2LGC: An Atomic-Unit Garbage Collection Scheme with a Two-Level List for NAND Flash Storage

Sanghyuk Jung and Yong Ho Song
Department of Electronics Computer Engineering, Hanyang University, Seoul, Korea

Abstract - *In NAND flash memory devices, pages marked "invalid" can remain in blocks and occupy flash space. Therefore, it is necessary to physically eliminate invalid pages and collect valid pages from the victim blocks in order to sustain flash write performance and storage lifespan. Although there have been many research studies on efficient garbage collection techniques, research has focused on victim selection methodologies and no solutions have been proposed for the victim selection process cost overhead. Indeed, the host system quite often suffers unendurable storage-access delays because garbage collection produces much computational over-head when doing victim selection. A novel garbage col-lection mechanism, called "Two-Level List-based Garbage Collection", is proposed in this paper. The victim block selection overhead can be efficiently reduced in this scheme; hence, the responsiveness to host requests is significantly improved.*

Keywords: flash memory, garbage collection, SSD

1 Introduction

There has been a revolutionary change in data storage fields since the development of NAND flash memories. NAND flash memories have been widely used as the storage media of embedded systems such as MP3 players, mobile devices, and digital cameras owing to their non-volatile, high random access performance, and low power consumption flash characteristics. The unit price of flash memory is constantly decreasing because the vendors of flash memories are trying to squeeze more capacity into constantly shrinking silicon dies and adopting *multi-level cell* (MLC) technology [1]. NAND flash storage devices (*i.e., solid state drives*) are becoming a viable solution for satisfying the high performance and low power consumption demands of notebooks and desktop-PCs as well as portable embedded systems with continuing improvements in both capacity and price.

However, NAND flash memory has several restrictions resulting from its architectural characteristics. First, *pages (the minimum data access unit of a flash memory)* are designed to share an identical *word-line* and *blocks (consisting of several pages)* are designed to share an identical *bit-line* in order to provide high density memory devices. The unit sizes of the erase and read/write operations are asymmetric for this reason: read/write operations are performed in a page unit while erase operations should be executed in a block unit. Second, electrons in the flash memory data cells can only be removed through an erase operation once the floating gates of the data cells are charged with electrons; thus, the write operation may have to be preceded by an erase operation. This characteristic is sometimes called "*erase-before-write*". Third, NAND flash causes an unpredictable electron-leakage problem due to the wearing out of the silicon oxide film which is located between the floating gate and the transistor gate in a cell. The electron-leakage problem mainly causes uncorrectable bit errors and, therefore, the lifespan of flash memory expires after performing a limited number of *program/erase (P/E) cycles*.

In order to hide these constraints of NAND flash memories, current flash-based storage systems use a special interface called a *flash translation layer (FTL)* [2-5], which is supported by the storage firmware. The main role of the FTL is to make flash storage a virtual in-place updatable storage device. For example, the FTL redirects each write request to the physical flash area and marks the previously programmed page *invalid* when the host repetitively issues write operations on the same address space. The flash storage can generate a relatively small number of page-copy operations and block-erase operations from this FTL emulation technique, so it is helpful for improving NAND flash memory durability.

However, a problem may arise when pages marked "invalid" remain in blocks and occupy flash space. Therefore, it is necessary to physically eliminate invalid pages and collect valid pages from the victim blocks in order to sustain flash write performance and storage lifespan. This sequence of operation processes is called *garbage collection* [6-7]. The performance and durability of the flash storage can be kept stable if the garbage collection mechanism is efficiently designed.

There has been much research [8-15] on efficient garbage collection techniques and various victim block selection methods that cut down on operational overhead have been proposed. However, these research studies have only focused on victim block selection methodologies and have not proposed any solutions for the cost overhead of victim selection processes. Indeed, the host system quite often suffers unendurable storage-access delays because garbage collection produces great computational overhead when performing victim selection processes. Therefore, the storage-access performance and responsiveness of flash storage can be improved by reducing the cost overhead of victim selection processes.

A novel garbage collection mechanism known as *Two Level list-based Garbage Collection (2LGC)* is proposed in this paper. In the proposed scheme, the FTL stores block addresses into two-level lists when the numbers of invalid-marked pages in those blocks pass a threshold. The stacked block map addresses in two-level lists are used for victim selection processes. The FTL can efficiently reduce the victim block selection overhead in this manner; hence, the responsiveness to host requests is significantly improved.

