PWL: A Progressive Wear Leveling to Minimize Data Migration Overheads for NAND Flash Devices

Fu-Hsin Chen[1,5], Ming-Chang Yang[2,3,6], Yuan-Hao Chang[3,7], and Tei-Wei Kuo[1,4,2,3,8]
[1]Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan (R.O.C.)
[2]Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 106, Taiwan (R.O.C.)
[3]Institute of Information Science, Academia Sinica, Taipei 115, Taiwan (R.O.C.)
[4]Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan (R.O.C.)
Email: {[5]r01922009, [8]ktw}@csie.ntu.edu.tw, {[6]mcyang, [7]johnson}@iis.sinica.edu.tw

***Abstract*—As the endurance of flash memory keeps deteriorating, exploiting wear leveling techniques to improve the lifetime/endurance of flash memory has become a critical issue in the design of flash storage devices. In contrast to existing wear-leveling techniques that aggressively distributes the erases to all flash blocks by a fixed threshold, we propose a *progressive wear leveling design* to perform wear leveling in a "progressive" way to prevent any block from being worn out prematurely, and thereby to ultimately minimize the performance overheads caused by the unnecessary data migration. The results reveal that, instead of sacrificing the device lifetime, performing wear leveling in such a progressive way can not only minimize the performance overheads but even have potentials to extend the device lifespan.**

I. Introduction

A flash storage device usually consists of several chips. Each chip is composed of multiple dies, each of which contains multiple planes. A plane is partitioned into blocks, each of which contains a fixed number of pages. Moreover, a flash block is the basic unit for erase operations, and each page of a flash block is the basic unit for read/write operations; meanwhile, a flash page is logically partitioned into two areas: the user area and the spare area. The user area is used for storing user data while the spare area is used to maintain the house-keeping information, such as the error correction code. Due to the asymmetric operation units of read/write and erase operations, a page could not be overwritten unless its residing block is first erased (referred to as *write-once property*). Furthermore, each block can only endure a limited number of erases or it might fail to hold the user data.

In recent years, because of the rapidly decreasing on block endurances of new-coming flash chips, the wear leveling strategies has becoming a critical design and lots of excellent wear leveling strategies have been proposed [1], [3]–[7], [9]. Through maintaining some effective variables/structures, the existing wear leveling methods can effectively capture the unevenness of erase distributions among all flash blocks, and they could be classified into two categories, *dynamic* and *static wear leveling*. The dynamic wear leveling methods, *e.g.*, [1], [3]–[5], [7], greedily allocate free blocks with fewer erase count to prevent over-using some blocks excessively; on the other hand, the static ones, *e.g.*, [6], [9], will furthermore migrate the infrequently-updated data between blocks in order to distribute the erases to all flash blocks. Most of the famous static wear leveling designs, such as the block erase table (BET) in [6] and the Rejuvenator design in [9], perform wear leveling by a fixed threshold during the whole device lifetime. For example, BET tends to perform wear leveling when the block erases are excessively concentrated on some certain flash blocks; while the key idea of the Rejuvenator is to limit the erase count of all flash blocks in a fixed range. In this paper, we also focus on the design of static wear leveling since static wear leveling designs have been proved to be effective in prolonging the endurance of flash memory than the dynamic designs [6].

In contrast to the past static wear leveling strategies that aggressively bound the erase counts of all flash blocks in a certain range, we propose a *progressive wear leveling design* to perform wear leveling in a "progressive" way based on the average erase count of blocks to only prevent any block from being worn out prematurely so as to ultimately minimize the performance degradation caused by the unnecessary data migration. Meanwhile, a *victim block selection strategy*, which is based on (1) the block erase count and (2) the data access patterns, is also proposed to furthermore improve the capability of the proposed design by avoiding performing wear leveling actions over young flash blocks that contain frequently-updated (or hot) data. The results show that the proposed design could outperform Rejuvanator by at least 84.5% in terms of performance overhead, and could even improve the device lifetime, as compared to BET and Rejuvanator.

The rest of paper is organized as follows. The background and research motivation are presented in Section II. Section III presents the proposed progress wear leveling design. Section IV reports the experiment results. Section V is the conclusion.

