IEICE TRANS. INF. & SYST., VOL.E101–D, NO.9 SEPTEMBER 2018

LETTER

Data Recovery Aware Garbage Collection Mechanism in Flash-Based Storage Devices

Joon-Young PAIK[†a)], ***Nonmember***, **Rize JIN**[††b)], ***Member***, ***and*** **Tae-Sun CHUNG**[†c)], ***Nonmember***

SUMMARY In terms of system reliability, data recovery is a crucial capability. The lack of data recovery leads to the permanent loss of valuable data. This paper aims at improving data recovery in flash-based storage devices where extremely poor data recovery is shown. For this, we focus on garbage collection that determines the life span of data which have high possibility of data recovery requests by users. A new garbage collection mechanism with awareness of data recovery is proposed. First, deleted or overwritten data are categorized into shallow invalid data and deep invalid data based on the possibility of data recovery requests. Second, the proposed mechanism selects victim area for reclamation of free space, considering the shallow invalid data that have the high possibility of data recovery requests. Our proposal prohibits more shallow invalid data from being eliminated during garbage collections. The experimental results show that our garbage collection mechanism can improve data recovery with minor performance degradation.

key words: *NAND flash memory, Flash Translation Layer (FTL), storage system, data recovery*

1. Introduction

Data recovery is an indispensable factor in general systems. Data recovery is required to retrieve data deleted, overwritten, or damaged by human error, malicious behavior, or disaster from storage devices. A lack of data recovery capability can result in the permanent loss of valuable data. To avoid such a critical situation, many techniques for data recovery were proposed, including the Sleuth, FTK Imager, Recuva, and EnCase tools, especially in the forensic data area [1].

In commercial hard disk drives (HDDs), most of the deleted files were recovered by state-of-the-art tools, which showed data recovery rates of over 99.99% [1]. However, challenges have occurred recently, as NAND flash memory is quickly replacing traditional HDDs. Commercial solid-state devices (SSDs) based on flash memory showed extremely poor data recovery at a maximum rate of 5.43% [1].

Flash memory has different properties from HDDs: not-in-place updates of write operations, a new erase operation, the disparities in size and access speed among basic operational units, and so on. Thus, new software components have been designed for flash-based SSDs, including the Flash Translation Layer (FTL), wear leveling component, and garbage collector [2], [3]. The prior efforts have dramatic performance enhancement. However, these components make it difficult to recover data that are deleted, overwritten, or corrupted. First, the dynamic location policy of FTL and the wear leveling component prohibit the internal layout of stored data from being exposed to the host system. Second, the garbage collector physically eliminates meaningless data in order to create more free space for incoming write requests. The hidden information about the physical locations managed by these internal mechanisms in SSDs makes the host-level recovery techniques have inherited restrictions for data recovery. Therefore, it is required to design internal components that constitute an SSD, taking data recovery into account.

In this study, we target the garbage collector for the improvement of data recovery because it determines the life span of data, called invalid data, for which the user might demand recovery. Such invalid data are generated by the requests for overwriting or by the TRIM command [4]. We propose a Data Recovery aware Garbage Collection mechanism (DaRe-GC), which aims at prolonging the life span of invalid data that have a high possibility of data recovery requests.

The remainder of this paper is structured as follows. Section 2 describes garbage collection mechanism and related work. In Sect. 3, we explain our new garbage collection mechanism, DaRe-GC, to improve data recovery. Then, we compare DaRe-GC with a traditional garbage collection mechanism in terms of data recovery and performance in Sect. 4. Finally, we conclude our paper in Sect. 5.

2. Background and Related Work

In this section, we describe garbage collection mechanism. Then, we briefly discuss related work on data recovery in flash-based storage devices.

Garbage collection is an internal mechanism for flash-based storage systems [3], [5]–[8] that was designed to handle problems incurred by the not-in-place updates of write operations. In flash memory, fresh data are stored in a free location and overwritten data are marked as invalid data. Free space for write operations decreases as the invalid data increase. Eventually, this trend would exhaust available space. Garbage collection recycles such invalid space into

Manuscript received November 22, 2017.
Manuscript revised May 14, 2018.
Manuscript publicized June 20, 2018.

