# Applying Background Garbage Collection to the SBAST Flash Translation Layer Scheme

Ilhoon Shin
Department of Electronic Engineering & IT Media
Seoul National Univ. of Science & Tech.
Seoul, South Korea
*ilhoon.shin@snut.ac.kr*

***Abstract***—NAND-based block devices have the overhead of performing the garbage collection to reclaim clean pages. A feasible solution to this problem is performing a background garbage collection that is executed in advance in idle time. Because the background garbage collection can hurt the latency of the foreground requests, it needs to identify stable states of the background garbage collection so that it can be terminated instantly when a new I/O request arrives. This work applies the background garbage collection to one of the hybrid FTL, the SBAST scheme by analyzing the garbage collection process and identifying the stable states. The analysis shows that the worst influence on the foreground requests is limited to multiple page copies and one block erase.

***Keywords****-background garbage collection; SBAST; flash translation layer; NAND flash memory*

*****

## I. INTRODUCTION

NAND flash memory is widely used as storage media for mobile devices. Its strengths such as light-weight, silence, shock-resistant, and energy efficiency have driven the success and made it possible to replace hard disk drives. The restriction of NAND flash memory is that it does not provide an overwrite operation directly because of an erase-before-write feature. To overcome it, NAND-based block devices employ a firmware called flash translation layer (FTL) to emulate the overwrite operation. The FTL processes an overwrite request in an out-of-place update, which writes new data to a new clean page. By the out-of-place update, clean pages are eventually exhausted and a garbage collection process is initiated to reclaim the invalidated pages to clean pages.

The detailed process of the garbage collection is different according to the FTL schemes. The basic steps are selecting the victim block, moving the valid pages of the victim blocks, and reclaiming clean pages by an erase operation. It generally accompanies multiple page read/write and block erasures, which degrades the response time of a foreground request, significantly. Thus, it is important to reduce the frequency and the latency of the garbage collection. One possible solution for it is to perform the garbage collection in advance in idle time, which is called a background garbage collection. The previous research [1] has designed and evaluated the background garbage collection for the page mapping scheme [2]. However, as to other FTL schemes, the background garbage collection has not been designed in detail. The goal of the work is to design the background garbage collection for one of the hybrid mapping schemes, the SBAST (Shared Block Association Sector Translation) scheme [3].

The remainder of the paper is organized as follows. Section 2 gives a description of background knowledge such as NAND flash memory and the representative FTL schemes. Section 3 designs the background garbage collection for the SBAST scheme. Section 4 draws a conclusion.

## II. BACKGROUND

NAND flash memory is a semiconductor that consists of blocks and pages. A page is a read/write unit, and a block is an erase unit. A block consists of multiple pages. NAND flash memory has the erase-before-write feature. Data can be written only in clean pages. Once data are written to a page, the page cannot be updated. In order to write new data to the page, the page should be changed to clean status, in other words, the block that the page belongs to should be erased. Thus, the overwrite operation is generally supported by the out-of-place update.

In the out-of-place update, the location of valid data is changed on every write. Thus, we need to remember the current location of valid data. For this purpose, FTL maintains the mapping table between the logical address space that is used by file systems and the physical address space. By the granularity of the mapping table, FTL is classified to page mapping [2], block mapping [4], and hybrid mapping [3, 5, 6].

In the page mapping scheme [2], the mapping granularity is a page. Upon a write request, data are written in clean pages in a page unit. If clean pages are scarce, the garbage collection is performed. It selects a victim block, moves valid pages of the victim to an extra clean block, and then reclaims clean pages by erasing the victim. It delivers a good performance, and however large memory is required to maintain the mapping table because FTL maintains the mapping between logical page number (LPN) and its physical page number (PPN).

The block mapping scheme [4] uses a block as the mapping granularity to reduce the mapping table. Upon a write request, data are written in clean blocks in a block unit. If the modified data are smaller than a block, the unmodified data of the old block are also copied to the new block. Thus, its performance is significantly downgraded.

