Efficient Read Disturb Management Schemes in Resource-constrained Flash Memory Controller

Ikjoon Son and Jin-Soo Kim
Dept. Computer Science and Engineering
Seoul National University
{ikjoon.son, jinsoo.kim}@snu.ac.kr

***Abstract*—Whenever a read operation is performed in a flash memory storage device, the surrounding cells in the same block are affected and their reliability gradually decreases. This phenomenon is known as *read disturb* problem. A remedy to this read disturb is to count the number of read operations for each block and perform *read reclaim*, the process of moving the existing data to a new block when the count reaches a predefined threshold. However, as the number of blocks in the flash memory device increases, maintaining the per-block read count requires a great amount of memory space in the controller. Therefore, system designers are forced to maintain read counts for a group of blocks (i.e. *superblock*) which results in the loss of accuracy.**

This paper proposes novel read disturb management schemes that can reduce the number of read reclaims significantly even if the read count is maintained for each superblock. In the proposed *Pointer-based* scheme, we keep track of the last block read so as to avoid excessive increase in the read count when the data is read sequentially. We also propose the *Bitmap-based* scheme that can successfully approximate the actual read count in the presence of random reads, with a negligible space overhead. Our experiments with real-world traces show that the Pointer-based scheme and the Bitmap-based scheme reduce the number of read reclaims by 65.5% and 90.5% on average, respectively, compared to the conventional scheme.

***Index Terms*—Read disturb, Read reclaim, NAND flash memory, SSDs (Solid State Drives)**

I. Introduction

For a long time, computer systems have relied on magnetic HDDs (Hard Disk Drives) as long-term non-volatile storage devices. During the past couple of decades, however, the market is being replaced rapidly with SSDs (Solid State Drives) based on NAND flash memory technology.

NAND flash memory has increased the performance of storage devices dramatically, but also introduced new problems. In addition to the basic flash management functionalities such as address mapping and garbage collection, flash memory controllers have been constantly striving to meet ever-increasing high reliability requirements as non-volatile devices. For example, in modern flash memory controllers, it is considered essential to use strong ECCs (Error Correction Codes) such as BCH [1] and LDPC [2] to recover from various errors in NAND flash memory.

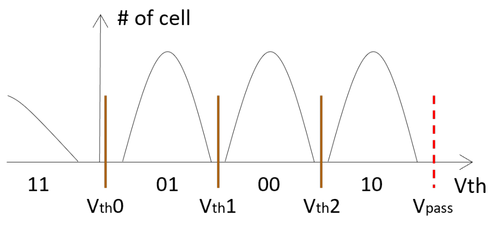
In particular, *read disturb* is one of the significant factors that affects the reliability of NAND flash memory. It is the well-known characteristic of NAND flash memory that the repeated read operations to the same block widen the state distributions of nearby cells [3], [4]. This read disturb problem not only increases the error rate in the surrounding pages, but also makes it difficult to detect in real-time. Also, it is challenging to predict which block is more affected than others.

A classic approach to cope with the read disturb stress is to track the number of read operations for each block and moves all the valid data within the block to another block when a certain threshold is reached. This is called a *read reclaim* operation. However, today's flash-based storage devices are composed of multiple dies and each die in turn has hundreds or thousands of blocks. Therefore, maintaining the per-block read count requires a great amount of memory space in the controller. Specifically, the controller needs to manage all firmware code and data in SRAM whose size is usually about 1MiB in mobile storage devices or DRAM-less SSDs. Even for the SSDs that are equipped with a DRAM, most of the DRAM capacity is devoted to the address mapping table, leaving only a several tens of Kilobytes of memory space for storing per-block read count.

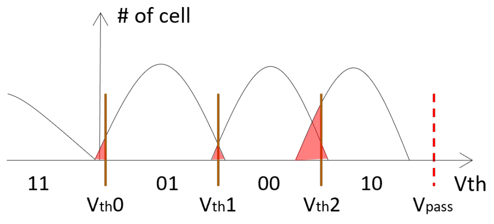
One way to solve this problem is to maintain the read counts in a more coarse-grained way by managing a single read count for multiple blocks. Usually, flash-based storage devices utilize the *superblock* technique, where multiple blocks across different dies and planes are grouped as one large logical block to exploit parallelism during read/write operations and garbage collection [5]. Therefore, it is conventional to maintain a single read count for each superblock.

