FluidSMR: Adaptive Management for Hybrid SMR Drives

FENGGANG WU, University of Minnesota, Twin Cities, USA
BINGZHE LI, Oklahoma State University, USA
DAVID H. C. DU, University of Minnesota, Twin Cities, USA

Hybrid Shingled Magnetic Recording (H-SMR) drives are the most recently developed SMR drives, which allow dynamic conversion of the recording format between Conventional Magnetic Recording (CMR) and SMR on a single disk drive. We identify the unique opportunities of H-SMR drives to manage the tradeoffs between performance and capacity, including the possibility of adjusting the SMR area capacity based on storage usage and the flexibility of dynamic data swapping between the CMR area and SMR area.

We design and implement FluidSMR, an adaptive management scheme for hybrid SMR Drives, to fully utilize H-SMR drives under different workloads and capacity usages. FluidSMR has a two-phase allocation scheme to support a growing usage of the H-SMR drive. The scheme can intelligently determine the sizes of the CMR and the SMR space in an H-SMR drive based on the dynamic changing of workloads. Moreover, FluidSMR uses a cache in the CMR region, managed by a proposed loop-back log policy, to reduce the overhead of updates to the SMR region.

Evaluations using enterprise traces demonstrate that FluidSMR outperforms baseline schemes in various workloads by decreasing the average I/O latency and effectively reducing/controlling the performance impact of the format conversion between CMR and SMR.

CCS Concepts: • **Information systems** → **Record and block layout**; **Magnetic disks**; *Storage management;*

Additional Key Words and Phrases: Hybrid SMR drive, data management, format conversion

ACM Reference format:
Fenggang Wu, Bingzhe Li, and David H. C. Du. 2021. FluidSMR: Adaptive Management for Hybrid SMR Drives. *ACM Trans. Storage* 17, 4, Article 32 (October 2021), 30 pages.
https://doi.org/10.1145/3465404

1 INTRODUCTION

With the ever-increasing demand for storage capacity, **Shingled Magnetic Recording (SMR)** drives were introduced in which consecutive disk tracks are written overlapped in a shingled fashion to achieve a higher areal density when compared with **Conventional Magnetic Recording (CMR)** drives. However, the increased capacity of SMR comes with a price that updating these

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439622, 1525617, 1536447, and 1812537.

Authors' addresses: F. Wu and D. H. C. Du, University of Minnesota, 200 Union St SE, Minneapolis, Minnesota 55455; emails: {wuxx0835, du}@umn.edu; B. Li, Oklahoma State University, 215 General Academic Building, Stillwater, Oklahoma 74078; email: bingzhe.li@okstate.edu.

1553-3077/2021/10-ART32 $15.00
https://doi.org/10.1145/3465404

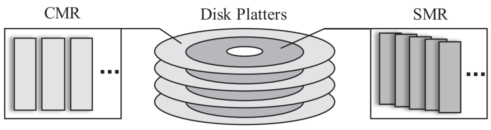

Fig. 1. H-SMR Drive.

SMR tracks will cause expensive *read-modify-write overhead* (*SMR update overhead*). To reduce *SMR update overhead* and have predictable performance, three types of SMR drives are developed, including Drive-Managed, Host-Managed, and Host-Aware drives. However, the tradeoffs between performance and space capacity in SMR drives remain to be managed.

Recently, Google introduced the idea of **Hybrid SMR (H-SMR)** drives [20]. As shown in Figure 1, a single H-SMR drive can have both CMR and SMR areas, and the recording format can be converted from one type to the other using H-SMR APIs. Both Seagate and Western Digital plan to produce H-SMR drives [6, 15]. H-SMR enables a promising way of combining and exploring the benefits of both CMR and SMR formats. To the best of our knowledge, other than the preliminary version of this article [52], there is no existing work specially addressing the data and space management issues of H-SMR drives.

We identify unique opportunities in H-SMR drives to strike favorable tradeoffs between space capacity and performance. First, unlike pure SMR and CMR drives, H-SMR drives can adjust the proportion of SMR and CMR areas based on storage capacity usage. When a drive is less than 100% full, some of the disk areas can be formatted into CMR to avoid the SMR update overhead. Second, when there is a mix of CMR and SMR areas in the same drive, hot (frequently updated) data can be dynamically swapped to the CMR area to alleviate the SMR update overhead. Third, different from the existing SMR implementations that use a fixed configuration of the on-disk cache, CMR space, and SMR space [1, 51, 54], H-SMR has the flexibility to adaptively adjust the size of each of such disk components to fit the changing workload.

However, there are still many challenges before we can leverage these opportunities of H-SMR. First, a growing capacity requires converting some disk area from CMR into SMR. Such a format conversion will destroy the existing data in the disk area to be converted. Therefore, these data need to be migrated before the conversion. Such data migration will degrade the I/O performance of on-going applications. Second, to reduce the SMR update overhead, ideally frequently updated data (*hot data*) should be placed in the CMR area, while less frequently or no updated data (*cold data*) are more suitable to be stored in the SMR area. Designing effective criteria for performing data relocation and zone format conversion is challenging, considering the SMR-specific zone updating and data migrating cost. Third, it is difficult to determine a favorable configuration of the on-disk cache, CMR space capacity, and SMR space capacity, given the intricate relations between the CMR area, SMR area, and the capacity usage. The dynamically changing workload further complicates the design choices about when/how to perform the data migration and zone format conversion.

In this article, we propose *FluidSMR* to address all of the challenges above. First, FluidSMR supports the growing usage by a two-phase allocation scheme that distinguishes high and low usage to conduct allocation differently. We propose a proactive and quantized migration scheme in FluidSMR, which carries out the data migration beforehand during idle time to further reduce the performance impact of format conversion and valid data migration. Second, by formalizing and solving the H-SMR format configuration problem, we design an adaptive format conversion algorithm in FluidSMR that can intelligently adjust the format of zones and the sizes of the CMR, SMR, and Cache space capacities and identify an ideal format configuration best fitted the changing workload. Finally, FluidSMR uses a proposed *loop-back log* policy to manage a CMR cache, and a novel *cache-occupancy*-based algorithm to manage the zone swapping.

Evaluations using **Microsoft Research (MSR)** Cambridge traces [37] show that the proposed FluidSMR outperforms several baseline schemes by reducing the average I/O latency in various workloads with different disk usages. It can also decrease the space re-allocation response time and limit the application I/O's performance during the space re-allocation.

The rest of the article is organized as follows. Section 2 provides the background and motivates our design. We give an overview of FluidSMR in Section 3. Section 4 describes how FluidSMR handles a growing space usage with a proposed two-phase allocation scheme. The adaptive H-SMR format configuration algorithm is presented in Section 5. The CMR cache replacement algorithm and the occupancy-based zone-swapping scheme are discussed in Section 6. Evaluation results are shown in Section 7. We summarize the related works in Section 8 and conclude the article in Section 9.

2 BACKGROUND AND MOTIVATION

2.1 SMR Background

The increase of areal data density in SMR comes with the cost of amplified updates on existing SMR data tracks. When updating data on a shingled track, the data on one of its adjacent tracks may have to be read out and re-written back to avoid data loss. Re-writing the adjacent tracks will destroy the data tracks further away, creating a domino effect (considered as *SMR update overhead*) with data on many adjacent tracks to be updated [51, 54]. Therefore, to reduce this domino effect, SMR tracks are partitioned into *zones* (typically 256 MB) with guard tracks introduced between zones. As a result, any update on one zone can be independently handled without interfering data on other zones. Nevertheless, updating existing data in one SMR zone will still cause SMR update overhead within the zone.

