A Read-Disturb Management Technique for High-Density NAND Flash Memory

Keonsoo Ha, Jaeyong Jeong, and Jihong Kim
Department of Computer Science and Engineering, Seoul National University, Korea
{air21c, jyjeong, jihong}@davinci.snu.ac.kr

Abstract

The read-disturb problem is emerging as one of the main reliability issues for future high-density NAND flash memory. A read-disturb error, which causes data loss, occurs to data in a page when a large number of reads are performed to its neighboring pages in the same block. In this paper, we propose a novel read-disturb management technique which reduces the frequency of expensive data migrations needed for avoiding read-disturb errors. Our technique proactively converts highly skewed read accesses to a small number of blocks into more balanced read accesses to a large number of blocks by intelligently changing data block locations accessed. Our experimental results show that our technique is effective in handling the read-disturb problem, reducing the time overhead of data migrations on average by 50% over an FTL based on the existing read-disturb management technique.

1 Introduction

As the density of NAND flash memory increases using advanced process techniques such as shrinking processes (e.g., 20 nm and below process technology) and multi leveling (e.g., triple-level cell (TLC)), the read-disturb problem is expected to emerge as a major reliability concern for future high-density NAND flash memory [7]. When a page P in a block B is read in NAND flash memory, a read-disturb error may occur to P's neighboring pages in the block B. Since NAND cells are serially connected in a string structure, cells in the same string are unintentionally programmed when one of their neighboring cells is read. As the number of unintentional programs increases for the same cell, the logic state of the cell may change. If the number of changed bits by read disturbs exceeds the number of recoverable bits by an error correction code (ECC), a read-disturb error occurs.

In order to avoid data corruption by a read-disturb error, an anticipatory prevention procedure, called *read reclaim (RR)*, is required. Since a disturbed block B can return to its initial undisturbed status when it is erased, the block B is erased during RR when RR decides that the current level of read disturbance of the block B is sufficiently high. If valid data exist in the erased block B, they must be moved to other healthier blocks before the block B is erased. One straightforward RR technique maintains the number of performed read operations per block to predict the read-disturbance status of a block [5]. When a block undergoes more read operations than a preset upper bound on the number of read operations allowed per block, called *RR threshold*, RR is triggered. (We call this method `baseline`.)

Up to 30 nm multi-level cell (MLC) NAND flash memory, RR is rarely activated because of the strong read-disturb resistance. However, as the techniques for increasing the density of NAND flash memory is evolved, the resistance has been significantly weakened, resulting in quick decreases in the maximum allowable read count between two consecutive block erasures. (In this paper, we call it *the maximum read count*.) Fig. 1 shows a future trend on the maximum read count for MLC and TLC NAND flash memories, which is estimated by using an FN-tunneling equation (in a similar fashion used for forecasting the read-disturb trend of the MLC flash memory [3]). For a block in TLC NAND flash memory, the maximum read count may be just about 40,000. This trend is indicated as a dashed line with a square symbol (i.e., the min case) in Fig. 1. For such a small maximum read count, RR will occur quite

APSys '13, July 29-30 2013, Singapore, Singapore
Copyright 2013 ACM 978-1-4503-2316-1/13/07 ...$15.00.

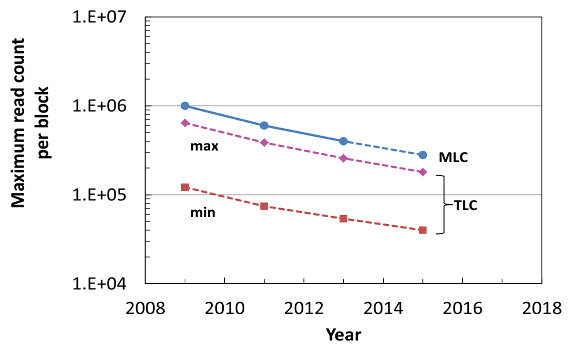


Figure 1: A projected read-disturb trend of future MLC and TLC devices.

frequently in future TLC NAND flash memory.

