Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2013-IC-001

2013-IC-001

Optimizing Translation Information Management in NAND Flash Memory Storage Systems

Qi Zhang, Xuandong Li, Linzhang Wang, Tian Zhang, Yi Wang, Zili Shao

Proceedings of the 18th Asia and South Pacific Design Automation Conference 2013

Optimizing Translation Information Management in NAND Flash Memory Storage Systems

Qi Zhang[†‡], Xuandong Li[†‡], Linzhang Wang[†‡], Tian Zhang[†‡], Yi Wang[§], Zili Shao[§]
[†]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093
[‡]Department of Computer Science and Technology, Nanjing University, Nanjing 210093
[§] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
zhangqi@seg.nju.edu.cn {lxd, lzwang, ztluck}@nju.edu.cn, {csywang, cszlshao}@comp.polyu.edu.hk

Abstract— Address mapping is one of the major functions in managing NAND flash. With the capacity increase of NAND flash, it becomes vitally important to reduce the RAM print of the address mapping table while not introducing big performance overhead. Demand-based address mapping is an effective approach to solve this problem, in which the address mapping table is stored in NAND flash (called translation pages), and mapping items are cached on-demand in RAM. Therefore, it is critical to manage translation pages in demand-based address mapping. This paper solves two most important problems in translation page management. First, to reduce frequent translation page updates caused by data requests, we propose a page-level caching mechanism to exploit the fundamental property of NAND flash where the basic read/write unit is one page. Second, to reduce the garbage collection overhead from translation pages, we propose a multiple write pointers strategy to group data pages corresponding to the same translation page into one data block, by which, when the data block is reclaimed via the garbage collection, we only need to update one translation page. We evaluate our scheme using a set of benchmarks from both real-world and synthetic traces. Experimental results show that our techniques can achieve significant reduction in the extra translation operations and improve the system response time.

I. Introduction

NAND flash memory has been widely adopted in various storage systems from USB drives, smart phones, digital memory cards, to SSDs (Solid State Drives) due to its non-volatility, low power consumption, high density and good shock resistance. However, NAND flash memory also has some constraints such as "out-of-place updates", and limited lifetime of physical blocks. To conceal these unfavorable characteristics and make NAND flash work like an ordinary block device, an intermediate software module called flash translation layer (FTL) is employed to serve I/O requests and manage NAND flash correspondingly [13]. Address translation is one of main functions in FTL, and it serves to translate a logical address from file system to a physical address in the flash memory. Thus, managing address translation becomes an important task in managing NAND flash.

In FTLs, there are mainly three kinds of address mapping schemes: block-level, page-level, and hybrid-level mapping. The granularity of page-level mapping schemes is one data page so FTL can directly locate the data. However, it requires large RAM space to maintain the page-level mapping table. In block-level mapping schemes [5, 8, 9, 12], a block-level mapping table is maintained so a logical block can be mapped to a physical block and much less mapping information needs to be stored in RAM. Block-level schemes use the block offset to locate the pages within a block so that it may trigger large number of garbage collections due to locating collision. In hybrid-level schemes [6, 7, 16, 19], physical blocks are logically partitioned into data blocks and log blocks. A block-level mapping table is maintained so a logical block can be mapped to a data block. Log blocks are a few physical blocks to hold all page updates for all data blocks and the update data is located through page-level mapping table. However, the drawbacks of block-level and page-level mapping scheme are still not solved completely in the hybrid-level mapping scheme.

Compared to the block-level and hybrid-level mapping, page-level mapping schemes are more efficient in address translation. However, due to its large mapping table in the RAM, pure page-level mapping schemes are hard to apply for embedded systems. In recent years, demand-based approaches [11, 15, 14] are proposed to significantly reduce the RAM footprint. In DFTL [11], only a small number of mapping items are cached in the RAM, while the entire page-level mapping table is stored in translation pages in NAND flash. The performance of DFTL is highly influenced by the characteristics of the workloads. In order to improve the cache hit ratio, Qin et al. proposed a two-level cached demand-based scheme, which makes use of the spatial locality of the workloads so that significantly improves the cache hit ratio and performance [15]. However, only the cached mapping mechanism is considered in the scheme, it does not optimize the management of translation information existing in both cache and NAND flash memory. There are also some techniques [17, 18] proposed to improve the write buffer cache management, but they can not be directly applied into the translation caching.