Traditionally, there are three models of SMR drives: drive-managed, host-aware, and host-managed [26, 27]. Drive-managed SMR drives can be used as a "drop-in" replacement for CMR disks by hiding all SMR features behind a mapping layer (Shingled Translation Layer). By contrast, host-aware and host-managed SMR drives expose zone information (such as the *write pointers*) to the host so that the host can query such information from the drive to make better decisions about where to issue write requests. The *write pointer* of a zone is the LBA location where the next write should go. This write pointer can be considered as a log head pointer in a sense that one SMR zone is abstracted into a log-structured model. A write request targeting at the write pointer is considered as a *sequential write*. Data will be written to the disk begin with the current write pointer location, and the write pointer will be advanced accordingly after the write. Otherwise, a write operation not beginning at the write pointer is called a **non-sequential write (NSW)**.

SMR zones in a host-aware SMR drive are *sequential write preferred zones*. When an NSW comes to such type of zones, the data will be redirected to an on-disk SMR area called Media Cache in a log-structured manner. Media Cache has a fixed location and capacity. When the Media Cache capacity is used up, the redirected data will have to be migrated back to its targeted locations with a read-modified-write manner (called blocking cleaning). Such read-modify-write operations introduce many extra IOs, resulting in a severe *SMR update overhead* and deficient performance during cleaning. In a host-managed SMR drive, the SMR zones are *sequential write required* zones. The host should ensure the write aims at the write pointers of such zones or else the write will be rejected. Therefore, when an NSW comes, if the NSW targets on a location before the write pointer, then the host should read the zone data, write the zone data back combined with the change. If the NSW aims to an LBA after the write pointer, then the host should perform extra effort, including

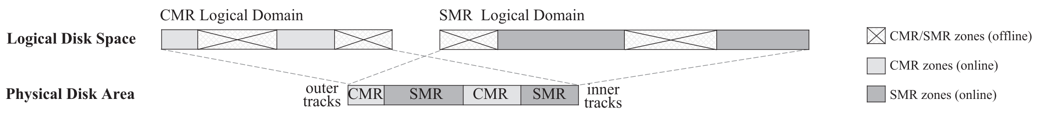

Fig. 2. Logical and physical view of Hybrid SMR drive. Only online CMR/SMR zones are backed with physical zones, so that they can accept I/O requests from the host.

filling in garbage between the write pointer and the beginning address of the NSW and updating the write pointer. Such a read-modify-write operation introduces a lot of extra IOs, also resulting in serious *SMR update overhead*. Both host-aware and host-managed SMR drives can optionally have a fixed CMR area.

H-SMR drives are very close to the host-aware and host-managed types, since they enable hosts to take action on SMR drives based on available APIs. These SMR drives gain increasing popularity and adoption in both academia and industry [13, 33–35, 51, 54, 60, 61]. SMR zones in H-SMR drive can be either the sequential write preferred or required zones [44].

2.2 Hybrid SMR Preliminaries

Compared with SMR drives, CMR drives have better performance as it has no SMR update overhead. However, SMR drives have a higher areal density where more data can be stored in the same physical disk area to reduce the Total Cost of Ownership. To better address the tradeoffs between space capacity and performance, a new SMR model is proposed recently, namely H-SMR [6, 15, 16, 20, 42, 46], where one drive can have both CMR and SMR formats and can dynamically convert between these two formats based on workload and space requirements.

According to Reference [15] (Figure 2), the host can convert any number of zones from one format to the other format using H-SMR APIs, which take an extent of zones from one format offline and bring the corresponding zones in the other format online. Here "online" means the zone is backed with the *physical disk area* and vice versa. Only online zones can accept I/O requests from the host while offline zones cannot. Due to the different areal densities of CMR and SMR, the number of zones will usually be different before and after the conversion. For example, supposing an SMR to CMR areal density ratio is 1.5:1 (i.e., an *SMR density gain* of 1.5× compared with that of SMR), 2 GB of CMR space (8 zones) can be converted into 3 GB of SMR space (i.e., 12 zones). Note that these conversion APIs will destroy the stored data in the zones being converted. Thus the host is responsible for protecting the valid data in these zones by migrating the data first to other persistent locations (*valid data migration*).

2.3 H-SMR Opportunities and Challenges

Dynamic Formatting Based on the Growing Usage - Different from the existing SMR products, the H-SMR drive has the flexibility to convert a portion of CMR area into SMR or SMR back to CMR depending on the capacity usage. Such flexibility gives an excellent opportunity to optimize performance when the capacity usage is not high (less than 100% full), because part of the disk can be formatted into CMR that is free of the SMR update overhead. When the usage increases, more disk areas will be gradually converted to SMR to achieve a higher space capacity. However, there are two challenges, along with this opportunity: how to design the data layout and mapping to support a growing space usage and how to reduce/control the performance overhead due to valid data migration when increasing storage capacity is necessary.

Dynamic Data Placement. When there is a mix of SMR and CMR areas on the disk, we have the opportunity of putting frequently updated data (*hot data*, hereafter) in the CMR area, while allocating update-light data (*cold data*, hereafter) in the SMR area. Such a data allocation can reduce the SMR update overhead, because we can directly update in-place in the CMR area, which does not incur SMR update overhead. However, it is still challenging to make allocation and relocation decisions due to the SMR-specific zone-cleaning cost. As described in Section 2.4, a simple NSW-Frequency-based swapping scheme still incurs severe performance degradation, especially in a higher space capacity usage.

Adaptive Format of Disk Components. We make the design choice to have a CMR-format Cache in H-SMR drive as it can redirect SMR updates and accumulate many subsequent updates before eviction. Such a cache can reduce SMR update overhead without swapping the whole zone, as does in the Dynamic Data Placement described above.

Leveraging H-SMR's unique format-conversion ability, Fluid-SMR creates the extra space capacity by changing a small portion of the CMR area into SMR format and allocating more data into the SMR region. Because SMR has a higher storage density, the same disk area will have a larger space capacity than needed by the user data, and therefore the surplus CMR space can be used as the on-disk Cache. Given specific workloads, it is challenging to determine the CMR, SMR, and Cache spaces' right sizes. When the workload dynamically changes, it is even more challenging to decide how to perform necessary data migration and format conversion to fit a new workload phase.

2.4 Evaluating NSW-Frequency-based Zone-Swapping

To inspire the FluidSMR design, we start from a simple version of the H-SMR management problem: zone-swap-only management for H-SMR drives. In other words, given an H-Partition with CMR and SMR zones both filled with user data, the problem is to find an efficient zone swapping schemes for better performance.

NSW-Frequency-based zone swapping. One simple approach is to leverage popular schemes used in hybrid storage systems or cache policies, such as recency-based or frequency-based schemes. These policies are well studied and can be easily adopted. However, such generic replacement policies do not consider the unique SMR write properties. First, SMR zones have the read/write asymmetry, i.e., only write will potentially cause the performance overhead because of the update constraint, while there is no constraint on reading existing data on the SMR zones. Second, there are two types of write operations, sequential write and non-sequential write, where only NSW incurs performance overhead. Therefore, in this scheme, we use an *NSW-Frequency-based* zone swapping, where only the number of NSWs is used to calculate the frequency, which in turn is used to aid the zone swapping decisions.

