IEEE TRANSACTIONS ON

COMPUTERS

FSC
Mixed Sources
Product group from well-managed forests, controlled sources and recycled wood or fiber
Cert no. SW-COC-002272
www.fsc.org
© 1996 Forest Stewardship Council

Caring about the environment

Because the IEEE Computer Society is dedicated to environmental responsibility, we have asked our printer to change from a UV-coated cover stock to an aqueous coating. We have also switched to FSC certified paper to futher reduce our impact on the enviroment.

IEEE TRANSACTIONS ON COMPUTERS is published monthly by the IEEE Computer Society. **IEEE Corporate Office:** Three Park Avenue, 17th Floor, New York, NY 10016-5997 USA. Responsibility for the content rests upon the authors and not upon the IEEE or the IEEE Computer Society. **IEEE Computer Society Publications Office:** 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314 USA. **IEEE Computer Society Headquarters:** 1730 Massachusetts Ave. NW, Washington, DC 20036-1992 USA. **Back issues:** IEEE members $20.00, nonmembers $130.00 per copy. (Note: Add $4.00 postage and handling charge to any order from $1.00 to $50.00, including prepaid orders). Complete price information available on request. Periodicals postage paid at New York, NY, and at additional mailing offices. **Postmaster:** Send address changes to IEEE TRANSACTIONS ON COMPUTERS, IEEE, Membership Processing Dept., 445 Hoes Lane, PO Box 1331, Piscataway, NJ 08854-4141 USA. GST Registration No. 125634188. Canada Post Publications Mail Agreement Number 40013885.

Improving Flash Wear-Leveling by Proactively Moving Static Data

Yuan-Hao Chang, *Member*, *IEEE*, Jen-Wei Hsieh, *Member*, *IEEE*, and Tei-Wei Kuo, *Senior Member*, *IEEE*

Abstract—Motivated by the strong demand for flash memory with enhanced reliability, this work attempts to achieve improved flash-memory endurance without substantially increasing overhead and without excessively modifying popular implementation designs such as the Flash Translation Layer protocol (FTL), NAND Flash Translation Layer protocol (NFTL), and Block-Level flash translation layer protocol (BL). A wear-leveling mechanism for moving data that are not updated is proposed to distribute wear-leveling actions over the entire physical address space, so that static or rarely updated data can be proactively moved and memory-space requirements can be minimized. The properties of the mechanism are then explored with various implementation considerations. A series of experiments based on a realistic trace demonstrates the significantly improved endurance of FTL, NFTL, and BL with limited system overhead.

Index Terms—Flash memory, wear leveling, endurance, reliability.

1 Introduction

While flash memory remains one of the most popular storage-system designs for embedded systems, its applications have far surpassed its original design goals. The two popular NAND flash-memory designs are Single Level Cell (SLC) flash memory and Multiple Level Cell (MLC) flash memory. Each SLC flash-memory cell can accommodate one-bit information while each $\mathrm{MLC}_{\times n}$ flash-memory cell can contain n-bit information. As n increases, the endurance of each block in MLC flash memory decreases substantially. In recent years, flash memory has become one part or layer in many system designs. Well-known examples are the flash-memory cache of hard disks proposed by Intel [2], [3], [4] and the Microsoft Windows Vista fast booting service [5], [6]. Such developments reveal the limitations of flash memory, especially in terms of endurance.

The reliability of flash memory is an even more challenging problem now that low-cost flash-memory designs are gaining market momentum [7]. For example, the endurance of an $\mathrm{MLC}_{\times 2}$ flash-memory block is only 10,000 (or 5,000) erase cycles whereas that of its SLC flash-memory counterpart is 100,000 erase cycles [8], [9]. As the number of bits of information per cell would keep increasing for MLC in the near future, the endurance of a block might also get worse, such as few thousand or even hundred erase cycles. This underlines the reliability issue of flash memory. However, improving reliability is problematic because flash-memory designs allow little compromise between system performance and cost, especially for low-cost devices. These observations motivate the current proposal for enhancing flash-memory endurance with minor modifications of popular implementation designs.

