READ-MODIFY-WRITE OPTIMIZATION FOR SHINGLED MAGNETIC RECORDING STORAGE SYSTEMS

CHENLIN MA

PhD

The Hong Kong Polytechnic University

2019

THE HONG KONG POLYTECHNIC UNIVERSITY

DEPARTMENT OF COMPUTING

Read-Modify-Write Optimization for Shingled Magnetic Recording Storage Systems

Chenlin MA

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

April 2019

CERTIFICATE OF ORIGINALITY

___________________________________(Signature)

__________Chenlin Ma____________ (Name of Student)

ABSTRACT

Recently, Shingled Magnetic Recording (SMR) disks have been adopted to satisfy the capacity requirement for big data applications. Compared with traditional Hard Disk Drives (HDDs), SMR disks are more cost-effective for its capacity and low cost (i.e., cost-per-gigabyte is competitive). However, SMR disks have poor performance (e.g., low responding time) due to internal unique characteristics (shingled tracks). That is, writing to a certain track may destroy the stored data on the subsequent tracks. To avoid data loss, a read-modify-write (RMW) operation is incurred to (1) read out all the stored data on the subsequent tracks; (2) modify the required data; and (3) write back all the data one-track by one-track sequentially. Such time-consuming RMWs can bring a significant negative effect on the overall system performance and should be avoided as many as possible. In this thesis, we address the RMW issue from several aspects including a decentralized approach without the cache-assistance and two cache optimizations by the integration of NAND flash and SMR disks.

First, we focus on optimizing the shingled magnetic recording storage system through a decentralized approach to get rid of the need of RMW operations. To alleviate the RMW effect, some previous works adopt a centralized over-provisioned persistent cache to temporarily buffer incoming data and migrate the data back to the disk once the cache is full. The persistent cache uses an out-of-place scheme to sequentially log writes on the tracks from outside to inside in an appending mode. In this way, the persistent cache avoids RMWs to some extent by supporting log-structured writes. However, when the persistent cache is used up, the aggregated data will be written/cleaned back to the SMR disk recklessly which still leads to a large number of RMWs. In this thesis, to eliminate the RMW effect, we for the first time propose a decentralized approach called Tiler to manage the SMR disks.

Our basic idea is to separate the whole SMR disk space into individual log-structured autonomous regions (ARs). We propose a two-level mapping scheme to record the mapping between SMR logical addresses and ARs and a three-state space management design to efficiently manage AR spaces. In this way, we can maximize the efficiency of the SMR storage system by eliminating/minimizing RMWs. We have built a trace-driven SMR disk simulator and implemented our proposed Tiler mechanism with this simulator. The experimental results show that Tiler can shorten the overall average response time by 49% and the average cleaning time can be reduced by 25 times.

Second, we propose a new cache management scheme named *Dual-buffer* to effectively manage the persistent cache of SMR disks. There are several challenges to be conquered in order to effectively manage the persistent cache: first, the persistent cache does not distinguish hot/cold data (related to frequently or infrequently updated requests, respectively). Thus, when a cleaning operation is triggered, the hot data may introduce unnecessary writes; second, it also incurs significant overhead by keeping the magnetic read/write heads being routed between the persistent cache at the outer diameter and the native locations at the inner diameter; third, the capacity of the persistent cache is on the scale of several gigabytes. How to effectively manage the persistent cache remains an open problem. In this thesis, we present *Dual-buffer* to solve the above-mentioned challenges. Different from conventional single-buffer-based schemes, *Dual-buffer* partitions the persistent cache into two separate buffers, namely the persistent buffer and the filter buffer, that are used to handle incoming data requests and to hold hot data, respectively. The basic idea is to keep hot data in the filer buffer as long as possible, instead of writing them back to their native locations during a cleaning operation. In this way, cleaning operations only trigger a few RMW operations, thereby alleviating the hot data write-back effect and reducing access latencies in SMR disks. Specifically, to effectively manage the persistent buffer and the filter buffer, we propose a prediction-based dynamic partitioning mechanism to reconfigure the sizes of the persistent buffer and the filter buffer so as to cache hot data as much as possible by adapting to different workloads. We also propose an address mapping scheme based on a B+ tree data structure so the address mapping of the persistent buffer and the filter buffer and the address transition during cleaning operations can be efficiently accomplished. The experimental results show that *Dual-buffer* can improve the access latency by 55.16% on average and reduce the total RMW operations by 98.76% on average.

Third, we study the internals of SMR disks to solve the RMWs issues by integrate NAND flash into the cache optimization. Some previous works devote 1%$\sim$10% of the overall disk space, as an over-provisioned persistent cache to alleviate the RMW effect. By adopting the persistent cache, the performance can be improved to some extent. However, once the persistent cache is full, a cleaning process is triggered to clean back all the aggregated data to SMR disks recklessly which inevitably incurs a large number of RMWs. Therefore, the persistent cache can be the performance bottleneck of the whole SMR system. As the RMWs should be avoided as many as possible, in this thesis, we propose to deploy built-in NAND flash as a cache (namely RMW-F cache) along with the SMR disk and implement a dual-space management scheme that can eliminate the need for RMWs. we propose to distribute the writes that will incur RMWs (if written back) to RMW-F while the other writes are performed in the SMR disk directly. In this way, our design ensures that no RMWs are needed and thus the system performance can be improved. The experimental results show that RMW-F can shorten the overall system average response time by over 79% and improve the cleaning efficiency by approximately 15.6 times.

In summary, we have proposed three main schemes to optimize the SMR storage system including (1) a decentralized approach called Tiler to manage the whole SMR disk space; (2) a cache optimization scheme called *Dual-buffer* to improve the overall performance of the SMR storage system; and (3) an integration of NAND flash as cache (namely RMW-F cache) to eliminate the need of RMWs and accelerate the SMR disk. Some different directions can be explored in the future researches of our works. First, crash recovery is an important issue of drive-managed SMR devices since the mapping are dynamically mapped and the system mainly relies on the address translation to perform reads/writes. How to combine our schemes to effectively perform crash recovery can be a future direction for us to explore. Second, we can combine our Dual-buffer and RMW-F schemes together for key-value stores in the SMR device. How to design the key-value SMR caching system can be an interesting direction in future work. Third, our RMW-F scheme is mainly based on flash-based hardware. We will extend our approach to other emerging non-volatile-memories (NVMs) to further improve the SMR storage system performance.

Keywords: Shingled magnetic recording, cache management, garbage collection, hot/cold data, NAND flash memory.

PUBLICATIONS ARISING FROM THE THESIS

Journal Papers

1. **Chenlin Ma**, Zhaoyan Shen, Yi Wang, Zili Shao, "Alleviating Hot Data Write Back Effect for Shingled Magnetic Recording Storage Systems", Accepted in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2018.

2. Yong Guan, Guohui Wang, **Chenlin Ma**, Renhai Chen, Yi Wang, Zili Shao, "A Block-Level Log-Block Management Scheme for MLC NAND Flash Memory Storage System", IEEE Trans. Computers (TC) 66(9): 1464-1477 (2017).

3. Renhai Chen, Zhaoyan Shen, **Chenlin Ma**, Zili Shao, Yong Guan, "NVMRA: utilizing NVM to improve the random write operations for NAND-flash-based mobile devices.", Softw., Pract. Exper., 46(9): 1263-1284 (2016).

4. **Chenlin Ma**, Zhaoyan Shen, Jihe Wang, Yi Wang, Renhai Chen, Yong Guan, Zili Shao, "Tiler: An Autonomous Region-Based Scheme for SMR Storage", Under Submission in IEEE Trans. Computers (TC), 2019.

Conference Papers

1. **Chenlin Ma**, Zhaoyan Shen, Lei Han, and Zili Shao, "FC: Built-in Flash-cache with fast cleaning for SMR Storage", Accepted in IEEE International Conference on Embedded Software and Systems (ICESS' 19), 2019.

2. Jihe Wang, **Chenlin Ma**, Zhaoyan Shen, Shahher Muhammad, and Zili Shao, "Tiler: An Autonomous Region-Based Scheme with Fast Cleaning for SMR Storage", The 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA' 18), 2018.

3. **Chenlin Ma**, Zhaoyan Shen, Lei Han, and Zili Shao, "RMW-F: A Design of RMW-Free Cache Using Built-in NAND-Flash for SMR storage", Under Submission in International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS' 19), 2019.

ACKNOWLEDGEMENTS

First of all, my sincere gratitude should be expressed to my supervisor, Prof. Zili Shao, who has been watching over me for more than five years. I want to thank him for giving me an opportunity to be a MPhill student at first and helping me to transfer to a Ph.D. student later. I want to thank him for guiding me and never giving up on my research, his patience and profession are my motivation to study. I want to thank him for his valuable time spent in the weekly regular meeting and the discussions we had were something I would remember for my whole life. This thesis would not have been possible without the kind considerations and helps of Prof. Shao.

I also want to thank Prof. Shuai Li for giving me a chance to continue on my Ph.D. study in the last few months. His advice, guidance, and encouragement are valuable treasure during my Ph.D. study. I also express my gratitude to the other members of Prof. Shao's research group - Dr. Yi Wang, Dr. Duo Liu, Dr. Zhiwei Qin, Dr. Renhai Chen, Dr. Zhaoyan Shen, Dr. Lei Han —for the assistance that they provided during my Ph.D. study. I also would like to thank all of my teachers from whom I learned so much during my long journey of acquiring a formal education.

I am grateful to Prof. Bin Xiao for his patience, encouragement, and valuable guidances throughout my Ph.D. study. I appreciate his vast knowledge and skill in many areas and his professional supervision.

I recognize that this thesis would not have been possible without the financial assistance that I received from the Hong Kong Polytechnic University. I thank Prof. Shao and the Department of Computing for offering me travel grants to attend several international conferences. I acknowledge the grant for the Research Student Attachment Program from the Hong Kong Polytechnic University.

Finally, I want to thank my family. Without their endless and selfless love, I would not be able to complete and finish my Ph.D. study smoothly. I want to thank my father and mother for educating me and supporting me in every aspect throughout my Ph.D. study. I want to thank my wife, Yanfang, for taking good care of my son and daughter and not letting me worry about them so that I can devote myself to the study. Thank my family for their understanding and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Shingled Magnetic Recording [88,89,91,95,96] is a non-volatile media that can provide high capacity for next-generation storage devices. By utilizing the overlapping shingle-like structure, SMR storage systems do not require a significant change but can provide two to three times of capacity [22, 26] compared with the conventional hard disk drive. The increased density of SMR disks provides a promising solution to satisfy the capacity requirement of big data applications. However, with the asymmetric sizes of the write-head and read-head, a write operation to one track will destroy the data on several adjacent tracks. This feature makes it challenging for SMR disks to effectively support random writes. To solve this problem, RMW (Read-Modify-Write) operations are introduced to prevent data loss in SMR disks. With RMW operations, all data involved in all tracks will be read out and modified in the external storage space (e.g. RAM space) and then written back to the SMR disk. Since multiple read and write accesses are involved, RMW operations inevitably increase access latencies. Therefore, RMWs become the major performance challenge and should be avoided as much as possible.

To alleviate the RMW effect, some previous works [2, 9, 90] focus on adopting an over-provisioned cache to accelerate the system performance while leaving the underlying SMR disk space unchanged. Therefore, the full optimization potential of the whole SMR device is not exploited well and we advocate to reconsider the detailed design of the management of the SMR disk space. In terms of the cache designs [2, 9], some further optimizations can be done with the following two aspects including (1) distinguishing different data write-back behaviors and cache accordingly; and (2) integration of NAND flash to accelerate performance.

We first advocate reconsidering the drive-managed scheme and for the first time propose a decentralized approach called Tiler to manage the SMR disk space. Our basic idea is to separate the whole SMR disk space into individual log-structured autonomous regions (ARs). In this way, not only the RMW effect can be eliminated but also get rid of a centralized over-provisioned cache. By reconsidering the drive-managed scheme, a variety of new optimization opportunities can become true: (1) the SMR disk space is partitioned into individual log-structured autonomous regions evenly; (2) a two-level mapping scheme is adopted which includes a space-efficient coarse-grained mapping to statically map the SMR logical addresses into ARs at the region-level and a fine-grained dynamic mapping is used to record the mapping between logical-AR-offset and physical-AR-offset at the block level; and (3) a three-state space management design including logging write, lazy write and common cleaning is adopted in managing the SMR space in which: a logging write process handles writes in an AR with an append manner and thus ensures no RMWs are required; when there are no more free blocks, a lazy write process is triggered to locate some schedulable data blocks and postpone the triggering of a common cleaning; a common cleaning is triggered to reclaim the space of an AR when there are no more schedulable data blocks. We have built an SMR simulator and evaluated our proposed scheme with various real-world collected traces to demonstrate the effectiveness of this scheme.

Second, to optimize the circular buffer/cache of the SMR disk, we propose a new cache management scheme called *Dual-buffer* to manage the SMR storage system. Different from conventional single-buffer-based schemes, *Dual-buffer* partitions the persistent cache into two separate buffers, namely the persistent buffer and the filter buffer, that are used to handle incoming data requests and to hold hot data, respectively. The basic idea is to keep hot data in the filer buffer as long as possible, instead of writing them back to their native locations during a cleaning operation. In this way, cleaning operations only trigger a few RMW operations, thereby alleviating the hot data write-back effect and reducing access latencies in SMR disks. Specifically, to effectively manage the persistent buffer and the filter buffer, we propose a prediction-based dynamic partitioning mechanism to reconfigure the sizes of the persistent buffer and the filter buffer so as to cache hot data as much as possible by adapting to different workloads. We also propose an address mapping scheme based on a B+ tree data structure so the address mapping of the persistent buffer and the filter buffer and the address transition during cleaning operations can be efficiently accomplished. We have implemented the proposed scheme inside the embedded controller of our SMR simulator and evaluated the effectiveness of the scheme with different traces.

Third, in order to eliminate the RMW effect and to accelerate the SMR storage system, we propose to integrate a promising cost-effect NAND flash as a persistent cache (namely RMW-F cache). Specifically, for the writes to SMR disks, some writes will incur RMWs while the other writes will incur no RMWs. Therefore, we propose to distribute the writes that will incur RMWs (if written back) to the flash cache while the other writes are performed in the SMR disk directly. In this way, our design ensures that no RMWs are needed and thus the system performance can be improved. Three challenges have been tackled including: (1) a new internal architecture for both flash-cache and SMR disks is needed to co-manage the system; (2) how to map the SMR logical spaces into the NAND flash physical spaces; (3) how to perform cleaning when the available spaces in the cache becomes low. We have built a trace-driven flash and SMR simulator and implemented our scheme with this simulator. The performance of the system is evaluated with collected traces from Microsoft Research Cambridge [1].

1.1 Related Work

In this section, we briefly discuss previous approaches to optimizing the SMR storage system. In previous studies, work has been done in three main domains: (I) device management optimization, (II) cache management optimization, and (III) integration of NAND flash as cache. We briefly describe these techniques and present detailed comparisons with representative techniques in the respective chapters.

1.1.1 Device Management Optimization

For the device management optimization of SMR disk, log-structured writes have been widely adopted to elminate the needs of RMWs. Log-structured write naturally coincides with the unidirectional writing of SMR head, thus, the data layout on disk is researched in [5,6] to amplify the logging benefit.+ HiSMRfs [36] designs and implements a file system that is suitable for SMR drives. Data and meta data are sepated and managed independently. The work in [90] partitions the whole disk into multiple zones and manages the SMR disk in zone-based API in a host-aware mode. Since accurately modeling a drive-managed SMR disk is challenging, the work in [71] presents a simulation model to evaluate drive-managed SMR disks. Algorithms can be teested and implemented within the simualtor of [71] and therefore, some device management optimizations can be evaluated. A shingled-aware persistent cache management scheme has been implemented in [98] to frist merges cached updated by flashing writes that can be safely written to the SMR disk and then reclaim more cache spaces by another cache merging process.

1.1.2 Cache Management Optimization

The work in [95] proposes a virtual persistent cache design to reduce the long latency of host-aware SMR drives. The idea is to involve the host to adaptively reshape the access patterns to SMR drives as as to avoid the occurrences of long latencies. Skylight [2] performs reverse engineering to figure out the internal managerial operations of SMR devices. Specifically, Skylight manages the shingled persistent cache by implementing an indirection system to eliminate RMW operations [2]. The work in [9] proposes to manage the SMR disk by using an S-block structure and presents a workable dynamic mapping to manage the persistent cache of SMR disk. A hybrid wave-like shingled recording disk-cache is designed to improve both the performance and the capacity of a shingled-write-disk [54].

1.1.3 Integration of NAND Flash as Cache

FlashTier [69] proposes a system architecture built upon flash-based cache and designs an interface for caching. The work in [40] can improve the NAND flash based disk caches by separating read and write regions and improve reliability by a programmable flash memory controller. I-CASH [97] adopts SSD to store seldom changed and mostly read data and uses HDD to store logs, the idea is to utilize the high read performance of SSDs. SMRC [94] proposes to filter out sequential writes and non-sequential writes and then direct non-sequential writes to the SSD as a cache layer. The work in SMRC mainly focuses on optimizing the cache performance of host-aware SMR drives. Some [81] proposes to use SSD as the first-level cache of the SMR device, that is, all incoming writes will be performed in the SSD first. The authors in [81] is able to reduce the write amplification factor to some extent by restricting the write range of logical block addresses (LBAs) of zones. HS-BAS [92] also uses SSD as the first-level write buffer cache for SMR devices and focuses on reducing the cleaning/collection cost by adopting different policies. An application aware hybrid storage system named (Apas), consists of SSDs and SMR disks, has taken the application characteristics into account to mitigate the write amplification problem. A hrbrid wave-like shingled recording disk system (HWSR) proposes to use SSD as a disk cache to mainly improve random read performance rather than write performance [55, 80].

1.2 The Unified Research Framework

In this section, we present the unified research framework for the proposed techniques. A sketch of our research framework is given in Figure 1.1.

In this thesis, our works are mainly implemented inside the SMR device and run as the software layers. As shown in Figure 1.1, in Chapter 3, we propose a decentralized approach to manage the whole SMR disk independently and separately in an autonomous way. In Chapter 4, we propose a new cache management scheme to improve the overall system performance of the SMR device. The underlying storage media of our first two works is mainly SMR disk as shown in Figure 1.1. In Chapter 5, we distinguish data according to

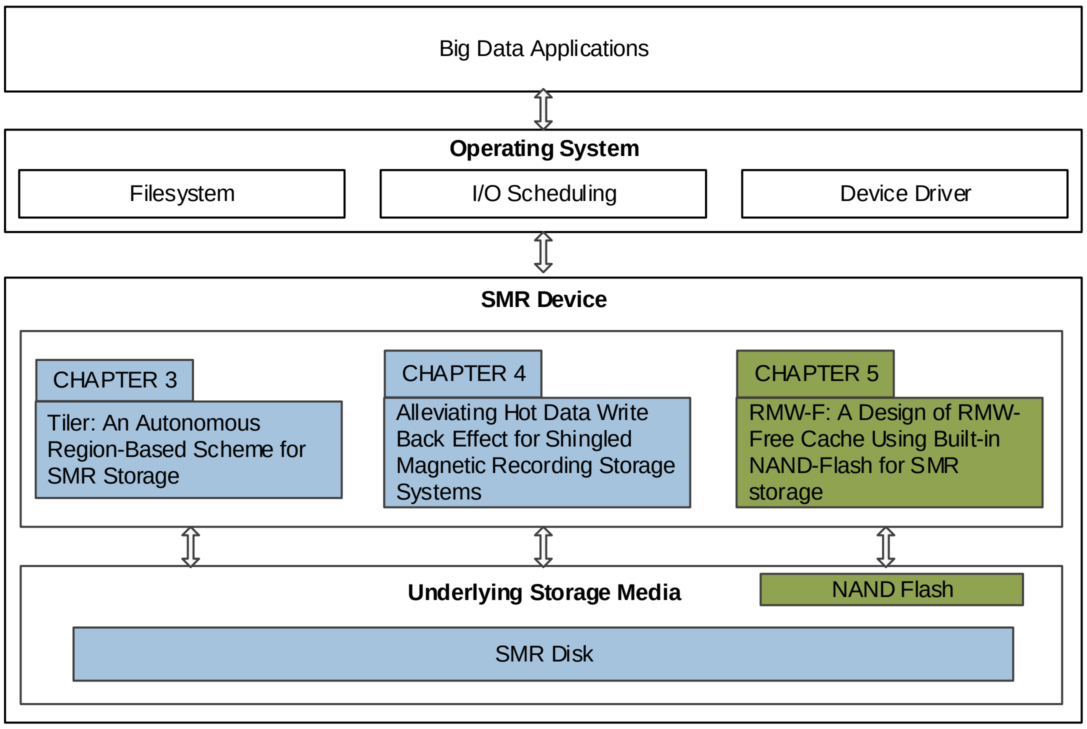

Big Data Applications
Operating System
Filesystem
I/O Scheduling
Device Driver
SMR Device
CHAPTER 3
Tiler: An Autonomous Region-Based Scheme for SMR Storage
CHAPTER 4
Alleviating Hot Data Write Back Effect for Shingled Magnetic Recording Storage Systems
CHAPTER 5
RMW-F: A Design of RMW-Free Cache Using Built-in NAND-Flash for SMR storage
Underlying Storage Media
NAND Flash
SMR Disk

Figure 1.1: The Unified Research Framework.

their write-back behavior and integrate NAND flash as part of the underlying storage media to accelerate the SMR device.

1.3 Contributions

The contributions of this thesis are summarized as follows.

- We propose a decentralized approach which separates SMR disks into individual log-structured ARs to eliminate the time-consuming RMW effect. A two-level mapping scheme is proposed to record the mapping between SMR logical addresses and ARs. A three-state space management design to efficiently manage AR spaces.

- We present a novel cache management scheme by which the persistent cache is partitioned into a persistent buffer and a filter buffer and hot data can be cached in the filter buffer as much as possible. We propose a prediction-based dynamic partitioning mechanism so hot data can be cached as much as possible by reconfiguring the sizes of the two buffers in the persistent cache. An address mapping scheme is developed to achieve effective and efficient space management for the persistent cache.

- We propose a new architecture within which adopts built-in NAND flash as the RMW-free cache of the SMR system. Some new modules are implemented to accelerate the management efficiency of the hybrid storage system. A hybrid two-level mapping scheme is proposed to handle the address translations between the SMR logical addresses and the NAND flash physical addresses. We improve the cleaning efficiency by a heuristic model that takes both write-back cost and data popularity into account.
- We have built an SMR and flash simulator and evaluated our proposed techniques with various traces and the experimental results prove the effectiveness of the proposed techniques.

1.4 Thesis Organization

The rest of this thesis is organized as follows.

- In Chapter 2, we introduce some background information about the SMR disk.
- In Chapter 3, we for the first time propose a decentralized approach to partition the whole SMR disk space into autonomous self-managed regions evenly. Optimizations can be done within the SMR device.
- In Chapter 4, to optimize the persistent cache of the SMR disk, we propose a new cache management scheme called *Dual-buffer* to manage the SMR storage system. The basic idea is to keep hot data in the cache as long as possible, instead of writing them back to their native locations during a cleaning operation.
- In Chapter 5, in order to eliminate the RMW effect and to accelerate the SMR storage system, we propose to integrate a promising cost-effect NAND flash as a persistent cache (namely RMW-F cache).
- In Chapter 6, we present our conclusions and propose possible future directions of research arising from this work.

CHAPTER 2
BACKGROUND

In this chapter, we will introduce the detailed characteristics of SMR disks.

SMR disks can achieve high density with minimal changes to the functionality of hard disk drives. Similar to conventional disk drives, an SMR disk also consists of multiple tracks. Each track includes multiple blocks (or sectors), and a block is the basic read/write unit in an SMR disk. The major difference between SMR disks and conventional hard disks is the adoption of the shingle-like structure (as shown in Figure 2.1(b)). An SMR disk overlaps one track onto another to provide more condensed space. This new infrastructure does not affect read operations since the overlapped track is adequate to cater the read operations with a very small magnetic read head.

However, the magnetic write head is typically 3-8 times larger than the read head [50]. This wider and stronger write head will cover a large number of physical tracks. Then, a write operation to one track will destroy data on the adjacent tracks.

