}\right]^B.$$

When the HLC scheme is applied to a bad-page set, the number of error bits that can be corrected by ECC is improved from 4 bits per sector to 4 bits per half-sector. Since the size of data which we want to encode is fixed ($k = 4096$ bits = 512 bytes) and the original size of ECC parity is 64 bits, $N$ could be either 4160 (= 4096+64) or 4224 (= 4096+2×64) bits.[5] Fig. 7 shows the relationship of different ECC capabilities among P/E cycles (top $x$-axis), RBER ($x$-axis), and PER ($y$-axis) [33]. Although we plot the figure from RBER = $10^{-7}$, the initial RBER of MLC is $10^{-6}$ [34]. With the increase of P/E cycles, RBER and PER also increase. For the ECC that could correct four error bits per sector, when the number of P/E cycles reaches about 1.5e+3, its RBER becomes 4e-7 and the PER will reach $10^{-15}$. If we can double the ECC capability (i.e., eight error bits/sector), the page could endure about 7.5e+3 P/E cycles (with RBER ≈ 1.8e-5), while its PER reaches $10^{-15}$. The PER is nearly equal to one when RBER exceeds $10^{-2}$. From this figure, we can observe that when the HLC scheme is applied to bad pages, PER can be improved.

[5]The size of the ECC parity varies depending on the correction capability and is usually limited by the size of spare area in a page. Using Bose–Chaudhuri–Hocquenghem code, the number of parity bits needed to correct four error bits in 4096 bits data is 52 bits [2], [32]. In order to align with the power of two, we set it to 64 bits.

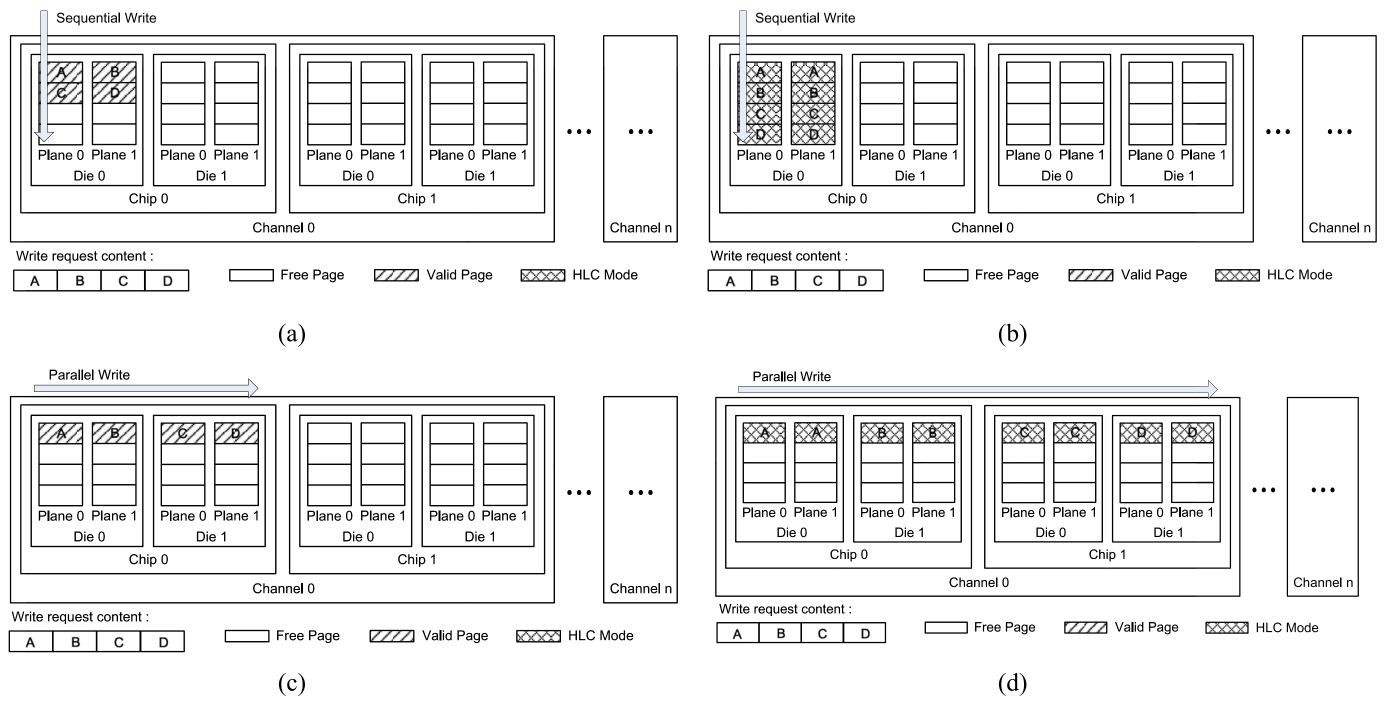


Fig. 8. Impacts of applying multilevel parallelism over normal pages or HLC pages for four-page writes. (a) Plane-level parallelism over normal pages. (b) Plane-level parallelism over HLC pages. (c) Multilevel parallelism over normal pages. (d) Multilevel parallelism over HLC pages.

## E. Management of Bad-Page Sets

To reduce the overhead of serving one logical page with two bad pages, the HLC scheme adopts two-plane commands to read or write two bad pages in parallel. However, the two pages must be in different planes of the same die with the same page offset. The technical issue is how to have bad pages follow the requirement of two-plane commands. Although adopting two-plane commands to serve normal requests can let pages in the page set have similar wearing statuses, we still cannot guarantee these pages become bad pages at the same time.

In the proposed HLC scheme, page-set-level mapping is adopted.[6] Each page set contains two pages in different planes of the same die with the same page offset. A *page-set mapping table* is maintained to keep track of the mapping between logical pages and physical page sets. We also maintain a *page-set status table* to manage page sets; each 3-bit entry contains the status bits for the associated page set. Table II summarizes the meaning of status bits. As shown in the table, the most significant bit in the status bits indicates whether the associated page set is in HLC mode. The lifetime of a page set can be categorized into four stages.

1) In the beginning, both pages in the page set are good pages. Thus the page set can serve two logical pages, and the value of the status bits is 000.
2) After sustained P/E cycles, some page in the page set might become a bad page. Since the other page can still be protected by its ECC, the page set can use this page to serve one logical page without applying the HLC scheme. In this stage, the value of the status bits is either 001 or 010.
3) With continuous usage, the remaining good page in the previous stage would become a bad page as well. Now the page set would change to HLC mode. We have two bits for the two bad pages to indicate the offset of their strong segments. Thus the status bits could be 100, 101, 110, or 111.
4) Finally, if some bad page in the page set has more than $2n$ error bits, the page set cannot be reused by the HLC scheme and is no longer available. The value of the status bits would be 011.