[†]The authors are with the Dept. of Computer Engineering, Ajou University, Republic of Korea.
[††]The author is with the School of Computer Science & Software Engineering, Tianjin Polytechnic University, China.
a) E-mail: lucadi@ajou.ac.kr
b) E-mail: jinrize@tjpu.edu.cn
c) E-mail: tschung@ajou.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2017EDL8255

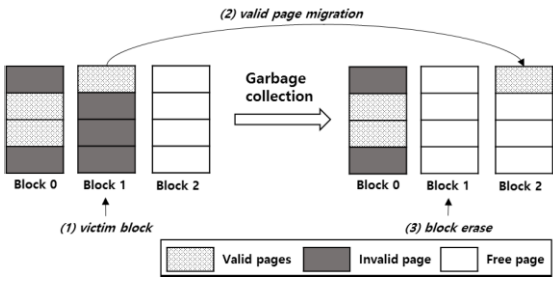


Fig. 1 Example of the garbage collection procedure.

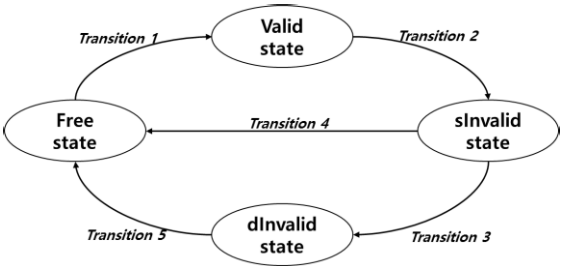


Fig. 2 State transition diagram of a physical page.

available space for write operations. Figure 1 elaborates how garbage collection makes free space by erasing a victim block, following the migration of valid pages in the block.

The extra operations for valid page migrations, involving garbage collections, impose a burden on performance. The most popular garbage collection mechanism, *greedy* scheme, aims at reducing garbage collection overhead by aggressively considering the migration costs of valid pages. For this, it selects a block of the lowest utilization of valid pages as a victim block. It accelerates performance by reducing the number of migrated pages.

There are several schemes for crash recovery in flash-based storage devices [9]. They aims at maintaining consistency between FTL metadata and data in flash memory from sudden crashes such as when powering off. To recover corrupted FTL metadata, they leverage a log-based recovery mechanism as in traditional database system. Our DaRe-GC as well as the previous schemes target the resident data in flash-based storage devices. However, our scheme focuses on the recovery of invalid data while they do on the recovery of FTL metadata.

3. DaRe-GC Mechanism

We propose a new garbage collection mechanism called DaRe-GC. Our goal is to prolong the life span of deleted or overwritten data, which are likely to be retrieved, and to retain more of such data on flash-based SSDs. For this, DaRe-GC selects a victim block for space reclamation based on the possibility of requests for data recovery. It prevents blocks that include invalid data with a high data-recovery possibility from being chosen as victim blocks.

3.1 State Transition of Physical Pages

To effectively secure invalid data of high possibility for data recovery requests in advance of actual recovery requests, we should solve a problem: how to estimate the possibility of data recovery requests. We exploit the depth of versions of invalid pages related to deleted or overwritten data to solve this. The latest versioned invalid data have a higher possibility of data recovery requests than the older-versioned ones. This is fundamentally similar to the approaches of data forensic tools in that they adopt the latest information about any actions to files deleted or corrupted as important evidence for data recovery.

In flash-based SSDs, a valid page has several versioned invalid data owing to its out-of-place manner. We categorize an invalid page into a shallow invalid page (*sInvalid* page) and a deep invalid page (*dInvalid* page) according to the degree of importance of data recovery. An sInvalid page indicates the latest invalidated page of a valid page, and a dInvalid page refers to one of the invalid pages that is not the latest invalid page of a valid page. Therefore, sInvalid pages are more valuable than dInvalid pages in terms of data recovery.