In Figure 2.1(c), the write-head (i.e., W_head) width is k (=3) times as large as the

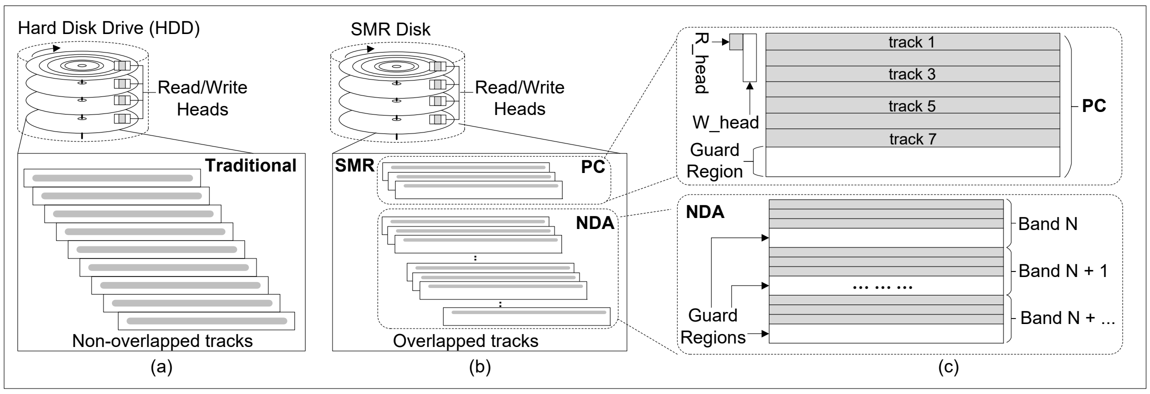


Figure 2.1: The layout comparison between HDD and SMR: (a) the conventional HDD, (b) the SMR, and (c) the geometry tracks of the SMR.

read-head (i.e., R_head) width. Therefore, writing to track 1 will damage the data stored in the subsequent two tracks (i.e., track 2 and track 3). This unique property of the SMR disk limits its ability to perform random writes and makes in-place updates more difficult. In order to update the data on track 1, the data on track 2 and track 3 should be firstly read out and written back once the update is done. Two tracks (i.e., k-1) containing no data are provided as a guard region. Rewriting the data tracks before the guard region (e.g., track 7) will not affect any tracks following the guard region.

In Figures 2.1(b) and 2.1(c), the SMR disk consists of the native data area and the persistent cache. Usually, 1%∼10% of the SMR disk space is devoted to constructing the persistent cache [2]. The native data area is placed at the inner diameter, and this area will be used to store the native data. The persistent cache is located at the outer diameter and is served as the write buffer of native data. The persistent cache contains 7 data tracks followed by a guard region with a width of two (i.e., k-1) tracks. The native data area is partitioned into multiple bands (each band size is 15∼40MB [2]), separated by the guard regions.

CHAPTER 3
TILER: AN AUTONOMOUS REGION-BASED SCHEME FOR SMR STORAGE

3.1 Introduction

SMR [17,31,42,67,77,88,89,91,95,96] has been provided as a promising solution to satisfy the capacity requirement of big data applications. Compared with conventional HDDs, SMR disks benefit in higher density and lower cost. However, SMR disks [8,19,61,63,71] also suffer from lower responding time due to their intrinsic overlapped-track limitation. That is, writing to a track leads to damage to the original data stored on the subsequent tracks [29,33,46]. To prevent data loss, a time-consuming RMW is performed to firstly copy out the involved data and then to write them back after the modification is done. In this chapter, we for the first time propose to divide the whole SMR space into decentralized autonomous regions to eliminate the need for RMWs.

To alleviate the RMW effect, some previous works [2,9,90] adopt a centralized over-provisioned persistent cache to temporarily buffer incoming data and migrate the data back to the disk once the cache is full. The persistent cache uses an out-of-place scheme to sequentially log writes on the tracks from outside to inside in an appending mode. In this way, the persistent cache avoids RMWs by supporting log-structured writes. A dynamic mapping table is used to record the addresses of stored data in the persistent cache. However, when the cache is used up, the aggregated data will be written/cleaned back to the SMR disk recklessly which still leads to a large number of RMWs.

To eliminate the RMW effect, In this chapter, we for the first time propose a decentralized approach called Tiler to manage the SMR disks. Our basic idea is to separate the whole SMR disk space into individual log-structured ARs. Writes to each AR are done in an appending mode along with the shingled direction which guarantees no RMWs are incurred. However, the proposed approach brings three new challenges: (1) how to partition the SMR disk space with ARs; (2) a mapping table is required to record the mapping between SMR logical addresses and AR blocks; (3) how to manage the AR space including space allocations and reclamation (e.g., cleanings).

To handle the first challenge, we propose an AR design to evenly partition the whole SMR disk space. Unlike the centralized approach that aggregates data in the persistent cache, data are written to their corresponding ARs in Tiler in a decentralized manner.

To address the second challenge, we propose a space-efficient coarse-grained mapping to statically map the SMR logical addresses into ARs at the region level. Within each AR, a fine-grained dynamic mapping is used to record the mapping between logical-AR-offset and physical-AR-offset at the block level. Specially, we store the block level dynamic mapping on a track where in-place update is supported. Each AR's dynamic mapping will be loaded into memory space on-demand for fast-accessing and memory-saving. Note that the static mapping requires no memory and the dynamic mapping consumes little memory space. Thus, our two-level mapping (including region-level and block-level) is space-efficient.

To conquer the third challenge, Tiler adopts a three-state space management design including logging write, lazy write and common cleaning. A logging write process handles writes in an AR with an append manner and thus ensures no RMWs are required. When there are no more free blocks, a lazy write process is triggered to locate some schedulable data blocks and postpone the triggering of a common cleaning. A common cleaning is triggered to reclaim the space of an AR when there are no more schedulable data blocks.

We have built a trace-driven SMR disk simulator and implemented our proposed Tiler mechanism with this simulator. We have evaluated Tiler with several real-world traces collected from data centers organized by Microsoft Research Cambridge [1]. The experimental results show that compared with Skylight [2], Tiler shortens the overall average response time by 49%. The average cleaning time can be reduced by 25X.

The main contributions of this chapter are summarized as follows:

- We propose a decentralized approach which separates SMR disks into individual log-structured ARs to eliminate the time-consuming RMW effect.
- We propose a two-level mapping scheme to record the mapping between SMR logical addresses and ARs. We propose a three-state space management design to efficiently manage AR spaces.
- We have built an SMR simulator and evaluated our proposed Tiler with various traces.

The rest of this chapter is organized as follows. Section 3.2 gives the motivation for this work. Section 3.3 describes the design and implementation details of our proposed scheme. Section 3.4 presents the performance analysis of Tiler. Experimental results are provided in Section 3.5. Section 3.7 concludes the chapter.

3.2 Motivation

By accommodating update/write operations in an appending mode along with the rotate direction, the centralized the persistent cache reduces the negative effect of RMWs to some extent. However, when the persistent cache is full, the aggregated data within the cache will be written back into the native data area in a cleaning process to reclaim the space of the persistent cache. Writing the data back to their native locations (i.e., a certain block of a track) will inevitably incur RMW operations and therefore, all the stored data on the subsequent tracks will be affected. Thus, this may lead to two issues: First, writing back data from the persistent cache to the native data area will keep the read/write-head busy with shifting back and forth and these head movements can significantly degrade system performance; Second, the large number of incurred RMW operations are time-consuming since all the affected data should be copied out and written back.

As a motivational example, we have conducted some preliminary experiments on three of the collected traces and the results are shown in Figures 3.1a and 3.1b. The experimental results are collected from the environment described in Section 3.5 and the detailed characteristics of the traces can be found in Table 5.1.

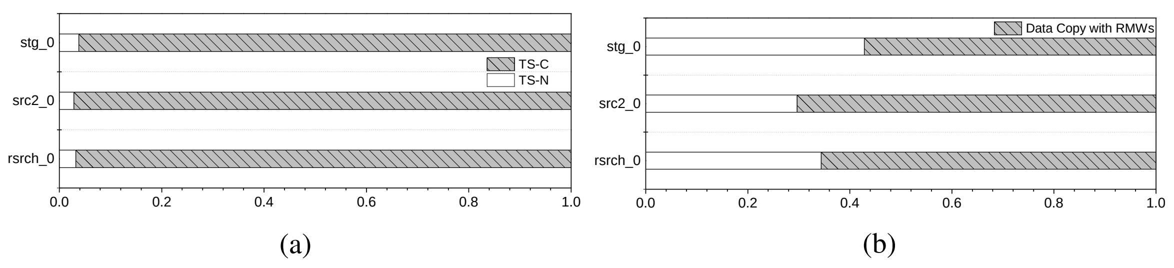


Figure 3.1: Preliminary Experiments. (a) A preliminary experiment to show the proportion of track-seek count in cleanings (TS-C) and track-seek count in the normal state (TS-N). (b) A preliminary experiment to show the proportion of data copy with RMWs.

In order to perform a quantitative analysis on the negative effect brought by RMWs, we count the total volume of data copy with RMWs during the cleaning process and compare that with the total amount of data written to SMR. As shown in Figure 3.1b, the proportion of data copy with RMWs contributes to over 70% of the total amount of data written to SMR. Among the three evaluated traces, the proportion of data copy has a contribution of 64% on average. Therefore, RMWs are significant overheads that we should try to alleviate.

On operating data in SMR disks, the read/write head needs to be moved from the source track to the destiny track. This positioning is called track seek and the delay of track seek dominates. Therefore, to measure the negative effect of a large number of read/write head movements, we evaluate the total track-seek count involved in the cleaning (denoted as TS-C) and track-seek count without cleaning (denoted as TS-N). The corresponding proportion of both TS-C and TS-N among three traces are calculated. As shown in Figure 3.1a, TS-C contributes to over 96% of the total track-seeks on average. Therefore, by reducing the number of read/write head movements (i.e., track-seeks), the system performance can potentially be improved.

3.3 Tiler: a decentralized approach for SMR disks

In this section, we first present an overview of our design of ARs in Section 3.3.1. Then, we describe our two-level mapping including static mapping and dynamic mapping in Sections 3.3.2-3.3.2, respectively. Finally, we discuss the space management in Tiler which involves three processes including logging write, lazy write and cleaning in Sections 3.3.3-

4.3.2.

3.3.1 Overview of ARs Design

In this section, the overview of ARs design will mainly introduce the system layout of Tiler and the functionality of each system components (i.e., zone, AR). Each zone on each disk surface is partitioned into different individual ARs. An AR consists of three major components including a *data region*, a *map region*, and a *guard region*. The detailed illustrations are as follows:

As Figure 3.2 depicts, an SMR disk has multiple platters (two surfaces on each) and in Tiler, we partition each surface space into N_{Zone} number of zones as shown in Figure 3.2 (a). The capacity of each track within a zone is identical (i.e., each track consists of an identical number of blocks) according to Zone-Bit-Recording (ZBR) [83], As shown in Figure 3.2 (b), each zone is further partitioned into different individual ARs along with the rotational direction. Since the number of blocks on each track within each zone varies due to ZBR, the number of ARs within each zone is different.

As mentioned above, an AR has three regions with different functionality. As shown in Figure 3.2, in a *data region*, there is M number of data tracks along with the shingled direction(i.e., Track_0 - Track_{M-1}) which are used to store data and each track in an AR consists of **S** number of data blocks along with the rotational direction. Writes/updates into an AR are served in an append mode along the rotational direction in the *data region*.

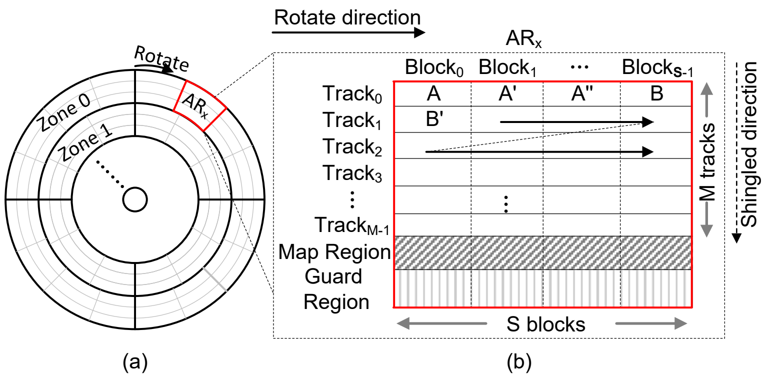


Figure 3.2: System layout of Tiler. (a) Surface diagram of a disk platter in Tiler. (b) Layout overview of an Autonomous Region (AR).

Note that the number of tracks within each zone is identical and the number of blocks on each track varies. Therefore, the parameter M indicating the number of tracks is fixed while the parameter S is adjustable. A larger S indicates that each track within an AR can allocate more blocks to serve data requests within one rotational operation. The sequential-accessing within one rotational operation can potentially reduce the number of the track-seek counts and therefore improve system performance. However, more data are aggregated in each AR with a larger S, it may take longer to clean up an AR. Thus, the worst case response time can be enlarged as S increases. Detailed analysis for how to determine S is introduced in Section 3.4 and evaluations are included in Section 3.5.3.

The *map region* consists of k tracks (e.g., three tracks). We use the last track (called **map track**) to store the mapping information and keep the first two tracks empty. A *guard region* (containing two tracks store no data) follows the *map region*. Therefore, the **map track** supports in-place updates since writes to this track will not affect others. Note that each zone is separated by the *guard region*s.

3.3.2 Address Mapping in Tiler

Based on the above-mentioned design, the whole SMR disk space is partitioned into individual ARs, therefore, the second challenging issue is how the data associated with a logical block address can be mapped into its corresponding AR. To address this issue, Tiler adopts a two-level mapping scheme to manage the translation information between SMR logical block addresses (LBAs) and AR physical block addresses (PBAs). The two-level mapping includes a **static mapping** at the region-level to statically map the LBA into a corresponding AR and a **dynamic mapping** at the block level to locate the allocated PBA.

• Static mapping

An LBA is at a block/sector level while AR is at region-level, therefore, to map a given LBA into its corresponding AR, we adopt a static mapping design. The range of both logical block addresses and that of SMR physical block address are equal. For the SMR physical block addresses, we know the exact number of ARs as well as address ranges within

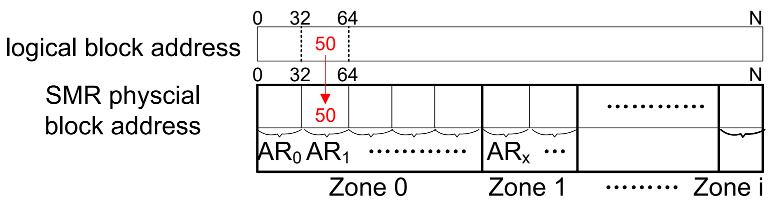


Figure 3.3: An illustrative example of static mapping.

each zone. Therefore, given an LBA, knowing the range of an AR, we can determine which AR the LBA should be mapped into.

As an example in Figure 3.3, given an LBA=50, since the addresses of AR_1 is from 32 to 63, we can determine that LBA=50 should be mapped into AR_1 of $Zone_0$. Since the mapping between an LBA and the corresponding AR is fixed and can be calculated logically, no memory overhead is needed to maintain this static mapping. Note that the offset of 50 from 32 is 18, therefore, the logical block offset (L-ARO) of LBA=50 within AR_1 is 18. Let the logical AR id and the logical block offset within an AR be L-AR and L-ARO, respectively. By adopting a static mapping, given an LBA, we can directly calculate the corresponding L-AR (e.g., AR_1) and L-ARO (e.g., 18).

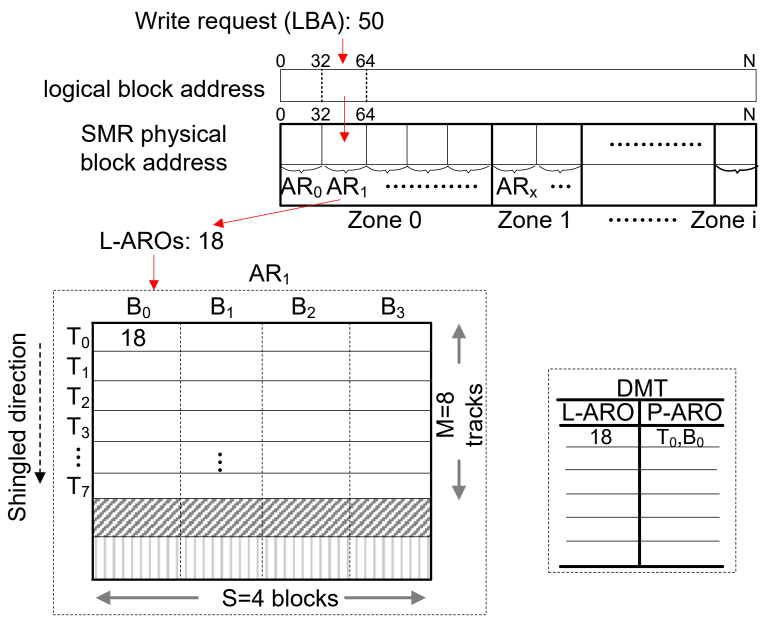


Figure 3.4: An illustrative example of dynamic mapping and the format of the dynamic-mapping-table (DMT).

• Dynamic mapping

The above mentioned static mapping addresses the issue of how the data associated with an LBA can be mapped into the corresponding AR. Since writes/updates are served in an append mode along the rotational direction in the data region of an AR, the mappings between an LBA and an allocated PBA is dynamically changing. Therefore, the next issue is how LBAs are mapped within an AR. Note that a PBA is represented by the physical block offset within an AR (denoted as P-ARO).

Given an LBA, the corresponding L-ARO of an AR is obtained by the above mentioned static mapping. To serve a write/update request in the AR, an available P-ARO is allocated. Therefore, the L-ARO is dynamically mapped to a P-ARO within an AR. We implement a dynamic mapping called the *dynamic-mapping-table* (DMT) in our design.

Note that the mentioned map track in the *map region* supports in-place update. Therefore, the corresponding DMT of each AR is stored on its own map track. The DMT is loaded into memory for fast accessing at runtime and will be written back to the map track periodically.

An example is shown in Figure 3.4 to show how an LBA is mapped within an AR. For illustration, let M and S be 8 and 4, respectively. Thus, there are 8 data tracks along with the shingled direction and 4 blocks on each along with the rotational direction (i.e., 32 data blocks in total). For a write request with LBA=50, the L-ARO is 18 (i.e., offset from 32) in AR_1 of $Zone_0$. The first available data block (P-ARO in the form of track T_0, block B_0) is allocated to serve the request. Thus, the mapping between L-ARO=18 and P-ARO (T_0, B_0) is recorded in DMT.

A read request (e.g., read(50)) can now be handled to access the required data by looking up the DMT. More details about how a read request is served are shown in Algorithm 1.

Algorithm 1: Tiler Read

```
Input: LBA.
Output: The data stored in a P-ARO.
1: Obtain the corresponding L-AR and L-ARO
   by the static mapping.
2: Look up the L-ARO in the corresponding DMT of L-AR.
   Retrieve the P-ARO.
   Read data from the P-ARO.
3: Return the obtained data.
```

3.3.3 Tiler Space Management

The third challenging issue to be addressed is how space is managed within an AR. The space management of Tiler consists of the following three processes: 1) a logging-write process, which ensures no RMWs are needed when handling writes within an AR; 2) a lazy-write process, which keeps locating some schedulable blocks for future writes and acts as an intermediate process to effectively postpone triggering a common cleaning process; 3) a common cleaning process, which is used to reclaim the invalidated spaces within an AR.

As shown in Figure 3.5 (a), when an AR is full during a logging write process (i.e., no more free spaces), a common cleaning should be triggered to reclaim the invalidated space to make room for incoming writes. However, we place a lazy write process between a logging write process and a common cleaning process as shown in Figure 3.5 (b) that can effectively reduce the triggering rate of a cleaning by postponing the cleaning. The details about the three processes are discussed as follows:

• Logging write

During a logging write process, to handle write requests, Tiler will keep writing to a

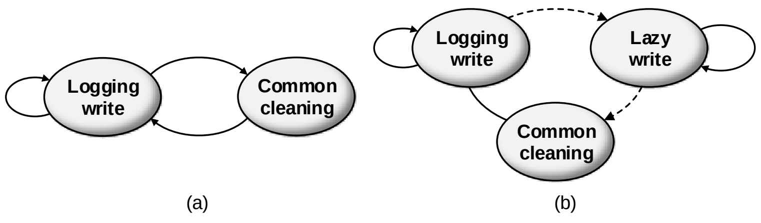


Figure 3.5: The AR state machines for (a) a two-states space management design (b) a three-state space management design adopted in Tiler.

next available free data block on each track along with the rotation direction in an appending mode. When a track is full, the adjacent subsequent track will be used to serve incoming write requests. Following this allocation order, no RMWs are needed during the logging write process.

An example is shown in Figure 3.6. The write requests sequence are handled accordingly and the corresponding DMT is updated correspondingly as shown in Figure 3.6 (a). For instance, as track T_0 of AR_1 is full, the first block B_0 on track T_1 is allocated and the associated mapping is recorded in the DMT. Figure 3.6 (b) illustrates how an update request with LBA=32 is handled. Since there are still some free blocks that can be allocated on track T_1, Tiler allocates the next available free block on track T_1 which is (T_1, B_1) to serve the update request with L-ARO=0 (by calculation). In this way, the corresponding entry of DMT is updated accordingly. Note that the update request will invalidate the previous data block.

Eventually, there will be no more free blocks allocated, then a lazy write process will be triggered. We will introduce the details about the lazy write next.

• Lazy Write

The lazy write process is SMR-aware since it exploits the asymmetric read/write-

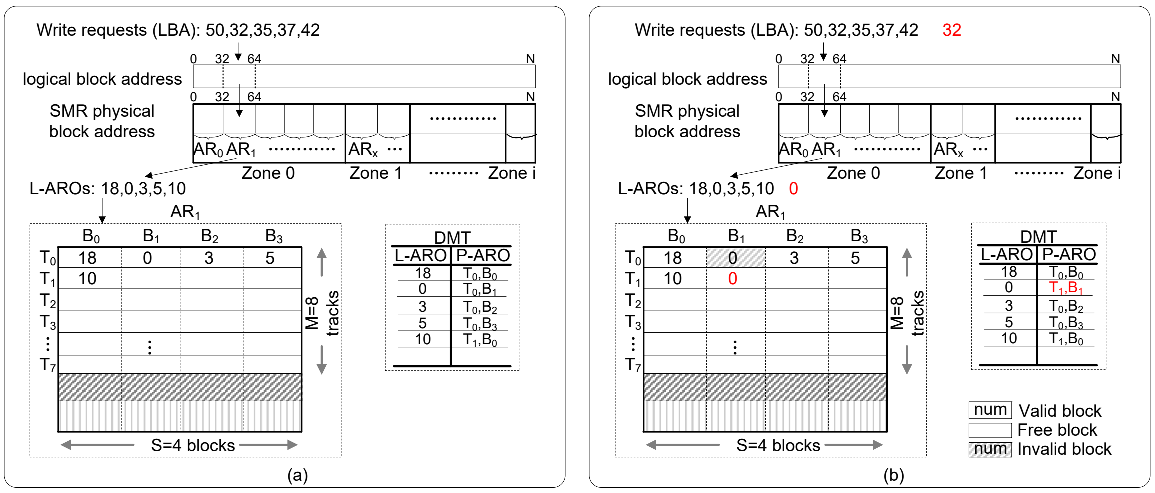


Figure 3.6: An example to illustrate how an update request is served in logging write process (a) before the update request is processed (b) after the update request is processed.

head width characteristic of SMR disks. For illustration, let k (the ratio of the write-head width over the read-head width) be three. With SMR-aware, if there are three adjacent invalidated blocks along with the shingled direction, writing to the first one will not introduce data loss since the two subsequent data blocks have been invalidated already.

A partial diagram (i.e., not including all the tracks) of an AR is shown in Figure 3.7, where three blocks including (T_1, B_0), (T_1, B_3) and (T_2, B_3) are schedulable (i.e., can be allocated to serve incoming write requests) since writes to these blocks will not cause data loss. Note that schedulable data blocks may be generated by invalidating blocks. For example, when the block on (T_2, B_1) is invalidated, two more schedulable blocks including (T_0, B_1) and (T_1, B_1) can be regarded as schedulable. A common cleaning will not be triggered until no more schedulable blocks can be found in the lazy write process. The effectiveness of the lazy write will be demonstrated in Section 3.5.2.

• **Cleaning**

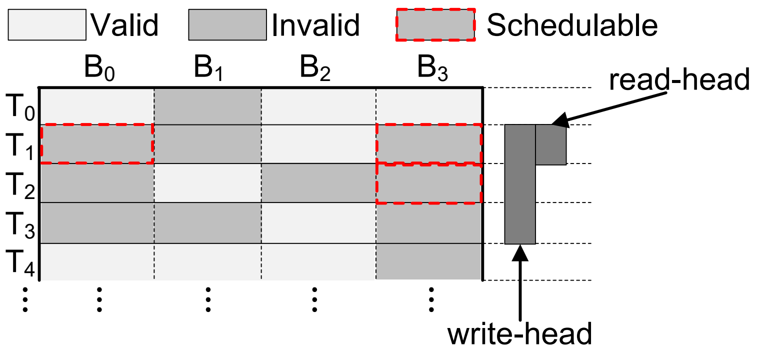


Figure 3.7: The diagram of schedulable blocks when an AR is full. S=4 and write-head-width/read-head-width=3.

