Chapter 1

Introduction

The out-of-place update behavior of flash memory is derived from the compromise of its innate physical constrains, such as erase-before-write, and the large size gap between write and erase unit. This operation needs to be supported by a flash translation layer (FTL) which keeps mapping information between logical sectors and physical memory locations and also periodically reclaims space occupied by outdated data by means of erasure.

FTL is implemented in the firmware, and its mapping resolution has direct impact on random-write performance and the mapping table size. A typical design of solid-state disk (SSD) can offer a few tens of kilobytes for storing the mapping table. For example, GP5086 from Global Uni-Chip has only 32 KB as table memory. In the past decade, hybrid mapping is a popular approach for good balance between random-write performance and table sizes. It restricts the page-level mapping scope for log blocks only and using block-level mapping for data blocks. However, hybrid mapping needs to perform merging of data in log blocks back into block-level data blocks to produce free space. And the overhead of merge can grow extremely high under random access pattern. Some enterprise-level SSDs start adopting high-profile hardware specifications. Under this kind of environment with abundant resource, page-level mapping has become feasible.

Although the page-level mapping scheme is implemented by plenty of recent SSD products, they are still using very primitive garbage collection policies. The bottleneck of evolution resides in the difficulty to scale the structure to gigabytes of flash memory. For example, many of the prior approaches require to store page-level timestamps of data and to compute block-level heuristic values for garbage collection. Also some of them need

the intervention from human to tune a collection of parameters for the optimal performance. However, using the same parameter settings under various host workloads could unexpectedly incur high overheads for garbage collection.

This paper has proposed a novel garbage-collection algorithm for page-level mapping based SSDs. There are two major design challenges. First, even though this study targets at high-end SSDs, since the mainstream flash capacity dramatically increases recently, the time and space overheads of the proposed method still need to be very low. Second, the proposed method should be adaptive to various conditions which has great impact on the performance of garbage collection, including the host write pattern, flash geometry, and flash spare size. Third, the proposed method should at least perform as good as or even better than the previously proposed algorithms that require large amounts of resources and manual parameter tuning.

The following of this paper is organized as followed: chapter 2, an abbreviation of important issues for designing GC policy for page-level mapping FTLs and related works; chapter 3, our proposed data separation policy, involving cost down on memory requirement; chapter 4, proposed victim selection policy which strengthen the greedy GC policy to dual greedy policy; chapter 5, performance evaluation of dual greedy policy comparison against existing algorithms and chapter 6 conclusion.

Chapter 2

Background

2.1 page-level mapping and garbage collection

Data mapping emulates a more convenient interface for operating system by neutralizing flash memory's physical limits for example, explicit erase before write and the large size gap of write and erase units. There different mapping policies regarding different abundances of resource and expectations of performance. Page-level mapping uses page mapping table to keep track of mapping information of every logical page. This table translates the logical sector number of requests into corresponding physical page number therefore the merge operation which is originally implemented for re-ordering those non-mapped data in data block is dismissed.

There are two important issues for designing a GC policy for page-level mapping FTLs: data placement policies and victim selection policies. Data with high re-referenced possibility is called *hot data*, and those remains intact for a long time are *non-hot data*. The job of data placement policy is to separate those two. A good data placement policy can successfully gather invalid spaces and create good GC targets, therefore enables efficient free space reclaiming with less erase counts. Victim selection policy is responsible for recognizing those good GC targets. Traditional greedy GC policy concerns only the space utilization of GC victim, nevertheless, those blocks tend to have hot valid data. Because they are erased earlier before hot data are fully invalidated, not only the copies of hot valid data are wasted, but also the GC efficiency is decreased. Therefore, we need to consider both the utilization of GC candidates and their invalidation recency, because those who are

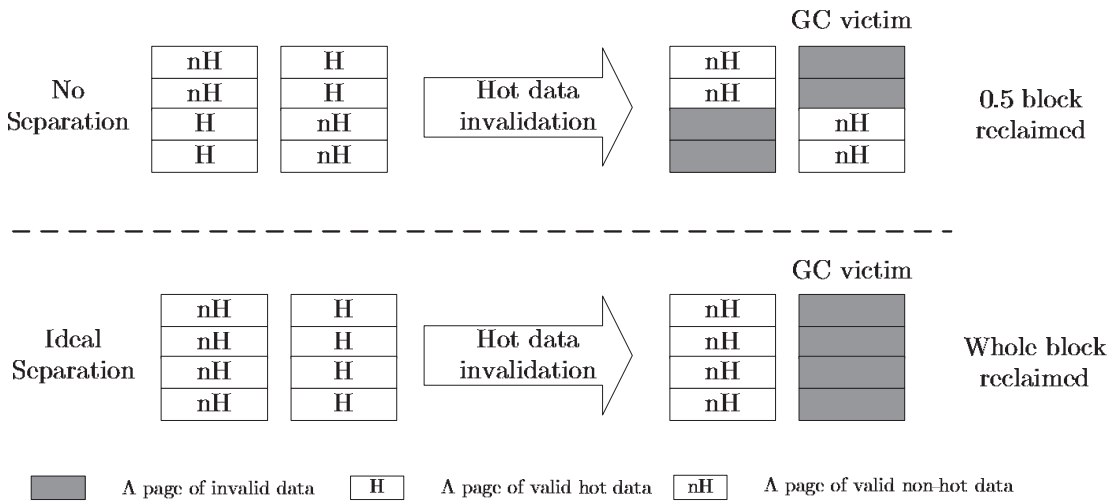

Figure 2.1: Separating hot data and non-hot data in flash blocks greatly improves garbage-collection performance

updated recently will be accessed soon based on temperal locality and create more invalid space. Consequently, it is not appropriate to choose them as GC victim.

