Exploiting Latency Variation for Access Conflict Reduction of NAND Flash Memory

Jinhua Cui*, Weiguo Wu*, Xingjun Zhang*, Jianhang Huang*, Yinfeng Wang†
*Xi'an Jiaotong University
cjhnicole@gmail.com, {wgwu, xjzhang}@mail.xjtu.edu.cn, huangjhsx@gmail.com
†ShenZhen Institute of Information Technology
wangyf@mailst.xjtu.edu.cn

***Abstract*—NAND flash memory has been widely used in storage systems by offering greater read/write performance and lower power consumption than mechanical hard drives. Recently, the tradeoff between endurance, write speed, and read speed has been exploited from many ways for I/O performance improvement, which also induce the read/write latency variation. In this paper, the latency variation is exploited in I/O scheduling for access characteristic guided read and write latency minimization. First, with the understanding of the relationship among read latency, write latency and raw bit error rates (RBER), different ways to exploit the relationship for read and write latency reduction is discussed. Then, an I/O scheduling scheme is proposed by using hotness and retention age of accessed data to determine the speed of writes or reads, giving scheduling priority to fast writes and fast reads for conflict reduction. Experiments with various traces reveal that the proposed technique achieves significant read and write performance improvements.**

I. Introduction

NAND flash-based solid-state drives (SSDs) have gained tremendous popularity in data centers and personal computers by offering several advantages over hard disk drives (HDDs), including higher random access performance, lower power consumption, shock resistance and lack of noise [1]. Over the past decade, the capacity of NAND flash memory has been increasing continuously, as a result of technology scaling from 65nm to the latest 10nm technology and the bit density improvement from 1 bit per cell to the latest 6 bits per cell [2, 3]. Unfortunately, as flash density increases, NAND flash reliability is significantly degraded by several sources of errors including retention noise, read disturbance noise, cell-to-cell program interference noise and program/erase (P/E) cycling noise, which brings in new challenges [4]. On one hand, write speeds are decreased to compensate for the lifetime reductions and raw bit error rates (RBER) growths. On the other hand, stronger error correction capability driven by more complex Error Correction Code (ECC) schemes such as Low-Density Parity-Check Code (LDPC) would increase access latency (read/write/erase operations) remarkably. Thus, the search for methods to improve flash I/O performance is motivated.

A promising technique for I/O performance improvement is to take the tradeoff between RBER, write speed, and read speed into consideration. Flash read speed is highly correlated with the RBER of flash memory cell. The higher the RBER, the stronger the required ECC capability, as well as the higher the complexity of ECC scheme and the slower the read requests. Besides, there is a close relationship between RBER and the speed of write operations. Several works [7, 8, 9] point out that using a smaller program step size ΔV_p of the incremental step pulse programming (ISPP) scheme, which gradually accumulates voltage to transcend the threshold voltage, would decrease RBER at the cost of write speed degradation. Therefore, of the three objectives: low RBER, fast read, and fast write, any two can be optimized at the cost of compromising the third.

Specifically, increasingly significant process variation (PV) observed in flash memory [5, 6, 7, 24] makes this tradeoff tunable separately for different blocks in flash memory. PV is the phenomenon that pages in different memory blocks may have largely different worst-case RBER under the same P/E cycling, equivalent to largely different P/E cycling endurance within different memory blocks when given the same ECC. It is caused by significant variability of oxide thickness and gate width/length, which continues to increase with the technology scaling. Thus, in making the described tradeoff, approaches can either assume worst-case block behavior, or optimize separately for different block strengths.

Up to now, much effort has been invested in attempts to exploit the tradeoff for accelerating performance, which can be categorized into three groups. The first group of work is about taking advantage of PV. The sole existing work to date is presented by Shi et al. [7], using coarser ΔV_p for strong pages which do not accumulate errors as fast and allocating hot data to them. The second group is about adapting cell programming/read-out parameters based on data retention age – the length of time since a flash cell was programmed. Lower read-out thresholds can be applied as the actual age of the data increases [4], while higher programming voltages [22] or finer ΔV_p [20] can be used when the estimated retention time is higher. Another group is to identify read-only/write-only pages so that read and write operations can be speed up dependently with less effect between each other [23]. When exploiting the described tradeoff for improved I/O performance, the requests are only partially accelerated such as read/write hot requests, which inevitably lead to the significant read and write speed

978-1-4673-9055-2/16/$31.00

variation and hence motivate the search for methods to exploit it for further optimization.

