PhasedRR: Read Reclaim Scheduling without Page-level Access Counting

Cai Li, Jun Li, Zhibing Sha, Zhigang Cai, Jianwei Liao
College of Computer and Information Science, Southwest University, Chongqing, China
Corresponding Author: Jianwei Liao, E-mail: *liaotoad@gmail.com*

***Abstract*—Read reclaim (RR) is used to avoid read disturb errors by periodically migrating all valid data pages of the RR block to other free blocks. In order to reduce the negative effects of RR that will block user I/O requests for a long time, existing RR optimization schemes migrate the hot read data pages in advance before the appearance of read disturb errors, if there are idle intervals between two I/O requests. Such advanced RR schemes, however, employ page-level counters for tracking read frequency on the pages in the RR block to direct early hot read data migrations, which must require considerable space overhead and then impact I/O responsiveness. To address this issue, this paper proposes a novel RR scheduling scheme, called as *PhasedRR*. Instead of using page-level access counters, *PhasedRR* employs working sets to track read frequency on the data pages of the RR blocks, thus reducing the space overhead caused by maintaining the counters. Furthermore, *PhasedRR* supports phased migration for moving the data pages having varied levels of read hotness in the RR block to another block(s) in different phases before the hard RR threshold, meanwhile the cold data pages are remained in the RR block until the block read count reaches the threshold. Through a series of simulation experiments based on several realistic disk traces, we illustrate that our proposal can reduce the overall I/O latency by `31.3%` on average, in contrast to existing RR scheduling methods.**

***Index Terms*—Solid-state drivers, read disturb, read reclaim , reliability.**

I. Introduction

NAND flash memory-based solid-state drives (SSDs) have become the mainstream storage devices, thanks to their advantages of high performance and small size [1], [2]. To minimize the per-unit price of SSDs, the feature size of flash cells is gradually reduced to below `10nm`, and flash density is then driven to Multi-Level Cell (MLC), Triple-Level Cell (TLC) or even Quad-Level Cell (QLC) [3]. The feature size shrinking of flash cells results in the flash memory being more vulnerable to its inherent noises, such as programming interference noise, retention disturb and read disturb [4]. Therefore, efficiently coping with such noises to guarantee the reliability of flash memory-based SSDs becomes a challenging issue in both academics [5], [6] and industry [7], [8].

Specially, read disturb is an unavoidable phenomenon that impacts the threshold voltages of the unread pages when performing read operation on specific pages in the same block. With the accumulation of such side-effects, raw bit errors may happen in the unread data pages [9], [10]. Error correction codes (ECCs) have been applied in modern SSDs to recover the data having raw bit errors through read retries, with the cost of impacting I/O responsiveness of SSDs. However, ECCs does not work if the number of raw bit errors in the original data page surpasses the correction capacity of ECCs, indicating the data is permanently damaged [11].

In order to avoid permanent data lost caused by accumulated impacts of read disturb, an operation of read reclaim (RR) has been proposed to migrate the valid data pages to another block before they are damaged, to reset the impacts of read disturb [12]. When the read count on the block is greater than a preset bound, termed as the hard RR threshold, an RR operation is triggered to avoid read disturb errors [13], [14]. In the process of read reclaim, it first reads all valid pages in the RR block and flush them to another block(s), while the accumulated read count on the block approaches the fixed RR threshold. After that, the RR block is rejuvenated as a new block after an erase operation. Since SSD devices cannot satisfy user I/O requests during the RR process, I/O responsiveness will be greatly affected. More importantly, high-density flash memory (e.g. TLC or QLC) is more vulnerable to read disturb [4]. For example, the RR threshold of SLC SSDs is one million, whereas the threshold of TLC SSDs decreases to `20`K-`40`K [10], [15], [16]. In other words, high-density SSDs require triggering RR operations more frequently, which must impact I/O responsiveness and the lifetime of SSDs[1].

A number of RR optimizations aim to mitigate negative effects of read disturb in high-density SSDs [17], [20], [21], [22]. Liu et al. [17] presented an innovative approach utilizing shadow blocks to hold hot read data. Specifically, besides holding the hot read data, the shadow blocks consist of certain invalid data pages or free pages, since both kinds of pages are immune to read disturb. This mechanism, however, purposely retains certain free pages and valid pages in the shadow blocks, which confines the improvement on read latency and read reclaim cycles. To further reduce the negative effects on I/O responsiveness caused by RR operations, Liao et al. [20] constructed mathematical models to direct migrating the hottest data pages in advance, for minimizing side-effects of read disturb and the number of RR operations, on the basis of their previous work [22]. In addition, Zhang et al. [21] proposed a strategy dispersing hot read page across multiple

[1]The SSD devices support a limited number of erases on the block, and every RR process is completed with an erase operation on the block after the valid data pages have been migrated.

blocks to prevent the concentration of hot read data in few blocks, thus reducing the number of RR operations.

We summarize that migrating the frequently accessed data pages of the RR block in advance can reduce the impacts of read reclaim, as page migrations can be completed in idle intervals between two I/O requests. However, such optimization schemes employ page-level counting to record the read frequency of data pages, and such counters consume space overhead on dynamic random-access memory (DRAM) of SSDs, which must impact I/O responsiveness. This is because the available DRAM space for caching the most frequently accessed data is reduced. In order to minimize negative impacts of read disturb and the expense in RR operations, this paper proposes a counter-less RR scheduling mechanism, called *PhasedRR* on the basis of recent working sets of page access tracks. In brief, this paper makes the following three contributions:

- We introduce a working set-based scheme for identifying the hotness level of data pages in the RR block, which does not require page-level counting to track read frequency, thus reducing the space overhead caused by keeping the counters to direct RR scheduling.
- We propose a phased migration scheme for moving the data pages having varied level of hotness in the RR block to another block(s) in different phases, meanwhile the cold data pages are remained in the RR block. It can cut down the ECC overhead and improve I/O responsiveness, as the most frequently data pages are migrated to other available block(s) in early phases.
- We perform a series of evaluation tests by using disk traces of real-world applications. Our measurements show that our proposal can reduce the average read latency and erase operations by `33.6%` and `6.1%` on average, in contrast to existing RR scheduling methods.