Flash storage performance was analyzed using *a flash storage prototype platform* [16], which consists of hardware parts (i.e., NAND flash controllers, memory controllers, and a CPU) and software parts (i.e., FTL) in order to verify the effectiveness of the 2LGC scheme. The resulting 2LGC scheme offered significant benefits, such as a high performance storage-access ability and a host command delay drop, compared to an *on-demand victim search technique* in our experiments.

The rest of this paper is organized as follows. A preliminary overview of garbage collection mechanisms and latency hiding skills is described in the next section. Related works, including victim block selection techniques and their latencies are reviewed in Section III. The proposed garbage collection scheme is explained in Section IV and its feasibility is discussed. A comparison of the 2LGC scheme to the on-demand victim selection technique in terms of responsiveness is analyzed in Section V. Finally, conclusions are drawn from this study in Section VI.

2 Preliminaries

2.1 Garbage Collection Mechanism

The FTL address re-mapping technique has a problem in that invalid-marked pages occupy the flash area without being erased. If these invalid-marked pages accumulate in the flash storage, the problem of no more available free blocks arises, although the capacity of used space is much smaller than that of the flash storage. Therefore, it is necessary to physically eliminate invalid-marked pages and make free blocks available in order to sustain flash write performance and storage capacity. Garbage collection is thus needed for reclaiming invalid-marked pages scattered over blocks so that the invalid-marked pages can again become free pages. The garbage collection sequence and its operational cost are summarized as follows.

- The FTL selects some blocks which are expected to have the lowest garbage collection cost as victim blocks when garbage collection is triggered. The P/E cycles or hot/cold identification of each block as well as the number of invalid pages can be considered when selecting victim blocks. The FTL targets entire blocks as victim-candidate blocks of flash storage, which may cause a serious operational cost.
- The FTL implements block erase operations in order to physically remove the invalid-marked pages from the selected victim blocks. However, the FTL should allocate a free block and then copy all the valid pages from victim blocks to the free block because the victim blocks may have valid pages. Consequently, the valid pages scattered over victim blocks are copied to one free block.
- The FTL updates the mapping information after generating a data block full of valid pages and the victim blocks are then physically erased. The erased blocks are logically located in the *free block pool* and reallocated when necessary.

2.2 Latency Hiding of Garbage Collection

The storage access frequency of a host system is called *data access intensity* and is affected by workload characteristics. Storage I/O performance is directly influenced by the factor of data access intensity. During the high data access intensity, the host event queue can be filled with storage-access requests and hence must be handled immediately. On the other hand, during the low data access intensity, the storage system becomes largely idle and its bandwidth is greatly under-utilized [17]. Therefore, a large portion of the run-time garbage collection cost can be saved if the FTL implements time-consuming operations during low data access intensity.

However, the *request-pending* problem must be considered when adopting an idle-time garbage collection technique. The FTL should suspend garbage collection and immediately back into the request handling process when the host issues storage-access requests during low data access intensity. Therefore, the responsiveness to the host storage-access requests can be improved if the garbage collection operations are *preemptively* designed.

3 Related Works

In this section, victim block selection techniques of garbage collection are analyzed and an explanation is given for the operational latencies incurred by those techniques.

3.1 Victim Block Selection

Various cost-based garbage collection techniques [6-15] have been proposed over the past several years. In the *Greedy algorithm*, Wu et al. [8] first suggested that the FTL selects blocks having the largest number of invalid-marked pages as victim blocks. In this way, the FTL can reduce the number of page program operations from victim blocks and improve the performance and durability of NAND flash storage. However, the subsequently proposed algorithms, such as the *Cost-benefit* scheme [9], indicate a problem in that the Greedy algorithm is not suitable for prolonging the lifespan of flash storage because the Greedy algorithm selects victim blocks without considering their P/E cycles.

Therefore, many subsequent studies [10-15] have proposed victim block selection techniques considering wear leveling costs. However, the *dynamic wear leveling* and *static wear leveling* schemes [18-19] have already been adopted inside flash storages, so there is no need to consider both

wear leveling and garbage collection issues together. Consequently, if the wear leveling cost is not taken into consideration, the Greedy algorithm shows the highest performance in terms of garbage collection cost compared with other victim selection techniques.