In this paper, we propose a retention-aware and hotness-aware I/O scheduling algorithm (RHIO) for NAND flash memory. Our key insight is that the speed variation induced by tradeoff-aware techniques can be exploited for maximal benefit by giving scheduling priority to fast writes and fast reads. First, in order to amplify the benefit brought by strong blocks, hot writes are scheduled in priority, where the data is allocated to stronger blocks with low write latency. Second, read requests are sorted based on the actual retention age of accessed data, and hot reads with low read latency are also preferentially scheduled. Thus, employing RHIO can achieve dramatic I/O performance improvement by using three techniques including PV-aware write speed regulation, retention-aware read speed regulation, and shortest-job-first scheduling.

We evaluated the effectiveness of the proposed algorithm through representative workload traces and trace-based simulations, and results show that RHIO can achieve significant performance improvement by 39.11% and 29.92% for read and write requests, on average. Furthermore, by collecting the percentages of prioritized write and read requests with RHIO scheduler, results clearly demonstrate the effectiveness of RHIO in reducing I/O latency especially for read and write intensive applications.

The rest of this paper is organized as follows. Section II presents the background and related work. Section III describes the design techniques and implementation issues of our I/O scheduling scheme for flash storage devices. Experiments and result analysis are presented in Section IV. Section V concludes this paper with a summary of our findings.

II. Background and Related Work

In this section, we first pursue a better understanding of the tradeoff between RBER, read speed and write speed. Then, previous studies related to this tradeoff are introduced for further work in this area.

A. Tradeoff between RBER, Write Speed and Read Speed

The tradeoff between RBER, write speed, and read speed is due to two relationships: 1) ECC complexity, the error correction capability and read speed. 2) RBER, the program step size and write speed. In this work, we take LDPC as the default ECC scheme, which brings superior error correction capability as well as read performance degradation at the same time.

The first relationship is due to soft-decision memory sensing, which uses more than one quantization levels between two adjacent storage states [9]. On the one hand, as the number of quantization levels used between two adjacent storage states increases, the read operations which aim to sense and digitally quantize the threshold voltage of each memory cell are delayed. On the other hand, the number of sensing levels also affects the error correction strength of LDPC code decoding. More sensing levels mean a preciser memory sensing in the context of NAND flash memory, leading to more accurate input probability information of each bit for LDPC code decoding, which improves its error correction capability. Therefore, the tradeoff between error correction capability and read speed can be explored.

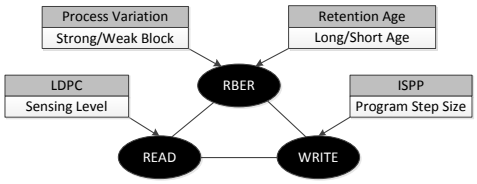

Fig. 1. Tradeoff between RBER, read speed and write speed.

The second relationship is due to the ISPP scheme. To program data into flash memory, the ISPP scheme is commonly applied, which uses Fowler-Nordheim (FN) tunneling to increase the V_{th} of flash memory cells by a certain step size ΔV_p, where ΔV_p directly affects write speed and RBER. On the one hand, larger ΔV_p means less steps to the desired level, thereby, resulting in shorter write latency. On the other hand, the margin for tolerating retention errors is reduced as ΔV_p gets larger, leading to higher RBER. Therefore, the tradeoff between RBER and write speed can be studied.

Based on the precondition that RBER should be within the error correction capability of the deployed LDPC code, the tradeoff between RBER, read speed and write speed can be concluded from above two relationships. For blocks with lower RBER, either read or write speed can be improved, and vice versa. In this work, we focus on the read and write speed variation induced by such tradeoff.

