HLC: Software-Based Half-Level-Cell Flash Memory

Han-Yi Lin
National Taiwan University
Taipei, Taiwan
Email: m10115011@mail.ntust.edu.tw

Jen-Wei Hsieh
National Taiwan University of Science and Technology
Taipei, Taiwan
Email: jenwei@mail.ntust.edu.tw

***Abstract*—In recent years, flash memory has been widely used in embedded systems, portable devices, and high-performance storage products due to its non-volatility, shock resistance, low power consumption, and high performance natures. To reduce the product cost, multi-level-cell flash memory (MLC) has been proposed; compared with the traditional single-level-cell flash memory (SLC) 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 storage devices from a totally different perspective. We propose a half-level-cell (HLC) management scheme to manage and reuse the worn-out space in solid-state drives (SSDs); through our management scheme, the system can treat two corrupted pages as a normal page without sacrificing performance and reliability. To the best of our knowledge, this is the first research that reclaims free space by reviving the corrupted pages. The experiment results show that the lifetime of SSD can be extended by 48.54% for the trace of general users applications with our proposed HLC management scheme.**

I. Introduction

In recent years, flash memory has been widely used in embedded systems, portable devices, and high-performance storage products due to its non-volatility, shock resistance, low power consumption, and high performance natures. Although flash memory has many advantages, it also has some inherent characteristics that affect the performance. Flash memory is a kind of electrically-erasable programmable read-only memory (EEPROM) that must conduct an erase operation over a block before a program operation can be performed on its page. The asymmetric program/erase units make in-place-update impractical; thus out-place-update is usually adopted in the management. A mapping table is required to keep track of the current version of data, and garbage collection is required to reclaim free space from old version of data.

In addition to the above-mentioned characteristics, flash memory also has the reliability issue. A flash memory block might suffer from frequent read/write errors after sustaining about 3,000 program/erase (P/E) cycles. If these error bits cannot be corrected by the associated error correction code (ECC), the 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.

Considering the reliability issue, various researches have been proposed. Some researches exploit data de-duplicate or data compression to reduce the write traffic [1], [2], [3]. In [1], a Content-Aware Flash Translation Layer (CAFTL) is proposed to reduce write traffic by removing unnecessary duplicate writes and extend available free flash memory space by coalescing redundant data in SSDs. In [2], zFTL is 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 is designed to minimize compression overhead and power consumption. In [3], Lee et al. combine several lifetime-enhancement schemes, including de-duplicate, lossless compression, and performance throttling, in an integrated fashion so that the lifetime of SSDs can be maximally extended.

Some researches focus on distributing P/E cycles evenly across all blocks of flash memory [4], [5]. In [4], Yang et al. noted that flash blocks endured the same P/E cycles usually have different numbers of error bits. Thus they propose 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 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 use a more sophisticated wear index to indicate the wear status of all the blocks [6]. They found that the program/erase latencies have a relation with the degree of page wear. In their method, both bit error rates and program/erase latencies are taken into consideration to produce a more precise wear index, from which a better wear leveling can be achieved.

Other researches focus on the reclamation of downgraded flash memory. MLC requires the controller to be able to identify four different states to store 2-bit data per cell. In [5], Jimenez et al. proposed to revive those flash blocks that are becoming unreliable to store multiple bits per cell by storing only a single bit per cell. 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. In [7], Hsieh et al. present a set-based mapping strategy with low hardware resource requirements for making downgraded flash-memory chips useable in products.

In this paper, we propose a software-based half-level-cell

978-3-9815370-4-8/DATE15/©2015 EDAA

(HLC) management scheme to improve the reliability and extend the product lifetime of SSDs. Different from previous researches, our scheme focus on managing and reusing the already worn-out space. 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. By reusing those corrupted space and enhancing the ECC capability, we could obtain more available space to reduce the burden on other normal space. To the best of our knowledge, this is the first research to recycle those already worn-out pages. The rest of this paper is organized as follows: We present the detail of our research in Section II and evaluate its performance in Section III. Finally, we conclude the paper in Section IV.

II. Design of HLC Management Scheme

A. Overview of HLC

The design goal of HLC management scheme is to extend the lifetime 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] Since extra free space can be provided from bad pages by our scheme, garbage collection would be triggered less frequently. Thus the lifetime of flash-memory storage devices can be further extended.