3.2 Victim Selection Latency

Measuring computational latencies from garbage collection is a totally different issue because the previously published research on garbage collection schemes did not focus on victim block management costs such as victim selection overheads and sorting delays. Thus, victim block management costs must be carefully analyzed in order to decrease garbage collection latencies.

The garbage collection operational delays, based on previous research, are as follows. First, the FTL extracts the number of invalid-marked pages from each block by searching the entire flash memory space. Extracting the number of invalid-marked pages will take longer for larger capacity flash storage because flash storage has the same number of block map entries as the number of data blocks. Second, the FTL continuously compares the number of invalid-marked pages from each block until enough victim blocks are selected in order to select victim blocks having the largest number of invalid-marked pages for each block. Although it is assumed that the FTL uses a quick sort algorithm which has the best performing speed among the well-known sorting algorithms, the operational delay becomes O(*NlogN*) in general case, and O(N^2) in the worst case.

4 2LGC

4.1 Victim Block Selection

The 2LGC scheme is able to isolate victim block selection from garbage collection. In short, this scheme maintains the victim priority of target blocks by sorting the blocks by the number of invalid-marked pages during run-time. The two-level lists are used to implement the run-time victim block searching technique, as shown in Figure 4: a candidate list and a garbage block list (along with other de-tails that are explained in Section IV (C)). In the 2LGC scheme, the FTL stores physical block addresses in the two-level lists depending on the numbers of invalid-marked pages and uses them when necessary. This allows the flash storage to reduce block searching overheads and victim block sorting costs during garbage collection. Figure 1 rep-resents the controller architecture of a NAND flash storage and the location of the *2L-list.*

Figure 2 shows the 2LGC map entries. First, the page map table is an essential data structure of a page mapping FTL. The main role of this table is to translate *logical page numbers (LPNs)* from a host system into *physical page numbers (PPNs)* in NAND flash memories. Second, the aim of the block map table is to store physical block information whether the block is available or not when the FTL allocates free blocks. The block map table is also an essential data structure for supporting garbage collection or wear leveling algorithms.

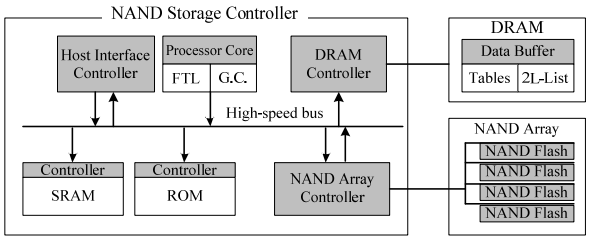

Fig. 1. Flash storage controller architecture.

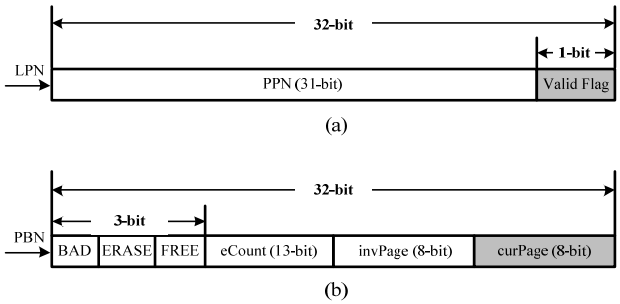

Fig. 2. 2LGC map entries: (a) page map entry and (b) block map entry.

As shown in Figure 2(a), each page map table entry is composed of a 31-bit PPN and a 1-bit *page validation-mark flag*. The number of entries in a page map table is the same as the number of pages in the flash storage. If the storage access request from the host is issued to the flash storage, the FTL searches the 31-bit PPNs of the page map table using the LPNs (if a physical page size of 8KB is assumed, the 31-bit page number can represent 2^{44} bytes or 16TB). On the other hand, the FTL has to check the page validation-mark flag using the PPNs in order to confirm whether or not the data in physical page space are valid. The address space can be more efficiently saved by combining a 31-bit PPN and a 1-bit page validation-mark flag into a single 32-bit I/O bus width register.

There are six entries in the block map table as shown in Figure 2(b). The usage of each entry is as follows. First, 2LGC uses three flags for supporting address translation. A 1-bit *bad block-mark flag*, a 1-bit *free-mark flag*, and a 1-bit *erase-mark flag* represent whether the physical block is bad or not, free or not, and erased or not, respectively. Second, 2LGC uses two page offset entries for maintaining page information within a block. An 8-bit *invPage offset* shows how many invalid pages are involved within a block and an 8-bit *curPage offset* explains which page of the block is available for programming. Lastly, the 13-bit *eCount number* stands for the number of times each block has been erased.