[6]The notion of the HLC scheme can easily accommodate to any kind of address translation mapping unit, e.g., page-level mapping, block-level mapping, or hybrid mapping; it can also be applied to any kind of NAND flash memory cell, e.g., SLC, MLC, or TLC.

TABLE II
STATUS BITS FOR THE MANAGEMENT OF BAD-PAGE SETS

| Value | HLC Mode | Served Pages | Meaning |
|---|---|---|---|
| 000 | No | 2 | Both pages in the page set are good pages. |
| 001 | No | 1 | The page in the 2nd plane is a bad page. |
| 010 | No | 1 | The page in the 1st plane is a bad page. |
| 011 | No | 0 | The page set is no longer available. |
| 100 | Yes | 1 | Strong segments in two pages = $(S_1, S_1)$. |
| 101 | Yes | 1 | Strong segments in two pages = $(S_1, S_2)$. |
| 110 | Yes | 1 | Strong segments in two pages = $(S_2, S_1)$. |
| 111 | Yes | 1 | Strong segments in two pages = $(S_2, S_2)$. |

## IV. IMPLEMENTATION REMARK

### A. Multilevel Parallelism

In the HLC scheme, two bad pages in a bad-page set are used to serve one logical page. Although two-plane commands can hide the overhead incurred by two-page writes, a proper management is still required for large writes. Fig. 8 illustrates

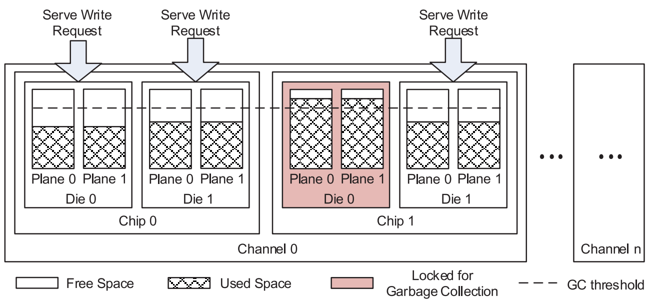


Fig. 9. Die locked for garbage collection.

the impacts of applying multilevel parallelism over normal pages or HLC pages for large writes. Suppose a request is issued to write four pages. With only plane-level parallelism, it takes two write-operation time to complete the request over normal pages, as shown in Fig. 8(a); the HLC scheme has to take four write-operation time to complete the request, as shown in Fig. 8(b). However, if plane-level, die-level, chip-level, and channel-level parallelism can be fully utilized, both normal pages and HLC pages require only one write-operation time to complete the request, as shown in Fig. 8(c) and (d). Suppose an SSD device has four channels, each channel contains two chips, each chip consists of two dies, and each die is composed of two planes. In this device, the HLC scheme can serve a 16-page write with one write-operation time when multilevel parallelism is fully utilized.

## B. Distributed Garbage Collection

Due to the out-place-update nature of flash memory, garbage collection is indispensable to reclaim free space from invalid pages. However, garbage collection is time consuming since it requires erase operations and usually accompanies with live-page copies. If a write request has to wait for a garbage collection to complete, its response time would be greatly delayed. Thus garbage collection is usually triggered during system idle time. Even if the system is always busy, we can still serve a small write request and trigger garbage collection in parallel by arranging them on different dies to minimize the adverse impact incurred by garbage collection, as shown in Fig. 9.

To fully utilize plane-level, die-level, chip-level, and channel level parallelism, we must ensure each plane has enough free space to serve a write request. Thus the notion of distributed garbage collection [26] is adopted. If the number of free pages in a plane is less than the *garbage collection threshold*, garbage collection would be scheduled for the plane to reclaim free pages. Notably, the total number of remaining free pages in a plane is used as an indication to trigger garbage collection. We will select the block with the highest invalid page ratio in the plane as a victim block for the reclamation. During garbage collection, the plane cannot serve other requests until sufficient free pages are reclaimed. Since we use two-plane writes and striping technique to serve write requests, the number of free pages consumed in two planes of the same die would be similar. Thus two-plane erases can usually be employed during garbage collection to further reduce the timing overhead of garbage collection.

## C. Overheads Analysis

This section is meant to quantitatively analyze/estimate the overheads incurred by the HLC scheme in terms of the increased energy consumption, the overhead of mapping tables, and the increased read overhead. Since the HLC scheme serves a logical page by two bad pages with two-plane commands, it incurs extra energy consumptions. According to [35], for 5x-nm MLC, the power consumptions of a read operation and a write operation are 0.0283 and 0.0556 W, respectively. Thus reading a page normally requires 3.86 $\mu$J ($= 136.42\ \mu\text{s} \times 0.0283$ W), while reading a page via the HLC scheme requires 6.31 $\mu$J ($= 222.96\ \mu\text{s} \times 0.0283$ W); it takes 54.847 $\mu$J ($= 986.46\ \mu\text{s} \times 0.0556$ W) to write a page normally and 59.68 $\mu$J ($= 1073.38\ \mu\text{s} \times 0.0556$ W) to write a page via the HLC scheme.[7] Notably, this overhead only occurs during the late stage of the SSD's lifetime for reclaiming bad pages; in the early stage of the SSD's lifetime, data are accessed normally.

As mentioned in Section III-E, the HLC scheme maintains a page-set mapping table to keep track of the mapping between logical pages and physical page sets; it also maintains a page-set status table to manage page sets. Suppose the capacity of the SSD is $x$ GB with $y\%$ physical space as the overprovision space, and the page size is $z$ KB. The total number of pages in this device is $(x \times 2^{20})/z$, and the total number of physical page sets is $(x \times 2^{19})/z$. Since $y\%$ of physical space is reserved as the overprovision space, the total number of entries in the page-set mapping table is $(x \times 2^{20} \times (1 - y\%))/z$. To be able to index all the physical page sets, each entry in the page-set mapping table requires $\lceil \log_2((x \times 2^{19})/z) \rceil = \lceil 19 + \log_2(x/z) \rceil$ bits. Thus the size of the page-set mapping table is $(x \times 2^{17} \times (1 - y\%) \times \lceil 19 + \log_2(x/z) \rceil)/z$ bytes. Since the total number of entries in the page-set status table is $(x \times 2^{19})/z$ and each entry in the table is 3 bits, the size of the table is $(x \times 2^{16} \times 3)/z$ bytes. For a 128-GB SSD with 20% overprovision space and 4-KB pages, the HLC scheme requires a 80-MB page-set mapping table and a 6-MB page-set status table. Compared with the conventional SSD that requires 83.2-MB page-level mapping table, the space overhead is only 3.365%. For performance and persistency consideration, we can store both page-set mapping table and page-set status table in NVRAM or battery-backed DRAM. If NVRAM or battery-backed DRAM is limited, we could store the two tables in flash memory and loading the required part on demand. However, it incurs extra overhead of flash-memory reads/writes [36], [37].

