Reducing Solid-State Drive Read Latency by Optimizing Read-Retry

Jisung Park[1] Myungsuk Kim[2,3] Myoungjun Chun[2] Lois Orosa[1] Jihong Kim[2] Onur Mutlu[1]

[1]ETH Zürich Switzerland
[2]Seoul National University Republic of Korea
[3]Kyungpook National University Republic of Korea

ABSTRACT

3D NAND flash memory with advanced multi-level cell techniques provides high storage density, but suffers from significant performance degradation due to a large number of read-retry operations. Although the read-retry mechanism is essential to ensuring the reliability of modern NAND flash memory, it can significantly increase the read latency of an SSD by introducing multiple retry steps that read the target page again with adjusted read-reference voltage values. Through a detailed analysis of the read mechanism and rigorous characterization of 160 real 3D NAND flash memory chips, we find new opportunities to reduce the read-retry latency by exploiting two advanced features widely adopted in modern NAND flash-based SSDs: 1) the `CACHE READ` command and 2) strong ECC engine. First, we can reduce the read-retry latency using the advanced `CACHE READ` command that allows a NAND flash chip to perform consecutive reads in a pipelined manner. Second, there exists a large ECC-capability margin in the final retry step that can be used for reducing the chip-level read latency. Based on our new findings, we develop two new techniques that effectively reduce the read-retry latency: 1) *Pipelined Read-Retry (PR^2)* and 2) *Adaptive Read-Retry (AR^2)*. PR^2 reduces the latency of a read-retry operation by pipelining consecutive retry steps using the `CACHE READ` command. AR^2 shortens the latency of each retry step by dynamically reducing the chip-level read latency depending on the current operating conditions that determine the ECC-capability margin. Our evaluation using twelve real-world workloads shows that our proposal improves SSD response time by up to 31.5% (17% on average) over a state-of-the-art baseline with only small changes to the SSD controller.

CCS CONCEPTS

• **Hardware → External storage**.

KEYWORDS

solid state drives (SSDs), NAND flash memory, latency, read-retry

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446719

ACM Reference Format:
Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. Reducing Solid-State Drive Read Latency by Optimizing Read-Retry. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3445814.3446719

1 INTRODUCTION

NAND flash memory is the prevalent technology for architecting storage devices in modern computing systems to meet high storage-capacity and I/O-performance requirements. 3D NAND technology and advanced multi-level cell (MLC) techniques enable continuous increase of storage density, but they negatively affect the reliability of modern NAND flash chips. NAND flash memory stores data as the *threshold voltage (V_{TH}) level* of each flash cell, which depends on the amount of charge in the cell. New cell designs and organizations in 3D NAND flash memory cause a flash cell to leak its charge more easily [7, 65, 66]. In addition, MLC technology significantly reduces the margin between different V_{TH} levels used to store multiple bits in a single flash cell. Consequently, the V_{TH} level of a 3D NAND flash cell with advanced MLC techniques (e.g., triple-level cell (TLC) [6, 37] or quad-level cell (QLC) [32, 44]) can quickly shift beyond the read-reference voltage V_{REF} (i.e., the voltage used to distinguish between cell V_{TH} levels) after programming, which results in an error when reading the cell.

To guarantee the reliability of stored data, a modern SSD commonly adopts two main approaches. First, a modern SSD employs strong *error-correcting codes (ECC)* that can detect and correct several tens of raw bit errors (e.g., 72 bits per 1-KiB codeword [73]). Second, when ECC fails to correct all bit errors, the SSD controller performs a *read-retry operation* that reads the erroneous page[1] again with *slightly-adjusted* V_{REF} values. Since bit errors occur when the V_{TH} levels of flash cells shift beyond the V_{REF} values, sensing the cells with appropriately-shifted V_{REF} values can greatly reduce the number of raw bit errors [6–8, 10–14, 16, 64–66, 84].

Although read-retry is essential to ensuring the reliability of modern NAND flash memory, it comes at the cost of significant performance degradation. A read-retry operation *repeats* a retry step until it finds V_{REF} values that allow the page's raw bit-error rate (RBER) to be lower than the ECC correction capability (i.e., the number of errors correctable) or finds for sure that the page cannot be read without errors. Recent work [84] shows that a modern SSD with long *retention age values* (i.e., how long data is stored after it is programmed) and high program/erase (P/E) cycles (i.e., how

[1]A NAND flash memory concurrently reads and writes multiple cells at a page (e.g., 16 KiB) granularity (see Section 2.2).

many writes/erases are performed) suffers from a large number of read-retry operations that increase the read latency linearly with the number of retry steps. Our experimental characterization using 160 real 3D TLC NAND flash chips, in this work, shows that a read frequently incurs *multiple* retry steps even under modest operating conditions. For example, under a 3-month data retention age at *zero* P/E cycles (i.e., at the beginning of SSD lifetime), we observe that every read requires more than three retry steps.

Prior works [12, 13, 64–66, 77, 84] attempt to reduce the number of retry steps by quickly identifying near-optimal V_{REF} values, but read-retry is a fundamental problem that is *difficult to completely eliminate* in modern SSDs. For example, an existing technique [84] reads a page using V_{REF} values that have been recently used for a read-retry operation on other pages exhibiting similar error characteristics with the page to read. Doing so significantly reduces the number of retry steps by starting a read (and retry) operation with the V_{REF} values close to the *optimal* read-reference voltage (V_{OPT}) values. However, this technique cannot completely avoid read-retry: *every read* still incurs at least three retry steps in an aged SSD [84]. This is because, in modern NAND flash memory, the V_{TH} levels of flash cells change quickly and significantly over time, which makes it extremely difficult to identify the exact V_{REF} values that can avoid read-retry before reading the target page.

In this paper, we identify new opportunities to reduce the read-retry latency by exploiting two advanced architectural features widely adopted in modern SSDs: 1) the *CACHE READ command* [55, 67, 69] and 2) *strong ECC engine* [6, 7]. First, we find that it is possible to reduce the total execution time of a read-retry operation using the CACHE READ command that allows a NAND flash chip to perform consecutive reads in a pipelined manner. Since each retry step is effectively the same as a regular page read, the CACHE READ command also enables concurrent execution of consecutive retry steps in a read-retry operation.

Second, we find that a large ECC-capability margin exists in the final retry step. Although a read-retry occurs when the read page's RBER exceeds the ECC capability (i.e., when there is no ECC-capability margin), once a read-retry operation succeeds, it allows the page to be eventually read *without* any uncorrectable errors (i.e., there exists a *positive* ECC-capability margin in the final retry step). We hypothesize that the ECC-capability margin is large due to two reasons. First, a modern SSD uses *strong* ECC that can correct several tens of raw bit errors in a codeword. Second, in the final retry step, the page can be read by using *near-optimal* V_{REF} values that drastically decrease the page's RBER. If we can leverage the large ECC-capability margin to reduce the *page-sensing latency* tR, it allows not only the final retry step to quickly read the page without uncorrectable errors but also the earlier retry steps (which would fail anyway even with the default tR) to be finished more quickly. To validate our hypothesis, we characterize 1) the ECC-capability margin in each retry step and 2) the impact of reducing tR on the page's RBER, using 160 real 3D TLC NAND flash chips. The results show that we can safely reduce tR of each retry step by 25% even under the worst-case operating conditions prescribed by manufacturers (e.g., a 1-year data retention age [24] at 1.5K P/E cycles [73]).

Based on our findings, we develop two new read-retry mechanisms that effectively reduce the read-retry latency. First, we propose *Pipelined Read Retry (PR²)* that performs consecutive retry steps in a pipelined manner using the CACHE READ command. Unlike the regular read-retry mechanism that starts a retry step *after* finishing the previous retry step, PR² performs page sensing of a retry step during data transfer of the previous retry step, which removes data transfer and ECC decoding from the critical path of a read-retry operation, reducing the latency of a retry step by 28.5%. Second, we introduce *Adaptive Read Retry (AR²)* that performs each retry step with reduced page-sensing latency (tR), leading to a further 25% latency reduction even under the worst-case operating conditions. Since reducing tR inevitably increases the read page's RBER, an excessive tR reduction can potentially cause the final retry step to fail to read the page without uncorrectable errors. This, in turn, introduces one or more additional retry steps, which could increase the overall read latency. To avoid increasing the number of retry steps, AR² uses the best tR value for a certain operating condition that we find via extensive and rigorous characterization of 160 real 3D NAND flash chips.

Our two techniques require only small modifications to the SSD controller or firmware but no change to underlying NAND flash chips. This makes our techniques easy to integrate into an SSD along with existing techniques that aim to reduce the *number* of retry steps per read-retry operation [12, 13, 64–66, 77, 84]. Our evaluation using twelve real-world workloads shows that our two techniques, when combined, significantly improve the SSD response time by up to 50.8% (35.2% on average) in a baseline high-end SSD. Compared to a state-of-the-art research baseline [84], our proposal reduces SSD response time by up to 31.5% (17% on average) in read-dominant workloads.

This paper makes the following key contributions:

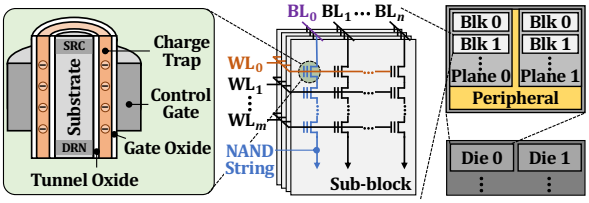
- To our knowledge, this work is the first to identify new opportunities to reduce the latency of each retry step by exploiting advanced architectural features widely adopted in modern SSDs.
- Through extensive and rigorous characterization of 160 real 3D TLC NAND flash chips, we make three new observations on modern NAND flash memory. First, a read-retry occurs frequently even under modest operating conditions (Section 3.1). Second, when a read-retry occurs, there is a large ECC-capability margin in the final retry step even under the worst-case operating conditions (Section 5.1). Third, there is substantial margin in read-timing parameters, which enables safe reduction of the page-sensing latency in a read-retry operation (Section 5.2).
- Based on our findings and characterization results, we propose two new techniques, PR² and AR², which effectively reduce the latency of each retry step, thereby reducing overall read latency. Our techniques require only very small changes to the SSD controller or firmware. By reducing the latency of each retry step while keeping the same number of retry steps during a flash read, our proposal effectively complements existing techniques [12, 13, 64–66, 77, 84] that aim to reduce the number of retry steps, as we empirically demonstrate (Section 7).

2 BACKGROUND

We provide brief background on relevant aspects of NAND flash memory necessary to understand the rest of the paper.

2.1 NAND Flash Organization

NAND flash memory is hierarchically organized. Figure 1 illustrates the organization of a 3D NAND flash chip. Multiple (e.g., 24 to 176) flash cells (Figure 1(a)) are vertically stacked and form a *NAND string* (Figure 1(b)) that is connected to a bitline (BL). NAND strings at different BLs compose a *sub-block*. The control gate of each cell at the same vertical location in a sub-block is connected to the same wordline (WL), which makes all the cells at the same WL operate concurrently. A *block* consists of several (e.g., 4 to 8) sub-blocks, and thousands (e.g., 3,776 [37]) of blocks constitute a *plane*. A NAND flash chip contains multiple *dies* (Figure 1(c)), each of which comprises multiple planes (e.g., two or four planes per die [32]). Dies in a NAND flash chip can operate independently of each other, while planes in a die can concurrently operate under limited conditions as they usually share the same row decoder [31].



(a) Flash cell (b) NAND block (c) NAND flash chip

Figure 1: Organization of 3D NAND flash memory.