4.2 Single-Block Garbage Collection

The FTL selects multiple victim blocks, copies valid pages into one free block, and invalidates the victim blocks in the on-demand victim selection techniques. For example, the

FTL updates the page map table with a new physical page number and erases victim blocks whose physical block numbers are 2, 4, and 5 when page copy operations are finished, as shown in Figure 3(a). The FTL needs to copy four pages and erase three blocks as well as to search victim blocks and update map tables during this multiple-block garbage collection process. As seen in this example, the on-demand victim selection techniques can make relatively more reusable free blocks, but may cause a large peak delays within only one garbage collection

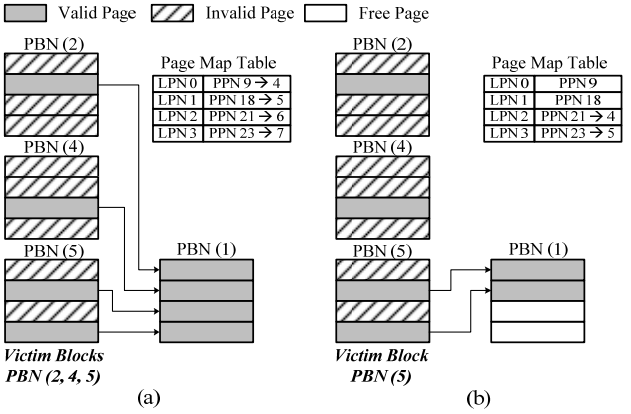

Fig. 3. (a) An existing garbage collection and (b) 2LGC mechanism.

On the other hand, the 2LGC scheme can separate a garbage collection sequence into several single-block garbage collection operations, so it is possible to improve responsiveness and make the flash storage preemptive. Note that the 2LGC can implement single-block garbage collection mainly because the FTL can use a page offset for each block stored in the curPage offset entries, as shown in Figure 2(b). All the blocks can be reallocated as non-free states through the use of curPage offsets. For example, as shown in Figure 3(b), the FTL updates the page map table with a new physical page number and erases the block whose physical block number is 5 when page copy operations are finished. In this case, the FTL needs to copy two pages and erase one block and update the map tables during the single-block garbage collection process. Compared with on-demand victim selection mechanisms, 2LGC single-block garbage collection is quite effective for supporting a preemptive storage system

4.3 Algorithm

Figure 4 shows the two-level lists used in the 2LGC algorithm. The operational sequence is as follows. The FTL continuously checks the number of invalid-marked pages for the corresponding block whenever the page validation-mark flag of each page map entry is updated. If the number of invalid-marked pages in that block is over a threshold value, the 2LGC stores the block address in the *Candidate list*. The entries in Candidate list are not to be sorted in the initial state. The 2LGC can reconstruct the Candidate list only when the following two cases occur.

(1) If the Candidate list is full, a block whose entire pages are invalid is demoted into the *Garbage block list*. If *user workloads*, such as file copies and internet explorations, are used, several blocks can be expected to be demoted into the Garbage block list because they include a large number of sequential program operations. However, if the Candidate list does not have such blocks any more, the 2LGC sorts the blocks of the Candidate list in the order of invalid-marked page numbers and removes a block address from the tail of Candidate list. The block address currently being added to the Candidate list is quickly demoted to the Garbage block list if pages in the block are sequentially programmed.

(2) When garbage collection is triggered, the 2LGC firstly checks the Garbage block list and selects a block from the head of the Garbage block list as the victim. If the Garbage block list is empty, the 2LGC then checks the Candidate list and selects a block from the head of the Candidate list.