As mentioned in Section III-C1, after the output data is written to a bad-page set, a two-plane read is conducted to read out the data just stored in the bad-page set to identify the strong/weak segments of bad pages in the bad-page set. Although identifying strong/weak segments during the write to a bad-page set incurs extra overhead of two-plane read, it helps in quickly finding the strong segment during a read

[7]Since we cannot get the detailed information about the power consumptions for two-plane commands, we assume the power consumptions of two-plane read/write are the same as that of normal read/write. But two-plane commands and normal operations differ in their operation time.

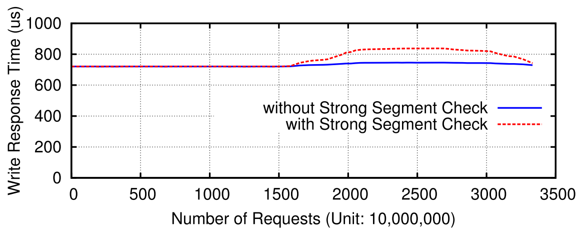


Fig. 10. Average write response time with/without strong segment check.

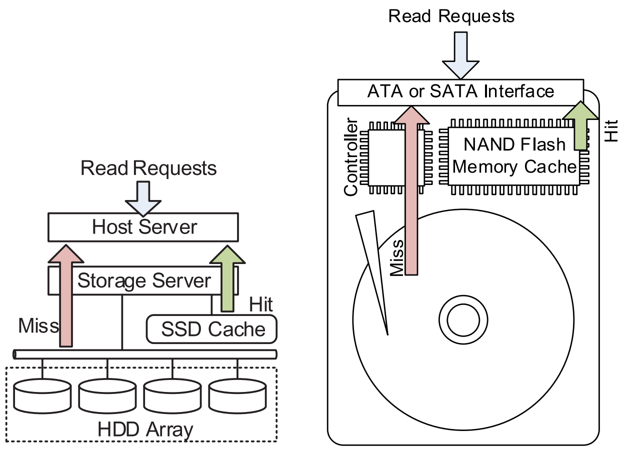


Fig. 11. Two different kinds of hybrid storage systems. (a) Combine SSDs and HDDs in data center and (b) flash memory and HDD in one device.

operation. We conduct an experiment to evaluate this overhead by comparing the average write response times with/without strong segment check. As shown in Fig. 10, there is no difference in the early stage of the SSD's lifetime due to no access to bad-page sets. In the late stage of the SSD's lifetime, the average write response times with/without strong segment check slightly differed due to the overhead of strong segment check. As shown in the figure, the overhead incurred by the strong segment check is about 4.66%. During the end of SSD's lifetime, since bad pages suffered from too many error bits ($> 2n$), bad-page sets would gradually became unavailable. Thus the average write response times with/without strong segment check converged again.

### D. Apply HLC Scheme to Hybrid Storage Systems

In recent years, the notion of hybrid storage system has been proposed. It combines heterogenous storage media to achieve a balance between performance and cost. Some data centers adopt a small number of flash-memory-based SSDs as caches for an HDD array, as illustrated in Fig. 11(a). The design goal is to have the performance as good as a pure SSD array and the hardware cost as low as a pure HDD array. Some vendors apply the notion to a single device, e.g., Seagate's solid state hybrid disk. In this kind of devices, flash memory is used as a cache or as a part of mass storage medium, as shown in Fig. 11(b).

However, flash memory has the endurance problem. MLC-based SSD might not be able to sustain the heavy workloads

TABLE III
TIMING PARAMETERS OF FLASH-MEMORY CHIP [39]

| Operation | Total Time |
|---|---|
| Read | 136.42$\mu s$ |
| Two-Plane Read | 222.96$\mu s$ |
| Write | 986.46$\mu s$ |
| Two-Plane Write | 1073.38$\mu s$ |
| Block Erase | 2000.14$\mu s$ |
| Two-Plane Block Erase | 2000.74$\mu s$ |

from data centers. Liu *et al.* [38] proposed to handle the uncorrectable errors in the SSD cache as cache misses and extract correct data from the HDD arrays. By sacrificing available capacity to allocate more ECC parities associated with data, the SSD caches can continue operating.

In addition to pure flash-memory-based SSDs, the proposed HLC scheme can also be easily applied in the hybrid storage systems to extend the lifetime of flash memory. We conduct an experiment in Section V-B3 to evaluate the impact of applying the HLC scheme over the flash-memory cache. Since flash memory has limited lifetime, the flash-memory cache size would shrink after sustained usage. As the size of flash-memory cache became smaller, the cache hit ratio drops. The experiment results showed that the flash-memory cache can achieve a stable cache hit ratio for read requests when our scheme is applied.

## V. PERFORMANCE EVALUATION

This section is meant to evaluate the impacts of applying the HLC scheme to a pure flash-memory-based SSD and a hybrid storage system, respectively.

1) For the pure flash-memory-based SSD, we conducted experiments to evaluate the HLC scheme in terms of the extended product lifetime and average response time. For the extended product lifetime, we compared the HLC scheme with the baseline SSD, i.e., SSD without any reviving scheme, Phoenix [14], and HLC+Phoenix. We also conducted an experiment to evaluate the impact of overprovision space with different sizes over the product lifetime.
2) For the hybrid storage system, we considered the cache hit ratio of read requests and average response times under three read-intensive traces. Since the HLC scheme can retard the diminishing of flash-memory cache, a more stable cache hit ratio could be guaranteed, from which a better average response time could be achieved. We also conducted an experiment to evaluate the impacts of different cache sizes over cache hit ratio and lifetime.