The rest of this paper is structured as follows: Section II introduces the background knowledge and our motivations. Section III describes the details of the proposed mechanism. The evaluation experiments and discussions are presented in Section IV. Finally, the paper is concluded in Section V.

II. Background and Motivations

A. Flash Memory and Read Disturb

NAND flash memory is organized as two-dimensional arrays of floating-gate transistors. A number of cells electrically connected to a wordline (WL) that consists of multiple pages in high-density flash memory and multiple WLs form a block. Specifically, flash-based SSDs usually consist of one or more planes, and each plane contains several blocks. A block is a basic unit for erasing, and is composed of many pages that are the basic units of write/read operations [23].

Read disturb is a circuit-level noise in NAND flash memory-based SSDs, which is caused by intensive read operations [14]. Figure 1(a) shows the voltage settings of a read operation in the TLC flash memory. As seen, a read reference voltage of V_r is applied to the corresponding wordline (i.e., WL_1, and a pass-through voltage of V_{pass} that is higher than V_r, is exerted to other victim wordlines in the same blocks. It is true that V_{pass} is lower than the programming voltage, it induces a weak programming effect on the flash cells, and will shift their threshold voltages unintentionally. These shifts accumulate over time, and they will become significant enough to change the state of some flash cells, thus leading to the occurrence of raw bit errors.

A representative example of state change in the flash cell is shown in Figure 1(b), in which the reference voltage of V_{r1} fails to distinguish the original *P1* state and the disturbed *ER* state. Although modern SSDs are equipped with ECCs to recover raw bit errors, the consequent cost of read retries impacts I/O responsiveness of SSDs. More importantly, it will lead to data damages of SSDs if bit errors accumulate beyond the capacity of ECCs, which compromises the reliability and practicability of flash memory-based SSDs.

B. Read Reclaim and Optimizations

The read reclaim (RR) strategy is proposed for avoiding permanent data damages, through migrating data pages from an impacted block to another free block(s) [12]. When a block undergoes more reads than a preset upper bound on the number of read operations allowed by the block, called the hard RR threshold, the RR operation is consequently triggered. In an RR operation, it first reads all valid pages in the RR block and flush them to another block(s), termed as page moves or page migrations, while the accumulated read count on the block approaches the hard RR threshold. After that, the RR block will be erased as a new block.

In fact, the page moves in RR processes correspond to multiple pairs of read and write on the flash memory, which block enqueued user I/O requests and then degrade I/O performance of SSDs. For better improving read reclaim efficiency, numerous optimization strategies have been proposed, which can be classified into three categories:

Employing dedicated blocks for holding hot read data. Liu et al. [17] presented read-leveling to allocate hot read data pages to dedicated shadow blocks, which originally hold invalid pages and free pages that are immune to read disturb. As a result, it can minimize the frequency of RR operations on these shadow blocks, though they hold some hot read data. Note that, however, this approach does not perform well for utilizing the storage space in the shadow blocks, thus limiting the improvements on read latency and read reclaim cycles.

Similarly, Wu et al. [18] proposed an scheme of adaptive cell bit-density with in-place reprogramming (IPR), to reduce the total number of read reclaim. This approach reprograms the old Most Significant Bit (MSB) pages and transforms TLC blocks into MLC ones, by considering the read limit for triggering read reclaim operations on MLC blocks is much greater than the limit on the original TLC blocks. Consequently, the read reclaim operations on the reprogrammed blocks can be significantly postponed.

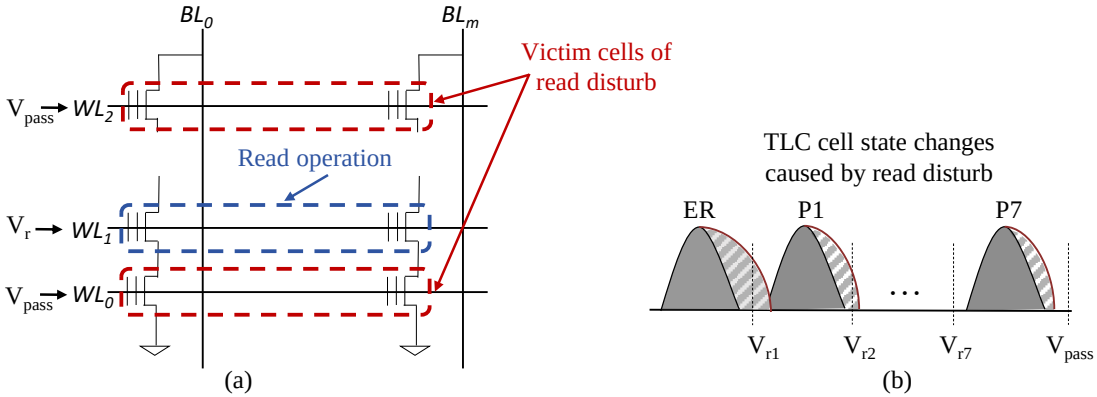

Fig. 1: Read disturb in TLC flash memory. (a) Voltage settings for satisfying a read operation, including the reference voltage of V_r and read pass voltage of V_{pass}. (b) The reference voltage of V_{r1} cannot identify the disturbed *ER* state and the original *P1* state [16].

Distributing hot read pages onto multiple blocks. Zhao et al. [19] considered SSD blocks have disparate initial attributes of read counts and P/E cycles, and then proposed a model to estimate the optimal read limits of blocks for optimizing read reclaim. Specifically, it relocates data pages to the blocks by matching read count of move-in pages and the estimated read limit of the destination blocks when carrying out RR operations. This approach effectively reduces the frequency of read reclaim operations and minimizes the ECC overhead of read data pages. In order to better conduct RR operations, Liao et al. [20] built two mathematical models to decide the time point of triggering RR process and the order in which pages are migrated in the RR process.

In addition, Zhang et al. [21] proposed a strategy of dispersing read hot data widely across multiple available blocks, thereby preventing the concentration of hot read data within only few blocks. To this end, their method migrates some hot read data pages of RR block to the blocks containing a certain proportion of cold read data pages that are extracted from user I/O requests. This method increases the number of destination blocks in each RR operation, which can ultimately reduce the frequency of RR operation.