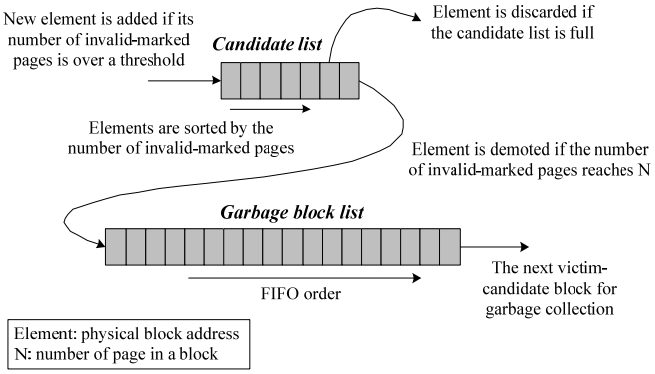

Fig. 4. Two-level lists in the 2LGC algorithm

In 2LGC, the FTL is designed to use a single-block garbage collection scheme, as mentioned in Section IV (B). Although the host issues storage-access requests during garbage collection, the FTL can back into the request handling process without putting a bookmark in the garbage collection sequence because the block addresses already exist in the Candidate list. In the same way, background garbage collection can be implemented atomically. The 2LGC has only to add the block address to the Garbage block list when finishing background single-block garbage collection.

5 Experiments

5.1 Performance Evaluation

In order to verify the effectiveness of 2LGC scheme, we conducted real-system based experiments using a flash prototype platform board [16] equipped with an *INDILINX barefoot SSD controller*. The SSD controller of the platform board consists of hardware parts (i.e., NAND flash controllers [20], memory controllers, and a CPU) and software parts (i.e., FTL), so an algorithmic evaluation can be performed by redesigning the firmware inside the SSD controller. Moreover, the most accurate and reliable experiments can be conducted through this platform because the platform board is connected

by a *SATA2 interface* using a notebook or desktop as storage [21]. The platform board and the specification of INDILINX barefoot SSD controller are shown in Figure 5 and Table 1, respectively.

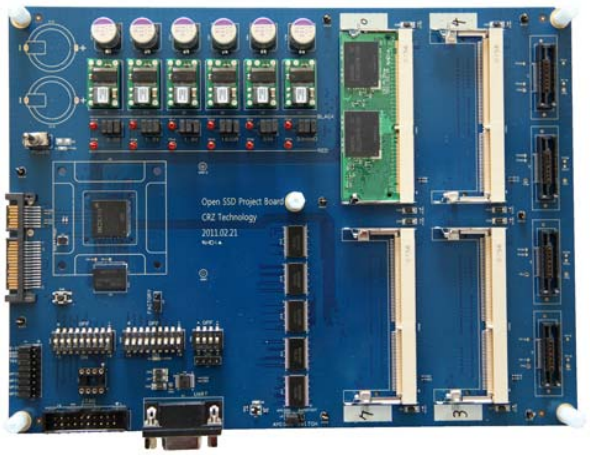

Fig. 5. Flash prototype platform board.

Table 1. SSD controller specification of platform board

Component	Specification
Core processor	ARM7TDMI-S (87.5MHz)
Host interface	SATA2 (3Gbps)
Internal SRAM size	96KB
Mobile SDRAM size	64MB (175MHz)
Number of channels	8
Number of ways	2
Clustered page size	32KB
NAND flash model ID	K9LCG08U1M
NAND package/die size	8GB/4GB
SSD capacity	64GB
ECC module	8/12/16-bit BCH per sector

The *IOMeter benchmark* [22] is used for generating meaningful workloads in the experiments. In order to extract the accurate experimental data, we exclude data I/Os caused by operating systems or file systems. The platform board is connected to the host as flash storage without installing any operating systems or formatting any file systems because the IOMeter benchmark can handle direct storage-access operations to the unformatted data storage. The IOMeter benchmark can also organize workloads of various read/write and random/sequential access intensities; thus, configurable workloads with the desired properties can be generated. The workload variation is shown in Table 2 (the minimum storage access unit size is 32KB due to the page clustering technique). Finally, the flash storage is programmed with a random/sequential write ratio (r: 50/ s: 50) for aging entire pages to enable the measurement of garbage collection operation latencies. In this experiment, the threshold value is defined as 3/4 of the number of pages in a block, the Candidate list size is 1/20 of storage capacity, and the Garbage block list size is 1/10 of storage capacity, respectively.