In this paper, we propose a scheme called *TPM* (Translation Page Management) to optimize the translation information management for demand-based page-level address mappings in NAND flash memory storage systems. In TPM, we first address the problem of reducing frequent translation page updates caused by data page updates. By exploiting the fundamental property of NAND flash in which the basic read/write unit is one page, we change the caching granularity to be one translation page, and achieve faster address translation by directly connecting cache with the global address indexing. We found

978-1-4673-3030-5/13/$31.00 ©2013 IEEE

this mechanism can better capture translation page updates in RAM for both sequential and random writes. Furthermore, in order to reduce the number of translation page copies during the garbage collection, TPM allocates write requests that share the same translation page into one data block. By doing this, the translation page copy operations will be significantly reduced, and the average response time will be effectively improved.

We evaluate TPM using a set of benchmarks from both real-world and synthetic traces. FlashSim [1] is used in our experiments to simulate a 32GB NAND flash memory storage system. The experimental results show that TPM can achieve an average 90.93% reduction in the number of translation page operations. Moreover, our mechanism achieves a 22.14% improvement in the average system response time and a 26.51% reduction in the block erase counts compared with the previous work.

This paper makes the following contributions:

- We present for the first time an optimized translation information management scheme to solve two important problems in demand-based address mapping.
- We demonstrate the effectiveness of this scheme by applying this into a representative demand-based page-level address mapping scheme and comparing it with the representative schemes using a set of application traces.

The remainder of this paper is organized as follows. Section II introduces the observations of demand-based page-level mapping schemes and our motivation. Section III presents the proposed translation information management scheme. Section IV presents the experimental results. The conclusion is presented in Section V.

II. Problem Analysis and Motivation

In this section, we first introduce DFTL and its translation information management. Then we analyze the critical issues in the translation information management.

DFTL, as a representative demand-based page-level mapping scheme, is proposed to improve the system performance by using page-level mapping and reduce the requirement of large RAM space. In order to store the entire address translation information (i.e., address mapping information), there are two kinds of blocks in DFTL: *Data Block* and *Translation Block*. Data blocks store the data of the requests from file system, while translation blocks are used to maintain the page-level mapping table. There is no fixed area preallocated for translation blocks so that data blocks and translation blocks share the whole flash physical space. One translation page in the translation block contains 512 logical-to-physical mapping items (each item costs 4 bytes) and is tracked by *Global Translation Directory (GTD)*. Moreover, a *Cached Mapping Table (CMT)* maintains a small number of frequently used mapping items in the RAM to improve the performance by exploiting temporal locality. GTD and CMT are two important components in the translation information management of DFTL.

The operations on translation information mainly come from two sources. First, in the cached mapping mechanism, fetching

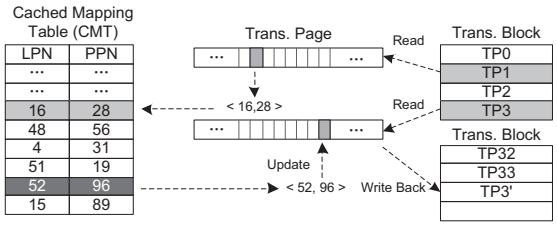

Fig. 1. The Cache Mapping Mechanism in DFTL.

a mapping item from flash device to RAM cache causes a read operation of translation page. When there is a victim updated mapping item should be evicted from cache to flash, the related translation page needs to be read first and the corresponding mapping item in the page is updated to the version of that in the cache. At last, the updated translation page is rewritten to another free space. That is, one cache miss leads to one read operation in the best case, but produces two read operations and one write operation in the worst case. Figure 1 shows an example. Suppose that the mapping item (16,28) needs to be fetched in the CMT. DFTL first reads TP1 from the translation block and then put one mapping item (16,28) from 512 items to the CMT. Moreover, if an updated mapping item (52,96) needs to be evicted from the cache, the corresponding translation page TP3 will be read first. Then the updated item updates the old version in the TP3. Finally, the updated TP3 is written back to the current translation block. From the example, we can see that there are many extra read and write operations on translation information following a single data request, which significantly degrades the system performance.

