An Endurance-Enhanced Flash Translation Layer via Reuse for NAND Flash Memory Storage Systems

Yi Wang, Duo Liu, Zhiwei Qin, Zili Shao
Department of Computing
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
{csywang, csdliu, cszqin, cszlshao}@comp.polyu.edu.hk

***Abstract*—NAND flash memory is widely used in embedded systems due to its non-volatility, shock resistance and high cell density. In recent years, various Flash Translation Layer (FTL) schemes (especially hybrid-level FTL schemes) have been proposed. Although these FTL schemes provide good solutions in terms of endurance and wear-leveling, none of them have considered to reuse free pages in both data blocks and log blocks during a merge operation. By reusing these free pages, less free blocks are needed and the endurance of NAND flash memory is enhanced. We evaluate our reuse strategy using a variety of application specific I/O traces from Windows systems. Experimental results show that the proposed scheme can effectively reduce the erase counts and enhance the endurance of flash memory.**

I. Introduction

NAND flash memory is widely used in embedded systems as a non-volatile storage device. Different from hard disk drives, NAND flash memory has several distinct characteristics that impose challenges for its management. First, NAND flash memory suffers from "out-of-place update". One page in NAND flash memory once be written cannot be updated (re-written) until it is erased, and the erase operation is always triggered by the time-consuming process called garbage collection. Second, NAND flash blocks have a limited erase lifetime. A block will be broken if its erase counts reaches the limit. Third, for some NAND flash management schemes, not all blocks in NAND flash get the same erase counts, so the lifetime of specific blocks may become shorter than other blocks. This would affect the usefulness of the entire flash memory. To address these problems, it is very important to guarantee that the lifetime of NAND flash (known as endurance) is prolonged, and write operations are evenly distributed across all blocks (known as wear-leveling).

In order to solve these constraints, an intermediate software module called Flash Translation Layer (FTL) is designed to emulate the flash memory as a block device so that it can provide transparent storage service to file system users. According to the granularity of mapping unit, there are three types of FTL schemes: page-level mapping, block-level mapping, and hybrid-level mapping. Among them, hybrid-level FTL schemes take advantage of both page-level mapping and block-level mapping and provide good address translation time with limited memory usage.

In hybrid-level FTL schemes, physical blocks are logically partitioned into data blocks (primary blocks) and log blocks (replacement blocks). Data block is used to store the first written data, while the updated data is stored in log blocks. Log-based hybrid-level FTL schemes utilize limited number of blocks as log blocks, which can provide relatively good system utilization, system response time and endurance [1], [2], [3], [4]. In this paper, we focus on log-based FTL designs and further improve their endurance and wear-leveling.

Although log-based FTL schemes have some advantages, there are several open issues that should be considered when adopting log-based FTL schemes. In a merge operation in log-based FTL schemes, valid pages scattered on a data block and its corresponding log blocks are copied into a free block. Both these data blocks and log blocks are known as *dirty blocks*, and erase operation is triggered to reclaim the dirty data block and log blocks. By applying such merge operations, free pages within dirty blocks are wasted, and unnecessary erase operations are needed, which can also degrade the endurance of the entire NAND flash memory.

In recent years, many studies for FTL schemes (especially hybrid-level FTL schemes) have been proposed [5], [6], [7], [8], [9], [10], [11]. Although the above FTL schemes provide good solutions in terms of endurance, wear-leveling, memory usage, and response time, none of them have considered to reuse free pages in both data blocks and log blocks in a merge operation. By applying our reuse strategy, these techniques can further improve the write performance and enhance the endurance.

In this paper, we propose a novel NAND flash translation layer (FTL) scheme to enhance the endurance of log-based NAND flash memory FTL schemes through reuse strategy. Our basic idea is to prevent a dirty block (a data block or a log block) with many free pages from being erased in a merge operation. The dirty block with many free pages will be preserved and further reused as a log block. By reusing free pages in dirty blocks, less free blocks are needed and the life time of NAND flash memory is prolonged. To the best of our knowledge, this is the first work that applies the reuse strategy to reuse both data blocks and log blocks in FTL design.