Machine learning-based optimization on RR scheduling. To further alleviate the impacts of RR on user I/O requests, Li et al. [22] incorporated reinforcement learning into read-refresh (RR) scheduling, to direct the page moves and the erase operation during idle intervals between I/O requests. More clearly, it predicts idle time intervals according the history information on I/O workloads by using the reinforcement learning model, to help deciding the number of page moves and whether perform an erase operation or not. This method can minimize the side-effects of read reclaim, as RR-relevant operations are completed in the idle intervals, without impacts on subsequent I/O requests as much as possible.

C. Motivations

The routine RR operation migrates all valid data pages in the RR block when its endured read accesses reaches the hard RR threshold. It has been proven that the distribution of access frequency varies across different benchmarks, and the valid pages in the same RR block may have varied level of access hotness [20], [21]. Consequently, some previous studies tend to treat frequent access data specially for relieving the impacts of read disturb, such as migrating hot read data pages to insusceptible blocks (shadow blocks or blocks with high optimal read count), or distributing them to a range of blocks to avoid hot read data within only a few blocks.

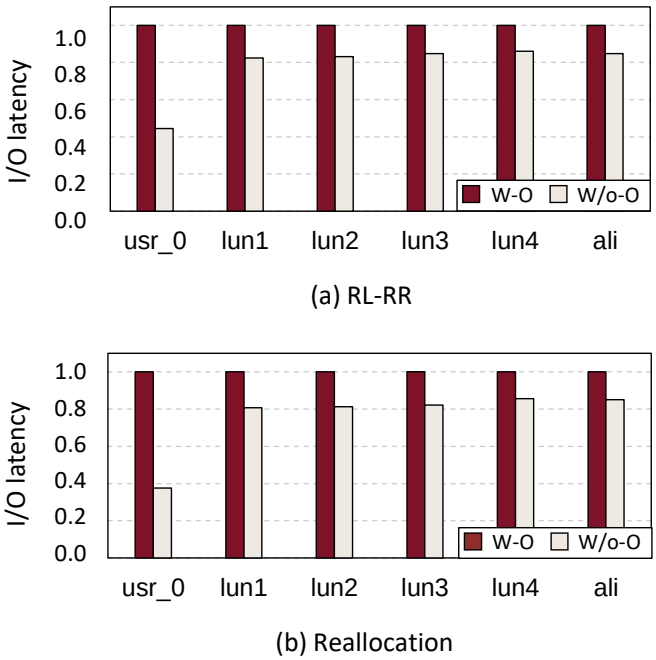

Fig. 2: The results of average I/O latency when using RR scheduling methods of *RL-RR* [22] and *Reallocation* [20], with (*W-O*) or without space overhead (*W/o-O*). Note we assume both *RL-RR* and *Reallocation* do not require memory space for holding page-level counters, to quantify the negative impacts caused such counters.

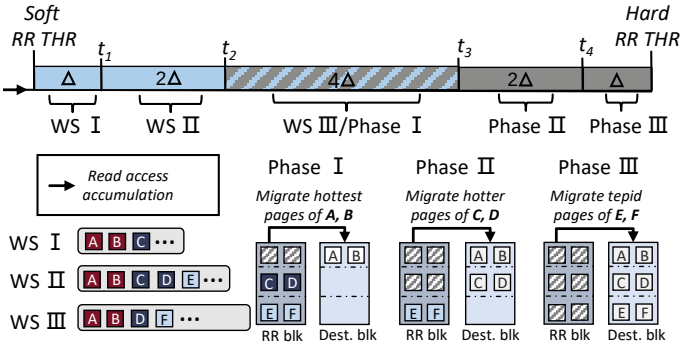

Fig. 3: High-level architectural overview of *PhasedRR*. Note that Δ is used to decide the window size at different phases for obtaining the working set of read accesses and carrying out page migrations. It is a parameter of read accesses on the block, related to the configurable soft RR threshold and the hard RR threshold.

The fundamental similarity among these advanced RR optimization strategies lies in the identification of hot read data. However, page-level counting is commonly used to identify hot read data pages, which results in certain space overhead, as the page-level counters are required to be kept in dynamic random-access memory (DRAM). In other words, the size of available DRAM space used for caching hot data becomes smaller, which must impact I/O responsiveness of SSD devices.

In order to quantify the negative effects caused by page-leveling counting in RR optimization, we replayed some read-intensive I/O traces of real-world applications in a SSD simulator, when using two state-of-the-art RR scheduling methods of *RL-RR* [22] and *Reallocation* [20]. In the tests, we also assume they do not require page-level counting to identify hot read data pages to direct RR scheduling. Section IV will present the experimental settings and benchmark specifications.

Figure 2 illustrates the results of average I/O latency after replaying the selected benchmarks. Obviously, page-level counters does greatly impacts I/O responsiveness by between `13.9%` and `62.4%`. This fact verifies page-level counters occupy the DRAM space, indicating less DRAM space can be dedicated for buffering hot accessed data to absorb read and write requests, thus leading to more accesses onto the underlying NAND flash array and worsening I/O performance of SSD devices.

III. Design and Implementation of *PhasedRR*

A. System Overview

The basic principle of *PhasedRR* is to use working sets of read accesses for sifting hot read data pages, to support migrating the data pages with varied hotness levels in the different phases, after the RR process is triggered. Figure 3 illustrates the high-level architectural overview of *PhasedRR*. When the access count of the given block approaches a predefined threshold (i.e., *Soft RR THR* in the figure), we put the block into the RR candidate list. After that, we generate three working sets of read accesses (i.e. *WS I-WS III*) in the following three windows. By resorting to the obtained working sets, we can identify hot data pages and then migrate the pages having varied levels of read hotness in advance from the RR block at different phases, called as *phased read reclaim*. Note that, we not only construct the working set of *WS III*, but also identify hot read data pages in the third time window, which is also named as *Phase I*.

According to the design principle of *PhasedRR*, the data pages having less access frequency can remain in RR blocks until the read count of block reaches the hard RR threshold. As a result, it can reduce the time caused by error corrections since the hot pages are migrated to another block(s), and then improve I/O responsiveness for application requests.

B. Hot Data Page Identification

For the purpose of identifying hot read data pages in the RR block without page-level read counters, when the read count of the block reaches the pre-defined soft RR threshold, we propose a working set-based sifting method. By referring to the working set model in virtual memory management [24], we define a working set of read accesses as $W(t, \tau)$ to record the collection of data pages that were accessed in the interval of $(t- \tau, t)$, with respect to a given block.