A flash cell encodes bit data using its threshold voltage (V_{TH}) level. As shown in Figure 1(a), a flash cell has a special material, called a charge trap[2], which can hold electrons without power supply. The larger the number of electrons in the charge trap, the higher the cell's V_{TH} level. In single-level cell (SLC) NAND flash memory, for example, a cell can encode one bit of data by encoding its high and low V_{TH} levels as '0' and '1', respectively.

2.2 NAND Flash Operation

Three basic operations enable access to NAND flash memory: 1) program, 2) erase, and 3) read.

Program and Erase Operations. A program operation *injects* electrons into a cell's charge trap from the substrate by applying a high voltage (> 20 V) to the WL, which increases the cell's V_{TH} level (i.e., program operation can only change a cell's data from '1' to '0' assuming the SLC encoding described above). As a set of flash cells are connected to a single WL in NAND flash memory (i.e., the same voltage is applied to the control gate of every cell in the same WL), data is written at *page granularity* (e.g., 16 KiB) such that each cell at the same WL stores one bit of the page.

An erase operation *ejects* electrons from a cell's charge trap by applying a high voltage (> 20 V) to the substrate, which decreases the cell's V_{TH} level. As program and erase operations are *unidirectional*, a page needs to be erased first to program data (*erase-before-write*). A NAND flash chip performs an erase operation at block granularity (for cost reasons). This leads to a high erase bandwidth because a block consists of hundreds (e.g., 576 [37]) or thousands (e.g., 1,472 [44]) of pages, but also causes the erase latency `tBERS` to be much longer than program latency `tPROG` (e.g., 3.5 ms vs. 660 μs [37]).

Read Operation. NAND flash memory determines a cell's data (i.e., the cell's V_{TH} level) by identifying whether current flows through the corresponding BL. Figure 2 depicts the read mechanism of NAND flash memory that consists of three phases: 1) *precharge*, 2) *evaluation*, and 3) *discharge* [68].

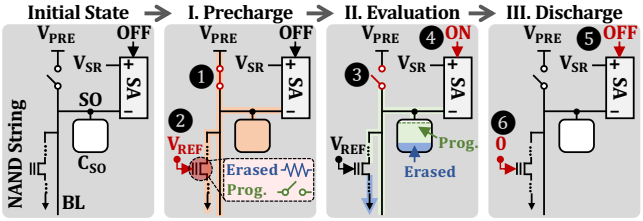


Figure 2: Read mechanism of NAND flash memory.

In the precharge phase (**I** in Figure 2), a NAND flash chip charges each target BL and its sense-out (SO) capacitor C_{SO} to a specific voltage V_{PRE} (❶). The chip also applies the read-reference voltage V_{REF} to the target cell (i.e., WL) at the same time (❷), which enables the BL to sink current through the NAND string depending on the cell's V_{TH} level (i.e., the BL can sink current when $V_{TH} < V_{REF}$).[3] The chip then enters the evaluation phase (**II** in Figure 2) in which it disconnects the BL from V_{PRE} (❸) and enables the sense amplifier (SA) (❹). If the target cell had been programmed (i.e., $V_{TH} > V_{REF}$), the capacitance of C_{SO} hardly changes as the BL cannot sink current. In contrast, if the cell had been erased, charge in C_{SO} quickly flows through the BL, which rapidly decreases the SO-node voltage below the SA's reference voltage V_{SR}. Finally, the chip discharges the BL (**III** in Figure 2) to return to the initial state for future operations (❺ and ❻). As a result, the chip-level read latency `tR` can be expressed as follows:

$$\texttt{tR} = N_{SENSE} \times (\texttt{tPRE} + \texttt{tEVAL} + \texttt{tDISCH}) \quad (1)$$

where N_{SENSE} is the number of sensing times required to read a page, and `tPRE`, `tEVAL`, and `tDISCH` are the *timing parameters* that define the latency for the precharge, evaluation, and discharge phases, respectively. In SLC NAND flash memory, $N_{SENSE} = 1$ because there are only two V_{TH} states, while N_{SENSE} increases up to 3 in TLC NAND flash memory to identify a specific V_{TH} state out of eight ($= 2^3$) different V_{TH} states [6, 7].

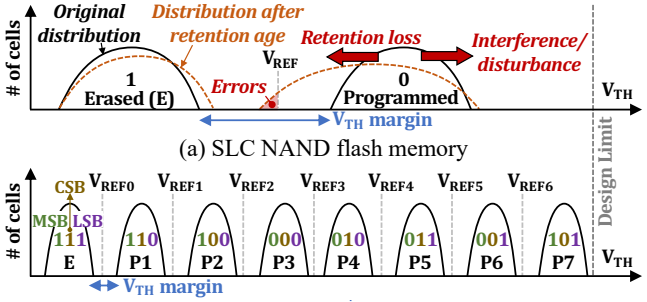
Manufacturers carefully decide the three timing parameters to ensure correct operation. For example, if `tPRE` is too short to fully charge the BL and C_{SO}, V_{SO} can be lower than V_{SR} in the evaluation phase even when the target cell is programmed. A too-short `tDISCH` can also lead to raw bit errors by leaving some BLs partially charged. Since it takes more time to stabilize all BLs when there are some partially-charged BLs compared to when all BLs are fully discharged, the next precharge phase would likely fail to properly set all BLs to V_{PRE} within the `tPRE` latency.

[2] It is also possible to design 3D NAND flash memory with floating-gate cells [92], but most 3D NAND flash chips adopt cylindrical charge-trap cells (e.g., TCAT [33], p-BICs [38], and SMArT [22]) [6, 7, 65, 66, 84].

[3] To ensure that only the target cell's V_{TH} level affects the current through the BL, the gate voltage of all other cells in the same NAND string is set to V_{PASS} (> 6V), which is much higher than the highest V_{TH} level of any flash cell [6, 7, 12]

2.3 Reliability Problems in NAND Flash

In NAND flash memory, a variety of sources including program interference [8, 13, 48, 79], read disturbance [11, 28], and data retention loss [12, 14, 15, 66] introduce bit errors in stored data [6, 7, 9, 10]. Figure 3(a) shows the V_{TH} distribution of a WL in SLC NAND flash memory and how it is affected by various error sources. Reading or programming a flash cell (i.e., WL) slightly increases the V_{TH} level of other *cells* (in other WLs) in the same block by unintentionally injecting electrons to their charge traps (i.e., read disturbance and program interference). A flash cell also leaks electrons in its charge trap over time (i.e., retention loss), which decreases the cell's V_{TH} level. If a cell's V_{TH} level moves beyond the V_{REF} value, a bit error occurs as the cell's data is sensed to be different from the data originally programmed into it. Prior works show that retention loss is the dominant source of errors in 3D NAND flash memory [7, 49, 65, 66, 84]. Compared to 2x-nm planar NAND flash memory, 3D NAND flash memory experiences 40% less program interference and 96.7% weaker read disturbance while it suffers from a larger number of retention errors that occur faster [66].



(a) SLC NAND flash memory

(b) TLC NAND flash memory

Figure 3: V_{TH} distribution of NAND flash memory cells.

A flash cell becomes more susceptible to errors as it experiences more program and erase (P/E) cycles [10, 35, 64]. The high voltage applied to the WL and substrate during program and erase operations damages the flash cell's tunnel oxide, which causes its charge trap to more easily get/leak electrons. After a certain number of P/E cycles, a flash cell is *worn out* (i.e., it cannot be used any longer) as it cannot retain its stored data for a required *retention age*, i.e., how long data is stored after it is programmed (e.g., 1 year [24, 34]).

The multi-level cell (MLC) technique aggravates the reliability problems in NAND flash memory. As shown in Figure 3(b), TLC NAND flash memory stores three bits in a single cell using eight (i.e., 2^3) different V_{TH} states (i.e., levels). To pack more V_{TH} states within the same voltage window, MLC NAND flash memory inevitably *narrows the margin* between adjacent V_{TH} states, which increases the probability that a cell programmed into a particular V_{TH} state is misread as belonging to an adjacent V_{TH} state.

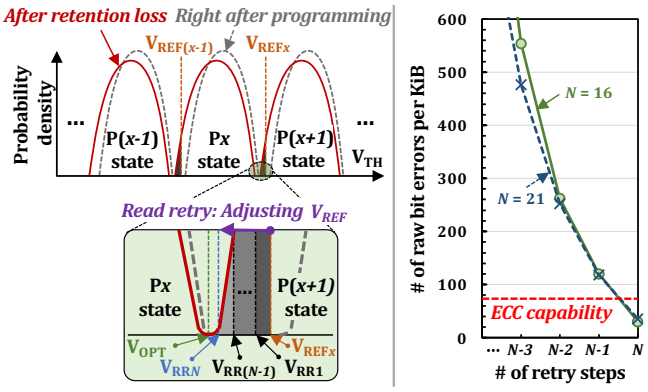
2.4 Reliability Management in NAND Flash

Error-correcting Codes (ECC). To guarantee the reliability of stored data, it is common practice in modern SSDs to employ error-correcting codes (ECC). ECC can detect and correct bit errors within a unit of data, called a codeword, by storing redundant bits (i.e., ECC parity) into the codeword. To address significant reliability degradation in modern NAND flash memory, a modern SSD typically adopts sophisticated ECC, such as Bose-Chauduri-Hocquenghem (BCH) [5] and low-density parity-check (LDPC) [27] codes, which can correct up to several tens of raw bit errors within a codeword (e.g., 72 bit errors per 1-KiB codeword [73]).

Read-Retry Operation. As modern NAND flash memory becomes more susceptible to errors, it is challenging even for strong ECC to guarantee the reliability of stored data: a page's *raw bit-error rate* (RBER, the fraction of error bits in a codeword before ECC) quickly increases beyond the *ECC capability* (i.e., the error-correction capability of ECC, defined as the number of error bits correctable per codeword). This significantly degrades the lifetime of NAND flash memory since a block's lifetime is determined by the number of P/E cycles that can be performed until the block can retain the RBER lower than the ECC capability for a minimum retention requirement [6, 24, 34]. Using more sophisticated ECC (with higher ECC capability) can mitigate the lifetime degradation, but it also introduces significant area and latency overheads [6, 7, 15].

To address this, a modern SSD commonly adopts a mechanism called *read-retry* [9, 10, 26, 54, 84, 93]. Figure 4 shows how read-retry reduces a page's RBER to be lower than the ECC capability. As shown in Figure 4(a), retention loss *shifts* and *widens* the V_{TH} distribution of each state, increasing the number of flash cells whose V_{TH} level moves beyond the corresponding V_{REF} value (e.g., V_{REFx} for the P(x + 1) state). When the number of such cells becomes higher than the ECC capability, a *read failure* occurs, and the SSD controller invokes a read-retry operation for the page. The read-retry operation reads the page *again* with *different* V_{REF} values (e.g., V_{RRi} at the i-th read-retry step in Figure 4(a)), which decreases the number of cells misread as belonging to another V_{TH} state. The controller performs further retry steps until it either successfully reads the page without uncorrectable errors or fails to reduce the page's RBER to a value lower than the ECC capability even after trying all of the V_{REF} values that are available to the mechanism.