### A. Experimental Setup

The experiments were conducted by a trace-driven simulator. The simulator was designed from scratch. Our simulator takes the trace file as the input. Each line in the trace file represents a request by specifying the request type (read or write), the starting LBA of the request, and the number of sectors the request would access. A command-line interface is provided for users to configure the storage device, including the capacity of the device, the size of the overprovision

TABLE IV
CHARACTERISTICS OF TRACES [40]–[42]

| Trace Name | Total Requests | Write Ratio | Read Ratio | Update Ratio | Sequential Ratio | Average Size(KB) |
|---|---|---|---|---|---|---|
| Windows | 1,633,444 | 48.03% | 51.97% | 42.6% | 26.2% | 17.46 |
| Linux | 2,317,948 | 64.70% | 35.30% | 60.4% | 18.4% | 17.48 |
| Multimedia | 350,268 | 99.69% | 0.31% | 17.4% | 90.7% | 59.42 |
| Financial1 | 5,334,987 | 76.84% | 23.16% | 75.1% | 2.0% | 3.38 |
| Financial2 | 3,699,194 | 17.65% | 82.35% | 16.6% | 1.4% | 2.39 |

space, the endurance of the flash memory, etc. The chip we adopted in the experiment is Intel MD516 NAND flash memory [39]. Table III summarizes the timing parameter of each operation in the simulator. The error patterns of MLC were referenced from [23]. The simulator processes the trace line by line to simulate the behavior of the storage device. The statistic results would be outputted to files periodically.

The adopted trace suite for the pure flash-memory-based SSD includes the workload over Windows XP (Windows), the workload over Ubuntu 9 (Linux), the multimedia workload over Windows CE (Multimedia) [40], [41], and I/O traces from online transaction processing applications running at two large financial institutions (Financial1 and Financial2) [42]. The characteristics of the trace suite are listed in Table IV. The file system of Windows XP is NTFS, while the file system of Ubuntu 9 is Ext4. The activities of both traces include Internet surfing, video streaming, and e-mail sending/receiving. The average write sizes of Windows and Linux are also similar. However, the two traces represent the access patterns of different operating systems. The file system of Windows CE is FAT32. Both Multimedia and Financial1 are write-intensive traces. The major difference of the two traces is the size of their write requests: Multimedia is the trace of large sequential writes, while Financial1 is the trace of small random writes. Financial2 is a read-intensive trace. The access pattern of Financial1 and Financial2 are small and random.

Due to the out-place-update nature of flash memory, it is meaningless to improve cache hit ratio for write requests. It is because either cache hit or cache miss for a write request would result in a write operation to the flash-memory cache. Thus we should improve cache hit ratio for read requests. We adopted another trace suite for the performance evaluation of the hybrid storage system. The trace suite we adopted were three I/O traces from a popular search engine, including WebSearch1, WebSearch2, and WebSearch3 [42]. All the three Web search I/O traces are read-intensive (over 90% of requests are read requests).

For the experiments of the pure flash-memory-based SSD, the default setting was 64-GB SSD with additional 16-GB overprovision space. Typically, the overprovision space is reserved for bad-block replacement and garbage collection. Since it is used for bad-block replacement, the capacity of the overprovision space will continue to decrease. If the overprovision space is exhausted, it is regarded as the end of the product lifetime. In our experiment, the product lifetime is measured in terms of the total number of write requests that can be handled before the overprovision space is exhausted. For the experiments of the hybrid storage system, the physical capacity of the flash-memory cache is 6 GB. As the flash memory continues to wear, the cache size of the hybrid storage system would diminish, which result in the drop of cache hit ratio for read requests.

### B. Experiment Results

*1) Product Lifetime:* This section includes two parts.

1) We evaluate Phoenix [14] and the HLC scheme in terms of the product lifetime of SSDs.
2) We evaluate the impact of overprovision space with different sizes over the product lifetime of SSDs.

Although both Phoenix and the proposed HLC scheme are designed to reclaim bad pages for reuse, Phoenix is implemented in hardware level, while the HLC scheme is implemented in software/firmware level. Since the two schemes are implemented in different levels, we can combine both schemes to further extend the product lifetime of SSDs, referred to as HLC+Phoenix. The basic idea is as follows.

1) When an MLC page is about to become a bad page, transform it to an SLC-mode page by Phoenix.
2) Manage SLC-mode pages with the proposed page-set status table.
3) When a pair of SLC-mode pages is about to become bad pages, apply the proposed HLC scheme to reclaim them.

Notably, although Phoenix must reclaim worn-out space in block level due to hardware constraint, we assume it could reclaim worn-out space in page level in the simulation to simplify the design of HLC+Phoenix. However, even with this flexibility, HLC still outperformed Phoenix in terms of extended product lifetime.

Fig. 12 shows the relationship between the number of processed requests and the corruption status of flash-memory pages. Through this figure, we could know handling how many requests would make the storage device begins to suffer from bad pages and how many requests would exhaust the overprovision space. It can also be observed that the SSD with a reviving scheme, including HLC, Phoenix, or HLC+Phoenix, can retard the corruption rate of the overprovision space and prolong its product lifetime. As shown in Fig. 12, the SSD with any reviving scheme could have a longer product lifetime than the baseline SSD, i.e., the SSD without any reviving scheme; SSD with HLC performed better than SSD with Phoenix, while SSD with HLC+Phoenix achieved the longest product lifetime.

In Fig. 12(a), the SSD began to suffer from bad pages when the number of handled requests reaches 2.7E+10 under the workload of Windows. When bad pages were detected, Phoenix and HLC+Phoenix began to revive MLC pages as SLC-mode pages, while the HLC scheme reclaimed bad-page sets for reuse. The SSDs with reviving schemes can make