Table 2. Workloads

Trace	Read/Write Ratio	Random/Sequential Access Ratio
Workload 1	r: 80/ w: 20	r: 100/ s: 0
Workload 2	r: 80/ w: 20	r: 0/ s: 100
Workload 3	r: 20/ w: 80	r: 100/ s: 0
Workload 4	r: 20/ w: 80	r: 0/ s: 100

5.2 IOPS and Execution Time

Figure 6 shows the flash storage IOPS varied with the garbage collection scheme during the time intervals. In this experiment, the IOPS represents the number of page-level commands generated by the FTL, not the number of request-level commands issued by the host. The IOPS of the 2LGC scheme were compared to that of the on-demand victim selection technique using the IOMeter benchmark workloads shown in Table 2. As shown in the figure, the average IOPS of the 2LGC scheme is superior to that of the on-demand victim selection technique because 2LGC saves garbage collection costs. There is a significant decrease in the IOPS of the on-demand victim selection technique because it causes entire block searching and victim selection overhead when garbage collection is triggered (see Section III (B) for more details of garbage collection latency). On the other hand, the 2LGC scheme can reduce the system latencies caused by garbage collection because it maintains victim-candidate blocks within the two-level lists during system run-time. Moreover, the peak delays can be minimized and responsiveness to the host improved because of the effectiveness of the single-block garbage collection technique.

6 Conclusion

In this paper, we have studied the operational mechanisms and the computational overheads of garbage collection. The garbage collection was found to have too much computational overhead to find victim blocks, resulting in unendurable host system access latency (very low responsiveness) and performance degradation.

However, the proposed 2LGC garbage collection scheme eliminated the computational overheads due to victim block selection from the critical path of the garbage collection operations. The responsiveness to host system requests was also improved by making the garbage collection operation preemptive. The 2LGC scheme achieved significant performance improvement in flash storage bandwidth and request processing latency in comparison to the on-demand victim selection technique in our experiments.

7 Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0017147).