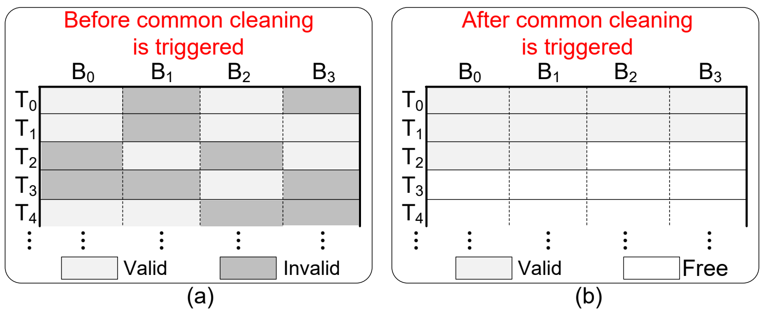

Before common cleaning is triggered
B0
B1
B2
B3
T0
T1
T2
T3
T4
Valid
Invalid
(a)
After common cleaning is triggered
B0
B1
B2
B3
T0
T1
T2
T3
T4
Valid
Free
(b)

Figure 3.8: An example of a cleaning process. (a) Before a cleaning process is triggered. (b) After a cleaning process is triggered.

When there are no more schedulable data blocks (i.e., three adjacent invalidated blocks) within an AR, a cleaning process is triggered to reclaim the invalidated spaces. By looking up the DMT, Tiler can determine the locations of valid data blocks (e.g., blocks in gray color as shown in Figure 3.8). With a common cleaning, the associated data are copied out and then rearranged back into the AR as shown in Figure 3.8 (b). Since the rearrangement is along with the shingled direction (similar to writes in the logging write process), no RMWs are incurred in the cleaning process. As there will be some free data blocks after a cleaning process, Tiler can enter into logging write process to continue on serving new write requests. We describe the detailed cleaning process in Algorithm 2 below.

Algorithm 2: Cleaning

```
Input: DMT
Output: AR (cleaned) and new DMT.
 1: for each entry (L-ARO→P-ARO) in DMT.
    Read out the corresponding valid data
    and buffer it into the memory space. do
 2:     Write/append the valid data one after another
        into the AR.
        Update the corresponding entry in the DMT.
 3: end for
```

The three above mentioned processes related to the space management of Tiler are depicted in algorithm 3 for write operations in Tiler.

Algorithm 3: Tiler Write

```
Input: LBA and the data
Output: Write data to the allocated PBA.
 1: if Logging write then
 2:     Tiler allocator allocates a next free data block as the P-ARO.
        Write data to this P-ARO and update the DMT.
 3:     Return.
 4: end if
 5: if Lazy write then
 6:     A schedulable data block serves as the PBA.
        Write data to this P-ARO and update the DMT.
 7:     Return.
 8: else
 9:     Trigger a common cleaning.
10: end if
11: Return.
```

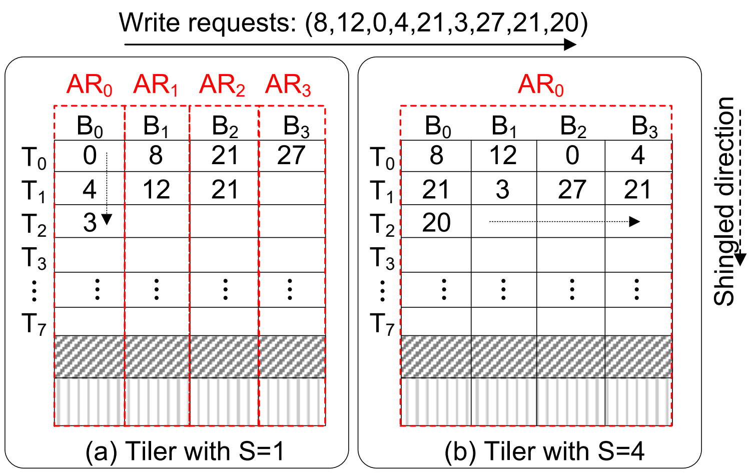


Figure 3.9: An example of different data placement in (a) Tiler with S=1 and (b) Tiler with S=4.

3.4 Performance Analysis

In this section, we will first discuss the data placement with different configurations in Tiler in Section 3.4.1. For data access patterns with different ARs, the corresponding performance modeling will be given in Section 3.4.2. Finally, performance overhead will be analyzed in Section 3.4.3.

3.4.1 Data placement with different ARs configurations

With different configurations, the data placement will be different and the corresponding data access pattern may vary. As shown in Figure 3.2, within a data region of an AR, there are M number of data tracks along with the shingled direction and S number of data blocks along with the rotate direction on each track. M is a fix value which indicates an identical number of data tracks in every zone. The parameter S is a tunable and configurable parameter of AR in our design. A larger S indicates that each track within an AR can allocate more blocks to serve data requests within one rotate operation. The sequential-accessing within one rotate operation can potentially reduce the number of track-seek count and therefore improve system performance.

We show an example of how the data are mapped under two configurations of S=1 and S=4 in Figure 3.9. Tiler with S=1 is shown in Figure 3.9 (a), there are four ARs including AR_0, AR_1, AR_2 and AR_3 with addresses range of [0,7], [8,15], [16,23] and [24,31], respectively. Since the requests with the LBAs including 0,4,3 are within the range [0,7], they are mapped into AR$_0$ in an appending mode. Similarly, other writes are mapped within their ARs accordingly.

Tiler with S=4 is shown in Figure 3.9 (b) and the AR$_0$ has an address range of [0,31]. Therefore, all write requests are appended along with the rotate direction within the AR.

3.4.2 Performance modeling

In this section, we analyze the system response time in Tiler to explore the effect of parameter S on the system performance. The accessing of AR can be classified into two major states including normal state (i.e., without triggering a cleaning process) and cleaning state.

• **Normal state.** For the normal state, no cleaning process occurs when read/write requests are handled. Symbols are listed below.

T$_s$	The avg_time to seek the required track
T$_r$	The avg_time to rotate to the required block
T$_t$	The avg_time to transfer data
S	Each track consists of S number of blocks
M	The number of data tracks within an AR

As shown in Figure 3.2 (b), M $\times$ S blocks are sequentially accessed. Therefore, the average response time in the normal state can be modeled by the following equation:

$$T_{avg_N} = \frac{(M-1) \times T_s + M \times (\mathbf{S}-1) \times T_r + T_t}{M \times \mathbf{S}} \tag{3.1}$$

Since M is a fixed value, Equation 3.1 indicates that the average response time T$_{avg_N}$ is negatively correlated with parameter S.

• **Cleaning state.** As shown in Figure 3.8, a cleaning process will detect the valid data blocks within the AR and rearranged them into the AR sequentially. Therefore, the time consumed is associated with the number of valid data blocks. Let N$_V$ be the total number of valid data blocks to be rearranged. The average response time in the clean state can be modeled by the following equation:

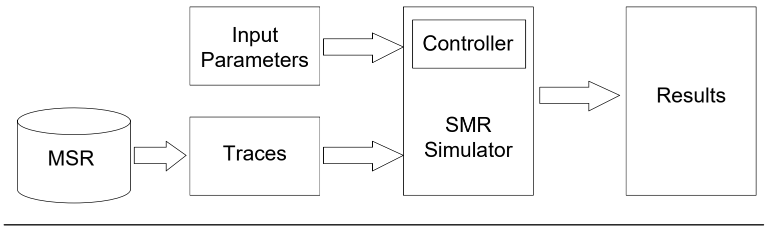


Figure 3.10: The framework of the simulation platform.

$$T_{avg_C} = \frac{(M + \frac{N_V}{\mathbf{S}}) \times T_s + 2N_V \times (T_r + T_t)}{N_V} \tag{3.2}$$

As shown in equation 3.2, the performance overhead of a cleaning is mainly affected by the total number of valid data blocks (i.e., N_V). On one hand, enlarging parameter S will potentially aggregate more valid data blocks compared with a smaller S. On the other hand, with a larger S, there is more opportunity for the lazy write process to find more schedulable data blocks that can postpone triggering of a cleaning. As different workloads can have different behaviors, we will evaluate the effect of parameter S on cleaning efficiency with different traces in the evaluation part.

3.4.3 Overhead analysis

Since Tiler uses a dynamic mapping table to record the translation information between LBAs and physical AR blocks, and stores the mapping stable in both memory and disk. So, Tiler introduces the memory and disk space overheads. Besides, the AR cleaning process will include valid data copy operations and thus incur write amplification. However, the write amplification is acceptable since Tiler eliminates all the RMW effect, which is exhibited in our experiments.

On the other hand, for SMR read operations, in the worst-case, when the required DMT is not presented in RAM, an additional map track read is required to first load the DMT before reading the data. This process may sacrifice some read performance.

3.5 Evaluation

To evaluate Tiler, we conduct a series of experiments. For comparison, we have implemented Skylight in [2] as a baseline. Tiler is implemented with four different AR configurations, namely, Tiler with S=1, S=2, S=4 and S=8, respectively.

In this section, we first introduce the experimental environment and traces. Then, we present the results and analysis.

3.5.1 Experimental Setup

We developed a simulator which simulates a Segate ST8000AS0011 8TB SMR disk based on the description in [2]. The framework of our simulation platform is shown in Figure 3.10. The management scheme of both Tiler and Skylight are implemented in the embedded controller of our simulated SMR disk. The following parameters are inputted into the simulator: 1) the write/read-head width ratio, k is set as three, which indicates that the write-head is two times larger than the read-head; 2) The rotation speed is set at 5900 rpm and the disk has 16 surfaces; thus, each surface has a capacity of 512 GB. 3) M is set as 4096 which indicates there are totally 4096 data tracks within each zone. 4) PC in Skylight occupies around 1%$\sim$10% of the overall space [2], and we use 1% to construct PC (i.e., 5 GB on each surface) while the remaining space is NDA.

We have evaluated the proposed scheme with different traces. The traces are collected from data centers organized by Microsoft Research Cambridge [1]. The detailed characteris-

Table 3.1: The characteristics of the traces.

Traces	Data Size (GB)	Percentage of Write Operations	Percentage of Update Operations
rsrch_0	8.6	91.82%	96.34%
src1_2	36.2	83.70%	89.56%
src2_0	6.9	86.56%	94.13%
src2_2	53.3	68.21%	22.64%
stg_0	12.6	77.51%	94.46%
stg_1	46.5	20.88%	91.33%
ts_0	8.6	84.28%	95.81%
usr_0	21.0	61.41%	91.47%
web_0	15.9	63.32%	93.93%
web_2	18.8	1.88%	42.83%

tics of the traces including data size (GB), write ratio (%) and update ratio (%) are shown in Table 5.1. Among these traces, four traces represent write-intensive workloads and the other four represent write-medium workloads. The remaining two traces represent read-intensive workloads. Each trace is named in a *hostname_disknumber* form. For example, the *stg* trace is for web staging and the disk number is 0.

3.5.2 Performance Evaluation

In this Section, we compare the following five performance metrics for both Tiler and Skylight: 1) average response time; 2) write amplification; 3) read/write-head movements; 4) average cleaning time; 5) write requests distribution.

• Average response time

We firstly evaluate the system overall performance improvement by comparing the average response time of Skylight and Tiler with S=1. The experimental result is shown in Figure 5.8. Tiler benefits from the reduction of read/write-head movements, getting rid unnecessary RMWs and the postponed AR cleaning process. We can observe that Tiler can shorten the average response time by up to 49% and 34% on average comparing to that of Skylight.

• Write amplification

In this section, we compare the write amplification of both Skylight and Tiler with

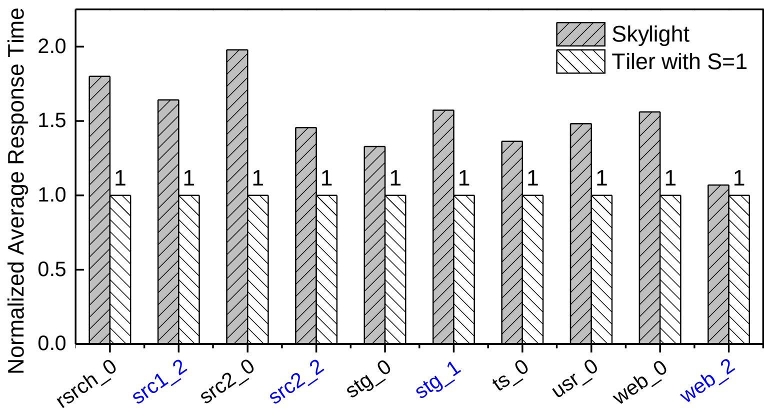


Figure 3.11: Normalized average response time.

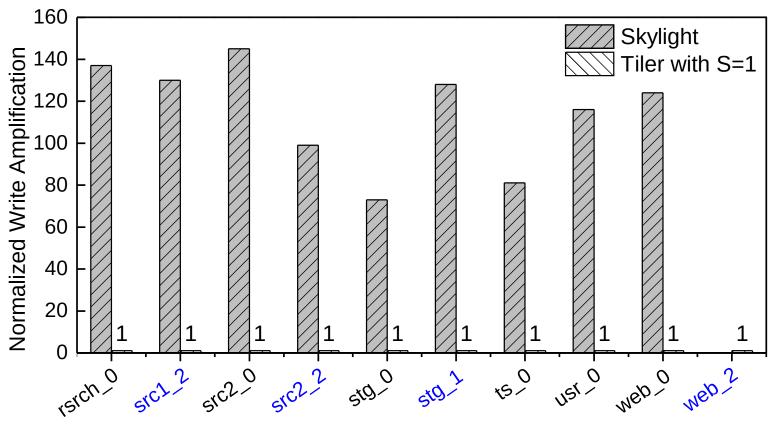


Figure 3.12: Normalized write amplification.

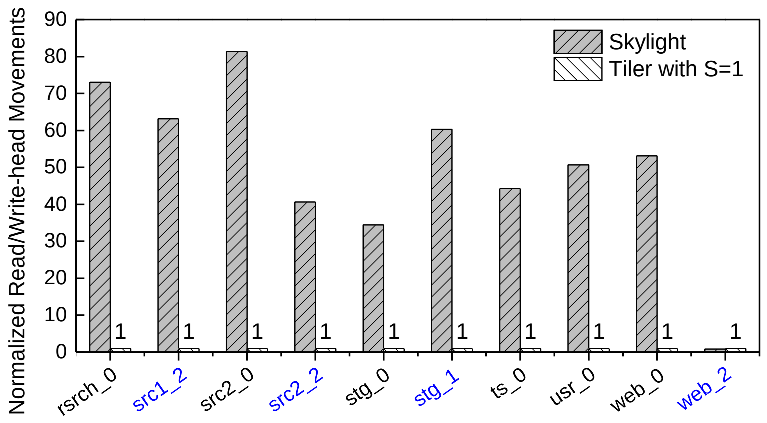


Figure 3.13: Normalized read/write-head movement.

S=1. The write amplification of Skylight consists of two parts including the valid data copy and RMWs triggered when writing to NDA. Since our AR design eliminates the need of RMWs, the write amplification of Tiler only consists of the valid data copy during the AR cleaning processes.

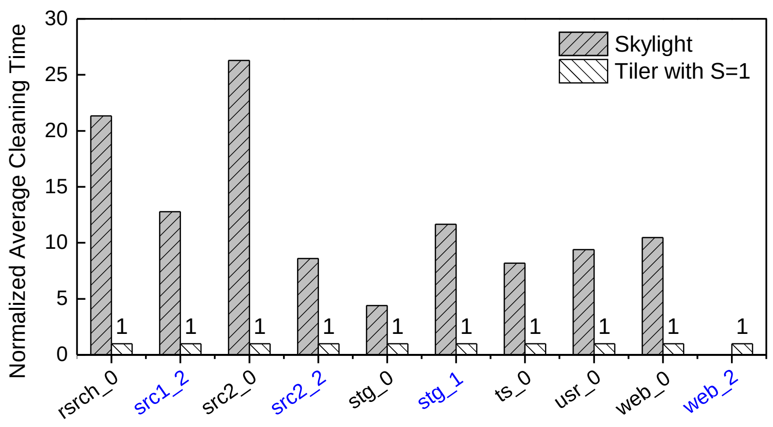


Figure 3.14: Normalized average cleaning time.

We normalize the write amplification value of Skylight to that of Tiler with S=1. As shown in Figure 3.12, the write amplification can be reduced by up to 99.31% and 99.03% on average, which shows the effectiveness of our proposed approach.

• Read/write-head movements

As each read/write request will route the read/write-head to the required position, we count the total read/write-head movements for comparison. We normalize the read/write-head movements of Skylight to that of Tiler with S=1.

During a cleaning, Skylight cleans all the valid data from PC to NDA, thus, resulting in a frequent head movements between the PC at OD and the NDA at ID. Rather than rotating the head between PC and NDA back and forth, Tiler is capable of restricting the head movements within different ARs. As shown in Figure 5.9, the maximum reduction is 81X and the average reduction is 51X.

• Average cleaning time

Since both Skylight and Tiler adopt an out-of-place scheme to manage part of their spaces, cleaning is periodically triggered to reclaim invalidated spaces. At runtime, the system is paused and cannot serve incoming requests until a cleaning process is finished. The average cleaning time is the total time spent on cleaning over the total cleaning count. In this section, we normalize the average cleaning time of Skylight to that of Tiler with S=1.

As shown in Figure 5.11, the average cleaning time can be reduced by up to 25X and a reduction of 10X on average. Compared to Skylight, Tiler spends less time in cleaning ARs and thus is more cleaning-efficient.

• Write requests distribution

We count the write/update requests served in the logging write process and the lazy write process. The write request distribution is shown in Figure 3.15. If a lazy write process is not triggered, common cleaning will be triggered and pause the system. By invoking a lazy write, as shown in Figure 3.15, a lot more write requests can be served until a common cleaning is finally triggered. Postponing a common cleaning can potentially aggregate more

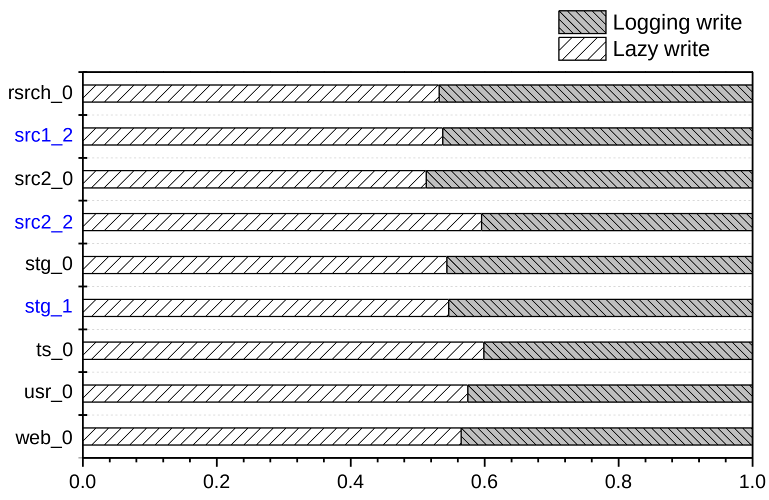


Figure 3.15: Write requests distribution of logging write and lazy write.

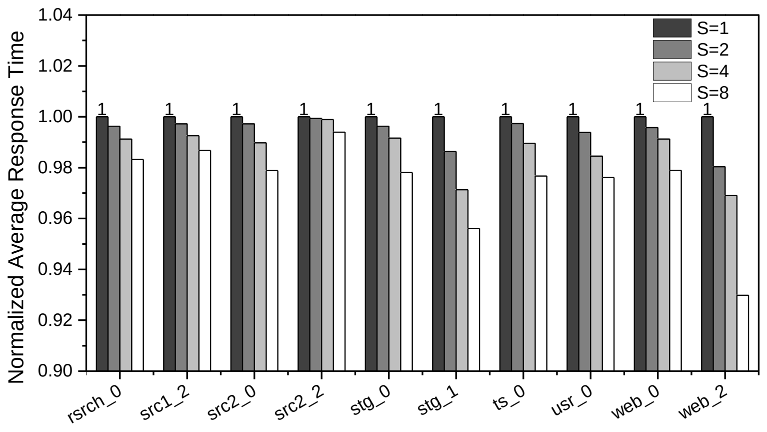


Figure 3.16: Normalized average response time of Tiler with different S.

update requests and therefore, the valid data get more chances to be updated. Thus, the cleaning overhead can be reduced.

3.5.3 Performance evaluation with different configurations of S

We design a set of experiments to show the influence of parameter S on the system performance in Tiler. Tiler is implemented with four different configurations of S, namely, S=1, S=2, S=4 and S=8. We compare the following performance metrics: 1) average response time; 2) track-seek count; 3) cleaning count; 4) valid blocks copy; 5) worst-case response time.

• Average response time

We normalize the average response time to the one of Tiler with S=1 and the results are shown in Figure 3.16. It can be observed that with a larger parameter S, the average response time is shortened. The performance gains mainly come from two aspects including more sequential accessing along rotate direction and the reduced cleaning overheads.

• Track-seek count

Among three time-related components including T_s, T_r and T_t as mentioned in Section 3.4.2, the time spent on track-seek (i.e., T_s) dominates, therefore, by reducing the number of track-seek, the system average response time can be shortened. We evaluate the total track-seek count of Tiler with different S and normalize the results to the one of S=1. As shown in Figure 3.17, as S increases, the total track-seek count is significantly reduced.

• Cleaning count

A common cleaning is triggered to reclaim the invalidated spaces when there are no more schedulable blocks in the lazy write process. We count the corresponding triggered cleaning count of Tiler with different parameter S and normalize the results to the one of S=1. As shown in Figure 3.18, the triggered cleaning count can be significantly reduced as S increases. An AR with a larger S can potentially increase the opportunities to locate more schedulable blocks in the lazy write process. Therefore, the postponed triggering of common cleanings contribute to reducing cleaning count.

• Valid blocks copy

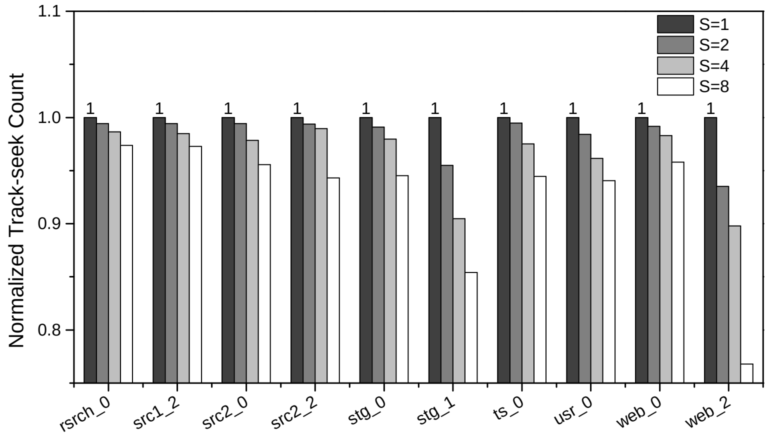


Figure 3.17: Normalized track-seek count of Tiler with different S.

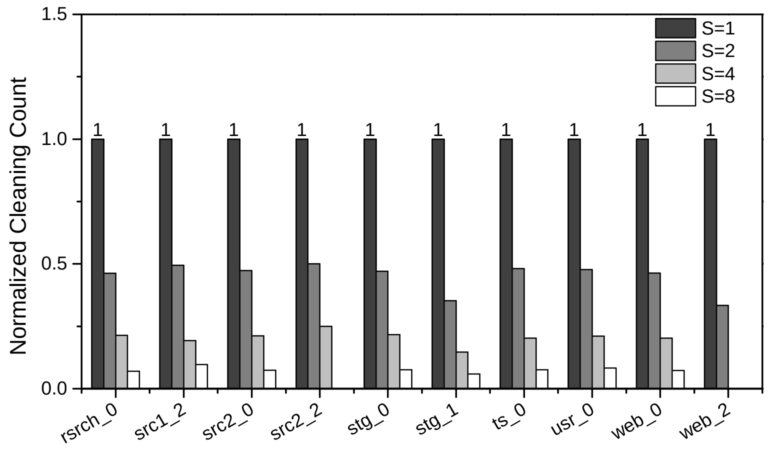


Figure 3.18: Normalized cleaning count of Tiler with different S.

In a common cleaning process, the valid blocks are rearranged back into an AR and therefore, the cleaning efficiency is mainly affected by the total number of valid blocks copy operations. We count the corresponding number of valid blocks copy and normalize the results to the one of S=1. A larger S indicates a larger capacity of an AR and thus, more write/update requests can be served in an AR which can potentially decrease the total number of valid data blocks. Therefore, valid blocks copy decreases as S increases as shown in Figure 3.19.

