Leveraging Approximate Data for Robust Flash Storage

Qiao Li*, Liang Shi‡, Jun Yang§, Youtao Zhang§, Chun Jason Xue*

*City University of Hong Kong, ‡East China Normal University, §University of Pittsburgh

ABSTRACT

With the increasing bit density and adoption of 3D NAND, flash memory suffers from increased errors. To address the issue, flash devices adopt error correction codes (ECC) with strong error correction capability, like low-density parity-check (LDPC) code, to correct errors. The drawback of LDPC is that, to correct data with a high raw bit error rate (RBER), read latency will be amplified. This work proposes to address this issue with the assistance of approximate data. First, studies have been conducted and show there are ample amount of approximate data available in flash storage. Second, a novel data organization is proposed to fortify the reliability of regular data by leaving approximate data unprotected. Finally, a new data allocation strategy and modified garbage collection scheme are presented to complete the design. The experimental results show that the proposed approach can improve read performance by 30% on average comparing to current techniques.

ACM Reference Format:
Qiao Li*, Liang Shi‡, Jun Yang§, Youtao Zhang§, Chun Jason Xue*, *City University of Hong Kong, ‡East China Normal University, §University of Pittsburgh . 2019. Leveraging Approximate Data for Robust Flash Storage. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, Las Vegas, NV, USA, 6 pages. https://doi.org/10.1145/3316781.3317848

1 INTRODUCTION

NAND flash memory is now widely deployed as the storage of mobile devices, laptops, and servers. However, NAND flash is inherently prone to different types of errors, including retention time errors, wearing errors, and voltage disturbance errors. The problem worsens with the increasing bit density and the adoption of 3D NAND. To address this issue, error correction codes (ECC) have been adopted in flash devices to correct errors. Recently, low-density parity-check code (LDPC), which has strong error correction capability, has been adopted in NAND flash memories [1][2]. The principle of LDPC is to use the likelihood information of flash memory cell to correct errors. For data with a high RBER, LDPC requires several sensing iterations to acquire accurate likelihood information, which would significantly degrade the read performance.

To optimize the access performance of LDPC enabled flash memory, previous studies have proposed approaches to exploit the error characteristics on flash memory from different aspects[3][4]. Differently, in this work, approximate data will be leveraged to fortify the data reliability. Approximate data are able to tolerate a predefined number of errors. For example, multimedia data can tolerate some errors without impacting the user experience. Recent studies found that approximate data are seen in various applications. Previous works proposed to exploit the error tolerance for performance and energy optimization [5][6][7][8]. One common approach is to relax the reliability requirement on approximate data by reducing the noise margin between the threshold voltage states of flash memory [8][9]. However, this scheme needs the support of multiple programming voltages from flash chip, which is not always available for the designers. Another strategy is to apply ECC with reduced error correction strength for approximate data [6][7]. However, implementing multiple ECCs for flash memory introduces significant hardware and power costs to the flash devices. In addition, none of these work proposed to exploit approximate data for read performance improvement. Different from previous work, we analyze a set of data with error tolerance capability and propose a new scheme to exploit it for read performance improvement.

In this work, we first analyze a large amount of accessed data in the flash storage of mobile devices and personal computers and find some characteristics. For example, when watching an online video, the video file will be cached. When the cache is full, video data will be written to the flash storage. The cached video data can endure a high RBER due to 2 reasons: 1) most of these data are encoded multimedia data, which presents high error tolerance; 2) even if they are corrupted, the system can also download them from the internet. This paper defines this type of data as approximate data, and other data as regular data that are guaranteed to be error free. Then, based on the observation, we propose a scheme to leverage the error tolerance characteristic of approximate data for read performance improvement. The basic idea of the scheme is to organize both approximate data and regular data in one data page. The error correction capability of the page will all be used to correct the errors in regular data while approximate data will be stored without ECC protection. Basically, this is achieved by combining regular data with know data, like all '1's to form the encoded information in ECC codeword. In this case, the RBER of encoded data is reduced and the access performance of regular data will be improved. There are several challenges to realize the aforementioned idea. First, each ECC codeword needs to be constructed with approximate data and regular data for reliability enhancement. Second, if approximate data have a large number of errors, they should be set as invalid. To solve above challenges, several approaches are proposed. First, a data allocation scheme is proposed to construct the ECC codeword during host accesses. Second, a new garbage collection (GC) scheme is proposed to construct more ECC codewords with the