Fig. 1 illustrates the overall system architecture. The proposed HLC management scheme is introduced between flash translation layer (FTL) and multi-chipped memory technology device (MTD) layer. FTL realizes high-level management algorithms, including logical/physical address translation, garbage collection, and wear leveling policy. Low-level functionalities of flash memory, including read, write, and erase, are implemented in MTD layer. With the layered design, hardware manufacturers can integrate HLC management scheme into their products easily without significant modifications.

The HLC management 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 HLC management 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 MTD layer and record the physical page address of bad pages. In our design, two bad pages (referred to as a *bad-page set*) are combined 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 II-C. 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 command is then issued to write two bad pages in parallel. Since 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.

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

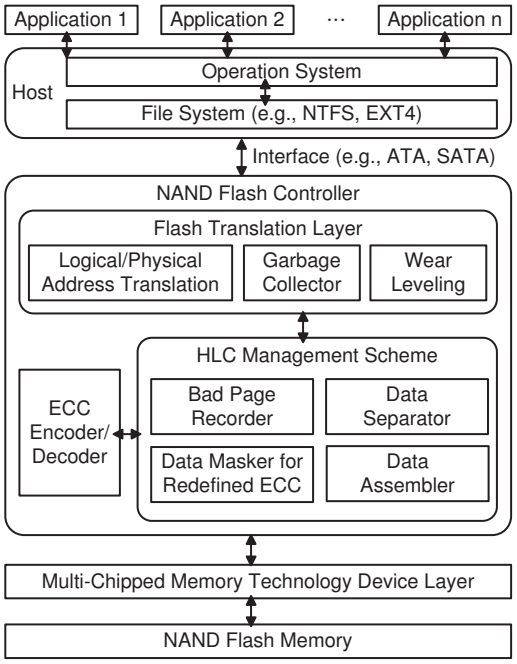


Fig. 1. System Architecture

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. A flash chip consists of multiple dies, and each die contains multiple planes. A plane is composed of a cache register, a data register, and a fixed number of blocks. 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., [8], [9], [10], [11], have built-in two-plane commands in their products.

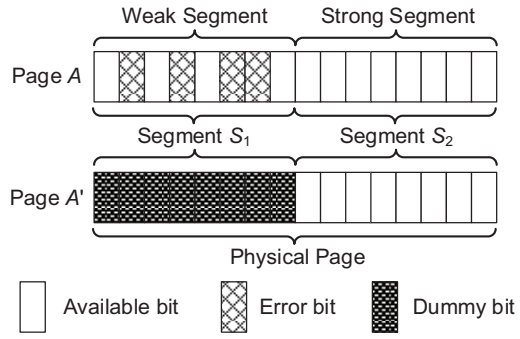
There is a restraint when we conduct two-plane commands on two pages/blocks: offsets of the two pages/blocks must be identical in two planes of the same die. Taking this restraint into consideration, we adopt two-plane commands to serve requests [12]. As a result, the wear statuses of pages in a page set are almost the same. Thus we can easily find two bad pages in the same page set. When the proposed HLC management scheme transforms one-page data into two-page data and writes them into flash memory, the two-plane write command is issued to deal with the two-page data. With two-plane commands, bad pages can be reclaimed to extend lifetime of flash-memory storage devices with minimized performance overhead.

B. HLC Data Structure

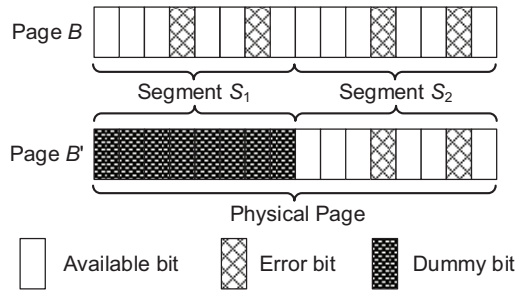
A bad page is the page that has more errors than that can be corrected by its ECC; it is no longer available for normal use. However, the proposed HLC management scheme could reclaim bad pages for further reuse. Suppose the number of errors that the ECC can correct is no more than n bits. As long as the number of error bits in a bad page does not exceed $2n$, we can always reclaim two such pages to store one-page

data. We divide 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.[2]