By referring back to Figure 3, *PhasedRR* generates three working sets successively while the read count exceeds the pre-defined soft RR threshold, labeled as $W(t_1, \tau_1)$, $W(t_2, \tau_2)$ and $W(t_3, \tau_3)$, respectively. In which, the size of τ_1, τ_2, and τ_3 is defined as Δ, 2Δ, and 4Δ. Note that Δ is a parameter related to the configurable soft RR threshold and the hard RR threshold, according to the design of *PhasedRR*. As a consequence, we can categorize the hot data pages of the RR block into three types according to their read hotness, by following Equations (1) - (3).

$$HTT\ pgs\ =\ \bigcap_{i=1}^{3} W(t_i, \tau_i) \tag{1}$$

$$HTR\ pgs\ =\ \bigcup_{\substack{i=1,j=1\\ i\neq j}}^{3} (W(t_i, \tau_i) \cap W(t_j, \tau_j)) - HTT\ pgs \tag{2}$$

$$TPD\ pgs\ =\ \bigcup_{i=1}^{3} W(t_i, \tau_i) - HTT\ pgs - HTR\ pgs \tag{3}$$

where *HTT pgs*, *HTR pgs* and *TPD pgs* represent hottest pages, hotter pages and tepid pages respectively.

To be specific, the valid data pages that appear in all three working sets are classified as hottest pages, the pages that appear in two of working sets are classified as hotter pages, and the pages that appear in only one of working sets will be classified as tepid data pages. On the contrary, the data pages that do not appear in the working sets, will be classified as cold data pages.

C. Phased Data Page Migration

When the data pages of the RR block have been identified as different types by using the generated working sets of read access tracks, *PhaseRR* adopts migrating the data pages having varied levels of read hotness in different phases before reaching the hard RR threshold. As illustrated in Figure 3, the hottest data pages will be migrated to other block(s) in *Phase I*, the hotter data pages and the tepid data pages will be migrated in *Phase II* and *Phase III*, respectively.

In order to minimize the impacts of migrating data pages, our proposed phased migration scheme tries to move data pages while there is an interval between two I/O requests in the specific phase. If there are not enough idle intervals in the given phase, the remainder of data pages that are required to be migrated before the end of the phase, regardless of I/O intensity of user applications.

Note that all cold data pages in the RR block are expected to be migrated to other block(s) with a mandatory fashion, once the read count of the block reaches the hard RR threshold.

D. Implementation Specifications

Algorithm 1 demonstrates the implementation details of the proposed scheme, with respect to identifying hot read data pages, and migrating the pages having varied hotness levels in different phases. As seen, *Lines 3-15* elaborate the process of identifying the hot read data pages without page-level read counting after the read count of the block approaches the soft RR threshold. Then, *Lines 16-28* illustrate how to migrate data pages of the RR block based on their appearance in the previous working sets, before the block read count exceeds the hard RR threshold. At last, *Lines 29-34* present the specification on carrying out mandatory page moves and the erase operation to complete the RR operation of the block while the hard RR threshold is reached.

Algorithm 1 Hot data identification and phased migration.

1: **Input:** The *page_num* of the accessed page;
2: **Output:** Null;
3: **if** $blk.rd\_cnt \geq soft\_THR$ **and** $blk.rd\_cnt < t_1$ **then**
4: /* Add page_num to working set I*/
5: $Add(page\_num, WS\ I)$;
6: **else if** $blk.rd\_cnt \geq t_1$ **and** $blk.rd\_cnt < t_2$ **then**
7: /* Add page_num to working set II*/
8: $Add(page\_num, WS\ II)$;
9: **else if** $blk.rd\_cnt \geq t_2$ **and** $blk.rd\_cnt < t_3$ **then**
10: /* Add page_num to working set III*/
11: $Add(page\_num, WS\ III)$;
12: **if** The visiting page appears in all *WSs* **then**
13: /* Trigger the page moves for the hottest pages*/
14: $Trigger\_PM\_Idle\_Time()$;
15: **end if**
16: **else if** $blk.rd\_cnt \geq t_3$ **and** $blk.rd\_cnt < t_4$ **then**
17: **if** The visiting page appears in two of *WSs* **then**
18: /* Trigger the page moves for the hotter pages*/
19: $Trigger\_PM\_Idle\_Time()$;
20: **end if**
21: **else if** $blk.rd\_cnt \geq t_4$ **and** $blk.rd\_cnt < hard\_THR$ **then**
22: **if** There remain hotter pages in block **then**
23: /* Migrate all of the hotter pages immediately*/
24: $Trigger\_PMs\_ALL()$;
25: **else if** The visiting page appears in one of *WSs* **then**
26: /* Trigger the page moves for the tepid pages*/
27: $Trigger\_PM\_Idle\_Time()$;
28: **end if**
29: **else if** $blk.rd\_cnt \geq hard\_THR$ **then**
30: /* Migrate all of the valid pages immediately */
31: $Trigger\_PMs\_ALL()$;
32: /* Erase the RR block */
33: $EraseBlock()$;
34: **end if**
35: **return**

IV. Experiments and Evaluation

A. Experiment Settings

To assess the effectiveness of our proposed scheme, we employed the *SSDsim* simulator, to replay the selected disk access traces of real-world applications. In other words, we implemented the proposed mechanism as a part of the flash translation layer (FTL) inside *SSDsim*. We use a local ARM-based machine as SSD controllers which usually have limited computation power and memory capacity [22]. The machine has an ARM Cortex A7 Dual-Core with 800MHz, 128MB of memory and 32-bit Linux (*ver 3.1*).

The LDPC scheme is employed to correct raw bit errors in our configuration, as the read latency relies on the level of LDPC soft decision [25], [26]. The initial read time is set as 0.085 ms, with an increment of 0.024 ms per read retry. As a result, the read time will span from 0.085 ms to 1.099 ms, corresponding to seven-level of LDPC soft decisions. The parameter settings for our tests are specifically described in Table I, and the RR threshold is set to 25k by referring to [27]. To reflect the impact of diminished cache space caused by page-level counting, we set the size of each page-level counter as 2 bytes when implementing the related work, by following [18].