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06...$15.00
https://doi.org/10.1145/3316781.3317848

proposed design. Once RBER of approximate data reaches a predefined threshold, they will be set as invalid. Note that from our studies, we find that the defined approximate data always have a short retention time. This is because they are only used for cache for a short time and persisted after checkpoint or data syncing for consistency consideration. With this characteristic, errors rarely happen in the defined approximate data. Experimental results show that the proposed approach can improve read performance by 12% in early life stage and 30% in late life stage of flash storage. The following contributions are made in this paper.

- Identified that a large amount of data generated during application execution is error-tolerance;
- Introduced a new ECC encoding scheme to enhance the reliability of regular data through leveraging the error tolerance of approximate data;
- Proposed a data allocation scheme and GC scheme to assist the implementation of the new ECC encoding design;
- Implemented and evaluated the proposed scheme. Experiments show encouraging performance improvement over the current schemes.

In the remaining of this paper, Section 2 presents the background and related work. Section 3 is the motivation. Section 4 presents the proposed approaches. Section 5 shows the experiments and results. Section 6 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Background

NAND flash memory stores charges in flash cells and represents data based on the amount of charges. For the flash memory with n bits per cell, the threshold voltage is divided into 2^n regions to represent different data. If the threshold voltages of some flash cells shift out of their original regions due to charge leakage over time or charge increasing from interference, raw bit errors will happen on the stored data. To increase the storage density and capacity, the trend is to store more bits in one cell and decrease the technology size, which leads to narrow voltage regions. In addition, the advent of 3D NAND introduces more errors because of its complex structure. As a result, NAND flash memory becomes more vulnerable to various types of errors.

The common solution to recover flash errors is to apply error correction codes. For each unit of data, ECC stores a certain amount of redundant bits in the out-of-band areas of the flash pages. A limited number of raw bit errors can be corrected, where the number is determined by the strength of the applied ECC. Low-density parity-check (LDPC) code has been adopted on NAND flash memory, due to its strong error correction capability. Different from traditional ECC codes, LDPC uses both hard and soft decoding to correct data with high RBER. The process involves soft information obtaining, which needs multiple voltage sensing operations on flash cells. Thus, LDPC needs longer access latency to decode data, especially for data with a high RBER, which deteriorates flash performance.

2.2 Related Work

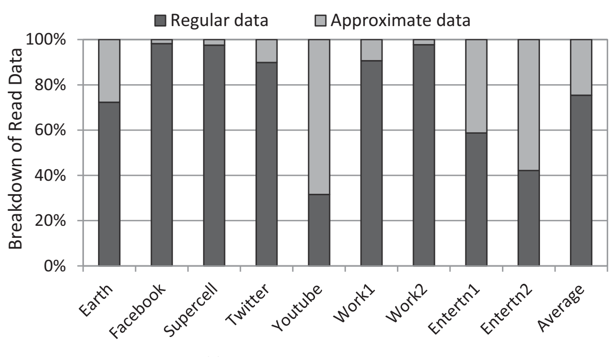
To improve the read performance on LDPC based flash memory, previous studies have proposed approaches from different aspects. The first group of work [10][1][3][4] leveraged specific error characteristics on NAND flash memory to optimize the soft information gaining strategy of read operations for LDPC decoding. The second group of work [2][11] proposed approaches to make tradeoffs in exchange of RBER reduction on read-critical data which will greatly improve the read performance, thus improving the overall access performance. The third group of work [12][13] strived to reduce RBER from the view of cell voltage characteristics on NAND flash memory. By contrast, our work will leverage the error tolerance of approximate data to fortify the reliability of regular data.