• worst-case response time

We also quantitatively evaluate a shortage of Tiler with large S. The worst-case response time refers to the maximum time spent on a cleaning. Tiler with larger S indicates that ARs are of larger capacity, therefore, the data volume to be cleaned in the worst-case

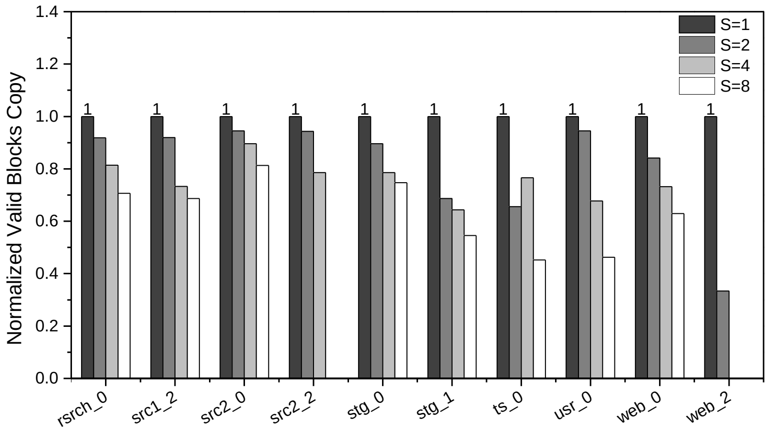


Figure 3.19: Normalized valid blocks copy of Tiler with different S.

Table 3.2: The worst-case response time of Tiler with different S.

Traces	S=1 time (ms)	S=2 time (ms)	S=4 time (ms)	S=8 time (ms)
rsrch_0	3.44	6.32	11.62	30.25
src1_2	2.39	4.28	8.44	15.76
src2_0	3.27	6.26	13.04	33.55
src2_2	2.86	6.51	20.01	0
stg_0	3.68	6.87	12.88	34.57
stg_1	4.89	9.40	20.86	43.91
ts_0	3.60	4.87	13.14	20.74
usr_0	6.88	13.37	21.55	37.16
web_0	4.16	7.33	14.31	34.18
web_2	0.95	1.10	0	0

increases. As shown in Table 3.2, in the worst-case, the cleaning time keeps increasing as S increases. Therefore, Tiler with larger S can potentially pause the system longer. We can find some zeros in the table since the triggered cleaning count of some traces are zero.

3.6 Related Work

There are three dimensions to accelerate SMR-disk, i.e. *device-management*, *host-management*, and *host-awareness*.

The *device-management* techniques minimize the negative effects of SMR properties, e.g. track overlapping and RMWs, to achieve a high throughput of disk. Log-structured write naturally coincides with the unidirectional writing of SMR head, thus, the data layout on disk is researched in [5, 6] to amplify the logging benefit. Based on the logging technique, the disk surfaces are further partitioned into cache regions and native region [2, 9] to obtain the tradeoff between fast logging and slow writing-back. However, more and more researches [38] recognize that common cleaning has become the most severe overhead of logging that keeps SMR-disk busy even without data request. Therefore, some researches [32, 79] focus on how to increase the proportion of in-place updates so that a heavy workload of cleaning could be mitigated.

The *host-management* techniques enable hosts to govern SMR disks by utilizing the host resources, e.g. RAM and SSD, as the unshingled space for random writes [20]. With the overall view by hosts, the in-place update slots can be quickly found in memory [18], which gives the designer more chance to optimize the LBAs stream for SMR-disk. Thus, the shingled-friendly file system [44] is built up in the host. As one of the simplest instances, ShingledFS [75], used for Hadoop, follows *Strictly Append* design, which allows host writes to occur only at the append point of a band. Also, SSDs are another potential choice in hosts to accelerate random accessing, and some hybrid architectures [23, 36, 82] cache active data into SSDs and write inactive data back to SMR-disks.

The *host-awareness* techniques are the compromising designs between the above two, which manages hot data by hosts for high efficiency and cold data by devices for huge capacity. The work in [37] uses write frequency accumulated in the host to classify data into hot and cold, and some band-occupancy-aware algorithms can determine which band to be cleaned by checking the hot ratio of the band. The work in [46] designs a window-based hot data identification to effectively manage data in the hot bands and the cold bands such that it can significantly reduce the cleaning overhead while preventing the random write/update interference.

3.7 Summary

In this chapter, we present the motivation, design, analysis, and evaluations of Tiler. To overcome lengthy cleaning operation (i.e., the major bottleneck of SMR disks performance), we divide the whole disk surfaces into individual small autonomous regions and design a two-level mapping mechanism to manage these ARs. The first-level mapping statically locates an AR in the disk, and the second-level mapping dynamically logs the write requests within each AR. Furthermore, a lazy write process is invoked to select those schedulable blocks so as to postpone the cleanings and reduce the cleaning frequency. Our experimental results show that Tiler can shorten the overall system average response time by up to 49% and 34% on average. The average cleaning time can be reduced by up to 25X and 10X on average.

CHAPTER 4
ALLEVIATING HOT DATA WRITE BACK EFFECT FOR SHINGLED MAGNETIC RECORDING STORAGE SYSTEMS

4.1 Introduction

SMR [7, 16, 24, 25, 27, 49, 51, 66, 68, 78, 94, 100] is a non-volatile media that can provide high capacity for next generation storage devices. By utilizing the overlapping shingle-like structure, SMR storage systems do not require a significant change but can provide two to three times of capacity [22, 26] compared with conventional HDD. The increased density of SMR disks provides a promising solution to satisfy the capacity requirement of big data applications. However, with the asymmetric sizes of the write-head and read-head, a write operation to one track will destroy the data on several adjacent tracks. This feature makes it challenging for SMR disks to effectively manage in-place updates.

To solve this problem, RMW operations are introduced to prevent the data loss in SMR disks. With RMW operations, all data involved in all tracks will be read out and modified in the external storage space (e.g. RAM space) and then written back to the SMR disk. Since multiple read and write accesses are involved, RMW operations inevitably increase access latencies. In order to reduce this overhead, previous work utilizes a log-based persistent cache (the persistent cache) to temporarily hold the data before writing them to the SMR disk [2]. The log-based shingled the persistent cache serves incoming write/update requests along the rotation direction in an appending mode from outer tracks to inner tracks so as to avoid RMW operations. Since update requests will invalidate the stored data, a cleaning procedure is introduced to reclaim the invalidated space.

Several challenges need to be conquered in order to effectively manage the persistent cache. First, the persistent cache does not distinguish hot/cold data (related to frequently or infrequently updated requests, respectively). Thus, when a cleaning operation is triggered, the hot data may introduce unnecessary writes. Second, it also incurs significant overhead by keeping the magnetic read/write heads being routed between the persistent cache at the outer diameter and the native locations at the inner diameter. Third, the capacity of the persistent cache is on the scale of several gigabytes. How to effectively manage the persistent cache remains an open problem.

In this chapter, we present *Dual-buffer* that is a cache management scheme for the persistent cache. Different from conventional single-buffer-based schemes, *Dual-buffer* partitions the persistent cache into two separate buffers, namely the persistent buffer and the filter buffer, that are used to handle incoming data requests and to hold hot data, respectively. The basic idea is to keep hot data in the filter buffer as long as possible, instead of writing them back to their native locations during a cleaning operation. In this way, cleaning operations only trigger a few RMW operations, thereby alleviating the hot data write-back effect and reducing access latencies in SMR disks. Specially, to effectively manage the persistent buffer and the filter buffer, we propose a prediction-based dynamic partitioning mechanism to reconfigure the sizes of the persistent buffer and the filter buffer so as to cache hot data as much as possible by adapting to different workloads. We also propose an address mapping scheme based on a B+ tree data structure so the address mapping of the persistent buffer and the filter buffer and the address transition during cleaning operations can be efficiently accomplished.

We simulate a Seagate drive-managed SMR disk described in Skylight [2] and implement our proposed scheme in the embedded controller. We evaluate the performance of *Dual-buffer* based on a variety of traces. The traces are collected from benchmarks in Microsoft Research Cambridge [1]. Skylight is selected as the baseline scheme for comparison. The experimental results show that *Dual-buffer* can improve the access latency by 55.16% on average and reduce the total RMW operations by 98.76% on average compared with Skylight.

The main contributions of this chapter are:

- We present a novel cache management scheme by which the persistent cache is partitioned into a persistent buffer and a filter buffer and hot data can be cached in the filter buffer as much as possible.

- We propose a prediction-based dynamic partitioning mechanism so hot data can be cached as much as possible by reconfiguring the sizes of the two buffers in the persistent cache.

- An address mapping scheme is developed to achieve effective and efficient space management for the persistent cache.

- Simulation results show that the proposed technique can effectively alleviate the hot data write-back effect in SMR disks and minimize the read/write head block movement.

The remainder of this chapter is organized as follows. Section 4.2 depicts the motivation. Section 4.3 presents the proposed *Dual-buffer* cache management scheme. Section 4.4 shows the experimental results with analysis. Section 4.5 discusses the related work. Finally, Section 4.6 concludes the chapter and discusses future work.

4.2 Motivation

In order to fully exploit the unique aspects of an SMR disk and take advantage of its persistent cache, several issues have to be considered. Writing the data back to their native locations will incur RMW operations and affect the stored data on subsequent tracks. We call this phenomenon the interference effect. Given multiple cleanings, the hot data may remain hot

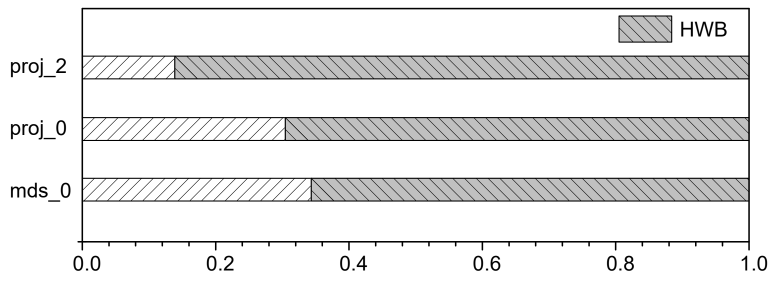


Figure 4.1: A motivation example to show the hot data write-back overhead.

and be written back more frequently than the cold data, leading to more severe interference effect.

To demonstrate the interference effect, we conduct the preliminary experiments on three of the collected traces (in Figure 4.1) as a motivational example. The experiments are conducted based on the environment described in Section 4.4 and the characteristics of the traces can be found in Table 5.1. For each trace, we count the total number of RMW operations (denoted as T-RMW) triggered and the ones introduced by writing back hot data (denoted as HWB), respectively. We show the ratio of HWB over T-RMW for each trace in Figure 4.1. As shown, the RMW operations introduced by writing back hot data contribute to 73% of the total number of RMW operations on average among the evaluated traces. Therefore, we propose to leave as much hot data as possible in the persistent cache.

However, leaving hot data in the persistent cache requires over-provisioned space. This reserved space should be dynamically changed to cater for hot data among multiple cleanings, so the space partition of the persistent cache should not be fixed. Besides, the capacity of the persistent cache is on the scale of several gigabytes. Managing such large-scale address translations and space allocations becomes a critical issue. These observations motive us to propose a cache management scheme to alleviate the hot data write-back effect for SMR disks.

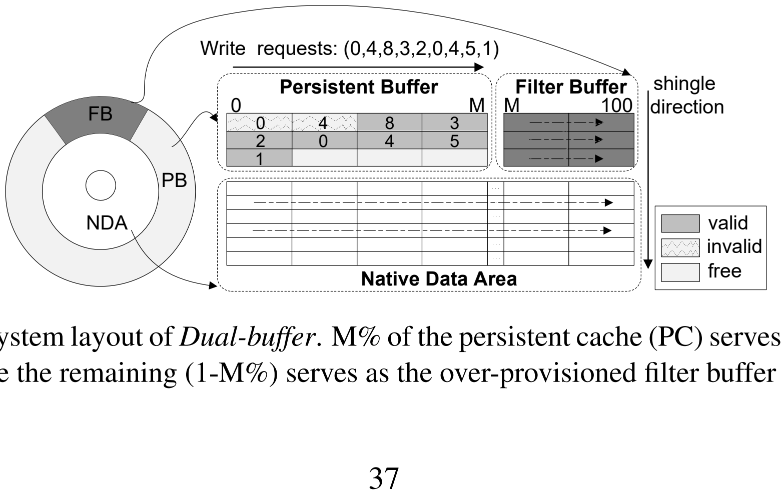


Figure 4.2: The system layout of *Dual-buffer*. M% of the persistent cache (PC) serves as the persistent buffer (PB), while the remaining (1-M%) serves as the over-provisioned filter buffer (FB).

4.3 A Dual-Buffer Management Scheme for SMR Disks

In this section, we first present a system overview in Section 4.3.1. Then, we describe our *Dual-buffer* management scheme including partitioning, space management and cleaning in Sections 4.3.2, Section 4.3.2 and Section 4.3.2, respectively. Finally, we discuss the reliability, overhead issues and alternative solutions in Section 4.3.3.

4.3.1 System Overview

The system overview consists of the following two parts: *1) an architectural overview*, which depicts the system layout and the functionality of each system components (i.e., PC, PB, FB, and NDA); 2) read/write operations and address mapping, which presents how logical addresses are mapped to physical addresses and how read/write operations are operated in *Dual-buffer*.

• **Architectural overview with Dual-buffer**

In the system architecture of our design, an SMR disk consists of the native data area and the persistent cache. Both of them adopt the SMR technology with compact data tracks. NDA occupies the majority of the disk spaces. PC is used to cache incoming write requests and is served as the write buffer of NDA.

As Figure 4.2 depicts, in *Dual-buffer*, PC is further partitioned into two separate buffers, namely the persistent buffer and the filter buffer. The partition point within a round is decided either statically or dynamically by our partitioning process (see Section 4.3.2 for details).

PB serves as the write buffer of NDA in our design. Once PB is full, *Dual-buffer* will issue a cleaning operation to migrate the hot and cold data in PB to FB and NDA, respectively. In other words, FB is used as the eviction buffer of PB, and hot data will be held in FB instead of being written back to NDA. Thus, we can alleviate the hot write-back effect and eliminate unnecessary write operations to NDA.

• **Read/write operations and address mapping** In *Dual-buffer*, read/write operations are

Algorithm 4: *Dual-buffer* write

```
Input: LBA and the data
Output: PBA.
1: if PB is full then
2:     Trigger cleaning.
3: end if
4: Allocate the next available block in PB as $PBA_{PB}$.
5: Write the data to $PBA_{PB}$.
6: Update the next available block in PB,
   and record the mapping ($LBA \rightarrow PBA_{PB}$) in PC-Tree.
7: Return $PBA_{PB}$.
```

Algorithm 5: *Dual-buffer* Read

```
Input: LBA
Output: The data stored in LBA.
1: Search LBA in PC.
2: if LBA is found in PC-Tree then
3:     Obtain the corresponding $PBA_{PC}$ as $PBA$.
4: else
5:     Based on the static mapping,
       translate LBA into $PBA_{NDA}$ as $PBA$.
6: end if
7: Read from $PBA$ and return the data.
```

handled differently. A write request will be served with an appending mode in PB first. Writes to FB and NDA will only occur during a cleaning operation (e.g., cleaning data from PB to FB and NDA accordingly). On the other hand, a read request may involve PC (i.e., PB and FB) and NDA. Specially, for a read request, we will first search it in PC, and if it cannot be found, we will locate it in NDA. The detailed processes of read operations and write operations can be found in Algorithms 5 and 4, respectively.

The address mappings in PC and NDA are managed with different manners. For PB and FB in PC, as data are written in an appending manner, we propose one B+-tree-based data structure (called PC-Tree) so a logical block address (LBA) given by operating systems can be efficiently mapped to a physical block address (PBA) in PB or FB (See Section 4.3.2 for details). In contrast, a static mapping mechanism is adopted in NDA, in which the mapping between LBAs and PBAs is fixed.

By assigning a native location (a PBA) to each LBA within NDA, no memory space is needed for maintenance (unlike dynamic mapping approaches). Therefore, this static mapping approach is adopted. Figure 4.3 shows an example of the static mapping in NDA. An

SMR disk normally consists of multiple 2-surface platters, indexed by the first most significant bits (i.e., the first 4 bits). Each surface has several bands indexed by b bits, and each band consists of some tracks indicated by t bits. A track contains multiple blocks and the last e bits are used to index the block offset within the track. A block is the basic access unit from operating systems. The static mapping is a four-level mapping to track the map information for surface, band, track, and block.

Given an LBA, in Figure 4.3, the leftmost 4 bits are directly routed to the disk controller and used to select a surface in one of the sixteen surfaces. For example, 1100 can route all disk operations to the twelfth surface. The b-bits in the second segment indicate the band on which the operations reside. Then parsing the t-bits in the track segment routes the operation to the specific track and finally, the e-bits indicates the actual block offset within the track. In this way, a particular LBA is translated into a specific PBA in the NDA.

4.3.2 *Dual-buffer* Management

Our *Dual-buffer* management mainly consists of three components: *1) partitioning* is to manage how to partition PB and FB in the persistent cache; *2) space management* is for how to allocate space and perform address mapping in PB and FB; *3) cleaning* is to manage the data movement from PB to FB and NDA when PB is full.

• Partitioning

The physical space partition of PB and FB can be decided either by **static partition-**

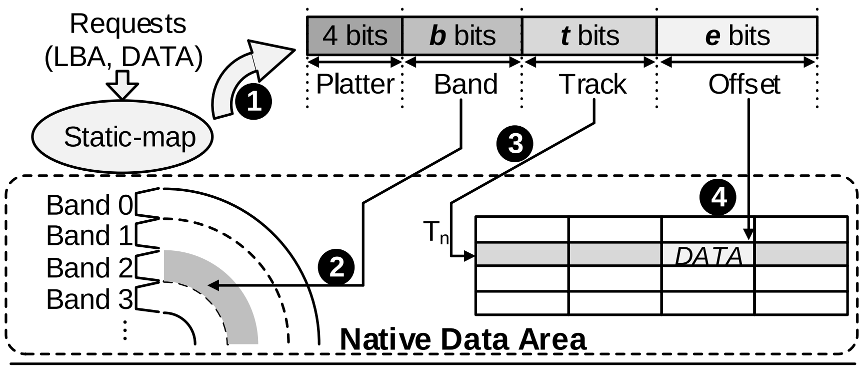


Figure 4.3: An example to illustrate how an LBA is parsed into a PBA within NDA.

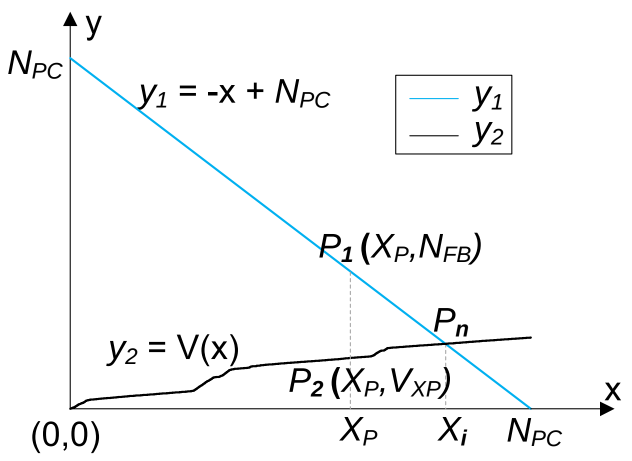


Figure 4.4: An illustrative example of the dynamic partitioning model: line y_1 models the relationship between PB and FB; line y_2 is the plotted curve of replaying a sample trace; P_1 and P_2 are the corresponding points on line y_1 and line y_2; P_n is the intersection point.

ing or **dynamic partitioning**.

• **Static partitioning:** Dual-buffer-static indicates that the partitioning of PC is predefined in advance. The partitioning point M is fixed (e.g., at 60%) and will not be changed between multiple rounds of request serving. Although this static partitioning scheme gains some improvement over the baseline, it cannot adapt to the access behavior of different workloads. Thus, we propose a dynamic partitioning scheme for further optimization next.

• **Dynamic partitioning:** As mentioned above, PB is the write buffer to serve incoming write requests in an appending mode and FB is utilized to hold hot data in cleaning. Let N_{PC}, N_{PB} and N_{FB} be the total available data blocks in PC, PB and FB, respectively. When N_{FB} is small, it is not sufficient to cache all the valid data during cleaning. When N_{FB} is large, space may not be used up, which incurs storage space waste. Besides, in terms of the capacity of PB, the smaller N_{PB} is, the earlier cleaning process is triggered. Therefore, we should find a good partitioning point.

The relationship for space partition can be modeled in Equation 4.1.

$$N_{FB} = -N_{PB} + N_{PC} \tag{4.1}$$

Given an N_{PB}, we have a corresponding N_{FB}. For example, line y_1 in Figure 4.4

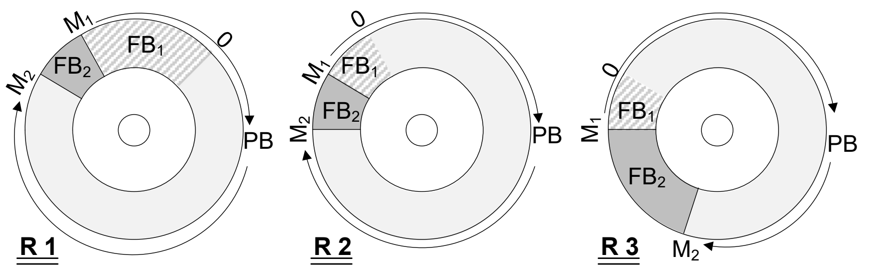


Figure 4.5: An illustration of dynamic partitioning. R1, R2, and R3 refer to Round 1, Round 2, and Round 3, respectively; FB_1 and FB_2 are the last-round FB and the current-round FB, respectively; M_1 and M_2 are the last-round partitioning point and current-round partitioning point, respectively. PB starts at M_1 and ends at M_2.

illustrates this relation, in which, point P_1 indicates that given x = X_P, and the corresponding N_{FB} can be obtained by subtracting X_P from N_{PC}.

When PB is filled up, a cleaning process is triggered. For any allocated blocks (x-values on line y_2), we can obtain the amount of valid data (y-values on line y_2). By replaying a trace named *rsrch_0* (detailed characteristics can be found in Table 5.1), we plot every allocated block (x-value) in the PB and corresponding valid data (y-value), then we obtain line y_2 in Figure 4.4. For instance, the point P_2 on line y_2 indicates that when X_P blocks are allocated, there are V_{XP} valid data blocks.

X_P is the partitioning point and at the same time, it also serves as the cleaning triggering point since there are only X_P blocks that can be allocated. By analyzing points P_1 and P_2, we can conclude that when cleaning is triggered at X_P, N_{FB} exceeds V_{XP}. This indicates that (N_{FB} - V_{XP}) blocks are wasted. As shown in Figure 4.4, the intersection point P_n, is the best partitioning point as well as the cleaning triggering point since V_X equals to N_{FB} at this intersection. For any point following P_n, N_{FB} will be smaller than V_X, which means that the amount of valid data cannot be fitted into FB. As a result, a portion of the data need to be transferred to NDA which incurs the interference effect.

Although we know the intersection of the two lines (i.e., P_n) is the ideal point, obtaining it at runtime is challenging. We adopt a prediction process to find the golden point. By collecting some sampling points, we are capable of fitting a curve, y_2', to closely approach line y_2. By finding the intersection of y_2' and line y_1, a predicted partitioning point can be generated.

At runtime, for every allocation in PB, we form each point (X, V_X) as a sampling/prediction point. Since the data block allocation is along with the rotating direction, the partition should be accomplished before the allocation of the first track ends. Therefore, the sampling points can only be picked within the first track.

As shown in Figure 4.5, in each round, FB_1, which is the last-round FB, occupies a portion of PB (i.e., $M_1 \sim 0$). The next available PBA starts from position 0 in PB and after allocating S% (e.g., 50%) of the first track of PB as sampling points, we determine the current-round partitioning point M_2. FB_2 (i.e., the current-round FB) is reserved for cleaning and it will become the FB_1 of next-round. For example, FB_2 of R1 becomes FB_1 of R2 and FB_2 of R2 becomes FB_1 of R3.

• Space allocation and address mapping

The unique architectural layout of *Dual-buffer* makes the space management challenging for two major aspects. First, the total available space is partitioned into a dual-buffer (i.e., PB and FB), which needs a special space allocation scheme. Second, since PC adopts a dynamic address mapping between LBAs and PBAs, another challenging issue is to efficiently manage this dynamic address mapping. To this end, we adopt the persistent buffer allocation range table (PB-ART) and the filter buffer allocation range table (FB-ART) to address the first issue and the persistent cache tree (PC-Tree) and the filter buffer tree (FB-Tree) for the second issue as shown below.