The hybrid schemes mix both schemes. They use a portion of NAND blocks as write buffer called log blocks, which are managed by the page mapping. The other blocks called data blocks are managed by the block mapping. In the BAST (Block Associative Sector Translation) scheme [5], log blocks and data blocks are associated with one to one mapping. On a write request, data are written to the log block associated with the target data block. If there is no associated log block, a clean log block is allocated. If there is no clean log block, the garbage collection is started. It selects a victim log block, copies valid pages of the victim log block and the old data block to a new data block. Finally, the victim log block and the old data block are erased. The BAST scheme is vulnerable in a random write pattern because log blocks are merged and erased with having clean pages [6].

In order to solve the drawback of the BAST scheme, the FAST (Full Associative Sector Translation) scheme [6] lets data blocks share log blocks. On a write request, data are written to a working log block regardless of the target data block. If there is no clean page in the working log block, a clean log block becomes a new working log block. If there is no clean log block, the garbage collection is started. It selects a victim log block and merges it with the associated data blocks. Different from the BAST scheme, log blocks can be associated with multiple data blocks and thus the merge operation is repeatedly performed until there is no valid page in the log block. Therefore, the latency of the garbage collection can be prolonged.

The SBAST scheme [3] also allows data blocks to share log blocks. However, a data block can be associated with only one log block. Thus, the association degree of log blocks is mostly lower than the FAST scheme. But, the garbage collection latency can still be prolonged according to the association degree. In order to solve this problem, previous research [7] has proposed to perform the log block move selectively. For example, if the victim log is associated with multiple log blocks, the garbage collection copies valid pages of the victim log to a clean data block. The victim log block is erased and inserted to a pool of clean data blocks. The data block that has moved valid pages is switched to a log block. This log block move and the repeated log block merge are selectively performed based on the cost-benefit analysis.

## III. Designing Background Garbage Collection for the SBAST Scheme

The background garbage collection is an effective solution to hide the overhead of the long garbage collection. In this section, we design the background garbage collection for the SBAST scheme. When designing the background garbage collection, we should minimize the harmful influence on a foreground request. In other words, if a new request arrives during the background garbage collection, it should be terminated to a stable state as soon as possible to serve the foreground request [1].

Fig. 1 shows the state transition of the log block move process in the SBAST scheme. If the garbage collection is performed with the log block move, it first selects a victim log block. Then, if there are valid pages in the victim, copying valid pages to a clean data block are repeatedly performed until there are no valid pages in the victim log. Then, the victim is erased, and the victim and the data block are switched. Thus, there are seven states (S0 – S6) from the start to the termination of the garbage collection.

If new I/O requests arrive in the middle of the background garbage collection, it should be terminated in a stable state. Among the states, only S1 is a stable state except the start and the termination. Once copying valid pages to a data block is started (S2), the states are unstable because the valid pages are scattered to the victim log block, the current data block, and a new data block. Thus, we wait until the garbage collection is finished. However, the waiting time is not very long because it performs multiple page copies and one block erase.

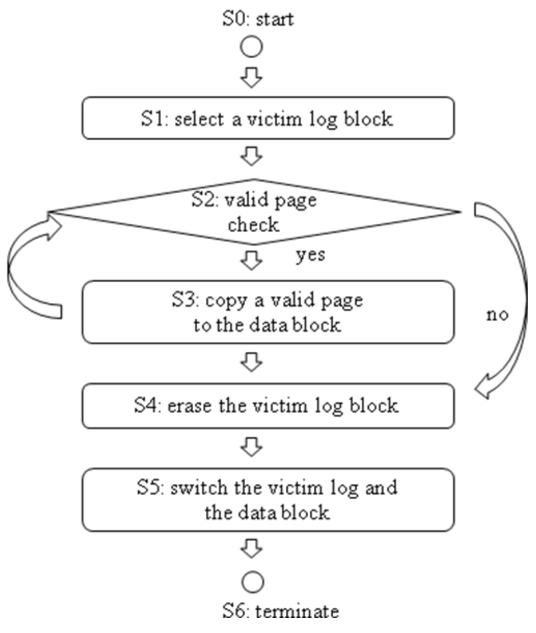
S0: start
S1: select a victim log block
S2: valid page check
yes
no
S3: copy a valid page to the data block
S4: erase the victim log block
S5: switch the victim log and the data block
S6: terminate

Figure 1. Work flow of the log block move.