Frequent RR operations negatively affect the performance of NAND flash memory because of extra data migrations and block erasures during RR invocations. In our evaluation, `baseline` technique was quite ineffective under small maximum read counts even with a 256 MB read buffer. In our observation, for many read dominant applications such as a web search engine server, an ad server, and a database server, a large number of read reclaims still occur (even with a large read buffer in a storage device) because such applications often have a very large working set for read accesses. On average, 90% of total read requests from host systems are passed to physical NAND pages when a 256 MB read buffer was used in the storage device. When the maximum read count was 40,000, `baseline` increased overhead time by about 10 times on average over when no RR was considered. When an I/O request was overlapped with an RR procedure, the I/O response time was increased by up to about 66 times. This significant performance penalty is mainly because frequently-read data (which we call *read-hot* data) are read from a small number of blocks. Since `baseline` does not modify the read skewness of a given workload, it simply moves the same read-hot data to a different block, thus repeating RR soon.

We propose a novel read-disturb management technique which reduces the occurrence of RR. Our technique detects read-hot pages in a partially disturbed block and *proactively* moves them to other healthier blocks before RR is activated. By distributing read requests, our technique reduces RR occurrences. Moreover, by avoiding simultaneous data migrations, our technique better balances I/O response times under RR activations. Based on our proposed technique, we have designed a new read disturb-aware flash translation layer (RedFTL) for high-density NAND flash memory. Experimental results show that RedFTL can reduce the time overhead of RR on average by 50% over the `baseline` technique.

2 Basic Idea

In order to understand how the `baseline` technique works for a future high-density NAND flash memory, we evaluated `baseline` using read-intensive applications. Fig. 2 shows to what extent the total overhead execution time increases during garbage collection (GC) and RR when `baseline` is applied. The x-axis denotes various maximum read counts, and the y-axis represents the normalized total overhead execution times. On the x-axis, the ∞ maximum read count indicates when no RR is activated. Each overhead execution time is normalized to the total execution time spent for GC for the ∞ case. As shown in Fig. 2, the smaller the maximum read count is, the more overhead time is spent because of more frequent RR activations. In particular, when the maximum read count was 40,000, data migrations during RR accounted for 88% of the total overhead execution time. This result shows that lots of valid pages exist in disturbed blocks, and the time overhead of moving them during RR can be considerable if they simultaneously migrate.

From a detailed analysis, we observed that a small number of heavily read blocks are responsible for frequent RR activations in `baseline`. Fig. 3.(a) illustrates why `baseline` works poorly using an example. Each block has four pages, and each page is represented with a rectangle. A tuple (I, N) in a rectangle indicates that data in a page are read I times per given time period p, and the data have been read N times since the last block erasure. Assume that the RR threshold value is 1,400. As shown in Fig. 3.(a), RR is invoked for Block 0 at time t_1 because the read access count to Block 0 reaches the RR threshold. As a result, the Pages A, B, C, and D are copied to Block 1, and Block 0 is erased. Since those pages are read 1,400 times for every time period p, RR is invoked for Block 1 once again at time $(t_1 + p)$. If the frequently read Pages A and B had not been migrated to-

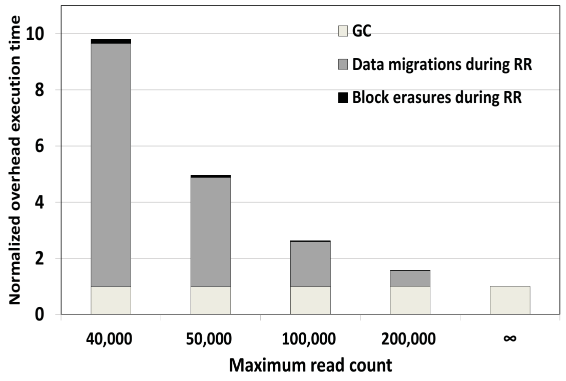
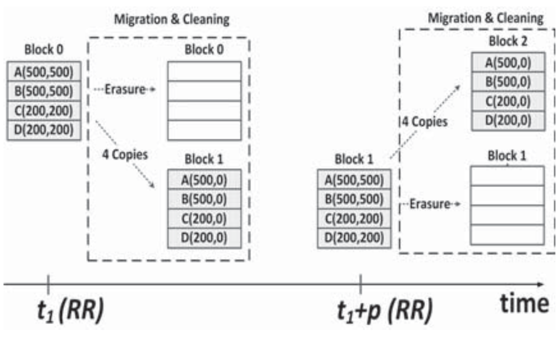
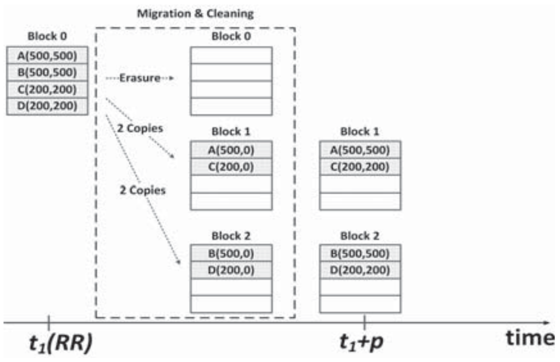