• Management of PB: PB-ART manages the allocation of free blocks in PB. It contains three indexes, *Physical-Track-Number (PTN)*, *Range* and *Offset*. From the rotating-direction point of view, *Range* is utilized to indicate a consecutive region of a particular track.

PC-Tree records the mapping (between an LBA and the corresponding allocated PBA) and the update count (denoted as Cnt) of the LBA. It is a B+ tree using LBAs as the index and {LBA, PBA, Cnt} as the leaf node. Given an LBA, with {LBA, PBA, Cnt}, we can find the physical block allocated as well as the update count of this LBA. Once an

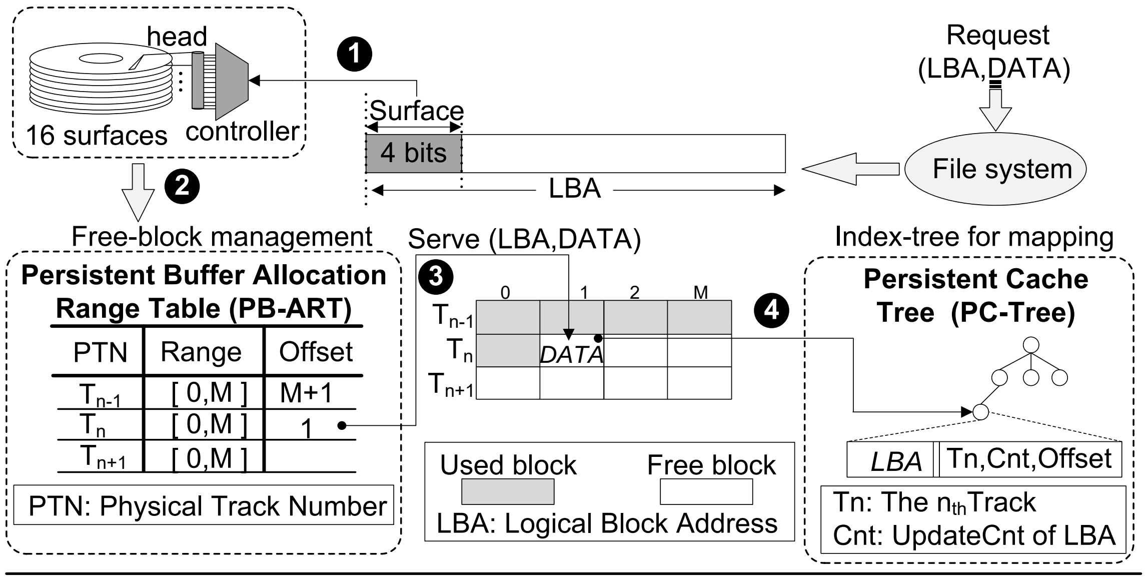


Figure 4.6: An example to illustrate how PB-ART and PC-Tree are updated when one write request is handled.

incoming request is served, the corresponding Cnt will be increased by one. In this way, we can utilize Cnt to identify hot and cold data in cleaning.

An example is shown in Figure 4.6. In the beginning, the leftmost bits of the LBA will be retrieved to route the operation into one of the platter surface. As the predicted partitioning point is at M as assumed in Section 4.3.2, each track of PB can allocate up to M blocks. PB-ART will keep tracking the next available offset (block) on one particular track. Therefore, offset (M+1) of track T$_{n+1}$ indicates that this track is full. Since offset 0 of track T$_n$ is used, offset 1 will be used to serve the request. The pair (T$_n$, offset) represents a unique PBA so that our PC-Tree records the LBA to PBA mapping. If LBA exists in the tree, then the Cnt will be increased by 1; otherwise, Cnt will be initialized as 0.

• **Management of FB:** The allocation and address mapping in FB are similar to those of PB. The FB-ART is in charge of the space allocation while the address mapping is recorded in FB-Tree. There are also three indexes in FB-ART. One major difference is the range that indicates the available write region of one track. For example, if one track has a total N blocks following the rotation direction, PB's range is from 0∼M while the remaining M∼N is served as the allocation pool of FB.

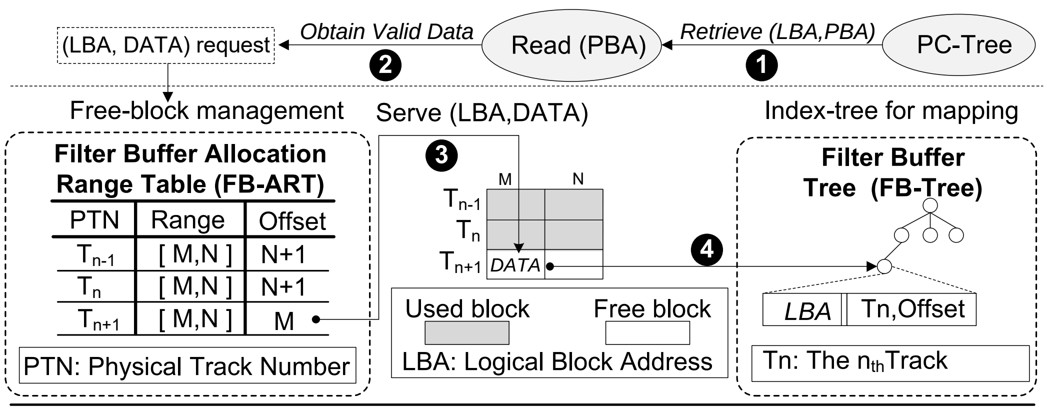


Figure 4.7: An example to illustrate how FB-ART and FB-Tree are updated during cleaning.

As shown in Figure 4.7, every single track shares the same range from offset M∼N and the entry offset indicates that the next available block can be allocated. Offset N+1 of track T$_{n-1}$ and T$_n$ means the tracks are full. Therefore, offset M of track T$_{n+1}$ is allocated for the data. FB-Tree is responsible for recording the address mapping between LBAs and PBAs.

Note that the FB-Tree and the FB-ART are only utilized in a cleaning operation. As shown in the cleaning below, after a cleaning is finished, the FB-Tree becomes the new PC-Tree in next-round.

• Cleaning

When PB is full, a cleaning operation is triggered and Algorithm 6 depicts the detailed process as follows:

Hot/cold data are classified based on Cnt that is the update count of LBA in {LBA, PBA, Cnt} in PC-Tree. Let $N_{leafnodes}$ and N_{FB} be the total valid data blocks in PB and the available free blocks in FB, respectively. 1) When $N_{leafnodes} > N_{FB}$, it indicates that the valid data blocks cannot be all migrated to FB. In this case, we sort the leaf nodes in PC-Tree by Cnt in descending order. Then the first sorted N_{FB} valid data blocks are treated as hot data and migrated into FB. The remaining ($N_{leafnodes}$ - N_{FB}) valid data blocks are treated as cold data and are migrated into NDA; 2) If FB is available to cater all the valid data blocks in PB (i.e., $N_{leafnodes} \leqslant N_{FB}$), these valid data blocks will all be treated as hot data and thereby be held in FB. Once a cleaning operation is accomplished, PB is empty and hot data

Algorithm 6: Cleaning

```
Input: PC-Tree
Output: PB (empty), FB (hot data) and new PC-Tree.
 1: $N_{leafnodes} \leftarrow$ The number of leaf nodes in PC-Tree;
 2: $N_{FB} \leftarrow$ The total block number in FB .
 3: if $N_{leafnodes} > N_{FB}$ then
 4:     Sort the leaf nodes in PC-Tree by Cnt
        in descending order.
 5:     for each sorted leaf node with {LBA, PBA$_{PC}$, Cnt} do
 6:         Read the data from PBA$_{PC}$.
 7:         if FB is not full then
 8:             Move the data to FB.
                Update FB-Tree.
 9:         else
10:             Move the data to NDA.
11:         end if
12:     end for
13: else
14:     for each leaf node in PC-Tree do
15:         Move the data to FB.
            Update FB-Tree.
16:     end for
17: end if
18: PC-Tree $\leftarrow$ FB-Tree.
```

are moved to FB.

During the cleaning process, FB-Tree is built up to record all data in FB with the format {LBA, PBA, Cnt}, in which Cnt is reset to zero. After one cleaning operation is accomplished, the previous PC-Tree is abandoned and the FB-Tree becomes the new PC-Tree. In this way, PC-Tree can index all hot data. Accordingly, hot data in FB can be invalidated or updated by utilizing PB as the write buffer.

4.3.3 Discussion

• Reliability

To handle reliability issues, we reserve multiple innermost tracks within the ID as Reliability Region (R-Region). Our allocation tables and address mapping are periodically checkpointed (every 10 seconds in the experiments) to R-Region. When the system boot-up, the information stored in R-Region will be loaded into memory so the normal operations can be supported.

There is a potential risk that the mapping tables may be modified when the system crashes. Since the allocation is in a deterministic manner, we adopt a crash recovery scheme similar to [103]. Note that to support the crash recovery, the LBA is stored along with the data into each allocated PBA in PB.

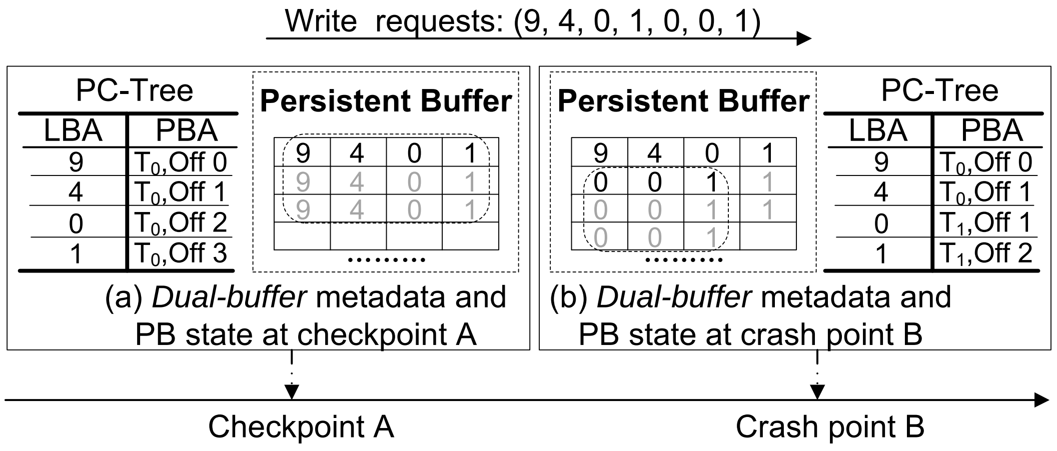
An illustrative example is given in Figure 4.8 to show how the system can be recovered from a crash by using the last checkpoint and the *Dual-buffer* allocation strategy. In Figure 4.8 (a), the first track T_0 is allocated to serve the write requests and the corresponding PC-Tree has been checkpointed. Assume that the write-head width is three times as large as the read-head width (same as track width), thus the data (in gray color) will also be stored on track T_1 and track T_2.

In Figure 4.8 (b), three additional write requests (i.e., LBAs 0, 0 and 1) are served on the blocks of track T_1 between the checkpoint A and crash point B and thus, the data on the first three blocks of track T_1, track T_2, and track T_3 have been updated. Upon the crash, inconsistency occurs since data are updated but the up-to-date PC-Tree (at point B) stored in RAM is lost.

During the crash recovery process, we firstly scan the first block after checkpoint at (T_1, Off 0) and notice that the data has been modified since originally it should store LBA 9. Based on this deterministic behavior, since the last blocks of track T_1 and track T_2 are still storing LBA 1 while the different data pattern can be found from (T_1, Off 3), we can conclude that the last update/write stops at (T_1, Off 2). By knowing these updates between the checkpoint and crash point, we can update the corresponding entries (associated with the obtained LBAs) of the last-checkpointed PC-Tree so as to rebuild the up-to-date one (PC-Tree at point B).

• **Overhead analysis**

In order to distribute the write requests based on the access behavior, *Dual-buffer* adopts a different allocation scheme for dealing with valid data. This introduces a new layer of indirection in order to maintain address mapping. Therefore, it will introduce time and memory space overhead. *Dual-buffer* uses PC-Tree and PB-ART to manage the address map-



Write requests: (9, 4, 0, 1, 0, 0, 1)
PC-Tree
LBA
PBA
9
T0,Off 0
4
T0,Off 1
0
T0,Off 2
1
T0,Off 3
Persistent Buffer
(a) Dual-buffer metadata and PB state at checkpoint A
Persistent Buffer
PC-Tree
0
T1,Off 1
1
T1,Off 2
(b) Dual-buffer metadata and PB state at crash point B
Checkpoint A
Crash point B

Figure 4.8: An illustrative example of the crash recovery process.

ping and space allocation information in the persistent cache. These mapping information are maintained as the leaf nodes in the data structure.

The major memory space required by *Dual-buffer* is for PC-Tree and PB-ART, in which the address mapping and space allocation information in the persistent cache are stored. To save memory space, *Dual-buffer* can adopt the demand-based strategy to selectively cache the mapping information to RAM.

For example, given an 8TB SMR disk, the total amount of the space required to maintain the whole mapping information needs several tens megabytes. Our experimental results in Table 4.4 show that *Dual-buffer* uses a maximum 11.57 MB memory space for management.

In terms of the performance overhead, *Dual-buffer* uses a B+ tree structure to facilitate the address translation. Since the B+ tree support efficient search, this does not incur significant time overhead. Our proposed *Dual-buffer* aims to use FB to capture the hot data and prevent the unnecessary accesses to NDA in the SMR disk. This strategy can effectively reduce the read-modify-write operations.

As *Dual-buffer* can supplement the current SMR design and significantly reduce the time consuming read-modify-write operations, the system performance can be improved significantly.

• **Alternative solutions**

There are other possible solutions, for example, to statically partition PC into multi-log based caches [70, 84]. When one of the buffers is full and needs to be cleaned, another empty buffer can be directly allocated. Two issues need to be solved to adopt this scheme: first, the partitioning of PC is determined in advance, by which it may not well adapt to the variations of workloads; second, the smaller the capacity of a cache becomes, the more frequently a cleaning is triggered. How to make the multi-log caching approach more adaptive with optimal partitioning can be an interesting problem for future research.

Another possible solution is dynamic caching transferring. The dynamic caching scheme [101] mainly focuses on caching data that are frequently accessed/read but not written. It can be an interesting research direction to investigate how to integrate dynamic caching transfer mechanisms into persistent cache management so as to further alleviate the hot data write-back effect in SMR disks.

4.4 Evaluation

4.4.1 Experimental Setup

We simulate a Seagate ST8000AS0011 8TB SMR disk and use Skylight [2] as the baseline scheme for comparison. The simulated SMR disk has a rotation speed of 5900 rpm and the write head width is 2 times larger than that of the read head. A guard region consists of 2 tracks and is located adjacent to the data tracks [2]. The Seagate SMR disk has 8 platters and 16 surfaces (each platter has 2 surfaces), thus for an 8 TB disk, each surface has a capacity of 512 GB . Since around 1%$\sim$10% [2] space will be used as the persistent cache, we choose around 1% of 512 GB for evaluation and therefore, each surface's 5 GB persistent cache is located at the outermost part and the remaining space is NDA in the SMR disk.

The smallest read/write unit is a 4 KB physical block and there are around 1 million blocks in the persistent cache on each surface. The persistent cache contains 9 tracks which are followed by a guard region. The guard region consists of 2 tracks. Log-structure is used in the persistent cache, and it adopts a dynamic mapping strategy. We implemented the dynamic mapping using B+ tree (PC-Tree and FB-Tree) to manage the address translation of

Table 4.1: The characteristics of the traces.

Traces	Data Written (GB)	Data Read (GB)	Writes (%)	Updates (%)
rsrch_0	7.63	0.94	91.82%	96.34%
src1_2	33.48	2.76	83.70%	89.56%
src2_0	5.83	1.11	86.56%	94.13%
src2_2	32.63	20.68	68.21%	22.64%
stg_0	10.53	6.09	77.51%	94.46%
stg_1	5.47	10.94	20.88%	91.33%
ts_0	7.69	1.89	84.28%	95.81%
usr_0	11.06	15.42	61.41%	91.47%
web_0	13.17	10.78	63.32%	93.93%
web_2	1.33	45.21	1.88%	42.83%
mds_0	19.35	3.17	85.91%	96.12%
proj_0	22.01	4.27	51.37%	92.76%
proj_2	28.76	46.93	21.79%	32.08%

1 million blocks. In the native data area, parsing a given LBA by the static mapping guides the header to locate the particular PBA.

A representative STL scheme called Skylight [2] is selected for comparison. The traces we used are collected from enterprise data centers by Microsoft Research Cambridge [1]. The detailed characteristics of traces are presented in Table 5.1 that includes the unique data size of the trace, the write ratio and corresponding update ratio among writes. Each trace file is named in the *<hostname>_<disknumber>* form. Trace *rsrch* is collected from the research project server while traces *src* are representing source control related traces. The *stg* traces are collected for web staging and *ts* is the abbreviation of a terminal server. Traces web are collected mainly from the web/SQL server. The number following the trace name indicates the disk-number, rsrch_0 was traced on the 0-th disk of the research project server. Other traces follow the similar format. The total number of requests of the first ten traces is 1,000,000, in which the data written range from 1.33 GB to 33.48 GB and the data read range from 0.94 GB to 45.21 GB. The total number of requests of *mds_0*, *proj_0* and *proj_2* are 919,569, 1,537,898 and 3,624,879, respectively, in which the data written range from 19.35 GB to 28.76 GB and the data read range from 3.17 GB to 46.93 GB.

The percentage of write operations show the write ratio among all the requests while the percentage of update operations indicate the update ratio among the write requests. Ten of the thirteen traces are write-intensive while the other three traces are read-intensive (i.e.,

Table 4.2: Variances between the ideal and predicted partition points.

Traces	Ideal M (%)	Predict M (%)	Gap over ideal(%)
rsrch_0	95.89	94.83	-1.06
src1_2	95.95	87.85	-8.10
src2_0	90.80	96.02	5.22
src2_2	79.05	89.83	10.78
stg_0	95.77	97.41	1.64
stg_1	92.93	92.63	-0.30
ts_0	94.40	92.05	-2.35
usr_0	88.80	95.64	6.84
web_0	57.24	57.40	0.16
mds_0	94.55	96.18	1.63
proj_0	81.89	93.19	11.30
proj_2	60.12	65.79	5.67

web_2, *stg_1* and *proj_2*).

4.4.2 Results and Discussion

In order to evaluate the effectiveness of the proposed *Dual-buffer* scheme, we use performance metrics including performance improvement, read-modify-write operations, worst case response time, average response time, filer buffer space utilization, ram space overhead and r/w_head movement to compare it with Skylight. We implemented the proposed *Dual-buffer* scheme as a shingled translation layer in the embedded controller in an SMR disk. To perform a comprehensive evaluation for different partitioning schemes, we further implemented Dual-buffer-static and Dual-buffer-dynamic in which the static and dynamic partitioning schemes are adopted, respectively.

• **Performance improvement.**

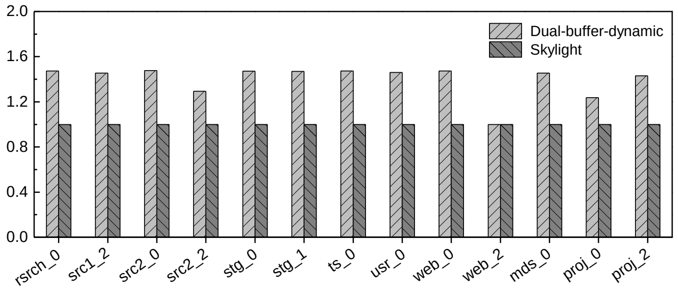


Figure 4.9: Normalized performance improvement over the baseline Skylight [2].

We first evaluate the overall system performance and the normalized results are shown in Figure 4.9. Note that the cleaning counts of each trace are summarized as follows: 1, 5, 1, 5, 2, 1, 1, 5, 5, 0, 4, 4, 5. The experimental results show that *Dual-buffer* can improve overall system performance by up to 55.13%. We can also observe from the results that the average performance improvement is 51.66% among all traces compared with Skylight. The performance gain mainly comes from our locating well-predicted partitioning points during runtime of each trace. Since PB and FB are well partitioned, upon the triggering of cleaning, almost all the valid data can be fitted into FB. In this case, the write-back effect is significantly alleviated.

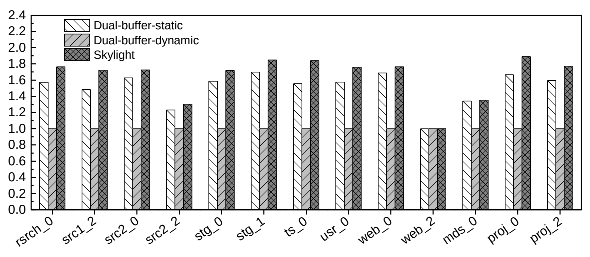
Table 4.2 shows the variances between the ideal points and our predicted partition points. The variance is calculated by subtracting the ideal M from predicted M. A positive value indicates that predicted M falls behind ideal M so that the total volume of valid data (V_x) exceeds the capacity of the FB (N_{FB}). On the opposite, a negative value shows that the reserved FB is sufficient to cater for all the valid data. Most of the variances have a value within 10% and the maximum is 11.30%.

• Read-modify-write operations.

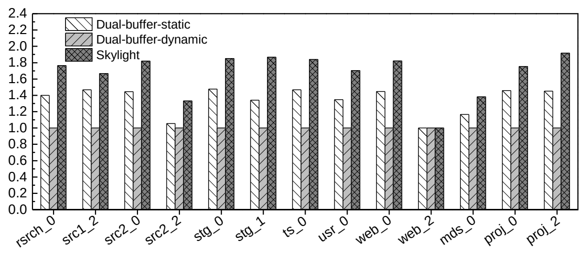
When data are written back to NDA, RMW operations are introduced. Therefore, we count the total number of RMW operations of both *Dual-buffer* and Skylight for comparison.

Table 4.3: The number of RMW operations.

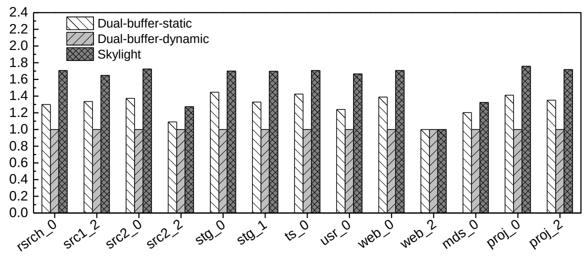
Traces	Dual-buffer-dynamic (times)	Skylight (times)	Reduction (%)
rsrch_0	2,106	206,466	98.98
src1_2	9,266	506,330	98.17
src2_0	4,680	292,482	98.40
src2_2	19,056	3,175,932	99.40
stg_0	3,052	265,356	98.85
stg_1	841	109,282	99.23
ts_0	3,524	214,874	98.36
usr_0	7,898	319,748	97.53
web_0	162	231,560	99.93
mds_0	3,280	196,358	98.33
proj_0	231,384	3,225,692	92.83
proj_2	38,250	546,478	93.00



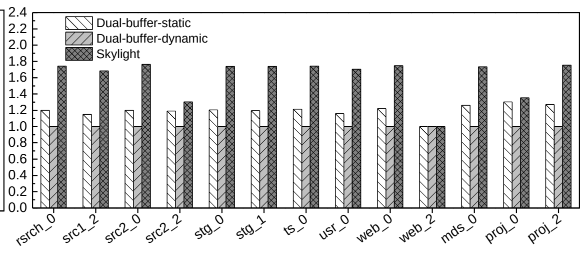
(a) PB = 20% of PC.



(b) PB = 40% of PC.



(c) PB = 60% of PC.



(d) PB = 80% of PC.

Figure 4.10: Normalized average response time.

As there is an acceptable accuracy gap between the exact ideal partitioning point and our predicted partition point, there exists some cold data which needs to be written back to NDA and thus introduces read-modify-write operations.

Table 4.3 shows the experimental results. From the results, it can be concluded that the total number of RMW operations is reduced by up to 99.93% for trace *web_0* and reduced by 98.76% on average among all the traces. As *web_2* is a read-intensive trace in which cleaning operations do not occur, no read-modified-write operations are observed for both *Dual-buffer* and Skylight. Therefore, the result for this trace result is not shown in the Table 4.3.