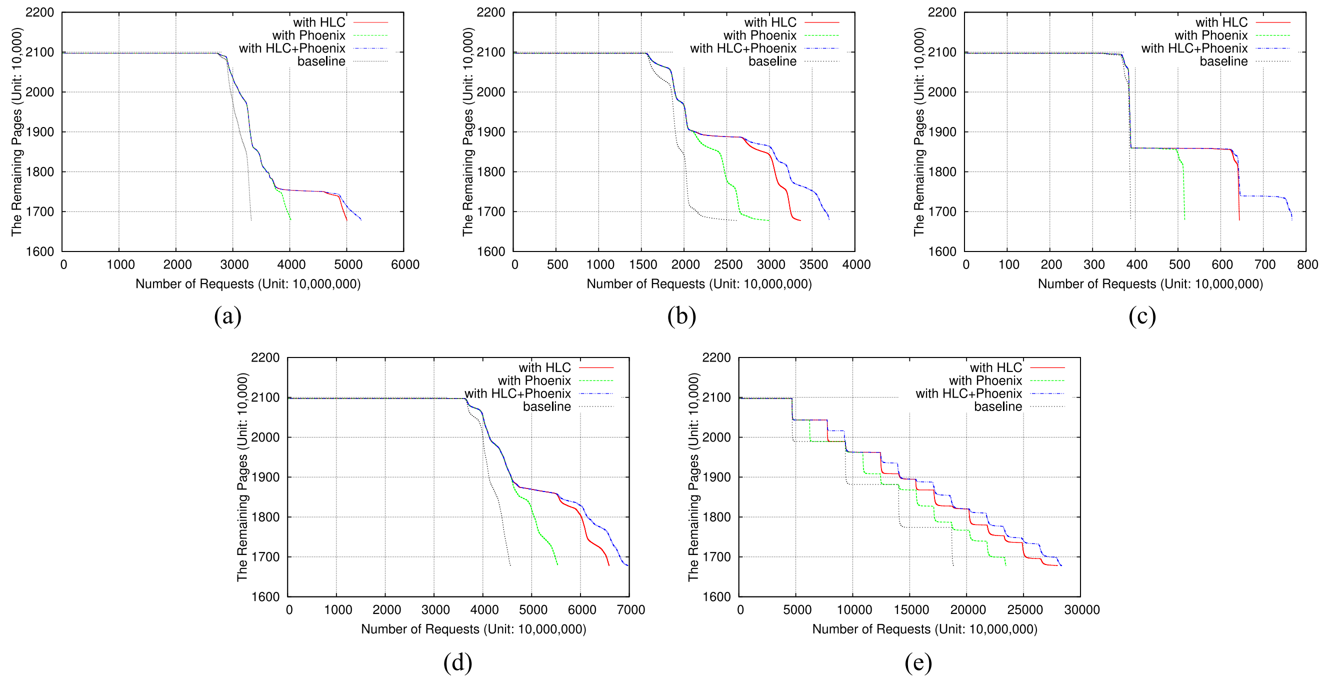


Fig. 12. Comparison of product lifetimes in terms of processed request counts under various traces. (a) Windows: extended lifetime. (b) Linux: extended lifetime. (c) Multimedia: extended lifetime. (d) Financial1: extended lifetime. (e) Financial2: extended lifetime.

TABLE V
IMPROVEMENT OF LIFETIME

| | Total Number of Requests | | | | Improvement | | |
|---|---|---|---|---|---|---|---|
| | baseline | with HLC | with Phoenix | with HLC+Phoenix | with HLC | with Phoenix | with HLC+Phoenixe |
| Windows | 33,244,301,495 | 50,055,208,784 | 40,267,755,851 | 52,570,962,613 | 50.56% | 21.12% | 58.13% |
| Linux | 26,212,830,249 | 33,660,170,700 | 30,045,929,196 | 37,022,183,082 | 28.41% | 14.62% | 41.23% |
| Multimedia | 3,891,314,473 | 6,438,264,570 | 5,158,625,988 | 7,672,251,224 | 65.45% | 32.56% | 97.16% |
| Financial1 | 45,682,147,176 | 65,878,335,105 | 55,391,461,870 | 69,845,452,131 | 44.21% | 21.25% | 52.89% |
| Financial2 | 188,484,611,588 | 279,774,955,869 | 234,634,532,637 | 283,515,071,471 | 48.43% | 24.48% | 50.41% |

the corruption rate slower than the baseline SSD. When the number of handled requests was about 3.3E+10, the baseline SSD exhausted its overprovision space; on the other hand, when HLC (/Phoenix) was applied, the SSD can endure 5E+10 (/4E+10) requests; the gain from HLC (/Phoenix) was about 1.7E+10 (/7E+9) requests. Fig. 12(b) is the experiment result under the workload of Linux; it shows that the request gain of the SSD with HLC (/Phoenix) was 7.44E+9 (/3.83E+9). Different from above two traces, the access pattern of Multimedia was large sequential writes, thus the wear status among flash-memory pages were even. The remaining pages exhausted rapidly when the number of handled requests exceeded 3.89E+9. With HLC (/Phoenix), the product lifetime can be extended to endure 6.43E+9 (/5.15E+9) requests, as shown in Fig. 12(c).

In Fig. 12(d) and (e), since most accesses of Financial1 and Financial2 were small and random, the utilization of bad-page sets was better than that of the multimedia workload; the gains of the number of requests from HLC (/Phoenix) were about 2.01E+10 (/9.7E+9) for Financial1 and 9.12E+10 (/4.61E+10) for Financial2. Compared with Phoenix, the proposed HLC scheme have better performance. It was because HLC scheme has more flexibility to chose the strong segment to store data. The improvement of product lifetime

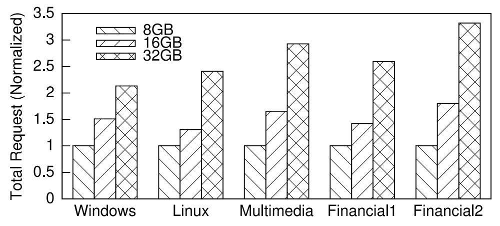


Fig. 13. Impact of different overprovision sizes over product lifetime.

achieved by Phoenix ranged from 14.62% to 32.56%, while the HLC scheme can achieve from 28.41% to 65.45%. The HLC+Phoenix performed the best with the improvement ranged from 41.23% to 97.16%. The ratios of the lifetime improvement under different traces are shown in Table V.

We also conducted an experiment to evaluate the impact of overprovision space with different sizes over the product lifetime. Given a 64-GB SSD, suppose an extra overprovision space with 8, 16, and 32 GB were provided, respectively. We run five traces over the above three settings with the HLC scheme to observe the relationship between the reserved space size and the SSD lifetime. As shown in Fig. 13, more reserved space will further extend the lifetime. When we adjusted the size of overprovision space from 8 to 16 GB, the lifetime can