Figure 2: Normalized total overhead execution times for GC and RR.



(a) The `baseline` technique



(b) The proposed one-to-many migration technique

Figure 3: A snapshot comparison of RR using different data migration techniques.

gether to the same Block 1, the second RR might have been avoided because more blocks could be evenly read.

Based on this observation, we propose a one-to-many data migration technique which splits pages in the same block into multiple groups and moves each group to a different block. Our technique detects read-hot pages in a partially disturbed block and moves them to less disturbed blocks during RR. Those moved read-hot pages are less likely to cause another RR activation because they have been moved to blocks with small read counts. In Fig. 3.(b), our technique moves Pages A and C to Block 1, while Pages B and D are copied to Block 2. In this case, RR does not occur at time $(t_1 + p)$ due to low read access counts in each block. If Blocks 1 and 2 are erased by GC or WL before their read access counts get close to the RR threshold value, their read disturbance can be fully recovered without RR.

RR also incurs a significant fluctuation of I/O response times. Fig. 4 shows a snapshot of response time variations after about 800 million read requests of the websearch benchmark trace were performed. The x-axis and y-axis represent the *logical* read access time and I/O response time for a read request, respectively. The logical read access time increases by one whenever a read operation (to any page) is performed. The maximum read

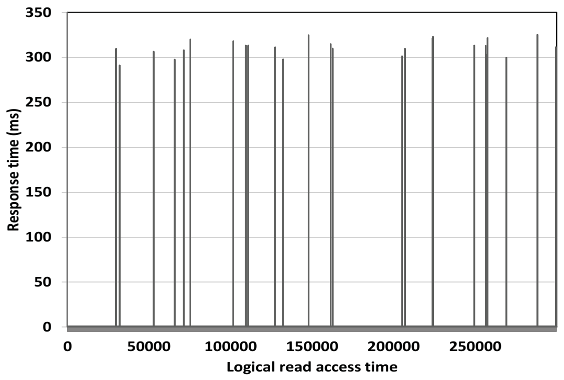


Figure 4: A snapshot of response times when websearch is executed.

count was set to 40,000. In Fig. 4, there are many high peaks of the response time because of simultaneous data migrations and block erasures during RR. In our observation, 85% of pages in a block are moved during RR, thus taking a long time to complete a single RR activation. If a block filled with a large number of valid data is erased during RR, as shown in Fig. 4, the response time is increased to 332 *ms*, about 66 times increase over a normal block erasure response time. If several pages in those valid pages are moved to other healthier blocks before an RR activation, there would be less response time fluctuations.

Based on this observation, we also suggest a proactive data migration technique as part of our main technique, which mitigates the fluctuations of I/O response time. By moving potential read-hot pages in advance before an RR activation, our technique spreads the time overhead of data migrations for a longer time period.

3 RedFTL: Read Disturb-Aware FTL

3.1 Overview of RedFTL

Based on two ideas explained in Sec. 2, we have designed a new read disturb-aware FTL, called RedFTL, for high-density NAND flash memory. Fig. 5 shows an organizational overview of RedFTL. It consists of common modules of a typical FTL as well as several special modules which are specifically designed for read-disturb management support such as a read-hot page separator, a good block pool, a migration manager, and a replica mapping table.