Prior work has studied the error tolerance of the data in different applications and proposed approaches to store approximate data [5][6][7][8]. It was found that data from many applications, such as multimedia, scientific computation, and cloud computing, shows high error resiliency, and can produce acceptable results even with a high RBER. Previous studies have proposed different storage schemes by leaving some errors to approximate data. Sampson et al. [9] and Cui et al. [8] proposed to tighten the noise margins between the threshold voltage states. Holcomb et al. [14] and Nelson et al. [15] proposed to increase the storage density for approximate data. Guo et al. [6] and Jevdjic et al. [7] adopted multiple ECCs for data with different error requirements. Xu et al. [5] further proved that it is possible for some approximate data to achieve the data-level error tolerance without using any ECC protections. *Different from all of these work, this paper is the first one on exploiting approximate data for read performance improvement.* In addition, the proposed scheme does not introduce any extra hardware overheads.

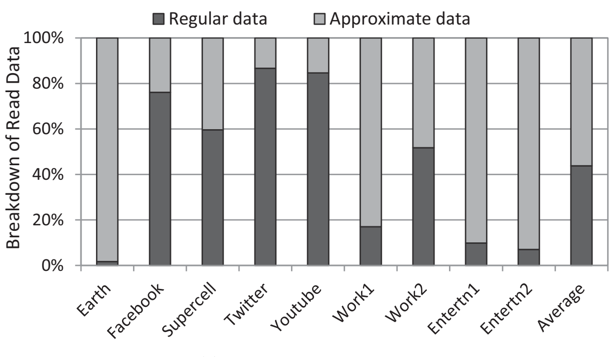
3 MOTIVATION

During the last decades, the amount of data has increased significantly for various storage systems, from mobile, personal computers, to data centers. Among all the data stored in storage devices, there are a great deal of cache data and temporary data during the execution of applications. For example, when we watch an online video on mobile phones, the video file shall be downloaded from the servers and maintained in the cache. When the cache is full, video data will be evicted out of the cache and be written to flash memory. As presented above, ECC is applied to eliminate raw bit errors for all the data stored in storage. However, for those cached data stored in flash memory, the reliability of them is not a necessary requirement. If some bit errors occur, resulting that users cannot access the data, users can just download another copy from the server. This paper defines this type of data as approximate data, since the data can tolerate errors without affecting normal execution of applications.

To study the amount and percentage of approximate data during application execution, several representative workloads are collected in the block layer on a mobile phone and a personal computer, which adopts NAND flash memory as storage devices. By extracting the type of data file of each request, Figure 1 presents the ratio of regular and approximate data in the workloads. The results show that all the workloads access both regular and approximate data. The percentages of approximate data for write request are much higher than that for read requests. These written cached data



(a) Breakdown of read data



(b) Breakdown of write data

Figure 1: Breakdown of read and write data. The accessed data in flash memory includes both regular and approximate data.

are often deleted after manually controlled system acceleration on mobile devices or power off on personal computers. This characteristic further confirms that we can relax the reliability requirement on these data. In addition, based on the analysis on the lifetime of these data, we found that they are kept for no more than 1 hour. Most of the data are deleted by the system in seconds. This is true since these data are cached data. Therefore, this work proposes to leave them without ECC protection. We use error detection code (EDC), which consumes much less overheads than ECC, to detect the errors on the data during read operations, in case they are corrupted due to too many errors. If the detected error bits are above a threshold, which will rarely happen, they will be set as invalid in the flash memory and downloaded again from servers.

4 ROBUST FLASH DESIGN

4.1 Overview

Figure 2 shows the overview of the proposed approach. A new data organization is proposed to enhance the error correction capability on regular data, while approximate data is written without ECC protection. This design enables multiple error correction capabilities with only one ECC, which involves no extra ECC overhead. In the following, the new ECC encoding scheme is first presented. Then, a new data allocation scheme and a GC scheme are proposed to construct the encoding units.

4.2 Robust ECC

In traditional ECC encoding, the total length of one ECC codeword is n, the length of user data, which can be either regular data or approximate data, is k, and the remaining $n-k$ bits are ECC parity. Inside each flash page, the OOB area is used to store ECC parity bits and other information. Since the space for parity is limited, the error correction strength for a data page is limited. The basic idea is to store both regular data and approximate data in one page and apply all the error correction capability of the page on regular data.