B. Related Work

Several methods for improving I/O performance have been suggested for exploiting the tradeoff between RBER, read speed and write speed. They include making full use of strong blocks induced by PV, taking the data with short retention time requirement into account, and regulating program step size and sensing level based on access characteristics. Figure 1 shows the multiple ways for exploiting the tradeoff. To the best of our knowledge, this is the first presentation of such a diagram with the utilization of the tradeoff between RBER, write speed, and read speed. We believe that it will be enlightening to system designers for further optimizations of the flash I/O performance and endurance.

1) Process variation: PV is the phenomenon that pages in different memory blocks may have largely different worst-case RBER under the same P/E cycling, equivalent to largely different P/E cycling endurance within different memory blocks when given the same ECC. Previous work [5] showed that the RBER of flash blocks tends to follow a log Gaussian-like distribution. PV results in blocks with stronger and weaker cells, which accumulate bit errors more slowly or faster, respectively. In this work, PV is measured by periodically reading blocks and finding out how many bits have to be corrected by ECC.

Recently, techniques exploiting PV have mostly been focused on wear-leveling for its ability to make full use of the strong blocks within SSDs to maximize lifetime. For example, Pan et al. [5] extended flash memory lifetime by using RBER statistics as the measurement of memory block wear-out pace for the wear-leveling algorithm. Woo et al. [6] introduced a new measure that predicts the remaining lifetime of a flash block more accurately than the erase count based on the findings that all the flash blocks could survive much longer than the guaranteed numbers and the number of P/E cycles vary significantly among blocks. To the authors' knowledge, the PV-aware data allocation method presented by Shi et al. [7] is the only one which considers both PV and the tradeoff between RBER and write speed, using coarser ΔV_p for strong pages which do not accumulate errors as fast and allocating blocks in a way that hotter data are matched with faster blocks. In this paper, our hotness-aware write scheduling algorithm also takes advantage of strong blocks based on the PV-aware data allocation.

2) Retention age variation: Data retention age is the length of time since a flash cell was programmed. Data retention error, which is caused by the charges leaking from floating gates as time goes by, is one of the dominant errors. Therefore, the retention age variation would result in different RBERs, which could be exploited in the same way as PV-aware techniques do.

The impact of data retention skew on storage system performance has been thoroughly analyzed. Some works focus on minimizing refresh cost. For example, Luo et al. [21] introduced a write-hotness aware retention management policy called WARM for NAND flash memory, which allows flash controller to relax the flash retention time for write-hot data without the need for refresh, by exploiting the high write frequency of this data. Most recently Di et al. [24] proposed a refresh minimization method by writing the data of long retention time requirement into high endurance blocks. Another set of approaches adapt cell programming/read-out parameters for improved performance. For example, Cai et al. [4] presented a retention optimized reading (ROR) method that periodically learns a tight upper bound and applies the optimal read reference voltage for each flash memory block online. Shi et al. [22] proposed a retention trimming approach for wearing reduction by decreasing programming voltages when the estimated retention time is lower. Liu et al. [20] achieved write response time speedup based on the estimated retention time, by adapting both the programming step size ΔV_p and ECC strength. These studies demonstrate that retention age variation in workloads is evident and useful. In this paper, our retention-aware read scheduling algorithm takes advantage of data with low retention age based on the retention-aware ECC adaptation.

When using PV-based fast write and retention age-based fast read, the requests are accelerated in varing degrees, which inevitably lead to the significant read and write speed variation. Fortunately, I/O scheduler is a good candidate for taking advantage of speed variations to improve read/write performance. While most flash-based I/O schedulers focused on how to reduce the access conflict and improve chip utilization by exploiting the internal parallelism of SSDs [8, 15, 18, 19], we focus on the reduction of access conflict latency when conflicts are unavoidable anymore, by taking advantage of speed variations.