Considering read disturb generally takes place in read-intensive workloads, our evaluation tests primarily utilized 6 read-intensive I/O traces, as well as two write-dominate I/O traces from several I/O trace repositories. Among them, 4 traces are collected by an Enterprise Virtual Desktop Infrastructure [28], including *additional-03-2016021711-LUN3*, *additional-03-2016021812-LUN0*, *additional-03-2016021808-LUN0* and *additional-03-2016021812-LUN6*, labelled as *lun1* to *lun4*. In addition, *websearch_1* (labeled as *ws_1*) is from the UMass Trace Repository [29], *usr_0* and *web_0* are from Microsoft Research Cambridge [30], and *alibaba_121* (labeled as *ali*) is from the Alibaba center [31].

Table II presents the details on the selected 8 block I/O traces. In which, the metric of *Hot read ratio* indicates the

TABLE I: Experimental settings of *SSDsim*

SSD parameters	
(Channel, Chip)	(8, 2)
(FTL, GC trigger)	(Page-level, 30%)
Overprovide	25%
Transfer time per byte	5ns
DRAM capacity	64MB
Hard RR threshold	25K
Extra read-retry time	0.024ms
Maximum LDPC level	7
Chip parameters	
(Die, Plane, Block, Page)	(1, 2, 1280, 256)
(Page size, Cell density)	(8KB, TLC)
Program latency (LSB-CSB-MSB)	(0.5, 2, 5.5)ms
(Read latency, Erase latency)	(0.085, 15)ms

TABLE II: Specifications on selected traces

Traces	**Req. #**	**Read R**	**Read SZ**	**Hot R**	**FP**
ws_1	1,055,448	99.9%	15.2KB	90.8%	0.02GB
usr_0	2,237,889	40.4%	40.9KB	46.2%	2.44GB
web_0	2,029,945	29.8%	29.0KB	19.7%	7.26GB
lun1	1,728,463	69.4%	21.9KB	3.2%	26.0GB
lun2	2,683,696	72.0%	26.6KB	0.4%	38.8GB
lun3	2,162,563	81.3%	20.7KB	0.5%	38.5GB
lun4	1,896,006	79.4%	24.1KB	0.7%	31.2GB
ali	2,340,016	77.2%	17.9KB	39.2%	1.11GB

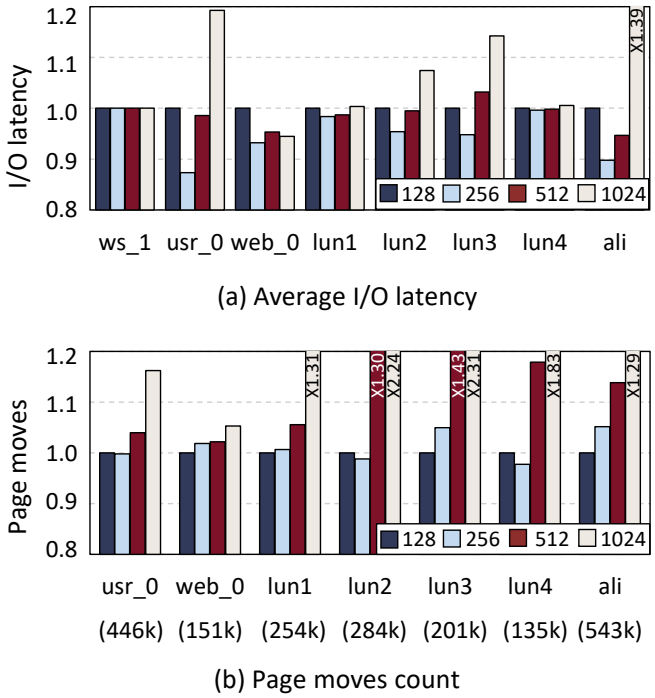

Fig. 4: Sensitivity analysis on the value of Δ in *PhasedRR*. Note that the numbers under the X-axis are the absolute values in the cases of Δ is `128`.

concentration level of read access addresses within the traces. In other words, it denotes the ratio of frequently requested addresses (i.e., those are accessed not less than four times) to the entire read address space. The *Footprint* metric represents the total size of requested addresses when running the benchmark. In order to trigger a considerable number of RR operations, we repeat the execution of the selected block I/O traces, by referring to [16], [20], [22].

Apart from the proposed method of *PhasedRR*, the following three schemes are selected as comparison counterparts in our evaluation.

- *Baseline*, which is the conventional read refresh. It moves all valid data from the RR block to another new block when the hard RR threshold is approached.
- *RL-RR* [22], which employs reinforcement learning to predict idle time intervals between two I/O requests, and then makes use of idle time intervals to perform optimal (partial) read refresh operations (i.e. page moves or erase). It can minimize the side-effects of routine RR operations, thus enhancing I/O responsiveness.
- *Reallocation* [20], which utilizes a matching model to decide where to migrate hot data pages in the RR block, and a timing selection model to decide when to migrate the data pages in the RR blocks.

Before carrying out this experiment, the SSD device was warmed up by running write-intensive workloads [30], to simulate an SSD that has been used for a certain period. In other words, all SSD blocks may exhibit varying characteristics with regard to their P/E cycles.

B. Sensitivity Study on the Value of Δ

This section carries out sensitive tests to decide the default value of Δ in *PhasedRR*, which is directly related to the size of time window to collect working sets of read accesses and perform phased page migrations. In fact, Δ is a sensitivity parameter, and the choice means an engineering trade-off. A small value of Δ may fail to sift the hot read data pages and reduce the chance of migrating pages early during idle intervals between two I/O requests, whereas a large value will lead to an increase on page migrations before reaching the fixed hard RR threshold.

We have conducted sensitive tests with different values of Δ, varying from `128` to `1024`. Figure 4(a) and 4(b) present the results of average I/O latency and page moves respectively, when using varied size of Δ with *PhasedRR*. As illustrated, it can yield the best results in most traces, while the value of Δ is `256`. Therefore, we employ `256` as the default value of Δ in *PhasedRR* in this paper, implying the soft RR threshold is `22.44`K.

C. Performance Results and Discussions

To validate the effectiveness of our proposed RR scheme, we used three primary performance measures in our experiments: (a) *I/O latency*, (b) *RR statistics*, and (c) *Erase count*.