In the new design, $k/2$-bit approximate data will not be included in the ECC encoding due to its error tolerance. All $(n-k)$-bit parity will be used to protect $k/2$-bit regular data. First, $k/2$-bit known information, for example, all-one vector $\mathbf{1}_{k/2}$, will be combined with the regular data vector $\mathbf{r}_{k/2}$ to construct a new k-bit user data vector $[\mathbf{r}_{k/2}, \mathbf{1}_{k/2}]$. Second, ECC encoding will be conducted on the k-bit vector to generate $(n-k)$-bit parity $\mathbf{p}_{n-k}$. Third, the

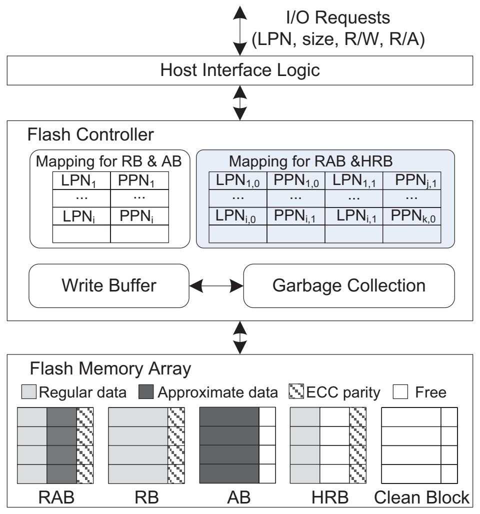


Figure 2: Overview of the proposed robust flash memory.

known information is removed from the vector. Finally, the encoded codeword, which includes three parts, $k/2$-bit approximate data, $k/2$-bit regular data and $(n-k)$-bit parity, are programmed to the flash memory.

During read operation, $k/2$-bit approximate data will be read out directly without ECC decoding. All-one vector $\mathbf{1}_{k/2}$ will be automatically combined with $k/2$-bit sensed regular data to form a k-bit vector, ready for ECC decoding. Since there are no bit errors in the known information part, the RBER of the ECC codeword will be reduced. Suppose there are t-bit errors in one ECC codeword, the RBER of the original codeword will be calculated as t/n, denoted as $RBER_1$. The RBER of the new codeword $RBER_2$ can be expressed as:

$$RBER_2 = RBER_1 \cdot (1 - \frac{k}{2n}). \quad (1)$$

As an example, for an ECC with code rate of 8/9, the RBER can be reduced to 5/9. With the reduction of RBER, two objects can be optimized for NAND flash memory. First, in the early lifetime, since read latency depends on the RBER of data, the read performance can be improved on these data. Second, in the late lifetime, with the same ECC, the acceptable RBER on the regular data is higher and therefore the lifetime can be improved.

4.3 Data Allocation

The above robust ECC design can benefit the performance and lifetime of NAND flash memory. However, it requires the incoming write requests to access equal size of regular and approximate data, which cannot be guaranteed in different applications. Based on the access ordering, four types of blocks are defined and the data of write requests will be programmed to one type of the blocks based on the ratio of regular and approximate data in current request queue. 1) **RAB** (Regular and Approximate Block): Both regular data and approximate data are stored in each page, organized in the way described above. Inside this type of blocks, the reliability of regular data is improved; 2) **RB** (Regular Block): Only regular data are stored in the traditional way. k bits regular data are encoded with $n - k$ bits ECC parity; 3) **HRB** (Half Regular Block): Only regular data are stored by occupying half of the user-data space in each page, leaving the other half to be empty. These blocks are used to replace **RB** at the end of lifetime when approaching the maximum P/E cycles for reliability concern. 4) **AB** (Approximate Block): Only approximate data are stored without ECC protection.

Before the end of flash lifetime, ECC can provide error correction capability for k bits user data. During this period, three active blocks, including **RAB**, **RB** and **AB**, are maintained in each flash plane for programming. The goal is to maximize the amount of regular data written into **RAB** to enhance the reliability. In the flash controller, a write buffer is equipped to accommodate the data of the dispatched requests, where the data can be reorganized before programming. This work utilizes the write buffer to reorganize the approximate data and regular data written to **RAB**. If there exist both regular and approximate pages in the write request queue, a regular data page and an approximate data page will be selected to construct two new data pages written to a **RAB**. If there are only regular/approximate pages, a regular/approximate page will be selected, written to a **RB**/**AB**.