II. Background and Motivation

As shown in Figure 1, a flash storage device is usually managed by two software layers, *i.e., flash translation layer* (FTL) and *memory technology device layer* (MTD). The FTL usually consists of three components, *e.g., allocator*, *garbage collector*, and *wear leveler*, and is designed to overcome several inherent constraints of flash memory. The MTD provides unified primitive operations (*e.g.*, read, write, and erase) for the FTL, so that the FTL software can access and control various flash-memory chips produced by different flash vendors. In the FTL, the allocator is in charge of the address translation from the logical block addresses (LBAs) used by the above file system to the actual physical page addresses, because several versions of data of an LBA might coexist in different flash pages due to the *out-place update* to tackle the *write-once* nature of flash pages. When there are too many "out-of-date"

978-3-9815370-4-8/DATE15/©2015 EDAA

flash pages, the garbage collector will be triggered to erase some flash blocks so as to reclaim the space occupied by those old versions of data. Moreover, due the limited and rapidly decreasing program/erase cycles (referred to as *PE cycles*) of flash blocks in new coming flash memory chips, the wear leveler takes the responsibility for prolonging the device lifespan by distributing erases to all flash blocks as evenly as possible.

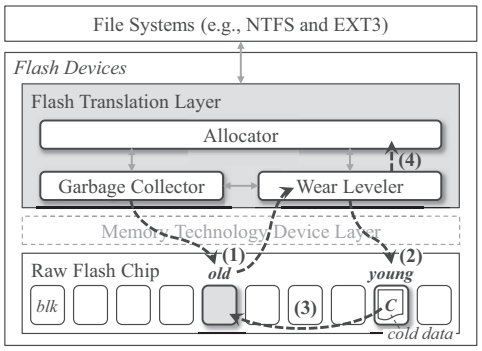


Fig. 1. A typical system architecture of flash storage devices.

In general, as illustrated in Figure 1, the typical wear leveling procedure might involve the following steps. (1) First, when some blocks (*e.g.*, the "old block" shown in Figure 1) are excessively erased by the garbage collector, the wear leveler is invoked. (2) Then, the wear leveler would pick some blocks that have endured less PE cycles so far and mainly contain infrequently-updated data (*i.e.*, *cold data*) as *victim blocks* for the garbage collector to erase. For example, the "young block" depicted in Figure 1 is selected as the victim block since it endures less PE cycles and contains cold data (labeled as "*C*"). (3) After that, the wear leveler would migrate the cold data contained in the "young block" (i.e., the victim block) into the over-erased "old block", so as to prevent the old block from being used/erase continuously; meanwhile, the "young block" would become a free block so that it could share more erasures in the near future. Note that, the concept of selecting flash blocks that contain cold data and endure less PE cycles as victim blocks is also referred to *static wear leveling* in the literature [6], [9], compared to its counterpart *dynamic wear leveling* [1], [3]–[5], [7] which simply selects the blocks with less PE cycles as the victim ones. (4) Finally, due to the urgent needs of address translation for flash memory, the wear leveler should invoke the allocator to update the corresponding address translation information for the migrated data to complete the whole procedure of wear leveling.

Because of the rapidly decreased endurance of MLC flash chips, the design of wear leveling becomes a critical factor in extending the device lifetime. In order to ensure that block erases can be evenly spread among all blocks, many state-of-the-art wear leveling strategies [6], [9] invoke wear leveling procedures by a fixed threshold to bound the erase counts of flash blocks in a certain range during the lifespan of the storage device, where the *erase count* of a flash block represents the number of PE cycles of a block. Among them, *Rejuvanator* is an effective well-known *fixed-threshold-based strategy* that triggers wear leveling by a fixed threshold. Figure 2 shows our experiments to evaluate the characteristics and effectiveness of Rejuvanator. The x axis denotes the amount of written data. The y axis on the left-hand side (with bars) denotes the number of times that the wear leveling is triggered (referred to as *the number of triggered WLs* for short) in every 0.9 TB written data, and that on the right-hand side (with lines) denotes the ratio of the accumulated erase count to the erase count guaranteed by the flash chip. As shown in Figure 2, when the average erase count (plotted with the "Mean" line) of all blocks increases linearly, Rejuvanator can effectively control the erase counts of all blocks in a certain range, *i.e.*, the gap between the lines "Max" and "Min", with a stable number (around 150K) of triggers WLs in every 0.9 TB of written data.