For a given read request for a block, RedFTL first checks whether RR is likely to be activated soon for the block or not. If the read access count of the block is greater than a preset threshold, called the *replica creation threshold*, RedFTL detects read-hot pages in the block and moves them to other healthier blocks in order

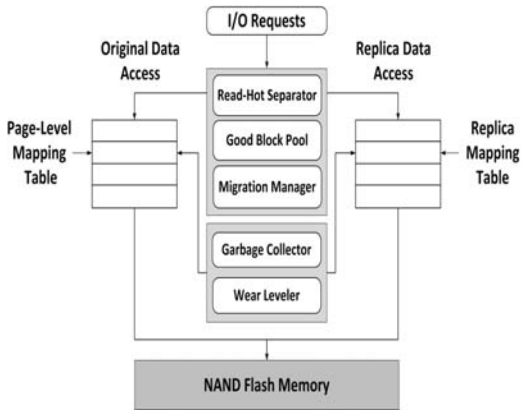


Figure 5: An organization of RedFTL.

to avoid an occurrence of RR. The read-hot page separator classifies read-hot pages in a block based on the read access pattern of each page. The migration manager copies a replica of a read-hot page to a healthier block, and the location information of the replica page is updated in the replica mapping table. When a replica page is created, if the original read-hot page of a replica page is immediately invalidated, the block with the original page is likely to be selected as a GC victim block because the number of invalid page was increased. In order to avoid such GC behavior, the migration manager makes a copy of the original page instead of moving the original page. A replica page is invalidated either when the original page of the replica page is updated or when the block with the replica page is erased during garbage collection or wear leveling. RedFTL distributes read requests to more blocks by changing read requests to be read in a replica page if the replica page exists.

Once replica pages in a block have serviced a significant number of read requests, the replica page is permanently moved to the new block by invalidating the original read-hot page of that replica page. This is determined by checking whether the read access count of the block with a replica page exceeds another preset threshold, called the *migration threshold*, or not.

3.2 Read-Hot Page Separation

In order to select a read-hot page in a block, the read-hot page separator compares the read-access rate R_P of a page P with the read-access rate R_B of a block B. The read-access rate R_P is defined as $\frac{r_P}{t_P^{last} - t_P^{first}}$ where r_P denotes the read access count of the page P, and t_P^{first} and t_P^{last} represent the first and the last logical read access times of the page P, respectively. The read-access rate R_B can be defined similarly by $\frac{r_B}{t_B^{last} - t_B^{first}}$ where r_B indicates read access count of the block B, while t_B^{first} and t_B^{last} denote the first and the last logical read access times of the block B, respectively. The page P is classified as a read-hot page if $R_P > \alpha \times R_B$. The constant parameter α is used to control the access skewness of read requests in determining a read-hot page. (In the current version, we set α with 2 based on several experiments.)

If many pages in the block are continuously read at a similar pace, they are likely to activate RR frequently if they remain in the same block together. Thus, if read accesses to the block B are almost evenly distributed among its pages, the read-hot page separator selects randomly a half of the valid pages as read-hot pages.

3.3 Good Block Pool Management

RedFTL manages a pool of good blocks, which we call the *good block pool (GBP)*. GBP maintains less disturbed healthy blocks, which are used in allocating the replicas of read-hot pages. In order to prevent read-hot pages in a block from being stored in the same block, a read-hot page is allocated to a healthier block in the GBP according to FIFO. If a block stores more replica pages than a preset maximum number of replica pages per block, the block is removed from GBP because storing a large number of replicas in a block may waste too much space. Moreover, if a replica page is created in a block, this block is removed from GBP because a block with one or more read-hot pages may have been already partially disturbed.

4 Experimental Results

A trace-driven FTL simulator was used in our experiments to evaluate our technique. Tables 1 and 2 summarize various parameters of our simulator and the characteristics of the benchmark traces used for our experiment, respectively. These parameters are based on the recent TLC NAND specification [8]. GC was triggered when the total number of remaining free blocks was less than 4% of the total number of blocks, and it was continued until 6% of the entire blocks became free blocks. The entire blocks was set to 65,536 except for mds. Since mds requires more blocks due to its large working set size,