Unfortunately, the conventional approach with the per-superblock read count has a number of problems. Let us assume that T denotes the per-block threshold value that initiates a read reclaim operation. If we increase the threshold to $n \times T$ where n is the number of blocks in each superblock, there is no way to prevent the read disturb problem from taking place when only a single block is repeatedly accessed. Inevitably, if we apply the same threshold T for each superblock, it will overestimate the actual read count when the blocks within a superblock are sequentially accessed, causing premature read reclaim operations.

Excessive read reclaim operation causes performance degradation due to massive data copy operations. In particular, in situations where read operations are intensive to reach the limit of read disturbance, the program operation for reclaim causes QoS (Quality of Service) degradation due to long tail latency. Excessive reclaim also increases the number of block erasures,



(a) Initial program



(b) After degradation

Fig. 1: Cell distribution over time (Red areas represent erroneous cells)

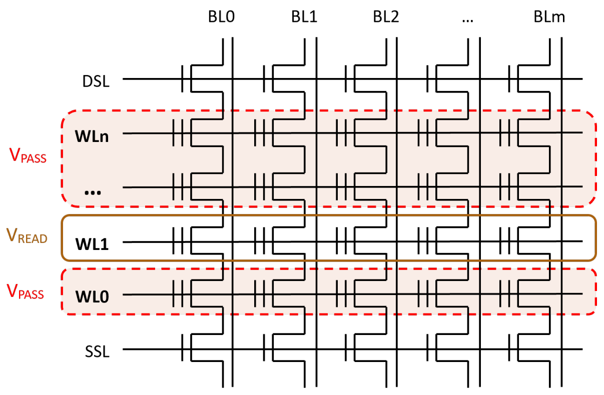


Fig. 2: Applied voltages when reading WL1

which reduces the lifetime of the entire device. Unnecessary block erasure also reduces the lifetime of the entire device. As the data density of NAND increases, the limit on the number of erase operations is steadily decreasing. Therefore, it is more important to minimize the number of block erasures.

This paper presents novel read disturb management schemes that can solve the aforementioned problems. Our goal is to make the per-superblock read count closely track the worst per-block read count within a superblock as much as possible with negligible additional memory overhead. To this end, we devise sophisticated strategies that manipulate the per-superblock read count by identifying the patterns of read operations performed in each superblock.

First, we propose the *Pointer-based* scheme where we keep track of the last block read to avoid excessive increase in the per-superblock read count when the data is read sequentially. We also propose the *Bitmap-based* scheme that can show better performance in the presence of random reads. Throughout comprehensive trace-driven simulations with six real-world traces, we show that the Pointer-based scheme and the Bitmap-based scheme reduce the number of read reclaims by 65.5% and 90.5% on average, respectively, compared to the conventional scheme. The additional memory space required for the Bitmap-based scheme is only about 7KiB for a 1TiB flash storage device.

II. Background & Related Work

NAND flash memory stores data in a group of cells having a threshold voltage (V_{th}) set to a specific level. Setting up the V_{th} of cells is called a *program* operation while reading data according to the V_{th} of cells is called a *read* operation. The unit of the read and program operation is a page that usually consists of hundreds of thousands of cells. To store new data in the programmed page, the erase operation must be preceded for a block containing the page. Typically, each block contains hundreds of pages. By using multiple levels of V_{th}'s, each cell can store more than one bit of information. NAND flash memory is classified from SLC (Single Level Cell) to QLC (Quadruple Level Cell) based on the number of bits each cell represents.

The V_{th} of the programmed cell changes over time due to various factors. Typical examples include *program disturb* that occurs while programming a page, *read disturb* while reading a page, and *retention errors* caused by charge leakage over time [6]. Although each factor has slightly different characteristics, they generally widen the cell distribution, which makes it challenging to identify the state of each cell, as illustrated in Fig. 1.