We conduct experiments using a set of traces collected from real workloads by DiskMon. We applied our reuse strategy to a representative log-based FTL design KAST [4], and compared with KAST in terms of erase counts and endurance with various configurations. Experimental results show that the

978-3-9810801-7-9/DATE11/©2011 EDAA

proposed scheme can effectively reduce the erase counts and enhance the endurance of flash memory as compared with the representative FTL scheme proposed in the previous work.

The reminder of this paper is organized as follows. Section II shows the background and motivation. Section III presents our proposed NAND flash translation layer scheme. Section IV presents the experimental results. Section V concludes the paper and discusses future work.

II. Background and Motivation

A. System Architecture for NAND Flash Memory

A typical NAND flash storage system normally consists of two layers, Flash Translation Layer (FTL) and Memory Technology Device (MTD) layer, as shown in Figure 1. MTD layer implements primitive functions over flash memory, such as read, write, and erase operations. FTL emulates the flash memory as a block device. The main role of FTL is to redirect logical addresses from file system into physical addresses in NAND flash memory, and to maintain the mapping table. FTL also provides useful components, such as garbage collector and wear-leveler, to optimize the space utilization and maintain the same level of wear for each block in NAND flash memory.

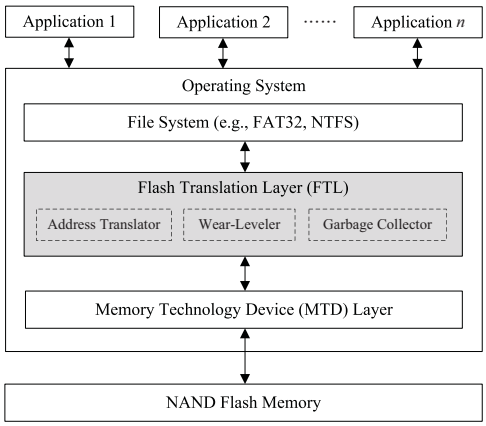


Fig. 1: System architecture for NAND flash memory.

B. Motivational Example

In this section, we briefly revisit the implementation of a well-known hybrid-level FTL scheme KAST [4] and present the motivation of this paper. In hybrid-level FTL schemes, especially log-based FTL schemes, a logical page number (LPN) is divided by the number of pages per block to obtain the logical block number (LBN) and the block offset, where the LBN is the quotient, and the block offset is the remainder of the division. For block-level mapping, a block-level mapping table maps one LBN to one physical block number (PBN), and this physical block is known as data block. Log blocks are used to store updated data, and log blocks use the page-level mapping scheme. The maximum number of log blocks is used to limit the size of page-level mapping table. Therefore, log-based FTL schemes only keep a limited number of blocks as log blocks. In KAST, each log block is enforced to be associated with only K number of data blocks to guarantee the worst-case log block merge time.

The space utilization of KAST can be further improved by adopting our reuse strategy. A motivational example is shown in Figure 2. In the example, each block has 8 pages, and the number of log blocks is 4. The maximum associativity of each log block is 4. The access sequence of write (wr) operations are listed in Figure 2(a). Based on the given access sequence, physical blocks are allocated to store the contents.

When a write operation is performed, the content of the write operation is first written to the page with the corresponding block offset in a data block. In Figure 2(a), the first request is written to LPN #16. A new data block (physical block number #0) is allocated, and the content A is stored in the first page (the reminder of 16/8) of PBN #0. Likewise, physical blocks PBN #1 to PBN #10 are allocated for the second to the twentieth requests. For the 21st request, the content of LPN #80 is updated. In log-based schemes, two kinds of log blocks are used to store the updated content. Among them, *sequential write log block* is used to store the sequential updated data, while *random write log block* is to store the updated data for random write operations. Since the offset for LPN #80 is 0, a sequential write log block is assigned to store the updated content ($I1$). Similarly, for the 22nd and the 23rd requests, a sequential write log block (PBN #12) is allocated to store the updated data for data block PBN #0. The 24th request updates the third page (page number #2) of data block PBN #1. As the page number is not equal to zero, a random write log block (PBN #13) is chose to store the content ($F1$).