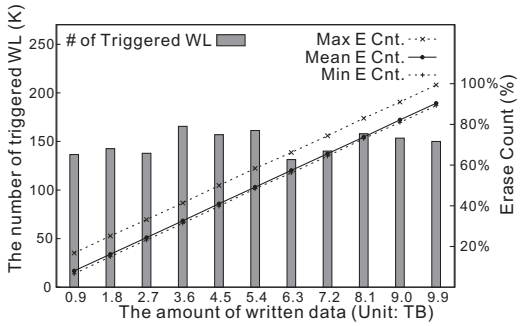


Fig. 2. Motivational results with the well-known JEDEC workload [8].

Nevertheless, even though fixed-threshold-based wear leveling strategies can achieve good evenness of block erases during the device lifespan, these designs might be comparatively too aggressive in the early stage and too conservative in the late stage of the device lifespan. The rationale behind this is that performing wear leveling by a fixed threshold might let some (cold) data be migrated from one block to another old block again and again, rather than be migrated to an old-enough block directly. Especially, please allow us to note that, the wear leveling procedure is a time-consuming process and it might impose a nearly 0.2–0.3 ms latency to the device's response time according to our experiment results. Such observations strongly motivate us to propose a *progressive wear leveling design* to effectively reduce the frequency of triggering the wear leveling procedure, so as to ultimately minimize the total performance overheads caused by wear leveling without sacrificing the device lifetime. Different from existing wear leveling strategies that aggressively bound the erase counts of all flash blocks in a certain range, our design aims to prevent any block from being worn out prematurely in a progressive way.

III. A Progressive Wear Leveling Design

An efficient wear leveling policy should be able to (1) properly prevent old blocks being worn out without sacrificing too much system performance and (2) accurately select appropriate victim blocks to avoid potential counteractions on lifetime prolongation. Thus, in this section, we shall introduce our progressive wear leveling policy. In Section III-A, we first put forward a *progressive threshold* design to minimize the performance overheads introduced by wear leveling procedures. Next, in Section III-B, we will introduce how to accurately identify proper victim block to ultimately improve the effectiveness of wear leveling.

A. Progressive Threshold

To achieve the progressive effects on performing wear leveling, the proposed PWL design adopts a threshold-driven approach on selecting victim block for erases. When the erase count of a flash block exceeds the pre-defined *threshold*, the proposed PWL design would trigger to wear leveling procedure to perform wear leveling on this block. The considered *threshold* is proportionally increased with the average erase count of flash blocks (denoted as $erase_{avg}$). Note that using average erase count, instead of the largest/smallest erase count, of flash blocks to adjust the *threshold* is because the average erase count of blocks would more precisely reflects the access behaviors of the system to most flash blocks. Comparatively, using the largest (resp. smallest) block erase count to adjust the *threshold* would be too aggressive (resp. passive) without considering the general situations of most blocks. The *threshold* can be derived as follows:

$$threshold = erase_{avg} \times \frac{BLK_{endurance} - THR_{init}}{BLK_{endurance}} + THR_{init}$$

where the constant THR_{init} is the smallest value that triggers progressive wear leveling to prevent wear leveling from being triggered too aggressively or too early. Note that THR_{init} should be set appropriately, since a too small THR_{init} value would result in poor performance in the early stages of flash lifespan and a too large THR_{init} value would result in poor device lifetime. In this work, we set THR_{init} to 50% of the PE cycles (denoted as $BLK_{endurance}$) guaranteed by flash chips for both performance and lifetime considerations.

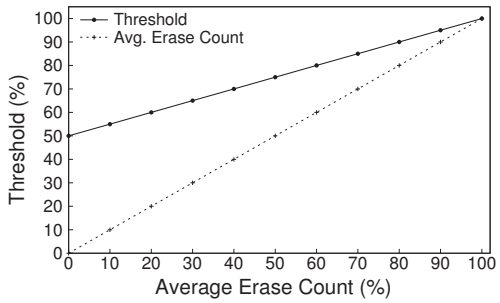


Fig. 3. The relationship between $erase_{avg}$ and $threshold$.

Figure 3 shows the relationship between $erase_{avg}$ and $threshold$, where both $erase_{avg}$ and $threshold$ are normalized with $BLK_{endurance}$. As shown in Figure 3, because the growing rate of $threshold$ is less than that of $erase_{avg}$, the difference between $threshold$ and $erase_{avg}$ would become smaller and smaller when the the average erase count $erase_{avg}$ increases. Consequently, the wear leveling procedure would be activated less frequently (resp. to more frequently) in the early (resp. to late) stages of the device lifetime. In other words, the activation frequency of the wear leveling procedure will increase in a progressive trend.