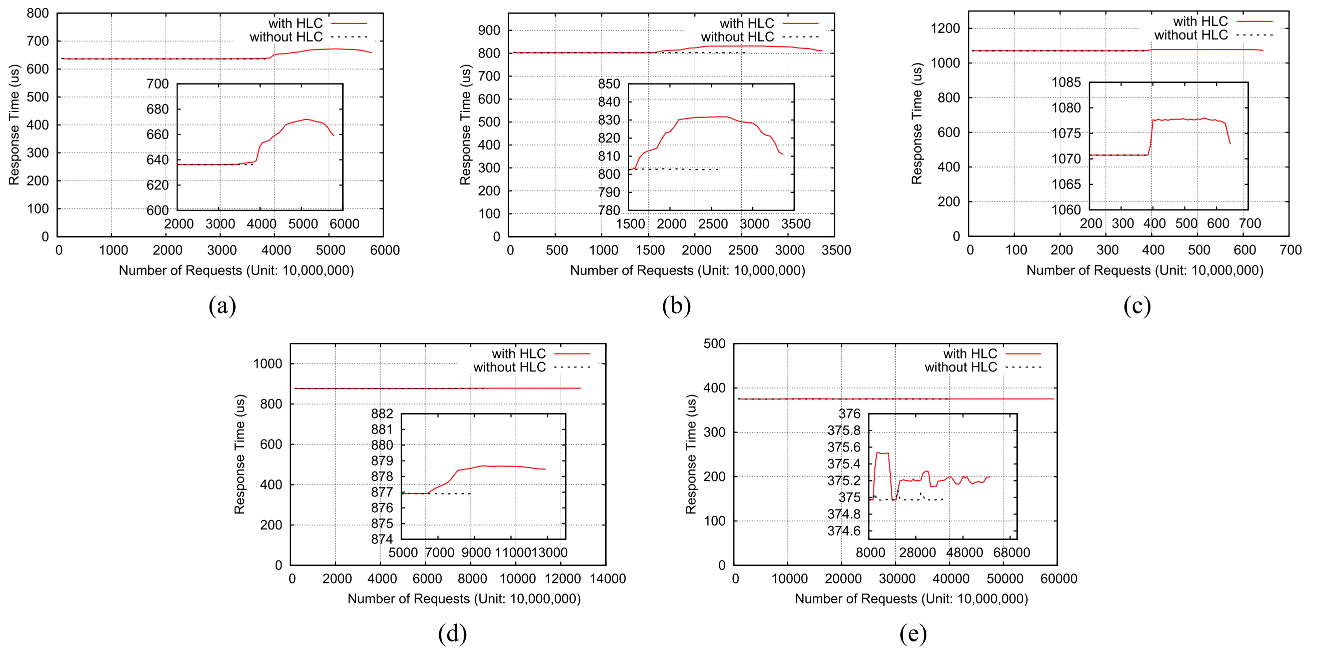


Fig. 14. Comparison of average response times under various traces. (a) Windows: average response time. (b) Linux: average response time. (c) Multimedia: average response time. (d) Financial1: average response time. (e) Financial2: average response time.

be extended about 1.5 times; if the size was enlarged from 8 to 32 GB, the lifetime can be further extended about 2∼3 times.

*2) Average Response Time:* This section is meant to evaluate the overhead of applying the HLC scheme to SSDs in terms of average response time. We did not conduct an experiment to evaluate access latency and bandwidth of Phoenix and the HLC scheme since SLC-mode pages would definitely have a better performance than MLC pages. Since the HLC scheme adopts multilevel parallelism to serve write requests, we could multiply the number of available planes that could serve the request by one page size to derive the amount of data that could be written in a page program time. Thus, the response time of a write request was estimated by $\lceil w_{\text{size}}/(p_{\text{avail}} \times \text{pg}_{\text{size}})\rceil \times t_{\text{prog}}$, where $w_{\text{size}}$ is the size of the write request, $p_{\text{avail}}$ is the number of available planes, $\text{pg}_{\text{size}}$ is the size of a page, and $t_{\text{prog}}$ is the program time of a page. Different from a write request that could allocate available planes dynamically, the planes involved in a read request depend on where the requested data resided. Thus, the response time of a read request was dominated by the involved plane which required the maximum number of page read; it was estimated by $\max_{i\in\pi(r)}\{\text{pg}_{\text{read}}(i)\} \times t_{\text{read}}$, where $\pi(r)$ is the subset of planes involved in the read request $r$, $\text{pg}_{\text{read}}(i)$ is the number of page reads required in the plane $i$ to serve the request, and $t_{\text{read}}$ is the read time of a page.

Fig. 14 compares the average response times under various traces. As shown in the figure, the average response times were almost the same in the early stage, whether the HLC scheme was applied or not. It was because bad pages were rare in the early stage. In the middle stage when bad pages emerged, the HLC scheme began to serve requests with bad page sets. Since two-plane commands were not always applicable, there was minor performance degradation in the middle stage (compared with that in the early stage). Although SSD with the HLC scheme suffered minor performance degradation in the middle stage, it had extended product lifetime as return.[8] During the end of its product lifetime, SSD with the HLC scheme had better average response time compared with that in the middle stage. It was because bad pages worn out gradually and could not be utilized by the HLC scheme anymore; thus requests were served by normal pages again.

*3) Cache Hit Ratio of Read Requests:* In addition to pure flash-memory-based SSDs, the proposed HLC scheme can also be easily applied in the hybrid storage systems. In this section, we evaluated the performance impacts of applying the HLC scheme to flash-memory cache of HDD in terms of cache hit ratio of read requests and average response times under three read-intensive traces, including WebSearch1, WebSearch2, and WebSearch3. We also conducted an experiment to evaluate the impact of different cache sizes.

As shown in Fig. 15(a), for a 6-GB flash-memory cache without the HLC scheme, the cache hit ratio of read requests began to drop when the number of handled requests reached 1.38E+10 under WebSearch1. It was because the cache size shrunk due to bad pages emerged. When the number of handled requests exceeded 3.68E+10, the deterioration of the cache hit ratio became severer. As a result, the average response time of the storage system increased rapidly. The deterioration of the cache hit ratio and average response time could be mitigated by the proposed HLC scheme. As shown in the figure, the cache hit ratio could be improved by 2.2%, while improvement of the average response time could achieve up to 20.57%. Similar experiment results could also be observed under WebSearch2 and WebSearch3, as shown in Fig. 15(b) and (c). For WebSearch2 (/WebSearch3), when the

[8]SSD without the HLC scheme quickly ran out its overprovision space and its product lifetime ended soon in the middle stage.

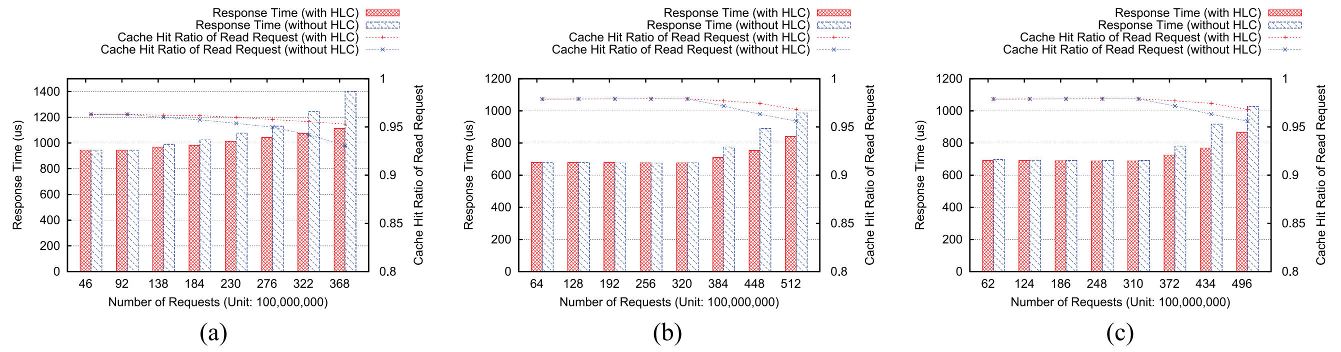


Fig. 15. Comparison of cache hit ratio of read requests and average response times under various traces. (a) WebSearch1: cache hit ratio and average response time. (b) WebSearch2: cache hit ratio and average response time. (c) WebSearch3: cache hit ratio and average response time.