The distinct feature of KAST is that it restricts the maximum number of data blocks that can be associated with a log block. For the 25th request, the updated content should be written to a random write block. At this time, only three blocks are allocated as log blocks. PBN #14 is allocated as a random write log block, such that the associativity of log block PBN #13 does not increase. Similarly, for the 26th to 32nd requests, the updated contents always try to find the random write log block with the least associativity. The 33rd request updates the content in data block PBN #10. At this stage, random write log blocks PBN #13 and PBN #14 are all associated with 4 data blocks, which is equal to the maximum associativity of each log block. The 33rd request cannot be written to PBN #13 or PBN #14. As a result, one random write log block has to be selected as a victim log block, and a merge operation is issued. As shown in Figure 2(b), random write log block PBN #13 is picked up as a victim log block. During the merge operation, valid pages in victim log block and its associated data blocks are copied into new data blocks while dirty blocks (the victim log block and the associated data blocks) are erased. Since log block PBN #13 is associated with 4 data blocks (PBN #1, PBN #2, PBN #4, and PBN #8), four new data blocks (PBN #15 to PBN #18) are allocated to copy the valid pages in both log block (PBN #13) and its corresponding data block. After that, both victim log block (PBN #13) and 4 data blocks (PBN #1, PBN #2, PBN #4, and PBN #8) are erased for further usage.