2.2 Data-Separation and Garbage Collection

The purpose of data separation is to cluster data with similar update frequencies into the same block. Those blocks which stores lots of hot data will produce good GC victims with massive amount of invalid space ,therefore GC overhead is reduced. Figure 2.1 illustrates the effects of 2 different cases. Ideal separation policy creates a fully invalidated victim and reclaim a full block space with no extra valid data copy which is much more efficient than the case in which separation is not considered. [1]

In separate segment cleaning policy proposed by Kawaguchi et al. [1], they used 2 different blocks, log block and GC block, to respectively store written pages and collect valid pages in GC victim. Since data copied from GC victim is recognized as non-hot, GC blocks implicitly realizes data separation by copying relatively cold valid pages to a separated GC block. However, data is again mixed when re-written into log block.

Chiang et al. proposed DAC[3] that adopts multiple regions and allocates one log block for each of them. It uses different threshold in terms of the length of data update periods to

[1]Compared to page-level mapping FTL, data separation is almost negligible in hybrid-mapping FTLs, such as FAST[4] since prior data placement will be neutralized after merge operation.

promote and demote logical pages. A piece of data elevated to the next higher-level region if its update period is shorter than the promotion threshold, and degraded during GC if its update period is longer than the threshold. However, the performance of DAC would be sensitive to the selection of region number and threshold. Our later experimental results will show that inadequate settings of those parameters will results in severe performance degradation.

Hsieh et al. proposed Bloom filters [2] that uses a table of counters. It hashes the logical addresses of data to multiple counters with several hash functions. Whenever a logical page is updated, its corresponding counters are incremented. Those data whose hashed counters are all larger than a threshold are recognized as hot data and would be written into a separated block. However, like DAC, this approach requires manual intervention for optimum performance.

For 2-level lists [7] which is proposed by Chang and Kuo and n-chance policy [8]that proposed by Im and Shin, though less storage cost demanded, the size of list and the value of n are both unresolved parameters which need manual intervention. And for the length-filter method[10] proposed by Chang, it identifies small write requests as update to hot data. However among the data written by small requests may exist part of cold data, therefore causes possible misjudgement.

The prior data-separation policies mentioned are not directly applicable to real solid-state disks because they suffer from two common problems. First, manual-tuned parameters and second, unaffordable amount of resource consumption. The following sections introduce an efficient data-separation policy that addresses these two problems.

2.3 Victim Selection and Garbage Collection

A good victim selection policy should minimize the page-copy overhead of garbage collection. In the short term, this policy should select flash blocks with small amounts of valid pages for erasure. In the long term, the policy should avoid selecting flash blocks whose valid page could become invalid in the near future. Except for data separation policy, victim selection is another issue for designing page-level mapping FTLs. Though data separation has successfully produce numerous good GC candidates, GC process still needs to be smart enough to recognize them to bring good overall performance.

GC policy	Victim selection	Drawbacks
Greedy	Choose the block with least valid pages	Premature GC
Cost-benefit	a*(1-u)/2u, choose the block with the highest score	Computing overhead
CAT	u/((1-u)*a)*t, choose the block with the lowest score	Computing overhead

Table 2.1: Prior GC policy for page-level mapping FTL

In [1], the greedy policy considers only space utilization but ignores temporal localities of writes, causing premature erasure of flash blocks. *Premature GC* is the erasing of a block with hot data. Those hot data is very possible to be invalidated soon and creates better GC victims. Premature GC not only wastes time on copying of hot valid data but also causes hot data re-scattering.

For cost-benefit [1] and CAT [3]. They require re-computing of the scores of all the flash blocks and thus is infeasible in large scale SSDs. Table 1 is the formulations and brief summarization of the above 3 victim selection policies, in which a, u and t stands for block age, block utilization and erase count respectively.

By summarizing the above, we have targeted at designing a low overhead victim selection policy which evenly evaluates GC candidates by both their block space utilization and probability of premature GC, furthermore, the policy should guarantee adaptivity under various access patterns.

Chapter 3

Data Separation Policy

3.1 The Basic Data Structure for Garbage Collection

By summarizing the above, in order to efficiently select good GC victim, there are 2 factors need to be monitored: space utilization and invalidation recency.We have designed a special data structure called *multi-LRI(Least Recently Invalidated) lists* to smartly manage GC candidates by these 2 indexes. Its arrangement is demonstrated by figure 3.1 . In our proposed algorithm, all the full blocks are manage with multil-LRI lists structure for more efficient GC victim selection. Every level is composed by blocks with the same amount of valid pages. The latest invalidated block is linked to the tail of its corresponding level. Therefore, tail block always has the highest recency within its level. The highest level which is formed by blocks with lowest space utilization is the top level. Under this organization, recency sorting is achieved horizontally and utilization sorting is managed vertically, and a fast selection process of GC victim is therefore realized.