Fig. 2 illustrates the concept of bad page and segments. Suppose page A has 4 bit errors, and its adopted ECC can correct only 2 bit errors. Since the ECC is incapable of correcting all errors, page A is regarded as a bad page and cannot be used normally. Our HLC management scheme divides page A into two equal-sized segments S_1 and S_2, as shown in Fig. 2(a). The segment with less(/more) error bits is regarded as the strong(/weak) segment. In the best case, all the error bits might occur in one segment, leaving the strong segment with no errors, e.g., S_2 in page A. In the worst case, all the error bits occur evenly in two segments, as shown in Fig. 2(b). However, if the adopted ECC is dedicated to one segment, either S_1 or S_2 in page B, we can still guarantee the reliability of data in the segment.



(a) Best Case for Selecting Segment.



(b) Worst Case for Selecting Segment.

Fig. 2. Bad Page and Segments.

Since the available space of a bad page is now halved, HLC management scheme groups two bad pages as a bad-page set to serve one-page data. Fig. 3 illustrates the layout of bad-page sets. Two bad pages which have the same offset from two 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 will be discussed in the next two sections.

[2]The number of error bits in the other segment is from $2n$ to n, thus the total number of error bits is equal to $2n$.

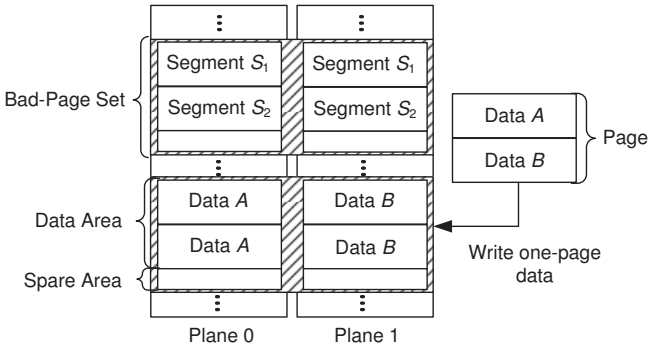


Fig. 3. Layout of Bad-Page Set.

C. Write Flow of Bad-Page Set

Fig. 4 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, the data is divided into two equal-sized parts; the size of each part is equal to the size of a segment. Each part of data will be arranged to store in the strong segment of different pages in a bad-page set. Instead 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 the total number of error bits does not exceed $2n$.

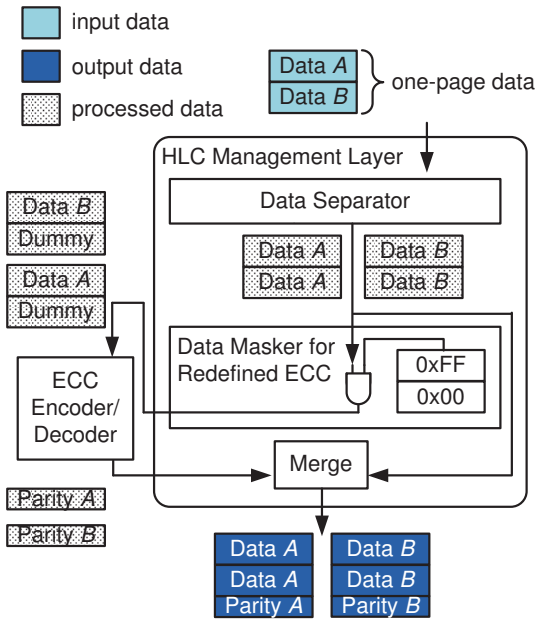


Fig. 4. The Process of Data Write in Bad-Page Set.

Notably, HLC management scheme does not send the data generated by data separator to ECC encoder directly. Instead,

data masker will generate the input to ECC encoder so that the strong segment can be fully protected by the ECC. 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 the bad-page set, a read-after-write operation is conducted to check the correctness of the data in the page, from which the strong segment and the weak segment can be identified. We can record which segment is strong in RAM space to help us quickly find the strong segment during a read operation. If we lose the information due to a power failure or a system crash, we can still use *ECC decoder* to detect which segment holds the correct data.