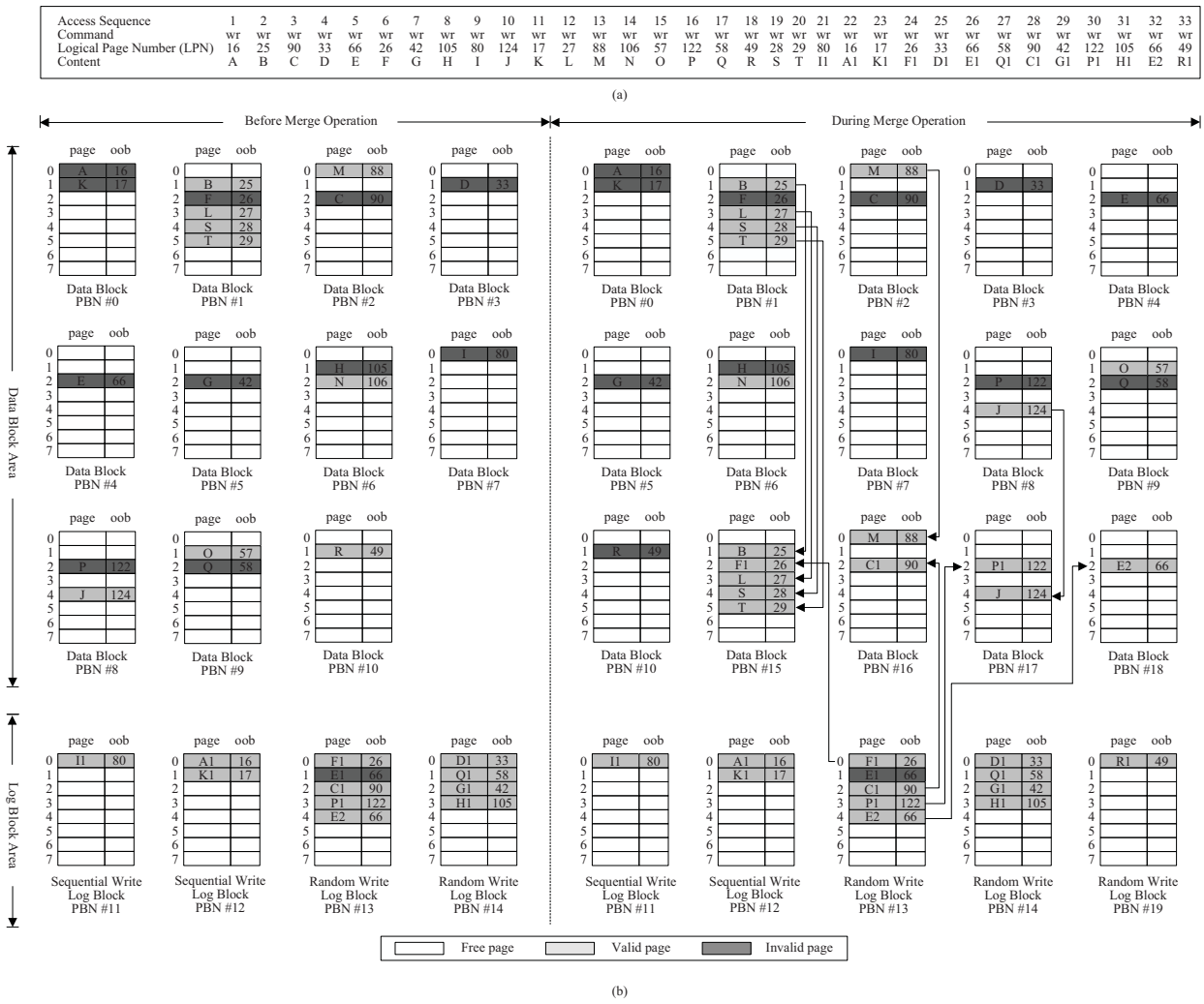


Fig. 2: Motivational example.

It is noticed that the space utilization of KAST can be further improved. When a merge operation is triggered, only one page (page number #2) in PBN #4 is used before it is got erased. Other dirty blocks (PBN #1, PBN #2, and PBN #8) and the victim log block (PBN #13) also consist of many unused free pages.

C. Motivation

In hybrid-level log-based FTL schemes, both victim log block and its associated data block(s) are erased during a merge operation. However, there may exist many free pages in the dirty blocks (victim log block and the associated data blocks). These unused free pages are wasted and can be reused to store data. Furthermore, the erasure of dirty blocks in a merge operation may also degrade the endurance of NAND flash. If these blocks with many unused free pages can be reused as random write log blocks, less free blocks are needed and the life time of NAND flash memory is prolonged. These observations motivate us to propose a novel NAND flash translation layer using reuse strategy to effectively improve the performance (especially space utilization and endurance) of NAND flash memory.

III. EE-NFTL: Endurance-Enhanced NAND Flash Translation Layer

In this section, we introduce our endurance-enhanced NAND flash translation layer (EE-NFTL) scheme to effectively improve the endurance and space utilization of NAND flash memory.

A. EE-NFTL through the reuse strategy

The basic idea of EE-NFTL is to preserve a dirty block (a data block or a log block) with many free pages from being erased if there are many free pages in this dirty block. We employ a reuse strategy to further utilize all free pages inside a preserved dirty block. To achieve this, we put the preserved dirty blocks into a *reuse block list*, in which each dirty block will be further reused as a random write log block. Unused free pages in a dirty block are fully utilized. Then, the erase counts of a physical block is reduced and the endurance can be enhanced.

In EE-NFTL, the reuse strategy is performed based on the number of free pages in a dirty block. If there are many free pages in a dirty block, this dirty block should not be erased in a merge operation. On the other hand, if there exists only a limited number of free pages in a dirty block, this dirty block can be erased like that in the conventional scheme. This is because only the limited number (e.g, 1 or 2) of free pages may not be valuable for reusing. Therefore, we compare the actual free page ratio (the number of free pages in a block divided by the number of pages per block) with a predefined threshold to decide whether or not to reuse a dirty block in a merge operation. The free page ratio of a dirty block is denoted by $\mathcal{R}$, and the threshold is denoted by $\mathcal{T}$. In our reuse strategy, if $\mathcal{R} \geq \mathcal{T}$, which means there are many free pages in a dirty block. In this case, this dirty block is preserved and reused as a random write log block. Otherwise, this dirty block will be erased in a merge operation.