3.2 Identifying Hot Data and Non-hot Data

The page mapping table is used to keep mapping information of logical and physical pages. Logical pages are requests delivered from operating system which shows obvious temporal locality and links tightly to the idea of data temperature. However with the physical constrain and special management of flash memory, the data of the same logical page may be updated to different physical pages. We have designed a data-separation policy based

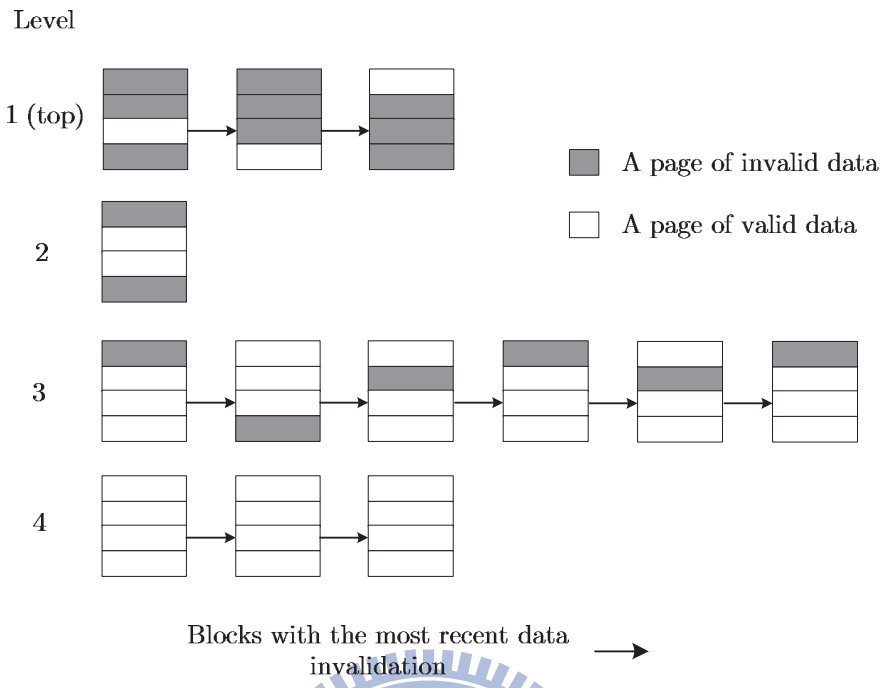

Figure 3.1: Multi-LRI list structure

on the observation of lifetime distribution of logical pages. Let *page lifetime* denotes the period between write and invalidation to a logical page. A logical page may have multiple lifetimes with respect to different corresponding physical pages. However, it assigns lifetime only to invalid physical pages since a valid physical page has not been invalidated yet, therefore has no lifetime or an infinite lifetime. Figure 3.2 demonstrates how a piece of data is moving among several blocks and creates different lifetimes. Lifetime is a good clue for distinguishing the data temperature since its an observation of accessing behavior to logical pages which is related strongly to temporal locality. A trace with moderate locality usually has distinctive hot data out of a pile of non-hot data, hence, it is believed that with an appropriate selection of lifetime threshold, data separation can be implemented.

Figure 3.3 represents an observation of written data's lifetime throughout 100,000 requests after a short warm-up of 400,000 requests. The distribution is bimodal and there is a clear cut between those frequently updated data (i.e., hot data, lifetime ¡=200,000) and the other data (i.e., non-hot data). Thus, our idea is to use a lifetime threshold to separate hot data and non-hot data and write these two kinds of data to separate flash blocks, hot and non-hot log blocks. This result simplifies the tuning of regions numbers in DAC since two-region structure is sufficient to illustrate the bimodal distribution and can

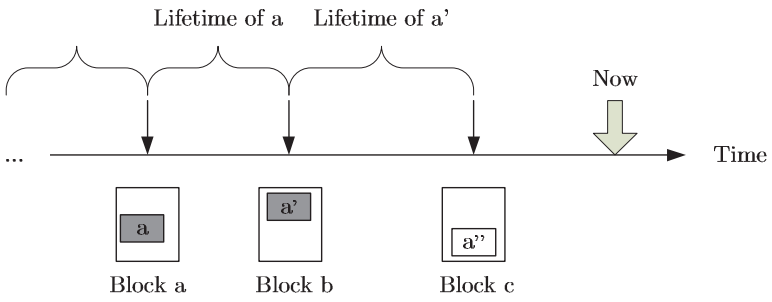

Figure 3.2: Definition of flash page lifetime

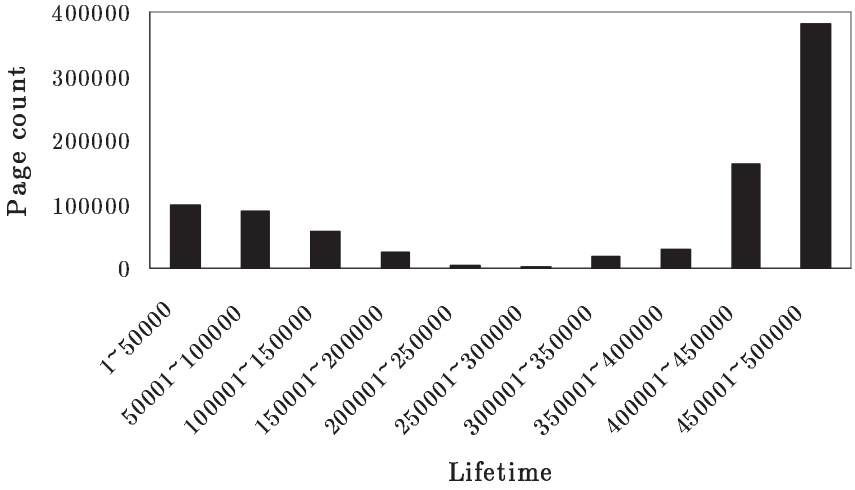

Figure 3.3: The distribution of page lifetimes after processing 100,000 write requests under the disk workload of Windows XP

large decrease the design complexity.