A NAND flash-memory chip contains many blocks, and each block consists of a fixed number of pages. A block is the smallest unit involved in erase operations while read and write operations are done in pages. Each page of small-block(/large-block) SLC flash memory can store 512 B(/2 KB) data, and each block contains 32(/64) pages. The configuration of $\mathrm{MLC}_{\times 2}$ flash memory is the same as that of large-block SLC flash memory, except that each block is composed of 128 pages [10]. Notably, recent MLC designs are intended to reduce costs by employing large-capacity pages such as 4 KB pages and 8 KB pages [11].

Flash memory is usually managed by a block-device-emulating layer so that file systems can be built on top (see below discussion regarding Flash Translation Layer protocol (FTL), NAND Flash Translation Layer protocol (NFTL), and Block-Level flash translation layer protocol (BL)). Such a layer implementation can be carried out by either software on a host system (as a raw medium) or hardware/firmware on the flash device. In the past decade, numerous research results and implementation designs have been proposed to satisfy the performance needs of flash-memory storage systems, e.g., [12], [13], [14], [15], [16], [17], [18], [19], [20]. Some authors have proposed different system architectures and layer designs, e.g., [16], [17], [18], and some have exploited large-scaled storage systems and data compression, e.g., [19], [20].

Given the characteristics of flash memory in out-place updates, data to be updated must be written to another page of flash memory, where the original page of the data is marked as invalid. The out-place-update characteristic results in the wear-leveling issue over flash memory because any recycling of invalid pages introduces block

- *Y.-H. Chang and T.-W. Kuo are with the Department of Computer Science and Information Engineering, Graduate Institute of Networking and Multimedia, National Taiwan University, CSIE Building, No. 1, Sec. 4, Roosevelt Rd., Taipei 106, Taiwan. E-mail: {johnson, ktw}@csie.ntu.edu.tw.*
- *J.-W. Hsieh is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, IB-1032, CSIE, No. 43, Sec. 4, Keelung Rd., Taipei 106, Taiwan. E-mail: jenwei@mail.ntust.edu.tw.*

Manuscript received 20 Sept. 2007; revised 1 Mar. 2009; accepted 11 Mar. 2009; published online 10 Sept. 2009.
Recommended for acceptance by A. Gonzalez.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-09-0476.
Digital Object Identifier no. 10.1109/TC.2009.134.

0018-9340/10/$26.00 Published by the IEEE Computer Society

erasing. The objective of wear leveling is to evenly distribute block erases over flash memory so that flash-memory endurance can be improved. Thus, various promising approaches based on dynamic wear leveling have been proposed, e.g., [16], [21], [22], [23], [24]. Dynamic wear leveling achieves wear leveling by recycling blocks of hot (i.e., dynamic or frequently updated) data areas. Such approaches require efficient identification of hot data, and several excellent designs have been proposed, e.g., [16], [25], [26], [27]. Although dynamic wear leveling does substantially enhance wear leveling, the endurance improvement is stringently constrained by its nature: that is, blocks of cold data (i.e., infrequently updated data) are likely to stay intact regardless of whether updates of hot data wear out other blocks. In other words, updates and recycling of blocks/pages will only happen to blocks that are free or occupied by hot data.

Static wear leveling is orthogonal to dynamic wear leveling. Its objective is to proactively move static or infrequently updated data to other locations so as to prevent any static or infrequently updated data from staying at any block for a long period of time. Thus, wear leveling can be evenly applied to all blocks. For example, assume that an extremely large quantity of data must be written or even repeatedly updated to a 2 GB flash memory. If cold data, such as movies, comprise 1.5 GB of the flash memory, then updating those data would be constrained in the reusing of blocks in the one fourth of the flash memory. No dynamic wear leveling can effectively resolve the endurance problem because cold data always stay at their blocks. Only static wear leveling can ensure that blocks are evenly utilized and have even erase counts because it can keep cold data from staying at their blocks.

Although static wear leveling was first defined in early 2000, e.g., [28], there are still limited results reported in the literature [21], [22], [28] due to historic market needs. Particularly, Ban and Hasbaron proposed randomly erasing blocks after a fixed number of erase or write requests [22]. Ban and Hasbaron's algorithm was analyzed both theoretically and experimentally by Ben-Aroya and Toledo and later implemented by Spivak and Toledo in a flash-based storage system [29], [30]. Although Ban and Hasbaron's algorithm has proven being effective, the algorithm has difficulty in identifying hot and cold data. Its inadequate tracking of data-access locality limits its effectiveness since hot data are frequently updated and are not necessary to be moved around by a wear-leveling algorithm. The popularity of MLC flash memory and the widespread adoption of flash memory in various products with high update frequencies have focused much attention on static wear leveling in recent years. For example, some recent flash-memory products have adopted Ban and Hasbaron's algorithm for static wear leveling. The main technical challenge of static wear leveling is on the endurance improvement with respect to extra overheads that are considered reasonable in selected products.