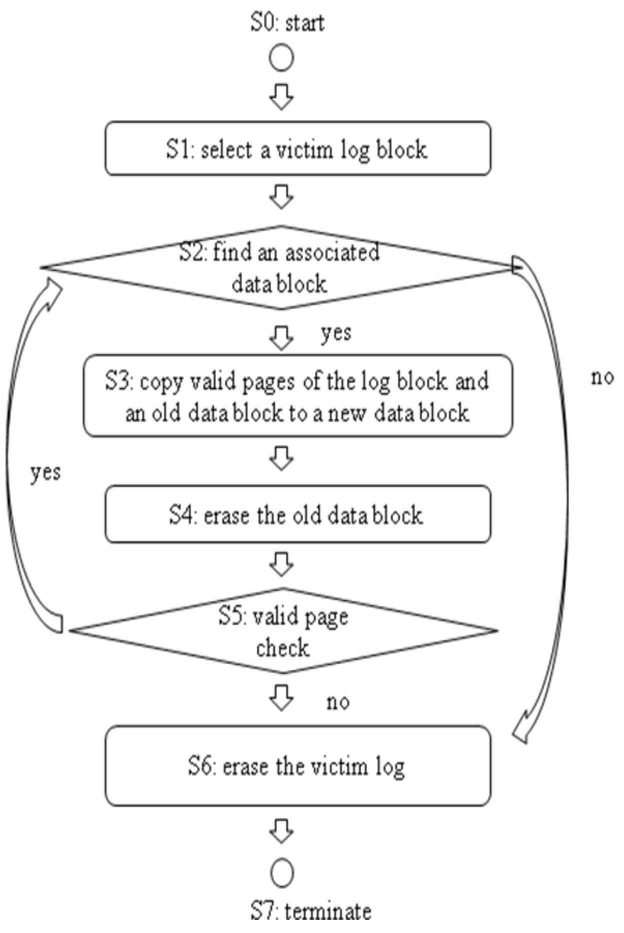
S0: start
S1: select a victim log block
S2: find an associated data block
yes
S3: copy valid pages of the log block and an old data block to a new data block
no
yes
S4: erase the old data block
S5: valid page check
no
S6: erase the victim log
S7: terminate

Figure 2. Work flow of the log block merge.

Meanwhile, Fig. 2 shows the state transition of the repeated log block merge process. It first selects a victim log block. Then, it searches for an associated data block. If there is no associated data block, that is, there is no valid pages, it erases the victim block and the garbage collection is finished. However, if the associated data block is found, the merge with the data block is performed. Valid pages are copied to a new data block, and the old data block is erased. This merge is repeated performed until there is no valid page. Finally, the log block is erased. Thus, there are eight states (S0 – S7) from the start to the termination of the garbage collection.

Among the states, S3 and S4 are unstable. Once the merge with a data block is started, valid pages are scattered. Thus, we wait until the merge with the data block is finished. Then, the garbage collection is instantly terminated. When the next garbage collection is initiated, it will find this victim log block again, and its association degree is lower than before. The worst-case waiting time is one block merge time, which includes multiple page copies and one block erase. Fig. 3 shows the pseudo code of the background garbage collection.