TABLE VI
IMPACTS OF DIFFERENT CACHE SIZES OVER CACHE HIT RATIO AND LIFETIME OF HLC-BASED FLASH-MEMORY CACHE

| | WebSearch 1 | | |
|---|---|---|---|
| | 2GB | 4GB | 6GB |
| Cache Hit Ratio | 27.95% | 81.95% | 96.29% |
| Total Requests | 3,457,949,680 | 13,894,142,078 | 60,791,216,798 |
| | WebSearch 2 | | |
| | 2GB | 4GB | 6GB |
| Cache Hit Ratio | 28.94% | 72.69% | 97.88% |
| Total Requests | 3,512,384,509 | 14,392,922,505 | 75,299,242,507 |
| | WebSearch 3 | | |
| | 2GB | 4GB | 6GB |
| Cache Hit Ratio | 27.91% | 72.38% | 97.75% |
| Total Requests | 3,384,154,639 | 13,745,052,980 | 72,015,486,624 |

number of handled requests reached 5.12E+10 (/4.96E+10), the cache hit ratio and the average response time could be improved by 1.21% (/1.35%) and 17.78% (/15.63%) with the proposed HLC scheme.

We also conducted an experiment to evaluate the impacts of different cache sizes over cache hit ratio and lifetime of HLC-based flash-memory cache; the lifetime was measured in terms of the total number of write requests that can be handled until the cache size shrunk to 1/128 of its original size. As shown in Table VI, a larger cache size resulted in a higher cache hit ratio and longer lifetime. When the cache size reached 6 GB, the cache hit ratio was saturated. At this moment, further increasing cache size would not improve the cache hit ratio, but it could continue to improve lifetime. Taking the cost of the storage system into consideration, if your goal is to improve performance, you need to analyze your workload to know how large the cache size can satisfy your requirement. If the lifetime is your major concern, you should have your cache size as large as possible.

## VI. CONCLUSION

In this paper, we propose the HLC scheme to extend the product lifetime of flash-memory storage devices by utilizing strong half-page segments from two bad pages to store one-page data. By padding dummy data to the other half of pages, we could further extend the ECC capability without changing the hardware of ECC encoder/decoder. In our design, four major components are adopted to maintain and revive the corruption space in flash memory. *Bad page recorder* is responsible for monitoring the error status and recording the physical address of bad pages. *Data separator* cooperates with *data masker* to transform one-page data so that the transformed data can be stored in bad pages reliably. *Data assembler* is in charge of recovering the stored data correctly from bad pages. Based on our design, we could easily extend the HLC scheme to any kind of memory that also has limited lifetime. In addition to pure flash-memory-based SSDs, the proposed HLC scheme can also be easily applied in the hybrid storage systems.

The experiment results showed that our approach could extend the product lifetime of SSDs up to 65.45% under the workload of large sequential writes, 44.21% for the workload of small random writes, 50.56% for the workload of Windows, and 28.41% for the workload of Linux. We evaluate the overhead of applying the HLC scheme by examining the average response times. The experiment result showed that the average response times were almost the same in the early stage, whether the HLC scheme was applied or not. When bad pages were revived by the HLC scheme in the middle stage, only minor performance was sacrificed. When applying the HLC scheme to flash-memory cache of HDD, the cache hit ratio could be improved, while the improvement of the average response time could achieve up to 20.57% for the trace of WebSearch1.

## REFERENCES

[1] S. Jung, S. Lee, H. Jung, and Y. H. Song, "In-page management of error correction code for MLC flash storages," in *Proc. IEEE MWSCAS*, Seoul, South Korea, Aug. 2011, pp. 1–4.

[2] Y.-H. Chang and T.-W. Kuo, "A reliable MTD design for MLC flash-memory storage systems," in *Proc. ACM EMSOFT*, Scottsdale, AZ, USA, 2010, pp. 179–188.

[3] J.-W. Hsieh, C.-W. Chen, and H.-Y. Lin, "Adaptive ECC scheme for hybrid SSD's," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3348–3361, Dec. 2015.

[4] M.-C. Yang, Y.-H. Chang, C.-W. Tsao, and P.-C. Huang, "New ERA: New efficient reliability-aware wear leveling for endurance enhancement of flash storage devices," in *Proc. ACM/IEEE DAC*, Austin, TX, USA, May/Jun. 2013, pp. 1–6.

[5] Y.-J. Woo and J.-S. Kim, "Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs," in *Proc. ACM/IEEE EMSOFT*, Montreal, QC, Canada, Sep. 2013, pp. 1–10.

[6] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. USENIX FAST*, 2011, pp. 77–90.

[7] Y. Park and J.-S. Kim, “zFTL: Power-efficient data compression support for NAND flash-based consumer electronics devices,” *IEEE Trans. Consum. Electron.*, vol. 57, no. 3, pp. 1148–1156, Aug. 2011.

[8] G. Wu and X. He, “Δ-FTL: Improving SSD lifetime via exploiting content locality,” in *Proc. ACM EuroSys*, Bern, Switzerland, Apr. 2012, pp. 253–266.

[9] X. Zhang *et al.*, “Realizing transparent OS/apps compression in mobile devices at zero latency overhead,” *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1188–1199, Jul. 2017.

[10] S. Lee, T. Kim, J.-S. Park, and J. Kim, “An integrated approach for managing the lifetime of flash-based SSDs,” in *Proc. IEEE DATE*, Grenoble, France, Mar. 2013, pp. 1522–1525.

[11] L. Zhang *et al.*, “Mellow writes: Extending lifetime in resistive memories through selective slow write backs,” in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, Seoul, South Korea, Jun. 2016, pp. 519–531.