In our scheme, the preserved dirty block is treated as a reused block and put into a reuse block list. The reused blocks in the reuse block list are sorted in descending order by the free page ratio $\mathcal{R}$. Since some advanced hybrid-level FTL schemes keep a limited number of log blocks to store the updated data, it is necessary to restrict the length of the reuse block list. In our reuse strategy, if a preserved dirty block will be put into the reuse block list, it will first check whether or not the reuse block list is full. If the list is not full, this reused block will be inserted into the reused block list. On the other hand, if the reuse block list is full, the reused block with the lowest free page ratio $\mathcal{R}$ will be evicted from the list for erase operation.

If a dirty block is preserved for reusing, the valid pages in this dirty block will be marked as invalid. When a new random write log block is needed to store the updated data, the reused block in the reuse block list with the maximum free page ratio will be selected as a random write log block. If the reuse block list is empty, a new block from the free block list will become the random write log block.

An example of EE-NFTL is shown in Figure 3. In this example, the threshold $\mathcal{T}$ is set as 50%. Then a dirty block (a data block or a log block) will be reused if its free page ratio $\mathcal{R}$ is greater than or equal to 50%. Otherwise, this dirty block will be erased in a merge operation. During the merge operation, data blocks (PBN #1, PBN #2, PBN #4, and PBN #8) and the victim log block (PBN #13) trigger erase operation. Since the free page ratio $\mathcal{R}$ of data block PBN #1 and that of the log block PBN #13 are less than 50% (3 free pages out of 8 pages in one block), both PBN #1 and PBN #13 will be erased. Then these two blocks are put into the free block list. Since the free page ratio $\mathcal{R}$ of data blocks PBN #2, PBN #4, and PBN #8 are all greater than 50%, these three data blocks will be reused and put into the reuse block list in the descending order. After adopting the reuse strategy, free pages in dirty blocks are fully utilized. Consequently, the erase counts of each block can be further reduced, and the endurance and space utilization of all blocks in NAND flash memory can be enhanced.

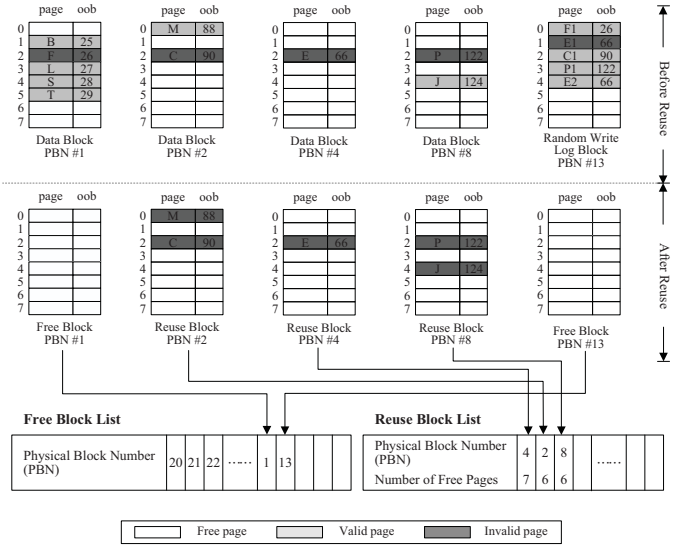


Fig. 3: The reuse strategy of EE-NFTL.

B. The Analysis of EE-NFTL

This section presents the analysis of EE-NFTL. We will analyze the performance improvement of EE-NFTL over representative FTL scheme for two extreme cases. We will also discuss how the length of the reuse block list influences the performance of our scheme.