• **Worst case response time.**

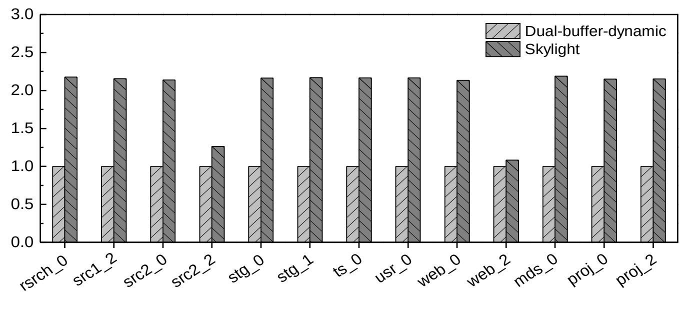
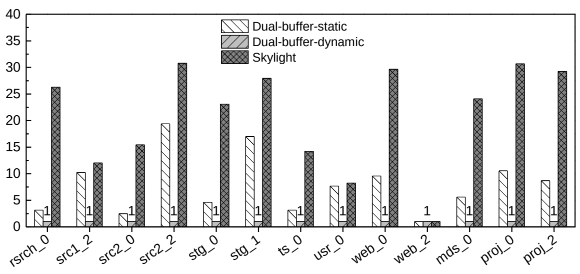
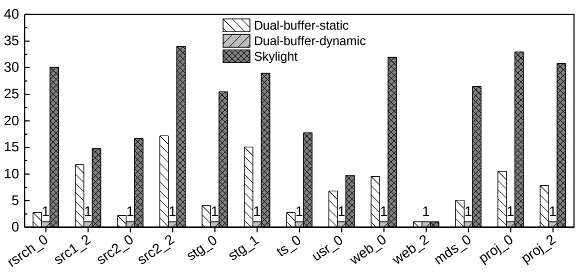


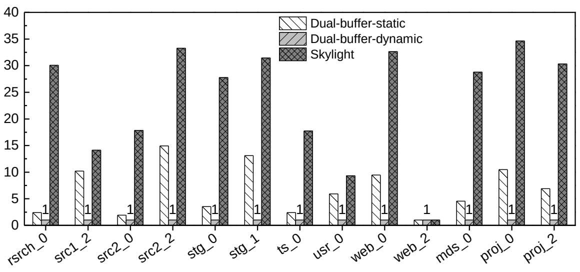
Figure 4.11: Normalized worst case response time.



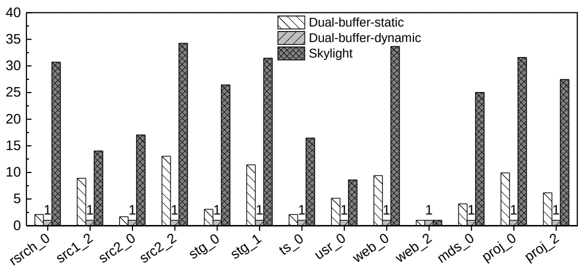
(a) PB = 20% of PC.



(b) PB = 40% of PC.



(c) PB = 60% of PC.



(d) PB = 80% of PC.

Figure 4.12: Normalized r/w_head movement.

The worst case response time is the metric used to measure the longest time of finishing a read/write request.

In this section, we compare the worst case response time of Dual-buffer-dynamic and Skylight. The experimental results are normalized and shown in Figure 4.11. We can observe that our optimized *Dual-buffer* scheme can shorten the worst case response time by up to 2.1x and 1.8x on average. The minimum reduction is from *web_2* trace and no cleaning operations are triggered, so the reduction on worst-case response time is small.

• **Average response time.**

The average response time refers to the time that for the system to respond to each request on average. For the average response time, we compare the results from Dual-buffer-static, Dual-buffer-dynamic and Skylight. Dual-buffer-static adopts 4 configurations of the persistent buffer at 20%, 40%, 60% and 80% of the total available capacity of the persistent cache. Since the persistent cache has a capacity of 5 GB on every surface, PB of Dual-buffer-static is partitioned at 1 GB, 2 GB, 3 GB and 4 GB while FB is the remaining capacity, respectively.

Figure 4.10 illustrates the normalized average response time of three schemes. The experimental results indicate that even using a static configuration, *Dual-buffer* outperforms Skylight. The response time of Dual-buffer-static is reduced by 33% on average. The performance gain mainly comes from caching valid data in FB during the cleaning operations so as to avoid unnecessary writes. Dual-buffer-dynamic performs the best among all the traces since almost all the valid data can be cached in FB and the write-back operations can be eliminated.

• FB utilization.

The filer buffer space utilization is used to measure the utilization rate of the reserved FB.

To further demonstrate the effectiveness of our prediction-based dynamic partitioning scheme, we calculate the actual space utilization among all the evaluated traces and show the results in Figure 4.13. We can conclude from the results that in the traces including src2_0, src2_2, stg_0, usr_1, web_0 and web_2, the space of FB is 100% utilized. The remaining traces show a space utilization at around 98% on average. It indicates that among these remaining traces, a very small portion of space is wasted since the capacity of FB is slightly less than the size required to hold all valid data.

• Memory overhead.

The RAM space overhead is quantified by measuring the actual usage of the memory space of *Dual-buffer*.

Dual-buffer maintains a B+ tree (PC-Tree) to perform data management in the per-

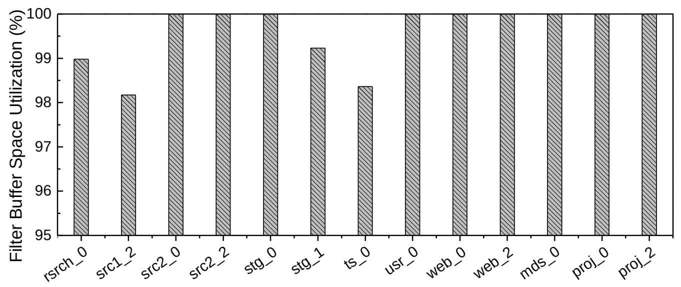


Figure 4.13: The space utilization (%) of filter buffer.

sistent buffer. The number of leaf nodes represents the space overhead to conduct the address mapping. We capture the maximum live leaf nodes during runtime of all the traces and the results are shown in Table 4.4. The corresponding RAM space overhead is quantified. From the results, in the worst case, we need to hold as many as 758,555 nodes and the related RAM space overhead is 11.57 MB. For read-intensive workload, e.g. *web_2*, the minimum size of PC-Tree is observed which indicates a RAM space overhead of 0.13 MB. Skylight adopts a block-level mapping scheme and yields a 160 MB mapping table [2]. As a comparison, our proposed scheme is more memory-efficient than Skylight.

• **Read/Write head movement.**

Since each read/write request needs to route the read/write-head to the required PBA, we count the total read/write-head movement for comparison.

The experimental results of the read/write head movement during cleaning are shown in Figure 4.12. During cleaning, Skylight will clean up all the valid data in PC into NDA, the R/W head will keep shifting from PC to NDA back and forth. On the other hand, as our Dual-buffer-dynamic scheme well reserve FB to cache almost all the valid data, few head movement is required to transfer cold data back to NDA. For Dual-buffer-static, a medium volume of cold data filtered out from the valid data needs to be cleaned up to NDA, thus it incurs some r/w head movement overhead.

Table 4.4: RAM space overhead of *Dual-buffer*.

Traces	#. of leaf nodes in PC-Tree	Memory Used (MB)
rsrch_0	758,555	11.57
src1_2	709,671	10.83
src2_0	745,778	11.38
src2_2	453,473	6.91
stg_0	749,323	11.43
stg_1	756,566	11.54
ts_0	755,539	11.53
usr_0	731,794	11.17
web_0	749,131	11.43
web_2	8,859	0.14
mds_0	759,799	11.59
proj_0	736,165	11.23
proj_2	519,772	7.93

4.5 Related Work

SMR File Systems. File systems can be designed to provide support to manage SMR disks [59, 102]. In SFS [44], a 64 MB band is assumed and sequential-write-required bands similar to the persistent cache are used to store data. Manglogs [65] improve the disk bandwidth by implementing scattered logs within the magnetic recording file system. SMRfs is an effective user-level file system designed for managing SMR disks in the host [38]. Ext4-*lazy* [3] applies a small change to the Linux Ext4 file system to significantly improves the throughput of SMR disks. Since these file systems are tailored for SMR disks, they require supports and changes from the host side which is not the focus of our work.

Hybrid System. Some schemes integrate RAM and solid-state drive (SSD) [10, 34] in current SMR infrastructures as over-provisioning space to avoid the read-modify-write effect [20, 29]. Other schemes utilize SSD to replace the persistent cache to effectively manage SMR disks [81]. For example, in HS-BAS [93], they use SSD as the cache to reduce the effect of high cleaning cost of the persistent cache. All these schemes well utilize high-performance memory devices to address the random update and write amplification issues of SMR disks.

Drive-Managed. A drive-managed mode SMRs includes an embedded controller to hide its unique characteristics from the upper layers (e.g., operating system). The embedded controller deals with the read-modify-write (RMW) operations inside SMR disks with various management schemes [5, 6]. Similar to the flash translation layer in flash memory management [11, 74], drive-managed techniques conceal the unique characteristics of SMR disks. The drive-managed SMR disks are compatible with the block-layer interface since SMR disks normally incorporate the persistent cache to temporally store the data. Optimization of the cache region and a cleaning of the cache are proposed for drive-managed techniques [32, 79].

Different architectural layouts are designed to accelerate data management in SMR disks. The SMR disk in Skylight manages the shingled persistent cache by implementing an indirection system to eliminate RMW operations [2]. The work in [9] not only adopts a workable dynamic mapping to manage the persistent cache but also optimizes the native data area management by using an S-block structure. Our work focuses on alleviating the hot-data write-back effect and can be combined with S-block [9] to improve the performance of both the persistent cache and the native data area of SMR disks. A hybrid wave-like shingled recording disk system is designed to improve both the performance and the capacity of a shingled-write-disk [54]. The work in [90] partitions the whole disk into multiple zones and manages the SMR disk in zone-based API in a host-aware mode.

Some previous studies suggest classifying data into cold and hot data [33, 45]. For example, the work in [37] uses write frequency to separate hot and cold blocks to reduce data movement. SMORE [58] develops an efficient cold data storage in order to achieve full disk bandwidth when ingesting data. Our proposed scheme adopts the write frequency to rank the active valid LBAs when the filter buffer is not capable of caching all the valid data. The focus of our scheme is predicting the ideal partitioning point at runtime but not classifying hot/cold data. There exist some prediction-based schemes in [21, 41, 104] which can be combined with our work.

4.6 Summary

In this chapter, we proposed *Dual-buffer*, a cache management scheme for the SMR disk. *Dual-buffer* uses a two-level buffer cache architecture to postpone the frequent write-back operations of hot blocks. Based on this architecture, we present a prediction-based dynamic configuration to partition the persistent cache. The proposed fine-grained prediction model is a promising work as it can maximize the space utilization ratio. An efficient dual-buffer-aware space management and allocation is introduced to effectively manage the data stored in the persistent cache. Experimental results show that our scheme can significantly reduce unnecessary read-modify-write operations by up to 99.93% and therefore achieve a better access latency (i.e., up to 55.13% performance improvement). In terms of the future work, we plan to extend *Dual-buffer* to host-managed and host-aware SMR management schemes, which may require different architectural and operating system optimization strategies.

CHAPTER 5
RMW-F: A DESIGN OF RMW-FREE CACHE USING BUILT-IN NAND-FLASH FOR SMR STORAGE

5.1 Introduction

Recently, SMR disks [15, 43, 52, 53, 56, 57, 62, 64, 76, 98] have been proposed to be an alternative to satisfy the capacity requirement for big data applications. Compared with traditional HDDs, SMR disks are more cost-effective for its capacity and low cost (i.e., cost-per-gigabyte is competitive). However, SMR disks have poor performance (e.g., high responding time) due to the internal unique characteristics (shingled tracks). That is, writing to a certain track may destroy the stored data on the subsequent tracks [6, 29]. To avoid damaging the data, a time-consuming operation named RMW is incurred to read out the stored data and write them back with the modified data in sequential order. In this chapter, for the writes to SMR disks that can incur RMWs, we propose to leave them in a built-in NAND flash [12–14, 28, 30, 35, 39, 47, 48, 72, 73, 85–87, 99] to eliminate the need of RMWs.

Previous works use SSDs [2, 49, 81] or SMR disks [9, 90] as a first-level persistent cache to buffer all incoming writes without introducing RMWs. Since an SMR disk is usually of terabytes-level, the persistent cache is relatively small to buffer all the writes and can be full frequently. A cleaning process is triggered to move the data from the cache back to the SMR disk and this write-back procedure will inevitably incur a large number of RMWs within the SMR disk. Therefore, the overall system performance can be degraded by the RMWs significantly.

In this paper, we no longer regard the persistent cache as a first-level cache but change the functionality of the cache by taking the intrinsic characteristic of SMR disks into considerations. Specifically, we distinguish data according to their write-back (to SMR disks) behavior, i.e., for the writes to SMR disks, some writes will incur RMWs while the other writes will incur no RMWs. Therefore, we propose to leave the writes that can incur RMWs into our built-in NAND flash cache while the remaining writes are performed directly to the SMR disk. In this way, our design not only can ensure that no RMWs are needed but also can reduce the cleaning frequencies so as to improve the system performance. However, the proposed design brings some new challenges: (1) a new internal architecture for both flash-cache and SMR disks is needed to co-manage the system; (2) how to map the SMR logical spaces into the NAND flash physical spaces; (3) how to perform cleaning when the available spaces in the cache becomes low.

To address the first challenge, we propose a dual-space management module within the controller of the SMR device to co-manage the system efficiently. Writes to the SMR device will be classified into two main categories and handled accordingly.

To handle the second challenge, we propose to use a hybrid two-level mapping to record the mapping between SMR logical addresses and physical flash addresses. The first-level block mapping is used to map the same set of sectors within a zone into a single physical flash block which is statically assigned. After the sector is mapped, within the block, the SMR logical sector is dynamically associated with a flash physical page.

To conquer the third challenge, when one of the blocks of RMW-F becomes full (i.e., the remaining free flash block reach a low watermark), a cleaning process is triggered. Instead of cleaning up the whole flash-cache that can significantly hurt system performance, flash-cache is cleaned at the flash-block level. A heuristic model which takes both write-back cost and data popularity is adopted to further improve the cleaning efficiency.

We have built a trace-driven flash and SMR simulator and implemented our scheme with this simulator. The performance of the system is evaluated with collected traces from Microsoft Research Cambridge [1]. The experimental results show that compared with Skylight-SMR-2 [2], RMW-F can shorten the overall system average response time by over 79% and improve the cleaning efficiency by approximately 15.6 times.

The main contributions of this chapter are summarized as follows:

- We propose a new architecture which adopts built-in NAND flash as the RMW-free cache of the SMR system. Some new modules are implemented to accelerate the management efficiency of the hybrid storage system.
- We propose a hybrid two-level mapping scheme to handle the address translations between the SMR logical addresses and the NAND flash physical addresses.
- We improve the cleaning efficiency by a heuristic model that takes both write-back cost and data popularity into account.
- We have built an SMR and flash simulator and evaluated our proposed RMW-F with various traces.

The rest of this chapter is organized as follows. Section 5.2 gives the motivation for this work. Section 5.3 describes the design and implementation details of our proposed scheme. Experimental results are provided in Section 5.4. Related works are presented in Section 5.5. Section 5.6 concludes the chapter.

5.2 Motivation

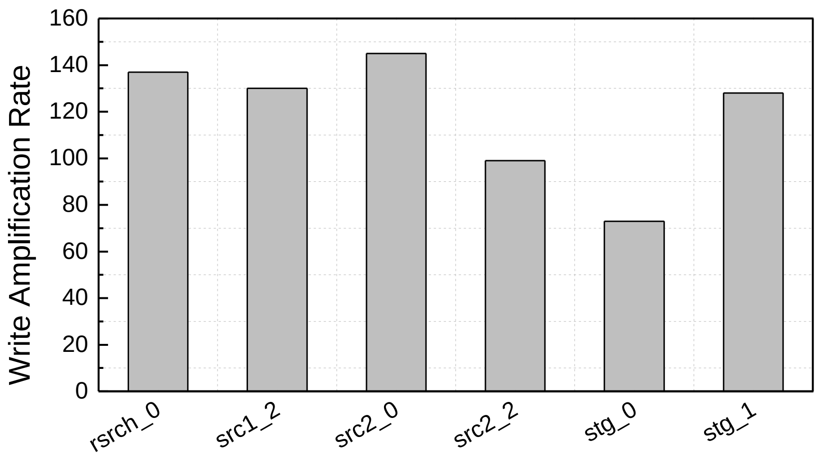


Figure 5.1: A preliminary experiment to measure write amplification brought by RMWs.

By serving write requests in an appending mode, no RMW is needed in PC and thus reduces the negative effect to some extent. However, when PC is full, all the still valid data in the cache will be moved back to the SMR disk during a cleaning process to clean up the cache. Writing the data back to their native tracks within the SMR disk will inevitably incur RMW operations since all the stored data on the subsequent tracks will be affected. Therefore, the write amplification rate brought by the RMWs can be large and the system performance is degraded significantly.

We have conducted a preliminary experiment to show the negative effect brought by RMWs by measuring the write amplification rate among some collected real-world traces. The write amplification rate is calculated as the ratio of the total inputted data over the total amount of data written to the SMR disk (includes the written data involved in RMWs).

As shown in Figure 5.1, all the collected traces achieve very high write amplification rate ranging from 73$\sim$145 and thus RMWs can be the system performance bottleneck and should be avoided as many as possible.

5.3 RMW-F Design

In this section, we first present an overview of our RMW-free architecture. After that, we present the dual-space management in both NAND flash and SMR disk followed by the sector locator in the SMR disk and the address translator along with the garbage collector in the flash.

5.3.1 System Overview

Figure 5.2 shows the general architecture of RMW-free cache. RMW-free cache tightly couples the components in both of the NAND flash and the SMR disk. In particular, RMW-F implements a dual-space management module to efficiently allocate spaces in the hybrid storage (including NAND flash and SMR disk) to ensure no RMWs are needed. Both NAND flash and SMR disk expose their information about the physical blocks and the internal layout of SMR disk platters to the controller and thus, the controller can pass the data along with its characteristic to different locations. Especially, our RMW-free cache no longer serves as a centralized first-level cache of the SMR disk.

To ensure no RMWs are needed in the SMR device, the dual-space management

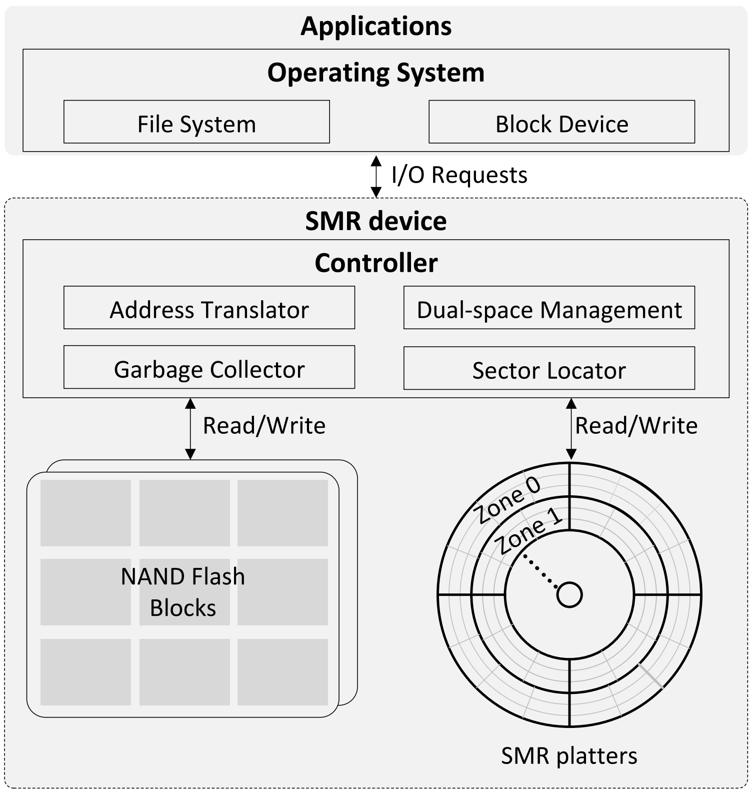


Figure 5.2: An overview of the system architecture of RMW-F.

module is responsible for distinguishing different types of writes, that is, writes to the SMR disk can only be performed when RMW operation is not needed while writes that can incur RMW in the SMR disk will be cached in the NAND flash. When writes are performed accordingly, the address translator of NAND flash and the sector locator of the SMR disk should be updated to record the mappings between the logical sector addresses (LSA) of the SMR disk and the physical locations (including physical page number in flash and physical sector number in the SMR disk). Although not every writes is cached in the NAND flash cache, it will be eventually filled up with RMW-free data and the garbage collector should be invoked to reclaim the spaces of NAND flash to make room for incoming writes.

5.3.2 Dual-space Management

In order to ensure no RMW is needed when performing writes in the SMR disk, one of the key functionalities of our dual-space management module is to distinguish writes according to their write-back behavior. For those write-back (to SMR) which will not incur RMW, they can be performed in the SMR disk directly and a dual-space manager will allocate the associated physical sectors for them via sector locator (which detailed mechanism is introduced in the later section). Otherwise, for those data which will incur RMW when writing back, they will be sent to the NAND flash for caching.

We use an example as shown in Figure 5.3 (a) to show the logical sector number (LSN) representation within an SMR disk. There are totally four data tracks and five sectors on each track, therefore, LSNs range from 0 to 19 as shown in Figure 5.3 (a). To interpret a certain LSN, for example, LSN equals to 6, it is indicating the position on (track T2, sector S2).

As shown in Figure 5.3 (b), a write requests sequence is handled by our dual-management module. Since the first six writes will incur no RMW operation within the SMR disk, they can be directly performed on the track T1 and (track T2, sector S1).

However, for those writes that will incur RMW if performed in the SMR disk, they will be directed to the NAND flash cache instead. A block containing some free pages will be selected to append the writes and the corresponding mapping will be recorded in the address translator (which will be included in the following section). As shown in Figure 5.3 (b), for the second identical LSN=0, it is regarded as an update operation and since rewriting it to position (track T1, sector S1) will destroy the data on the following tracks, this write will be routed to NAND flash cache. Another write associated with LSN=7 should not be performed

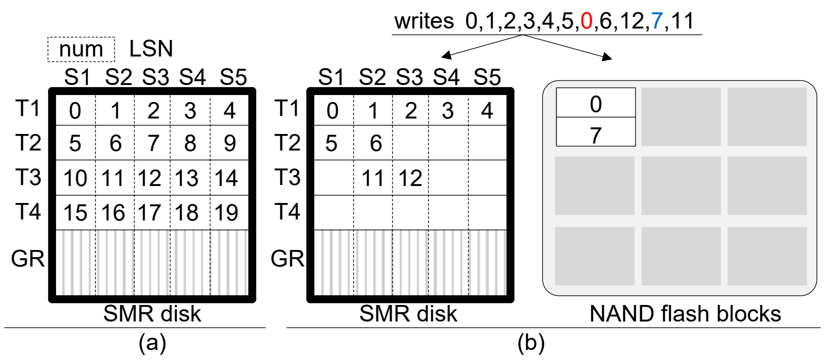


Figure 5.3: (a) An example of logical sector numbers in an SMR disk. (b) An example of write requests handling by the dual-space management module. Abbreviations: LSN: logical sector number; T: track; S: sector; GR: Guard Region.

in the SMR disk either since writing to position (track T2, sector S3) will destroy stored data on position (track T3, sector S3) and thus will introduce RMW.

In order to determine a certain write to be performed in the SMR disk or the NAND flash cache, the dual-space management module should tell whether there are stored data on the subsequent tracks or not (e.g., the example of writing LSN=0 and LSN=7 as shown in Figure 5.3 (b)). Therefore, each set of sectors should have a track pointer to record the last written track (i.e., T2,T2,T3,T1,T1 for S1,S2,S3,S4,S5, respectively).

The details about the above mentioned dual-management module to handle write requests can be also found in algorithm 7.

Algorithm 7: Dual-space management write

```
Input: write (LSN, data)
Output: The data stored in a physical SMR sector or flash page.
 1: Distinguish the write according to their write-back behavior:
    parse LSN into (track $T_i$, sector $S_i$) form via sector locator.
 2: Obtain the track pointer $T_p$ of sector $S_i$.
 3: if $T_p$ ¿ $T_i$ then
 4:     send to NAND flash cache to be handled by address translator.
 5:     Return.
 6: else
 7:     instruct disk-head to write data to (track $T_i$, sector $S_i$) directly.
 8:     update track pointer: $T_p = T_i$.
 9:     Return.
10: end if
```

After the data have been placed in the dual-space, on accessing/reading the data for a given LSN, the dual-space manager will lookup the required data in the flash-cache at the first place via the assistance of address translator (which will be introduced in the following section). If the data can be found in flash-cache, it will be returned from the physical flash page indicated by the address translator. Otherwise, if the data is not found, this case indicates that the data resides in the SMR disk and the sector locator is involved to parse the LSN into the actual physical SMR sector (i.e., PSN). Then, by routing the read-head to that PSN, the data can be read out directly. The read requests handling by the dual-management module can be concluded in algorithm 8.