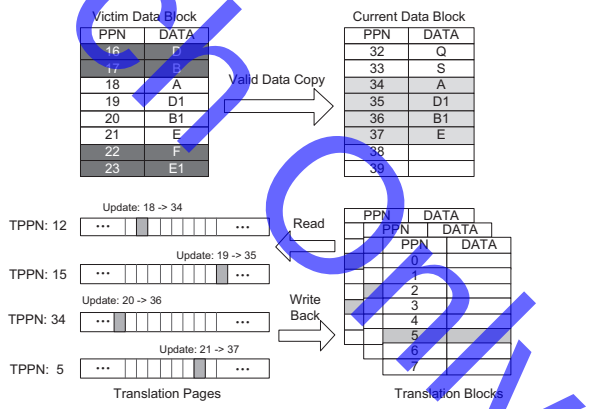

Fig. 2. The Translation Page Updates during Garbage Collection in DFTL.

The second source of translation page operations is from garbage collection. When a data block is reclaimed via garbage collection, all corresponding translation pages for valid data pages in this block should be updated after the valid data pages are copied. Thus, there exists extra translation overhead during the data block garbage collection. As shown in Figure 2, there are four valid pages in the victim block (PPN from 18 to 21) and these data pages correspond to different translation pages, whose TPPNs are 12, 15, 34, and 5, respectively. After copying valid data pages to the current data block, four translation pages need to be read and written back after updating only one map-

ping item in each translation page. Therefore, when there are many valid data pages in the victim block, it may produce lots of extra translation operations during the garbage collection.

There are two problems in translation information management in DFTL. First, the caching mechanism is based on mapping items. That is, after reading the whole translation page, it only caches one mapping item from 512 items in the page. However, for sequential read/writes, it is very possible that we may need other mapping items very soon. Second, when handling write requests, data is stored sequentially in physical pages. Therefore, data pages that belong to different translation pages may map into one data block. If the number of corresponding translation pages for the victim block is very large, it may cause many translation page copies during the garbage collection. In this paper, we aim to solve these problems and present an optimized translation information management scheme.

III. TPM: Optimized Translation Information Management

In this section, we present our optimized translation information management scheme in NAND flash memory storage systems. We will first introduce the system architecture. Then, we present the new caching mechanism and multiple-write-pointers strategy in our management. Finally, the process of garbage collection under TPM is introduced.

A. Overview

In TPM, the page-level address mapping table is stored in translation pages in NAND flash. Each translation page contains multiple address mapping items. TPM optimizes translation information management in two aspects, *Caching Mechanism* and *Multiple Write Pointers Strategy*. The architecture of TPM is shown in Figure 3. In our caching mechanism, the granularity of CMT is a translation page, so we can fully make use of the translation information in the translation page. The new CMT also records the start logical address (SLA) of the translation page, which is used to locate the GTD item when evicting the mapping item. To support our caching mechanism and multiple write pointers strategy, GTD is extended to record write pointers and cache indexes. Therefore, GTD can combine with the CMT so as to jointly manage the translation mappings, data writing points, and CMT items. A *Free Block Pool* is used to manage free blocks in the flash and allocate free block on demand.

B. Caching Mechanism

In TPM, the granularity of CMT is a translation page. With our scheme, both temporal locality and spatial locality are improved. Compared with the caching mechanism in DFTL, TPM can get benefits once there is more than one access on the same cached translation page.

Since CMT and GTD have the same mapping granularity, our cached mapping mechanism can easily combine with GTD. We maintain the cache index for each item in the GTD and change the access way of CMT. GTD item index can be located first by dividing logical page number with the number of

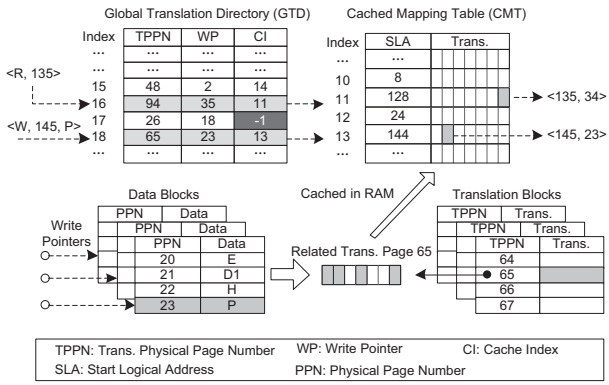

Fig. 3. The Overview of TPM.

mappings that one translation page can store (e.g., 512). After getting the GTD item index, we can get both the cache index and write pointer of the item, where the cache index can directly locate the corresponding mapping information while the write pointer that points to a physical page the data can be written. Therefore, we reduce the searching time of CMT by using index to locate the mappings. "-1" in the cache index means that the corresponding mapping is not cached in the CMT. Figure 3 shows an example of reading data. Suppose each translation page stores 8 mapping items and LPN of the coming read request is 135. We first get the GTD index by computing $\lfloor 135/8 \rfloor = 16$ and then get the cache index that is 11. Therefore, the mapping of (135,34) is located in the cached translation page and we can read the data with the PPN 34.