D. 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*. Fig. 5 illustrates the process of data read in a bad-page set. We need to obtain two parts of data, i.e., Data A and Data B, from strong segments of 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 strong flags that are set during the write operation. With data assembler, data from two strong segments can be combined to recover the required data.

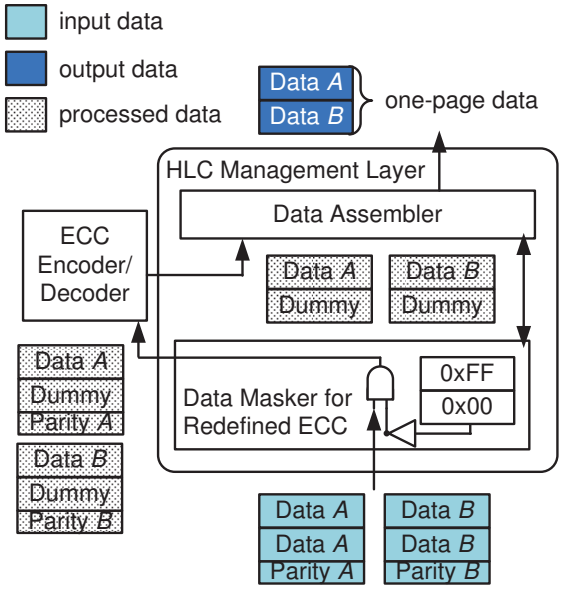


Fig. 5. The Process of Data Read in Bad-Page Set.

However, if the storage device has just suffered a power failure or a system crash, we would lose the information of strong flags stored in RAM space. 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 assembler treats it as S_1 in a page with S_2 filled with all 0. (3) The assembled page would be send to ECC decoder to detect 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 the weak segment. Data assembler would request data masker to mask out the other segment and continued from Step (2) to Step (3).[3] (5) After the strong segment in 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, as shown in Fig. 5.

E. Redefined ECC

In this section, we will discuss how to extend the *ECC capability*, i.e., the number of error bits that can be corrected by ECC, 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 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, 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 to protect one-page data, we use two normal-sized ECCs to protect two half-page data. We add dummy bits with all 0s to the half of one-page data to fit the input format of ECC encoder. With this approach, we can know half content of the data in advance, thus the ECC only needs to protect the other half of the data.

The whole process of ECC encoding is illustrated in Fig. 4. Since the strong segment in a bad page cannot be determined before the read-after-write operation is conducted, each part of data is duplicated to ensure the data can be written to the strong segment. However, the duplicated data cannot be submitted to ECC encoder directly; otherwise, the ECC capability could not be extended. For the content to be submitted to ECC encoder, S_1 is always allocated for the half-page data while S_2 is padded with 0s by data masker. After the ECC is generated, HLC management scheme merges it with the duplicated data and writes the merged data to a bad-page set.

III. Performance Evaluation

This section is meant to evaluate the performance of the proposed HLC management scheme. The proposed HLC management scheme can accommodate to any kind of address translation schemes, including page-level mapping, block-level mapping, and hybrid mapping. It can also be applied to any kind of NAND flash memory cells, e.g., SLC and MLC. In the experiment, we adopt page-level mapping and MLC for the simulation.

[3] 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.

TABLE I. PARAMETER SETTING OF FLASH-MEMORY CHIP [13].

Operation	Total Time
Read	136.42μs
Two-Plane Read	222.96μs
Write	986.46μs
Two-Plane Write	1073.38μs
Block Erase	2000.14μs
Two-Plane Block Erase	2000.74μs

A. Experiment Setup

The experiment is conducted by a trace-driven simulator. The configuration of the simulated flash memory in our simulator is listed in Table I. The physical capacity of the simulated SSD is 80GB with 20%, i.e., 16GB, reserved as the *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.

The adopted trace suite includes the workload over Windows XP (Windows), the workload over Ubuntu 9 (Linux), the multimedia workload over Windows CE (Multimedia) [14], [15], and I/O traces from on-line transaction processing (OLTP) applications running at two large financial institutions (Financial1 and Financial2) [16]. The characteristics of the trace suite are listed in Table II. 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.