Figure 2 shows the state transition of a physical page: free, valid, sInvalid, and dInvalid states. Transition 1 is invocated when a physical page stores new data of a write operation in which the physical page becomes a valid state. In Transition 2, a physical page that has stored the valid data changes to an sInvalid state when valid data existing on the physical page are requested for an update or for a TRIM command. When the valid data that have the corresponding sInvalid data are updated by a write operation again, the state of the sInvalid data is changed from sInvalid to dInvalid data, thereby causing Transition 3. Finally, the sInvalid state and a dInvalid state are transited to a free state by erase operations, which indicates Transition 4 and Transition 5, respectively.

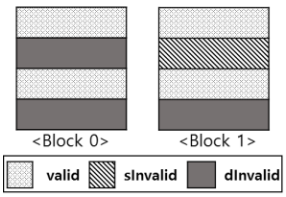
3.2 Proposed Victim Block Selection Policy

The victim block selection policy of DaRe-GC takes account of sInvalid pages for data recovery. When an amount of available space is less than a certain threshold, our DaRe-GC scheme selects a victim block that minimizes the following formula:

$$\frac{\#\text{ of valid pages}}{\#\text{ of pages in a block}} + \frac{\#\text{ of sInvalid pages}}{\#\text{ of pages in a block}} \times \text{Weight} \quad (1)$$

where *Weight* indicates how much an sInvalid page is considered as important as a valid page.

In Eq. (1), the former part, $\frac{\#\text{ of valid pages}}{\#\text{ of pages in a block}}$, calculates the migration overhead for valid pages, and the latter part, $\frac{\#\text{ of sInvalid pages}}{\#\text{ of pages in a block}} \times \text{Weight}$, shows the degree of data recovery requests for invalid pages in a block. The equation implies that blocks that have fewer valid pages for performance and



<Greedy garbage collection>

	Utilization
Block 0	0.5
Block 1	0.5

<DaRe-GC>

	Weight=0	Weight=0.5	Weight=1.0
Block 0	0.5	0.5	0.5
Block 1	0.5	0.625	0.75

Fig. 3 Comparison of a victim selection in DaRe-GC and greedy scheme.

fewer sInvalid pages for data recovery are selected. Therefore, the blocks with more sInvalid pages are unlikely to be chosen as victims. Then, the value of *Weight* ranges from 0 to 1.0. The *Weight* of 1.0 indicates that an sInvalid page is treated the same as a valid page, and *Weight* of 0.0 indicates that DaRe-GC ignores sInvalid pages where DaRe-GC works exactly the same as a greedy victim selection policy [5]. That is, our proposed victim block selection policy can be flexibly adjusted according to the importance of performance and data recovery by changing the value of *Weight*.

Figure 3 shows how DaRe-GC selects a victim block with different values of *Weight* in DaRe-GC. Each of Block 0 and Block 1 has two valid pages. A greedy garbage collection considers the two blocks as a victim block evenly because they are utilized in the same manner. In DaRe-GC, when *Weight* is 0, either block is selected, as in greedy garbage collection, because the blocks have the same calculated value of Eq. (1). When *Weight* is 0.5 or 1.0, Block 0 becomes a victim block. Block 0 has the lower value of Eq. (1) than Block 1 at each case because Block 1 has one sInvalid page. Thus, DaRe-GC prohibits a sInvalid page from being eliminated, compared to the greedy algorithm.

To handle information about the states of physical pages for victim selection and track the relations between valid pages and their respective invalid pages, we introduce a new data structure called TrackTable. Each entry in it consists of a flag to indicate the free, sInvalid, dInvalid, or valid state and a physical page address of the previous versioned page from a valid or an invalid page. TrackTable changes accordingly to the transitions of pages and is used for victim block selections with awareness of data recovery.

4. Evaluation

In this section, we evaluate how DaRe-GC affects data recovery and performance, compared to traditional garbage collection using greedy victim selection (hereafter referred to as *greedy-GC*). In Sect. 4.1, we describe the experimental environments used to evaluate DaRe-GC. Then, we compare DaRe-GC with greedy-GC with realistic traces in Sect. 4.2.

4.1 Experimental Environments

To evaluate the effects of DaRe-GC on data recovery and performance, we implemented DaRe-GC on an SSD simulator, EagleTree [10], which supports a variety of functionalities of multichannel parallelism [11], FTLs, garbage collectors, and so on. Table 1 shows the experimental configuration of the SSD.