Preliminary Test Result. We carried out a preliminary test to compare the performance of the recency and frequency baseline with our the NSW-Frequency-based swapping scheme. In the recency baseline, when an SMR zone is accessed, it will be swapped into the CMR area with least recently accessed CMR zone. Here "access" includes both read and write. In the frequency baseline, we use a counter for each zone (including CMR and SMR) to record the number of accesses, including reads and writes. Then, during each time window (24 hours), we rank the zones based on the counter (i.e., access frequency) and put low-frequency zones in the SMR area and high-frequency zones in the CMR area by swapping zones between CMR and SMR areas. The counter is reset at the beginning of each 24-hour time window. Our NSW-Frequency-based scheme keeps a record on the number of NSWs occurred in each zone, and those zones in SMR space with a large number of NSWs are considered as hot zones and will be swapped with the zones in the CMR space with fewer NSWs during a 24-hour observation period. Other details about the test setup can be found in the evaluation section (Section 7). The results show NSW-Frequency-based swapping scheme reduces the average latency of the recency and frequency baselines by 22.4% and 29.4%, respectively (workload `ts_0`, usage = 0.9), because it considers the SMR-specific SMR update overheads.

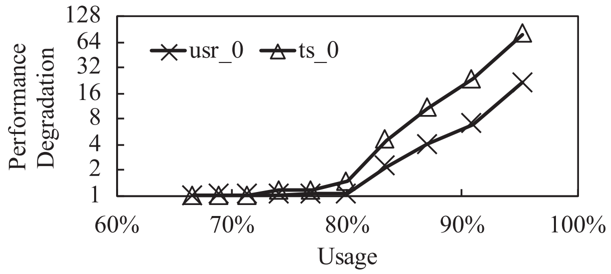

Fig. 3. Performance degradation using NSW-Frequency-based zone-swapping scheme. Here the performance degradation is defined as the average latency normalized to the pure-CMR average latency. For both traces, the average latency increases sharply as usage increases. (Note the logarithm scale in the y-axis).

However, even using this NSW-Frequency-based zone swapping, the H-SMR performance still degrades seriously in higher space usages where most of the disk space is in the SMR format. As shown in Figure 3, when the usage is beyond 80%, the average latency rockets up exponentially.

Observations. We have three observations based on this zone-swapping scheme. First, usage-aware formatting is the H-SMR's unique advantage over the traditional fix-formatted SMR drives. By only formatting the necessary amount of disk area into SMR, we can effectively reduce the SMR update overhead. When the usage is low (below 70%, Figure 3), the zone-swapping scheme achieves no performance degradation from the ideal pure-CMR case, which is not possible in the traditional SMR models where all the disk area is permanently formatted into SMR format. Second, NSW plays an important role in the H-SMR scenario in addition to commonly used factors such as recency or frequency. The preliminary result analysis shows that an SMR-specific zone swapping that considers the NSW-specific characteristics has more advantages than those generic cache eviction algorithms that ignore the NSW factor. This motivates the NSW-based conversion in Section 5.2. Third, only using a zone-swapping scheme is not sufficient. Figure 3 shows a serious performance degradation in the high storage usage situations for those swap-only schemes. The main problem of a zone-swapping-only scheme is that in some cases if only a small portion of an SMR zone gets NSWs, then the other blocks of the SMR zone that do not experience NSWs are still swapped to CMR. A whole-zone swap is inefficient as it introduces a large swapping overhead but only benefits a small number of blocks (ones that have NSWs). Such inefficiency of whole-zone swapping motivates us to create a more effective on-disk CMR-format cache that only redirects the NSWs into the CMR (the cache) instead of swapping the whole zone to the CMR (the CMR zones).

3 ASSUMPTIONS AND FLUIDSMR OVERVIEW

3.1 Assumptions

We assume a physical space partitioner will partition the whole physical area of the disk into multiple *physical partitions* (namely *H-Partition* with *H* standing for H-SMR). The partitioner specifies the size of the H-Partition. How to implement this physical space partitioner is out of the scope of this article. We focus on space management within one H-Partition.

The physical area of one H-Partition is further divided into *CMR area* and *SMR area* (Figure 4). When the size of CMR areas is zero, the H-Partition is in a *pure-SMR* state that presents the maximum space capacity the H-Partition can support. However, when the SMR areas' size is zero, the H-Partition is in a *pure-CMR* state that exhibits the least space capacity with no SMR update overhead. Generally, the higher the space usage, the higher the SMR area ratio is needed to create

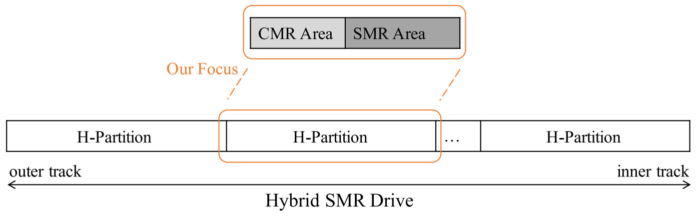

Fig. 4. Disk Layout and H-Partition.

enough capacity for the user data. With a given space usage, FluidSMR may choose to convert some extra CMR area to SMR than required by the space usage to create an on-disk cache (Section 5) in the CMR area. In other words, moving more data into the SMR area and some extra free space in CMR can be used for caching. When one H-Partition is newly created, the space usage is zero, and the H-Partition is in the pure-CMR state [16].

3.2 Design Overview

FluidSMR has three managing components: *usage growth management*, *adaptive format configuration management*, and *caching and swapping management*.

FluidSMR's usage growth management (Section 4) handles the application's space allocation requests to fulfill a growing usage of the application. The format configuration management (Section 5) adjusts the disk's CMR/SMR format by adaptive conversion according to the workload. The caching and swapping management (Section 6) makes decisions on how the CMR cache handles the NSWs and how to swap the zones between the CMR and SMR areas.

4 USAGE GROWTH MANAGEMENT

4.1 Growing Usage in H-Partition

The maximum capacity of an H-Partition (i.e., its pure-SMR capacity) is fixed once created. We assume the application has an increasing demand for space and will request new storage space from FluidSMR by issuing *space allocation requests* at a granularity of *chunks*. Here we set the chunk size as 256 MB to match the underlying zone size. So one chunk is also called one *logical zone*, or *zone* for short in this article. Each time the application will request one or multiple zones of space in the space allocation request. FluidSMR fulfills such space allocation request until the usage reaches 100% of the H-Partition's maximum capacity. We define the *capacity usage* (or *usage* for short) of the H-Partition as the ratio between the total accumulated size used by the user and the maximum capacity of the H-Partition. The usage of the H-Partition increases as the application extends its logical space for more capacity.

As a referencing example, one H-Partition with a maximum capacity of 150 GB (i.e., 150 GB in pure-SMR, or 100 GB in pure-CMR, or any space capacity between 100 GB and 150 GB with a combination of CMR and SMR assuming an SMR density gain of 1.5×) is serving a file system with an initial size of 20 GB. Initially, FluidSMR will map 80 physical zones of the H-Partition to 80 logical zones, presenting a continuous 20 GB LBA space to the file system. Later, when the file system's usage space expands to 50 GB, FluidSMR will make necessary operations (more details in the following sections) within the H-Partition and append another new 30 GB space to the original 20-GB LBA space. The file system can then be enlarged to utilize the new 30 GB space using commands like `resize2fs`. FluidSMR can continuously append new space to the LBA space to be consumed by the file system until the H-Partition's maximum capacity is used up (150 GB). Besides file systems, FluidSMR can expose the LBA space directly to applications too, as long as

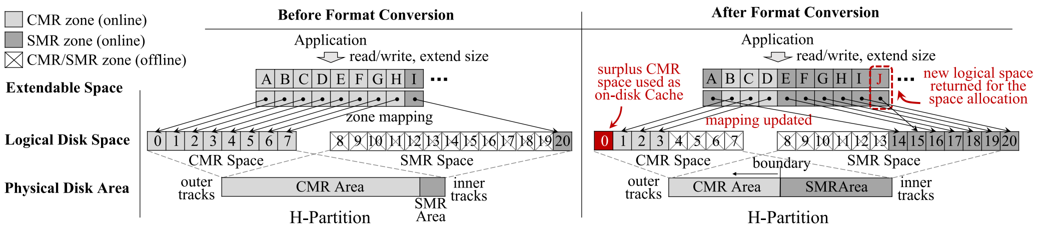

Fig. 5. Zone mapping and format conversion.

the applications can request to enlarge the LBA space when needed. When making LBA extension, the file system or application can optionally specify an “SMR-hint,” which indicates the newly allocated space can be in SMR format (e.g., intended for read-only data) for internal optimization in FluidSMR (see Section 4.3).