As an example, if there are 5 pages of regular data and 3 pages of approximate data for writing, the first 3 regular data pages and all the 3 approximate data pages will be reorganized in the write buffer. Afterwards, the 6 organized pages of data will be processed and the data will be written to a **RAB**. The remaining 2 regular data pages will be written to a **RB**.

Flash Translation Layer Design: To support the strengthened ECC design, the logical page number to physical page number (L2P) mapping needs to be revised. Assume the baseline strategy is page mapping. For **RB** and **AB**, the L2P mapping is the same as traditional mechanism. For **RAB** and **HRB**, the mapping information will be doubled since the mapping unit is half page. As shown in Figure 2, one bit will be added to differentiate the two parts. Note that the block type is determined during runtime and a clean block can be used as any type.

4.4 Garbage Collection

The above data allocation scheme strives to write data into **RAB**s. There can be some data still written to other types of blocks. A new GC strategy is proposed to select two blocks, one **RB/HRB** and one **AB**, for GC and copy all the valid pages into a **RAB**.

Denote IV_{RA}, IV_R and IV_A as the largest number of invalid pages of **RAB**, **RB** and **AB** blocks respectively. The corresponding three blocks are B_{RA}, B_R and B_A. The largest number of invalid pages among all the blocks is:

$$IV_{max} = max(IV_{RA}, IV_R, IV_A) \tag{2}$$

Considering the GC efficiency, default greedy GC strategy will choose the block whose invalid page number is IV_{max}, as the victim block for GC, to minimize the number of valid page copy. The proposed scheme takes both reliability enhancement for regular data and GC efficiency into consideration. If the following condition is satisfied, the block pair B_R and B_A will be chosen for GC.

$$IV_{max} - min(IV_R, IV_A) \leq T \tag{3}$$

In the expression, the threshold T bounds the influence on GC efficiency. If the condition is not satisfied, the block with the largest number of invalid pages will be chosen as the victim block.

As an example, we assume each block has 64 pages and IV_{RA}, IV_R and IV_A equal to 62, 62 and 61 respectively. If T is set as 2, the condition is satisfied and B_R and B_A will be selected. There are 2 valid pages in B_R and 3 valid pages in B_A required to be copied. On average, each block has 61.5 invalid pages. The GC efficiency, which is defined as the percentage of invalid page, is 96.1%. While traditional GC strategy will choose B_{RA} or B_R as victim block and one block has 62 invalid pages with GC efficiency 96.9%. The GC efficiency of the proposed approach is sacrificed a little to promote data allocation to **RAB**. If T is set as 1, there won't be GC efficiency loss in the example.

Note that **RB** will be replaced by **HRB** at the end of flash lifetime. Since **HRB**s only store data by using half of the space, the condition will be changed to:

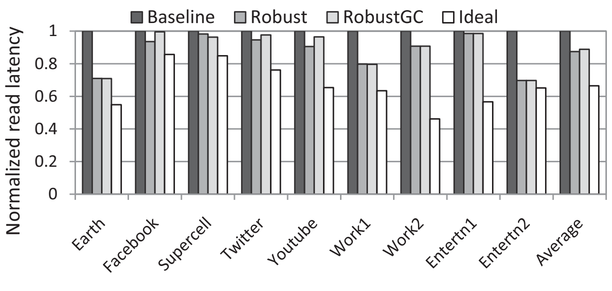
$$IV_{max} - min(2 * IV_{HR}, IV_A) \leq T \tag{4}$$

After selecting the victim block(s), the valid page will first be copied to the write buffer. Similar to the process of write requests, the page pairs including one regular data page and one approximate data page will be prioritized to be written to a **RAB**. The remaining data will be written to a block in one of the other types based on the data type.