How to adjust the V_{REF} values is the most critical design choice in the read-retry mechanism. The V_{TH} distribution of each state is *very narrow* in modern MLC NAND flash memory to store m bits per cell using 2^m V_{TH} states (e.g., 16 V_{TH} states in QLC NAND flash memory). This causes the page's RBER to be extremely sensitive to the distance of the V_{REF} value from the *optimal read-reference voltage* V_{OPT}. In Figure 4(a), for example, we can see that $V_{RR(N-1)}$



(a) V_{REF} adjustment in a read-retry operation

(b) RBER reduction in the last retry steps

Figure 4: RBER reduction via the read-retry mechanism.

leads to a significantly larger number of bit errors (i.e., a wider gray area before $V_{RR(N-1)}$) compared to V_{RRN}, which is closer to V_{OPT}.

Through extensive profiling of NAND flash chips, manufacturers provide sets of V_{REF} values used for a read-retry operation which guarantee the V_{REF} values in the final retry step to be substantially close to V_{OPT}. Figure 4(b) shows how two pages' RBER values change in the *last four* retry steps when reading the two pages requires 16 and 21 retry steps, respectively. We measure the RBER values from real 3D TLC NAND flash chips (see Section 4 for detailed description of our infrastructure and methodology). As shown in Figure 4(b), each page's RBER *drastically decreases in the final (i.e., N-th) retry step* due to the use of *near-optimal* V_{REF} values, enabling the correct reading of the page.

The read-retry mechanism is essential improving reliability and enhancing SSD lifetime, but a read-retry operation can significantly degrade SSD performance due to *multiple* retry steps it causes. In general, the page-read latency tREAD can be formulated as follows:

$$\texttt{tREAD} = \texttt{tR} + \texttt{tDMA} + \texttt{tECC} + \texttt{tRETRY} \quad (2)$$

where tR, tDMA, tECC, and tRETRY are the latencies of sensing the page data (Equation (1)), transferring the sensed data from the chip to the SSD controller, decoding the data with the ECC engine, and performing a read-retry operation, respectively. When a page read requires N_{RR} (≥ 0) retry steps, tRETRY can be expressed as follows [12, 84]:

$$\texttt{tRETRY} = N_{RR} \times (\texttt{tR} + \texttt{tDMA} + \texttt{tECC}). \quad (3)$$

Since a read-retry operation increases tREAD linearly with N_{RR}, it can significantly degrade SSD performance.

3 MOTIVATION

In this section, we 1) present read-retry characteristics of modern NAND flash memory, and 2) introduce two new opportunities for reducing the read-retry latency.

3.1 Read-Retry in Modern NAND Flash

To understand how many read-retry operations occur in modern NAND flash memory and how frequently they occur, we characterize 160 real 3D TLC NAND flash chips under different operating conditions (see Section 4 for detailed description of our infrastructure and methodology). We measure the number of read-retry steps for more than 10^7 pages that are randomly selected from the 160 NAND flash chips, under different operating conditions. Figure 5 shows the probability of occurrence of different numbers of retry steps (in gray scale) for different P/E-cycle counts and *retention ages*. A box at (x, y) represents the probability that a read requires a read-retry operation with y retry steps under x-month retention age. Figure 5 plots the probability under three different P/E-cycle counts, 0 (left), 1K (center), and 2K (right).

We make two observations from the results. First, a page read introduces a significant number of retry steps especially when the page experiences more P/E cycling and/or has a longer retention age. While a *fresh* page (i.e., with no P/E cycling and 0 retention age) can be read without a read-retry, the average number of retry steps significantly increases to 19.9 under a 1-year retention age at 2K P/E cycles, which in turn increases tREAD by 21× on average. Second, a read-retry occurs very frequently even under modest operating conditions, introducing a number of retry steps. Figure 5 shows that 54.4% of reads incur at least seven retry steps under a 6-month retention age even when the pages have *never* experienced P/E cycling (the dot-circle in the left plot). At 1K P/E cycles, at least eight read-retry steps are needed to read a page only after a 3-month retention age (the dot-circle in the center plot of Figure 5). This means that the performance degradation due to read-retry operations can be significant not only under worst-case conditions but also under the common case.

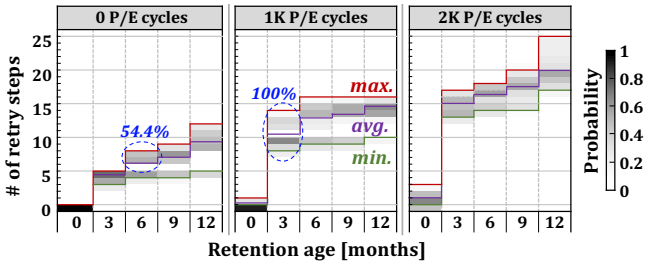


Figure 5: Read-retry characteristics of 160 3D TLC NAND flash memory chips under different operating conditions.

Our characterization results clearly show the importance of mitigating the read-retry overhead. Prior works propose several techniques that reduce the *number* of retry steps [12, 13, 64–66, 77, 84], but read-retry is difficult to *completely avoid* in modern SSDs as V_{OPT} quickly and significantly changes over time. For example, an existing technique can reduce the average number of read-retry steps by about 70% under a 1-year retention age at 2K P/E cycles, but for *every* page read, it requires at least three retry steps [84].

3.2 Optimization Opportunities for Read-Retry

We identify two new opportunities to reduce tRETRY by exploiting two advanced architectural features in modern SSDs: 1) the CACHE READ command and 2) the strong ECC engine.

3.2.1 ***Exploiting the CACHE READ Feature.*** Modern NAND flash memory supports an advanced command called CACHE READ [55, 67, 69, 71, 83, 87] that can effectively reduce tREAD by pipelining consecutive read requests.[4] Early generations of NAND flash memory support the CACHE READ feature only for *sequential reads* (i.e., only when the target page of an incoming read is *physically next* to the currently accessed) [69]. However, to improve the random-read performance, which is critical in popular applications [58, 80], such as key-value stores [4] and graph analytics [95], manufacturers including Samsung, Micron, and Toshiba have extended the CACHE READ command to support *any* consecutive page reads regardless of the locations of the pages to be read [55, 67, 71, 83, 87].

Figure 6 shows how an SSD controller reduces the latency of a page read using the CACHE READ command. As shown in Figure 6(a), with the basic PAGE READ command, an SSD controller can start reading page B *only after* finishing the data transfer of page A. (The data of page A is decoded by the ECC engine dedicated to the channel [6], so the SSD controller can concurrently perform sensing of page B with ECC decoding of page A.) In contrast, as shown in Figure 6(b), the SSD controller can issue a CACHE READ command for page B *before* starting the data transfer of page A so

[4] The CACHE READ command requires an additional *cache (i.e., page buffer)* in the NAND flash chip to store sensed data while transferring the previously-sensed data to the SSD controller.

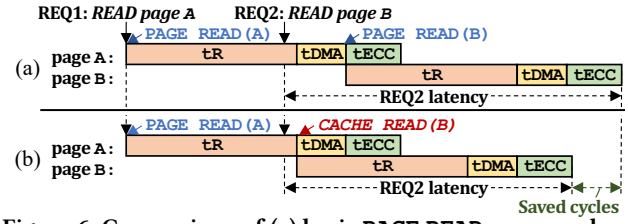


Figure 6: Comparison of (a) basic PAGE READ command and (b) CACHE READ command (see page B in each figure).

that the chip can concurrently perform both the data transfer of page A and sensing of page B. Since each retry step of a read-retry operation is effectively the same as a regular page read, we can also perform consecutive retry steps in a pipelined manner via the CACHE READ command, which in turn reduces the total execution time of a read-retry operation.

3.2.2 ***Exploiting Large ECC-Capability Margin.*** We find that there would be *a large ECC-capability margin*[5] *when a read-retry occurs*. This may sound contradictory as a read-retry occurs only when the page's RBER exceeds the ECC capability, i.e., when there is no ECC-capability margin. However, when a read-retry operation succeeds, the page is eventually read *without* any uncorrectable errors, which means that there exists *a positive* ECC-capability margin in the final retry step if it succeeds. We hypothesize that the ECC-capability margin is large due to two reasons. First, as explained, a modern SSD uses *strong* ECC that can correct several tens of raw bit errors in a codeword. Second, in the final retry step, the page can be read by using *near-optimal* V_{REF} values that drastically decrease the page's RBER as explained in Section 2.4.

If we can empirically demonstrate and methodically leverage the large ECC-capability margin in the final retry step to reduce the *page-sensing latency* tR, doing so allows us to reduce tRETRY considerably. This is because tR is the dominant factor in tRETRY especially when we use the CACHE READ command in a read-retry operation. Although reducing tR may increase the page's RBER as explained in Section 2.2, we can *safely* reduce tR for a read-retry operation as long as the number of additional bit errors introduced by the reduced tR is lower than the large ECC-capability margin in the final retry step. We hypothesize that this is the common case since manufacturers *pessimistically* set the timing parameters to cover for the *worst-case* operating conditions and process variation [19, 45, 52, 53, 74, 75]. For example, an outlier BL can have much higher capacitance than other BLs due to its geometry (e.g., thick wire, narrow contacts, and high parasitic capacitance), which significantly increases the time for the BL to be fully charged. Even if the fraction of such BLs may be very low, eliminating all of them from a chip either requires extreme effort or leads to a significant loss in chip yield. Consequently, such outlier BLs dictate tR, even though most BLs can correctly operate with reduced tR.

4 CHARACTERIZATION METHODOLOGY

To test our hypothesis in Section 3.2.2, we characterize 1) the ECC-capability margin in the final retry step and 2) the reliability impact of tR reduction, using 160 real 3D TLC NAND flash chips.

[5] *ECC-Capability Margin = Maximum Number of Raw Bit Errors a Given ECC can Correct per Codeword − Number of Raw Bit Errors Present in a Codeword.*

Infrastructure. We use an FPGA-based testing platform that contains a custom flash controller and a temperature controller. The flash controller allows us to access a NAND flash chip using all the commands implemented in the chip. It supports not only basic read/program/erase operations, but also dynamic change of timing parameters for a read by using the SET FEATURE command [90]. The temperature controller maintains the temperature of a NAND flash chip within ±1°C of the target temperature. This allows us to test a NAND flash chip under different operating temperatures (i.e., the temperature when a page is read or programmed) and accelerate retention loss based on Arrhenius's Law [2] (e.g., 13 hours at 85°C ≈ 1 year at 30°C). We characterize 160 48-layer 3D TLC NAND flash chips in which ⟨tPRE, tEVAL, tDISCH⟩ = ⟨24 μs, 5 μs, 10 μs⟩ (i.e., tPRE:tEVAL:tDISCH ≈ 5:1:2) by default.

Methodology. To minimize the potential distortions in our characterization results, we randomly select 120 blocks from each of the 160 3D NAND flash chips at different physical block locations and perform read tests for every page in each selected block. We test a total of 3,686,400 WLs (11,059,200 pages) to obtain statistically significant experimental results. Unless specified otherwise, we report a representative (i.e., maximum and/or average) value across all the tested pages from the 160 chips. For a *read test* of a page, we first read the target page with default read-timing parameters and measure the page's RBER. When a read failure occurs, we perform a read-retry operation while measuring the page's RBER in each retry step. Then, using the same V_{REF} values used in the final retry step, we repeat reading of the page and measure its RBER while reducing read-timing parameters, in order to evaluate the impact of reducing the timing parameters on the RBER in the final retry step.

We perform read tests while varying the P/E-cycle count, retention age, and operating temperature, all of which are shown to significantly affect a flash cell's error behavior [6, 7, 9, 10, 12–14, 64–66, 84]. We follow the test procedures of the JEDEC industry standard [34] in each read test. To increase the P/E-cycle count of a block, we repeat the cycle of 1) programming every page in the block with random data[6] and 2) erasing the block. For each target P/E-cycle count, we test each page while varying the retention age and operating temperature by using the temperature controller.