When a read operation is performed on a page, the read disturb occurs in the remaining pages of the block. This is because, in order to read the data of a specific wordline, a high voltage, V_{pass}, should be applied to the rest of the wordline as shown in Fig. 2. Because it is difficult to precisely measure the amount of disturb applied to nearby cells in real-time, most flash controllers indirectly use the number of read operations per block as a metric. When the read count reaches the threshold preset by the NAND manufacturer, the controller performs a read reclaim operation that moves the data into another block.

There are several studies that aim to reduce the overhead caused by the read reclaim operation. Read leveling [7] suppresses the intensive increase in the read count of a specific block by scattering read-hot data into multiple blocks. This approach can reduce the number of read reclaim operations due to read-hot data. However, read-hot data is also considered as write-cold data by the controller since it is not updated frequently. Distributing such write-cold data into multiple blocks can increase the garbage collection cost.

In order to reduce the effect of read disturb, RedFTL [8], [9] proposes to lower the read voltage through a fine-grained programming for such blocks that are expected to receive high read stress. Also, RedFTL tries to distribute data to LSB/CSB/MSB pages according to its hotness. However, RedFTL necessitates a 1-byte counter for each LBA (Logical Block Address), which would require a significant cost to be adopted in modern flash-based storage devices.

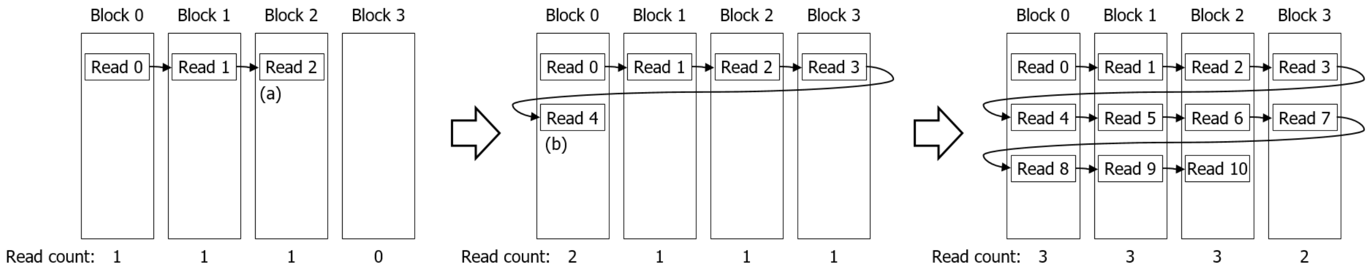


Fig. 3: Changes in read counts during sequential read operations

Cai et al. propose an adaptive scheme that determines whether a read reclaim is actually needed by performing the *test reads* at regular intervals after the read count reaches the threshold [10]. However, it is challenging to determine the right timing and the location of test read operations because there are pages that are more vulnerable due to the manufacturing process or whether the surrounding pages are programmed [11], and the error rate increases exponentially [12]. In addition, this scheme also requires the read count management, so it can be used in combination with the schemes proposed in this paper.

III. Design

A. Baseline schemes

In this paper, we consider two baseline schemes based on the granularity the read count is managed. In the *Ideal scheme*, an individual block keeps track of the number of read operations performed in the block. On the other hand, the controller maintains a single read count for each superblock in the *Conventional scheme*. For both schemes, we assume that the threshold for initiating the read reclaim operation is conservatively set to the same value.

Note that the Ideal scheme minimizes the number of read reclaims due to the fine-grained per-block read count, but it is impractical because of the enormous memory requirement. Compared to the Ideal scheme, the Conventional scheme can reduce the memory overhead by a factor of n where n is the number of blocks within a superblock. However, the Conventional scheme suffers from the fact that the per-superblock read count alone cannot properly reflect the actual states of individual blocks.

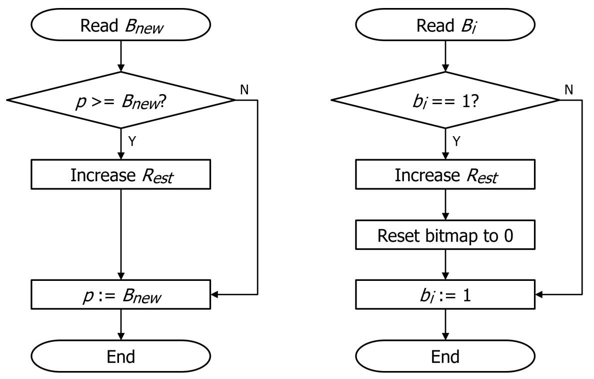
Assume that B_H indicates the block that receives the highest number of read operations within a certain superblock. In this case, we define the read count of the block B_H as the *effective read count* or $\bar{R}$ of the superblock. On the contrary, we define the per-superblock read count as the *estimated read count* or R_{est}. For the Ideal scheme, R_{est} is always equivalent to $\bar{R}$. However, depending on the read patterns, R_{est} can be much higher than $\bar{R}$ in the Conventional scheme.