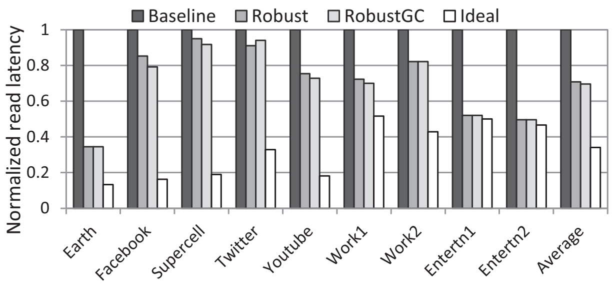
5 EXPERIMENT

5.1 Experimental Setup

The proposed approach is evaluated on a widely-used trace-driven simulator, SSDSim [16]. The simulated flash device has two channels, with two chips per channel, and four planes per chip. Inside each plane, there are 256 blocks, each of which contains 64 4KB pages. The workloads in the experiments are all collected in the block layer of storage systems, which adopt flash memory as storage devices. The first 5 workloads are collected on mobile devices during executing target applications. The other 4 workloads are collected on a personal computer, which simulate two scenarios, work and entertainment. For the 2 work workloads, the user mainly operates on documents and occasionally surfs on the Internet to search for browsing and downloading documents. For the 2 entertainment workloads, the user mainly watches videos and browses the latest news. One tag is added to differentiate regular data and approximate data based on the type of the collected data, same to previous work [8].



(a) Normalized read latency of early life stage



(b) Normalized read latency of late life stage

Figure 3: Read latency comparison of the proposed approach.

5.2 Experimental Results

The following four schemes are evaluated for comparison. *Baseline* is the traditional approach by protecting all the data with ECC. *Robust* is the proposed scheme with four types of blocks and the data allocation scheme. *RobustGC* is *Robust* plus GC optimization. *Ideal* is the ideal case by exploiting approximate data for maximizing performance, where all regular data are stored in **RAB** or **HRB**.

In *RobustGC*, we will change **RB** into **HRB** to extend flash lifetime at late life stage, which will introduce space reduction. The reduction is closely related to the workload characteristics. If all the blocks are used as **HRB**, the space is reduced to 50%, which is the worst case. Suppose the space is always enough to accommodate all write requests when using **HRB**, by using ECC with code rate of 8/9, the lifetime of *Robust* can be extended by 15%. In the following, we mainly compare the performance before the lifetime ending.

Access Performance: Figure 3 shows the normalized read latency in the early and late life stages. Compared to *Baseline*, the proposed approach can averagely reduce read latency by 12% and 30% in early and late life stages, respectively. The improvement comes from two aspects. First, the RBER on regular data is reduced, thus the read latency for LDPC decoding is reduced. Second, approximate data are stored without ECC encoding, where the read latency is also reduced. In early stage, the performance improvement is insignificant and *RobustGC* has worse performance than *Robust*. This is because the RBER is low which requires short read latency. The error reduction won't introduce significant benefits and even *Ideal* gains not much improvement. In this case, the benefits achieved from GC optimization are smaller than the interference from increased GC operations, which explains why *RobustGC* is worse than *Robust*. In late stage, with the increasing of P/E cycles, RBER is increased and therefore the performance improvement is greater. *RobustGC* has better performance than *Robust* for most workloads because the proposed GC scheme constructs more **RAB**s. One exception is Twitter, where GC with increased overhead introduces interference to host I/O. This issue can be avoided by changing the threshold setting for invalid page number.

Block Distribution: Figure 4 shows the distribution of blocks in different types for *RobustGC*. Note that there are only three types of blocks as explained above since **HRB** is only used in late stage. Among these blocks, **RAB** takes up 10% while either **RB** or **AB** dominates for different workloads. Even with GC modification, the percentage of **RAB** is low, therefore the overhead for mapping is low for FTL design.

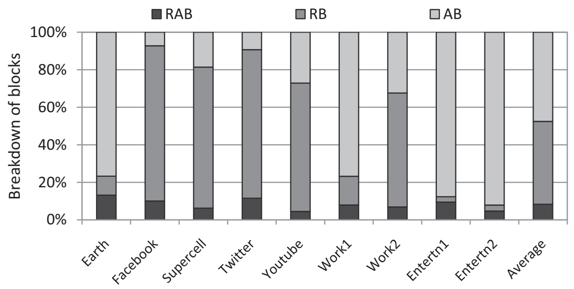
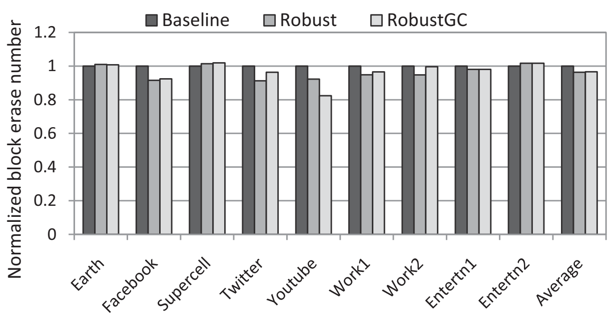


Figure 4: The breakdown of blocks in different types.