B. Experiment Results

We evaluate the performance of HLC management scheme by comparing the product lifetime of SSDs with/without our approach. Since our approach can revive bad pages, additional space can be obtained to apportion write burden of the storage system to prolong its lifetime. Fig. 6 shows the relationship between the number of processed requests and the corruption status of flash-memory pages. Through this figure, we could know how many requests will make the storage device begins to suffer from bad pages and how many requests will exhaust the overprovision space. It can also be observed that the SSD with our proposed HLC management scheme can retard the corruption rate of the overprovision space and prolong its product lifetime.

As shown in Fig. 6(a), the SSD began to suffer from bad pages when the number of requests reaches 3E+10 under the workload of Windows. When bad pages are detected, our approach began to reclaim them for reuse. The SSD with our approach can make the corruption rate slower than the one without our approach. When the number of requests is about 3.9E+10, the SSD without our approach exhausted its overprovision space; on the other hand, when our approach was applied, the SSD can endure 5.79E+10 requests; the gain is about 1.89E+10 requests. Fig. 6(b) is the experiment result under the workload of Linux; it shows that the request gain of the SSD with our approach is 7.45E+9. Different from above two traces, the access pattern of Multimedia is large sequential writes, thus the wear statuses among flash-memory pages are even. The remaining pages exhausted rapidly when the number of processed requests exceeded 3.90E+9. With our approach, the product lifetime can be extended to endure 6.45E+9 requests, as shown in Fig. 6(c). In Fig. 6(d) and Fig. 6(e), since most accesses of Financial1 and Financial2 are small and random, the utilization of bad-page sets is better than that of the multimedia workload; the gains of the number of requests are about 4E+10 and 19.32E+10, respectively. The ratios of the lifetime improvement under different traces are shown in Table III.

IV. CONCLUSION

In this paper, we present the HLC management scheme to extend the product lifetime of flash-memory storage devices. We utilize strong half-page segment from two bad pages to store one-page data. By padding dummy data to the other half of pages, we could 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 management scheme to any kind of memory that also has limited lifetime. To the best of our knowledge, this is the first research that reclaims free space by reviving the corrupted pages. 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, 44.77% for the workload of small random writes, 48.54% for the workload of Windows, and 28.38% for the workload of Linux.

ACKNOWLEDGMENT

This work was supported by the Ministry of Science and Technology, Taiwan, under Grant NSC101-2628-E-011-009-MY3.

REFERENCES

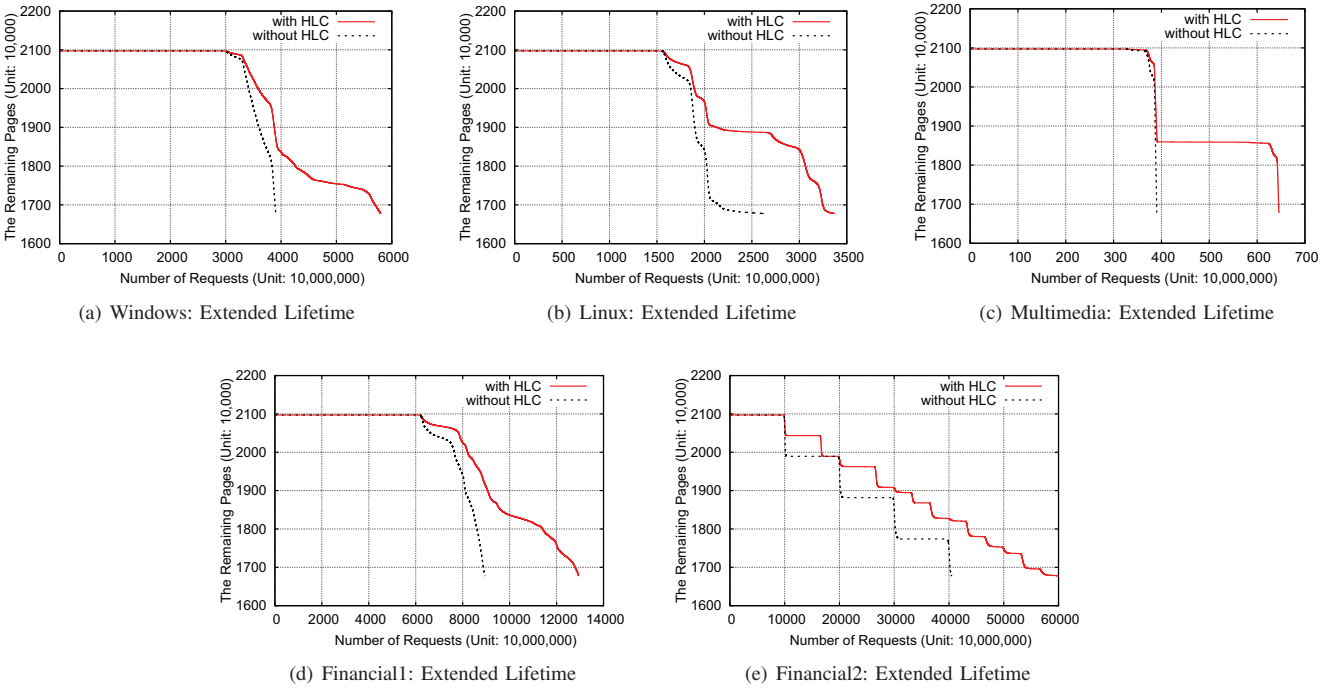
[1] F. Chen, T. Luo, and X. Zhang, "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives," in *FAST*, 2011, pp. 77–90.