III. Retention-aware and Hotness-aware I/O Scheduling

In this section, a retention-aware and hotness-aware I/O scheduling algorithm (RHIO) is proposed. The design principles of RHIO are based on the observation we draw from our dataset and analyses: If a tradeoff-aware technique improves I/O performance based on the variation characteristic of an attribute, the detection of the attribute can be implemented in I/O scheduling and thus the tradeoff induced speed variation can be exploited for maximal benefit by giving scheduling priority to fast writes and fast reads.

The basic idea of RHIO is to separate write requests into different queues depending on their hotness, while read requests are separated based on the retention ages of accessed data. For hot writes, their data are allocated to strong blocks using fast write, given scheduling priority. For reads accessing data with low retention ages, fast read is performed and scheduled preferentially to minimize the access conflict latency of I/O requests.

A. Hotness-aware Write Scheduling

Generally, the access latency of write requests is composed of the access conflict latency, the data transfer latency and the program latency. Compared with read requests, write requests are more time-consuming, leading to serious access conflict latency. The hotness-aware write scheduling scheme aims to decrease access conflict latency by exploiting PV-induced RBER variation, the tradeoff induced write speed variation and the access characteristic induced hotness variation. The advantages of separating data with different hotness have been exploited in several researches [7, 10, 11, 12].

In the proposed hotness-aware write scheduling scheme, we put hot data in strong blocks using fast write, and non-hot data into normal blocks with normal writes. Hot data on strong blocks is invalidated quickly, maximizing the benefit from strong blocks which have low RBER growth rates per P/E cycle. Furthermore, hot write requests are issued preferentially to reduce the conflict latency of next few requests in the queue. Note that the use of different hot/cold classification schemes is orthogonal to the design of our scheme. In this work, we identify hot/cold data according to the size of its file system I/O request, which has been shown to be simple and effective in several previous studies [7, 25]. It has been suggested that the larger the request sizes, the colder the data. The hotness groups are adopted as the main data structure to schedule the requests with the same hotness into same groups.

The proposed method can guarantee that the high-hotness write requests are not blocked by the low-hotness write requests. However, it may cause the low-hotness requests

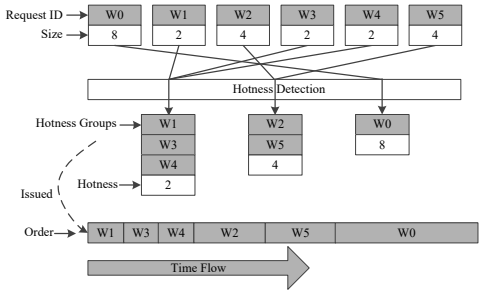

Fig. 2. An example of hotness-aware write scheduling.

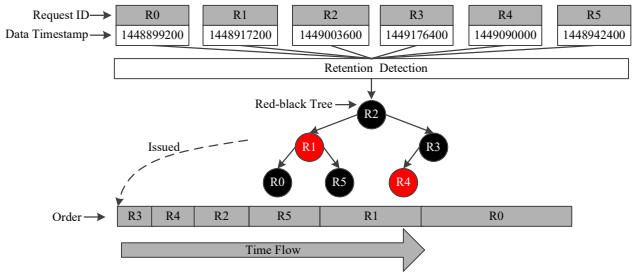

Fig. 3. An example of retention-aware read scheduling.

waiting in the queue for a long time without service. In order to avoid potential starvation of cold requests, each incoming request is assigned a deadline time that defines the latest timepoint before which the request should be issued. In a serial ATA (SATA) interface, the deadline information is configured by placing '01b' into the Priority (PRIO) field of NCQ commands (READ FPDMA QUEUED and WRITE FPDMA QUEUED) to mark them as isochronous, and then filling the corresponding deadline values in the Isochronous Command Completion (ICC) field [26]. Note that ICC Bit 7 is cleared to zero so that the time interval is fine-grained which is 10 msec. The First In First Out (FIFO) queue which links requests together in their arriving time order is adopted by RHIO and periodically checked. When the head request of the FIFO queue reaches the preset deadline restriction, the request cannot be blocked anymore and will be immediately processed to ensure that the hotness priority strategy does not cause the potential starvation.