Table 1 Configuration parameters for experiments.

Parameter	Value	Parameter	Value
No. of channels	8	No. of packages per channel	1
No. of dies per package	1	No. of planes per die	2
No. of blocks per plane	1024	No. of pages per block	128
Page size	4096 (bytes)	FTL scheme	Page-based mapping [2]

Table 2 Characteristics of realistic traces used.

	Total requests	Writes requests (%)	Reads requests (%)	Average size of writes (in KB)	Average size of reads (in KB)
hm_0	3,993,316	64%	36%	8.34	7.37
src1_2	1,907,773	75%	25%	32.51	19.11
stg_0	2,030,915	85%	15%	9.19	24.92
usr_0	2,237,889	60%	40%	10.28	40.92
wdev_0	1,143,261	80%	20%	8.2	12.57

Realistic read and write requests of Microsoft research traces (MSRs) [12] were used in our experiments. The features of the used traces are listed in Table 2. Each MSR trace was repeated more than one time per experiment so that it generated enough garbage collections to investigate the effects of our proposal on data recovery and performance.

4.2 Experimental Results

We measured the average number of sInvalid pages eliminated and the average number of valid pages migrated during 200,000 garbage collections with five realistic traces in greedy-GC and DaRe-GC, in which the value of *Weight* changed from 0.1 to 0.5 in DaRe-GC.

The average number of sInvalid pages in victim blocks with greedy-GC and DaRe-GC is shown in Fig. 4. It indicates how many sInvalid pages of a high possibility of data recovery requests are eliminated per garbage collection. For all traces, DaRe-GC allows less sInvalid pages to disappear per garbage collection, thereby securing more invalid data of high possibility for data recovery requests in flash memory. It is because a victim block is selected with awareness of sInvalid pages as well as valid pages in a block in DaRe-GC, unlike in greedy-GC. In case of MSR-src1_2, DaRe-GC reduces the number of sInvalid pages by up to 40.55. This trace has the largest-sized write requests, 32.51 KB, as indicated in Table 2. Those write requests of large data cause

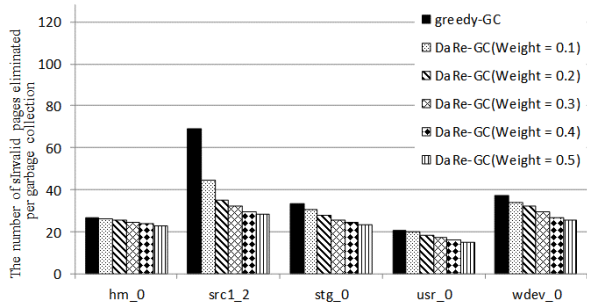


Fig. 4 Change in average number of sInvalid pages eliminated per garbage collection.

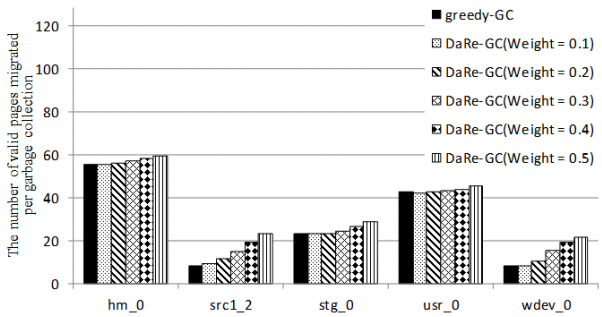


Fig. 5 Change in average number of valid pages migrated per garbage collection.

sInvalid pages rather than dInvalid pages with greedy-GC. It is because large-sized write requests have lower temporal locality than small-sized write requests [13].

In addition, larger values of *Weight* result in fewer sInvalid pages being eliminated during garbage collections because a larger value of *Weight* prohibits a block with more sInvalid pages from being selected as a victim block. As a result, DaRe-GC prolongs the life span of more sInvalid pages than greedy-GC, which increases the possibility of data recovery.