Algorithm 8: Dual-space management read

```
Input: read (LSN)
Output: The data associated with this LSN.
1: Look up flash-cache with the assistance of address translator.
2: if The LSN is recorded in address translator, return the physical flash page number (PPN). then
3:     Perform flash read on the PPN to get the stored data.
4:     Return.
5: else
6:     Parse LSN into track $T_i$, sector $S_i$ form via sector locator.
7:     Instruct the read-head to locate $T_i$, $S_i$ and perform the read operation to get the data.
8:     Return.
9: end if
```

5.3.3 Sector Locator

For a given LSN, our sector locator module is responsible for parsing the given LSN into physical sector via a fixed mapping (known as static mapping) to manage the mapping between SMR logical addresses and SMR physical addresses. As shown in Figure 5.4, the geometry layout of zone 0 in the SMR disk consists of M tracks and N sectors on each track, therefore, there will be M$\times$N sectors within zone 0. Given an LSN, as the addresses ranges are exposed to the sector locator, the zone which the LSN belongs to can be determined. Dividing the LSN by N, the quotient plus one will be the track number while the remainder plus one will be the sector number.

For example, as shown in Figure 5.3 (a), given an LSN=11, dividing 11 by 5, the quotient and the remainder are 2 and 1, respectively. Therefore, the corresponding track number and sector number are T3 and S2, respectively. On receiving an LSN, the sector

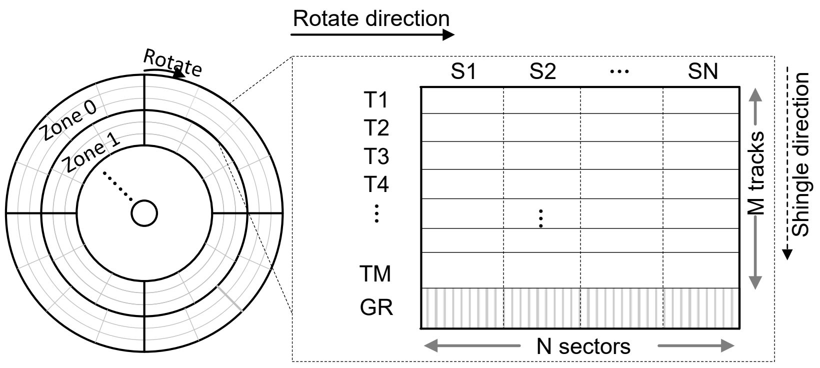


Figure 5.4: An example of static mapping used in the sector locator module.

locator can always transform the LSN into its corresponding track number along with its sector number. The read/write-head router can then perform track seek and can rotate to the destiny sector to access the physical sector in the SMR disk.

5.3.4 Address Translator

One key issue to be handled by the address translator module is address translation between SMR logical addresses and NAND flash physical addresses. We propose a block-level mapping to statically map the same set of SMR sectors into an independent NAND flash block as shown in Figure 5.5. For example, within zone 0 of the SMR disk, there are N sectors on each track and the same group of sectors along with the shingled direction can be denoted as S1, S2, S3, up to SN as shown in the upper part of Figure 5.5. The address translator statically associates a physical flash block (ranging from B1 to BN) to the same set of SMR sectors as shown in the lower part of Figure 5.5 (i.e., S1→B1,S2→B2...SN→BN). Note that, for illustrations, we only show the block-level mapping by using zone 0 as an example, other zones adopt the same mapping mechanism. Since this static mapping is fixed which requires no memory overhead, it is space-efficient.

A page-level dynamic mapping is used when an LSN is mapped into the flash block to record the sector-to-page mapping (S2P-map). As shown in Figure 5.5, LSN including 100, 102 and 104 have been originally stored into the SMR disk sequentially without introducing RMW. When the next incoming writes associated with LSN including 101 and 103 arrive, they will be sent by the dual-management module since they will affect the stored data in the SMR disk when writing back. Therefore, LSNs including 101 and 103 will be logged into the next free page of flash block B2 and the corresponding entry of the B2: S2P-map will be updated accordingly as shown in the lower-right-part of Figure 5.5. The popularity of each LSN will be varied according to different workloads, therefore, the address translator also keeps tracking of the access count of each LSN. The access count will be recorded in the S2P-map so as to distinguish hot and cold data. Specifically, a predefined threshold (denoted as T) is used to perform a comparison with the access count, the access count larger than T will be classified as hot data while the others will be regarded as cold data.

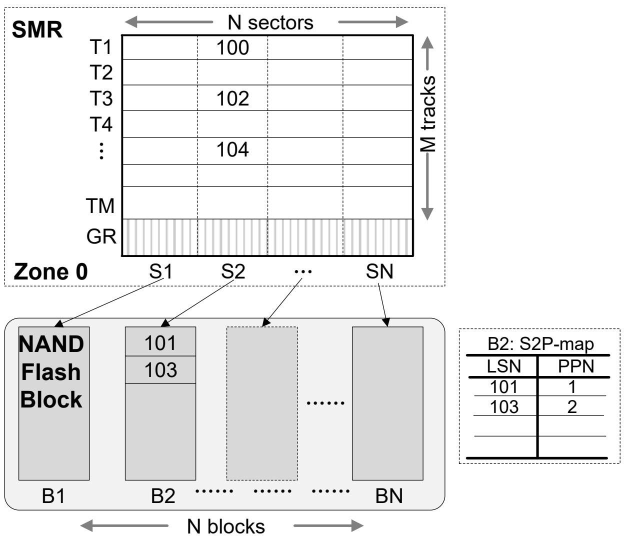


Figure 5.5: An overview of block-level mapping.

On handling the lookup request sent by the dual-space manager, the following algorithm 10 concludes the process. Given an LSN, it will be mapped into its assigned physical flash block by the above mentioned block-level mapping. Then, the S2P-map of the block can be accessed to locate the corresponding entry. If the recorded PPN can be found, it will be returned to the dual-space management module. Otherwise, a NULL flag is returned to indicate the lookup failure.

5.3.5 Garbage Collector

As mentioned in section 5.3.4, we adopt a two-level mapping to map the sparse SMR logical spaces into the flash physical spaces at the unit of blocks. For the same set of sectors within a zone, when there are no available pages within the assigned physical flash block, the garbage collector is invoked to reclaim spaces so as to serve future incoming writes. However, the garbage collector is well aware of the dual-space underlying storage. Therefore, for those still valid LSNs within a block during the garbage collection period, one key decision of our garbage collector to make is determining which LSNs should be written back to the SMR disk and the remaining ones should be rearranged back into the physical block.

Based on observation, for the same set of sectors (e.g., S_i), the write-back cost of each LSN is different. As shown in Figure 5.6, the same set of sectors S_3 is mapped into

Algorithm 9: Address translator allocation

```
Input: dual-mgt-write (LSN,data)
Output: write data to a PPN.
 1: Map the LSN into its assigned physical flash block $B_i$ via the block-level mapping scheme.
 2: Obtain the associated page-level mapping $B_i$:S2P-map and get the next available physical page number (PPN).
 3: if $B_i$ is not full. then
 4:     Write data to the PPN.
 5:     update the mapping of LSN$\rightarrow$PPN in the S2P-map.
 6: else
 7:     Trigger garbage collector.
 8:     Allocate the first available page of $B_i$ as PPN.
 9:     Write data to the PPN.
10:     Update the mapping of LSN$\rightarrow$PPN in the S2P-map.
11: end if
12: Return.
```

Algorithm 10: Address translator lookup

```
Input: dual-mgt-look-up (LSN)
Output: a recorded PPN or not found.
 1: Map the LSN into its assigned physical flash block $B_i$ via the block-level mapping scheme.
 2: Obtain the associated page-level mapping $B_i$:S2P-map.
 3: Obtain the corresponding entry of LSN within the S2P-map.
 4: if the associated PPN is found. then
 5:     return PPN.
 6: else
 7:     return NULL.
 8: end if
```

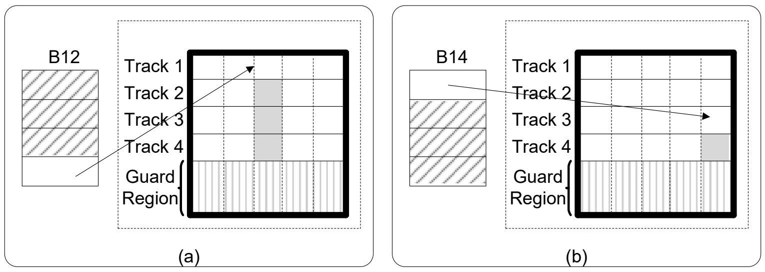


Figure 5.6: An example to show a different write-back cost of a stored LSN within its flash block. (a) writing back an LSN whose PSN resides in the outer track. (b) writing back an LSN whose PSN resides in the inner track.

block B12. The valid data is represented in white color and its native location in the SMR disk resides in the outermost track. Therefore, if writing back this valid data to the SMR disk, all the subsequent sectors (i.e., three sectors in gray color) will be affected as shown in Figure 5.6 (a). However, in Figure 5.6 (b), the same set of sectors S_5 are mapped into block

B14. The valid data in white color whose native location resides in the inner track, writing it back will only affect the following one sector in gray color.

Besides, the popularity of each LSN within the same set will be varied according to different workloads. The accessing count of each LSN is recorded in the S2P-map as mentioned in section 5.3.4. Without considering the write-back cost, the hotter data should be left in the flash cache while the colder data should be written back to make space for future incoming writes.

By taking both write-back cost and popularity into considerations as shown in Figure 5.7, the garbage collector will selectively choose those cold data with a low write-back cost to be written back at the first priority. The remaining ones will be rearranged back into the physical flash block.

5.4 Evaluation

In this Section, we will introduce the experimental setup in Section 5.4.1 and then present the results and discussion in Section 5.4.2.

5.4.1 Experimental Setup

We simulate a Seagate ST8000AS0011 8TB SMR disk based on the described description in [2]. Then, our flash-cache with the controller is built-in along with the SMR disk in the simulator. In our simulated disk, we use the following parameters: The write-head width is as 3 times large as the read-head width which indicates that writing to a track will affect the

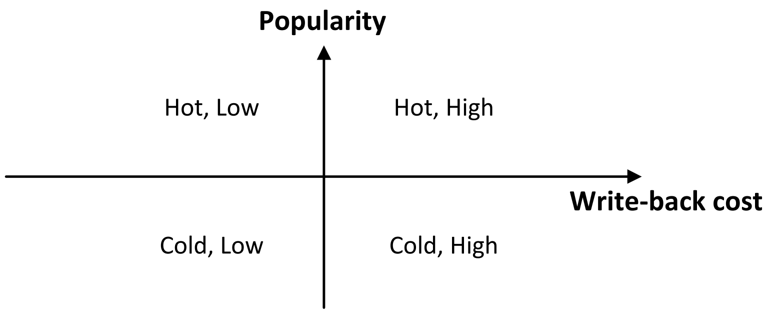


Figure 5.7: Classifying the LSN according to its write-back cost and data popularity.

Table 5.1: The characteristics of the traces.

Traces	Data Written (GB)	Data Read (GB)	Writes (%)	Updates (%)
rsrch_0	7.63	0.94	91.82%	96.34%
src1_2	33.48	2.76	83.70%	89.56%
src2_0	5.83	1.11	86.56%	94.13%
src2_2	32.63	20.68	68.21%	22.64%
stg_0	10.53	6.09	77.51%	94.46%
stg_1	5.47	10.94	20.88%	91.33%
ts_0	7.69	1.89	84.28%	95.81%
usr_0	11.06	15.42	61.41%	91.47%
web_0	13.17	10.78	63.32%	93.93%
web_2	1.33	45.21	1.88%	42.83%

subsequent two tracks. Thus, the guard region consists of two tracks which store no data so as to separate bands in the native data area. The rotation speed is set at 5900 rpm and the disk has 16 surfaces, thus, each surface has a capacity of 512 GB.

The management schemes of RMW-F are implemented within the controller. To perform a fair comparison, we implement the described scheme which utilizes NAND flash as cache (named emulated-SMR-2) in Skylight [2] as the baseline (we denote the baseline scheme as SMR-2 hereafter). Both RMW-F and SMR-2 have an identical flash cache size of 20GiB. The flash page size is 4KB which is of the same size as an SMR sector size (4KB) in our environment. We perform evaluations with real-world traces collected from data centers of Microsoft Research Cambridge [1]. The detailed characteristics of the traces including total data written/read to the SMR device (GB), write ratio (%) and update ratio (%) are shown in Table 5.1. The total number of requests of the traces is 1,000,000, in which the data written range from 1.33 GB to 33.48 GB and the data read range from 0.94 GB to 45.21 GB. Each trace is in a *hostname_disknumber* form. For example, the *stg* trace is for web staging and the disk number is 0.

5.4.2 Results and discussion

In this Section, we evaluate the effectiveness of RMW-F with the following performance metrics for both RMW-F and Skylight: 1) average response time; 2) read/write-head movements; 3) write amplification factor; 4) average cleaning time.

• **Average response time**

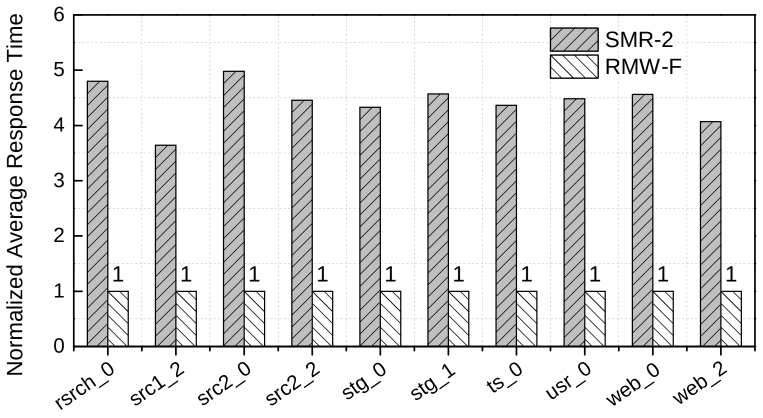


Figure 5.8: Normalized average response time.

In order to perform quantitative analysis on the overall system performance, we measure the average response time of both SMR-2 and RMW-F. The average response time is calculated as the ratio of total time spent on carrying out all the read/write requests over the total number of requests. The metric of SMR-2 is normalized to that of RMW-F and the results are shown in Figure 5.8. We can observe that RMW-F can shorten the overall average response time by over 79% when comparing to that of SMR-2. The performance improvement mainly comes from the significant reduction of write amplification factor (reduced RMWs). Besides, the cleaning process which takes both write-back cost and data popularity into account can also improve the overall system performance. Therefore, the result shows the effectiveness of RMW-F.

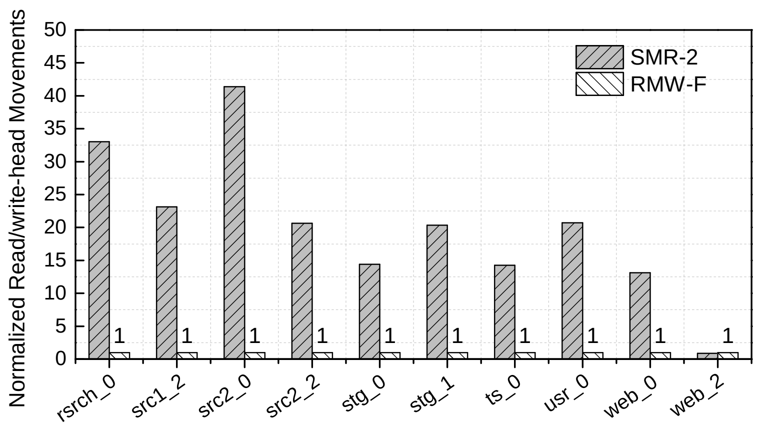


Figure 5.9: Normalized read/write-head movement.

• **Read/write-head movements**

In order to measure the cleaning efficiency at the SMR disk side, we have performed quantitative analysis on the read/write-head movements. During a cleaning, SMR-2 needs to clean up all the valid data in the first-level persistent cache to the SMR disk which incurs a large number of RMWs. Such RMWs keep the read/write head busy shifting between different tracks. Since every operation in the SMR disk requires the read/write-head to engage, the movements of head dominate the performance. RMW-F can eliminate the needs of RMWs while performing writes to the SMR system and only introduce very little RMWs during a cleaning process. Therefore, the read/write-head movements of RMW-F can be reduced significantly.

We normalized the read/write-head movements of SMR-2 to that of RMW-F and the results can be found in Figure 5.9. The maximum reduction is 41X and the average reduction is 16X. As a comparison, RMW-F performs a faster cleaning than SMR-2 since it benefits from the reduction in head movements.

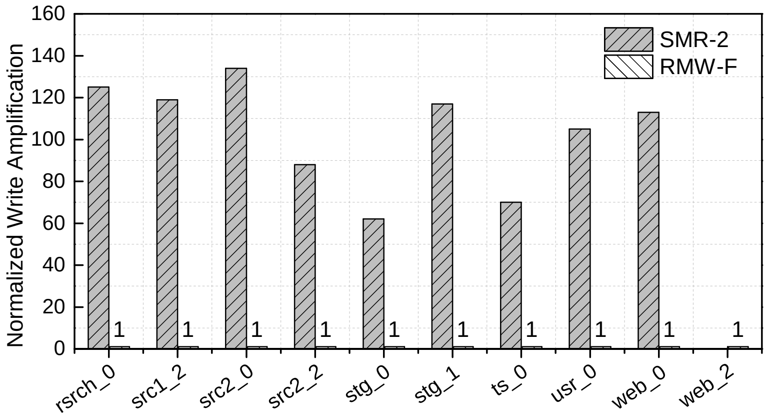


Figure 5.10: Normalized write amplification factor.

• Write amplification factor

As our dual-space management module mainly responsible for avoiding the read-modify-write operations so as to reduce write amplification factor (WAF), we measure the WAF of both SMR-2 and RMW-F in the experiment. We normalized the WAF of SMR-2 to that of RMW-F and the results can be found in Figure 5.10. The write-amplification of SMR-2 mainly comes from the RMWs introduced by the data write-back behaviors. Since RMW-F reduces the need of RMW when performing writes and can reduce write-back cost within the garbage collector module, the write-amplification factor is much smaller than that of SMR-2. As shown in Figure 5.10, the reduction of WAF ranges from 62 to 132 which shows the effectiveness of RMW-F.

• Average cleaning time

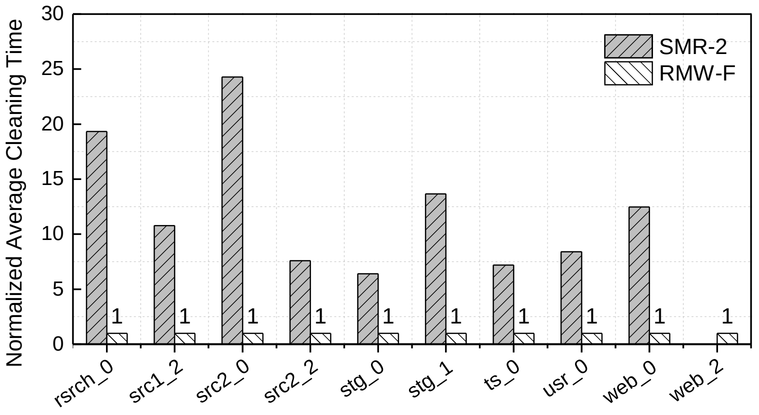


Figure 5.11: Normalized average cleaning time.

We use the average cleaning time to compare the cleaning efficiency of both SMR-2 and RMW-F. The average cleaning time is calculated as the total time spent on the cleaning processes over the total cleaning counts of the system. During a cleaning process, the system is paused and cannot serve incoming requests until a cleaning is finished. Therefore, the average cleaning time plays an important role in the user experience. We normalized the average cleaning time of SMR-2 to that of RMW-F and collected the results as shown in Figure 5.11.

From the results, we can observe that much more time is spent in a cleaning process of SMR-2 than that of RMW-F with an average ratio of 15.6X. Since RMW-F only performs cleaning at the unit of flash block, the cleaning scale is much smaller than that of SMR-2 and thus the pausing time is shorter. Besides, the reduction of the average cleaning time also shows the effectiveness of our write-back cost and hot/cold data popularity model.

5.5 Related Work

A lot of works have been done to improve the performance of SMR storage systems including the following three aspects: (1) SMR file system solutions, (2) SMR-based cache optimizations and (3) flash-based cache optimizations.

SMR File System Solutions. The file system layer which is in the middle of the system architecture between applications and SMR devices can be tailored and redesigned to provide support and optimizations. An effective file system called HiSMRfs mainly focus on improving the performance in the host by managing metadata and data storage separately on a hybrid system including SSDs and SMR disks [36]. By doing so, data writing to the SMR storage is ensured to be sequential and thus, no random writes are allowed to incur RMW operations. Another host-side file system named SFS is tailored for specific workloads like videos and implements different managerial policies for random write shingled zone and sequential write shingled zone [44]. ZEAFS [60] provides high-level abstractions for KV stores and builds an SMR compatible KV store system. The work in [4] adopts a legacy file system (EXT4) to DM-SMR drive to make it a promising SMR-friendly file system which requires little modifications on existing SMR devices.

SMR-based Cache Optimizations. Some works seek to reduce the random writes performance penalty by reconsidering the design of the SMR-based cache. A disk cache based architecture is proposed in [9] and an indirection buffer named S-blocks are managed to always map incoming data to the head of a circular buffer. A cleaning process is needed to move the still valid blocks back to the head when the buffer is full. E-region [29] is also a circular cache buffer that ensures data will not be destroyed by writes to support random writes in SMR devices.

The cleaning efficiency can be the system bottleneck of the SMR system. By using a mechanism named zone-oriented reordering, the cache cleaning efficiency can be improved greatly as mentioned in the work [90, 91]. Some work also considers performing data classification into hot data and cold data and then write them to different bands separately so as to improve cleaning efficiency [37]. By doing so, long-term data migration can be reduced. Another work in [46] also proposes to perform hot data identification and manage data in hot bands so as to reduce cleaning overhead.

Flash-based Cache Optimizations. Flash-based storage has been deployed to accelerate the performance of HDDs and SMR disks due to their promising behaviors and cost-effective characteristics. ROCO [49] proposes to adopt an SSD to improve the performance of a host-aware SMR drive. Since the optimization is mainly based on a host-aware SMR disk, writes can be classified with the assistance of host. However, our work focuses on drive-managed SMR disk and adopts a built-in NAND flash memory rather than an SSD and optimizations are done inside the SMR device and thus, our work is different from the ROCO work. The work in [81] proposes to use SSD as the first-level cache of the SMR device, that is, all incoming writes will be performed in the SSD first. By restricting the write range of logical block addresses (LBAs) of zones, [81] is able to reduce the write amplification factor to some extent. HS-BAS [92] also uses SSD as the first-level write buffer cache for SMR devices and focuses on reducing the cleaning/collection cost by adopting different policies. A hybrid wave-like shingled recording disk system (HWSR) proposes to use SSD as a disk cache to mainly improve random read performance rather than write performance [55, 80]. An application-aware hybrid storage system named (Apas), consists of SSDs and SMR disks, has taken the application characteristics into account to mitigate the write amplification problem.

5.6 Summary

In this chapter, we present the motivation, design, analysis, and evaluation of RMW-F. To avoid RMWs as many as possible, which is the major challenge of SMR disks, we propose a dual-space management scheme to distinguish writes according to their write-back behavior. A dual-space management module is implemented to handle write/read requests into the SMR system. The address translator is responsible for handling the mappings between the NAND flash and SMR addresses. The garbage collector adopts a heuristic scheme to determine whether a certain sector should be written back to the SMR disk or be left in the flash. Our experimental results show that RMW-F can shorten the overall system average response time by over 79% and improve the cleaning efficiency by 15.6 times.

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

SMR has recently been provided as a promising solution to satisfy the capacity requirement of big data applications. Compared with conventional HDDs, SMR disks benefit in higher density and lower cost. However, SMR disks also suffer from lower responding time due to their intrinsic overlapped-track limitation. That is, writing to a track leads to damage to the original data stored on the subsequent tracks. To prevent data loss, a time-consuming RMW is performed to firstly copy out the involved data and then to write them back after the modification is done. In this thesis, in order to avoid RMW effect, we optimize the SMR storage system in three main schemes, namely a decentralized approach without the cache-assistance and two cache optimizations by the integration of NAND flash and SMR disks.

For the first scheme, we first advocate reconsidering the drive-managed scheme and for the first time propose a decentralized approach called Tiler to manage the SMR disk space. Our basic idea is to separate the whole SMR disk space into individual log-structured ARs. We have built an SMR simulator and evaluated our proposed scheme with various real-world collected traces to demonstrate the effectiveness of this scheme. Our experimental results show that Tiler can shorten the overall system average response time by 49% and reduce the average cleaning time by 25X.