Figure 2 shows an example of hotness-aware write scheduling. As shown in Figure 2, six write requests accessing the same chip are added to the write request queue in the FIFO order, from left to right, where Size(W0) = 8, Size(W1, W3, W4) = 2, Size(W2, W5) = 4. Based on the hotness detection, three hotness groups are created, where HG1 = {W1, W3, W4}, HG2 = {W2, W5}, HG3 = {W0}. Finally, the hotness-aware write scheduling scheme issues the I/O requests in hotness groups, where all the hotness groups are processed in the hotness order. By denoting $PL(size)$ as the latency of writing one page for requests which have *size* pages, *N* as the number of requests enrolled in conflict and O_i as the issued order of request *i*, the average request response time can be defined as:

$$avg\_write = \left(\sum_{i=0}^{N-1} [PL(size_i) * size_i * (N - O_i)] \right) /N \quad (1)$$

In this example, we assume that $PL(2)$ = 200μs, $PL(4)$ = 220μs, and $PL(8)$ = 240μs, the average request response time of the proposed algorithm is 2053.3μs, while that of the normal algorithm is 3320.0μs. Therefore, the write speed variation is fully exploited in the hotness-aware write scheduling scheme by preferentially scheduling the high hotness requests which access strong blocks.

B. Retention-aware Read Scheduling

Similar to write requests, the access latency of read requests is composed of the access conflict latency, the data transfer latency and the sensing latency. Since overall system performance tracks storages average read response time, performance of read requests is critical. Therefore, read and write separation is commonly used to prioritize the scheduling of read requests, which guarantees that the read requests are not blocked by the time-consuming write requests. In this work, the retention-aware read scheduling scheme aims to decrease read conflict latency by exploiting the tradeoff induced read speed variation and the access characteristic induced data retention variation.

In the retention-aware read scheduling scheme, the read requests are sorted according to the retention age of the data to be read. For reads accessing data with low retention ages, fast read is performed and scheduled preferentially to minimize the access conflict latency of I/O requests. The identification of retention age can be achieved by extending each mapping entry in the FTL with a timestamp field and recording the timestamp when data is programmed. Different from the size-based hotness detection in write scheduling, the hotness detection in read scheduling is based on the retention age, which is an actual time instead of prediction time. Note that since PV has been exploited when writing, we only consider the RBER accumulation as a function of the data retention age, without taking the strength of blocks into consideration.

The scheme is implemented in the host interface logic (HIL), where the knowledge of both SSD specific characteristics and the data programming timestamp recorded in the flash translation layer (FTL) contributes to better device-specific scheduling decisions for I/O requests. Note that the FIFO queue proposed in hotness-aware write scheduling scheme also plays a role in preventing the occurrence of read starvation.

Figure 3 shows an example of retention-aware read scheduling. As shown in Figure 3, six read requests accessing the same chip are added to the read request queue in the FIFO order, from left to right. Based on the retention detection, the red-black tree is created. Since the timestamp of data accessed by R3 is the largest, which means the data is newest, R3 is the first request to be issued.

With the combination of hotness-aware write scheduling and retention-aware read scheduling, the implementation of RHIO

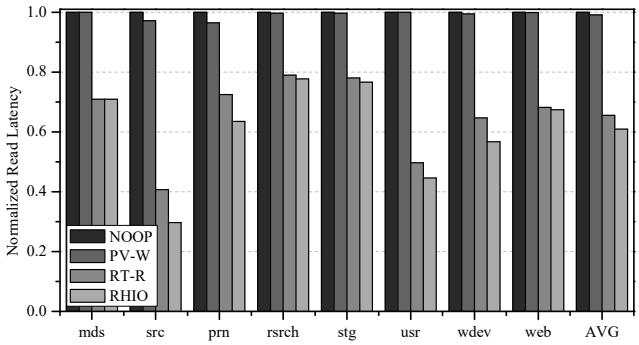

Fig. 4. Normalized average latency for read requests.