The replacement policy of the CMT is the *Least Recently Used (LRU)* algorithm but it is optimized by considering the update counts of cached translation pages. That is, if a translation page cached in the CMT has never been updated (i.e., the update counts is zero), it can be directly evicted from the cache without any write back operation.

C. Multiple Write Pointers Strategy

We propose a strategy called "multiple write pointers" to group data pages corresponding to one translation page into one data block. Different from the unified write pointer maintained in the CDB in DFTL, in our strategy, each GTD item maintains its own write pointer that points to an available free page in one data block. For each write request, through the corresponding GTD item located shown above, we can get the write pointer to write the data. By using the same write pointer for one translation page, data with the logical address corresponds to the same GTD item (i.e., translation page) is organized into the same data block. As TPM combines the GTD and CMT, GTD index is only computed once so that both the caching mechanism and multiple write pointers strategy can make use of the information of the GTD item.

At the initial time, all free blocks belong to *Free Block Pool* and write pointer for each GTD item is not allocated. To distinguish the initial state, we use "-1" to represent the state that write pointer is not available. With the coming write requests, if the write pointer of the corresponding GTD item is available, the pointed data page can be directly allocated to serve the request. Otherwise, as the write pointer is not available or the

corresponding data block is full, a new free block is allocated from free block pool to handle the request. After issuing write operation, the write pointer is updated to the next available free page of the allocated block. Since the data stored in a data block is allocated by the pointer of the same GTD item, we can ensure that each data block will correspond to only one translation page. When the free block pool becomes empty, garbage collection will be invoked and the reclaimed victim physical blocks are put back to the free block pool. Algorithm III.1 illustrates the process of writing data through our strategy.

Figure 3 shows an example of our multiple write pointers strategy. Suppose each block contains 4 physical pages and each translation page can store 8 mappings. There is a write request with LPN 145 and the corresponding GTD index is $\lfloor 145/8 \rfloor = 18$. Then we can get the write pointer that points to the data page with PPN 23. Thus, the data "P" can be directly written to the page. After writing, the translation page will be cached into the CMT and the mapping is updated to (145,23). To the current data block, the translation information corresponding to the pages with PPN from 20 to 23 belongs to only one translation page with TPPN 65. That means there is at most one translation page copy overhead when this block is selected as the victim block. Therefore, we can get benefits from our multiple write pointers strategy in the garbage collection, since the extra translation page updates are minimized.

D. Garbage Collection

There are two kinds of garbage collections in the TPM, *translation block garbage collection* and *data block garbage collection*. By optimizing the caching mechanism, translation pages with closed hot or cold degrees would be written back to the same translation block. Our mechanism adopts LRU algorithm which replaces the least accessed translation page from cache to flash. Since there is only one translation write pointer in our scheme, translation pages evicted in the continuous period are written into the same translation block. As a result, there is more possibility that lots of invalid translation pages exist in the victim translation block, which can further reduce the translation page copies overhead.

Algorithm III.1 Multiple Write Pointers Strategy

Require: A logical page number (LPN). N_m is the number of mapping items stored in a translation page
Ensure: An available physical page number (PPN) for (LPN).
1: Compute the index (i) of GTD by dividing LPN with N_m.
2: **if** $GTD[i].WP$ == -1 **then**
3: **if** *Free Block Pool* is $\emptyset$ **then**
4: **while** *Free Block Pool* is $\emptyset$ && $GTD[i].WP$ == -1 **do**
5: Trigger GC
6: **end while**
7: **end if**
8: **if** $GTD[i].WP$ == -1 **then**
9: Allocate a free block from *Free Block Pool* to $GTD[i]$.
10: $GTD[i].WP \Leftarrow first\_page(AllocatedBlock)$.
11: **end if**
12: **end if**
13: $PPN \Leftarrow GTD[i].WP$
14: **if** allocated block of $GTD[i]$ is full **then**
15: $GTD[i].WP \Leftarrow$ -1.
16: **else**
17: $GTD[i].WP \Leftarrow next\_page(GTD[i].WP)$.
18: **end if**

For data block garbage collection, the number of translation pages corresponding to each data block should be maintained to one after garbage collection. To achieve that, TPM reuses the multiple write pointers strategy in the data block garbage collection. In data block garbage collection, we first consult GTD to get the corresponding write pointer. After copying valid data by using the write pointer, if the corresponding write pointer is still available, we even don't need a swap block. Otherwise, when the allocated block corresponding to the GTD item is used up, we allocate another swap block. Then the swap block becomes an allocated block to the GTD item and the erased block is considered as a new swap block.