Table 1: Key parameters of the FTL simulator for our experiments

Flash Setting	Value	FTL Setting	Value
Pages per Block	192	Mapping	Page Level
Page Size	8 KB	GC	Greedy Policy
Page Read Latency	100 us	WL	Swapping
Page Write Latency	1600 us	Buffer Size	256 MB
Block Erasure Latency	5 ms	RR Threshold	38,000

Table 2: Summary of benchmark traces

Benchmark	Description	Read (%)	Trace Interval	Repeat Count
ads [1]	Display ads platform	96	1 day	150
mds [4]	Media server	98	1 week	200
tpc-h [6]	Accesses to a database	92	10 hours	50
websearch [2]	Search engine	100	4 days	100

the number of entire blocks was set to 287,995 which is five times of the working set size of mds. Furthermore, migration and replica creation threshold values were set to 90% and 70% of the maximum read count, respectively. In our simulator, wear leveling is activated if the difference of P/E cycles between the oldest block and the youngest block is greater than 40, but it was not triggered in our experiments.

We used highly read-dominant public benchmark traces which were collected from actual systems. The trace interval in Table 2 indicates the length of the time interval during which a corresponding trace was collected. We repeated the same benchmark trace multiple times to generate enough read requests in our experiment. The number of iterations for each trace is indicated as the repeat count in Table 2.

Fig. 6 shows normalized overhead execution times for data migrations and replica copies during RR when 15 pages in each block were used to store replica pages. The x-axis indicates benchmark traces and applied techniques, and the y-axis denotes the execution time for RR which is normalized to that of `baseline`. Since RedFTL creates the replica pages of read-hot pages and places them to multiple blocks, an extra time overhead occurs. In Fig. 6, in the case of tpc-h, many replica pages were created, but they were vanished by frequent GC procedures, thus increasing the extra time overhead. Moreover, in the case of mds, a significant number of pages were classified as read-hot pages by the read-hot page separator because many pages were evenly read at a similar pace. Although this time overhead occupied 56% of the total execution time for RR, RedFTL decreased the overall execution time for RR, on average, by 50% over `baseline` because it reduced the time overhead for data migrations by 78%.

Fig. 7 illustrates the total overhead execution time for GC and RR. The x-axis denotes the maximum number of replica pages per block and benchmark traces, and the y-axis represents the overhead execution time which is normalized over `baseline`. The `baseline` is represented by 0 on the x-axis. As shown in Fig. 7, RR activations were decreased as the maximum number of replica pages per block gets larger. Since a large number of replica pages can contribute to evenly distribute the read skewness of a given workload, the occurrences of RR was reduced. However, creating replica pages negatively affected performance in the cases of websearch

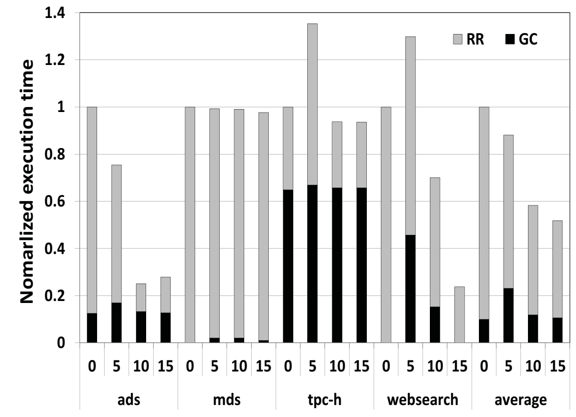


Figure 7: A breakdown of the normalized total overhead execution times.

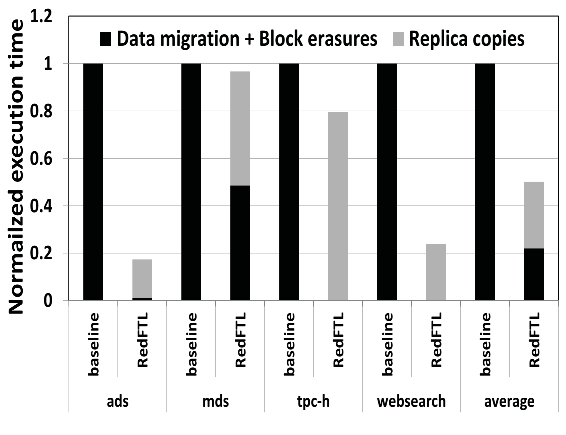


Figure 6: A comparison of the normalized overhead execution times for read reclaim.

