IEICE Electronics Express, Vol.17, No.8, 1–6

LETTER

Read disturb-aware write scheduling and data reallocation in SSDs

Bowen Huang[1], Jianwei Liao[1,2a)], Jun Li[1], Yang Chen[1], Zhigang Cai[1], and Yuanquan Shi[2b)]

Abstract Read disturb is a circuit-level noise in SSDs, which may corrupt existing data in SSD blocks, and then results in high read error rate and longer read latency. This paper proposes schemes of write scheduling and data reallocation, by taking account of read disturb. We first construct a model to estimate the block read error rate caused by read disturb, by referring the factors of block's P/E cycle and the accumulated read count to the block. Then, the data being intensively read are flushed to the block having a small read error rate. Moreover, we introduce a data reallocation mechanism, which is completed by read reclaim, for balancing read accesses in all blocks. Thus, the total read errors introduced by read disturb can be cut down. Through a series of emulation tests based on several realistic disk traces, we demonstrate that the proposed mechanism can yield attractive performance improvements on the metrics of read latency and read error rate.
Keywords: NAND flash memory, read disturb, write scheduling, data reallocation, read errors
Classification: Circuits and modules for storage

1. Introduction

As the feature size of NAND flash memory [1] cells reaches the limit of the 10 nm level, further flash density increases are then driven by TLC (3 bits/cell) combined with vertical stacking of NAND memory planes. However, the decrease in endurance and the increase in bit error rates accompanying with the feature size shrinking are now becoming the issues to be reckoned with [2, 3, 4].

Among the many noises that affect the endurance of flash memory, retention noise [5, 6, 7, 8], program interference noise [9, 10, 11], program/erase cycling noise [12, 13], and read disturb [14] are the most important types.

Specially, read disturb is an unexpected phenomenon in NAND flash memory, where reading data from a flash page can impact the threshold voltages of other (unread) pages in the same block. It has become a growing source of flash errors, and the situation gets worse in a compact NAND memory, such as TLC [2, 15]. This is because the geometry of a cell shrinks, the cross-coupling voltage noise gets even more intense that may consequently trigger more read disturb errors.

Solid-State Drives (SSDs) are typical products of NAND flash memory that develop into the dominant secondary storage in the coming years [16]. Then, this paper discusses the driven factors of read disturb in SSDs, and then builds a mathematical model for objectively assessing the read error rate caused by read disturb, with respect to a specific SSD block. As a result, it proposes a write scheduling approach and a dynamic data reallocation mechanism for SSDs, by taking account of the issue of read disturb. In summary, it makes the following three contributions:

- We systematically analyze impact factors of read disturb, including the number of block P/E cycles and accumulated block reads. Then, we build a comprehensive model to estimate the read error rate of given block by referring the aforementioned two SSD nature factors.
- We propose a write scheduling approach to map future hot read data to the block having a small read error rate. To better cut down negative effects of read disturb caused by frequently reading data on hot blocks, we further present a data reallocation approach to balance read accesses to involved blocks, through carrying out read reclaim tasks.
- We offer preliminary evaluation on several disk traces of real-world applications. As measurements indicate, our proposal can afford a better performance improvement on the metrics of average read latency, read error rate, and the overall I/O time.

Section 2 introduces the background knowledge and related work on read disturb. Section 3 describes the comprehensive model, the proposed write scheduling approach, and read balance-based data reallocation. Section 4 shows the evaluation methodology and reports the experimental results. The paper is concluded in Section 5.

2. Background and related work

Read disturb is a circuit-level noise in NAND-based memory, which is induced by read operations [17, 18, 19]. Fig. 1(a) shows the voltage settings in the case of dealing with a read operation. As seen, a reference voltage of V_r is applied to the target word line of WL_1, meanwhile a large read pass voltage of V_{pass} is imposed to other word lines.