5 CHARACTERIZATION RESULTS

We present and analyze our real-device characterization results on 1) the ECC-capability margin in the final retry step and 2) the reliability impact of reducing read-timing parameters, collected across 160 3D TLC NAND flash chips.

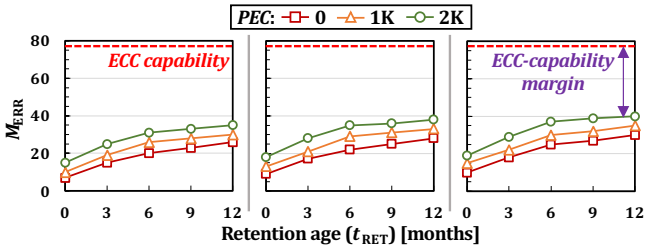
5.1 ECC-Capability Margin in Final Retry Step

Figure 7 depicts $M_{ERR}(PEC, t_{RET})$, i.e., the maximum number of bit errors per 1-KiB data in the final read-retry step under different P/E-cycle counts (PEC) and retention ages (t_{RET}, unit: months)[7], at three different operating temperatures: (a) 85°C, (b) 55°C, and (c) 30°C. We also plot the ECC capability at 72 errors per 1 KiB. We make three key observations. First, there is a large ECC-capability margin

[6] Although a page's RBER has data-pattern dependence [6, 9, 15, 66], we use random data because modern SSDs commonly use a *data randomizer* [6, 17, 43, 57] to avoid the worst-case data patterns that may cause unexpected read failures.

[7] A flash cell's retention loss is significantly affected by ambient temperature. We show the *effective* retention age at 30°C, following an industrial standard that specifies the retention requirements of NAND flash-based SSDs [34].

in the final retry step *even under the worst-case operating conditions* prescribed by manufacturers (e.g., a 1-year retention age [24] at 1.5K P/E cycles [73]). We observe that even M_{ERR}(2K, 12) at 30°C is quite low, leaving a margin as large as 44.4% of the ECC capability. This shows that, although strong ECC is an inevitable choice for a modern SSD, its high ECC capability is largely underutilized when a read-retry eventually succeeds, due to the use of near-optimal V_{REF} values in the final retry step.



(a) 85°C (b) 55°C (c) 30°C

Figure 7: ECC-capability margin in the final read-retry step.

Second, the ECC-capability margin decreases as the page experiences more P/E cycling and longer retention age (e.g., M_{ERR}(0, 3) = 15 while M_{ERR}(1K, 12) = 30 at 85°C). This is due to the inherent error characteristics of NAND flash memory. In fact, unlike what is idealistically shown in Figures 3 and 4,[8] two adjacent V_{TH} states slightly overlap even right after programming a *fresh* page, which makes *no V_{REF} value capable of achieving zero RBER* in modern NAND flash memory [6, 12, 64–66]. As P/E cycling and retention age shift and widen V_{TH} state distributions, even the optimal read-reference voltage cannot completely avoid the RBER increase.

Third, operating temperature also affects the ECC-capability margin in the final retry step, but its impact is *not* as significant as P/E cycling and retention age. Compared to 85°C, M_{ERR} at 30°C and 55°C is higher by 5 and 3 errors, respectively, all other conditions being equal. In 3D NAND flash memory, an electron's mobility in the poly-type channel, which decreases with operating temperature, is the dominant factor affecting the cell current through the BL [3]. Since an erased cell might be recognized as programmed due to reduced current, M_{ERR} slightly increases as operating temperature reduces. We observe the same relationship in all the tested chips (i.e., the lower the temperature, the higher the page's RBER).

We draw two conclusions based on our observations. First, we can use the large ECC-capability margin in the final retry step to reduce `tRETRY`, unless reduction of read-timing parameters significantly increases the page's RBER. Second, the ECC-capability margin highly depends on operating conditions, so we should carefully decide the reduction amount considering the current operating conditions.

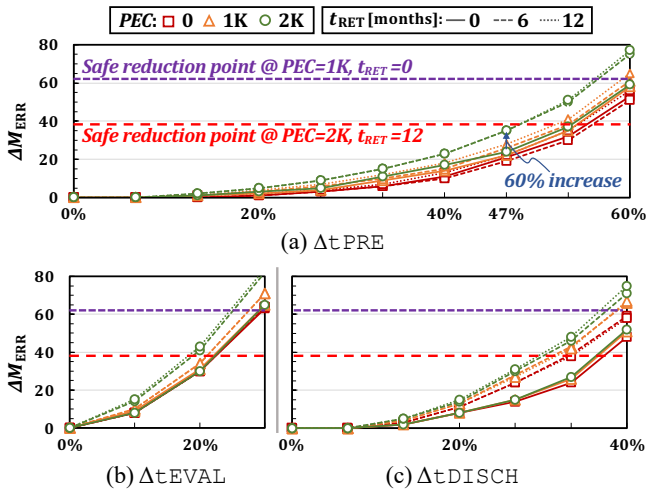
5.2 Reliability Impact of Reducing Read-Timing Parameters

We first present the effect of reducing individual read-timing parameters (Section 5.2.1). We then show the effect of reducing multiple timing parameters simultaneously (Section 5.2.2) and summarize our characterization results with the final timing parameters we decide for reliable `tRETRY` reduction (Section 5.2.3).

5.2.1 ***Reduction of Individual Parameters.*** We first evaluate the effect of reducing individual read-timing parameters under different operating conditions. Figure 8 shows ΔM_{ERR}, the maximum *increase* of raw bit errors per 1-KiB data when we read a page at 85°C[9] with reduced (a) `tPRE`, (b) `tEVAL`, or (c) `tDISCH`, compared to when using the default value.

We make three observations from the results. First, it is possible to safely reduce read-timing parameters for optimizing the read-retry latency. Even under a 1-year retention age at 2K P/E cycles (where M_{ERR} = 35), we can safely reduce `tPRE`, `tEVAL`, and `tDISCH` by 47%, 10%, and 27%, respectively. Second, reduction in `tEVAL` or `tDISCH` leads to faster increase in M_{ERR} compared `tPRE`, which implies that manufacturers set the default `tPRE` *more pessimistically* than the other parameters. As explained in Section 2.2, the precharge phase needs to *stabilize every BL* at a certain voltage level (V_{PRE}), which requires a large timing margin in `tPRE` for outlier BLs. On the other hand, the discharge phase requires a relatively-small timing margin in `tDISCH` compared to `tPRE` because it only pulls out V_{PRE} from BLs. Third, P/E cycling and retention age also affect the increase in bit errors due to reduced read-timing parameters as well as the ECC-capability margin in the final retry step. In particular, we observe non-trivial impact of retention age on ΔM_{ERR}. When reducing `tPRE` by 47%, for example, ΔM_{ERR}(2K, 12) is 60% higher than ΔM_{ERR}(2K, 0) (i.e., a 1-year retention age increases ΔM_{ERR} by 60% at 2K P/E cycles) as shown in Figure 8(a).

We draw two conclusions based on our observations. First, we can significantly reduce `tR` in a read-retry operation *even under the worst-case operating conditions* prescribed by manufacturers. Our results demonstrate that `tPRE` can be safely reduced by *at least* 40% under every tested condition, which leads to a 25% reduction in `tR`. Second, it is *very cost-ineffective* to reduce `tEVAL`. Reducing `tEVAL` by 20% introduces 30 additional bit errors (i.e., 41.7% of the



(a) Δ`tPRE` (b) Δ`tEVAL` (c) Δ`tDISCH`

Figure 8: Effect of reducing each read-timing parameter.

[8] The V_{TH} distribution of NAND flash memory is usually described with simplified figures (similar to Figures 3 and 4) to ease understanding [35, 65, 66, 79, 84].

[9] We show the effect of operating temperature in Section 5.2.3.

ECC capability) *even for a fresh page*. This significantly decreases the chance to reduce the other parameters while achieving only 2.5% tR reduction due to the low contribution of tEVAL to tR (1/8 tR only). Therefore, we decide to exclude tEVAL from our later analyses.

5.2.2 ***Reduction of Multiple Parameters.*** Although our results of the previous experiments promise a great opportunity for reducing each of tPRE and tDISCH alone, reducing one may decrease the chance of reducing the other (because the discharge phase of a read affects the precharge phase of the next read as explained in Section 2.2). To identify the potential for reducing both timing parameters simultaneously, we test all possible combinations of (tPRE, tDISCH) values while reducing tPRE by up to 60% and tDISCH by up to 40%. Figure 9 plots $M_{ERR}(PEC, t_{RET})$, the maximum number of bit errors per 1-KiB data in the final retry step when we read test pages while reducing tPRE and tDISCH simultaneously under five different operating conditions.

We make three key observations based on the results. First, reducing the two timing parameters simultaneously introduces more additional bit errors than reducing each parameter individually. For example, as shown in Figures 8(a) and 8(c), when we reduce tPRE by 54% and tDISCH by 20% individually, ΔM_{ERR}(1K, 0) is 35 and 8, respectively. Unfortunately, simultaneous reduction of the two timing parameters increases M_{ERR} far beyond the ECC capability; see the green line with marker × in Figure 9(a) (i.e., ΔtDISCH =20%), which is outside the plot at ΔtPRE = 54%. Second, it is more beneficial to reduce tPRE than to reduce tDISCH in most cases. M_{ERR} is smaller when ⟨ΔtPRE, ΔtDISCH⟩ = ⟨x%, y%⟩ compared to when ⟨ΔtPRE, ΔtDISCH⟩ = ⟨y%, x%⟩ for most values of x and y. Third, despite the higher reliability impact of tDISCH over tPRE (discussed in Section 5.2.1), reducing tDISCH by 7% hardly increases the number of bit errors (by 4 at most) under every operating condition.

Based on our observations, we conclude that it is effective to use the ECC-capability margin in the final retry step for *only reducing tPRE*. Although the increase in additional bit errors from reducing tDISCH by 7% is quite low (up to 4 additional bit errors), the cost

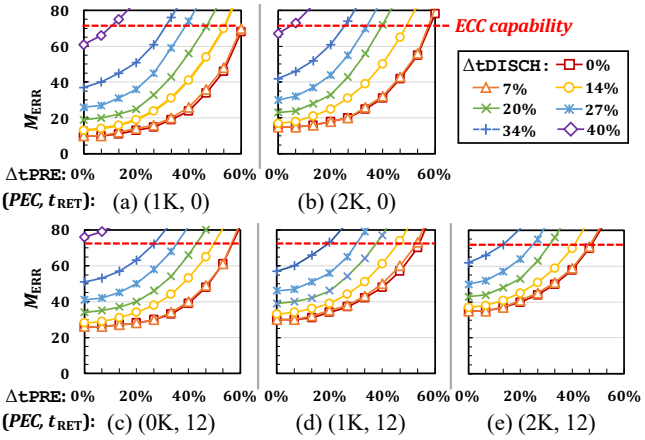


Figure 9: Effect of reducing multiple read-timing parameters, tPRE and tDISCH, under different P/E-cycle counts (*PEC*) and retention ages (t_{RET}, unit: months).

of doing so is larger than the benefit: considering that the fraction of tDISCH in tR is only 25%, a 7% reduction in tDISCH merely reduces tR by 1.75% (i.e., 0.07 × 0.25), while its cost could be up to 5.6% of the ECC capability (i.e., up to 4 additional bit errors under the ECC capability of 72 errors per 1 KiB).