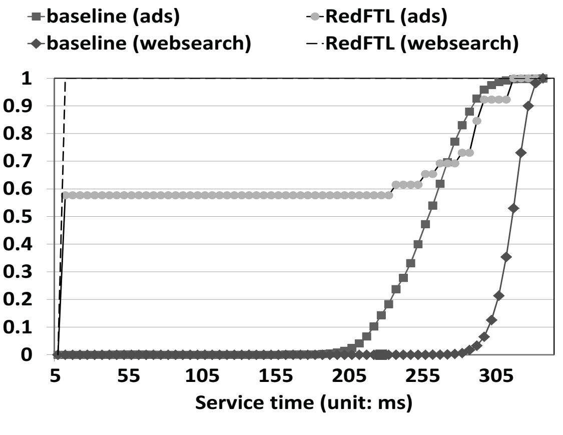


Figure 8: CDFs of service times of RR for `ads` and `websearch` in RedFTL and `baseline`.

and tpc-h when 5 pages per block are allowed to store replica pages. A small number of replica pages do not service many read requests. Furthermore, GC may be frequently activated because more many blocks are required to store the same number of replica pages as the number of replica pages per block decreases. Although creating replica pages increased total overhead execution time in the two cases, RedFTL reduced the total overhead execution time for GC and RR, on average, by 34% over baseline by reducing the number of RR occurrence.

Fig. 8 shows the cumulative distribution functions (CDFs) of the service times of RR procedures for ads and websearch. In both ads and websearch cases, the proactive data migrations of RedFTL limited most service times less than 10 ms. On the other hand, since baseline does not activate RR until the number of reads reaches the RR threshold, most of the valid pages in a disturbed block were simultaneously moved in our experiments, with the peak response time occurring around 332 ms.

5 Conclusion

We have proposed a novel read disturb-management technique, which can reduce overheads from frequent read reclaim procedures in high-density NAND flash memory. Our new technique distributes frequently-accessed pages in a small number of blocks to multiple blocks, thus significantly reducing the frequency of read reclaim procedures. Moreover, our technique spreads the time overhead of data migrations during read reclaim by migrating frequently-read pages early. Experimental results show that our proposed technique can reduce the overhead execution time for read reclaim by 50% over an existing read-disturb management technique. Our results demonstrate that proactively distributing read requests can be an effective method in managing the read-disturb problem.

Our technique can be extended into several directions. For example, the read-hot page separator of RedFTL can be further improved for more accurate read data separation. Furthermore, our read-hot page separator needs to be more tightly integrated with a write data separator of a typical FTL. Finally, In order to understand real benefit of our technique, we also plan to evaluate RedFTL on a real storage system based on sub-20 nm NAND flash memory.

6 Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education, Science and Technology (MEST) (No. 2012-0006417). The ICT at Seoul National University and IDEC provided research facilities for this study.

References

[1] http://iotta.snia.org/traces/158.

[2] http://traces.cs.umass.edu/index.php/Storage/Storage.

[3] A. A. CHIEN ET AL. Moore's Law: The First Ending and A New Beginning. Tech. rep., 2012.

[4] D. NARAYANAN ET AL. Write Off-Loading: Practical Power Management for Enterprise Storage. *ACM Transactions on Storage 4*, 3 (2008), 1–23.

[5] H. H. FROST ET AL. Efficient Reduction of Read Disturb Errors in NAND Flash Memory, 2010. US Patent 7,818,525.

[6] J. ZHANG ET AL. Synthesizing Representative I/O Workloads for TPC-H. In *Proc. of the International Symposium on High Performance Computer Architecture* (2004).

[7] M. KANG ET AL. Improving Read Disturb Characteristics by Self-Boosting Read Scheme for Multilevel NAND Flash Memories. *Japanese Journal of Applied Physics 48*, 4 (2009), 04C062–1–04C062–6.

[8] S.H. SHIN ET AL. A New 3-bit Programming Algorithm Using SLC-to-TLC Migration for 8MB/s High Performance TLC NAND Flash Memory. In *Proc. of the IEEE Symposium on VLSI Circuits* (2012).