In this analysis, N_{page} denotes the number of pages in a block; N_{wr} denotes the number of write requests to NAND flash memory; $N_{log-blk}$ denotes the maximum number of log blocks in a NAND flash memory; K denotes the maximum associativity of a log block.

Random update requests and sequential update requests form two special cases for the write requests of NAND flash memory. In this analysis, random update requests represent that, N_{wr} write requests are mapped to $N_{wr}/2$ data blocks with non-zero page offset and each page is updated for exactly once. Sequential update requests represent that, N_{wr} write requests are mapped to N_{wr}/N_{page} data blocks with page offset zero and each page is updated for $N_{page}/2$ times. Random update requests mainly use random write log blocks to store the updated data, while sequential update requests use sequential write log blocks. In real applications, all write requests are the mixture of random update requests and sequential update requests. Therefore, we analyze the performance of our EE-NFTL and representative FTL scheme KAST [4] under these two extreme cases.

For random update requests, after updating $K \times N_{log-blk}$ pages in $K \times N_{log-blk}$ different data blocks, all $N_{log-blk}$ blocks are random write log blocks, and the associativity of each random write log block is K. Given another update operation, one of the random write log block will be selected to trigger merge operation. For each merge operation, if the free page threshold $\mathcal{T}$ is less than or equal to $1/N_{page}$, K data blocks will be put into reuse block list. Therefore, after each merge operation, the block at the head of the reuse block

list will have $N_{page} - 1$ free pages. For this special case, to keep only one block in reuse block list is enough to handle each update request. Longer reuse block list will increase the memory space to store this list, and it will also increase the time complexity for comparing the free page ratio of each new coming reused block with that of the blocks in the reuse block list.

For sequential update requests, a sequential write log block is allocated to store the first updated data. The subsequent $N_{page}/2 - 1$ update operations will be written to the same sequential write log block, and this sequential write log block will be transformed into a random write log block. The merge operation is triggered when all log blocks are became random write log blocks and a sequential write request looks up for a new sequential write log block. In this case, a random write log block with $N_{page}/2$ free pages are selected as a victim log block. If the free page threshold $\mathcal{T}$ is no greater than $1/2$, both random write log block and its associated data block will be put into the reuse block list. In order to guarantee the benefit of our reuse strategy, the reuse block list should contain at least $N_{log-blk}$ blocks.

IV. Performance Evaluation

In this section, we present the experimental results with analysis. We compare and evaluate our proposed EE-NFTL scheme over the representative hybrid-level FTL scheme KAST [4] in terms of two performance metrics: the number of block erase counts, and the endurance of NAND flash memory.

The performance evaluation is through a trace-driven simulation. The trace of data request was collected from desktop running DiskMon with an Intel Pentium Dual Core 2GHz processor, a 200GB hard disk, and a 2GB DRAM. The trace data reflects the real workload of the system in accessing the hard disk for daily use. Table I shows the characteristics of traces.

TABLE I: Characteristics of Traces.

Trace	# of write operation	# of read operation	% of write	% of read
chatOnline	17,890,720	15,133,024	54.18%	45.82%
copyFile	268,671,488	113,664	99.96%	0.04%
office	123,364,096	14,356,928	89.58%	10.42%
p2p	687,524,064	548,744,032	55.61%	44.39%

In our experiments, a 4GB NAND flash memory is configured. The page size, number of pages in a block, and size of the OOB of each page are set as 2KBytes, 64, 32Bytes, respectively. The predefined threshold $\mathcal{T}$, the associativity K, the maximum number of log blocks, and the length of the reuse block list are set as 20%, 4, 128, 128, respectively.