B. Pointer-based scheme

The *Pointer-based scheme* is inspired by the observation that R_{est} becomes significantly higher than $\bar{R}$ when the blocks within a superblock are sequentially read in the Conventional scheme. Fig. 3 shows an example of this scenario. Usually the data to be read is striped across multiple blocks in order to increase the parallelism during write and read operations. Therefore, while a sequential read operation is performed, the controller repeats the following phases: (a) read the data from block i and moves to next block $i+1$, and (b) if the current block is the last block, moves to the first block in the superblock. In this example, the read count of an individual block is incremented only when one of pages in the block is read. However, the Conventional scheme simply increases R_{est} on every read operation to the superblock. After reading 11 pages sequentially in Fig. 3, $\bar{R}$ becomes only 3 while R_{est} becomes 11 in the Conventional scheme. Reading back a bunch of data from storage sequentially in the order it was written is a very common behavior, so the overestimation similar to this example may occur under normal circumstances.

The proposed scheme introduces a *pointer* p for each superblock in order to detect this type of read pattern. The pointer p is used to refer to the block where the latest read operation is performed. When a new read operation arrives, the previous block B_{last} pointed to by p is compared with the block B_{new} that will serve the new read operation. If the block number of B_{new} is larger than that of B_{last}, we do not increment R_{est} anticipating that the new read operation is likely to access the same stripe as the previous one. In other case where B_{new}'s block number is smaller than or equal to B_{last}'s, we increment R_{est}. If B_{new} is B_H, p always points to a block that is greater than or equal to the B_H. So when the next read occurs in B_H and $\bar{R}$ is incremented, it is guaranteed that R_{est} will also be incremented and no underestimation will occur. Note that it is important to increment R_{est} when the new request is directed to the same block where the last read has performed.

Now we estimate the memory overhead for the Pointer-based scheme. In this scheme, all we need besides R_{est} is a pointer. A single-byte pointer can manage up to 256 blocks. It is considered not a good practice to increase the superblock size beyond 256 blocks because the use of large superblocks brings inefficiency in terms of buffer management, garbage collection, and address mapping. Also, the larger the superblock size gets, the greater the discrepancy bewteen R_{est} and $\bar{R}$. Therefore, the one-byte pointer is sufficient to cover most of the superblock configurations used in real devices. Additionally, the packing of R_{est} and pointer into a single



(a) Pointer-based scheme (b) Bitmap-based scheme

Fig. 4: Flow charts of the proposed schemes

32-bit word can be considered. Since the read count is one of the most frequently updated counters, even though 3 bytes are sufficient to manage it until the threshold is reached, it is sometimes preferred to store the read count in a 4-byte array to avoid word-unaligned memory accesses. In this case, the Pointer-based scheme using single-byte pointers has virtually no additional memory overhead compared to the Conventional scheme.

C. Bitmap-based scheme

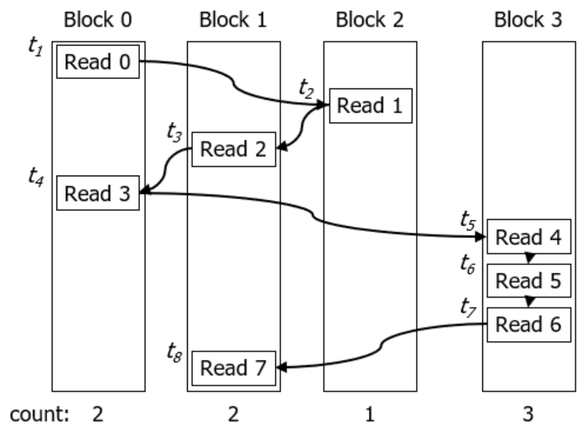
The Pointer-based scheme is effective in reducing the gap between R_{est} and $\bar{R}$ for sequential or strided read access pattern. However, although it performs better than the Conventional scheme, it still significantly overestimates $\bar{R}$ when the blocks are randomly accessed. In order to further reduce the gap between R_{est} and $\bar{R}$ in the presence of random read access pattern, we propose a novel Bitmap-based scheme.

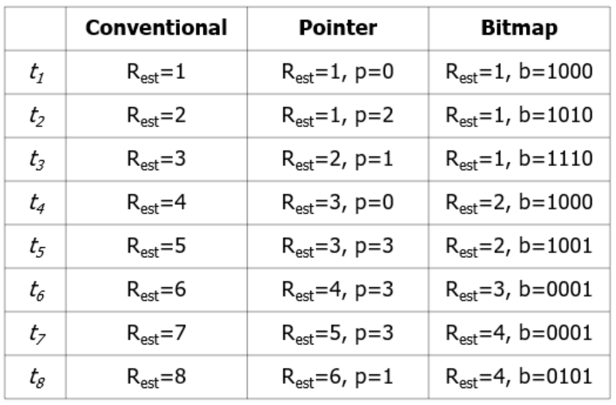
The idea of the Bitmap-based scheme is to replace the pointer in the Pointer-based scheme with the *bitmap* information for all the blocks in a superblock. Each bit b_i in the bitmap corresponds to the block B_i and represents the current state of the block B_i. If b_i is 0, it means that there was no read access recently in the corresponding block B_i. Whenever there is a new read access to the block B_i whose bitmap b_i is 0, we set b_i to 1 to denote that there was a read access to the block B_i. On the other hand, if the corresponding bitmap b_i is already 1, it indicates that the block B_i is receiving two read accesses in a short period time. In this case, we increment R_{est} by one and clear the bitmap for the rest of the blocks.

The rationale behind the Bitmap-based scheme is to record the read access history in the bitmap until one of the blocks receives two read accesses. If the block B_i is accessed twice, it is the time to increment R_{est}. At the same time, we clear the history for other blocks because the previous history is already reflected to R_{est} by the block B_i. Fig. 4 compares the overall flow charts of the Pointer-based scheme and the Bitmap-based scheme.

For the sequential read pattern, the Bitmap-based scheme works just like the Pointer-based scheme, resulting in the same value of R_{est}. In case of the random read pattern, the Bitmap-based scheme can prevent excessive increase in R_{est} by absorbing other read accesses until the same block is accessed twice. Note that when the same block is continuously accessed, R_{est} should be incremented continuously as is done in the Pointer-based scheme. The Bitmap-based scheme achieves this by leaving the bit for the last read block as 1 at the moment R_{est} is incremented, instead of clearing the entire bitmap. In addition, when a superblock is erased due to garbage collection, we initialize R_{est} to 0, but we set all the bits of the corresponding bitmap to 1 so that any subsequent read access can increment R_{est} immediately.

If the size of the superblock is less than or equal to 8 blocks, we can accommodate the bitmap information into the leftmost byte of the read count, in the same way as the Pointer-based scheme. If the superblock size exceeds 8 blocks, we need to allocate another $s \times (n - 8)$ bits in the metadata area where s and n denote the number of superblocks and the number of blocks within a superblock, respectively. Overall, the additional memory overhead for implementing the Bitmap-based scheme is merely about 7KiB for a 1TiB flash storage device. The detailed memory overhead is analyzed in more detail in Section IV-B.





	Conventional	Pointer	Bitmap
t_1	R_{est}=1	R_{est}=1, p=0	R_{est}=1, b=1000
t_2	R_{est}=2	R_{est}=1, p=2	R_{est}=1, b=1010
t_3	R_{est}=3	R_{est}=2, p=1	R_{est}=1, b=1110
t_4	R_{est}=4	R_{est}=3, p=0	R_{est}=2, b=1000
t_5	R_{est}=5	R_{est}=3, p=3	R_{est}=2, b=1001
t_6	R_{est}=6	R_{est}=4, p=3	R_{est}=3, b=0001
t_7	R_{est}=7	R_{est}=5, p=3	R_{est}=4, b=0001
t_8	R_{est}=8	R_{est}=6, p=1	R_{est}=4, b=0101