This study proposes a static wear-leveling mechanism for improving flash-memory endurance with limited memory-space requirements. Specifically, an adjustable housekeeping data structure is presented, and a cyclic-queue-based

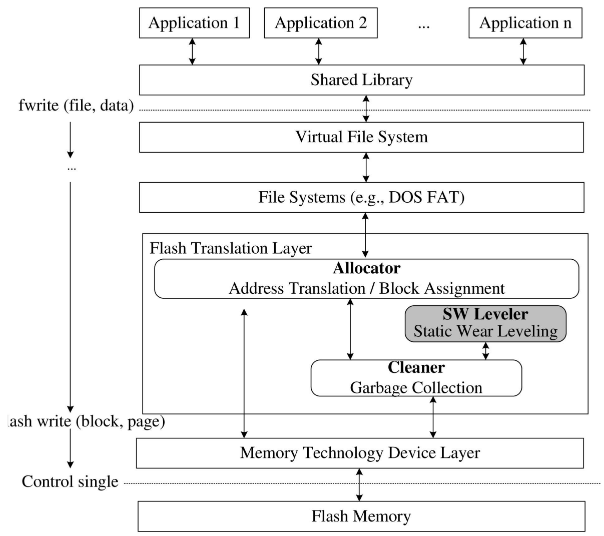

Fig. 1. The typical system architecture.

scanning procedure is proposed for static wear leveling. The objective is to improve the endurance of flash memory with limited overhead and without excessively modifying popular implementation designs, such as FTL, NFTL, and BL. The behavior of the proposed mechanism was analyzed with respect to FTL, NFTL, and BL, on extra block erases, extra live-data copying, address translation time, space utilization, and main-memory requirements. A series of experiments was then conducted based on a realistic trace. The experiments revealed endurance improvements of 103.37, 136.32, and 60.74 percent in FTL, NFTL, and BL-based storage systems, respectively, in terms of the first failure time (i.e., the first time that a worn-out block occurs). Erase-count distribution over blocks was also substantially improved.

The rest of this paper is organized as follows: Section 2 presents the system architectures and summarizes popular existing implementations of Flash Translation Layer for address translation and block assignment. Section 3 presents the research motivation. Section 4 proposes an efficient static wear-leveling mechanism. Section 5 proposes a worst case model to analyze the overhead of the proposed mechanism. Section 6 analyzes the performance of the proposed mechanism in FTL, NFTL, and BL. Section 7 summarizes the experimental results of the endurance enhancement and extra overheads. Finally, Section 8 concludes the study.

2 System Architecture

2.1 A Typical System Architecture

Fig. 1 shows the typical system architecture for flash-memory-based file systems. A Memory Technology Device (MTD) driver provides basic functions such as read, write, and erase over flash memory. The system also needs a Flash Translation Layer driver for address translation and garbage collection, and FTL, NFTL, and BL are its popular implementations. A typical Flash Translation Layer driver consists of an *Allocator* and a *Cleaner*. The Allocator handles

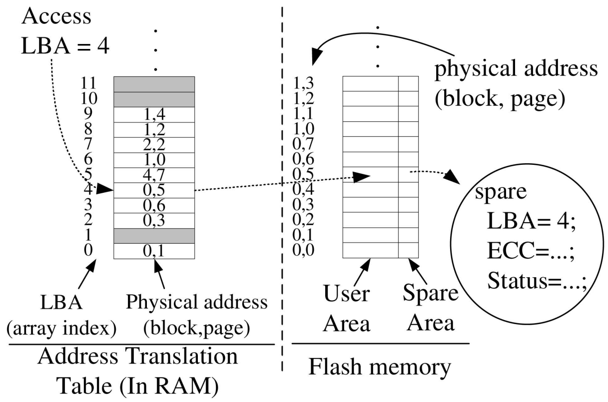

Fig. 2. FTL over small-block flash memory.

any translation of Logical Block Addresses (LBA) and their Physical Block Addresses (PBA). Different approaches have different address translation mechanisms, e.g., FTL, NFTL, and BL [13], [14], [15]. The Cleaner is to do garbage collection to reclaim pages of invalid data. Since garbage collection is done in the unit of a block, valid data in any pages of a block must be copied to other free pages when the block is to be erased.