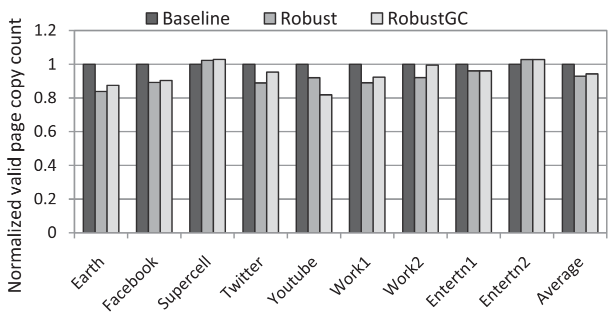
GC Impacts: Figure 5 presents the normalized block erase number and valid page copy number. For some workloads, such as Earth and EnterTn2, *Robust* and *RobustGC* have a little more GC operations, which introduces slight increasing on erase operations and valid page copies. Correspondingly, the read performance for the workloads achieves significant improvement, by more than 50%. Therefore, these workloads sacrifice small GC efficiency to gain better read performance. While for some workloads, such as Facebook and Youtube, the number of GC operations is reduced, with less erase operations and copies. This is because regular data and approximate data usually have different update frequencies. Since the proposed scheme separates two types of data into different blocks, the data are actually separated based on data hotness. As a result, the GC efficiency is improved with less performance improvement.

Based on the results, there exists a tradeoff between read performance and GC efficiency. If there are more **RAB**s, the performance will be better but the GC efficiency is worse and vice-versa. The number of **RAB** depends on the access characteristics of the applications. Write requests accessing both regular and approximate data in a short period of time will promote the allocation of **RAB**. How to allocate data with further consideration of workload characteristics will be our future work.

Sensitivity: The threshold of invalid page number T to decide how to conduct GC greatly impacts the performance of *RobustGC*. Sensitivity studies on T are conducted with setting T to 1, 2 and 3. Figure 6(a) shows the normalized read latency, where we have the following two observations. First, the read performance by setting T to 2 is the best on average among the three settings. A small value for T results in small number of **RAB**, thus read performance has little improvement. While a large value results in large number of GC which may interfere host I/O requests and impact access performance. Second, the best setting for different workloads is

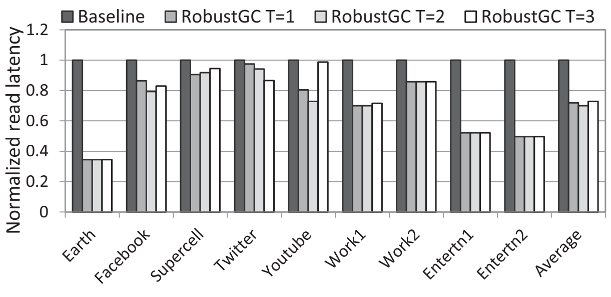


(a) Normalized block erase number

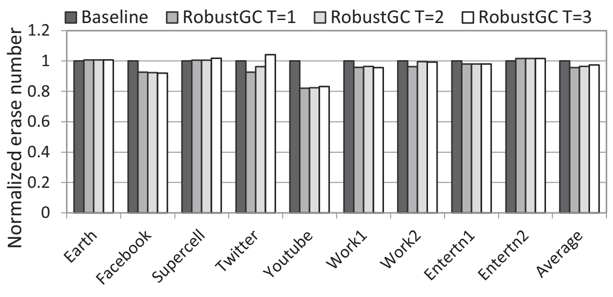


(b) Normalized valid page copy number

Figure 5: The normalized block erase number and valid page copy number in GC.