Fig. 5: Changes in read counts during random read operations

TABLE I: Default simulation parameters

NAND Configuration	
Planes	4
Physical blocks per plane	875
Logical wordlines	400 (512Gb NAND), 800 (1Tb NAND)
Cell type	TLC
Page size	16KiB
Threshold for read reclaim	100,000
SSD Configuration	
Over-provisioning (OP)	7%
Mapping unit	4KiB (page mapping)
Garbage collection	Greedy

Fig. 5 compares how the read count is managed in the Conventional scheme, Pointer-based scheme, and Bitmap-based scheme, when the controller reads 8 pages as shown in the left. In this scenario, $\bar{R}$ = 3 as the block 3 has performed three read operations. The Conventional scheme estimates the read count as 8 because it simply increments the counter for every read operation. We can see that R_{est} of the Bitmap-based scheme is closest to $\bar{R}$.

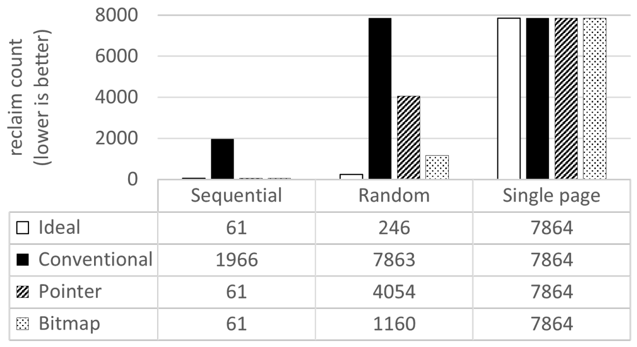
IV. Evaluation

A. Experimental Setup

We have built an in-house trace-driven SSD simulator for evaluating the proposed schemes. By default, we assume a 512GiB flash-based storage device consisting of eight 512Gb TLC NAND dies with 4 planes. All the blocks across 8 dies and 4 planes form a superblock. Therefore, one superblock is composed of 32 flash blocks. The input traces are replayed using a page-mapped FTL with the greedy garbage collection policy. The threshold value for initiating the read reclaim operation is set to 100K. Before each test, the entire LBA range was filled sequentially for preconditioning. The important simulation parameters are summarized in Table I.

B. Synthetic Workloads

First, we use three synthetic workloads: fully sequential reads, fully random reads, and single page reads. Each workload performs 3TiB of reads on a 1GiB area of the device. In single page reads, the same 4KiB data is repeatedly read 756M times. The total number of read reclaim operations performed for each workload is depicted in Fig. 6.



	Sequential	Random	Single page
□ Ideal	61	246	7864
■ Conventional	1966	7863	7864
▨ Pointer	61	4054	7864
▩ Bitmap	61	1160	7864

Fig. 6: Results of synthetic workloads

In fully sequential reads, the read reclaim count of the Conventional scheme is higher than that of other schemes by 32x. This is because each superblock consists of 32 blocks. On the other hand, we can see that the Pointer-based scheme and Bitmap-based scheme show the same number of read reclaim count as the Ideal scheme, even if they reduce the memory overhead by a factor of 16∼32.

In fully random reads, the Ideal scheme increments R_{est} by one for every 32 reads on average. Therefore, as in fully sequential reads, the read reclaim count of the Conventional scheme is higher than that of the Ideal scheme by 32 times. Compared to the Conventional scheme, the Pointer-based scheme reduces the reclaim count by almost half due to its strategy that R_{est} is not incremented when the next read access is going to the higher-numbered block. The Bitmap-based scheme improves the reclaim count by 85.2% compared to the Conventional scheme. Although the reclaim count of the Bitmap-based scheme is still larger than that of the Ideal scheme by 4.7 times, the Bitmap-based scheme only requires 7KiB of memory while the Ideal scheme requires 112KiB to store read counts (cf. Section IV-D).