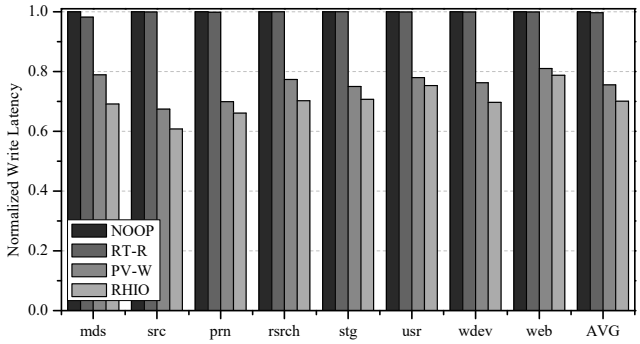

Fig. 5. Normalized average latency for write requests.

needs to maintain a hotness group list and a red-black tree in the I/O queue. Since the queue length of I/O scheduler is limited, the storage cost is negligible. The major computation overhead of the proposed method includes the time overhead incurred by finding the proper hotness group for write requests and inserting read requests into the red-black tree, where the complexity is proportional to the logarithm of the queue length. In addition, the state-of-the-art I/O scheduling algorithms, which exploit the internal parallelism of SSDs, are somewhat orthogonal to our work.

IV. Experiment and Analysis

A. Experimental setup

In this paper, we use an event-driven simulator to further demonstrate the effectiveness of the proposed RHIO. We simulate a 128GB SSD with 8 channels, each of which is connected to 8 flash memory chips. The RBER growth rate s that follows a Bounded Gaussian distribution is used to simulate the process variation of flash memory, where the mean μ and the standard deviation σ are set as 3.7×10^{-4} and 9×10^{-5} respectively [5]. We use 600 μs as the 2bit/cell NAND flash memory program latency when ΔV_p is 0.3, 90 μs as memory sensing latency and 80 μs as data transfer latency when using LDPC with seven reference voltages [8]. All these settings are consistent with previous works.

For validation, we implement RHIO as well as baseline NOOP scheduling, PV-W and RT-R. PV-W implements PV-aware write performance improvement without conflict-aware reordering, while RT-R implements retention-aware read performance improvement without reordering I/O requests sequence. We evaluate our design using real world workloads from the MSR Cambridge traces [14], which are widely used in previous works to study SSD performance [13, 16, 17].

B. Experimental Results

In this section, the experiment results are presented and analyzed. Read and write latency are commonly used to evaluate scheduling performance. Figure 4 and Figure 5 show the normalized average latency for read and write requests, respectively. Compared with the traditional NOOP, RHIO achieves significant read and write performance improvement. For read requests, benefit from both retention induced read speed variation and conflict-aware reordering, RHIO reduces read latency by 39.11% on average. For write requests, RHIO achieves 29.92% write latency reduction by giving scheduling priority to hot write requests and allocating their data to strong blocks with fast write. In addition, RHIO outperforms RT-R and PV-W with read latency reduction by 7.04% and 38.56% on average, respectively. This is because the retention-aware read scheduling scheme reduces the conflict latency by preferentially issuing fast read requests. Similar to read requests, RHIO achieves 29.71% and 7.12% write latency reduction compared to RT-R and PV-W respectively, by exploiting fast write requests for conflict reduction.

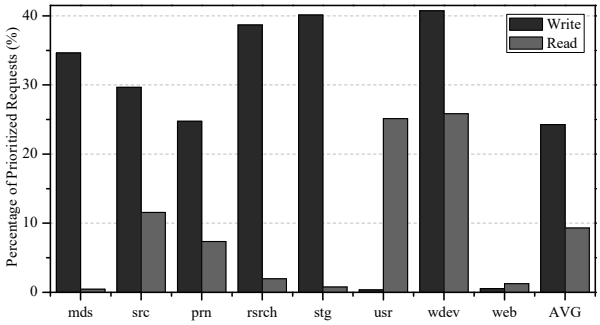

Fig. 6. Percentage of prioritized read and write requests in RHIO.