We therefore use a lifetime as the threshold to separate these 2 groups of data, and the selection of this value will be achieved by the support of multi-LRI data structure. Blocks with massive amount of garbage will be automatically filtered to highest level, and those blocks are considered hotter because with more hot data stored, blocks can generate more and faster invalidation. Therefore, under the 2-region separation structure, the number of invalid page is an evidence of its hotness. Those invalid pages in the top record lifetimes left by hot data, so it gives very good advice for deciding the lifetime threshold. We adopt the maximum lifetime in top level as threshold for separation because it involves less computation overhead and can conservatively includes all the pages which are as hot as those in the top level into hot log block.

3.3 Page Lifetime Computation

Identifying hot data and non-hot data requires to check a page's lifetime upon update. However, storing lifetime information at the page level imposes a large RAM-space overhead on flash management.

To save RAM space, this study proposes storing of the lifetime information at the block level. Figure 3.4 shows the timeline of page writes in a flash block.In figure 3.4, a quick-filling phenomenon is noticed. A block is usually intensively written as a log or GC block until it is full, and the page time stamps are kept only to differentiate those compact written times which produces lifetimes very close to each other. Consequently, those written time can be summarized by an uniform block time stamp called first page-write time. With this time stamp and the other latest page-invalidation time stamp, the maximum page lifetime of a block can be determined.

For realizing adaptive lifetime threshold, each block needs to store only two time stamps: the first page-write time and the latest page-invalidation time, and the threshold is simply refined upon GC. Because we are using the maximum lifetime in the top level as the separation threshold, there is no need to compute every lifetime within the list. First the maximum lifetime of every block is computed locally, and then the global maximum lifetime of top level can be decided. By this flow, the computation overhead can be largely eased. The maximum lifetime computation of individual block can also be further simplified by using an uniform first page-write time instead of distinctive real page-write time since even the maximum inaccuracy caused by this substitution is still small and negligible as the short writing period indicated in figure 3.4 and the storage requirements are distinctively lower than using page-level time stamps.

The above process seems to involve many blocks. Nevertheless, as free space is exhausted, GC grows more aggressive and the pace of invalidation gradually falls behind, the size of top level list will shrink considerably. Therefore the computation of top max lifetime merely includes limited number of blocks. In fact, in our experiment, since the all top level data are signatured with its hotness, a few top level blocks is enough to conclude a good max lifetime as threshold. With the above scheme, data separation has become realistic and affordable.

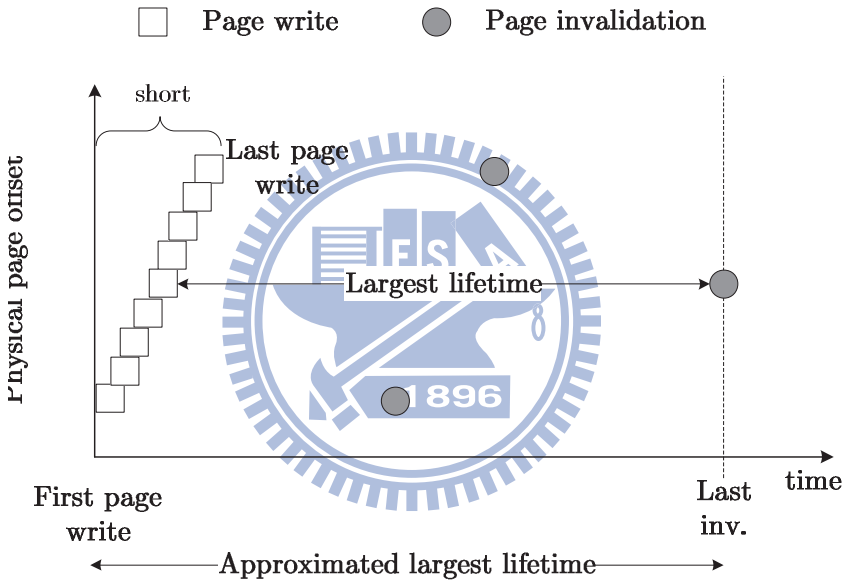

Figure 3.4: Page lifetime approximation

Chapter 4

Victim Selection Policy

4.1 Block Stability

The garbage collection is the process for reclaiming space occupied by invalidated data when free space is exhausted. Based on the proposed separation policy, we are able to create some favorable candidates for garbage collection. However, the conventional greedy policy is straightforward and only aims at minimizing the garbage-collection overhead in the short term which neglects the consequence brought by premature GC. This section presents a strategy to avoid prematurely erasing a flash block, even if this block is a good short-term candidate.

Let a block's *inactive period* indicates the interval between current time and its latest page invalidation, and it is measured by the request number. Inactive period implicitly explains the data temperature of a block. Figure 4.1 is an observation of invactive period of updated blocks throughout 100,000 requests. In this figure, a majority of updated blocks have relatively small inactive period which implies that they are updated recently, and according to temporal locality, they are very possible to be updated again soon. Those blocks are still carrying hot data and they are too unstable to recycle since it will result in premature GC. On the contrary, blocks with longer inactive period has little chance to be updated because their hot data is almost fully invalidated. In conclusion, a flash block is more *stable* than other blocks if its inactive period is longer than that of other blocks. In other words, block stability is a relative measure. A technical question is how to choose stable blocks among short-term candidates. This will be addressed in the next section.