5.2.3 ***Reliable Reduction of tPRE.*** As the final step of our characterization, we analyze the impact of operating temperature on the amount of tPRE reduction. Figure 10 plots ΔM_{ERR}, the increase in the maximum number of raw bit errors in the final retry step when a NAND flash chip operates at 30°C and 55°C, compared to at 85°C. We observe that operating temperature affects ΔM_{ERR} in a similar way as it affects M_{ERR}: the lower the operating temperature, the larger the ΔM_{ERR}, and the temperature effect becomes more significant under a longer retention age and higher P/E-cycle count. The increase in ΔM_{ERR} is also small: it is only up to 7 additional bit errors even under a 1-year retention age at 2K P/E cycles.

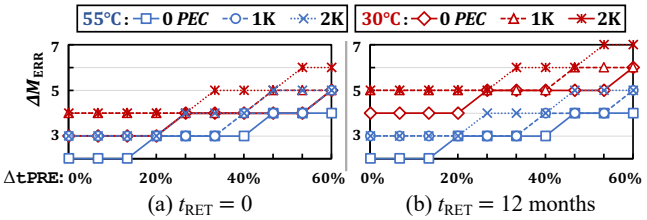


Figure 10: Effect of operating temperature on the number of additional errors due to tPRE reduction.

Based on these results, we conclude that we should incorporate a safety margin into reduced tPRE to ensure that a page's RBER is lower than the ECC capability in the final retry step under varying operating temperature. It is also possible to profile the *optimal* tPRE for each combination of (*PEC*, t_{RET}, T) where T is the operating temperature. However, we decide to determine a *good* tPRE value by considering only *PEC* and t_{RET}, and plan for sufficient ECC capability that can correct temperature-induced additional errors. This is due to two reasons. First, the effect of operating temperature on M_{ERR} is quite small compared to the effect of *PEC* and t_{RET}, and thus operating temperature does not significantly affect the reduction in tPRE. When we reduce tPRE alone by less than 40%, a substantial ECC-capability margin remains to correct temperature-induced additional errors under every operating condition. Second, our decision greatly reduces profiling effort and eliminates the need to monitor a wide range of temperatures. In particular, operating temperature may be difficult or costly to accurately measure for each retry step as it changes much more quickly than *PEC* and t_{RET}.

Figure 11 shows the values we select for safely reducing the read-retry latency under different operating conditions. To minimize the probability of increasing the number of retry steps for outlier pages (which could potentially be missed in the set of pages we test experimentally), we ensure that the selected tPRE value for each operating condition guarantees an ECC-capability margin of 14 bits in the final retry step (7 bits for temperature-induced errors and 7 bits for errors in outlier pages). We conclude that, even with the 14-bit margin, we can significantly reduce tPRE by at least 40% (up to 54%) under any operating condition, as shown in Figure 11.

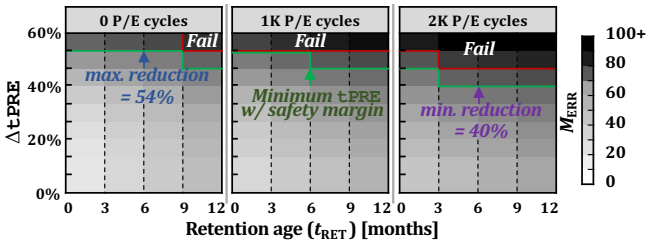


Figure 11: Minimum tPRE for safe tRETRY reduction.

6 READ-RETRY OPTIMIZATIONS

Motivated by our new experimental findings from real 3D NAND flash chips, we propose two new techniques that effectively reduce the read-retry latency: 1) Pipelined Read-Retry (PR^2) and 2) Adaptive Read-Retry (AR^2).

6.1 PR^2: Pipelined Read-Retry

PR^2 reduces the total execution time of a read-retry operation by pipelining consecutive retry steps using the CACHE READ command. Figure 12 compares PR^2 with a regular read-retry operation. As shown in Figure 12(a), the existing read-retry mechanism [6, 7, 9, 10, 26, 54, 84, 93] starts a new retry step *after* checking whether ECC decoding for the previous step succeeds, which places tECC on the critical path of tRETRY (Equation (3)).[10] the previous retry step is completed because *speculatively* starting a new retry step could delay other user requests waiting for the completion of the on-going read (and retries), if the speculative retry step ends up being not needed. In contrast, as shown in Figure 12(b), PR^2 starts the next retry step right after the chip completes page sensing of the current step (i.e., after tR of the current step) by issuing a CACHE READ command on the same target page. Since the SSD controller can return the read page once ECC decoding succeeds, when the read request requires N_{RR} retry steps, tRETRY in PR^2 can be formulated as follows:

$$\texttt{tRETRY} = N_{RR} \times \texttt{tR} + \texttt{tDMA} + \texttt{tECC}. \quad (4)$$

Thus, PR^2 reduces tRETRY by $(N_{RR}-1)\times(\texttt{tDMA}+\texttt{tECC})$ over the regular read-retry mechanism (Equation (3)). Considering that many reads require multiple retry steps even under modest operating conditions as we observe in Section 3.1 (e.g., *every* read requires more than *eight* retry steps under a 3-month retention age at 1K P/E cycles as shown in Figure 5), PR^2 significantly improves SSD performance by effectively reducing tRETRY.

When a read requires N_{RR} retry steps, PR^2 speculatively starts the $(N_{RR}+1)$-th retry step that is unnecessary to read the page. This unnecessarily-started retry step could negatively affect SSD performance by delaying other operations that may exist in the request queue, waiting for the completion of the read. PR^2 minimizes this potential performance penalty by using the RESET command that immediately terminates any on-going request that is being performed in the chip. As described in Figure 12(b), the SSD controller issues a RESET command as soon as ECC decoding succeeds, which takes only a few microseconds to terminate the unnecessarily-started retry step (the reset latency tRST = 5 μs for a read operation [70]).

[10]The SSD controller can issue a new PAGE READ command *before starting* ECC decoding of the currently read-out page (as described in Figure 6(a)). However, prior works [9, 10, 26, 54, 84, 93] assume that a new retry step starts *after*

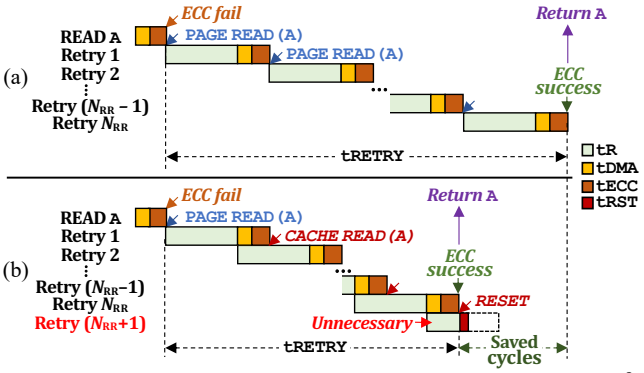


Figure 12: Comparison of (a) regular read-retry and (b) PR^2.

Overhead. PR^2 requires no change to NAND flash chips. It requires only slight modifications to the SSD controller or firmware to issue 1) a CACHE READ command for each retry step immediately after page sensing of the previous step and 2) a RESET command as soon as ECC decoding succeeds. The performance overhead of PR^2 is also small. The unnecessarily-started retry step could delay other operations (that are waiting for the completion of the read in the request queue) only for several microseconds at most. When the read requires more than one retry step, which is the common case (see Section 3.1), the latency benefit of PR^2 is always higher than its latency overhead.

6.2 AR^2: Adaptive Read-Retry

AR^2 optimizes the read-retry latency by using the large ECC-capability margin in the final retry step to reduce tPRE (and thus tR) for *every* retry step (*not* only for the final step). AR^2 carefully decides the tPRE reduction amount depending on the current operating conditions to avoid additional retry steps introduced by tPRE reduction. To this end, we propose that SSD manufacturers 1) identify the best tPRE values at different operating conditions for each NAND flash chip via offline profiling of the chip and 2) incorporate the information into the SSD in the form of a simple table, i.e., *Read-timing Parameter Table (RPT)*. The RPT stores the best profiled tPRE value for a given P/E-cycle count and retention age. At runtime, the SSD controller queries the RPT.

Figure 13 illustrates how the SSD controller can reduce the read-retry latency using AR^2.[11] Once a read failure occurs, ❶ AR^2 first decides the appropriate tPRE reduction amount by querying the RPT using the P/E-cycle count and retention age of the corresponding block.[12] ❷ AR^2 then changes the target chip's tPRE value by issuing a SET FEATURE command, and ❸ repeats performing retry steps until the page is successfully read without any uncorrectable error or until all possible retry steps are exhausted. Finally, ❹ AR^2 rolls back the target chip's tPRE value to the default tPRE value for future operations.

As explained above, in AR^2, the SSD controller changes the tPRE value *only once* (❷) for a read-retry operation, i.e., it performs *all* the retry steps using the same tPRE value. Doing so *does not* increase

[11]We assume that PR^2 is already implemented.

[12]A regular SSD already keeps track of the P/E-cycle count and retention age of each block (i.e., the information necessary to determine the best tPRE value) in order to ensure SSD lifetime and reliability (e.g., for wear leveling [21], periodic refresh of stored data [14, 15, 28], or optimal V_{REF} prediction [12, 64–66]).

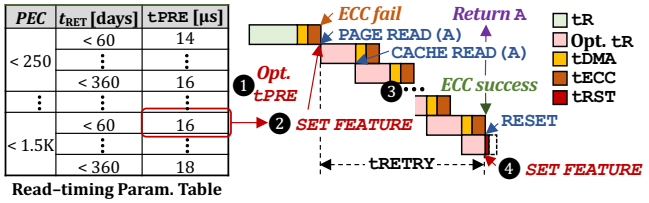


Figure 13: Read-retry latency reduction in AR^2.

the number of performed retry steps as long as the best tPRE value for a given (*PEC*, t_{RET}) is correctly found via offline profiling. As we observe in Section 5.1, the target page's RBER drastically decreases in the final retry step due to the use of near-optimal V_{REF} values, while ECC decoding would fail anyway in all previous steps *even if the default tPRE value were used*. In other words, with accurate profiling, the tPRE reduction does not affect the failure of the previous steps, while guaranteeing the success of the final retry step.

With AR^2, tRETRY can be expressed as follows:

$$tRETRY = tSET + \rho \times N_{RR} \times tR + tDMA + tECC \quad (5)$$

where tSET is the latency of the SET FEATURE command for adjusting tPRE, and ρ is reduction ratio in tR ($0 < \rho \leq 1$). As tR is the dominant factor in tRETRY, AR^2 can provide considerable performance improvement. For example, a 25% tR reduction (=22.5 μs) under a 1-year retention age at 2K P/E cycles is easily possible (Section 5.2.3). Note that tSET is almost negligible (e.g., < 1 μs [72]) compared to the total amount of tR reduction.

Overhead. AR^2 requires only small changes to the SSD controller to adjust tPRE based on the RPT. The storage overhead of the RPT to store the best tPRE values for tens of (*PEC*, t_{RET}) combinations is also very small. For example, with 36 (*PEC*, t_{RET}) combinations, we estimate the table size to be only 144 bytes per chip. Like other metadata related to each NAND flash chip (e.g., the P/E-cycle count of each block in the chip), the RPT can be stored in a specific page of each NAND flash chip and fetched into internal SRAM or DRAM at boot time so that the SSD controller can quickly access the RPT once a read-retry occurs.