Figure 5 describes the migration overhead per garbage collection. DaRe-GC has a negative impact on performance. MSR-hm_0, MSR-stg_0, and MSR-usr_0 traces have a similar number of valid pages in a victim block in which the increase is limited by five valid pages. However, MSR-wdev_0 and MSR-src1_2 traces cause more valid pages of 13 and 14 to be migrated, respectively, in DaRe-GC than in greedy-GC. The resultant performance degradation can be explained from the victim selection policy of DaRe-GC. A block with more valid pages can be chosen as a victim owing to the weighted value of sInvalid pages, even though other blocks have less valid pages.

Figures 4 and 5 show the trade-off between data recovery and performance. We can improve data recovery with a minor performance degradation by adjusting *Weight* according to the type of trace. The performance is estimated at an SSD level. The performance can be optimized further more via I/O stack of general systems (i.e., Linux, Windows, and SQLite). In particular, the overhead can be minimized by I/O optimization techniques through the I/O stack.

5. Conclusions

This paper presented a DaRe-GC mechanism that increases the likelihood of recovering data deleted or overwritten by a host system. In DaRe-GC, invalid data incurred by overwriting request or delete requests is separated into sInvalid data and dInvalid data. DaRe-GC selects victim blocks, considering sInvalid pages as well as valid pages. It prohibits blocks that include more sInvalid pages from being chosen as victim blocks, which increases the possibility of data recovery requests for data deleted or overwritten. The experimental results showed that DaRe-GC reduced the number of sInvalid pages eliminated per garbage collection, which was likely to achieve more data recovery requests. However, it caused larger garbage collection overhead. We plan to accelerate data recovery while relieving negative impacts on performance.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2017R1A6A3A11035567 and 2016R1D1A1B03934129).

References

[1] F. Geier, "The differences between SSD and HDD technology regarding forensic investigations," Degree Project, Linnaeus University, Sweden, 2015.

[2] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," J. Syst. Architect., vol.55, no.5-6, pp.332–343, 2009. DOI: 10.1016/j.sysarc.2009.03.005

[3] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, and T.-W. Kuo, "Garbage collection and wear leveling for flash memory: past and future," Proc. IEEE Smart Computing (SMARTCOMP), pp.66–73, 2014. DOI: 10.1109/SMARTCOMP.2014.7043841

[4] Information technology-ATA/ATAPI Command Set, [Online]. Available: http://www.t13.org/documents/UploadedDocuments/docs2009/d2015r1a-ATAATAPI_Command_Set_-_2_ACS-2.pdf.

[5] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," Proc. ASPLOS, pp.86–97, 1994. DOI: 10.1145/195473.195506

[6] A. Kauaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," Proc. USENIX Tech. Conf., TCON, 1995.

[7] M.-L. Chiang and R.-C. Chang, "Cleaning algorithms in mobile computers using flash memory," J SYST SOFTWARE, vol.48, no.3, pp.213–231, 1999. DOI: 10.1016/S0164-1212(99)00059-X

[8] O. Kwon, K. Koh, J. Lee, and H. Bahn, "FeGC: an efficient garbage collection scheme for flash memory based storage systems," J SYST SOFTWARE, vol.84, no.9, pp.1507–1523, 2011. DOI: 10.1016/j.jss.2011.02.042

[9] C. Zhang, Y. Wang, T. Wang, R. Chen, D. Liu, and Z. Shao, "Deterministic crash recovery for nand flash based storage systems," Proc. DAC, pp.1–6, 2014. DOI: 10.1145/2593069.2593124

[10] N. Dayan, M.K. Svendsen, M. Bjørling, P. Bonnet, and L. Bouganim, "EagleTree: exploring the design space of SSD-based algorithms," VLDB (demo), vol.6, no.12, pp.1290–1293, 2013. DOI: 10.14778/2536274.2536298

[11] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and H. Zhang, "Performance impact and interplay of SSD parallelism through advanced

commands, allocation strategy and data granularity," Proc. ICS, 2011. DOI: 10.1145/1995896.1995912

[12] UMASS Trace Repository, [Online]. Available: http://traces.cs.umass.edu/.

[13] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: locality-aware sector translation for nand flash memory-based storage system," SIGOPS Oper. Syst. Rev., vol.42, no.6, p.36, Oct. 2008.