4.2 Zone Mapping

As illustrated in Figure 5, FluidSMR has a two-level mapping. The address space exposed to the application is called *extendable space*. The extendable space is mapped to a *logical disk space*, which is a logical view of the address space of the H-SMR drive. The logical disk space is then mapped to a *physical disk area*. The extendable space is divided into *user zones*, and each zone matches the H-SMR zone size (256 MB). Each user zone is mapped to either a CMR logical zone or an SMR logical zone, and eventually to a CMR physical zone or an SMR physical zone. By a CMR/SMR zone, we refer to both an online CMR/SMR zone in the logical disk space and its counterpart in the physical disk area.

The logical disk space in an H-Partition consists of three logical components: the CMR space, the SMR space, and the Cache space (or *Cache* in short). Please note the *Cache* is located in the CMR area and controlled by users/applications, not in an SMR area, as in other types of SMR drives. Underneath, the physical disk area is divided into two types of areas, the SMR-format area and the CMR-format area (outer track portion). Please note that modern disk drives may have the CMR portion in the middle of the disk due to head geometry. In this design, we simplify the H-SMR model and follow the practices in Reference [20] and Reference [16] to lay the CMR area in the outer track portion for high throughput, but the design can be easily extended to more complex cases (for example, CMR is in the middle track portion, or spread in several locations).

When an application uses more storage space, empty (i.e., unused) CMR or SMR zones from the H-Partition will be considered new logical zones, and these logical zones can be used as available user zones. The zone mapping is updated accordingly to ensure a linear logical data space with no LBA gaps. The zone-mapping entry for each zone is also stored in this zone’s first sector for recovery purposes.

In the simplified example illustrated in Figure 5, the left figure shows the current format and zone-mapping status before the format conversion. When the application request to allocate one more zone of space, a format conversion happens on the H-Partition, converting four CMR zones (4–7) into six SMR zones (14–19). In this process, the user zones E–H are migrated to the newly formatted SMR zones 16–19 from the original CMR zones. The SMR zone 15 is mapped to the new user zone J and appended to the current extendable space to fulfill the application’s allocation request. In this example, FluidSMR migrates user zone A from CMR zone 0 to the newly formatted SMR zone 14, resulting in one CMR zone 0 as the on-disk cache. Finally, the zone mappings for user zones A, E–H, and J are updated and persisted accordingly.

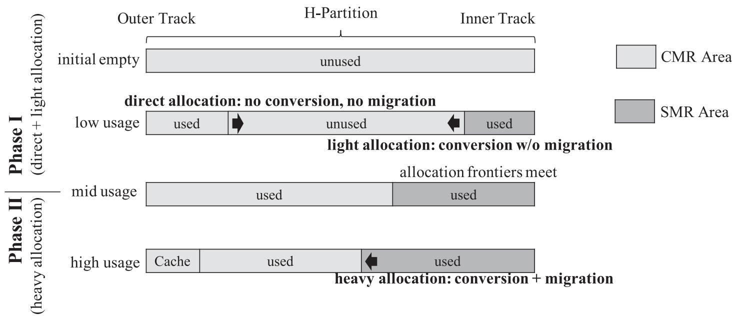
Outer Track
H-Partition
Inner Track
initial empty
unused
CMR Area
SMR Area
direct allocation: no conversion, no migration
low usage
used
unused
used
light allocation: conversion w/o migration
allocation frontiers meet
mid usage
used
used
high usage
Cache
used
used
heavy allocation: conversion + migration
Phase I (direct + light allocation)
Phase II (heavy allocation)

Fig. 6. Two-phase allocation.

FluidSMR persists the zone-mapping table to the disk after each zone allocation (also after each zone swapping and conversion operation, which will be introduced later in Section 5 and Section 6). A copy of the zone-mapping table is kept in memory. With each zone number taking 8 Bytes, the space overhead of a 20 TB H-SMR drive is 320 KB. Or equivalently, this zone-mapping table's memory overhead is 16 KB per TB of disk capacity.

4.3 Phase I: Direct Allocation and Light Allocation

As the usage grows from low to high, FluildSMR has two allocation phases (Figure 6). In the first allocation phase, the H-SMR partition starts as all CMR format [16]. As the second figure of Figure 6 shows, two types of allocation can happen, from the opposite side of the H-Partition. Allocation from the out track is called *direct allocation*, while the allocation from the inner track is called *light allocation*.

Direct allocation allocates CMR zones from the outer track. It is the default type of allocation when a new allocation request comes. FluidSMR maps new user zones to CMR zones from outer to inner tracks for higher performance. Direct allocation does not involve any format conversion.

Light allocation allocates SMR zones from the inner track. It is performed when the application specifies the SMR-hint (Section 4.1) to indicate the incoming data will not be updated at least in the near future. FluidSMR will format a portion of CMR from inner track portion into SMR to fulfill the allocation request. This will shift the CMR/SMR area boundary toward the outer track portion and increase the SMR area portion. Light allocation involves CMR to SMR conversion but does not introduce any data migration.

In light allocation, the overhead depends on the number of times that format conversions are carried out (about 50 ms per conversion command execution). Therefore, such conversion overhead can be further reduced if the user allocates a bigger SMR space at one time or specify an initial SMR space at the inner track portion during the H-Partition initialization. Such SMR-hint is suitable for archival files, backup files, or other data that are not expected to be updated soon. This user-provided SMR-hint will help FluidSMR make a better initial data placement decision and reduce the zone-swapping cost (zone-swapping will be introduced in Section 6).

4.4 Phase II: Heavy Allocation

In the second phase when usage is high (bottom figure in Figure 6), *heavy allocation* is used to create new capacity. In such high usage, both the CMR and SMR spaces are almost used up with user data, and conversion from CMR to SMR is needed to create more space capacity to fulfill subsequent space allocation requests. FluidSMR performs a CMR-to-SMR conversion at the CMR/SMR space boundary, moving the boundary toward the outer track portion. Therefore, heavy allocation involves both format conversion and data migration.

FluidSMR takes a proactive approach to make extra space for this space allocation request by making a guard range in preparation for future requests. In this case, the application's future allocation requests can be fulfilled right away if there are enough empty zones from the guard range. Such a guard range avoids a future allocation request wait for data migration to complete. The user can configure the size of this guard range. It depends on how many future allocations may occur in a short duration. If the user/application does not specify enough guard range for one allocation, then some future allocations may have to wait for the conversion to finish and introduce a long latency. When the unused area's size falls below a user-defined guard range size, FluidSMR will grow the space of the unused area by converting more CMR space to SMR until the unused size is above a pre-defined guard range again. The conversion and the subsequent valid data migration are performed at a pace matching the allocation intensity. FluidSMR uses a quantized migration (Section 4.5) as a knob to balance the migration I/Os with other application I/Os.