IV. Evaluation

In this section, we present the experimental results with analysis to demonstrate the effectiveness of our proposed scheme. We compare TPM with a representative demand-based page-level mapping scheme DFTL [11]. in terms of four performance metrics: cache hit ratio, the number of translation operations, system response time, and the number of block erase counts. We select DFTL for comparison because it has been compared with both block-level and hybrid-level mapping schemes and results show that it can provide better performance.

A. Experimental Setup

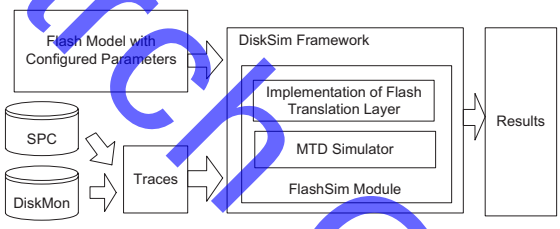

Fig. 4. The Framework of Simulation Platform

Figure 4 shows the framework of our simulation platform. FlashSim, a module of DiskSim [10], can manage and supply basic operations of a flash memory chip and has been widely used to evaluate the performance of FTL schemes. Thus, we adopt FlashSim as our simulator in the evaluation. We implemented our scheme and DFTL in FlashSim. In the simulation, a 32GB NAND flash memory storage system is configured. The parameters in the simulation are given in Table I.

We use a set of benchmarks from both real-world and synthetic traces to study the performance of different schemes. The traces used in our simulation are summarized in Table II. Among them, *Websearch* [4] is a read-dominant I/O trace obtained from Storage Performance Council (SPC) [3]. *Financial* is an I/O trace with high sequential accesses from an OLTP application running at a financial institution also obtained from SPC. *Systemdisk1-3* are traces collected from a laptop running Diskmon [2]. There is a warm-up process that writes the data into the NAND flash before the simulation starts so that all read requests can read data from simulator.

TABLE I
PARAMETERS OF THE NAND FLASH MEMORY

Parameter	Value
Total capacity	32GB
The percentage of reserved free blocks	15%
The minimum number of free blocks	3
The number of planes in the chip	8
The number of blocks per plane	2048
The number of pages per block	64
Page size	2KB
Page read latency	$0.029ms$
Page write latency	$0.2059ms$
Block erase latency	$1.5ms$

TABLE II
TRACES USED IN THE SIMULATION

Traces	Num.of Req.	Write (%)	Avg.Req.Size (KB)
Websearch	1,055,448	0.02	15.05
Financial	3,698,864	17.66	5.24
Systemdisk1	670,412	71.89	42.30
Systemdisk2	1,730,415	67.21	41.10
Systemdisk3	875,928	63.44	47.75

B. Results and Discussion

In this section, we present the experimental results of TPM and DFTL in terms of four performance metrics: cache hit ratio, the number of translation operations, system response time, and the number of block erase counts.

1) Cache Hit Ratio: Cache hit ratio is one of the most important factors that significantly influence the system performance. To make a fair comparison, we evaluate the performance under the same RAM footprint in our scheme and DFTL. The size of CMT is set to 128KB, 256KB, 512KB, and 1024KB. As shown in Figure 5, our caching mechanism can reach at least 89.27% cache hit ratio while the DFTL has an average of 40.14% cache hit ratio. In the demand-based approach, each translation page can hold the mapping items with 1MB ($512 \times 2KB$) consecutive logical address space. As most write requests are with consecutive logical addresses, caching the whole translation page can significantly improve the cache hit ratio. For read-dominant workload, the addresses from read operations are usually random so that it has only about 5% improvement on cache hit ratio when the cache size is increased. However, for write-dominant workload, there are a large amount of sequential write requests. Thus, the cache hit ratio has about 12% to 20% improvement as the cache size increases. Specially, since the address span of the data requests is relative small, the Financial trace can get higher cache hit ratio compared to others workloads.