Nevertheless, applying a high read pass voltage to victim word lines shifts cell threshold voltages through electron injection to floating gates of cells [17]. Though a single read operation does not modify the neighboring cell data immediately, the relevant cell data will be eventually altered when the side-effect is accumulated by repetitive read operations [18, 20]. Fig. 1(b) illustrates an example of cell data change resulted by read disturb. In the case, the

[1]College of Computer and Information Science, Southwest University, Chongqing 400715, China
[2]School of Computer Science and Engineering, Huaihua University, Hunan 400800, China
a) liaotoad@gmail.com
b) syuanquan@163.com

DOI: 10.1587/elex.17.20200015
Received January 11, 2020
Accepted February 26, 2020
Publicized March 24, 2020
Copyedited April 25, 2020

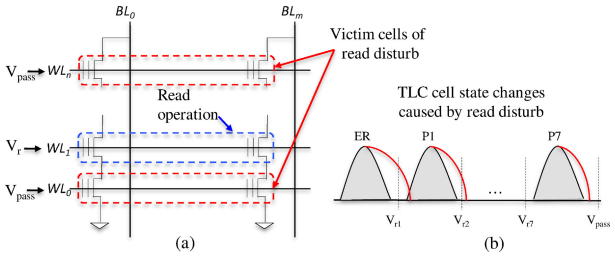

Fig. 1. (a) Voltage settings during a read operation and read disturb. (b) Reference voltage (V_{ri}) and read pass voltage (V_{pass}) of an TLC cell [18].

reference voltage of V_{r1} fails to clearly identify the states of (disturbed) *ER* and *P1*.

There have been a number of work on read disturb, which can be generally classified as hardware-based and software based mechanisms. Cai *et al.* [17] proposed learning the minimum pass-through voltage for each SSD block, to dynamically tune the pass-through voltage on a per-block basis for minimizing read disturb errors. Ha *et al.* [18] disclosed that read disturbance is positively correlated with V_{pass} and the duration of imposing V_{pass}. Then, they proposed to write read-hot data using narrow threshold voltage levels, so that such data can be read by leveraging a low V_{pass} within a short interval. However, this approach confines write performance. Wu *et al.* [19] have further proposed to reserve selected threshold voltage levels as guard levels for enlarging the tolerance margin between different states, at a cost of reduction of storage density.

Software-based techniques have also been designed to mitigate the negative effects of read disturb. Liu *et al.* [21] introduced *Read Leveling*, which takes advantage of shadow blocks. Specifically, besides purposely keeping the hot read data, the shadow blocks should contain a number of invalid data pages or free pages, as both kind of pages are immune to read disturb. However, *Read Leveling* does not perform well for utilizing the storage space in shadow blocks, which limits the improvement on read latency and read refreshing cycles. In order to ward off corrupting existing data resulted by read disturb, Werner *et al.* [22] proposed to relocate the data to other blocks if the original host blocks have reached a read disturb limit. That is to say, for avoiding data corruption by read disturbs, a process of read reclaim (RR) is expected in partially read-disturbed blocks [18]. Specifically, a process of read reclaim migrates all valid data in the disturbed block to a free block, and then reclaims the disturbed one [23].

3. Read disturb-aware scheduling and data reallocation

3.1 System overview

We first identify hot read data in the current time window, and then map their corresponding write requests (in the next window) to the block that are not susceptible to read disturb. After that, we re-allocate the heavily read data accompanying with the less accessed data onto randomly selected free blocks, when conducting read reclaim[1]. Thus, the adjusted data blocks may have a (near) unified access count and the effect of read disturb can be significantly limited.

[1]It completes hot read data re-distribution in the process of read reclaim.

3.2 Modeling read error rate

The block P/E cycle does directly impact the read error rate. On the other side, the factor of block read count affects the read error rate, but it also depends on the block P/E cycle. In other words, the read error rate resulted by read operations becomes bigger, in the case of the block P/E cycle is relative large. We thus build a non-linear regression model to profile the impacts on the rate of read errors regarding a given block.

$$P_i = \phi_0(PE_i) + \phi_1(PE_i) \cdot R_i + \epsilon_i \quad (1)$$

where P_i, PE_i, and R_i respectively denote the read error rate, the P/E cycle and the read count regarding the *ith* block.