AR^2 might potentially increase the number of retry steps for outlier pages that exhibit high RBER in the final retry step even with the default tPRE value. In the worst case, i.e., when a read-retry fails[13] with reduced tPRE, AR^2 needs to perform a read-retry operation on the same page using the default tPRE value since the previous read-retry operation might not have failed with the default tPRE value. However, the extremely-low probability of such a case (we never detect such a case in our characterization of more than 10^7 pages) makes the potential performance overhead of AR^2 negligible. Note that we also incorporate a safety margin for outlier pages into the reduced tPRE values, as shown in Figure 11.

7 SYSTEM-LEVEL EVALUATION

We evaluate the impact of PR^2 and AR^2 on system performance using a state-of-the-art SSD simulator [1, 85] and twelve storage I/O traces from two representative benchmark suites.

[13] As explained in Section 2.4, a read-retry operation fails if the page's RBER is higher than the ECC capability even after trying all available sets of V_{REF} values prescribed by manufacturers.

7.1 Methodology

We evaluate the effectiveness of PR^2 and AR^2 using MQSim [1, 85], an open-source multi-queue SSD simulator. We faithfully extend MQSim based on our real-device characterization results to simulate more realistic read-retry characteristics of modern SSDs. We modify the NAND flash model of MQSim such that each simulated block operates exactly the same as one of the real blocks that we test in Section 5. We randomly select real tested blocks and map each of them to a simulated block. We modify the data structure of each simulated block to contain a lookup table for the number of read-retry steps at a certain P/E-cycle count and retention age, which we profile from the corresponding real block. As MQSim maintains the P/E-cycle count and programming time of each page, a simulated block can accurately emulate the same read-retry behavior as the corresponding real block for every read. We simulate a 512-GiB SSD that contains 4 channels, 4 dies per channel, and 2 planes per die. A plane consists of 1,888 blocks, each of which has 576 16-KiB pages. We assume an ECC engine that corrects up to 72 bit errors per 1-KiB codeword within tECC =20 μs. Table 1 summarizes the timing parameters of our simulated NAND flash chip, which we obtain from the real NAND flash chips used in our characterization. The I/O rate is set to 1Gb/s, i.e., tDMA = 16 μs for a 16-KiB page.

Table 1: NAND flash timing parameters.

Parameter	Time	Parameter	Time
tR (avg.)[14]	90 μs	**tPROG**	700 μs
tPRE	24 μs	**tBERS**	5 ms
tEVAL	5 μs	**tSET**	1 μs
tDISCH	10 μs	**tRST**	5 μs for read

We evaluate twelve workloads from two benchmark suites, Microsoft Research Cambridge (MSRC) Traces [76] and Yahoo! Cloud Service Benchmark (YCSB) [23], which are widely used for performance evaluation of storage systems [15, 16, 49, 58, 63, 65, 66, 84–86]. MSRC Traces consist of 36 block I/O traces that are collected from enterprise servers during one week. We select six traces from the 36 total traces to have different I/O characteristics in terms of *read ratio* and *cold ratio*, as summarized in Table 2. Read ratio is the fraction of read requests in a workload. Cold ratio is the fraction of read requests whose target page is never updated during the entire execution of the workload. Such a page is programmed only once and thus experiences a long retention age compared to frequently-updated pages (i.e., write-hot pages).

Table 2: I/O characteristics of the evaluated workloads.

Workload	Read ratio	Cold ratio	Workload	Read ratio	Cold ratio
stg_0	0.15	0.38	**YCSB-A**	0.98	0.72
hm_0	0.36	0.22	**YCSB-B**	0.99	0.59
prn_1	0.75	0.72	**YCSB-C**	0.99	0.6
proj_1	0.89	0.96	**YCSB-D**	0.98	0.58
mds_1	0.92	0.98	**YCSB-E**	0.99	0.98
usr_1	0.96	0.73	**YCSB-F**	0.98	0.87

[14] As explained in Section 2.2, in multi-level cell NAND flash memory, tR varies depending on N_{SENSE}, the number of sensing times required for reading a page (Equation (1)). In TLC NAND flash memory, $N_{SENSE} = \langle 2, 3, 2 \rangle$ for ⟨LSB (least-significant bit), CSB (central-significant bit), MSB (most-significant bit)⟩ pages [6, 7].

7.2 SSD Response Time

We compare five SSD configurations with different read-retry mitigation schemes: 1) Baseline, 2) PR^2, 3) AR^2, 4) PnAR^2, and 5) NoRR. Baseline is a high-end SSD that 1) adopts out-of-order I/O scheduling [36, 86] and program/erase suspension [50, 91] techniques to provide high read performance, and 2) performs regular read-retry operations (described in Figure 12(a)). PR^2 and AR^2 are SSDs that implement each of our two techniques alone on top of Baseline, respectively. PnAR^2 (*Pipelined and Adaptive Read-Retry*) is an SSD that integrates both PR^2 and AR^2 to minimize the read-retry latency. NoRR is an *ideal* SSD where no read-retry occurs, showing the upper bound of eliminating read-retry on SSD performance. Figure 14 compares the performance (average response time) of all five SSD configurations, normalized to Baseline, under different operating conditions.

We make five key observations from Figure 14. First, our two techniques, either alone or when combined, significantly improve the I/O performance of a modern SSD. PR^2 and AR^2 reduce SSD response time by up to 38.3% and 18.1% (17.7% and 11.9% on average across all workloads) compared to Baseline, respectively. PnAR^2 provides higher improvements by using both PR^2 and AR^2, enabling up to 51.8% (28.9% on average) SSD response time reduction over Baseline. Second, PR^2 and AR^2 improve SSD performance in a *synergistic manner*. The average SSD response time improvement of PnAR^2 (28.9% on average across all workloads) is higher than simple aggregation of the individual gains of PR^2 and AR^2. Third, the worse the operating conditions, the larger the performance gain of the proposed techniques. For example, under a 6-month retention age at 2K P/E cycles, PnAR^2 reduces the average SSD response time across all workloads by 35.2% over Baseline. Fourth, our proposal is highly effective under read-dominant workloads, but it also provides considerable performance improvement for workloads with many writes. For stg_0, whose read ratio is only 0.15, PnAR^2 improves SSD performance by 34% under a 6-month retention age at 2K P/E cycles and by 18.7% on average under all operating conditions. The results suggest that, although the page-read latency (tR) is much shorter than the page-program latency (tPROG) and block-erasure latency (tBERS), it is important to optimize read-retry as frequent read-retry operations with multiple retry steps can cause read requests to bottleneck SSD performance. Fifth, PR^2 and AR^2 are effective at mitigating the performance overhead of read-retry, but there still exists large room for improvement. PnAR^2 reduces the response-time gap between Baseline and NoRR by 41% on average across all workloads, but still exhibits a 2.37× higher average response time compared to the ideal NoRR.

Based on our observations, we conclude that our proposal is effective at improving SSD performance by mitigating the significant overheads of read-retry. We also find that there is still large room for improvement to optimize PnAR^2, which motivates us to combine PR^2 and AR^2 with existing read-retry optimization techniques that aim to reduce the *number* of read-retry operations.

7.3 Comparison to Prior Work

To evaluate the effectiveness of our proposal when combined with existing read-retry optimization techniques, we compare two SSD configurations that adopt a state-of-the-art read-retry mitigation technique [84], called PSO (Process Similarity-aware Optimization). PSO reduces the number of retry steps by reusing V_{REF} values that are recently used for a read-retry operation on other pages exhibiting similar error characteristics with the target page to read. We simulate two PSO-enabled SSDs using MQSim: 1) PSO, which adopts only the PSO technique over Baseline, and 2) PSO+PnAR2, which integrates PR^2 and AR^2 on top of PSO. Figure 15 compares the average response time of the two SSDs under different conditions. All values are normalized to Baseline.

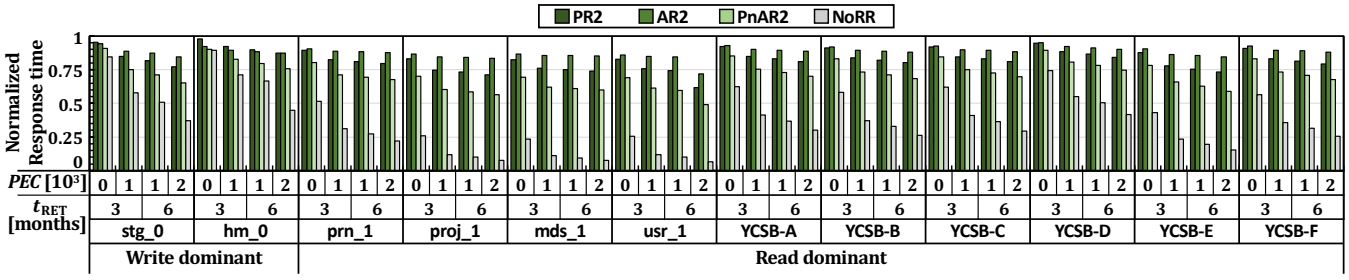


Figure 14: Response-time (RT) reduction of our proposal under different *PEC* (unit: $\times 10^3$) and t_{RET} (unit: months).

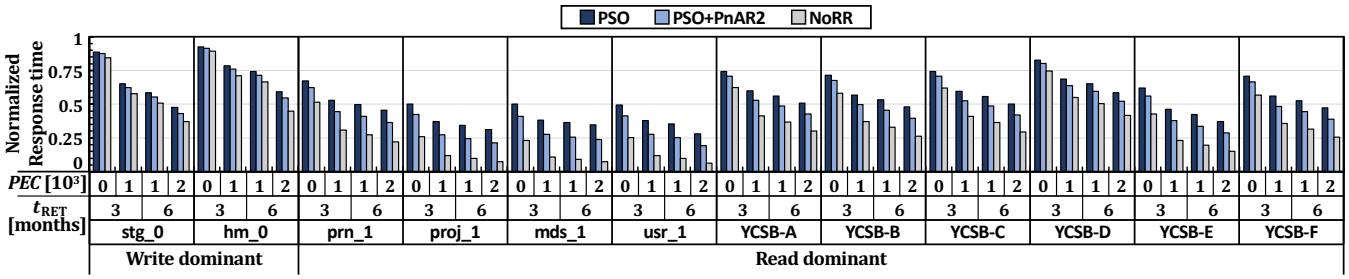


Figure 15: Performance improvement of our proposal when combined with an existing read-retry mitigation scheme [84].

We make three major observations. First, although PSO significantly reduces the average response time over Baseline, its performance is still far from the ideal NoRR. In particular, the average response time of PSO is up to 4.31× (1.92× on average) that of NoRR in read-dominant workloads. Second, PR^2 and AR^2 further improve SSD performance significantly when implemented on top of the PSO technique. Compared to PSO, PSO+PnAR2 reduces the average response time by up to 31.5% (17% on average) in read-dominant workloads and by up to 9.4% (3.6% on average) in write-dominant workloads. Third, PSO+PnAR2's average response time is 1.6× that of the ideal N_{RR} in read-dominant workloads. This shows that there is still some more room for optimizing read-retry in future work.

We conclude that our proposal effectively complements existing techniques to minimize the read-retry overhead and thus significantly improves the performance of modern SSDs. We believe that PR^2 and AR^2 are quite promising as their large performance benefits come with almost negligible overheads.

8 DISCUSSION

We briefly discuss the potential impact of our proposals on future research to optimize SSD read performance.