However, the read and write performance improvements for different traces are very different from each other. For example, compared to PV-W, the greatest write latency reduction observed in *mds* is 12.42%, while the smallest reduction observed in *web* is only 2.82%. Compared to RT-R, the greatest read latency reduction observed in *src* is 27.13%, while the smallest reduction observed in *mds* is only 0.1%. In order to understand the reason for different performance improvement among traces, the percentages of prioritized write requests and read requests collected with RHIO scheduler are presented in Figure 6. By comparing the results with the write performance improvements shown in Figure 5, it can be observed that the write latency reduction compared to PV-W is larger when the prioritized write ratio is higher. For example, the average write latency for most traces whose prioritized write ratios are more than 25% is significantly reduced, while that for *usr* and *web* which only have 0.34% and 0.56% prioritized writes in RHIO,

compared to PV-W, is slightly reduced by 3.34% and 2.82%, respectively. This is because more intensive I/Os in some traces would induce more conflicts and longer conflict latency, which have been reduced in our proposed RHIO by scheduling more write requests that have fast speeds in priority. The observation is also supported by the results of read. The percentages of prioritized read requests for most applications are within 5%, while for *src*, *prn*, *usr* and *wdev*, there are more than 5% of read requests prioritized, which induces improved read performance compared to RT-R. Overall, these results clearly demonstrate the effectiveness of RHIO in reducing the read and write latency, especially for read and write intensive applications.

V. Conclusion

In this paper, we have proposed a retention-aware and hotness-aware I/O scheduling algorithm (RHIO) for NAND flash-based SSDs. Different from previous works, the latency variation among blocks is exploited to guide both read and write latency minimization. The key insight behind the design of RHIO is that a hotness-aware write scheduling scheme can reduce write conflict latency by giving scheduling priority to hot write requests and allocating their data to strong blocks with fast write, and a retention-aware read scheduling scheme can reduce read conflict latency by preferentially scheduling read requests which access data with low retention ages using fast read. Extensive experimental results and detailed comparisons show that the proposed technique achieves significant performance improvement by 39.11% and 29.92% for read and write requests, on average.

Acknowledgment

The authors would like to thank our shepherd Peter Desnoyers and the anonymous reviewers for their valuable feedback. This work was supported in part by the National Natural Science Foundation of China under grant NO.91330117, National High-tech R&D Program of China (863 Program) under Grant No.2014AA01A302, the Shenzhen Scientific Plan under Grant No.JCYJ20130401095947230 and No.JSGG20140519141854753.

References

[1] C. Min, K. Kim, H.J. Cho, S.-W. Lee, and Y.I. Eom, "SFS: random write considered harmful in solid state drives," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2012, Feb., pp. 12-28.

[2] K.C. Ho, P.C. Fang, H.P. Li, C.Y. Wang, and H.C. Chang, "A 45nm 6b/cell charge-trapping flash memory using LDPC-based ECC and drift-immune soft-sensing engine," in *Proceedings of International Solid-State Circuits Conference (ISSCC)*, 2013, Feb., pp. 222-223.

[3] S. Zuloaga, R. Liu, P.Y. Chen, and S. Yu, "Scaling 2-layer RRAM cross-point array towards 10 nm node: A device-circuit co-design," in *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, 2015, May, pp. 193-196.

[4] Y. Cai, Y. Luo, E.F. Haratsch, K. Mai, O. Mutlu, "Data retention in MLC NAND flash memory: Characterization, optimization, and recovery," in *Proceedings of International Symposium on High Performance Computer Architecture (HPCA)*, 2015, February, pp. 551-563.

[5] Y.Y. Pan, G.Q. Dong, and T. Zhang, "Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 7, pp. 1350-1354, 2013.

[6] Y.J. Woo, and J.S. Kim, "Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs," in *Proceedings of International Conference on Embedded Software (EMSOFT)*, 2013, Sep., pp. 6-16.

[7] L. Shi, Y.J. Di, M.Y. Zhao, C.J. Xue, K.J. Wu, and H.-M. Sha, "Exploiting Process Variation for Write Performance Improvement on NAND Flash Memory Storage Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 99, no. 7, pp. 1-4, 2015.