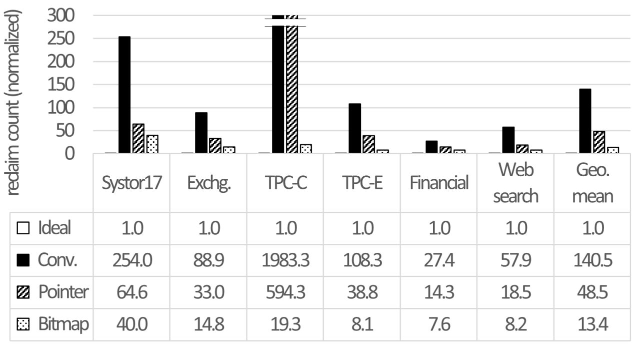
C. Real-world Workloads

We also evaluate the proposed schemes using real-world workload traces. The traces used in this paper are summarized in Table II. SYSTOR17 is the enterprise storage trace on commercial office VDI environment used in [13]. EXCHANGE, TPC-C, and TPC-E are the Microsoft Enterprise traces available from SNIA IOTTA repository [14]. FINANCIAL is the OLTP application trace running at two large financial institutions and WEBSEARCH is the block I/O trace of a popular search engine, both of which are available from UMass Storage Trace Repository [15].

For the workloads whose traces are obtained from multiple disks or LUNs, we have used the trace of a single storage unit that has the largest amount of read requests. Since it is required to simulate 1TiB or 8TiB SSDs to run the traces, we assume that 1TiB SSD is composed of 16 512Gb dies with 4 planes while 8TiB SSD is 64 1Tb dies with 4 planes. Therefore, the superblock size is 64 blocks for 1TiB SSD and 256 blocks for 8TiB SSD. We have also repeated each trace 10∼300 times as shown in Table II, because the amount of reads is not sufficient to compare the number of read reclaims among different scheme. Fig. 7 compares the normalized read reclaim counts of the traces.

TABLE II: Traces used in this paper (The last column represents the proportion of requests larger than 16KiB)

Workload	**Description**	**SSD Size**	**Repeat**	**R >16K**
SYSTOR17	Office VDI	8 TiB	10	81.1%
EXCHANGE	MS Exchange server	1 TiB	100	18.6%
TPC-C	OLTP benchmark	1 TiB	100	4.1%
TPC-E	OLTP benchmark	1 TiB	100	0.0%
FINANCIAL	OLTP application	1 TiB	300	12.9%
WEBSEARCH	Search engine	1 TiB	300	53.6%



	Systor17	Exchg.	TPC-C	TPC-E	Financial	Web search	Geo. mean
□ Ideal	1.0	1.0	1.0	1.0	1.0	1.0	1.0
■ Conv.	254.0	88.9	1983.3	108.3	27.4	57.9	140.5
▨ Pointer	64.6	33.0	594.3	38.8	14.3	18.5	48.5
⊡ Bitmap	40.0	14.8	19.3	8.1	7.6	8.2	13.4

Fig. 7: Results of real-world workloads

First, we can observe that the difference between the Ideal scheme and the Conventional scheme is largely affected by the superblock size except for TPC-C. In TPC-C, a large amount of data is periodically overwritten and many blocks are garbage collected before they reach the reclaim threshold in the Ideal scheme, which relatively increase the reclaim counts of other schemes.

The Pointer-based scheme considerably improves the reclaim counts in all traces compared to the Conventional scheme. This is because of the sequentiality that exists in each trace. We can observe that the size of about 81.1% of read requests are larger than 16KiB in SYSTOR17. So the larger the read size, the slower R_{est} will be incremented in the Pointer-based scheme compared to the Conventional scheme.

The sequentiality in the traces is also helpful for the Bitmap-based scheme. Additionally, the Bitmap-based scheme is more effective in mitigating the increase of R_{est} in workloads that are predominantly composed of small random reads such as TPC-C and TPC-E. They primarily comprise 95.9% and 100.0% of small-sized reads, respectively, which leads to a notable enhancement in reclaim count with the Bitmap-based scheme outperforming the Pointer-based scheme by 30.8x and 4.8x, respectively.