Latency Reduction for Regular Reads. A modern SSD's high ECC capability enables AR^2 to use a page's ECC-capability margin for reducing the sensing latency. Although AR^2 is used for only reducing the latency of a read-retry, we expect that its key idea can be used also for reducing the latency of a regular page read (i.e., a page read requiring no read-retry). For example, if we can accurately estimate a page's RBER and near-optimal V_{REF} values using an accurate error model (which could be obtained via extensive real-device characterization as done in [6, 7, 9–16, 64–66, 84]), it would enable us to safely reduce read-timing parameters for regular reads.

Further Reduction of Read-Retry Latency. We expect that the key idea of PR^2, i.e., speculatively starting a new retry step (assuming that the current retry step is likely to fail), can be extended to further reduce the read-retry latency. For example, if a page to read is likely to exhibit high RBER that would exceed the ECC capability, the SSD controller can speculatively start read-retry steps without reading the page using default timing parameters (which would likely fail). By using an accurate error model that can predict whether or not a page read would fail (e.g., as in [6, 7, 9, 10, 12–14, 64–66, 84]), such an approach could further reduce the effective read-retry latency without penalty.

9 RELATED WORK

To our knowledge, this paper is the first to 1) provide a detailed and rigorous understanding of the read-retry behavior and the effect of reducing read-timing parameters in modern NAND flash memory by characterizing a large number of real 3D NAND flash chips, and 2) propose two new techniques that effectively reduce the read-retry latency by exploiting advanced features of modern SSDs. We briefly discuss closely related prior work that aims to mitigate the read-retry overhead and improve system performance by exploiting the reliability margin in memory devices, specifically NAND flash memory and DRAM.

Read-Retry Mitigation. Prior works [14, 15, 28] propose to refresh a page (i.e., reset the page's retention age) before the page's RBER increases beyond the ECC capability to avoid a read failure that, in turn, causes a read-retry. Refresh-based approaches might be able to reduce the number of read-retry operations, but can negatively affect the overall SSD performance by wasting bandwidth and/or increasing wear due to refresh operations. As explained in Section 2.3, a page's RBER rapidly increases beyond the ECC capability in modern 3D NAND flash memory, which incurs a read-retry operation even under a zero retention age as shown in Figure 5. To avoid read-retry, therefore, refresh-based approaches should refresh (i.e., read and rewrite) soon-to-be-read pages *very frequently*, which could introduce significant performance overheads and wear, as shown in [14, 15].

Several prior works [12, 13, 64–66, 77, 84] propose to keep track of pre-optimized V_{REF} values for each block to use the V_{REF} values for future read requests. By starting a read (and retry) operation with pre-optimized V_{REF} values close to the optimal read-reference voltage V_{OPT}, they significantly reduce the number of read-retry steps. Unfortunately, read-retry is a fundamental problem that is hard to completely eliminate in modern SSDs. While the existing techniques reduce the *number* of read-retry steps, our techniques effectively reduce *the latency for performing the same number of retry steps*, which makes our techniques complementary to these existing techniques, as shown in Section 7.3. Considering the low overhead of our techniques, they can be easily combined with existing techniques to minimize the read-retry overhead that is expected to become larger in emerging NAND flash memory (e.g., 3D QLC NAND flash memory).

A concurrent study [56] with ours proposes a new V_{OPT} prediction technique to reduce the number of retry steps. The key idea is to store predefined bit patterns in a set of spare cells in each page, called *Sentinel* cells, so as to accurately estimate the page's current error characteristics (and V_{OPT}) based on errors incurred in the predefined bit patterns. Doing so allows the SSD controller to try near-optimal V_{REF} values right after the first regular page read, which significantly reduces the average number of read-retry steps (from 6.6 to 1.2). Both of our proposed techniques, PR^2 and AR^2, can complement the Sentinel-based approach [56] as well as other read-retry mitigation techniques [12, 13, 64–66, 77, 84] that aim to reduce the *number* of retry steps. First, once the SSD firmware accurately identifies the optimal V_{REF} values using the Sentinel cells, it is possible to reduce the latency of the following retry step(s) using AR^2, i.e., applying reduced read-timing parameters. Second, PR^2 reduces the latency of a read-retry operation when the Sentinel-based V_{OPT} prediction fails (as shown in [56], the Sentinel-based approach cannot completely avoid multiple retry steps), by speculatively issuing a new retry step.

Full Utilization of Reliability Margin in Memory Devices. There is a large body of prior work to improve DRAM performance and energy consumption by leveraging variation in access latency [18–20, 29, 30, 45–47, 51–53, 62, 89, 94] and data retention times [25, 39–42, 59, 60, 81, 82, 88] across DRAM cells. These works clearly show that manufacturers set timing parameters of a device conservatively to ensure correct operation of outlier cells, even though a significant majority of cells can reliably operate with lower timing parameters. Our work shows that there also exists a large reliability margin in modern NAND flash memory and demonstrates that the large margin can be used for reducing the read latency through careful real-device characterization.

A few prior works [61, 78, 84] propose to improve the write performance of SSDs by fully exploiting the underutilized ECC capability. For example, Liu et al. [61] propose to program a soon-to-be-updated page with coarse-grained voltage control, which greatly reduces the program latency. As the page will not experience a long retention age, programming the page with fine-grained voltage control unnecessarily increases the program latency while leaving a large ECC-capability margin for the page. Similarly, Shim et al. [84] propose to reduce the number of verify steps in a page-program operation, if the target WL is more robust to error than other WLs. By leveraging processing variation in modern NAND flash memory, they safely reduce the program latency for a large number of WLs. None of these techniques, however, leverage the ECC-capability margin to improve SSD read latency, which is often more critical for many key applications in modern computing systems.

10 CONCLUSION

This paper proposes two new read-retry mechanisms, PR^2 and AR^2, which significantly improve SSD performance by reducing the latency of read-retry operations. We identify new opportunities to optimize the read-retry latency by exploiting advanced architectural features widely adopted in modern SSDs. Through extensive real-device characterization of modern 3D TLC NAND flash chips, we demonstrate that it is possible to use the large ECC-capability margin present in the final retry step for reducing the latency of each retry step. Our results show that our techniques effectively improve SSD performance and complement existing techniques. We hope that our new findings on the read-retry behavior and the reliability impact of reducing read-timing parameters in modern NAND flash-based SSDs inspire new mechanisms to further improve SSD latency and performance, which are critical for modern data-intensive workloads.

ACKNOWLEDGMENTS

We would like to thank our shepherd Dan Tsafrir and anonymous reviewers for their feedback and comments. We thank the SAFARI Research Group members for feedback and the stimulating intellectual environment they provide. We thank our industrial partners, especially Google, Huawei, Intel, Microsoft, and VMware, for their generous donations. This work was in part supported by Samsung Research Funding & Incubation Center of Samsung Electronics, Republic of Korea under Project Number SRFC-IT2002-06, and by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science and ICT (MSIT) (NRF-2020R1A6A3A03040573). The ICT at Seoul National University provided research facilities for this study. (*Co-corresponding Authors: Jisung Park, Jihong Kim, and Onur Mutlu*)

REFERENCES

[1] MQSim GitHub Repository. https://github.com/CMU-SAFARI/MQSim. 2018.
[2] Svante Arrhenius. Über die Dissociationswärme und den Einfluss der Temperatur auf den Dissociationsgrad der Elektrolyte. *Z. Phys. Chem.* 1889.
[3] Pranav Arya. A Survey of 3D NAND Flash Memory. *EECS International Graduate Program, National Chiao Tung University.* 2012.
[4] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload Analysis of a Large-scale Key-value Store. In *SIGMETRICS.* 2012.
[5] Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a Class of Error Correcting Binary Group Codes. *Inf. Ctrl.* 1960.
[6] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Error Characterization, Mitigation, and Recovery in Flash-memory-based Solid-state Drives. *Proc. IEEE.* 2017.
[7] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. Reliability Issues in Flash-memory-based Solid-state Drives: Experimental Analysis, Mitigation, Recovery. In *Inside Solid State Drives* (2nd ed.). Springer. 2018.
[8] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques. In *HPCA.* 2017.
[9] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis. In *DATE.* 2012.
[10] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling. In *DATE.* 2013.
[11] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery. In *DSN.* 2015.
[12] Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery. In *HPCA.* 2015.
[13] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation. In *ICCD.* 2013.
[14] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Crista, Osman S. Unsal, et al. Error Analysis and Retention-aware Management for NAND Flash Memory. *Intel Tech. J.* 2013.
[15] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman S. Unsal, et al. Flash Correct-and-refresh: Retention-aware Error Management for Increased Flash Memory Lifetime. In *ICCD.* 2012.
[16] Yu Cai, Gulay Yalcin, Onur Mutlu, F. Erich Haratsch, Osman Unsal, Adrian Cristal, et al. Neighbor-cell Assisted Error Correction for MLC NAND Flash Memories. In *SIGMETRICS.* 2014.
[17] Jaewon Cha and Sungho Kang. Data Randomization Scheme for Endurance Enhancement and Interference Mitigation of Multilevel Flash Memory Devices. *ETRI Journal.* 2013.
[18] Karthik Chandrasekar, Sven Goossens, Christian Weis, Martijn Koedam, Benny Akesson, Norbert Wehn, et al. Exploiting Expendable Process-margins in DRAMs for Run-time Performance Optimization. In *DATE.* 2014.
[19] Kevin K. Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, et al. Understanding Latency Variation in Modern DRAM Chips: Experimental characterization, Analysis, and Optimization. In *SIGMETRICS.* 2016.
[20] Kevin K. Chang, A. Giray Yaglikci, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Adhijith Kashyap, et al. Understanding Reduced-voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms. In *SIGMETRICS.* 2017.
[21] Li-Pin Chang. On Efficient Wear Leveling for Large-scale Flash-memory Storage Systems. In *SAC.* 2007.
[22] Eun-Seok Choi and Sung-Kye Park. Device Considerations for High Density and Highly Reliable 3D NAND Flash Cell in Near Future. In *IEDM.* 2012.
[23] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *SoCC.* 2010.
[24] Alvin Cox. JEDEC SSD Endurance Workloads. In *FMS.* 2011.
[25] Anup Das, Hasan Hassan, and Onur Mutlu. VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency. In *DAC.* 2018.
[26] Aya Fukami, Saugata Ghose, Yixin Luo, Yu Cai, and Onur Mutlu. Improving the Reliability of Chip-off Forensic Analysis of NAND Flash Memory Devices. *Digital Investigation.* 2017.
[27] Robert Gallager. Low-density Parity-check Codes. *IEEE TIT.* 1962.
[28] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. An Integrated Approach for Managing Read Disturbs in High-density NAND Flash Memory. *IEEE TCAD.* 2015.
[29] Hasan Hassan, Gennady Pekhimenko, Nandita Vijaykumar, Seshadri Vivek, Donghyuk Lee, Oguz Ergin, et al. ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality. In *HPCA.* 2016.
[30] Hasan Hassan, Nandita Vijaykumar, Samira Khan, Saugata Ghose, Kevin K. Chang, Gennady Pekhimenko, et al. SoftMC: A Flexible and Practical Open-source Infrastructure for Enabling Experimental DRAM Studies. In *HPCA.* 2017.
[31] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity. In *ICS.* 2011.
[32] Hwang Huh, Wanik Cho, Jinhaeng Lee, Yujong Noh, Yongsoon Park, Sunghwa Ok, et al. A 1Tb 4b/Cell 96-Stacked-WL 3D NAND Flash Memory with 30MB/s Program Throughput Using Peripheral Circuit Under Memory Cell Array Technique. In *ISSCC.* 2020.
[33] Jaehoon Jang, Han-Soo Kim, Wonseok Cho, Hoosung Cho, Jinho Kim, Sun Il Shim, et al. Vertical Cell Array Using TCAT (Terabit Cell Array Transistor) Technology for Ultra High Density NAND Flash Memory. In *VLSI.* 2009.
[34] JEDEC. JESD218B.01: Solid-State Drive (SSD) Requirements and Endurance Test Method. https://www.jedec.org/standards-documents/docs/jesd218b01. 2016.