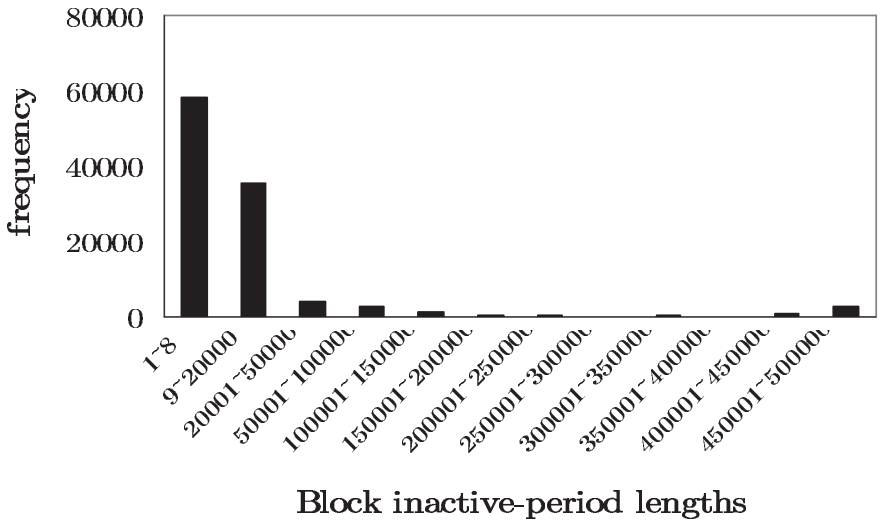

Figure 4.1: The length distribution of block stable periods after processing 100,000 write requests

4.2 The Dual Greedy Strategy

We have proposed the *dual greedy strategy* which considers not only space utilization but also stability for implementing greedy policy. There are 2 different cases in dual greedy policy with respect to the number of blocks in top level. At first GC process consecutively selects the stablest block as victim from the top level when number of blocks in top list is more than one. As depicted in case (1) of figure 4.2, head block of top list, namely the stablest one among all the blocks with lowest utilization is selected as GC victim. However, as free space becomes less and garbage collection also becomes aggressive, top list will shrink rapidly and gradually become too unstable that may cause premature GC. When there's only 1 block left in the top, it will trade low GC overhead for stability to recycle a stabler block.

In case (2) of figure 4.2, blocks which are less or evenly stable compared to top list head is surrounded by the shadow area. Those blocks are not stable enough to be recycled, thus dual greedy policy is constrained to those stabler blocks, and the stablest one with lowest utilization among them is chosen as victim. If the head of top list is stable enough, that is, there is no block stabler than top list's head, the whole strategy will degrade to traditional greedy policy by selecting the top list's head, the stablest blocks within the structure, as GC victim.

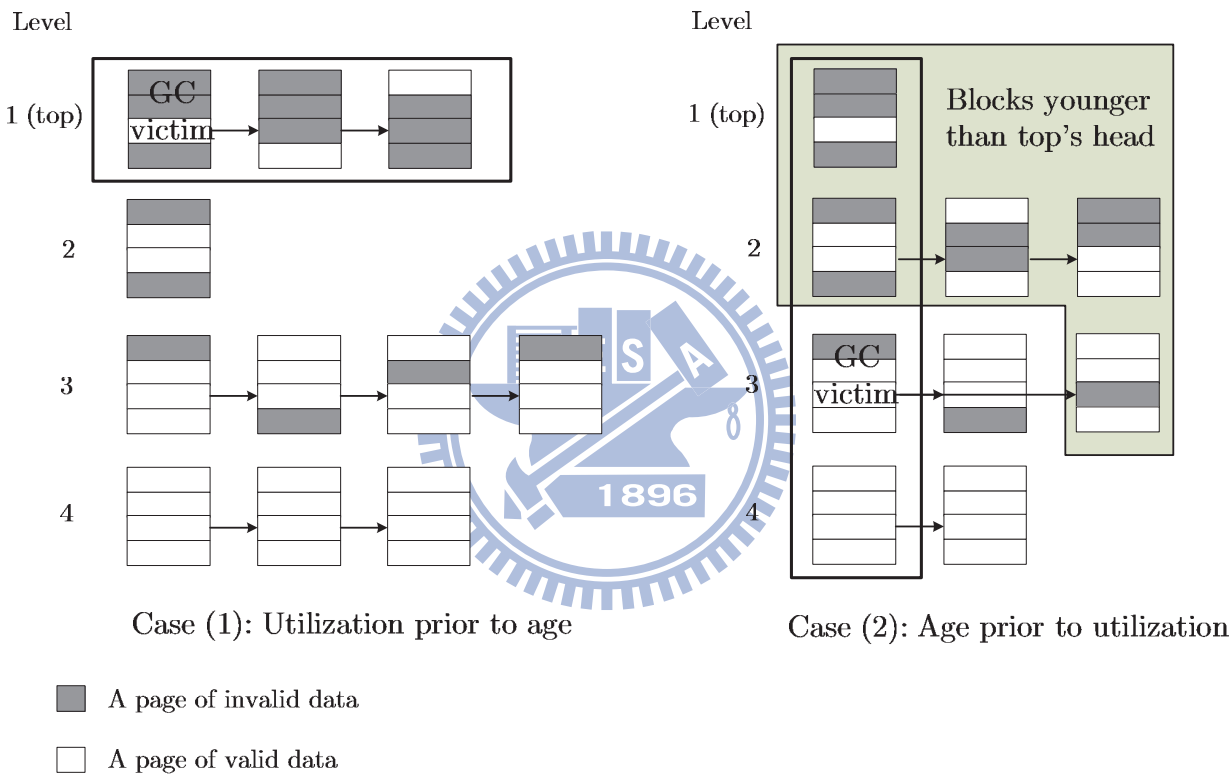

Figure 4.2: GC victim selection under dual greedy policy

Chapter 5

Experimental Results

5.1 Setup and Metrics

We have built a simulator based on C for performance evaluation. this simulator implements five state-of-the-art FTL designs: 1) FAST [4], which is based on hybrid mapping, 2) page-level mapping with the greedy policy [1] (i.e., PL+greedy), which adopts separate blocks for logging and garbage collecting, 3) DAC [3], which is based on page-level mapping and uses the cost-age-time policy (i.e., CAT) for garbage collection, 4) SuperBlock [11], which uses fully page-level mapping at the block level, and 5) the proposed Dual Greedy. For some of the algorithms with parameters, we have adopted the best setting acquired from off-line tuning.