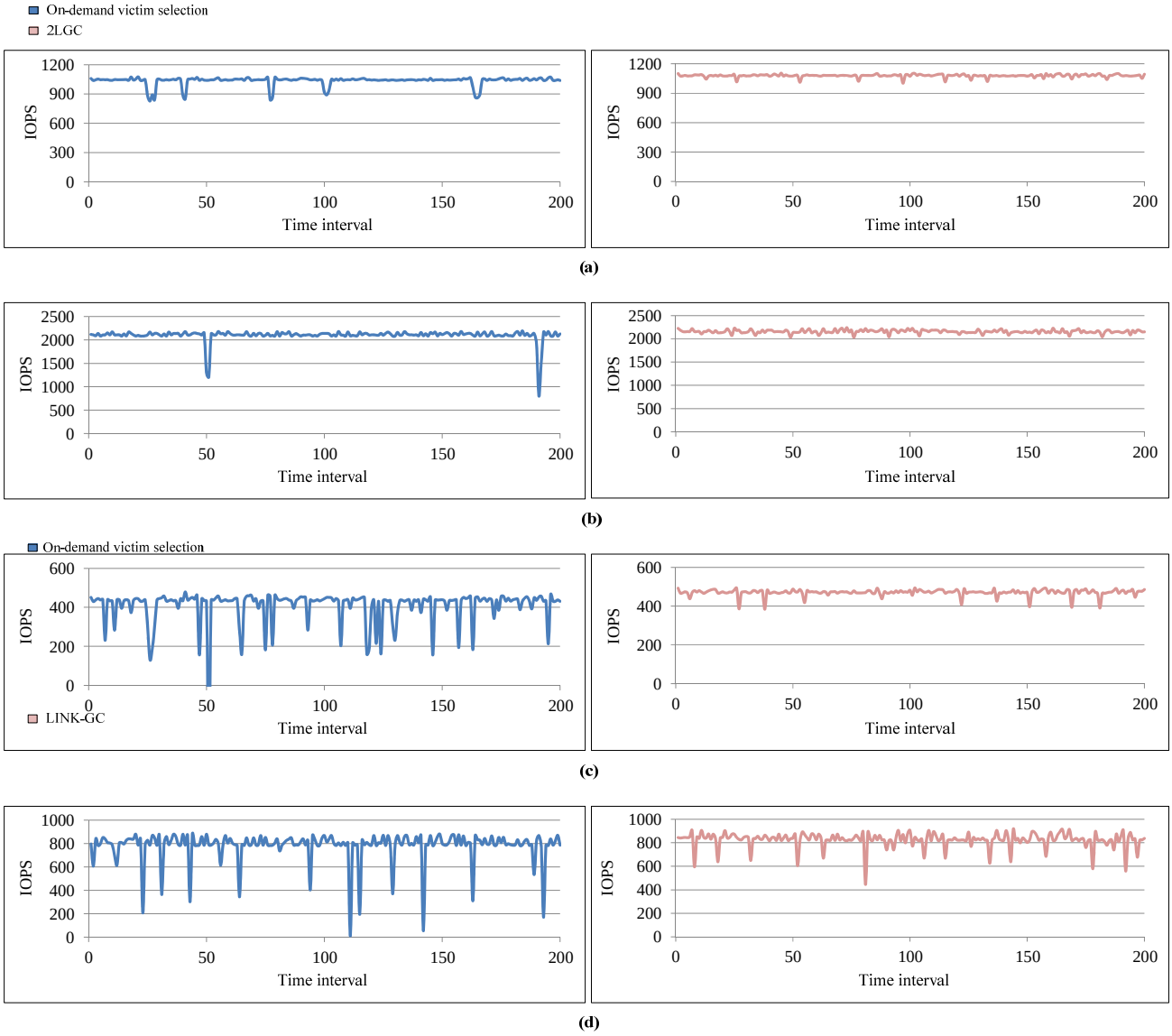

Fig. 6. IOPS comparison between the on-demand victim selection scheme and the 2LGC scheme, (a) workload 1, (b) workload 2, (c) workload 3, and (d) workload 4.

8 References

[1] Sanghyuk Jung, Sangyong Lee, Hoeseung Jung, and Yong Ho Song, "In-page error correction code management for MLC flahs storages," *Proceedings of the IEEE MWSCAS*, 2011.

[2] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sang-Won Park, and Ha-Joo Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions on Embedded Computing Systems, vol. 6, no.3,* article 18, July 2007.

[3] Sanghyuk Jung, Jin Hyuk Kim, and Yong Ho Song, "Hierarchical architecture of flash-based storage systems for high performance and durability," *Proceedings of the IEEE/ACM International Conference on DAC*, 2009.

[4] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar, "DFTL: A flash translation layer employing demand-based se-lective caching of page-level address mappings," *Proceedings of the ACM International Conference on ASPLOS*, 2009.

[5] Sanghyuk Jung, Yangsup Lee, and Yong Ho Song, "A process-aware hot/cold identification scheme for flash memory storage systems," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 339-347, May 2010.

[6] Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo, "Real-time garbage collection for flash-memory storage systems of

real-time embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 4, November 2004.

[7] Ohhoon Kwon, Kern Koh, Jaewoo Lee, Hyokyung Hahn, "FeGC: An efficient garbage collection scheme for flash memory based storage systems," *The Journal of Systems and Software*, pp. 1507-1523, 2011.

[8] Wu, M., Zwaenepoel, W., "eNVy: a non-volatile, main memory storage system," *Proceedings of the ACM International Conference on ASPLOS*, 1994.

[9] Kawaguchi, A., Nishioka, S., Motoda, H., "A flash-memory based file system," *Proceedings of USENIX Technical Conference*, 1995.

[10] Chiang, M., Lee, P.C.H., Chang, R., "Cleaning algorithms in mobile computers using flash memory," *Journal of Systems and Software*, 1999.

[11] Kim, H., Lee, S., "A new flash memory management for flash storage system," *Proceedings of the COMPSAC*, 1999.

[12] Manning, C., Wookey, "YAFFS Specification," *Aleph One Limited*, 2001.

[13] M-Systems, "TrueFFS Wear-Leveling Mechanism."

[14] Chang, L., "On efficient wear leveling for large-scale flash-memory storage systems," *Proceedings of the ACM SAC*, 2007.

[15] Du, Y., Cai, M., Dong, J., "Adaptive garbage collection mechanism for N-log block flash memory storage systems," *Proceedings of the ICAT*, 2006.

[16] http://www.openssd-project.org/wiki/The_OpenSSD_Project.

[17] Yangsup Lee, Sanghyuk Jung, and Yong Ho Song, "FRA: A flash-aware redundancy array of flash storage devices," *Proceedings of the CODES+ISSS*, 2009.

[18] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo, "Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design," *Proceedings of the IEEE/ACM International Conference on DAC*, 2007.

[19] Li-Pin Chang and Chun-Da Du, "Design and implementation of an efficient wear-leveling algorithm for solid-state-disk microcontrollers," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 1, December 2009.

[20] Samsung flash memory Spec, K9LCG08U1A, Datasheet.

[21] Technical Committee T13 AT Attachment, "Information Technology – ATA/ATAPI Command Set – 2 (ACS-2)," T13/2015-D, Revision 2, August 3, 2009.

[22] http://www.iometer.org/