[8] Q. Li, L. Shi, C.M. Gao, K.J. Wu and et al, "Maximizing IO performance via conflict reduction for flash memory storage systems," in *Proceedings of Proceedings of the 2015 Design, Automation Test in Europe Conference Exhibition(DATE)*, 2015, March, pp. 904-907.

[9] K. Zhao, W.Z. Zhao, H.B. Sun, T. Zhang, X.D. Zhang, and N.N. Zheng, "LDPC-in-SSD: making advanced error correction codes work effectively in solid state drives," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2013, pp. 243-256.

[10] Y. Gala, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving SSD erase costs using WOM codes," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2015, pp. 257-271.

[11] J. Xavier, N. David, and P. Ienne, "Wear unleveling: improving NAND flash lifetime by balancing page endurance," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2014, pp. 47-59.

[12] O. Saher, and Y. Cassuto, "NAND flash architectures reducing write amplification through multi-write codes," In *Proceedings of Symposium on Mass Storage Systems and Technologies (MSST)*, 2014, June, pp. 1-10.

[13] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S.Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," In *Proceedings of Proceedings of the international conference on Supercomputing*, 2011, May, pp. 96-107.

[14] W. Schrder-Preikschat, J. Wilkes, R. Isaacs, D. Narayanan, and et al, "Migrating server storage to SSDs: analysis of tradeoffs," In *Proceedings of the 4th ACM European conference on Computer systems(EuroSys)*, 2009, p. 145.

[15] C. Gao, L. Shi, M. Zhao, C. Xue, K. Wu and E. Sha, "Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives," In *Proceedings of Symposium on Mass Storage Systems and Technologies (MSST)*, 2014, June, pp. 1-11.

[16] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo and C. Ren, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," *IEEE Transactions on Computers*, 62(6): 1141-1155, 2013.

[17] M. Jung and M. Kandemir, "An evaluation of different page allocation strategies on high-speed ssds," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2012, pp. 9.

[18] M. Bo and S. Wu, "Exploiting request characteristics and internal parallelism to improve ssd performance," In *International Conference on Computer Design (ICCD)*, 2015, October, pp. 447-450.

[19] M. Jung, W. Choi, S. Srikantaiah, J. Yoo and M. Kandemir, "HIOS: a host interface I/O scheduler for solid state disks," *ACM SIGARCH Computer Architecture News*, 42(3): 289-300, 2014.

[20] R. Liu, C. Yang and W. Wu, "Optimizing NAND Flash-Based SSDs via Retention Relaxation," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2012.

[21] Y. Luo, Y. Cai, S. Ghose, J. Choi and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management," In *Proceedings of Symposium on Mass Storage Systems and Technologies (MSST)*, 2015, May, pp. 1-14.

[22] L. Shi, K. Wu, M. Zhao, C. Xue, D. Liu and E. Sha, "Retention Trimming for Lifetime Improvement of Flash Memory Storage Systems ," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1): 58-71, 2016.

[23] Q. Li, L. Shi, C. Xue, K. Wu, C. Ji, Q. Zhuge and E. Sha, "Access Characteristic Guided Read and Write Cost Regulation for Performance Improvement on Flash Memory," In *Proceedings of Conference on File and Storage Technologies(FAST)*, 2016, February, pp. 125.

[24] Y. Di, L. Shi, K. Wu and C. Xue, "Exploiting Process Variation for Retention Induced Refresh Minimization on Flash Memory," *the 19th Design, Automation Test in Europe (DATE)*, Dresden, Germany, March, 2016.

[25] S. Im and D. Shin, "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," *Journal of Systems Architecture*, 56(12): 641-653, 2010.

[26] D.J. Molaro, F.R.F. Chu, J.C. De Souza, A. Kanamaru, T. Kawa, and D.C. Le Moal, "Data storage devices accepting queued commands having deadlines," *U.S. Patent 8,539,176*, issued September 17, 2013.