[35] Jaeyong Jeong, Sangwook Shane Hahn, Sungjin Lee, and Jihong Kim. Lifetime Improvement of NAND Flash-based Storage Systems Using Dynamic Program and Erase Scaling. In *FAST*. 2014.

[36] Myoungsoo Jung and Mahmut T. Kandemir. Sprinkler: Maximizing Resource Utilization in Many-chip Solid State Disks. In *HPCA*. 2014.

[37] Dongku Kang, Woopyo Jeong, Chulbum Kim, Doo-Hyun Kim, Yong Sung Cho, Kyung-Tae Kang, et al. 256Gb 3b/Cell V-NAND Flash Memory with 48 Stacked WL Layers. In *ISSCC*. 2016.

[38] Ryota Katsumata, Masaru Kito, Yoshiaki Fukuzumi, Masaru Kido, Hiroyasu Tanaka, Yosuke Komori, et al. Pipe-shaped BiCS Flash Memory with 16 Stacked Layers and Multi-level-cell Operation for Ultra High Density Storage Devices. In *VLSI*. 2009.

[39] Samira Khan, Donghyuk Lee, Yoongu Kim, Alaa R. Alameldeen, Chris Wilerson, and Onur Mutlu. The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study. In *SIGMETRICS*. 2014.

[40] Samira Khan, Donghyuk Lee, and Onur Mutlu. PARBOR: An Efficient System-level Technique to Detect Data-dependent Failures in DRAM. In *DSN*. 2016.

[41] Samira Khan, Chris Wilkerson, Donghyuk Lee, Alaa R. Alameldeen, and Onur Mutlu. A Case for Memory Content-based Detection and Mitigation of Data-dependent Failures in DRAM. *IEEE CAL*. 2016.

[42] Samira Khan, Chris Wilkerson, Zhe Wang, Alaa R. Alameldeen, Donghyuk Lee, and Onur Mutlu. Detecting and Mitigating Data-dependent DRAM Failures by Exploiting Current Memory Content. In *MICRO*. 2017.

[43] Chulbum Kim, Jinho Ryu, Taesung Lee, Hyunggon Kim, Jaewoo Lim, Jaeyong Jeong, et al. A 21 nm High Performance 64 Gb MLC NAND Flash Memory with 400 MB/s Asynchronous Toggle DDR Interface. *IEEE JSSC*. 2012.

[44] Doo-Hyun Kim, Hyunggon Kim, Sungwon Yun, Youngsun Song, Jisu Kim, Sung-Min Joe, et al. A 1Tb 4b/cell NAND Flash Memory with tPROG=2ms, tR=110μs and 1.2Gb/s High-Speed IO Rate. In *ISSCC*. 2020.

[45] Jeremie Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines. In *ICCD*. 2018.

[46] Jeremie S. Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-reliability Tradeoff in Modern DRAM Devices. In *HPCA*. 2018.

[47] Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu. D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput. In *HPCA*. 2019.

[48] Myungsuk Kim, Jaehoon Lee, Sungjin Lee, Jisung Park, Youngsun Song, and Jihong Kim. Improving Performance and Lifetime of Large-page NAND Storages Using Erase-free Subpage Programming. In *DAC*. 2017.

[49] Myungsuk Kim, Jisung Park, Genhee Cho, Yoona Kim, Lois Orosa, Onur Mutlu, et al. Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-based Storage Systems. In *ASPLOS*. 2020.

[50] Shine Kim, Jonghyun Bae, Hakbeom Jang, Wenjing Jin, Jeonghun Gong, Seungyeon Lee, et al. Practical Erase Suspension for Modern Low-latency SSDs. In *USENIX ATC*. 2019.

[51] Skanda Koppula, Lois Orosa, A. Giray Yaglikci, Roknoddin Azizi, Taha Shahroodi, Konstantions Kanellopoulos, et al. EDEN: Enabling Energy-efficient, High-performance Deep Neural Network Inference Using Approximate DRAM. In *MICRO*. 2019.

[52] Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, et al. Design-induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms. In *SIGMETRICS*. 2017.

[53] Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin K. Chang, et al. Adaptive-latency DRAM: Optimizing DRAM Timing for the Common-case. In *HPCA*. 2015.

[54] Sang-Hoon Lee. Flash Memory System and Read Method in Flash Memory System. US Patent 8,850,292. 2014.

[55] Nancy Leong, Sachit Chandra, and Hounien Chen. Random Cache Read Using a Double Memory. US Patent 7,423,915. 2008.

[56] Qiao Li, Min Ye, Yufei Cui, Liang Shi, Xiaoqiang Li, Tei-Wei Kuo, et al. Shaving Retries with Sentinels for Fast Read over High-Density 3D Flash. In *MICRO*. 2020.

[57] Jason T. Lin, Steven S. Cheng, and Shai Traister. System, Method and Memory Device Providing Data Scrambling Compatible with On-chip Copy Operation. US Patent 8,301,912. 2012.

[58] Chun-Yi Liu, Jagadish B. Kotra, Myoungsoo Jung, Mahmut T. Kandemir, and Chita R. Das. SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs. In *ASPLOS*. 2019.

[59] Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu. An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms. In *ISCA*. 2013.

[60] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. RAIDR: Retention-aware Intelligent DRAM Refresh. In *ISCA*. 2012.

[61] Ren-Shuo Liu, Chia-Lin Yang, and Wei Wu. Optimizing NAND Flash-based SSDs via Retention Relaxation. In *FAST*. 2012.

[62] Haocong Luo, Taha Shahroodi, Hasan Hassan, Minesh Patel, A. Giray Yaglikci, Lois Orosa, et al. CLR-DRAM: A Low-cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-off. In *ISCA*. 2020.

[63] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management. In *MSST*. 2015.

[64] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. Enabling Accurate and Practical Online Flash Channel Modeling for Modern MLC NAND Flash Memory. *IEEE JSAC*. 2016.

[65] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-recovery and Temperature Awareness. In *HPCA*. 2018.

[66] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation. In *SIGMETRICS*. 2018.

[67] Macronix. Technical Note: Improving NAND Throughput with Two-Plane and Cache Operations. https://www.macronix.com/Lists/ApplicationNote/Attachments/1907/AN0268V1_Improving%20NAND%20Throughput%20with%20Two-Plane%20and%20Cache%20Operations.pdf. 2013.

[68] Rino Micheloni, Luca Crippa, and Alessia Marelli. *Inside NAND Flash Memories*. Springer. 2010.

[69] Micron. Technical Note: NAND Flash Performance Increase Using the Micron PAGE READ CACHE MODE Command. https://www.micron.com/-/media/client/global/Documents/Products/Technical%20Note/NAND%20Flash/tn2901.pdf. 2004.

[70] Micron. NAND Flash Memory Data Sheet: MT29F2G08AACWP, MT29F4G-08BACWP, MT29F8G08FACWP. 2005.

[71] Micron. NAND Flash Memory Data Sheet: MT29F16G08ABABA, MT29F32G-08AFABA, MT29F64G08A[J/K/M]ABA, MT29F128G08AUABA, MT29F16G-08ABCBB, MT29F32G08AECBB, MT29F64G08A[K/M]CBB, MT29F128G-08AUCBB. 2009.

[72] Micron. NAND Flash Memory Data Sheet: MT29F32G08CBACA, MT29F64G-08CEACA, MT29F64G08CFACA, MT29F128G08CXACA, MT29F64G08CECCB. 2010.

[73] Micron. Product Flyer: Micron 3D NAND Flash Memory. https://www.micron.com/-/media/client/global/documents/products/product-flyer/3d_nand_flyer.pdf?la=en. 2016.

[74] Onur Mutlu. Memory Scaling: A Systems Architecture Perspective. In *IMW*. 2013.

[75] Onur Mutlu and Lavanya Subramanian. Research Problems and Opportunities in Memory Systems. *SUPERFRI*. 2015.

[76] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write Off-loading: Practical Power Management for Enterprise Storage. In *FAST*. 2008.

[77] Shiqiang Nie, Youtao Zhang, Weiguo Wu, and Jun Yang. Layer RBER Variation Aware Read Performance Optimization for 3D Flash Memories. In *DAC*. 2020.

[78] Yangyang Pan, Guiqiang Dong, Qi Wu, and Tong Zhang. Quasi-nonvolatile SSD: Trading Flash Memory Nonvolatility to Improve Storage System performance for Enterprise Applications. In *HPCA*. 2012.

[79] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. Improving Performance and Lifetime of NAND Storage Systems Using Relaxed Program Sequence. In *DAC*. 2016.

[80] Jisung Park, Myungsuk Kim, Sungjin Lee, and Jihong Kim. Improving I/O Performance of Large-page Flash Storage Systems Using Subpage-parallel Reads. In *NVMSA*. 2018.

[81] Minesh Patel, Jeremie S. Kim, and Onur Mutlu. The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions. In *ISCA*. 2017.

[82] Moinuddin K. Qureshi, Dae-Hyun Kim, Samira Khan, Prashant J. Nair, and Onur Mutlu. AVATAR: A Variable-retention-time (VRT) Aware Refresh for DRAM Systems. In *DSN*. 2015.

[83] Samsung. 32Gb A-die NAND Flash Datasheet. 2009.

[84] Youngseop Shim, Myungsuk Kim, Myoungjun Chun, Jisung Park, Yoona Kim, and Jihong Kim. Exploiting Process Similarity of 3D Flash Memory for High Performance SSDs. In *MICRO*. 2019.

[85] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. MQSim: A Framework for Enabling Realistic Studies of Modern Multi-queue SSD Devices. In *FAST*. 2018.

[86] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, et al. FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives. In *ISCA*. 2018.

[87] Toshiba. NAND Memory Toggle DDR1.0 Technial Data Sheet. 2012.

[88] Ravi K. Venkatesan, Stephen Herr, and Eric Rotenberg. Retention-aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *HPCA*. 2006.

[89] Yaohua Wang, Arash Tavakkol, Lois Orosa, Saugata Ghose, Nika Mansouri Ghiasi, Minesh Patel, et al. Reducing DRAM Latency via Charge-level-aware Look-ahead Partial Restoration. In *MICRO*. 2018.

[90] ONFI Workgroup. Open NAND Flash Interface Specification Revision 4.2. https://media-www.micron.com/-/media/client/onfi/specs/onfi_4_2-gold.pdf. 2020.

[91] Guanying Wu and Xubin He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *FAST*. 2012.
[92] Qin Xiong, Fei Wu, Zhonghai Lu, Yue Zhu, You Zhou, Yibing Chu, et al. Characterizing 3D Floating Gate NAND Flash: Observations, Analyses, and Implications. *ACM TOS*. 2018.
[93] Jeff Yang. High-efficiency SSD for Reliable Data Storage Systems. In *FMS*. 2011.
[94] Xianwei Zhang, Youtao Zhang, Bruce R. Childers, and Jun Yang. Restore Truncation for Performance Improvement in Future DRAM Systems. In *HPCA*. 2016.
[95] Da Zheng, Disa Mhembere, Randal Burns, Joshua Vogelstein, Carey E. Priebe, and Alexander S. Szalay. FlashGraph: Processing Billion-node Graphs on an Array of Commodity SSDs. In *FAST*. 2015.