One important implementation issue for flash-memory management is wear leveling, which evenly distributes the number of erases for each block (due to limitations on the number of erases for blocks). This study proposes the SW Leveler to address this issue (see Section 4).

2.2 Existing Implementations on Flash Translation Layer

2.2.1 FTL

FTL [12], [13], [15] adopts a page-level address translation mechanism for fine-grained address translation. The translation table in Fig. 2 is an example of a fine-grained address translation. The LBA "4" is mapped to the PBA "(0,5)" by the translation table. The LBAs are addresses of sectors mentioned by the operating system. Each sector is 512 B, and each PBA has two parts, i.e., the residing block number and the page number in the block (note that for flash memory with 2 KB pages, the page number of a PBA only indicates the LBA of the first sector stored in this page since each page can store data for sectors of four consecutive LBAs.) When any data of a given LBA is updated, FTL must find a free page to store the data. If the number of free pages is insufficient, garbage collection is triggered to reclaim the space used by invalid pages by erasing their residing blocks. However, before a block is erased, data of any valid pages in the block must be copied to other free pages, and their corresponding translation table entries must be updated.

2.2.2 NFTL

NFTL adopts a block-level address translation mechanism for coarse-grained address translation [14]. An LBA under NFTL is divided into a virtual block address and a block offset. The virtual block address (VBA) is the quotient of the LBA divided by the number of sectors that can be stored in a block, and the block offset is the remainder of the division

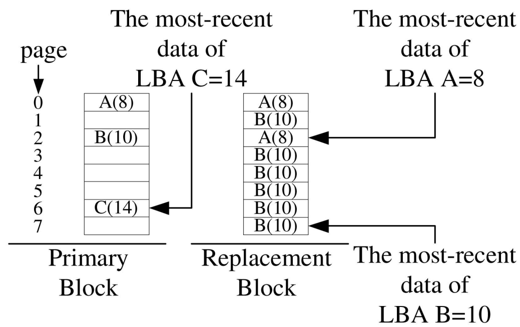

Fig. 3. NFTL over small-block flash memory.

(note that one page may store data for more than one consecutive sector according to the page size of the flash-memory chip). A VBA can be translated to a (primary) physical block address by the block-level address translation. When a write request is issued, the requested data are written to the page with the corresponding block offset in the primary block. Since the following write requests cannot overwrite the same pages in the primary block, a replacement block is needed to perform subsequent write requests, and the data of the (overwritten) write requests are sequentially written to the replacement block. Fig. 3 shows the case in which write requests to three LBAs, $A = 8$, $B = 10$, and $C = 14$, are issued three times, seven times, and one time, respectively, to a primary block and a replacement block. Fig. 3 also shows the most recent data of A, B, and F.

When a replacement block is full, valid pages in the block and its associated primary block are merged into a new primary block (and a replacement block if needed), and the previous two blocks are then erased by the garbage collection in the future. If the number of free blocks after reclaiming invalid blocks is insufficient, then garbage collection generates free blocks by merging the valid pages of primary blocks and their corresponding replacement blocks.

2.2.3 BL

BL is used in many low-end flash-memory products such as USB flash devices [31]. Its design is similar to that of NFTL but with a simplified block-level address translation mechanism. Like NFTL, each LBA is also divided into a VBA and a block offset, and a mapping table is adopted for VBAs and their PBAs. For each write operation, a free block is allocated to save the data of the remaining valid pages of the original mapped block and the new data of the write operation.

This study adopts an efficient implementation of BL that delays the copying of valid pages from the original mapped block to reduce the overheads of live-page copyings and free block allocations. The following example illustrates the concept: Fig. 4 shows a case in which data in some blocks are valid. Two writes are first issued to update data of LBAs 10-12 and 13-14. In contrast with the original BL implementation, only one free block is allocated to save the data of the valid pages of the original mapped block and the new data of the first and second write operations since LBAs 10-12 and LBAs 13-14 both belong to the same VBA. However, the data of the third write operation with LBA 20 are stored in another allocated free block because the VBA of LBA 20 differs from

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.