To make a fair comparison and show the results clearly, for other performance metrics, we select 512 KB as the CMT size in the experiments, and the following results of DFTL and TPM are based on this configuration.

2) Translation Page Operations: The main objective of our scheme is to reduce the extra translation operations in the demand-based mapping scheme. Since TPM uses new caching mechanism to cache translation pages so as to improve the cache hit ratio, the number of translation page operations is significantly reduced. Moreover, by using multiple write pointers

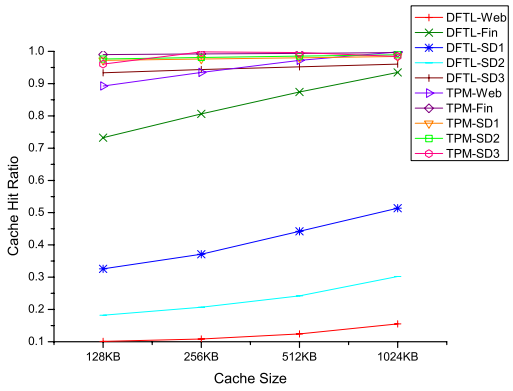

Fig. 5. The Cache Hit Ratio under Different Cache Size

strategy, our scheme minimizes the number of translation pages associated with a data block, so that the number of translation page copy operations in each garbage collection process is at most one. Therefore, our scheme can significantly reduce the translation page operations compared with DFTL. As shown in Figure 6, our scheme can achieve an average 90.93% reduction on translation page operations.

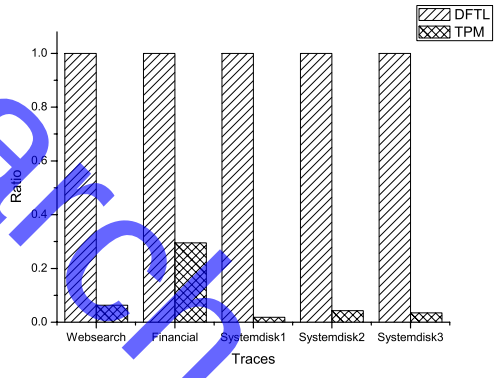

Fig. 6. The Normalized Number of Translation Operations.

3) System Response Time: System response time is one of the most important metrics in the design of NAND flash memory storage systems. We have compared the system response time of DFTL and our proposed scheme. The experimental results are shown in Figure 7. Compared with DFTL, our scheme achieves an average 22.14% reduction on average system response time. TPM can greatly improve the cache hit ratio, and effectively reduce the number of translation page operations. Therefore, our scheme has better average performance compared with the DFTL.

4) Block Erase Counts: Since TPM reduces significant translation operations by using optimized translation information management, it can effectively reduce the block erase counts and improve the endurance of the NAND flash memory. Moreover, different size of CMT leads to different reduction on erase counts. That is, the large CMT size will improve the cache hit ratio and indirectly reduce more translation page operations and block erase counts. As shown in Figure 8, our scheme can reduce an average of 26.51% block erase counts compared with DFTL.

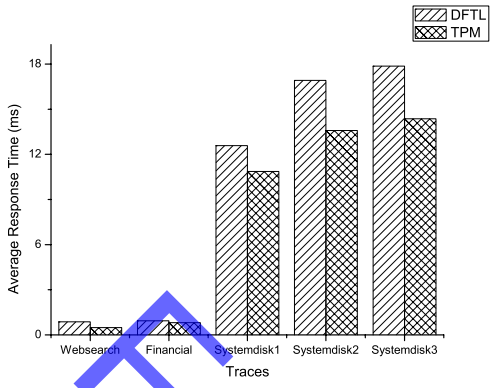

Fig. 7. Average System Response Time.