This study consider three types of host workloads. The first workload was collected from a desktop PC running Windows XP whose file system was NTFS (i.e., PC trace), the second workload is generated from a portable media player (i.e., SEQ trace) which performs massive amount of sequential requests. The last one is obtained from a benchmark which produces adjustable degree of random access behavior. In this experiment, we use the 100 percent random trace created by IOmeter to signature the purely random access behavior (i.e., RND trace). The detail of workload is illustrated in table 5.1.

Because the primary objective of garbage collection is to reduce the overhead of block erase, this study adopts the total erase count as the major performance metric in the experiment. Unless explicitly specified, the experiment adopted the following settings: the page size and block size are 4KB and 512KB, respectively, the overprovisioning ratio was

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

Table 5.1: The 3 experimental workloads

1.25, 2.5, 5 and 10 percent. As for SEQ trace, in order to observe obvious degradation caused by overprovisioning ration, we have tried extra 3 smaller size: 0.16, 0.31 and 0.63 percent.

5.2 Host Workloads

Figure 5.1(a) shows the results under the PC workload. This type of workload has many temporal localities of write. Dual Greedy and DAC[3] greatly outperformed FAST[4] and SuperBlock[11] even when the over-provisioning ratio was small. All flash-translation layers except FAST[4] are significantly benefited from high ratios of over-provisioning. This is because separating hot data and non-hot data at the page level requires more flash spare space to take effect. On the other hand, the merge-based approach, i.e., FAST[4], was not much benefited from large spare space because its overhead is related to the associativity of the victim block. Therefore large spare space did not help decrease the average number of data blocks needs to be merged for erasing a victim block.

In (b) of figure 5.1, under the SEQ workload, because lots of consecutive invalidations were made by this access pattern, even with small amount of free space, it was sufficient to yield good performance. Therefore not only merge-based but also page-level mapping FTLs were not sensitive to over-provisioning ratios. Nevertheless, PL+greedy[1] had inferior performance with a very small over-provision ratio as its data separation policy is not very effective.

Part (c) of figure 5.1 represents the performance under pure random access pattern,

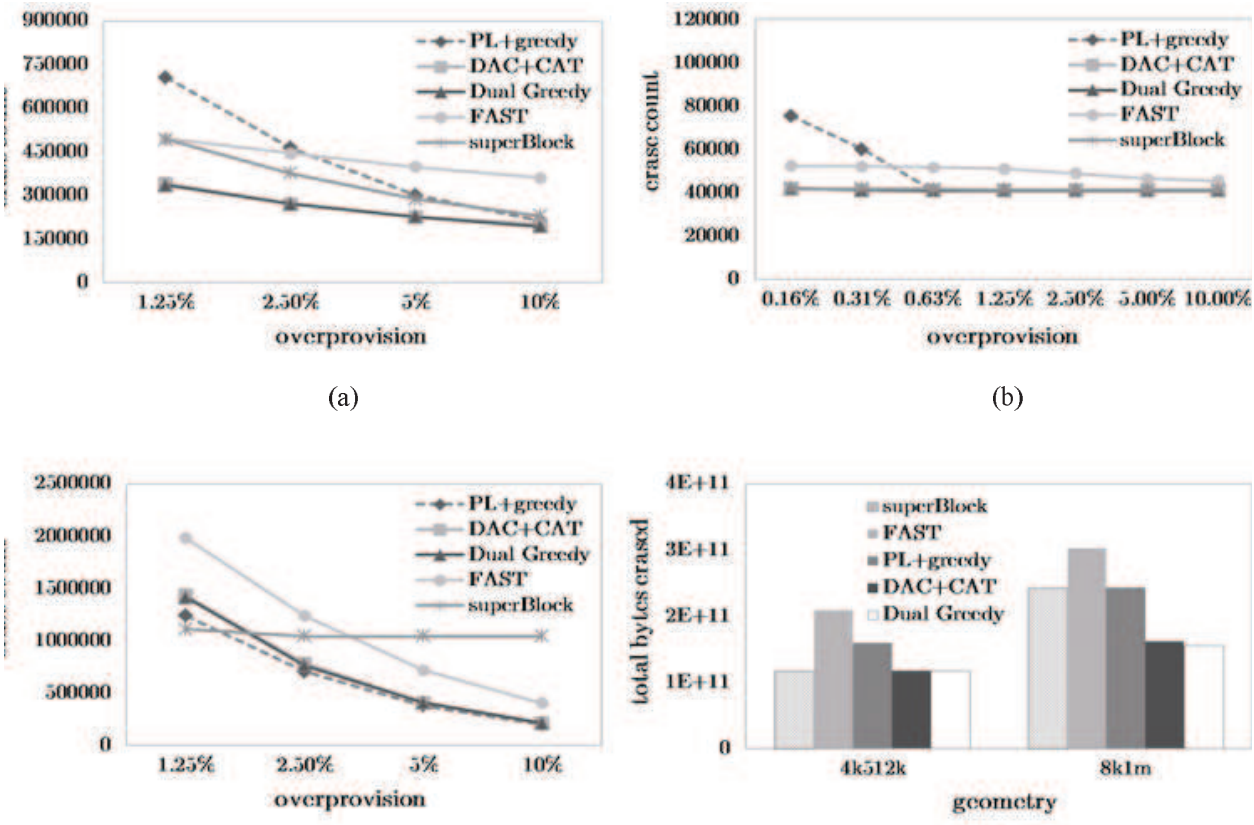

Figure 5.1: Experimental results under (a)-(c) different host workloads and (d) different geometry settings.