Moreover, ϕ_0 and ϕ_1 are two real-valued functions with argument PE, for weighting nonlinear effects of the P/E cycles and block read counts to the read error rate. Then, we use higher-order polynomials to estimate their values:

$$\phi_0(PE_i) = a_0 + a_1 \cdot PE_i + a_2 \cdot PE_i^2 + \cdots + a_p \cdot PE_i^p \quad (2)$$

$$\phi_1(PE_i) = b_0 + b_1 \cdot PE_i + b_2 \cdot PE_i^2 + \cdots + b_q \cdot PE_i^q \quad (3)$$

By taking advantage of the experimental data presented in [17], we employ the *Akaike* information criterion [24] to determine the orders of ϕ_0 and ϕ_1, and the outcomes are $p = 2$ and $q = 5$. Consequently, we obtain the functions values of ϕ_1 and ϕ_2, when the erase number scales from 4K to 20K, as reported in Table I. That is to say, we can figure out read error rates of given blocks, for classifying them into two categories, i.e. susceptible blocks and insusceptible blocks to read disturb.

3.3 Write scheduling

Fig. 2 illustrates the specifications on the proposed read disturb-aware write scheduling. As seen, the proposed scheme dispatches the received write requests, by referring the pre-identified hot data set. We regard the data on the addresses that will be requested multiple time (e.g. >2 in our tests) in the time window, as the hot data set.

Table I. Values of ϕ_0 and ϕ_1 with varied P/E cycles (unit: 10^{-3})

	4K	8K	12K	16K	20K
ϕ_0	0.251	0.590	1.070	1.693	2.457
ϕ_1	0.003	0.017	0.085	0.320	0.915

Note: The input data of our model are reported in [17]. The largest block P/E cycle in the available dataset is 16K, our model predicts the values of ϕ_0 and ϕ_1 in the case of the block P/E cycle is 20K.

Then, the write requests whose logic sector numbers are in the hot read set, will be mapped to an active block having a small read error rate. The purpose is trying to offset the side-effects of read disturb on the target block, though such data will be frequently read. Otherwise, the write data will be flushed to a block having a relatively large read error rate.

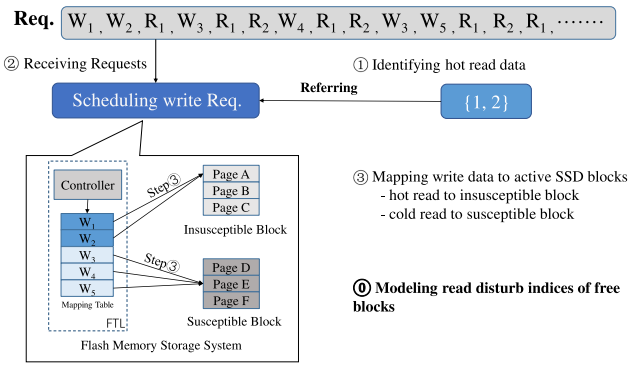

Fig. 2. High level overview of read disturb-aware write scheduling. *W*, *R* denote write and read requests separately, and their logical sector numbers are represented as *1*, *2*…, and *5*.

3.4 Data reallocation via read reclaim

The basic idea of data reallocation is to adaptively group hot read data with cold read data. The motivation is not only to unify erasure distribution of SSD blocks, but also to balance read accesses to the blocks for optimally limiting negative effects of read disturb (e.g. read error rate). Furthermore, in order to decrease the overhead of data reallocation, we carry out data migration in the process of read reclaim [23, 25].

Fig. 3 illustrates the workflow of the proposed read reclaim approach, for the purpose of unifying read counts among data blocks. As seen, it first identifies the target blocks, which contain hot read data needing to be migrated. After that, it splits the hot read data into two parts, and moves them onto two susceptible blocks that may have certain cold read data. In fact, the basic combining rule of read data is to ensure the total read count of all regrouped blocks is (roughly) the same.

We have verified that our reallocation scheme can yield a minimum read error count in total by evening read accesses across valid blocks. The mathematical proof can be found in ***Appendix A***.

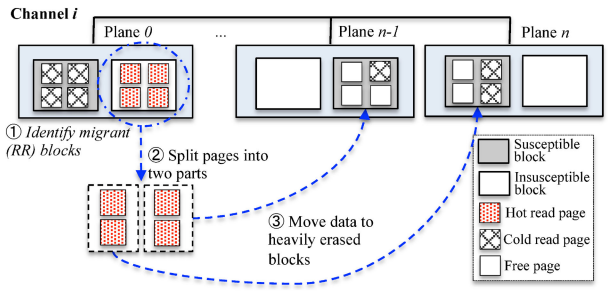

Fig. 3. Hot read data reallocation via read reclaim. Note that reading to hot data brings about read disturb to all other pages in the same block.

4. Experiments and evaluation

4.1 Experiment setup

The experimental platform was constructed by making use of a widely used SSD simulator of *SSDsim* (`ver 2.1`), to conduct trace-driven tests [26]. Note that *SSDsim* can only model the performance of SSDs, it cannot store and restore real data for each request. Table II presents our settings of *SSDsim* in experiments. Since the read time depends on not only the type of pages in the block, but also the block feature of read disturb, we separately set the time onto susceptible blocks and unsusceptible blocks by referring [27, 28]. For example, the time needed for reading a LSB page, which is located in a unsusceptible block to read disturb is 45 µs; otherwise the read time should be 75 µs. In addition, the threshold of read reclaim is set as 38000 [18].

Table II. Experimental settings of *SSDsim* (TLC cell)

Parameters	Values	Parameters	Values (µs)
Capacity	64K blocks	*LSB read*	(75, 45)
Page per block	384	*CSB read*	(110, 80)
Page size	8K	*MSB read*	(165, 135)
RR threshold	38000	*LSB write*	500
Overprovide	0.25	*CSB write*	2000
FTL scheme	Page mapping	*MSB write*	5500

We employed 5 commonly used disk traces, two from the block I/O trace collection of Microsoft Research Cambridge [29] and three from Umass Trace Repository [30]. The detailed specifications on the traces are shown in Table III. Specially, the metric of *Hot r/r* indicates the ratio of the frequently requested addresses (i.e. the accessed time is not less than 4) to all read address space.

Apart from the proposed scheme (labeled as *Relocation*), we have selected the default dynamic mapping scheme in *SSDsim* as a comparison counterpart (labeled as *Baseline*). Furthermore, the software-based mechanism of *Read leveling* [21] has been also employed as another counterpart in experiments. We argue that *Read leveling* is the most related work to ours. Similarly, it works at Flash Translation Layer of SSDs to distribute hot read data, and makes use of read reclaim to relieve the negative effects of read disturb.

Table III. Specifications on selected disk traces

Trace	# of Req.	Read ratio	Hot r/r	Read size
wdev0	1143261	20.1%	28.1%	12.6 KB
hm0	3993316	35.5%	15.4%	7.4 KB
websearch1	1055448	99.9%	87.8%	15.2 KB
websearch2	4579809	99.9%	96.8%	15.0 KB
websearch3	4261709	99.9%	96.0%	15.1 KB

In addition, the maximum number of I/O requests processed in each time window is configured as 8192 in the evaluation. The logical sector numbers of in-queue requests are leveraged for mining the hot read set. Then, the hot read set is used to mapping the write requests in the next time window.

4.2 Tests and benefit illustration

To measure validity of the proposed mechanism that aims to mitigate the negative effects resulted by read disturb in SSDs, we use the following two metrics in our tests: (a) *average read latency* and (b) *read error rate*.

4.2.1 Average read latency

As seen in Fig. 4, the proposed approach outperforms other two comparison counterparts in all cases. Regarding the

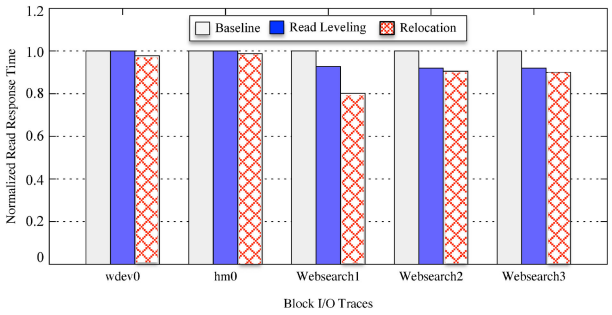

Fig. 4. Average read latency of selected block traces.

traces of *wdev0* and *hm0*, *Read Leveling* does not perform well, and the proposed scheme reduces the read time by less than 1.6%, compared with *Baseline*. In fact, while dealing with *wdev0* and *hm0*, which are two write intensive workloads, the process of read reclaim is not triggered. Thus, we cannot reallocate hot read data since they have been initially flushed.

But, in the case of processing other three read intensive workloads (i.e. *websearch* traces), a number of read reclaim processes occurred. So that the proposed *Relocation* scheme can cut down the read latency by 13.2% and 10.5% on average, in contrast to *Baseline* and *Read Leveling*. Then, we argue that data reallocation during read reclaim can effectively balance read counts among blocks, and then boost the read performance by limiting the impact of read disturb.

4.2.2 Read error rate

Raw bit error rate (*RBER*) is a measure of the number of bit errors that occur in a given number of bit transmissions without ECC. We specifically record the bit error rate on read operations (which is defined it as read error rate in this paper), when processing the read requests of selected traces.

As shown in Figure 5, the proposed *Relocation* scheme can reduce the read error rate by up to 58.4%, compared with *Baseline*. Besides, we see our proposal

Table IV. Page move in read reclaim on selected disk traces

Trace	**Baseline**	**Read Leveling**	**Relocation**
wdev0	0	0	0
hm0	0	0	0
websearch1	3032	3035	2922
websearch2	16366	34748	16000
websearch3	19068	39855	17053

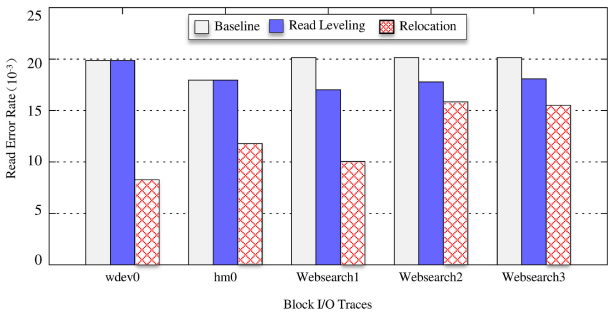

Fig. 5. Read error rate of selected block traces.

achieves reduction on read errors by 35.4% on average, in contrast the related work of *Read Leveling*. This is because the frequently requested data are kept by insusceptible blocks, which can confine the read errors caused by intensive read operations on the blocks at an early stage. Once the read count to the block reaches the threshold of read reclaim, data reallocation will balance read counts among involved blocks, which can contribute to minimizing read errors caused by read disturb.

4.3 Overhead analysis and overall processing time

4.3.1 Read reclaim overhead

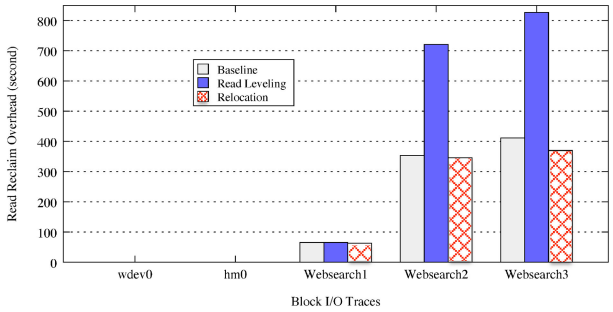

Fig. 6. Read reclaim time of selected block traces.

The read reclaim overhead consists the time required for erasing original block and the time needed for moving valid pages between original block and the target block. Fig. 6 presents the time required for completing read reclaim when running the selected traces.

Note that only *websearch* traces can triggered read reclaim. As seen, the proposed *Relocation* achieves the least read reclaim overhead, as all three schemes bring about almost the same number of read reclaim, but our proposal introduces the least number of page move in read reclaim, as reported in Table IV.

Another interesting clue shown in Fig. 6 is about the *Read Leveling* scheme causes the largest read reclaim overhead. In fact, the motivation of *Read Leveling* is to minimize the number of read reclaim, but it may lead to a large number of page move. In our tests, we set the threshold of read reclaim as 38000 by referring [18], instead of 1000 adopted by *Read Leveling* [21]. We argue that 1000 is too small to be the threshold of read reclaim, since the maximum read count of a block can reach 40000 in a TLC SSD [18].

As a consequence, all three schemes yield almost same number of read reclaim, which is small value, and less than 201. On the other side, *Read Leveling* results in the largest number of page move in read reclaim. Consequently, it brings about more read reclaim overhead, compared with *Baseline* and *Relocation*.

4.3.2 Mining and mapping overhead

Fig. 7 shows the time required by mining hot read set and mapping write requests, when using *Relocation*. As shown in the figure, the mining and mapping overhead is less than 6.9 seconds for all traces. Generally, the mining and mapping overhead is related to the number of total requests and the number of frequently read addresses in the trace.

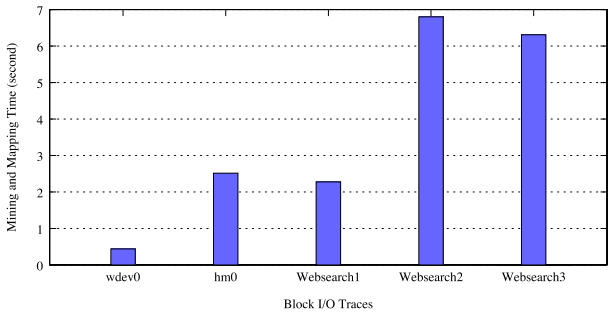

Fig. 7. Hot read set mining and write mapping overhead in the proposed *Relocation* Approach.

4.3.3 Overall processing time

Fig. 8 reports overall processing time, which contains the total I/O time (the read time and the write time) and the read reclaim time. In the *Relocation* approach, the overall processing time additionally includes the hot read set mining time and the time for mapping write requests.

As shown in the figure, *Relocation* yields the least overall processing time, even though only it requires extra time for mining hot read set and mapping relevant write requests. This is due to our proposal can guarantee the least read latency, and the least read reclaim overhead, which have been respectively demonstrated in Sections 4.2.1 and 4.3.1.

4.4 Summary

With respect to comparing existing schemes and the newly proposed mechanism, we emphasize the following two key observations. ***First***, the read disturb level-based data mapping scheme works at the early stage of application. ***Second***, the data reallocation scheme can balance read counts among SSD blocks, to minimize the negative effects of read disturb at the late stage of application. In brief, we conclude that the proposed mechanism is able to significantly reduce the negative effects introduced by read disturb.

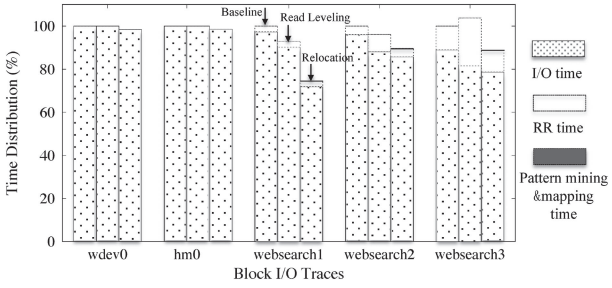

Fig. 8. The distribution of overall processing time of selected traces.

5. Conclusions

We have proposed and evaluated a scheme of write scheduling and data reallocation in SSDs by considering the factor of read disturb. We have built a mathematical model for assessing the read disturb level of block. Therefore, the frequently read data can be flushed to the blocks that are insusceptible to read disturb. Furthermore, we have proposed evenly distributing read data for yielding the minimum read error rate, though carrying out read reclaim tasks.

The evaluation tests show the newly proposed scheme outperforms other comparison counterparts, regarding the measurements of read error rate, read response time, and the overall processing time. In conclusion, our proposal can effectively mitigate the negative impacts of read disturb in modern SSDs.

Acknowledgment

This work was partially supported by “Chongqing Graduate Research and Innovation Project (No. CYS19112)”.

References

[1] R. Bez, *et al.*: “Introduction to flash memory,” Proc. IEEE **91** (2003) 489 (DOI: 10.1109/JPROC.2003.811702).
[2] Y. Cai, *et al.*: “Read disturb errors in MLC NAND flash memory,” arXiv preprint arXiv:1805.03283 (2018).
[3] X. Shi, *et al.*: “Program error rate-based wear leveling for NAND flash memory,” DATE (2018) 1241 (DOI: 10.23919/DATE.2018.8342205).
[4] W. Lee, *et al.*: “Interpage-based endurance-enhancing lower state encoding for MLC and TLC flash memory storages,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **27** (2019) 2033 (DOI: 10.1109/TVLSI.2019.2912228).
[5] Y. Cai, *et al.*: “Data retention in MLC NAND flash memory: Characterization, optimization, and recovery,” HPCA (2015) 551 (DOI: 10.1109/HPCA.2015.7056062).
[6] W. Lee, *et al.*: “Interpage-based endurance-enhancing lower state encoding for MLC and TLC flash memory storages,” Intel Technology Journal (ITJ) **17** (2013) 140.
[7] R. S. Liu, *et al.*: “Optimizing NAND flash-based SSDs via retention relaxation,” FAST (2012) 11.
[8] N. Mielke, *et al.*: “Bit error rate in NAND flash memories,” Reliability Physics Symposium (IRPS) (2008) 9 (DOI: 10.1109/RELPHY.2008.4558857).
[9] B. Choi, *et al.*: “Comprehensive evaluation of early retention (fast charge loss within a few seconds) characteristics in tube-type 3-D NAND flash memory,” IEEE Symposium on VLSI Technology (2016) 1 (DOI: 10.1109/VLSIT.2016.7573385).
[10] K.-T. Park, *et al.*: “A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories,” IEEE J. Solid-State Circuits **43** (2008) 919 (DOI: 10.1109/JSSC.2008.917558).
[11] J.-D. Lee, *et al.*: “Effects of floating-gate interference on NAND flash memory cell operation,” IEEE Electron Device Lett. **23** (2002) 264 (DOI: 10.1109/55.998871).
[12] L. Yang, *et al.*: “Word line interference based data recovery technique for 3D NAND flash,” IEICE Electron. Express **15** (2018) 20180762 (DOI: 10.1587/elex.15.20180762).
[13] N. Li, *et al.*: “A page lifetime-aware scrubbing scheme for improving reliability of flash-based SSD,” IEICE Electron. Express **14** (2017) 20170831 (DOI: 10.1587/elex.14.20170831).
[14] Y. Cai, *et al.*: “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” DATE (2012) 521 (DOI: 10.1109/DATE.2012.6176524).
[15] I. Narayanan, *et al.*: “SSD failures in datacenters: What? when? and why?,” SYSTOR **44** (2016) 1 (DOI: 10.1145/2928275.2928278).
[16] M. Bjorling, *et al.*: “LightNVM: The Linux open-channel SSD subsystem,” FAST (2017) 359.
[17] Y. Cai, *et al.*: “Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery,” DSN (2015) 438 (DOI: 10.1109/DSN.2015.49).
[18] K. Ha, *et al.*: “An integrated approach for managing read disturbs in high-density NAND flash memory,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **35** (2016) 1079 (DOI: 10.1109/TCAD.2015.2504868).
[19] T. Wu, *et al.*: “Flash read disturb management using adaptive cell bit-density with in-place reprogramming,” DATE (2018) 325 (DOI: 10.23919/DATE.2018.8342030).

[20] L. M. Grupp, *et al.*: "The bleak future of NAND flash memory," FAST (2012) 2.
[21] C.-Y. Liu, *et al.*: "Read leveling for flash storage systems," SYSTOR **35** (2015) 1079 (DOI: 10.1145/2757667.2757679).
[22] J. Werner, *et al.*: U.S. Patent Application 13/729966 (2014).
[23] Y. Seo, *et al.*: U.S. Patent 14/081371 (2013).
[24] Y. Sakamoto, *et al.*: *Akaike Information Criterion Statistics* (D. Reidel, Dordrecht, The Netherlands, 1986) 81.
[25] N. Kim and J. Jang: U.S. Patent 8 203 881 (2012).
[26] J. Li, *et al.*: "Frequent pattern-based mapping at flash translation layer of solid-state drives," IEEE Access (2019) 95233 (DOI: 10.1109/ACCESS.2019.2929056).
[27] M. Fukuchi, *et al.*: "20% system-performance gain of 3D charge-trap TLC NAND flash over 2D floating-gate MLC NAND flash for SCM/NAND flash hybrid SSD," ISCAS (2018) 1 (DOI: 10.1109/ISCAS.2018.8351309).
[28] Y. Du, *et al.*: "Enhancing SSD performance with LDPC-aware garbage collection," NVMSA (2017) 1 (DOI: 10.1109/NVMSA.2017.8064481).
[29] D. Narayanan, *et al.*: "Write off-loading: Practical power management for enterprise storage," ACM TOS (2008) 1 (DOI: 10.1145/1416944.1416949).
[30] Search Engine I/O. http://traces.cs.umass.edu/index.php/Storage/Storage.

Appendix A: Mathematical justification to the proposed scheme of data reallocation

Assuming there are `n` blocks, and each block has `m` pages. R_{ij}, P_{ij} and W_{ij} respectively indicate the read count, read error rate, and read error count with respect to the *jth* page in the *ith* block. R_i and W_i represent the read count and the read error count associating with the *ith* block. The variable of W implies the total read error count of all blocks.

The read error rate resulted by ***read operations*** on the block can be measured by the following equation.

$$P_{ij} = \alpha \cdot R_{ij} \tag{A·1}$$

where α is a constant, depending on SSD configurations, such as the number of P/E cycle of the target block.

$$\begin{aligned} W = \sum_{i=1}^{n}\sum_{j=1}^{m} W_{ij} &= \sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij} \cdot \frac{1}{m-1} \cdot \sum_{t\neq j} P_{it} \\ &= \frac{1}{m-1} \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij} \cdot \sum_{t\neq j} \alpha \cdot R_{it} \\ &= \frac{\alpha}{m-1} \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij} \cdot (R_i - R_{ij}) \\ &= \frac{\alpha}{m-1} \cdot \sum_{i=1}^{n}\left(R_i^2 - \sum_{j=1}^{m} R_{ij}^2\right) \\ &= \frac{\alpha}{m-1} \cdot \left(\sum_{i=1}^{n} R_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij}^2\right) \end{aligned} \tag{A·2}$$

The total read error count of all blocks is illustrated in Equation A·2. As seen, $\sum_{i=1}^{n}\sum_{j=1}^{m} R_{ij}^2$, n, m, and α in the equation are constants in a specific case. Considering $\sum_{i=1}^{n} R_i$ is the total read count to all blocks, so that $W \propto \sum_{i=1}^{n} R_i^2$ can reach the minimal value when $R_1 = R_2 = \cdots = R_n$.