D. Memory Efficiency

The amount of memory needed to maintain the read count is summarized in Table III for the assumed configurations of 512GiB, 1TiB, and 8TiB SSDs. For the 8TiB SSD, the Ideal scheme requires the memory close to 1MiB, which is difficult to accommodate even on an SSD with DRAM. The Conventional scheme can be implemented with a much smaller table, but the reclaim count increases too fast. As a compromise, the Pointer-based scheme requires only less than 1KiB of additional memory than the Conventional scheme, but shows up to 3.9x improvement in the reclaim count. The Bitmap-based scheme further reduces the reclaim count by adding only 28.0KiB for maintaining the bitmap information in the 8TiB SSD.

V. Conclusion

In flash-based storage devices, it is essential to manage read disturb to ensure data reliability. This paper proposes the Pointer-based scheme and the Bitmap-based scheme that can effectively prevent read disturb with low memory usage. The efficiency of both schemes varies depending on the workload characteristics. The Pointer-based scheme is useful when the workload has a large amount of sequential reads, while the Bitmap-based scheme further reduces read reclaims when the workload has random reads, with adding only a negligible amount of memory.

TABLE III: Memory overhead

SSD Capacity	**512 GiB**	**1 TiB**	**8 TiB**
NAND Die Size	512 Gb	512 Gb	1 Tb
Physical blocks per superblock	32	64	256
Ideal scheme	112.0 KiB	224.0 KiB	896.0 KiB
Conventional scheme	3.5 KiB	3.5 KiB	3.5 KiB
Pointer-based scheme	4.4 KiB	4.4 KiB	4.4 KiB
Bitmap-based scheme	7.0 KiB	10.5 KiB	31.5 KiB

Acknowledgment

We would like to thank our anonymous reviewers for their invaluable feedback. This work was supported by the National Research Foundation of Korea (NRF) grant (No. 2019R1A2C2089773), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant (No. IITP-2021-0-01363) funded by the Korea government (MSIT).

References

[1] R. C. Bose and D. K. Ray-Chaudhuri, “On a class of error correcting binary group codes,” *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.
[2] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.
[3] M. Kato *et al.*, “Read-disturb degradation mechanism due to electron trapping in the tunnel oxide for low-voltage flash memories,” in *Proceedings of 1994 IEEE International Electron Devices Meeting*, pp. 45–48, IEEE, 1994.
[4] N. Papandreou, H. Pozidis, T. Parnell, N. Ioannou, R. Pletka, S. Tomic, P. Breen, G. Tressler, A. Fry, and T. Fisher, “Characterization and analysis of bit errors in 3d tlc nand flash memory,” in *2019 IEEE International Reliability Physics Symposium (IRPS)*, pp. 1–6, IEEE, 2019.
[5] M. Asnaashari, A. Chen, S. Nemazie, and D. P. McNamara, “Memory super block allocation,” June 10 2014. US Patent 8,751,731.
[6] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, “Data retention in mlc nand flash memory: Characterization, optimization, and recovery,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 551–563, IEEE, 2015.
[7] C.-Y. Liu, Y.-M. Chang, and Y.-H. Chang, “Read leveling for flash storage systems,” in *Proceedings of the 8th ACM International Systems and Storage Conference*, pp. 1–10, 2015.
[8] K. Ha, J. Jeong, and J. Kim, “A read-disturb management technique for high-density nand flash memory,” in *Proceedings of the 4th Asia-Pacific Workshop on Systems*, pp. 1–6, 2013.
[9] K. Ha, J. Jeong, and J. Kim, “An integrated approach for managing read disturbs in high-density nand flash memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 7, pp. 1079–1091, 2015.
[10] Y. Cai, F. Zhang, J. Lee, and H. Li, “Adaptive read disturb reclaim policy,” Oct. 23 2018. US Patent 10,108,472.
[11] N. Papandreou *et al.*, “Effect of read disturb on incomplete blocks in mlc nand flash arrays,” in *2016 IEEE 8th International Memory Workshop (IMW)*, pp. 1–4, IEEE, 2016.
[12] L. M. Grupp *et al.*, “Characterizing flash memory: anomalies, observations, and applications,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 24–33, 2009.
[13] C. Lee *et al.*, “Understanding storage traffic characteristics on enterprise virtual desktop infrastructure,” in *Proceedings of the 10th ACM International Systems and Storage Conference*, pp. 1–11, 2017.
[14] SNIA IOTTA repository, http://iotta.snia.org.
[15] UMass trace repository, http://traces.cs.umass.edu.