To evaluate the effectiveness of the proposed scheme, we present the experimental results in terms of the block erase counts and endurance. Table II shows the experimental results for the number of block erase counts of our proposed scheme EE-NFTL and a representative hybrid-level FTL scheme KAST. In Table II, column 2-5 present the erase counts for data blocks and log blocks; columns 6-8 present the total number of erase counts for all blocks. From the results, our proposed EE-NFTL can achieve an average reduction of 47.93% among four test traces, and a maximum reduction of 75.27% (Trace p2p) in the total number of erase counts. We can also observe that our EE-NFTL may slightly increase the number of erase counts for log blocks (Trace office). That is because, data blocks with many free pages are reused as random write log blocks and these blocks are treated as log blocks when triggering erase operations. This observation is also proved by the experimental results for the erase counts of data blocks. As shown in columns 2-3, our EE-NFTL scheme can significantly reduce the number of erase counts for data blocks as compared with that of KAST.

TABLE III: The maximum number of block erase counts of EE-NFTL versus KAST [4].

Trace	KAST	EE-NFTL	Imp(%)
chatOnline	82	62	24.39
copyFile	1,302	1,049	19.43
office	823	554	32.69
p2p	9,380	6,393	31.84
Average			27.09

The endurance of NAND flash memory is mainly affected by the worst case erase counts of a physical block in the flash memory. Table III illustrate the improvement of our proposed EE-NFTL scheme over KAST in terms of the maximum number of erase count among all physical blocks. The experimental results show that, our approach can reduce 27.09% for the maximum erase counts, which represents that our approach can extend over 1/4 times longer life time for NAND flash memory.

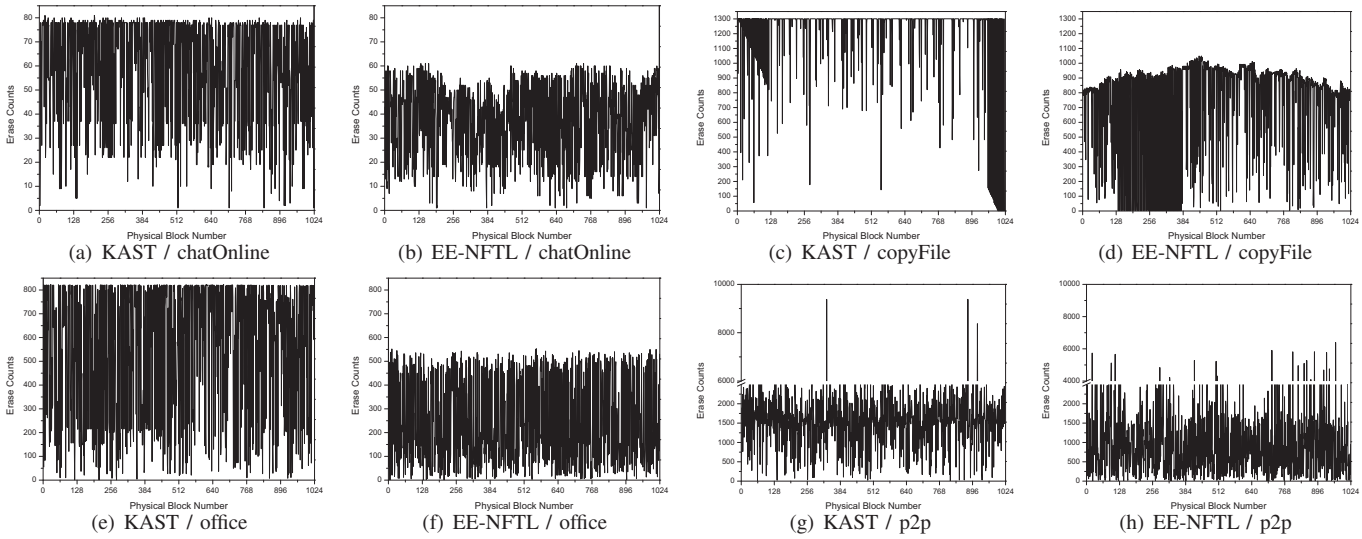
As the distribution of erase counts among physical blocks will directly influence the endurance of the NAND flash memory. For demonstration purpose, we picked up 1024 physical blocks (about 128MB) of 4GB memory to illustrate the distribution of the number of block erase counts for each block. As shown in Figure 4, our scheme can significantly reduce the erase counts for all four traces. From the experimental results, we can see that, our EE-NFTL scheme can not only reduce the total erase counts of NAND flash memory, but also enhance the lifetime of NAND flash memory by improving its maximum number of erase counts.