B. Victim Block Selection

When performing (static) wear leveling, appropriate *victim blocks* should be carefully selected to avoid excessively erasing some "over-erased" blocks. Thus, the accuracy of the victim block selection is a very critical design issue, since an inappropriate selection of victim blocks might contrarily hurt the device lifespan.

TABLE I. OBSERVATION OF DATA AND BLOCKS.

	Young Block	Old Block
Hot Data	Inadequate Selection	Triggering PWL
Cold Data	Appropriate Selection	N/A

Generally speaking, blocks in use can be conceptually classified into four different categories according to the *data hotness* and *block erase counts*, as shown in Table I. Notably, in this analysis, we distinguish user data into *hot data* and *cold data* based on its updated frequency, and categorize the physical blocks into *young block* and *old block* according to its erase count. Furthermore, since migrating hot data into old blocks is harmful to the device lifespan, blocks containing hot data are not suitable to be selected as victim blocks no matter what its erase count is. Meanwhile, blocks with higher erase counts (*i.e.,* old blocks) are also not a wise choice for victim blocks, because the victim blocks will be reclaimed to share more erases. Thus, as an ideal strategy, *young blocks containing cold data would be the most appropriate (and also might be the only) choice for being selected as victim blocks.*

IV. PERFORMANCE EVALUATION

A. Evaluation Metrics and Experiment Setup

In this section, we evaluate the capability of the proposed progressive wear leveling design (referred to as PWL) by trace-driven simulators. In Section IV-B1, we first show the efficacy of the proposed PWL on reducing the performance overheads, especially in the early stages of the device lifespan, in terms of the extra latency, *i.e.,* the extra average response time including the time for (cold) data migration and block-erasing. In Section IV-B2, we evaluate the efficacy of the proposed PWL on prolonging the device lifetime in terms of the first failure time, *i.e.*, the first time that an page error could not be corrected by the error correction code (ECC) stored in the page's spare area.

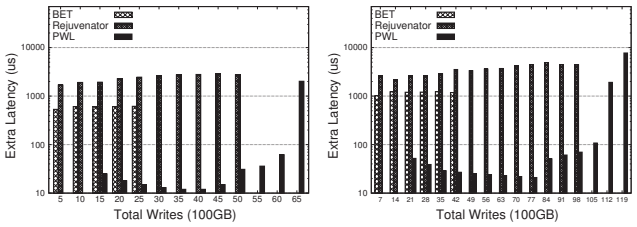
TABLE II. EVALUATED MLC FLASH-MEMORY CHIP CONFIGURATION.

# of chips	4	Read latency	$75\mu s$
Chip size	8GB	Write latency	$1300\mu s$
Block size	1MB (256 pages)	Erase latency	$3800\mu s$
Page size	4KB	P/E cycles	500

As shown in Table II, a 32 GB 4-chipped MLC NAND flash device was under investigation. The size of each MLC chip is 8 GB, which is composed of 8192 blocks. Thus, there are 32768 blocks in total, and each block contains 256 pages. Moreover, we set the THR_{init} value of the proposed PWL as the half of the maximal endurance of flash blocks, and progressively increase the $threshold$ as the $erase_{avg}$ increases. On the other hand, we compared the proposed PWL with two well-known static wear leveling strategies: BET and REJUVENATOR [6], [9]. In particular, we adopted the suggested settings described in [6], [9] to implement the two designs in order to make the two designs effectively and efficiently. Moreover, the three evaluated wear leveling strategies were implemented and applied to the pure page-level FTL scheme [2], which is usually regarded as the ideal FTL management scheme in terms of the average response time.

B. Experimental Results

1) Performance: Figure 4 shows the performance overheads introduced by different wear leveling designs, where the x-axis denotes the accumulated amount of written data (*i.e.,* different stages) and the y-axis denotes the average extra latency of wear leveling procedures in logarithmic scale. Under all traces, both BET and REJUVENATOR respectively introduced a similar performance overheads at all lifetime stages, while the proposed PWL could effectively reduce the extra latency introduced by wear leveling procedures, especially in the early stages of the device lifetime. Moreover, as compared to REJUVENATOR, which performs wear leveling procedures in a relatively aggressive way, the proposed PWL could introduce much less performance overheads under all cases. More specifically, the overall average extra latency of PWL is only about 6.5% and 8.8% of that of REJUVENATOR under the traces of file and source server, respectively. The rationale behind this is that the proposed PWL is effective in avoiding "over-performing" wear leveling procedures in early stages, and thereby can minimize the performance overheads and achieve smaller average latency.