FluidSMR will choose "cold" CMR zones to convert to SMR zones. Such "cold" CMR zones are defined as the CMR zones with small ranges of LBAs that are updated since the last allocation. We have more discussion in Section 5.4.

4.5 Quantized Migration

There is an intrinsic tradeoff between the performance of the application I/Os and the migration efficiency. Theoretically, the task of valid data migration can be broken down into small piecemeal migrations and opportunistically inserted into the time intervals between consecutive application I/Os (i.e., idle intervals). In this case, the application I/Os will experience less performance degradation. However, it will take a longer time for the conversion to finish, especially when the workload is intensive and does not have large time intervals between consecutive IOs. Such piecemeal and fractional migration is inefficient, because more time is spent on the seek time and rotational delay than migrating data. Worse still, a prolonged migration task is forced to share time with more application IOs, which further increases the end to end migration time.

Based on this observation, we propose a *quantized migration* mechanism in FluidSMR that helps the application find a favorable spot in the tradeoff space. We define a *migration quantum* as the smallest-size unit of data to migrate that will carry to completion even other application I/Os arrive (Figure 7). A small migration quantum will have less impact on the application, since the migration is done at a finer granularity, therefore, causing less increase of latency to other application I/Os. In contrast, a large migration quantum gives priority to the migration I/O so that the conversion will finish sooner, but other application I/Os will have an increased latency.

The key insight here is that the maximum latency increase is proportional to the migration quantum's size. Therefore, adjusting the migration quantum's size provides a mechanism that helps the application ensure a controllable performance degradation while conducting a conversion and valid data migration.

An application can specify how much the increase of latency is tolerable for the application IOs (*acceptable increase of latency*) based on its requirement of quality of services. How does the application translate the quality of service into the acceptable increase of latency is out of the scope of this article. FluidSMR only deals with how to calculate the right size of migration quantum to ensure that the acceptable increase of latency is not violated.

Note that the quantized migration is not only useful in the conversion during usage growth. It is also helpful in the valid data migration during format configuration adaption in the next section.

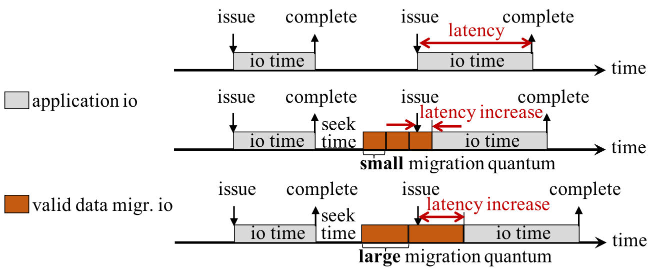

Fig. 7. Impact of the size of migration quantum on the application performance. The larger the migration quantum, the greater the increase of latency the application experiences, but the sooner the valid data migration completes.

5 ADAPTIVE FORMAT CONFIGURATION DESIGN IN H-SMR

In this section, we want to answer the key question:

Given certain workload, how to adapt the zone format of the H-Partition to improve the performance?

5.1 Problem Definition

In the heavy allocation phase, the H-Partition consists of three components: CMR space, SMR space, and Cache space (in CMR format). The space usage is beyond the pure-CMR capacity yet below the pure-SMR capacity of the H-Partition. There is no "unused" space for a direct or light allocation. The sizes of the CMR space, SMR space, and the CMR cache space need to satisfy the following two constraints:

- physical size constraint: The total sum of the physical H-Partition areas of the CMR space, SMR space, and the CMR Cache should be equal to the H-Partition's physical space capacity.
- user data size constraint: The total logical space capacity of the CMR space and the SMR space together should be equal to the total size of the user data.

We formally illustrate such intrinsic inter-dependencies among the size of Cache (S_{cache}), CMR space (S_{cmr}) and SMR space (S_{smr}) as follows.

$$(S_{\text{cache}} + S_{\text{cmr}}) * \alpha + S_{\text{smr}} = C, \qquad (1)$$

$$S_{\text{cmr}} + S_{\text{smr}} = U, \qquad (2)$$

where: α = SMR density gain,
$S_{[\cdot]}$ = size of $[\cdot]$ space,
C = full SMR capacity of H-Partition,
U = user data size.

We denote a *format configuration* of an H-Partition as one feasible combination of the CMR Cache, CMR data space, and SMR data space that satisfies the equations above. There are many challenges in determining how to adapt to an "ideal" format configuration best suited for the current workload. For example, how many and which CMR zones are going to be converted to SMR? However, how many and which SMR zones need to be converted back to CMR? How can we tell if the performance is improved after the conversion?

In the remainder of this section, we first analyze the impact of two-directional conversions and discuss the criteria to determine the conversion direction (Section 5.2), and then introduce our adaptive conversion algorithm (Section 5.3).

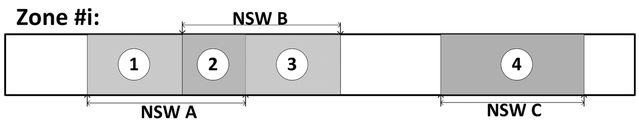

Fig. 8. An example of the range of non-sequential writes (NSW-Range). The NSW-Range is the total amount of gray areas.

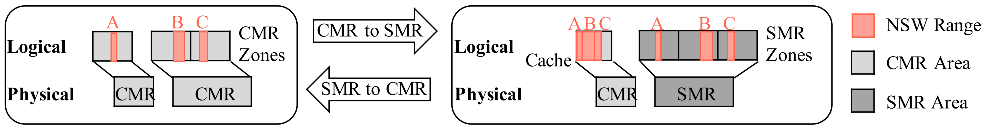

Fig. 9. Two-directional Conversion. FluidSMR has the ability to perform conversion in two directions to adjust the size of the Cache.

5.2 NSW-Range and Two-directional Conversion

NSW-Range. Motivated by the observation from Section 2.4, we define *ranges of non-sequential writes* or *NSW-Ranges* as the total covered ranges of NSWs in one zone during a time window (e.g., 24 hours, and FluidSMR has an adaptive time window design in Section 5.3) as shown in Figure 8. In Figure 8, the ranges of NSWs A, B, and C are the gray areas of ①, ②, ③, and ④. The gray area ② is the overlapping portion of NSWs A and B, and thus it is only counted once in the NSW-Ranges. The metric indicates how much cache size will be occupied by the NSWs happening to an SMR zone.

Two-directional Conversion. In one H-Partition, we can create a CMR-format Cache by converting more CMR space to SMR space. It may sound counter-intuitive to create a CMR cache by converting away from CMR format. However, when converting some CMR space to SMR format, the logical space capacity of one H-Partition increases, and thus the "surplus" space can be used as a Cache to hold NSWs to the SMR zones. Such process of CMR-to-SMR conversion is illustrated in Figure 9 (right arrow). Supposing that an SMR density gain is 1.5×, then two CMR zones can be converted into three SMR zones. So the user data from the three CMR zones in the left figure of Figure 9 can be migrated to the newly formatted SMR zones as shown in the right figures of Figure 9, resulting in one extra CMR-format Cache zone.

Such extra CMR cache can hold in-coming NSWs to the SMR zones, then subsequent updates to whose data can happen in-place without incurring any SMR update overhead. In an ideal case, if the total size of the NSW-Ranges of the three SMR zones is smaller and can fit into the newly created extra Cache zone, then such a CMR-to-SMR conversion is beneficial as it does not introduce new SMR update overhead but creates more available Cache space that can help to hold the NSWs of other SMR zones. In the example of the right figure of Figure 9, the newly available Cache space is the grey area of the Cache.