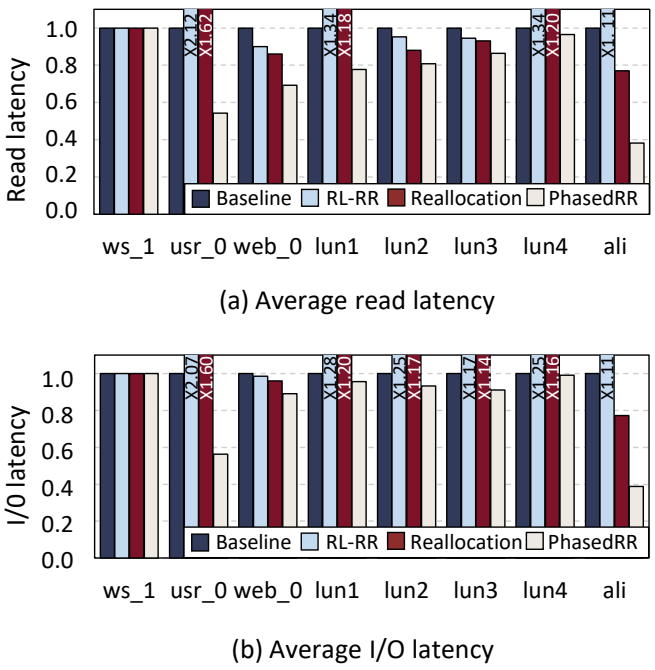

Fig. 5: I/O performance comparison after replaying the selected block I/O traces. Note that *ws_1* has a small size of footprint and all accessed data can be buffered in the cache, so that no RR operation is triggered and all RR scheduling schemes do not make any difference.

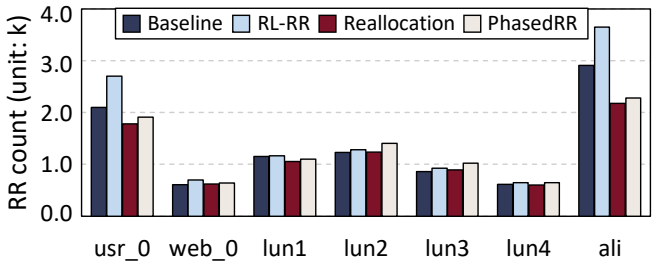

Fig. 6: The number of RR operations after running the selected benchmarks, with varied RR scheduling methods.

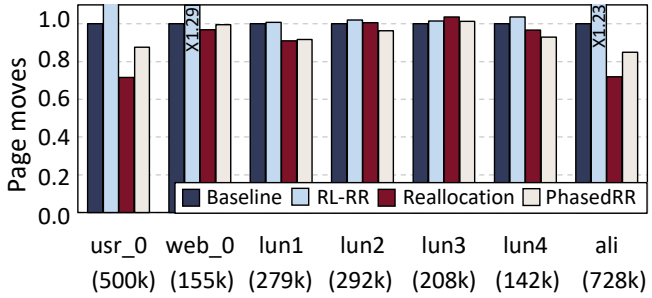

Fig. 7: The number of page moves occurred in RR processes when running the benchmarks. Note that the numbers under the X-axis are the absolute values of *Baseline*.

1) I/O latency: The I/O latency is the primary indicator of storage's performance, and Figure 5 shows the results of I/O latency after running the selected block traces. We can observe that the proposed *PhasedRR* scheme outperforms other comparison counterparts, in terms of average read latency and overall I/O latency. More exactly, *PhasedRR* declines the average read latency by 24.5%, 46.0%, and 30.3%, in contrast to *Baseline*, *RL-RR*, and *Reallocation* respectively. Consequently, *PhasedRR* can reduce the average I/O latency by 17.1%, 46.9%, and 30.0%, when comparing to the selected comparison counterparts.

Different from the related work of *Reallocation* and *RL-RR*, our proposal of *Phased RR* does not require page-level counting to identify hot read data to trigger early page migrations, which can leave more cache space for buffering the most frequently accessed data, thus improving I/O performance. Besides, compared with *Baseline*, our proposed *PhasedRR* approach migrates hot read pages from the RR block to other available block(s), once the read count of the block reaches the pre-defined soft threshold, so that it saves the time overhead caused by ECCs when reading on them. Note that such page migrations can be completed in idle intervals during I/O processing, thus alleviating I/O blocking caused by the RR operations, and boosting I/O responsiveness.

2) RR Statistics: The section analyzes RR operations after running the selected benchmarks with varied RR scheduling methods, and Figure 6 presents the relevant results. We define an RR operation as at least one data page of the RR block has been migrated to other available block.

As read, *PhasedRR* reduces the number of RR operations by 0.7%, and 3.1% on average, in contrast to *Baseline* and *RL-RR*. This is because *PhasedRR* does not require keeping page-level counters in the cache, indicating more hot accessed data can be buffered in the cache and more read requests can be absorbed by responding the buffered data. The noticeable clue shown in the figure is about *Reallocation* yields the best in terms of *RR count*. We suggest this is because *Reallocation* purposely distributes hot accessed data onto multiple blocks, which can balance read access distribution, thus decreasing the number of RR operations, even though the total number of read operations keep unchanged.

The measure of page moves in RR processes is another critical performance indicator of RR scheduling schemes. Figure 7 presents the results of page moves after running the selected traces. Obviously, it shows a similar tendency to the results of occurred RR operations after running the selected traces, which verifies the fact in which the number of page moves caused by RR operations is positively related to the number of RR operations.