[2] Y. Park and J.-S. Kim, "zFTL: power-efficient data compression support for nand flash-based consumer electronics devices," *IEEE Trans. Consumer Electronics*, vol. 57, no. 3, pp. 1148–1156, 2011.

[3] S. Lee, T. Kim, J. Park, and J. Kim, "An integrated approach for managing the lifetime of flash-based ssds," in *DATE*, 2013, pp. 1522–1525.

TABLE II. CHARACTERISTICS OF TRACES [14], [15], [16].

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



(a) Windows: Extended Lifetime

(b) Linux: Extended Lifetime

(c) Multimedia: Extended Lifetime

(d) Financial1: Extended Lifetime

(e) Financial2: Extended Lifetime

Fig. 6. The reducing of over-provision space with HLC management scheme.

TABLE III. THE IMPROVEMENT OF LIFETIME.

	Total Number of Requests		Improvement	
	Without HLC	With HLC	Gain Requests	Extended Lifetime
Windows	39,036,927,579	57,987,115,174	18,950,187,595	48.54%
Linux	26,252,971,996	33,704,928,539	7,451,956,543	28.38%
Multimedia	3,902,115,121	6,456,100,150	2,553,985,029	65.45%
Financial1	89,387,918,066	129,410,764,423	40,022,846,357	44.77%
Financial2	404,694,984,100	597,947,493,114	193,252,509,014	47.75%

[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 *DAC*, 2013, p. 163.

[5] X. Jimenez, D. Novo, and P. Ienne, "Phoenix: reviving MLC blocks as SLC to extend NAND flash devices lifetime," in *DATE*, 2013, pp. 226–229.

[6] Y.-J. Woo and J.-S. Kim, "Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs," in *EMSOFT*, 2013, pp. 1–10.

[7] 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.

[8] Micron, "NAND Flash Memory MT29F4G08AAA, MT29F8G08BAA, MT29F8G08DAA, MT29F16G08FAA," Micron, Tech. Rep., 2006.

[9] ——, "TN-29-25: Improving Performance Using Two-Plane Commands Introduction," Micron, Tech. Rep., 2007.

[10] Samsung, "K9GAG08B0M (2G x 8 Bit NAND Flash Memory) Datasheet," Samsung, Tech. Rep., 2006.

[11] ——, "NAND Flash Memory K9GAG08B0M, K9GAG08U0M, K9LBG08U1M," Samsung, Tech. Rep., 2007.

[12] J.-W. Hsieh, H.-Y. Lin, and D.-L. Yang, "Multi-channel architecture-based ftl for reliable and high-performance ssd," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, p. 1, 2013.

[13] Intel, "Intel MD516 NAND Flash Memory JS29F16G08AAMC1, JS29F32G08CAMC1, JS29F64G08FAMC1," Intel, Tech. Rep., 2007.

[14] L.-P. Chang, "A hybrid approach to nand-flash-based solid-state disks," *IEEE Trans. Computers*, vol. 59, no. 10, pp. 1337–1349, 2010.

[15] 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. Electr. Syst.*, vol. 15, no. 1, 2009.

[16] Umasstracerepository. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage