Dual Greedy: Adaptive Garbage Collection for Page-Mapping Solid-State Disks

Wen-Huei Lin
Department of Computer Science
National Chiao-Tung University
Hsin-Chu, Taiwan, ROC
linwh.tw@gmail.com

Li-Pin Chang
Department of Computer Science
National Chiao-Tung University
Hsin-Chu, Taiwan, ROC
lpchang@cs.nctu.edu.tw

***Abstract*—In the recent years, commodity solid-state disks have started adopting powerful controllers and implemented page-level mapping for flash management. However, many of these models still use primitive garbage-collection algorithms, because prior approaches do not scale up with the dramatic increase of flash capacity. This study introduces Dual Greedy for garbage collection in page-level mapping. Dual Greedy identifies page-accurate data hotness using only block-level information, and adaptively switches its preference of victim selection between block space utilization and block stability. It can run in constant time and use very limited RAM space. Our experimental results show that Dual Greedy outperforms existing approaches in terms of garbage-collection overhead, especially with large flash blocks.**

I. Introduction

Solid-state disks offer a less invasive way to deploy flash storage in mobile computers. They implement a firmware layer, i.e., the flash-translation layer, to emulate a collection of disk sectors and hide flash management from the hosts. Flash memory is a kind of erase-before-write non-volatile memory, and the unit of erasure is much larger than that of read/write. Thus, flash-translation layers handle data updates in a out-of-place manner and adopt a mapping scheme to translate logical sector numbers into physical flash locations.

Choosing the resolution of address mapping is an important design issue of flash-translation layers. Although page-level mapping has a natural appeal of its high write-performance, it requires very large mapping tables which are too large to fit in the RAM space of disk controllers. Thus, many entry-level flash-storage devices such as thumb drives adopt block-level mapping for the minimal table footprints. Many solid-state disks adopt hybrid mapping for good balance between write performance and table size. For example, solid-state disks based on the disk controller GP5086 from Global Unichip implement hybrid mapping and store their mapping table in the 64 KB embedded SRAM. Recently, advanced solid-state disks started adopting powerful controllers. For example, the Intel 510 series employ Marvell's dual-core disk controller Van Gogh with an accompanying 128 MB DDR RAM chip. These solid-state disks implement page-level mapping as their controllers have rich computational resources to do so.

Out-of-place updates produce outdated data in flash memory. Flash-translation layers must timely perform garbage collection that erases flash space occupied by stale data into free space. In addition to flash erasure, garbage collection could also involve copying valid data. Because garbage-collection activities could delay the processing of host requests, flash-translation layers adopt two strategies to reduce this impact: 1) data separation, which identifies frequently updated data (i.e., hot data) and writes data having similar update frequencies to nearby flash space, and 2) victim selection, which prevents garbage collection from copying valid data that will get invalidated in the near future.

Even though prior studies have proposed various techniques of garbage collection for page-level mapping [2], [4], [7], many commodity page-mapping solid-state disks still use primitive garbage collection algorithms. This is mainly because, when these prior approaches were introduced, the mainstream flash capacity was quite small at that time. These prior techniques do not scale up to gigabyte-level flash. In addition, these prior techniques use many parameters that require manual tuning, making them not useful in real products. Recent studies investigated garbage collection for hybrid mapping [8], [9], but these results are not applicable to page-level mapping.

This study presents Dual Greedy for efficient garbage collection in page-level mapping. Dual Greedy is designed to be adaptive and scalable. It requires only block-level information but identifies data hotness at the page level. When selecting victim blocks for garbage collection, Dual Greedy dynamically switches its preference between block space utilization and block stability. Dual Greedy requires no static parameters and is adaptive to various types of host workloads. The execution time of Dual Greedy is not related to the flash capacity, i.e., it can run in constant time. Compared to prior approaches, Dual Greedy is simple yet very efficient.

The rest of this paper is organized as follows: Section II introduces the fundamental issues of garbage collection for page-level mapping. Section III presents a strategy for identifying hot data and separating hot data from non-hot data in flash. Section IV shows an adaptive policy for victim

This work is in part supported by a research grant NSC-98-2221-E-009-157-MY3 from National Science Council, Taiwan, ROC and a research project form Global Unichip Corp.
978-3-9810801-8-6/DATE12/©2012 EDAA

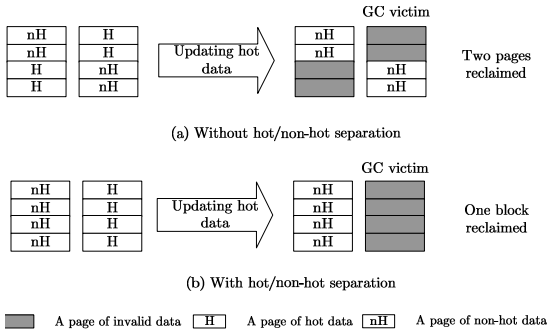

Fig. 1. Separating hot data and non-hot data in flash blocks improves garbage-collection performance.

selection. Section V are experimental results, and this paper is concluded in Section VI.

II. Background

A. Page-Level Mapping

A piece of flash memory consists of an array of blocks, and a flash block is of a fixed number of pages. Flash pages cannot be re-written unless they are erased. Because flash erases in terms of blocks instead of pages, data updates in flash are out-of-place to avoid erasing a flash block for updating a page. As the physical locations of data change after updates, solid-state disks implement a firmware layer called the flash-translation layer that maps logical sector numbers to flash memory addresses. The flash-translation layer adopts a mapping table for address translation, and the mapping table resides in the RAM of the disk controller. Address translation at the page level delivers better random-write performance because new data can be written to any free space in flash memory. However, page-level mapping requires large mapping tables. Gupta et al. and Qin et al. addressed this issue by caching portions of the entire mapping table [3], [10].

Updating data out of place leaves outdated versions of data in flash. When running low on free space, the flash-translation layer starts reclaiming available space occupied by invalid data via block erasure. Before erasing a block, the flash-translation layer must move all valid data out of this block. Such a series of copy and erase activities for reclaiming free space is referred to as *garbage collection*.

There are two important techniques pertaining to garbage collection for page-level mapping: *hot/non-hot separation* and *victim selection*. Hot data refers to the data that are frequently updated by the host. Hot/non-hot separation (or simply data separation) identifies the hotness of data upon write and then writes hot data and non-hot data to different flash blocks. Figure 1 shows how hot/non-hot separation benefits garbage collection. In Fig. 1(a), updating hot data produces two pages of invalid data in each of the two blocks. Erasing anyone of the two blocks effectively reclaims two pages of free space. In Fig. 1(b), because all hot data reside in the same block, after the updates to hot data, garbage collection can reclaim an entire block of free space without copying any valid data[1].

[1]Separating hot data from non-hot data has little effect on hybrid mapping [8], [9], because merge operations neutralize any efforts of data separation.

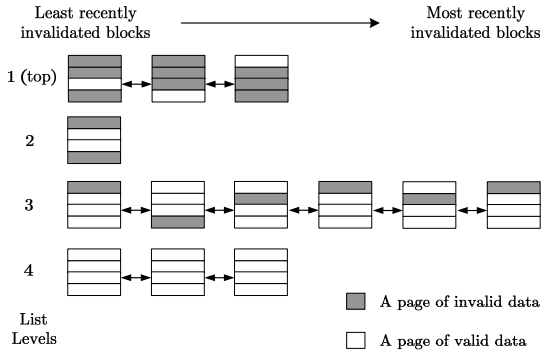

Fig. 2. The structure of the multi-LRI (least-recently invalidated) lists. A flash block has four pages in this example.

The victim-selection policy selects flash blocks as victims for garbage collection. A good victim selection policy should reduce the page-copy overhead. This policy should select flash blocks with small amounts of valid pages for overhead reduction in the short term. However, because hot data will soon become invalid, for overhead reduction in the long term, victim selection should not choose blocks having many hot (and valid) data even if these blocks are good candidates for short-term consideration.

B. Hot/non-Hot Separation

The flash-translation layer performs data separation before allocating flash space to data. The timings include processing write requests from the host and copying valid data during garbage collection. Kawaguchi [7] proposed writing newly written data and garbage-collected data to different flash blocks, because the valid data in victim blocks have remained alive for some periods of time and they are less hotter than the new data. However, this method ignores that hot data and non-hot data coexist in the new data. Chiang et al. [2] extended this concept to using multiple hotness levels of data. This approach, called Dynamic Data Clustering (i.e., DAC), logically partitions the flash space into regions. Each of the regions maintains a flash block for writing data and adopts a threshold for region elevation. A piece of data can be young or old with respect to the age threshold of its resident region. DAC elevates young data to higher-level regions on write, and demotes old data to lower-level regions on garbage collection. However, the performance of DAC is very sensitive to the selection of the total number of regions and the age thresholds of regions. It also requires prohibitively large RAM space for storing page-level age information.

Chang proposed identifying write requests smaller than a size threshold as carrying hot data [1]. However, this method overlooks that small write requests can also carry non-hot data. Hsieh et al. [4] proposed a frequency-based method of identifying hot data. This method uses a table of counters and a set of hash functions that map every logical sector number to several counters. A piece of data is considered hot if all the counters mapped to it are larger than a threshold. Im and Shin proposed the n-chance policy [5] that considers a class of warm data. These two methods are not adaptive because they have many parameters that require manual tuning for optimal

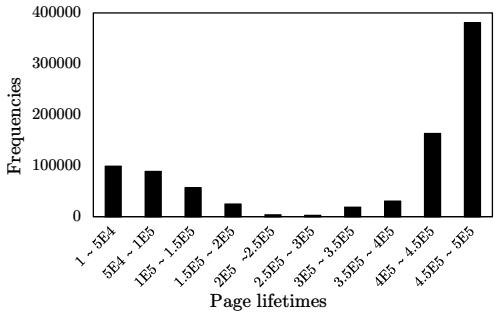

Fig. 3. The frequency distribution of page lifetimes produced by servicing 100,000 page-write requests of the disk workload from a Windows desktop.

performance.

C. Victim Selection

The well-known greedy policy [7] picks up a block of the largest amount of invalid data for garbage collection. However, realistic workloads have temporal localities of write, and even though a block is having the smallest amount of valid data among all blocks, this block may may continue to receive page invalidation. Thus, the greedy policy could result in premature erasure of flash blocks.

The cost-and-benefit policy proposed by Kawaguchi et al. [7] tries to avoid premature block erasure by giving lower priority of garbage collection to flash blocks that recently receive page invalidations. This method scores every flash block by a heuristic function $\frac{a\times(1-u)}{2u}$ and erases the most-scored block for reclaiming free space. Note that a and u in this function stand for the age and space utilization of a block, respectively. Chiang et al. proposed Cost Age Times (i.e., CAT) that takes both garbage collection and wear leveling into consideration using the function $\frac{u}{((1-u)\times a)\times t}$, where t is the block erase count [2]. CAT erases the least-scored block. However, both cost-and-benefit and CAT may need to re-score all blocks to select a victim of garbage collection. They suffer from poor scalability as the mainstream flash capacity has grown to several gigabytes when we wrote this paper [11].

III. Data Separation Policy

A. The Basic Data Structure: Multi-LRI Lists

The garbage-collection algorithm to be proposed is based on an essential data structure called multi-LRI (least-recently invalidated) lists. Figure 2 depicts the structure of the multi-LRI lists. This data structure has p parallel lists, where p is the total number of pages in a flash block. The elements of the lists are flash blocks. A flash block hooks on the level-i list if this block has i pages of valid data. Thus, the level-one list consists of flash blocks having exactly one page of valid data. If a flash block at level i receives a page invalidation, then this block is promoted to the (i-1)-th level list as the rightmost element (i.e., the list tail) at the new level. This way, the leftmost blocks (i.e., the list heads) are the least-recently invalidated blocks of every list. Note that the "level-0" list is not in the multi-LRI lists because blocks without any valid page data are no doubt the best candidates for garbage collection.

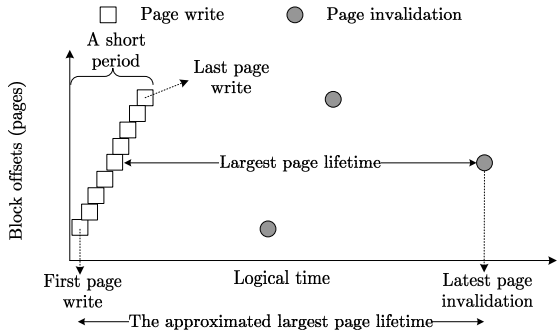

Fig. 4. Computing page lifetimes with block-level information. Each block stores the times of its first page write and its latest page invalidation.

B. Identifying Hot Data by Page Lifetimes

Realistic workloads have temporal localities of writing disk sectors. If a disk sector is recently modified by the host, then this sector will be modified for many times in the near future. Let the *lifetime* of a flash page (not a logical page) denote the total number of host write-requests arrived at the storage device during the period between writing new data to this page and invalidating (updating) data in this page. A valid flash page does not have a lifetime as its data are not invalidated yet. The rest of this paper adopts the total number of host write-requests arrived as the logical unit of time.

We collected the page lifetimes produced by servicing 100,000 page writes after a short warm-up of 400,000 page writes under the disk workload of a Windows desktop[2]. Note that page invalidations are irrelevant to flash management because they are host-level behaviors. Figure 3 shows that the frequency distribution of the collected page lifetimes is bimodal. In other words, the page lifetimes contributed by hot data are much shorter than that contributed by non-hot data. Thus, this study proposes using a lifetime threshold of identifying hot data. When a new page write arrives, it invalidates the latest copy of the page data and produces a page lifetime. If this page lifetime is shorter than the threshold, then the data of the new page write is recognized as hot data. The flash-translation layer should use two flash blocks and write hot data and non-hot data separately to these two blocks[3].

A technical question is then how to adaptively set the lifetime threshold for different types of workloads. Let the *top level* of the multi-LRI lists be the highest level which has at least one block. This study proposes using the largest page lifetime of the blocks at the top level as the lifetime threshold. Because hot data produce most of the invalid pages in flash, blocks having many hot data will reach the top-level list in a short period of time. Thus, the largest page lifetime of the top-level blocks is a good initial reference of data hotness. Provided that hot/non-hot separation is effective, there will be more blocks reach to an even higher level of the multi-LRI lists, and this change will further refine the age threshold.

[2]This is the PC workload in our experiment section.

[3]The flash-translation layer uses another flash block for writing garbage-collected data, because many of these data are non-hot or even immutable and they should not be mixed with new data.

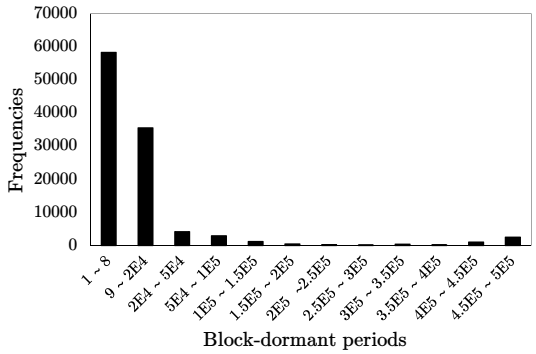

Fig. 5. The frequency distribution of block-dormant periods produced by servicing 100,000 page-write requests of the disk workload from a Windows desktop.

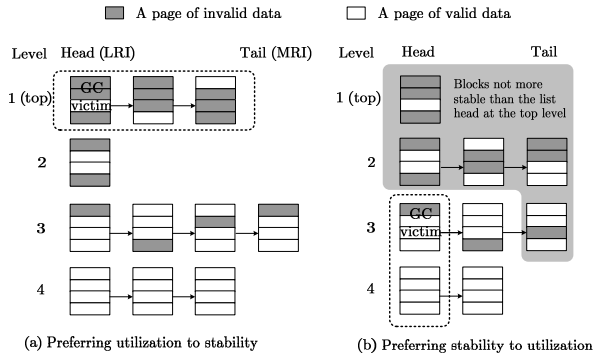

Fig. 6. The two modes of Dual Greedy. (a) The utilization mode. Dual Greedy selects the stablest one among the blocks having the lowest space utilization. (b) The stability mode. Dual Greedy selects the block whose space utilization is the lowest among the blocks that are stabler than the only top-level block.

C. Computing Page Lifetimes with Reduced Overheads

The proposed method of hot/non-hot separation requires checking page-level lifetime information. However, storing such all pages' lifetime information requires a large amount of RAM space. To save RAM space, this study proposes computing page lifetimes using block-level information. This technique is based on an observation that, since the flash-translation layer writes to the first page of a block, the rest of the pages in this block will all be written after a short period of time. Figure 4 shows a scenario of writing data to free space in a block. The flash-translation layer appends new data to the same block until this block runs out of free space. Thus, the time of writing the first page in a block can serve as an approximation of the times of writing data to the other pages in this block.

The flash-translation layer stores the first page-write time and the latest page-invalidation time of every block in RAM. The difference between these two times of a block represents the largest page lifetime of this block (as Fig. 4 shows). Upon garbage collection, the flash-translation layer updates the page-lifetime threshold with the maximum of the largest page lifetimes of blocks at the top-level list. Our current design checks only a fixed number (e.g., 8) of the least-recently invalidated blocks at the top-level list. This is because 1) less-recently invalidated blocks usually contribute large page lifetimes and 2) garbage collection frequently removes blocks from the top-level list for reclaiming free space and the total number of blocks at the top level is small during runtime.

Upon the arrival of a page write, the flash-translation layer first finds the page and block storing the latest version of this data. This page write produces a page lifetime that is equal to subtracting the found block's first page-write time from the current time. If this page lifetime is shorter than the threshold, then the new data are considered hot.

IV. Victim Selection Policy

A. Block Stability

Garbage collection selects blocks having low space utilizations for reducing its overhead in the short term. It must also avoid premature erasure of blocks for overhead reduction in the long term. Checking whether erasing a flash block is premature or not is not as straightforward as checking the space utilization of this block. In addition to space utilization, this study proposes using *block stability* as a reference for victim selection.

Observe the arrivals of page invalidations at flash blocks. Let the *dormant period* of a block be the interval between the current time and its latest page invalidation. Figure 5 shows the frequency distribution of the block dormant-periods produced by servicing 100,000 page-write requests. We recorded the lengths of block dormant-periods upon every page invalidation. This observation was made at the same time as that of Figure 3. The distribution shows that page invalidations produce a large number of short dormant periods. In other words, if a flash block has a short dormant period, then this block will receive another page invalidation sooner than a block of a long dormant period. Thus, block stability is a relative measure: blocks of long dormant periods are said to be more *stable* than blocks of short dormant periods.

B. The Dual-Greedy Strategy

This section introduces *Dual Greedy*, which is a strategy for victim selection that adaptively switches between utilization and stability. Dual Greedy has two modes, and its current mode is decided by how many blocks that the top level of the multi-LRI lists has. When the top-level list has more than one block, Dual Greedy will switch to the (space) utilization mode. As Fig 6(a) shows, Dual Greedy prefers utilization to stability in this mode, and chooses the stablest block from the top-level list as the victim. This block is the list head (i.e., the leftmost element) of the top-level list because this list head has the largest dormant period among the blocks at the top level.

As the demand for free space increases, garbage collection repeatedly removes blocks from the top-level list for erasure. Sometimes the top-level list has only one block left under high pressures of garbage collection. In this case, all flash blocks do not have long dormant periods, and thus Dual Greedy prefers stability to utilization for avoiding premature block erasure. It switches to the stability mode when there is only one block at the top level. In this mode, Dual Greedy selects the block whose utilization is the smallest among the blocks that are more stable than the only block at the top level. As Fig 6(b) shows, Dual Greedy rules out the blocks in the shadowed

Workload	Operating System	Volume Size	File System	Total Write
PC	Windows XP	40GB	NTFS	81GB
SEQ	Windows Mobile	20GB	FAT32	20GB
RND	Windows XP	16GB	NTFS	19.5GB

TABLE I
CHARACTERISTICS OF THE EXPERIMENTAL WORKLOADS

region because they are not more stable than the only top-level block. Dual Greedy selects the block whose utilization is the lowest among the list heads not in the shadowed region. This is because if the list head of a level is not more stable than the top-level block, then none of the blocks at this level would be. In the worst case that no list heads are more stable than the top-level block, then Dual Greedy selects the only top-level block as the victim.

The proposed victim selection strategy requires blocks' latest page-invalidation times to compute block dormant periods. The data separation policy also utilizes this information and can share it with the victim-selection policy.

V. EXPERIMENTAL RESULTS

A. Experimental Setup and Performance Metrics

We have built a simulator for performance evaluation. This simulator implements representative designs of flash-translation layers: 1) FAST [8], which is based on hybrid mapping, 2) page-level mapping with the greedy policy [7] (PL+greedy), 3) DAC [2], which is based on page-level mapping and uses the cost-age-time policy (CAT) for garbage collection, 4) SuperBlock [6], which uses page-level mapping inside of groups of flash blocks, and 5) Dual Greedy. Notice that PL+greedy uses separated blocks for writing new data and garbage-collected data. DAC and SuperBlock adopted their best parameter settings found by off-line exhaustive search. DAC ignored wear leveling for its best performance of garbage collection.

There are three types of workloads in our experiments. The first workload was collected from a desktop PC running Windows XP whose file system was NTFS (i.e., the PC workload), the second workload is generated from a portable media player (i.e., the SEQ workload) that repeatedly wrote large files. The last one (i.e., the RND workload) is obtained from the industrial-standard benchmark tool Iometer with a 4KB request size and 100% random write. Table I is a summary of these workloads. This study adopts the total erase count as the primary performance metric of garbage collection.

B. Host Workloads

This experiment adopts the geometry of a typical NAND flash [11] whose block size and page size are 512 KB and 4 KB, respectively. Let the over-provisioning ratio be the fraction of flash space provided as spare space. For example, if the logical disk is 40 GB, then with a 2.5% over-provision ratio the flash size is $40\times(1+2.5\%)$=41 GB.

Figure 7(a) shows the results under the PC workload. This type of workload has many temporal localities of write. Dual Greedy greatly outperformed FAST, and this advantage is obvious especially when the over-provision ratio was large. This is because large spare size improves the effectiveness of hot/non-hot separation. In contrast, the merge-based FAST did not much benefit from using large spare space. Compared to SuperBlock, Dual Greedy does not confine page-level mapping to small block groups, and thus Dual Greedy has more flexibility of separating hot data from non-hot data. Dual Greedy also surpassed PL+greedy because the proposed policies for data separation and victim selection performed better than the primitive counterparts in PL+greedy.

The experiment for the SEQ workload used smaller over-provision ratios to increase the pressure of garbage collection. Figure 7(b) shows that, benefited from switch merges, FAST delivered better performance than PL+greedy when the spare size was very small. However, this advantage quickly diminished as the over-provision ratio increased, because PL+greedy could always find blocks having no valid data for erasure.

Figure 7(c) shows the results under the RND workload. SuperBlock suffered from log-block thrashing under this random-write pattern even when the flash spare size was very large. FAST performed poorly when flash spare size was small because of its extremely large associativity of log blocks. Adding more flash spare space effectively relieved FAST of this problem. DAC and Dual Greedy performed as good as PL+greedy when the spare size was large, but neither of them can outperform PL+greedy. This is because the RND workload has a purely random write pattern. Any predictions on write behaviors based on locality will not be useful. Thus, a flash-translation layer could switch to the greedy policy when it detects that the host is accessing the solid-state disk with a purely random write pattern.

Recall that DAC requires considerable resources of time and space, and it needs manual parameter-tuning. Dual Greedy used very limited resources, but achieved the same or even better performance than DAC. Later sections will provide more discussion of resource requirements.

C. Flash Geometry

It is important to examine the performance of Dual Greedy with coarse-grained flash geometry because parallel flash structures such as planes, gangs, and interleaving groups would effectively increase the flash block size. Garbage collection becomes more difficult when the block size is large because erasing a large flash block could involve more data copying. This experiment adopted two geometry settings whose page sizes and block sizes were 4 KB/512 KB and 8 KB/1 MB, respectively. Because these two settings have different block sizes, the Y-axis of Fig. 7(d) indicates the total numbers of bytes erased from flash. The workload was the PC workload, and the over-provisioning ratio was 5%. Figure 7(d) shows that Dual Greedy had the smallest increase (i.e., 1.3 times) on the total byte erased when switching to large blocks.

D. The Needs for Adaptiveness and Overhead Analysis

A design goal of Dual Greedy is to eliminate the needs for static parameters. This experiment takes DAC as an example

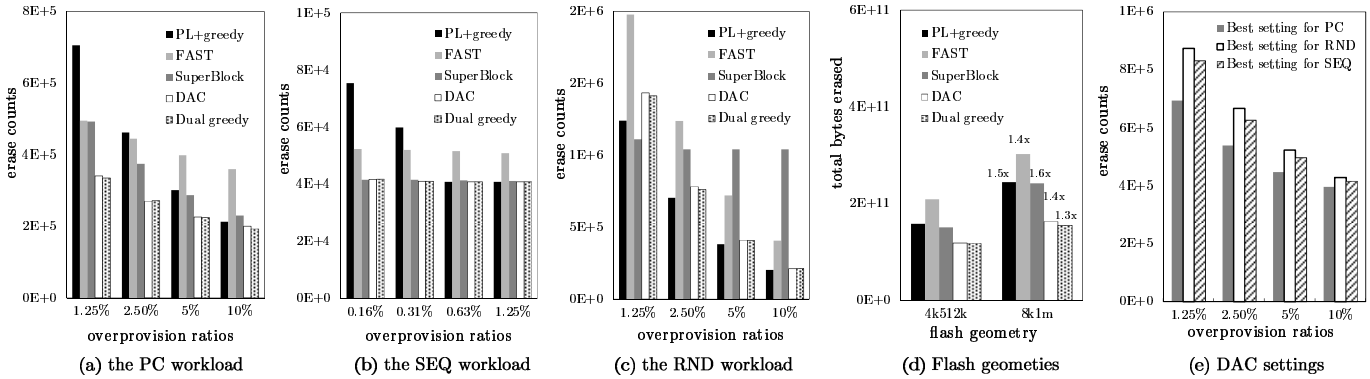

Fig. 7. Experimental results. (a)-(c) Results of using different host workloads. Note that the three Y-axis scales are different. Dual Greedy greatly outperformed SuperBlock, FAST, and page-level mapping with Greedy policy. Dual Greedy also performed as good as the resource-consuming DAC. (d) Results of using different geometry settings. Dual Greedy incurred the smallest overhead increase when switching to a large flash block size (e) The needs for adaptive parameter tuning. Under the PC workload, DAC performed poorly using the best static parameter settings for the SEQ or RND workload.

to show the needs for adaptive garbage collection. Recall that DAC requires several parameters including the total number of regions and the per-region age-threshold for elevating and demoting page data between regions. This experiment first found the best parameter settings for the three workloads, and then evaluated DAC with these three sets of parameter values under the PC workload. Figure 7(e) shows that, DAC performed very differently with these three parameter-value sets. In particular, under the PC workload DAC performed poorly with the best setting of the RND workload.

Resource efficiency is also important to the design of Dual Greedy. As mentioned in Section III-C, Dual Greedy updates the page-lifetime threshold by checking the largest page lifetimes of eight blocks at the top-level list. For victim selection, the total number of flash blocks that Dual Greedy examines is not larger than the total number of pages in a block, as described in Section IV-B. Thus, Dual Greedy takes constant time to pick up a victim for garbage collection. In contrast, DAC could require re-scoring all flash blocks using its cost-age-time function to find a victim.

As to RAM space requirements, Dual Greedy stores only two time stamps for every flash blocks. Consider the experiment with the PC workload. The flash size is 41 GB, the block size is 512 KB, and the page size is 4 KB. Suppose that every time stamp is of 4 bytes. Storing these time stamps requires only $((41\times2^{30})/(512\times2^{10}))\times2\times4$=656 KB of RAM space. In contrast, DAC stores a time stamp for every flash page, and storing these page time stamps requires $((41\times2^{30})/(4\times2^{10}))\times4$ = 41 MB.

VI. Conclusion

As flash capacity dramatically increased in the past decade, many solid-state disks adopted hybrid mapping for good balance between write performance and mapping-table size. Recently, high-end solid-state disks start adopting powerful controllers and implement page-level mapping because of the strong demand for random-write performance.

This study investigates an adaptive and resource-efficient garbage collection policy for page-mapping solid-state disks. The design of the proposed method is based on two key observations. After servicing a large number of write requests, the produced page lifetimes have a bimodal frequency distribution, while the produced block-dormant periods have a long-tail frequency distribution. Based on these findings, the proposed approach, named Dual Greedy, uses only block-level information for page-accurate identification of hot data, and it utilizes multilevel lists of blocks for adaptive victim selection. Dual Greedy can run in constant time with limited space, and it is workload-adaptive. Our experimental results show that Dual Greedy outperforms many prior approaches in terms of garbage-collection efficiency, resource requirements, or both, especially when the flash block size is large.

References

[1] L.-P. Chang. A hybrid approach to nand-flash-based solid-state disks. *Computers, IEEE Transactions on*, 59(10):1337 –1349, oct. 2010.

[2] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using data clustering to improve cleaning performance for flash memory. *SoftwarePractice and Experience*, 29:267–290, 1999.

[3] A. Gupta, Y. Kim, and B. Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. *SIGPLAN Not.*, 44:229–240, March 2009.

[4] J.-W. Hsieh, L.-P. Chang, and T.-W. Kuo. Efficient on-line identification of hot data for flash-memory management. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 838–842, 2005.

[5] S. Im and D. Shin. Comboftl: Improving performance and lifespan of mlc flash memory using slc flash buffer. *Journal of Systems Architecture*, 56(12):641 – 653, 2010.

[6] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. Superblock ftl: A superblock-based flash translation layer with a hybrid address translation scheme. *ACM Trans. Embed. Comput. Syst.*, 9:40:1–40:41, April 2010.

[7] A. Kawaguchi, S. Nishioka, and H. Motoda. A flash-memory based file system. In *Proceedings of the USENIX 1995 Technical Conference Proceedings*, TCON'95, pages 13–13, 1995.

[8] 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.*, 6, July 2007.

[9] 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.*, 7:38:1–38:23, August 2008.

[10] 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 *Proceedings of the 2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '11, pages 157–166, 2011.

[11] Samsung Electronics Company. *K9MDG08U5M 4G * 8 Bit MLC NAND Flash Memory Data Sheet*, 2008.