3) Erase Count: The metric of erase count induced by garbage collection (GC) and RR operations is used to reflect the endurance of the SSD, so that we record the number of erase operations after running the selected benchmarks, by using four RR scheduling approaches. As the results presented

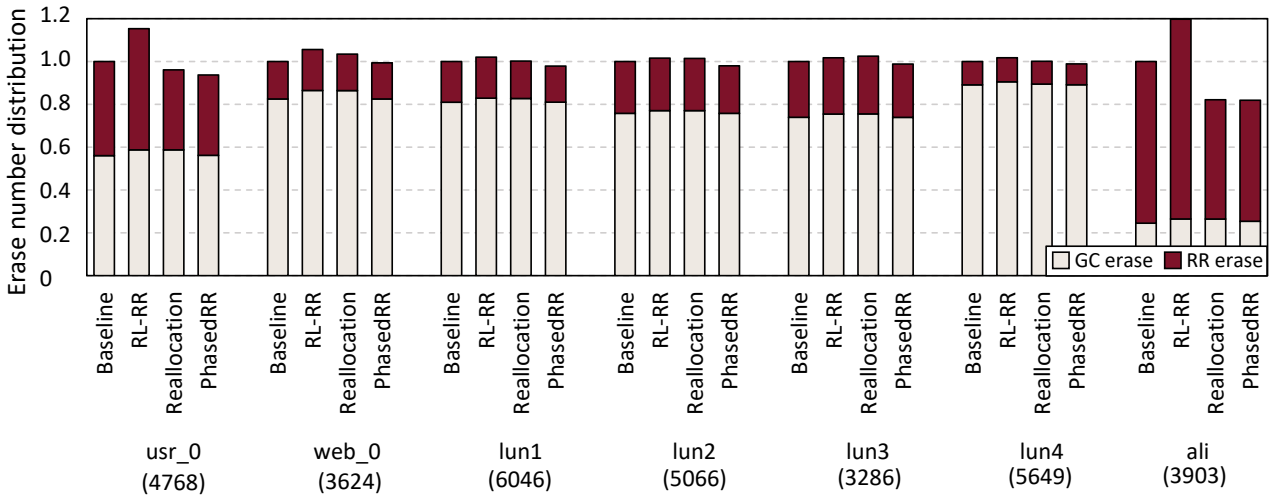

Fig. 8: Comparison of erase operations caused by both GC and RR operations, with varied RR scheduling schemes. Note that the numbers under the X-axis are the absolute numbers of erase operations with *Baseline*.

in Figure 8, the proposed *PhasedRR* approach yields the smallest number of erase operations after running all selected benchmarks, corresponding to a reduction of erases by 4.5%, 11.3%, and 2.5% in contrast to *Baseline*, *RL-RR*, and *Reallocation*.

With respect to the erase count caused by GC operations, we see *PhasedRR* and *Baseline* perform better than *RL-RR* and *Reallocation*. This is because *RL-RR* and *Reallocation* employ (in-cache) page-level counters to sift hot read data pages in the RR block and migrate them in advance, for relieving negative effects of read disturb. In other words, less cache space can be contributed to hold hot data, thus resulting in more flush operations onto underlying flash array, that lead to more GC operations after running the benchmark.

Regarding the erase count caused by RR operations, *PhasedRR* preferably migrates the frequently accessed data pages from the RR blocks having a high read error rate, which can minimize read retries on the RR block and then slow down the pace to the hard RR threshold. As a consequence, it yields a smaller number of erases resulted by RR operations.

D. Identification Accuracy and Overhead Analysis

1) Hot Data Identification Accuracy: To validate the effectiveness of working set-based hot data identification adopted by *PhasedRR*, we calculated the accuracy of identifying hot read data pages in the RR block. Specifically, the identification accuracy is defined as proportions of truly hot read pages to the identified hot pages, including the hottest pages, hotter pages, and tepid pages have been measured. By referring to [18], a data page can be defined as the hot read page if its read count exceeds the average read count of all valid pages within the same block.

According to the results shown in Figure 9, the percentage of hot read pages in the identified hottest pages, the hotter pages, and the tepid pages is 91.4%, 68.1%, and 59.2%

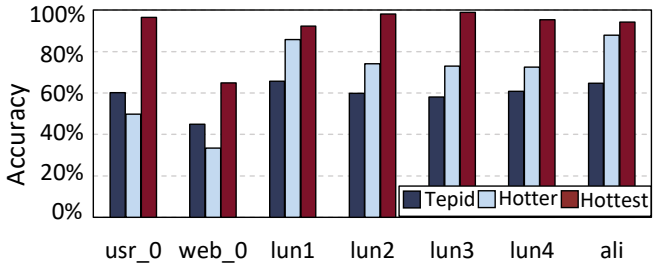

Fig. 9: Accuracy of identifying hot access data pages after running the selected benchmarks with *PhasedRR*.

respectively. In other words, it achieves the overall accuracy of 73.8%, indicating that the majority of genuinely hot pages are captured. Another interesting clue is about our approach yields a low identification accuracy when replaying the trace of *web_0*. We argue this is because this benchmark has a very small number of read requests, and a major part of hot read data pages can be buffered in the DRAM cache. As a result, the working set-based sifting scheme may fail to locate such hot data pages, by only analyzing the access tracks in the working sets.

2) Time and Space Overhead: Existing approaches commonly employ page-level counters to identify hot read pages for directing RR scheduling, which must lead to non-negligible space overhead. In our evaluation, we set each page-level counter occupies 2 bytes, implying the space overhead of *RL-RR* and *Reallocation* approaches becomes up to 16.0MB according to our experimental configuration.

The proposed *PhasedRR* method can identify the majority of hot read pages with less overhead, by using working sets of read accesses. With respect to each RR block, we maintain three block-level working sets to track read accesses.

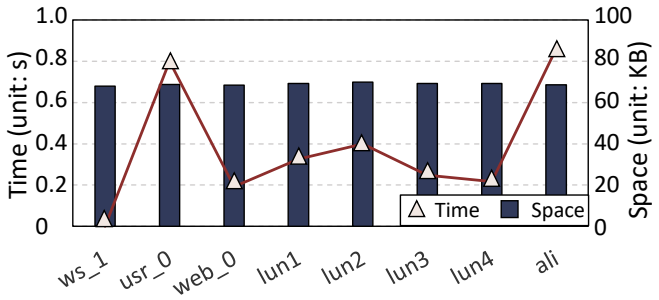

Fig. 10: Time and space overhead after running the selected benchmarks with *PhasedRR*.

In each working set, we employ a bit to reflect whether the relevant page has been accessed or not. Figure 10 presents the space overhead of *PhasedRR* after replaying the selected benchmarks. As seen, it consumes less than 80KB memory space for holding the working sets of read accesses, taking up 0.11% of DRAM cache space in our testbed.