(a) Normalized read latency



(b) Normalized block erase number

Figure 6: Sensitivity study on the threshold.

different due to the variation of the workload characteristics. This indicates a dynamic setting instead of static setting is required, which will be studied in our future work.

Figure 6(b) shows the normalized block erase number. With the increasing of T, the erase count is slightly increased because more block pairs will be chosen for GC. Nevertheless, the difference is negligible.

6 CONCLUSION

To address the read performance issue of LDPC based NAND flash memory, this work leverages the error tolerance of approximate data. By collecting the accessed data type of real applications, we found that a lot of approximate data exists in the flash storage. Due to the fact that approximate data can endure a high RBER, this work proposes to leave them without ECC protection and move the error correction capability on regular data to enhance the reliability. A new ECC codeword design is proposed to enable the realization of the idea. Then, a data allocation and a GC strategy are presented. Experimental results show that the proposed approaches can significantly improve read performance.

ACKNOWLEDGMENT

This work is supported by NSFC 61772092 and 61572411. The corresponding author is Liang Shi (shi.liang.hk@gmail.com).

REFERENCES

[1] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng, “LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives,” in *FAST*, 2013, pp. 244–256.

[2] Q. Li, L. Shi, C. J. Xue, K. Wu, C. Ji, Q. Zhuge, and E. H.-M. Sha, “Access characteristic guided read and write cost regulation for performance improvement on flash memory,” in *FAST*, 2016, pp. 125–132.

[3] Q. Li, L. Shi, C. J. Xue, Q. Zhuge, and E. H.-M. Sha, “Improving ldpc performance via asymmetric sensing level placement on flash memory,” in *ASP-DAC*, 2017, pp. 560–565.

[4] Y. Du, D. Zou, Q. Li, L. Shi, H. Jin, and C. J. Xue, “Laldpc: Latency-aware ldpc for read performance improvement of solid state drives.” MSST, 2017, pp. 1–13.

[5] X. Xu and H. H. Huang, “Exploring data-level error tolerance in high-performance solid-state drives,” *TR*, vol. 64, no. 1, pp. 15–30, 2015.

[6] Q. Guo, K. Strauss, L. Ceze, and H. S. Malvar, “High-density image storage using approximate memory cells,” in *ACM SIGPLAN Notices*, vol. 51, no. 4, 2016, pp. 413–426.

[7] D. Jevdjic, K. Strauss, L. Ceze, and H. S. Malvar, “Approximate storage of compressed and encrypted videos,” *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 361–373, 2017.

[8] J. Cui, Y. Zhang, L. Shi, C. J. Xue, W. Wu, and J. Yang, “Approxftl: On the performance and lifetime improvement of 3-d nand flash-based ssds,” *TCAD*, vol. 37, no. 10, pp. 1957–1970, 2018.

[9] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, “Approximate storage in solid-state memories,” *TOCS*, vol. 32, no. 3, p. 9, 2014.

[10] G. Dong, N. Xie, and T. Zhang, “Enabling NAND flash memory use soft-decision error correction codes at minimal read latency overhead,” *TOCS*, vol. 60, no. 9, pp. 2412–2421, 2013.

[11] Q. Li, L. Shi, Y. Di, Y. Du, C. J. Xue, and H. Edwin, “Exploiting process variation for read performance improvement on ldpc based flash memory storage systems,” in *ICCD*, 2017, pp. 681–684.

[12] T. Nakamura, Y. Deguchi, and K. Takeuchi, “Aep-ldpc ecc with error dispersion coding for burst error reduction of 2d and 3d nand flash memories,” in *IMW*, 2017, pp. 1–4.

[13] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen, “Flexlevel: a novel NAND flash storage system design for ldpc latency reduction,” in *DAC*, 2015, pp. 1–6.

[14] D. E. Holcomb and K. Fu, “Qbf-based synthesis of optimal word-splitting in approximate multi-level storage cells,” in *WACAS*, 2014.

[15] J. Nelson, A. Sampson, and L. Ceze, “Dense approximate storage in phase-change memory,” *ASPLOS Ideas & Perspectives*, 2011.

[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,” *TC*, vol. 62, no. 6, pp. 1141–1155, 2013.