[12] D. B. Strukov, “Endurance-write-speed tradeoffs in nonvolatile memories,” *Appl. Phys. A, Solids Surf.*, vol. 122, no. 302, pp. 1–4, Mar. 2016.

[13] Z. Deng, L. Zhang, N. Mishra, H. Hoffmann, and F. T. Chong, “Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in NVMs,” in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Cambridge, MA, USA, Oct. 2017, pp. 232–244.

[14] X. Jimenez, D. Novo, and P. Ienne, “Phoenix: Reviving MLC blocks as SLC to extend NAND flash devices lifetime,” in *Proc. IEEE DATE*, Grenoble, France, Mar. 2013, pp. 226–229.

[15] J.-W. Hsieh, Y.-H. Chang, and Y.-S. Chu, “Implementation strategy for downgraded flash-memory storage devices,” *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1s, p. 60, 2013.

[16] C. Wang and W.-F. Wong, “Extending the lifetime of NAND flash memory by salvaging bad blocks,” in *Proc. IEEE DATE*, Dresden, Germany, Mar. 2012, pp. 260–263.

[17] Y.-T. Chiu, “Forever flash,” *IEEE Spectr.*, vol. 49, no. 12, pp. 11–12, Dec. 2012.

[18] H.-T. Lue *et al.*, “Radically extending the cycling endurance of flash memory (to >100M cycles) by using built-in thermal annealing to self-heal the stress-induced damage,” in *Proc. IEEE IEDM*, San Francisco, CA, USA, Dec. 2012, pp. 199–202.

[19] L.-P. Chang, S.-M. Huang, and K.-L. Chou, “Relieving self-healing SSDs of heal storms,” in *Proc. ACM SYSTOR*, Haifa, Israel, May 2017, pp. 1–7.

[20] R. Chen, Y. Wang, D. Liu, Z. Shao, and S. Jiang, “Heating dispersal for self-healing NAND flash memory,” *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 361–367, Feb. 2017.

[21] Y.-M. Chang *et al.*, “On trading wear-leveling with heal-leveling,” in *Proc. ACM/IEEE DAC*, San Francisco, CA, USA, Jun. 2014, pp. 1–6.

[22] K. Lee and A. Orailoglu, “High durability in NAND flash memory through effective page reuse mechanisms,” in *Proc. IEEE/ACM/IFIP CODES+ISSS*, Scottsdale, AZ, USA, Oct. 2010, pp. 205–211.

[23] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” in *Proc. ACM/IEEE DATE*, Dresden, Germany, Mar. 2012, pp. 521–526.

[24] J.-U. Kang, J.-S. Kim, C. Park, H. Park, and J. Lee, “A multi-channel architecture for high-performance NAND flash-based storage system,” *J. Syst. Archit.*, vol. 53, no. 9, pp. 644–658, Sep. 2007.

[25] Y. Hu *et al.*, “Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance,” *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1141–1155, Jun. 2013.

[26] J.-W. Hsieh, H.-Y. Lin, and D.-L. Yang, “Multi-channel architecture-based FTL for reliable and high-performance SSD,” *IEEE Trans. Comput.*, vol. 63, no. 12, pp. 3079–3091, Dec. 2014.

[27] “NAND flash memory MT29F4G08AAA, MT29F8G08BAA, MT29F8G08DAA, MT29F16G08FAA,” Micron, Boise, ID, USA, Rep., 2006.

[28] “Improving performance using two-plane commands introduction,” Micron, Boise, ID, USA, Rep. TN-29-25, 2007.

[29] “NAND flash memory K9GAG08B0M, K9GAG08U0M, K9LBG08U1M,” Samsung, Seoul, South Korea, Rep. K9XXG08UXM, 2007.

[30] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND Flash Memories*. Dordrecht, The Netherlands: Springer, 2010.

[31] *AN1823: ECC in SLC NAND Flash Memory*, Micron Technol., Boise, ID, USA, 2010. [Online]. Available: http://www.beilenet.com/download/AN1823.pdf

[32] S. Jung, S. Lee, H. Jung, and Y. H. Song, “In-page management of error correction code for MLC flash storages,” in *Proc. IEEE MWSCAS*, Seoul, South Korea, Aug. 2011, pp. 1–4.

[33] Y. Cai *et al.*, “Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime,” in *Proc. IEEE ICCD*, Montreal, QC, Canada, Sep. 2012, pp. 94–101.

[34] “Solid state drives data reliability and lifetime white paper,” Sandisk, Milpitas, CA, USA, Rep., 2008.

[35] L. M. Caulfield, “Symbiotic solid state drives: Management of modern NAND flash memory,” Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. California, San Diego, San Diego, CA, USA, 2013.

[36] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings,” *SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, 2009.

[37] Z. Qin, Y. Wang, D. Liu, and Z. Shao, “Demand-based block-level address mapping in large-scale NAND flash storage systems,” in *Proc. IEEE/ACM/IFIP CODES/ISSS*, Scottsdale, AZ, USA, Oct. 2010, pp. 173–182.

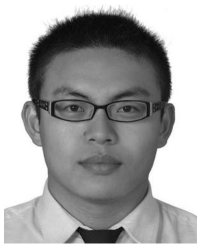
[38] R.-S. Liu, C.-L. Yang, C.-H. Li, and G.-Y. Chen, “Duracache: A durable SSD cache using MLC NAND flash,” in *Proc. ACM/IEEE DAC*, Austin, TX, USA, 2013, p. 166.

[39] “Intel® MD516 NAND flash memory JS29F16G08AAMC1, JS29F32G08CAMC1, JS29F64G08FAMC1,” Intel, Santa Clara, CA, USA, Rep. 316339-001US, 2007.

[40] L.-P. Chang, “A hybrid approach to nand-flash-based solid-state disks,” *IEEE Trans. Comput.*, vol. 59, no. 10, pp. 1337–1349, Oct. 2010.

[41] L.-P. Chang and C.-D. Du, “Design and implementation of an efficient wear-leveling algorithm for solid-state-disk microcontrollers,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 15, no. 1, 2009, Art. no. 6.

[42] *UMass Trace Repository*. Accessed: Jun. 1, 2007. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage



**Han-Yi Lin** (S’17) received the M.S. degree from the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, in 2014. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei.

His current research interests include flash-memory storage systems and mobile-embedded systems.

**Jen-Wei Hsieh** (SM’15) received the B.S., M.S., and Ph.D. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1999, 2001, and 2006, respectively.

He is currently a Professor with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei. His current research interests include flash-memory storage systems and real-time task scheduling.

Prof. Hsieh served as the Program Co-Chair of the 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications in 2017.