(a) File Server (b) Source Server

Fig. 4. Extra latency introduced by different wear leveling policies (μs).

Notably, in the two sub-figures of Figure 4, some methods would have no corresponding bar chart of extra latency after receiving a certain amount of written data. That is because after such stages of the device lifespan, those methods had already reached their first failure time, and we could not and should not measure the performance overheads of the corresponding method any more. Thus, even though BET had the fewest performance overheads under all traces, as we will also see in the next section, BET had the shortest first failure time.

2) Endurance: Figure 5 shows that, under both traces, the proposed PWL could effectively improve the endurance of a flash device, as compared to the other two fixed-threshold-triggered methods. For example, under the file server and source server traces, the proposed PWL also outperforms the other two methods by at least 36% and 20% respectively in terms of the extended device lifespan. The rationale behind this is that BET tends to perform wear leveling when erasures are not well-distributed rather than when some blocks are over-erased. Thus, BET results in the shortest first failure time under all traces. On the other hand, even though REJUVENATOR effectively bounds the erase count of all flash blocks in a certain range τ, REJUVENATOR might also generate more extra erasures and result in a shorter first failure time, as compared to the proposed PWL. Thus, the experimental results show that performing the wear leveling procedures in a progressive way, like the proposed PWL, might even have great potentials to improve the endurance and to ultimately extend the device lifetime.

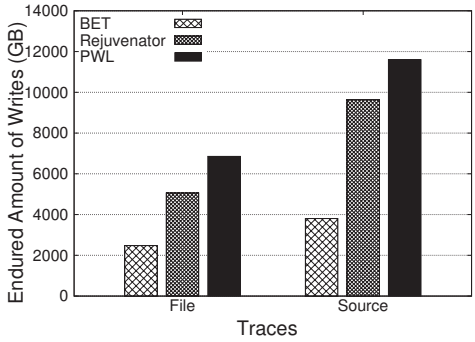


Fig. 5. Endurance achieved by different wear leveling policies.

V. CONCLUSION

This work proposes a progressive wear leveling design to prevent any block from being worn out prematurely so as to ultimately minimize the performance degradation caused by unnecessary data migration and improper victim block selection. Instead of aggressively activating wear leveling by a fixed threshold, the proposed design performs wear leveling according to the average block erase counts with different activation frequencies in different stages of device lifespan. The capability of the proposed design are verified by a series of experiments with respect to the system performance and reliability. The results show that the proposed scheme can not only reduce the performance but also prolong the lifetime of the investigated flash storage devices. In future work, we shall further extend the concept of progressive wear leveling for flash devices with the multi-chip architecture for massive parallelism. Furthermore, we shall also extend the proposed design for 3D flash chips to provide probabilities to overcome the exacerbated disturbance and endurance issues.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science and Technology under Grant Nos. 102-2221-E-002-078-MY2, 103-2221-E-001-012-MY2 and 102-2221-E-001-009-MY3, and by the project with VIA Technologies, Inc..

REFERENCES

[1] Increasing Flash Solid State Disk Reliability. Technical report, Silicon-Systems, Apr 2005.

[2] A. Ban. Flash File System. US Patent 5,404,485. In *M-Systems*, April 1995.

[3] A. Ban. Wear Leveling of Static Areas in Flash Memory. US Patent 6,732,221. *M-systems*, 2004.

[4] L.-P. Chang. On Efficient Wear-Leveling for Large-Scale Flash-Memory Storage Systems. March 2007.

[5] L.-P. Chang and T.-W. Kuo. An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems. In *the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 187–196, 2002.

[6] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo. Improving flash wear-leveling by proactively moving static data. *Computers, IEEE Transactions on*, 59(1):53–65, Jan 2010.

[7] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *ACM Computing Surveys*, 37(2):138–163, June 2005.

[8] JEDEC Solid State Technology Association. *Solid-State Drive (SSD) Endurance Workloads*, 2012.

[9] M. Murugan and David.H.C.Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimized overhead. In *MSST*, 2011.