RND. For superBlock[11], over-provision was never enough for resisting sever thrashing of log blocks under such a random write pattern. For FAST[4], though it was immune to thrashing, the associativity of victim block was unacceptably high. For those designs based on page-level mapping, i.e., Dual Greedy, DAC[3], and PL+greedy[1], they significantly outperformed the other approaches because high-resolution mapping is very beneficial to performance under such a random write pattern. However, PL+greedy[1] slightly outperformed Dual Greedy and DAC[3] when the over-provisioning ratio was small. This is because the write pattern is totally unpredictable (i.e., no temporal localities). Therefore their policies for data separation and victim selection did not deliver obvious performance improvement under this case.

FTL	Memory Cost	Manual-tuned parameters	Performance (normalized to DAC)
DAC	WinXP: 240M MM: 95M IOmeter: 92M	Region number Promote / Demote TH	-
Dual Greedy	WinXP: 6.4M MM: 2.73M IOmeter: 2.375M	None	WinXP: 96.4%~100.2% MM: 100%~100.3% IOmeter: 97.1%~100.4%

Table 5.2: Comparison between DAC and Dual Greedy

5.3 Page-Level Mapping vs. Hybrid Mapping: Pattern Switch

The part is to exploit the stability of FTLs. We have compared Dual Greedy with merge-based FAST under an intentionally create access pattern. The page size is 4KB, block size is 512KB, disk size is 16GB and 5 percent of overprovision is given. First, the disk is written with random trace generated by IOmeter benchmark. Then, after random trace finished, we create access pattern writing from the begining of the disk and write consecutively to the end with each request of the same lengh 8, a page size. We monitor interval page write count after processing every 128 requests. The interval write count can stand for its overhead. Figure 5.2 shows the result. The dotted line indicates the timing of pattern switch. During random trace, dual greedy obviously has much less overhead against FAST. This is because for maintaining the in-order arrangement in data blocks, FAST needs merge operation to re-arrange data during GC. However, under random access pattern, its log blocks usually have high associativily which leads to the merge with lots of data block within a single GC operation and results in high overhead. But FAST responds rapidly to pattern switch with the support of sequential block while dual greedy shows the effect after a lot of sequential patterns are written. It is because sequential trace produces very even invalidations to a disk full of random access patterns. Its needs massive sequential patterns to create noticeable amount of invalid space for a single block to become good GC

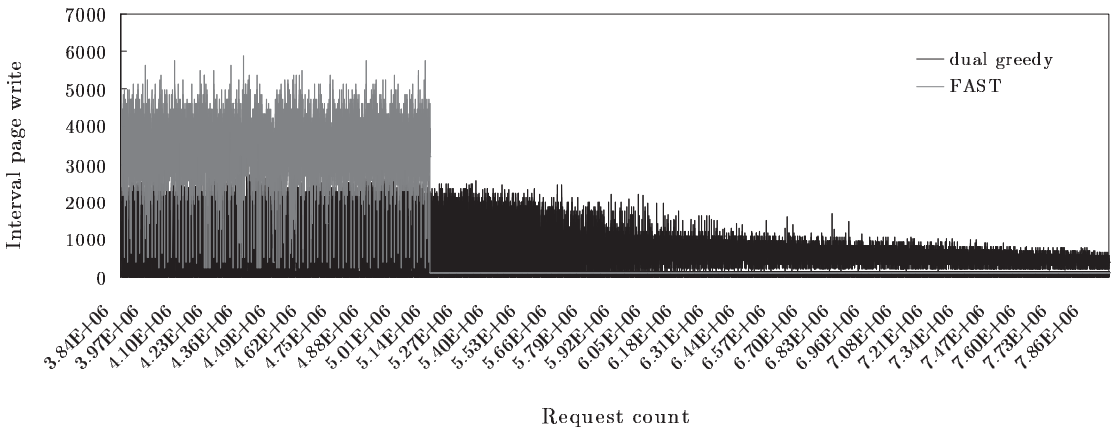

Figure 5.2: Observation under pattern switch

victim. Regardless of its late respond to change of access behavior, dual greedy has very good stability under pattern change which is also an important issue for designing a good FTL.

5.4 Remark: The Needs for Adaptiveness

In table 5.2 we have concluded a comparison of DAC and Dual Greedy since these 2 both consider fine-grained data separation and use delicate GC policies concern more than just the victim utilization. Dual greedy retrenches about 97.4 percent memory consumption and need no manual-tuned parameters. With noticeably reduction of resource usage and simplification of implementation, the performances of dual greedy are still competitive against DAC.