For the second scheme, to optimize the cache of the SMR disk, we propose a new cache management scheme called *Dual-buffer* to manage the SMR storage system. Different from conventional single-buffer-based schemes, *Dual-buffer* partitions the persistent cache into two separate buffers, namely the persistent buffer and the filter buffer, that are used to handle incoming data requests and to hold hot data, respectively. The basic idea is to keep hot data in the filter buffer as long as possible, instead of writing them back to their native locations during a cleaning operation. We have implemented the propose scheme inside the embedded controller of our SMR simulator and evaluated the effectiveness of the scheme with different traces. Experimental results show that our *Dual-buffer* scheme can shorten the average response time by 51.66% on average and reduce the total number of read-modify-write operations by 98.76% on average compared to the previous work.

For the third scheme, in order to eliminate the RMW effect and to accelerate the SMR storage system, we propose to integrate a promising cost-effect NAND flash as a persistent cache (namely RMW-F cache). Specially, for the writes to SMR disks, some writes will incur RMWs while the other writes will incur no RMWs. Therefore, we propose to distribute the writes that will incur RMWs (if written back) to the flash cache while the other writes are performed in the SMR disk directly. In this way, our design ensure that no RMWs are needed and thus the system performance can be improved. Our experimental results show that RMW-F can shorten the overall system average response time by over 79% and improve cleaning efficiency by approximately 15.6 times.

6.2 Future Work

The work presented in this thesis can be extended to different directions in the future as follows:

• Crash recovery is an important issue of drive-managed SMR devices since the mapping are dynamically mapped and the system mainly relies on the address translation to perform reads/writes. How to combine our schemes to effectively perform crash recovery can be a future direction for us to explore.

• Applying machine learning (ML) techniques to optimize the management of Dual-buffer will be an interesting direction. For example, ML techniques can be applied to locate an ideal partitioning point of the persistent buffer and the filter buffer.

• There are other possible solutions to partition the circular persistent cache, such as multi-log based caches. How to make the multi-log caching approach more adaptive with optimal partitioning can be an interesting problem for future research.

• Our proposed schemes are evaluated mainly on our built simulator, in the future works, we can explore and extend our works to the actual SMR device.

• We can combine our Dual-buffer and RMW-F schemes together for key-value stores in the SMR device. How to design the key-value SMR caching system can be an interesting direction in the future work.

• Our RMW-F scheme is mainly based on flash-based hardware. We will extend our approach to other emerging non-volatile-memories (NVMs) to further improve the SMR storage system performance.

• New challenges and problems will arise by adopting SMR devices as RAID. How to address these issues to effectively deploy SMRs as RAID will be a future direction of our works.

REFERENCES

[1] MSR Cambridge Block I/O Traces. http://iotta.cs.hmc.edu/traces/388.

[2] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. Skylight:a window on shingled disk operation. *ACM Transactions on Storage (TOS)*, 11(4):16, 2015.

[3] Abutalib Aghayev, Y Theodore, Garth Gibson, and Peter Desnoyers. Evolving ext4 for shingled disks. In *15th USENIX Conference on File and Storage Technologies (FAST)*, pages 105–120, 2017.

[4] Abutalib Aghayev, Theodore Tso, Garth Gibson, and Peter Desnoyers. Evolving ext4 for shingled disks. In *FAST*, pages 105–120, 2017.

[5] Ahmed Amer, JoAnne Holliday, Darrell DE Long, Ethan L Miller, Jehan-François Pâris, and Thomas Schwarz. Data management and layout for shingled magnetic recording. *IEEE Transactions on Magnetics*, 47(10):3691–3697, 2011.

[6] Ahmed Amer, Darrell DE Long, Ethan L Miller, Jehan-Francois Paris, and SJ Thomas Schwarz. Design issues for a shingled write disk system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–12. IEEE, 2010.

[7] Nadia Awad. On reducing the decoding complexity of shingled magnetic recording system. *University of Plymouth*, 2013.

[8] Jorge Campello. Smr: The next generation of storage technology. In *Storage Development Conference, SNIA, Santa Clara, CA*, 2015.

[9] Yuval Cassuto, Marco AA Sanvido, Cyril Guyot, David R Hall, and Zvonimir Z Bandic. Indirection systems for shingled-recording disk drives. In *Mass Storage Sys-*

tems and Technologies (MSST), 2010 IEEE 26th Symposium on, pages 1–14. IEEE, 2010.

[10] Li-Pin Chang, Yu-Syun Liu, and Wen-Huei Lin. Stable greedy: Adaptive garbage collection for durable page-mapping multichannel SSDs. *ACM Trans. Embed. Comput. Syst.*, 15(1):13:1–13:25, January 2016.

[11] Renhai Chen, Zhiwei Qin, Yi Wang, Duo Liu, Zili Shao, and Yong Guan. On-demand block-level address mapping in large-scale NAND flash storage systems. *IEEE Transactions on Computers*, 64(6):1729–1741, June 2015.

[12] Renhai Chen, Zhaoyan Shen, Chenlin Ma, Zili Shao, and Yong Guan. Nvmra: utilizing nvm to improve the random write operations for nand-flash-based mobile devices. *Software: Practice and Experience*, 46(9):1263–1284, 2016.

[13] Renhai Chen, Yi Wang, Jingtong Hu, Duo Liu, Zili Shao, and Yong Guan. Unified non-volatile memory and nand flash memory architecture in smartphones. In *The 20th Asia and South Pacific Design Automation Conference*, pages 340–345. IEEE, 2015.

[14] Renhai Chen, Yi Wang, Duo Liu, Zili Shao, and Song Jiang. Heating dispersal for self-healing nand flash memory. *IEEE Transactions on Computers*, 66(2):361–367, 2017.

[15] Shuo Han Chen, Wei Shin Li, Min Hong Shen, Yi Han Lien, Tseng Yi Chen, Tsan Sheng Hsu, Hsin Wen Wei, and Wei Kuan Shih. An update-overhead-aware caching policy for write-optimized file systems on smr disks. In *IEEE International Performance Computing Communications Conference*, 2017.

[16] K. Chooruang and M. M. Aziz. Experimental studies of shingled recording using contact recording test system. In *International Conference on Electrical Engineering/electronics*, 2013.

[17] Benixon Arul Dhas, Erez Zadok, James Borden, and Jim Malina. Evaluation of nilfs2 for shingled magnetic recording (smr) disks. *Stony Brook University, Tech. Rep. FSL-14-03*, 2014.

[18] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 1905.

[19] Moulay Rachid Elidrissi, Kheong Sann Chan, and Zhimin Yuan. A study of smr/tdmr with a double/triple reader head array and conventional read channels. *IEEE Transactions on Magnetics*, 50(3):24–30, 2014.

[20] Tim Feldman and Garth Gibson. Shingled magnetic recording: Areal density increase requires new data management. *USENIX; login: Magazine*, 38(3), 2013.

[21] Björn Forsberg, Andrea Marongiu, and Luca Benini. GPUguard: Towards supporting a predictable execution model for heterogeneous SoC. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 318–321, 2017.

[22] Garth Gibson and Greg Ganger. Principles of operation for shingled disk devices. In *3rd USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, pages 1–5, 2011.

[23] Garth Gibson and Milo Polte. Directions for shingled-write and twodimensional magnetic recording system architectures: Synergies with solid-state disks. *Parallel Data Lab, Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-PDL-09-014*, 2009.

[24] Steven Granz, Jason Jury, Chris Rea, Ganping Ju, Jan Ulrich Thiele, Tim Rausch, and Edward C. Gage. Areal density comparison between conventional, shingled, and interlaced heat-assisted magnetic recording with multiple sensor magnetic recording. *IEEE Transactions on Magnetics*.

[25] Steven Granz, Tue Ngo, Tim Rausch, Richard Brockie, Roger Wood, Gerardo Bertero, and Edward Gage. Definition of an areal density metric for magnetic recording systems. *IEEE Transactions on Magnetics*, 53(2):1–4, 2017.

[26] Simon Greaves, Yasushi Kanai, and Hiroaki Muraoka. Shingled recording for 2–3 tbit/in^2. *IEEE Transactions on Magnetics*, 45(10):3823–3829, 2009.

[27] Simon John Greaves, Yasukazu Kanai, and Hiroaki Muraoka. Shingled thermally assisted magnetic recording for 8 tbit/in. *Magnetics IEEE Transactions on*, 50(11):1–4, 2014.

[28] Yong Guan, Guohui Wang, Chenlin Ma, Renhai Chen, Yi Wang, and Zili Shao. A block-level log-block management scheme for mlc nand flash memory storage systems. *IEEE Transactions on Computers*, 66(9):1464–1477, 2017.

[29] David Hall, John H Marcos, and Jonathan D Coker. Data handling algorithms for autonomous shingled magnetic recording hdds. *IEEE Transactions on Magnetics*, 48(5):1777–1781, 2012.

[30] Lei Han, Zhaoyan Shen, Zili Shao, and Tao Li. Optimizing raid/ssd controllers with lifetime extension for flash-based ssd array. In *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 44–54. ACM, 2018.

[31] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A Chien, and Haryadi S Gunawi. The tail at store: A revelation from millions of hours of disk and {SSD} deployments. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pages 263–276, 2016.

[32] Weiping He and David HC Du. Novel address mappings for shingled write disks. In *HotStorage*, 2014.

[33] Weiping He and David HC Du. SMaRT: An approach to shingled magnetic recording translation. In *FAST*, pages 121–134, 2017.

[34] Sheng-Min Huang and Li-Pin Chang. Exploiting page correlations for write buffering in page-mapping multichannel SSDs. *ACM Trans. Embed. Comput. Syst.*, 15(1):12:1–12:25, January 2016.

[35] Yichen Jia, Zili Shao, and Feng Chen. Slimcache: Exploiting data compression opportunities in flash-based key-value caching. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 209–222. IEEE, 2018.

[36] Chao Jin, Wei-Ya Xi, Zhi-Yong Ching, Feng Huo, and Chun-Teck Lim. Hismrfs: A high performance file system for shingled storage array. In *Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on*, pages 1–6. IEEE, 2014.

[37] Stephanie N Jones, Ahmed Amer, Ethan L Miller, Darrell DE Long, Rekha Pitchumani, and Christina R Strong. Classifying data to reduce long-term data movement in shingled write disks. *ACM Transactions on Storage (TOS)*, 12(1):2, 2016.

[38] Saurabh Kadekodi, Swapnil Pimpale, and Garth A Gibson. Caveat-scriptor: Write anywhere shingled disks. In *HotStorage*, 2015.

[39] Yimei Kang, Xingyu Zhang, Zili Shao, Renhai Chen, and Yi Wang. A reliability enhanced video storage architecture in hybrid slc/mlc nand flash memory. *Journal of Systems Architecture*, 88:33–42, 2018.

[40] Taeho Kgil, David Roberts, and Trevor Mudge. Improving nand flash based disk caches. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 327–338. IEEE Computer Society, 2008.

[41] Seungwon Kim, SangGi Do, and Seokhyeong Kang. Fast predictive useful skew methodology for timing-driven placement optimization. In *54th Design Automation Conference (DAC)*, pages 1–6, 2017.

[42] SPM Chi-Young Ku and Stephen P Morgan. An smr-aware append-only file system. In *Storage Developer Conference, Santa Clara, CA, USA*, 2015.

[43] Quoc Minh Le, Ahmed Amer, and JoAnne Holliday. Smr disks for mass storage systems. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 228–231. IEEE, 2015.

[44] Damien Le Moal, Zvonimir Bandic, and Cyril Guyot. Shingled file system host-side management of shingled magnetic recording disks. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 425–426. IEEE, 2012.

[45] Yongkun Li, Biaobiao Shen, Yubiao Pan, Yinlong Xu, Zhipeng Li, and John CS Lui. Workload-aware elastic striping with hot data identification for SSD raid arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(5):815–828, 2017.

[46] Chung-I Lin, Dongchul Park, Weiping He, and David HC Du. H-SWD: Incorporating hot data identification into shingled write disks. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 321–330. IEEE, 2012.

[47] Duo Liu, Tianzheng Wang, Yi Wang, Zhiwei Qin, and Zili Shao. A block-level flash memory management scheme for reducing write activities in pcm-based embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1447–1450. EDA Consortium, 2012.

[48] Duo Liu, Lei Yao, Linbo Long, Zili Shao, and Yong Guan. A workload-aware flash translation layer enhancing performance and lifespan of tlc/slc dual-mode flash memory in embedded systems. *Microprocessors and Microsystems*, 52:343–354, 2017.

[49] Wen Guo Liu, Ling Fang Zeng, Dan Feng, and Kenneth B. Kent. Roco: Using a solid state drive cache to improve the performance of a host-aware shingled magnetic recording drive. *Journal of Computer Science and Technology*, 34(1):61–76, 2019.

[50] Wenguo Liu, Dan Feng, Lingfang Zeng, and Jianxi Chen. Understanding the SWD-based RAID system. In *Cloud Computing and Big Data (CCBD), 2014 International Conference on*, pages 175–181. IEEE, 2014.

[51] Wenguo Liu, Lingfang Zeng, and Feng Dan. Apas: An application aware hybrid storage system combining ssds and swds. In *IEEE International Conference on Networking*, 2016.

[52] Wenguo Liu, Lingfang Zeng, and Dan Feng. Cass: A cooperative hybrid storage system consisting of an ssd and a smr drive. In *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, 2018.

[53] Wenguo Liu, Lingfang Zeng, and Dan Feng. Cldm: A cache cleaning algorithm for host aware smr drives. In *International Conference on Algorithms and Architectures for Parallel Processing*, 2018.

[54] Dan Luo, Jiguang Wan, Yifeng Zhu, Nannan Zhao, Feng Li, and Changsheng Xie. Design and implementation of a hybrid shingled write disk system. *IEEE Transactions on Parallel and Distributed Systems*, 27(4):1017–1029, 2016.

[55] Dan Luo, Jiguang Wan, Yifeng Zhu, Nannan Zhao, Feng Li, and Changsheng Xie. Design and implementation of a hybrid shingled write disk system. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, pages 1017–1029, 2016.

[56] Chenlin Ma, Zhaoyan Shen, Yi Wang, and Zili Shao. Alleviating hot data write back effect for shingled magnetic recording storage systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[57] Liuying Ma and Xu Lu. Hmss: A high performance host-managed shingled storage system based on awareness of smr on block layer. In *IEEE International Conference on High Performance Computing Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science Systems*, 2017.

[58] Peter Macko, Xiongzi Ge, J Kelley, D Slik, et al. SMORE: A cold data object store for SMR drives. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST)*, pages 1–13, 2017.

[59] Adam Manzanares, Noah Watkins, Cyril Guyot, Damien Le Moal, Carlos Maltzahn, and Zvonimir Bandic. ZEA, a data management approach for SMR. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, pages 1–5, 2016.

[60] Adam Manzanares, Noah Watkins, Cyril Guyot, Damien LeMoal, Carlos Maltzahn, and Zvonimr Bandic. Zea, a data management approach for smr. In *HotStorage*, 2016.

[61] George Mathew, Euiseok Hwang, Jongseung Park, Glen Garfunkel, and David Hu. Capacity advantage of array-reader-based magnetic recording (armr) for next generation hard disk drives. *IEEE Transactions on Magnetics*, 50(3):155–161, 2014.

[62] Junpeng Niu, Mingzhou Xie, Jun Xu, Lihua Xie, and Xia Li. Smr drive performance analysis under different workload environments. *Control Engineering Practice*, 75:8697, 2018.

[63] Junpeng Niu, Jun Xu, and Lihua Xie. Analytical modeling of smr drive under different workload environments. In *2017 13th IEEE International Conference on Control & Automation (ICCA)*, pages 1113–1118. IEEE, 2017.

[64] Junpeng Niu, Jun Xu, and Lihua Xie. A deep look at smr performance via simulation approach. In *2017 13th IEEE International Conference on Control & Automation (ICCA)*, pages 713–718. IEEE, 2017.

[65] Tiratat Patana-anake, Vincentius Martin, Nora Sandler, Cheng Wu, and Haryadi S Gunawi. Manylogs: Improved cmr/smr disk bandwidth and faster durability with scattered logs. In *Mass Storage Systems and Technologies (MSST), 2016 32nd Symposium on*, pages 1–16. IEEE, 2016.

[66] Rekha Pitchumani. Data management for shingled magnetic recording disks. *Dissertations Theses - Gradworks*, 2015.

[67] Rekha Pitchumani, Andy Hospodor, Ahmed Amer, Yangwook Kang, Ethan L Miller, and Darrell DE Long. Emulating a shingled write disk. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 339–346. IEEE, 2012.

[68] M. Salo, T. Olson, R. Galbraith, R. Brockie, B. Lengsfield, H. Katada, and Y. Nishida. The structure of shingled magnetic recording tracks. *IEEE Transactions on Magnetics*, 50(3):18–23, 2014.

[69] Mohit Saxena, Michael M Swift, and Yiying Zhang. Flashtier: a lightweight, consistent and durable storage cache. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 267–280. ACM, 2012.

[70] Sudipta Sengupta, Jin Li, Cheng Huang, Timothy Andrew Pritchett, and Christopher Broder Wilson. Multi-tiered cache with storage medium awareness, 2013. US Patent App. 13/531,455.

[71] Mansour Shafaei, Mohammad Hossein Hajkazemi, Peter Desnoyers, and Abutalib Aghayev. Modeling drive-managed smr performance. *ACM Transactions on Storage (TOS)*, 13(4):38, 2017.

[72] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. Optimizing flash-based key-value cache systems. In *8th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

[73] Zhaoyan Shen, Feng Chen, Yichen Jia, and Zili Shao. Didacache: An integration of device and application for flash-based key-value caching. *ACM Transactions on Storage (TOS)*, 14(3):26, 2018.

[74] Liang Shi, Kaijie Wu, Mengying Zhao, C.J. Xue, Duo Liu, and E.H. Sha. Retention trimming for lifetime improvement of flash memory storage systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1):58–71, Jan 2016.

[75] Anand Suresh, Garth Gibson, and Greg Ganger. Shingled magnetic recording for big data applications. *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PD L-12–105*, 2012.

[76] R. Suzutou, Y. Nakamura, M. Nishikawa, H. Osawa, Y. Okamoto, Y. Kanai, and H. Muraoka. A study on relationship between recording pattern and decoding reliability in smr. *IEEE Transactions on Magnetics*, PP(99):1–1, 2017.

[77] Sophia Tan, Weiya Xi, Zhi Yong Ching, Chao Jin, and Chun Teck Lim. Simulation for a shingled magnetic recording disk. In *2012 Digest APMRC*, pages 1–7. IEEE, 2012.

[78] Kim Keng Teo, Moulay Rachid Elidrissi, Kheong Sann Chan, and Yasushi Kanai. Analysis and design of shingled magnetic recording systems. *Journal of Applied Physics*, 111(7):657–01, 2012.

[79] Kalyana Sundaram Venkataraman, Tong Zhang, Wenzhe Zhao, Hongbin Sun, and Nanning Zheng. Scheduling algorithms for handling updates in shingled magnetic recording. In *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on*, pages 205–214. IEEE, 2013.

[80] Jiguang Wan, Nannan Zhao, Yifeng Zhu, Jibin Wang, Yu Mao, Peng Chen, and Changsheng Xie. High performance and high capacity hybrid shingled-recording disk system. In *CLUSTER*, 2012.

[81] Chunling Wang, Dandan Wang, Yupeng Chai, and D Sun. Larger cheaper but faster: SSD-SMR hybrid storage boosted by a new SMR-oriented cache framework. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST)*, pages 1–16, 2017.

[82] Chunling Wang, Dandan Wang, Yupeng Chai, Chuanwen Wang, and Diansen Sun. Larger, cheaper, but faster: Ssd-smr hybrid storage boosted by a new smr-oriented cache framework. In *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.(MSST)*, 2017.

[83] Jun Wang and Yiming Hu. PROFS-performance-oriented data reorganization for log-structured file system on multi-zone disks. In *Modeling, Analysis and Simulation of*

Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on, pages 285–292. IEEE, 2001.

[84] Rui Wang, Qun Guo, Yixue Zhu, and Michael E Habben. De-duplication and completeness in multi-log based replication, 2012. US Patent 8,108,343.

[85] Yi Wang, Lisha Dong, Zhong Ming, Yong Guan, and Zili Shao. Virtual duplication and mapping prefetching for emerging storage primitives in nand flash memory storage systems. *Microprocessors and Microsystems*, 50:54–65, 2017.

[86] Yi Wang, Zhiwei Qin, Renhai Chen, Zili Shao, and Laurence T Yang. An adaptive demand-based caching mechanism for nand flash memory storage systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(1):18, 2016.

[87] Yi Wang, Zili Shao, Henry CB Chan, Luis Angel D Bathen, and Nikil D Dutt. A reliability enhanced address mapping strategy for three-dimensional (3-d) nand flash memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(11):2402–2410, 2014.

[88] Roger Wood, Mason Williams, Aleksandar Kavcic, and Jim Miles. The feasibility of magnetic recording at 10 terabits per square inch on conventional media. *IEEE Transactions on Magnetics*, 45(2):917–923, 2009.

[89] Chun-Feng Wu, Ming-Chang Yang, and Yuan-Hao Chang. Improving runtime performance of deduplication system with Host-Managed SMR storage drives. In *Proceedings of the 55th Annual Design Automation Conference*, page 57. ACM, 2018.

[90] Fenggang Wu, Ziqi Fan, Ming-Chang Yang, Baoquan Zhang, Xiongzi Ge, and David HC Du. Performance evaluation of host aware shingled magnetic recording (HA-SMR) drives. *IEEE Transactions on Computers*, 66(11):1932–1945, 2017.

[91] Fenggang Wu, Ming-Chang Yang, Ziqi Fan, Baoquan Zhang, Xiongzi Ge, and David HC Du. Evaluating host aware smr drives. In *HotStorage*, 2016.

[92] Wenjian Xiao, Huanqing Dong, Liuying Ma, Zhenjun Liu, and Qiang Zhang. HS-BAS: A hybrid storage system based on band awareness of shingled write disk. In *ICCD*, pages 64–71, 2016.

[93] Wenjian Xiao, Huanqing Dong, Liuying Ma, Zhenjun Liu, and Qiang Zhang. HS-BAS: A hybrid storage system based on band awareness of shingled write disk. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 64–71, 2016.

[94] Xuchao Xie, Liquan Xiao, Xiongzi Ge, and Qiong Li. Smrc: An endurable ssd cache for host-aware shingled magnetic recording drives. *IEEE Access*, 6(99):20916–20928, 2018.

[95] Ming-Chang Yang, Yuan-Hao Chang, Fenggang Wu, Tei-Wei Kuo, and David HC Du. Virtual persistent cache: Remedy the long latency behavior of host-aware shingled magnetic recording drives. In *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*, pages 17–24. IEEE, 2017.

[96] Ming-Chang Yang, Yuan-Hao Chang, Fenggang Wu, Tei-Wei Kuo, and David HC Du. On improving the write responsiveness for Host-Aware SMR drives. *IEEE Transactions on Computers*, 2018.

[97] Qing Yang and Jin Ren. I-cash: Intelligently coupled array of ssd and hdd. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 278–289. IEEE, 2011.

[98] Tianming Yang, Haitao Wu, Huang Ping, and Zhang Fei. A shingle-aware persistent cache management scheme for dm-smr disks. In *IEEE International Conference on Computer Design*, 2017.

[99] Lei Yao, Duo Liu, Kan Zhong, Linbo Long, and Zili Shao. Tlc-ftl: Workload-aware flash translation layer for tlc/slc dual-mode flash memory in embedded systems. In *2015 IEEE 17th International Conference on High Performance Computing and Communications*, pages 831–834. IEEE, 2015.

[100] Ting Yao, Zhihu Tan, Jiguang Wan, Huang Ping, Yiwen Zhang, Changsheng Xie, and Xubin He. A set-aware key-value store on shingled magnetic recording drives with dynamic band. In *IEEE International Parallel Distributed Processing Symposium*, 2018.

[101] Jiang Yu, Chun Tung Chou, ZongKai Yang, Xu Du, and Tai Wang. A dynamic caching algorithm based on internal popularity distribution of streaming media. *Multimedia Systems*, 12(2):135–149, 2006.

[102] Lingfang Zeng, Zehao Zhang, Yang Wang, Dang Feng, and Kenneth B Kent. CosaFS: A cooperative shingle-aware file system. *ACM Transactions on Storage (TOS)*, 13(4):1–23, 2017.

[103] Chi Zhang, Yi Wang, Tianzheng Wang, Renhai Chen, Duo Liu, and Zili Shao. Deterministic crash recovery for nand flash based storage systems. In *Proceedings of the 51st Annual Design Automation Conference (DAC)*, pages 1–6, 2014.

[104] Grace Li Zhang, Bing Li, and Ulf Schlichtmann. Effitest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers. In *53rd Design Automation Conference (DAC)*, pages 1–6, 2016.