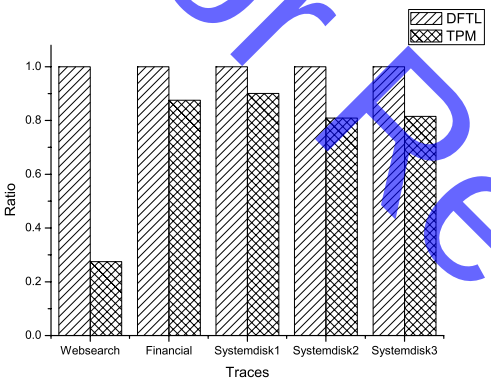

Fig. 8. The Normalized Number of Block Erase Counts

C. Overhead

In order to optimize translation information management, we extend GTD and CMT by recording extra information such as the write pointer and cache index for each GTD item. However, the size of GTD is very small that only needs 2KB RAM footprint to support 1GB space. In our scheme, GTD only costs 96KB of RAM footprint for 32GB NAND flash memory. Our multiple write pointers strategy only introduces about 4.74% extra space overhead. But TPM achieves an average 90.93% reduction in extra translation operations and a 22.14% improvement in the average response time compared with previous work.

V. Conclusion

In this paper, we have proposed TPM, an optimized translation information management for demand-based page-level mappings in NAND flash memory system. The scheme optimized the caching mechanism to cache translation pages and locate them with indexes in the GTD. Moreover, by using multiple write pointers strategy, data belongs to the same translation page are grouped into one data block. We have evaluated TPM using a set of benchmarks and compared with the representative scheme. The simulation results show that our scheme can significantly reduce translation page operations and improve the average system response time compared with the previous works.

Acknowledgments

The work described in this paper is partially supported by the National 863 High-Tech Programme of China (No.2011AA010103), the National Grand Fundamental Research 973 Program of China (No.2009CB320702), Intel China University Collaboration Fund, the Graduate Research and Innovation Project of Jiangsu Province (No.CXZZ12_0057), the Innovation and Technology Support Programme of Innovation and Technology Fund of the Hong Kong Special Administrative Region, China (ITS/082/10), National Natural Science Foundation of China (Project 61272103), and the Hong Kong Polytechnic University (G-YK24).

References

[1] A Simulator for various FTL schemes. *http://csl.cse.psu.edu/?q=node/322*.

[2] DiskMon for Windows. *http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx*.

[3] OLTP Trace from UMass Trace Repository. *http://traces.cs.umass.edu/index.php/Storage/Storage*.

[4] Websearch Trace from UMass Trace Repository. *http://traces.cs.umass.edu/index.php/Storage/Storage*.

[5] A. Ban. Flash-memory translation layer for NAND flash (NFTL). *M-systems*, 1998.

[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*, pages 858–863, 2009.

[7] H. Cho, D. Shin, and Y. I. Eom. KAST: K-associative sector translation for NAND flash memory in real-time systems. In *DATE '09*, pages 507–512, 2009.

[8] S. Choudhuri and T. Givargis. Performance improvement of block based NAND flash translation layer. In *CODES+ISSS '07*, pages 257–262, 2007.

[9] S. Choudhuri and T. Givargis. Deterministic service guarantees for NAND flash using partial block cleaning. In *CODES+ISSS '08*, pages 19–24, 2008.

[10] B. W. G.R. Ganger and Y. Patt. The DiskSim Simulation Environment Version 3.0 Reference Manual.

[11] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS'09*, pages 229–240, 2009.

[12] P.-H. Hsu, Y.-H. Chang, P.-C. Huang, T.-W. Kuo, and D. H.-C. Du. A version-based strategy for reliability enhancement of flash file systems. In *DAC '11*, pages 29–34, 2011.

[13] T.-W. Kuo, Y.-H. Chang, P.-C. Huang, and C.-W. Chang. Special issues in flash. In *ICCAD '08*, pages 821–826, 2008.

[14] 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*, pages 173–182, 2010.

[15] Z. Qin, Y. Wang, D. Liu, and Z. Shao. A two-level caching mechanism for demand-based page-level address mapping in NAND flash memory storage systems. In *RTAS '11*, pages 157–166, 2011.

[16] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan. MNFTL: an efficient flash translation layer for MLC NAND flash memory storage systems. In *DAC '11*, pages 17–22, 2011.

[17] L. Shi, J. Li, C. J. Xue, C. Yang, and X. Zhou. Exlru: a unified write buffer cache management for flash memory. In *EMSOFT'11*, pages 339–348, 2011.

[18] L. Shi, C. J. Xue, and X. Zhou. Cooperating write buffer cache and virtual memory management for flash memory based systems. In *RTAS'11*, pages 147–156, 2011.

[19] C.-H. Wu and T.-W. Kuo. An adaptive two-level management for the flash translation layer in embedded systems. In *ICCAD'06*, pages 601–606, 2006.