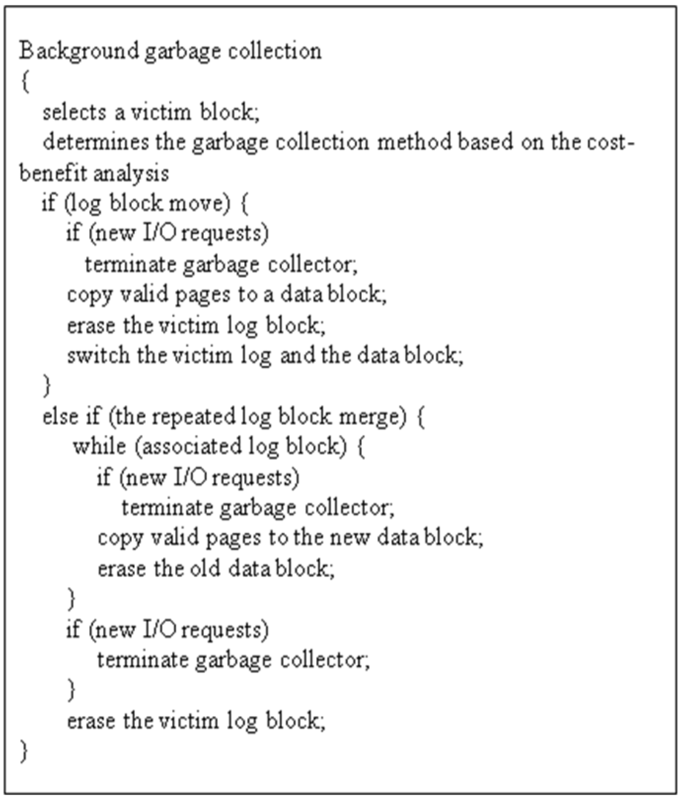
```
Background garbage collection
{
  selects a victim block;
  determines the garbage collection method based on the cost-
benefit analysis
  if (log block move) {
    if (new I/O requests)
      terminate garbage collector;
    copy valid pages to a data block;
    erase the victim log block;
    switch the victim log and the data block;
  }
  else if (the repeated log block merge) {
     while (associated log block) {
       if (new I/O requests)
          terminate garbage collector;
       copy valid pages to the new data block;
       erase the old data block;
     }
     if (new I/O requests)
       terminate garbage collector;
     }
     erase the victim log block;
}
```

Figure 3. Pseudo code of background garbage collection.

## IV. CONCLUSION

This work designed the background garbage collection that performed the garbage collection in advance in the idle time for the SBAST FTL scheme with the selective garbage collection. If the I/O requests arrived in the middle of the background garbage collection, it should be terminated in stable states to serve the requests instantly. For the purpose, we analyzed each garbage collection process and identified the stable states. From the analysis, the worst influence on the foreground requests was limited to multiple page copies and one block erase. With this cost, we can get a considerable performance benefit because log blocks are reclaimed in advance or the association degree of log blocks is lower.

## ACKNOWLEDGMENT

This study was supported by the Research Program funded by the Seoul National University of Science and Technology (2104-0744).

## REFERENCES

[1] I. Shin, “Performance Evaluation of Background Garbage Collection for Solid State Drives”, Information Journal, vol. 17, pp. 5557–5566, November 2014.

[2] A. Ban, “Flash file system”, U.S. Patent, 5,404,485, April 1995.

[3] I. Shin, “Light Weight Sector Mapping Scheme for NAND-based Block Devices”, IEEE Trans. on Consumer Electronics, vol. 56, pp. 651–656, May 2010.

[4] A. Ban, “Flash file system optimized for page-mode flash technologies”, U.S. Patent. 5,937,425, August 1999.

[5] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, “A space-efficient flash translation layer for compactflash systems”, IEEE Transactions on Consumer Electronics, vol. 48, pp. 366–375, 2002.

[6] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and H. J. Song, “A log buffer based flash translation layer using fully associative sector translation”, ACM Transactions on Embedded omputing Systems, vol. 6, 2007.

[7] I. Shin, “Selective Garbage Collection for Hybrid Mapping Flash Translation Layer based on Cost-benefit Analysis”, Information Journal, vol. 17, pp. 6427–6432, December 2014.