An important design objective of Dual Greedy is to adapt various conditions including the write pattern, flash geometry, and over-provision. This experiment shows evidences that a competitive prior approach, i.e., DAC, could suffer from severe performance deterioration when using static parameters under various system environments. Figure 5.3 is a proof of how efficiency is sacrificed. All the performance is obtained from the default setting: PC trace, disk size 40G, block size 512KB, page size 4KB and the trace is executed twice. However, we slightly differentiate them by using different DAC parameters. The best setting for PC is using the optimal off-line tuned parameters of the default environment. The one for RND is the best parameters of RND trace with block size 1MB, page size

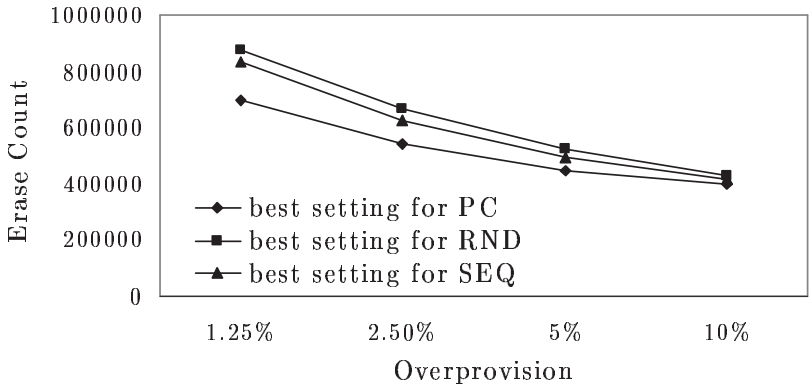

Figure 5.3: Under the PC workload, DAC delivered sub-optimal performance with the parameter settings which are the best choices for the RND and MM workloads.

8KB, and 1.25 percent over-provisioning ratio; the one for SEQ is from SEQ with the same default block/page size and 2.5 percent over-provisioning ratio. There maximum degradation caused by using inadequate settings is even up to 26 percent which indicates how crucial the selection of parameters is for DAC.

Chapter 6

Conclusion

Since ram is no longer a limit in the present flash storage structure, the rebirth of page-level mapping FTL is again brought to the table. However, flash storage is not as primitive as it used to be. Scalability has brought new challenges to resource requirement and performance. Even with abundant ram space, extravagant usage from previous algorithms will eventually dry out the resource. And the former exhausting victim selection policies which based on scoring of all GC candidates will ultimately become a bottleneck of performance when storage size becomes larger.

This article has proposed the dual greedy policy which successfully reduces the memory requirement by keeping track of only block-level information rather than storing per page time stamps. It is a huge leap from the old-fashioned management and memory space is substantially saved. Also dual greedy has taken the advantage of data structure to choose GC victim not only according to its utilization, but also stability which are sensitive alarm for preventing premature GC. This scheme involves no time-consuming scoring process and enables the system to adjust the threshold and victim selection standard at run-time which promises good flexibility and simplicity to diversified access patterns.

Dual greedy policy has competitive performance against many previous strategies. Even when compared to the resource-consuming DAC policy, dual greedy shows approaching even better results under various workloads. Also, dual greedy policy exhibits best resistency to large geometry environment which proves its potential to catch up on the trend of flash storage specification. When facing with radical change of access pattern, dual greedy holds good stability to pattern switch, therefore provides a guarantee to a reliable and stable

FTL.

Our future work is targeting at table management for page-level mapping FTL. Under large scale SSD, page-level mapping table will eventually become to large to afford, and how to manage such a big table is therefore vital. Also how to efficiently reconstruct the mapping information during star-up for large scale SSD is still thorny issue waiting to be solved.

Bibliography

[1] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. 1995. "A flash-memory based file system." In Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95). USENIX Association, Berkeley, CA, USA, 13-13.

[2] Jen-Wei Hsieh, Li-Pin Chang, and Tei-Wei Kuo. 2005. "Efficient on-line identification of hot data for flash-memory management." In Proceedings of the 2005 ACM symposium on Applied computing (SAC '05), Lorie M. Liebrock (Ed.). ACM, New York, NY, USA, 838-842.

[3] Mei-Ling Chiang, Paul C. H. Lee, and Ruei-Chuan Chang. 1999. "Using data clustering to improve cleaning performance for plash memory." Softw. Pract. Exper. 29, 3

[4] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. 2007. "A log buffer-based flash translation layer using fully-associative sector translation." ACM Trans. Embed. Comput. Syst. 6, 3, Article 18

[5] Chanik Park, Wonmoon Cheon, Jeonguk Kang, Kangho Roh, Wonhee Cho, and Jin-Soo Kim. 2008. "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications." ACM Trans. Embed. Comput. Syst. 7, 4, Article 38

[6] L. P. Chang, and T. W. Kuo,"A Real-time Garbage Collection Mechanism for Flash Memory Storage System in Embedded Systems," Proceedings of the The 8th International Conference on Real-Time Computing Systems and Applications, 2002.

[7] Li-Pin Chang and Tei-Wei Kuo. 2002. "An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems." In Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02) (RTAS '02). IEEE Computer Society, Washington, DC, USA, 187-.

[8] Soojun Im, Dongkun Shin, 2010. "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," Journal of Systems Architecture, Volume 56, Issue 12, December 2010, Pages 641-653

[9] M. L. Chiang, C. H. Paul, and R. C. Chang, "Manage flash memory in personal communicate devices," Proceedings of IEEE International Symposium on Consumer Electronics, 1997.

[10] Li-Pin Chang, "A Hybrid Approach to NAND-Flash-Based Solid-State Disks," IEEE Transactions on Computers, pp. 1337-1349, October, 2010

[11] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee. 2006. "A superblock-based flash translation layer for NAND flash memory." In Proceedings of the 6th ACM & IEEE International conference on Embedded software (EMSOFT '06). ACM, New York, NY, USA, 161-170.

[12] Li-Pin Chang and Li-Chun Huang. "A low-cost wear-leveling algorithm for block-mapping solid-state disks." In Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems (LCTES '11). ACM, New York, NY, USA, 31-40.