V. Conclusion

In this paper, we proposed an endurance-enhanced FTL scheme, that outperforms the representative log-based FTL scheme KAST. The performance improvement is achieved by reusing both data blocks and log blocks in a merge operation. By doing this, less free blocks are needed and the endurance of NAND flash memory is enhanced. We conduct experiments on a set of application specific traces, and the experimental results show that our scheme can significantly reduce the erase counts of blocks and improve the endurance.

TABLE II: The number of block erase counts of EE-NFTL versus KAST [4].

	data block		log block		total # of erase counts		
Trace	KAST	EE-NFTL	KAST	EE-NFTL	KAST	EE-NFTL	Imp(%)
chatOnline	959,681	255,232	964,927	962,655	1,924,608	1,217,887	36.72
copyFile	19,872,640	3,949,248	19,875,776	19,869,152	39,748,416	23,818,400	40.08
office	8,888,128	1,807,136	8,900,929	8,929,920	17,789,057	10,737,056	39.64
p2p	48,122,043	3,949,248	48,180,288	19,869,153	96,302,331	23,818,401	75.27
Average							47.93



(a) KAST / chatOnline (b) EE-NFTL / chatOnline (c) KAST / copyFile (d) EE-NFTL / copyFile

(e) KAST / office (f) EE-NFTL / office (g) KAST / p2p (h) EE-NFTL / p2p

Fig. 4: The endurance enhancement of EE-NFTL over KAST for traces *chatOnline, copyFile, office,* and *p2p*.

In the future, we plan to extend our reuse strategy to large scale flash memory and Solid State Disk (SSD) to improve its endurance and wear-leveling. Recently, some non-volatile memory techniques, such as Phase Change Memory (PCM) [12], are proposed to become promising candidates for the next generation memory to replace DRAM. How to apply reuse strategy to PCM to reduce write activities is also a topic for us to explore.

ACKNOWLEDGMENT

The work described in this paper is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (GRF PolyU 5260/07E and GRF PolyU 5269/08E) and HK PolyU 1-ZV5S.

REFERENCES

[1] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366–375, May 2002.

[2] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, p. 18, 2007.

[3] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 38:1–38:23, August 2008.

[4] H. Cho, D. Shin, and Y. I. Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems," in *DATE '09: Proceedings of the conference on Design, automation and test in Europe*, 2009.

[5] C.-H. Wu and T.-W. Kuo, "An adaptive two-level management for the flash translation layer in embedded systems," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 601–606.

[6] Y.-H. Chang and T.-W. Kuo, "A commitment-based management strategy for the performance and reliability enhancement of flash-memory storage systems," in *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 858–863.

[7] S. Choudhuri and T. Givargis, "Performance improvement of block based NAND flash translation layer," in *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2007, pp. 257–262.

[8] Y. Joo, Y. Choi, C. park, S. W. Chung, E.-Y. Chung, and N. Chang, "Demand paging for OneNAND flash eXecute-In-Place," in *CODES+ISSS '06: Proceedings of the 4th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, 2006, pp. 229–234.

[9] K. Lee and A. Orailoglu, "Application specific low latency instruction cache for NAND flash memory based embedded systems," in *SASP '08: Proceedings of the 2008 Symposium on Application Specific Processors*, June 2008, pp. 69–74.

[10] Y. Wang, D. Liu, M. Wang, Z. Qin, Z. Shao, and Y. Guan, "RNFTL: a reuse-aware NAND flash translation layer for flash memory," in *LCTES '10: Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*, 2010, pp. 163–172.

[11] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "Demand-based block-level address mapping in large-scale NAND flash storage systems," in *CODES/ISSS '10: Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 173–182.

[12] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 14–23.