However, such a conversion can happen in the reverse direction by converting the three SMR zones back to two CMR zones (see the right to left conversion in Figure 9). In this case, we lose one zone of the Cache space, and the data in two SMR zones can be migrated to these two newly created CMR zones such that the NSWs to the two migrated zones will not incur SMR update overhead anymore, because they are now stored in CMR zones. Such an SMR-to-CMR conversion is beneficial when the total NSW-Ranges for these two SMR zones cannot fit in one Cache zone. By converting the SMR zones into CMR zones, the overall Cache space contention is alleviated.

To avoid fractionalizing the physical space, a minimum number of *three* SMR zones needs to be converted together to get *three* CMR zones (Figure 9). In general, we define the basic conversion batch size S_{conv} as the minimum number of zones needed to be migrated (e.g., three) without creating fractions of physical zones. Given the SMR density gain is α, $S_{\text{conv}} = \frac{\alpha}{\alpha-1}$.

Criteria of the Conversion Direction. The criteria to determine if a zone should be in either SMR or CMR depends on the size of NSW-Ranges and the SMR density gain. In the example in Figure 9, converting three CMR zones creates one extra CMR cache space. If the total size of NSW-Ranges is smaller than the size of one zone, then converting CMR to SMR is beneficial and vice versa. In general, supposing the SMR density gain is α, if one chunk of data takes N CMR zones, then the same physical space could hold αN SMR zones. Denoting r as the NSW-Range ratio among the data, then fraction r of the data (i.e., the data the experiences NSW) will be redirected to the CMR cache. To hold all the redirected NSWs, the cache will take up rN CMR zones, i.e., a physical space equivalent of $r\alpha N$ SMR zones. At the break-even point, one physical space can hold N CMR zones, and it can also hold N SMR zones plus a CMR cache that can keep all NSW-Range of data. Let the r^* be the NSW-Range ratio at the break-even point, then $r^*N = N + r^*\alpha N$, i.e., $r^* = \frac{\alpha-1}{\alpha}$.

An NSW-Range ratio $r > r^*$ indicates that in the SMR + CMR cache configuration, the CMR cache cannot hold all the NSWs, and thus the extra NSW will incur cache eviction and SMR update overhead. Another way to interpret is that to hold the NSWs in Cache, the SMR space and CMR cache will take up more physical space than using pure CMR format to hold the same data. Therefore, the data should be stored in CMR format (converting following left arrow in Figure 9). However, if $r < r^*$, then storing the data in SMR plus CMR cache configuration takes less physical space than storing the same data in CMR format, and therefore the conversion should follow the right arrow, converting from pure CMR to the SMR + CMR cache setting. This analysis leads to the adaptive conversion algorithm in the next section.

5.3 H-SMR Adaptive Conversion Algorithm

Based on these observations, we introduce a format configuration algorithm (Algorithm 1). We introduce several principles to guide the format conversion while cross-referencing the corresponding lines in the pseudo-code in Algorithm 1.

Two-directional Conversion. Format conversions in both directions can adjust the current format configuration of the H-Partition toward a better configuration. According to the discussion in Section 5.2, converting CMR zones with NSW-Range ratio below r^* into SMR increases the relative Cache size (lines 17–23, Algorithm 1). Similarly, converting SMR zones with NSW-Range ratio above r^* also decreases the relative Cache size (lines 24–30, Algorithm 1).

Early Termination. In some cases, we do not need to convert all the CMR zones with $r < r^*$ to SMR format, neither do we need to convert every SMR zone with $r > r^*$ into CMR. As long as the Cache size is greater than the total size of NSW-Ranges (i.e., the relative Cache size is positive), the format configuration reaches the ideal state, and the adjustment can be early terminated (line 23 and 30 in Algorithm 1). Otherwise, the algorithm should keep adjusting until all the CMR zones have $r > r^*$ and all the SMR zones with $r < r^*$ (however, as the number of zones to be converted should be multiple of C_{conv}, there can be some zones left unconverted when the remaining number is less than C_{conv}). In such cases, any additional conversion will not increase the relative CMR size, and the gap between the Cache size and total size of NSW-Ranges is minimized.

Swapping before Conversion. When there exist both candidates SMR zones and CMR zones identified to be converted into the opposite format, we can directly swap the user data between a CMR-SMR zone pair. Using swapping instead of conversion saves data migration for the format adjustment procedure. We keep on swapping until one type of conversion candidate zones (CMR or SMR) is exhausted (lines 7–16, Algorithm 1). Then we perform the format conversion for the

ALGORITHM 1: H-SMR Adaptive Conversion Algorithm

```
 1: procedure Conversion procedure
 2:     $R_{\text{total}} \leftarrow$ total NSW-Range
 3:     $C_{\text{total}} \leftarrow$ Cache size
 4:     exit procedure if $C_{\text{total}} \geq R_{\text{total}}$                ▷ early termination
 5:     insert all CMR zones into queue $q_{\text{cmr}}$
 6:     insert all SMR zones into queue $q_{\text{smr}}$
 7:     while $q_{\text{cmr}}$ is not empty and $q_{\text{smr}}$ is not empty do
 8:         $z_{cmr} \leftarrow$ remove one CMR zone from $q_{\text{cmr}}$ with min NSW-Range
 9:         if $r > r^*$ for $z_{cmr}$ then
10:             break from the while-loop
11:         $z_{smr} \leftarrow$ remove one SMR zone from $q_{\text{smr}}$ with max NSW-Range
12:         if if $r < r^*$ for $z_{smr}$ then
13:             break from the while-loop
14:         swap $z_{cmr}$ and $z_{smr}$
15:         update $C_{\text{total}}$ and $R_{\text{total}}$
16:         exit procedure if $C_{\text{total}} \geq R_{\text{total}}$            ▷ early termination
17:     while $q_{\text{cmr}}$ has more than $C_{\text{conv}}$ CMR zones do
18:         $l_{\text{cmr}} \leftarrow$ remove $C_{\text{conv}}$ CMR zones from $q_{\text{cmr}}$ with smallest NSW-Ranges    ▷ $l_{\text{cmr}}$: candidate list
19:         if avg $r > r^*$ for $l_{\text{cmr}}$ then
20:             exit procedure
21:         convert CMR zones in $l_{\text{cmr}}$ to SMR
22:         update $C_{\text{total}}$ and $R_{\text{total}}$
23:         exit procedure if $C_{\text{total}} \geq R_{\text{total}}$            ▷ early termination
24:     while $q_{\text{smr}}$ has more than $C_{\text{conv}}$ SMR zones do
25:         $l_{\text{smr}} \leftarrow$ remove $C_{\text{conv}}$ SMR zones from $q_{\text{smr}}$ with largest NSW-Ranges    ▷ $l_{\text{smr}}$: candidate list
26:         if avg $r < r^*$ for $l_{\text{smr}}$ then
27:             exit procedure
28:         convert SMR zones in $l_{\text{smr}}$ to CMR
29:         update $C_{\text{total}}$ and $R_{\text{total}}$
30:         exit procedure if $C_{\text{total}} \geq R_{\text{total}}$            ▷ early termination
```

remainders zones (either line 17-23 or lines 24–30 will happen in Algorithm 1). Swapping can also be early terminated if the Cache space can grow above the total size of NSW-Ranges (lines 4 and 16, Algorithm 1).

Please note that it is not safe to swap two zones by just reading their data into memory and write back to the other zone's location, because such data will be lost if a crash happens before it is written back. In current design and evaluation, we assume the future H-SMR product will be equipped with a small amount of **nonvolatile memory (NVM)** to hold the in-flight zone data temporarily. However, if future H-SMR hardware does not have extra NVM for this purpose, then we can always reserve at least one free zone on the disk to hold one copy of the data temporarily before it is overwritten.

Prioritizing for a Larger Gain of Relative Cache Size. Not every zone is created equal. The smaller NSW-Range ratio for a CMR zone, the more gain we will get if we convert this CMR zone into SMR format. Similarly, the larger NSW-Range ratio for an SMR zone, the more gain will be there if we convert it to CMR format. Therefore, CMR zones with the largest NSW-Range ratios and SMR zones with the smallest NSW-Range ratios are prioritized to be swapped or converted (lines 8, 11, 18, and 25 in Algorithm 1). A better configuration can be achieved in these cases using less swapping and conversion, incurring less data migration.

More Details about Conversions. According to Reference [15], the host can convert any extent of physically consecutive zones from one format to the other using H-SMR APIs. However, the number of zones to be converted should not be too small, because the new format's physical zone boundary may not necessarily align with the other format's boundary and will, therefore, potentially waste some physical space at the "seam" between the two formats. In the example shown in Figure 9 where the SMR density gain is 1.5×, the number of consecutive CMR zones to be converted should be multiple of two, and similarly, the number of consecutive SMR zones to be converted should be multiple of three. Therefore, when converting SMR candidate zones into CMR format, a triplet of SMR zones with the largest total size of NSW-Ranges will be converted. The conversion will stop when the relative Cache size does not increase if the conversion were performed. If the three SMR zones are scattered in the H-Partition and are not consecutive, then swapping is needed during the conversion to create the candidate zones to be consecutive, and then conversion is performed. Please note that one zone of the Cache space will be reclaimed by migrating the cached data back to the SMR zones. However, the conversion from CMR to SMR follows a similar procedure. Two consecutive CMR zones are needed to re-format into three SMR zones. The third CMR zone, which does not need to be physically adjacent to the CMR zone being converted, will migrate is data to one of the newly formatted SMR zone, and this CMR zone becomes a new zone of the Cache.

Periodical Format Adjustment. Zone conversions will be performed periodically after each time window. Such zone format adjustment does not happen frequently. That is, the time window will not be too small for two reasons. (1) The statistics accumulated should be sufficient and accurate to make the right decisions. (2) A Conversion is an expensive operation, as it includes valid data migration and necessary Cache eviction if the size of Cache needs to shrink. FluidSMR sets the length of the time window as one hour by default. If the number of zones of the CMR/SMR/Cache space remains the same after one time window, then FluidSMR considers that the format configuration is ideal and the workload is stable, and it will double the duration of the next time window. The time window is at a maximum of 24 hours to match the diurnal pattern commonly observed in workloads [4, 8]. However, other schemes can also be explored, such as using real-time CPU cost or IO intensity to determine a reasonable timing to perform the conversion. Once the number of zones changes across two time windows, FluidSMR considers that there is a phase change of the workload, the time window is reset to the default value of one hour to speed up the adaption to the new phase of workload.

5.4 Remarks on Supporting Heavy Allocation

FluidSMR handles heavy allocation (Section 4.4) in a similar logic as the aforementioned CMR-to-SMR conversion. When FluidSMR needs to create more capacity by CMR-to-SMR conversion, CMR zones with $r < r^*$ are prioritized to be converted as they reduce the relative Cache size by the least amount. Similarly to the two-directional conversion described above, necessary zone swapping is needed to make the CMR candidate zones physically consecutive before converting to SMR as a group. Unlike the CMR-to-SMR conversion that creates a new Cache zone, here, such a conversion increases the logical capacity and will be used to store more user data.

5.5 Memory Requirement of NSW-Range Tracking

The adaptive conversion algorithm needs to track the NSW-Range of each zone. For every zone, FluidSMR maintains an NSW-Range counter (4 Byte/zone, counting the number of NSW sectors) and a sector bitmap (8 KB/zone, tracking if a sector has experienced NSW before). For a 20-TB

H-SMR drive, the counters' total size is 320 KB, and the bit map will take roughly 650 MB. In other words, the memory overhead for NSW-Range tracking is 32.5 MB per TB of disk capacity. Please note that the NSW-Range counters and bitmaps do not need to be persisted and will be re-accumulated if any crash or power loss happens.

There can be various approaches to reduce this memory overhead. For example, FluidSMR can track the NSW-Range with a coarse granularity of 32 KB chunks instead of 4 KB sectors. Bloom filters [5] can be used instead of a sector bitmap to determine if a sector has been written before. Interval trees [12] is also a good candidate to represent NSW-Ranges within each zone. We leave the investigation into the tradeoffs among memory/computation/accuracy of these optimizations as future work.

6 CACHING AND SWAPPING MANAGEMENT

6.1 Cache Replacement Algorithm

We have tried out the following cache replacement schemes to manage the redirected NSW data in the CMR on-disk cache. If an SMR write targets the write pointer, then it will be directly issued to the SMR zone without entering the Cache.

Improved Block-based LRU. A first intuition is to leverage existing caching policies designed for the main memory (e.g., LRU) to "cache" frequently updated blocks in the Cache. Consequently, subsequent updates to those data will happen in the CMR zones without introducing additional SMR update overhead. However, directly adopting those caching policies results in poor performance, because cache eviction frequently happens at the block granularity, and each eviction reclaims one block of free space but triggers one expensive zone read-modify-write. Therefore, we designed an *improved block-based LRU* policy that evicts the whole zone containing the victim block out of the Cache instead of only evicting the victim block. Here *evicting a zone* means evicting all blocks from the Cache that belongs to that zone. In this case, we pay the same price to read-modify-write one zone but reclaim more space, and thus the eviction happens less frequently. The idea of evicting a whole zone instead of individual blocks is seen in the Seagate translation layer described in Skylight. However, this Improved Block-based LRU follows the recency to determine which zone to evict instead of the logging sequence in the persistent Cache.

In-place FIFO Log. The improved block LRU policy will fragment the Cache's free space, causing random IOs that are not friendly to disk drive accesses. To avoid fragmenting the cache space, we design an *in-place FIFO log* policy (upper figure in Figure 10) that organizes the redirected data in a log structure. If already buffered in the log, then redirected data will be in-place updated, because the Cache is in CMR format. Otherwise, it will be allocated to the log head pointer. Free space is reclaimed from the log tail pointer by *log cleaning*, which evicts the zone containing the tail block (the block at the tail pointer) and advances the tail pointer. Note that if the tail block has already been evicted, then the tail pointer will be advanced. However, the drawback of an in-place FIFO log is that it strictly follows the FIFO ordering and does not distinguish different types of zones. We observe that frequently updated data blocks will come back to the Cache soon after they were evicted and take up the space that was just freed. In this case, the time consumed to read-modify-write the original SMR zone is wasted. Therefore, it is beneficial to distinguish different zones when making eviction decisions.

Loop-back Log. To improve the in-place FIFO policy, we propose a *loop-back log* policy for the Cache that identifies hot zones and "re-queues" the data of hot zones from the log tail to the log head without evicting them (called "loop-back," see the bottom figure in Figure 10). Similar techniques to retain hot data in the log is used in FLT design, too [31]. Here we first define an *epoch* as the time for the log head pointer to go through a full cycle and come back to the beginning of the

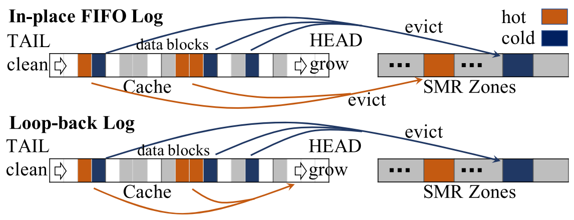

Fig. 10. Cache Policies: *In-place FIFO log* allocates redirected NSWs to the head and cleans from the tail by evicting every block belonging to the same zone as the tail block; *Loop-back log* distinguishes hot and cold SMR zones and only evicts data of cold zones out of Cache while keeping hot data in Cache by "re-queuing" it to the head of the log (loop-back).