The main time overhead of *PhasedRR* is caused by scanning the established working sets to categorize the hot read data pages into three types, when the RR operation is triggered. We record the extra time cost after running the benchmarks on the ARM-based platform, as shown in Figure 10, *PhasedRR* results in time overhead by between 0.19 and 0.86 seconds, accounting for less than 0.05% of the overall I/O time. Thus, we suggest that the time overhead of *PhasedRR* is acceptable, even running on the computing-limited platform.

V. Conclusion

This paper has proposed a novel read reclaim scheduling approach, called as *PhasedRR* to support migrating the most frequently accessed data pages in advance, and then decrease negative effects of read disturb. Different from conventional approaches that commonly employ page-level access counting to identify the frequently read data pages, for directing RR scheduling, *PhasedRR* employs working sets to track read frequency on the data pages of the RR block once the block read count approaches our pre-defined soft RR threshold, to reduce the space overhead caused by keeping the counters. After that, we introduce a phased migration scheme for migrating the data pages having varied levels of read hotness in the RR block to another block(s) in different phases, meanwhile the cold data pages are remained in the RR block until the block read count reaches the hard RR threshold.

Our experimental results demonstrate that our method can noticeably improve read responsiveness by 33.6% on average and reduce the erase operations by up to 37.8%, in contrast to state-of-the-art RR scheduling schemes.

Acknowledgment

This work was partially supported by "National Natural Science Foundation of China (No. 62032019)", and "Natural Science Foundation Project of CQ CSTC (No. 2022NSCQ-MSX0789)".

References

[1] Chen X, et al. Reducing flash memory write traffic by exploiting a few MBs of capacitor-powered write buffer inside solid-state drives (SSDs). In *Transactions on Computers* ***(TOC)***, 2018.

[2] Yu T C, et al. CRRC: Coordinating retention errors, read disturb errors and huffman coding on TLC NAND flash memory. In *IEEE Transactions on Dependable and Secure Computing* ***(TDSC)***, 2022.

[3] Shi X, et al. Program error rate-based wear leveling for NAND flash memory. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition* ***(DATE)***, 2018.

[4] Luo Y, et al. Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems* ***(POMACS)***, 2018.

[5] Gao, C, et al. Reprogramming 3D TLC flash memory based solid state drives. In *ACM Transactions on Storage* ***(TOS)***, 2022.

[6] Kim, B. S. et al. Design tradeoffs for SSD reliability. In *Proceedings of 17th USENIX Conference on File and Storage Technologies* ***(FAST)***, 2019.

[7] Kobayashi, A, et al. Investigation of read disturb error in 1Ynm NAND flash memories for system level solution. In *Proceedings of International Reliability Physics Symposium* ***(IRPS)***, 2017.

[8] Lv Y, et al. MGC: Multiple-Gray-Code for 3D NAND Flash based High-Density SSDs. In *Proceedings of International Symposium on High-Performance Computer Architecture* ***(HPCA)***, 2023.

[9] Mielke N, et al. Bit error rate in NAND flash memories. In *International Reliability Physics Symposium* ***(IRPS)***, 2008.

[10] Grupp L M, et al. Characterizing flash memory: Anomalies, observations, and applications. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* ***(MICRO)***, 2009.

[11] Desnoyers, Peter. Empirical evaluation of NAND flash memory performance. In *ACM SIGOPS Operating Systems Review* ***(OSR)***, 2010.

[12] Werner J, et al. Read disturb handling for non-volatile solid state media. In *U.S. Patent Application*, 2014.

[13] Cai Y, et al. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Preceedings of International Conference on Computer Design* ***(ICCD)***, 2012.

[14] Cai Y, et al. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *Proceedings of 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* ***(DSN)***, 2015.

[15] Manning C. Yaffs NAND flash failure mitigation. https://yaffs.net/sites/default/files/downloads/YaffsNandFailureMitigation.pdf.

[16] Ha K, et al. An integrated approach for managing read disturbs in high-density NAND flash memory. *In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* ***(TCAD)***, 2015.

[17] Liu C Y, et al. Read leveling for flash storage systems. In *Proceedings of the 8th ACM International Systems and Storage Conference* ***(SYSTOR)***, 2015.

[18] Wu T C, et al. Flash read disturb management using adaptive cell bit-density with in-place reprogramming. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition* ***(DATE)***, 2018.

[19] Zhao M, et al. Block attribute-aware data reallocation to alleviate read disturb in SSDs. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition* ***(DATE)***, 2021.

[20] Liao, J W, et al. Read Refresh Scheduling and Data Reallocation against Read Disturb in SSDs. In *ACM Transactions on Embedded Computing Systems* ***(TECS)***, 2022.

[21] Zhang G, et al. Cocktail: Mixing data with different characteristics to reduce read reclaims for NAND flash memory. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* ***(TCAD)***, 2022.

[22] Li J, et al. Mitigating negative impacts of read disturb in SSDs. In *ACM Transactions on Design Automation of Electronic Systems* ***(TODAES)***, 2020.

[23] Liu C Y, et al. SOML read: Rethinking the read operation granularity of 3D NAND SSDs. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems* ***(ASPLOS)***, 2019.

[24] Denning P J. The working set model for program behavior. In *Communications of the ACM* ***(CACM)***, 1968.

[25] Zhao K, et al. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *Proceedings of 11th USENIX Conference on File and Storage Technologies* ***(FAST)***, 2013.

[26] Du Y, et al. Adapting layer RBERs variations of 3D flash memories via multi-granularity progressive LDPC reading. In *Proceedings of the 56th Annual Design Automation Conference* ***(DAC)***, 2019.

[27] Lin H Y, et al. Revive bad flash-memory pages by HLC scheme. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* ***(TCAD)***, 2018.

[28] Lee C, et al. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *Proceedings of the 10th ACM International Systems and Storage Conference****(SYSTOR)***, 2017.

[29] Search Engine I/O. http://traces.cs.umass.edu/index.php/Storage/Storage

[30] Narayanan D, et al. Write off-loading: Practical power management for enterprise storage. In *ACM Transactions on Storage (****TOS****)*, 2008.

[31] Li J, et al. An in-depth analysis of cloud block storage workloads in large-scale production, *In Proceedings of IEEE International Symposium on Workload Characterization* ***(ISWC)***, 2020.