Cache. Then if the cached NSW blocks of one SMR zone during the current epoch will be updated again in the next epoch, then we define this SMR zone as a "hot zone." In this case, evicting this hot zone (i.e., migrating the NSW blocks from CMR cache back to the SMR zone) pays the price of one read-modify-write but gains little space back, so it is better to keep it in the Cache instead of evicting it. Please note this hot-zone concept is in a posterior sense, because it is impractical to identify hot zones without knowing the future. However, we can design zone selection policies based on history to estimate which zones will be the hot zones. Although alternatives exist, we find that block LRU is simple yet effective in predicting hot zones. Specifically, the block numbers of all blocks in the Cache are organized into an LRU list. At the beginning of each epoch, zones that have the blocks only appearing in the MRU half of the list are considered hot.

While the LBA-PBA mapping is cached in the memory for fast lookup, new mapping entries are periodically persisted to the disk for reliability purposes. The persistent period is set to 30 seconds, consistent with the Linux `pdflush` period [40]. Each mapping entries is a triplet of (`lba`, `pba`, `length`). We limit the CMR mapping table size overhead (mapping table size/total disk capacity) to be 25 MB/1 TB. For example, a 20 TB H-SMR drive will at most use 500 MB of the memory space to keep the mapping entries. If the cache map is full, then CMR cache will trigger cleaning to make room for new mapping entries to be inserted to the in-memory mapping table. In other words, cache cleaning will be triggered when CMR Cache is full or the in-memory mapping table is full, whichever happens first. Such a cache cleaning triggering mechanism based on both the Cache's capacity and the Cache's mapping table capacity is also seen in Seagate SMR drives. One possible optimization to reduce the memory footprint of the CMR cache mapping table it to only keep hot caching table entries in the memory, while having an index to find cold mapping entries that is on disk. The investigation about the performance tradeoff is left as future work.

6.2 Cache Occupancy-based Swapping

As an optimization, in complement to the zone swapping described in Section 5, FluidSMR also enables a more frequent but light-weight zone swapping within each format adjustment period. Such swapping introduces less migration overhead than the whole format adjustment procedure introduced in Section 5 and can fine-tune the performance by swapping the user data between "hot" SMR zones with "cold" CMR zones.

FluidSMR uses the *cache occupancy* to determine the swapping priority. Here the cache occupancy of one SMR zone is defined as the actual space taken by the NSWs from the SMR zone in the Cache. SMR zones taking more space in the Cache will be selected to swap. Different from the NSW-Range statistics used in Section 5, the space occupancy of a zone in the Cache captures timely information of the space taken by one zone, and such ephemeral statistics is more suitable for a

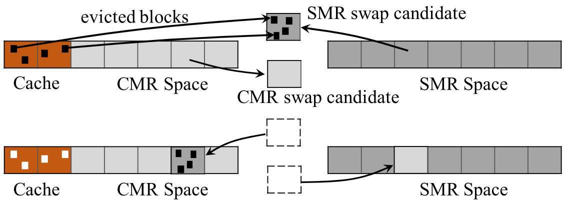

Fig. 11. Zone-Swap: CMR and SMR swap candidate zones are read out, combined with the updated blocks evicted from Cache, and written back to each other's location.

Table 1. MSR Trace Configurations

	Total Op. (Million)	Write Op. (Million)	Write Size (GB)	Write Ratio (%)	Max Offset (GB)
hm_0	4.0	2.6	20.5	64.50	15.0
prn_1	11.2	2.7	30.8	24.66	413.6
proj_0	4.2	3.7	144.2	87.52	17.4
proj_1	23.6	2.5	25.6	10.56	880.9
prxy_0	12.5	12.1	53.8	96.94	22.2
src1_1	45.7	2.2	30.3	4.74	293.6
src1_2	1.9	1.4	44.1	74.63	8.6
src2_2	1.1	0.8	39.2	69.67	182.1
stg_0	2.0	1.7	15.1	84.81	11.6
ts_0	1.8	1.5	11.3	82.42	23.6
usr_0	2.2	1.3	13.1	59.58	17.1
web_0	2.0	1.4	11.7	70.12	36.4

near-future fine-tuning. The occupancy-based swapping scheme evaluates all zones in the Cache at the beginning of each epoch (epoch as defined in the loop-back log) and labels the zones with occupancy above a threshold as "SMR swap candidates." Such SMR zones will be evicted when encountered by the log tail pointer in log cleaning. CMR zones that were not updated during the last epoch become "CMR swap candidates." When an SMR swap candidate zone is evicted, it will be paired up with a CMR swap candidate if available. Then the data in the CMR and SMR candidate zones will be read and written back to each other's location after combining the updated data evicted from the Cache (Figure 11). Here the zone swapping is integrated with the Cache eviction such that the write operations of the data evicted from the Cache are saved.

7 EVALUATION

7.1 Experiment Setup and Overall Performance Evaluation

As there are no H-SMR products available, we built an H-SMR simulator (3.0 K LOC C++ and 1.9 K LOC python) with disk models extracted from DiskSim [7]. We followed Ruemmler and Wilkes [41] to configure the seek time formula. The RPM for this simulated disk is set to be 10025, and the rotation delay is calculated based on this RPM. The average track size is 2 MB, and the maximum data transfer rate is set to 300 MB/s. The SMR density gain is 1.5:1, and the zone size is 256 MB. In all the tests, the write/read caches are disabled, and the queue size is set to one.

We use traces from the MSR Cambridge traces [37] that have a write footprint greater than 10 GB and large number of operations for evaluation (Table 1). This is because traces with a smaller write footprint and fewer operations do not fill up the cache and will not show a difference in performance. Traces are replayed as fast as the device can go without referring to the timestamps.

We test the performance of the workload on one H-Partition. To compensate the relative small LBA ranges of the MSR traces, the physical size of the hybrid partition and the usage is set differently according to each experiment's requirements. We use the max LBA as the basis and use the usage required by the experiment to calculate the H-Partition's size. For example, for trace `prn_1` with a max LBA of 413.6 GB, if we want to test the performance under a 90% usage, then the H-Partition size used in the evaluation would be 413.6 GB/0.9 = 460.0 GB, which is 1840 SMR zones. Given the mapping size overhead is set to 25 MB per TB of disk capacity (Section 6.1), the H-Partition's mapping table quota is $460.0\text{ GB} \times 25\text{ MB}/1\text{ TB} = 11.5\text{ MB}$. Adding the memory overhead of the zone-mapping table ($460.0\text{ GB} \times 16\text{ KB}/1\text{ TB} = 7\text{ KB}$, Section 4.2), and that of the NSW-Range tracking ($460.0\text{ GB} \times 32.5\text{ MB}/1\text{ TB} = 15.0\text{ MB}$, Section 5.5), the total memory requirement is 26.5 MB for this example. Similarly, in each test, all those parameters are set according to the corresponding max LBA and the disk usage. As a side note, in this `prn_1` example, the max LBA range is 413.6 GB, whereas the write size is about 10 times smaller (only 30.8 GB, see Table 1), imposing a relatively light pressure on the NSW handling.

The following schemes are evaluated as follows:

- `swap-only`: The scheme where only zone-swapping is used to re-locate user data as described in Section 2.4;
- `+cache-fix`: In addition to `swap`, convert 20% of the remaining CMR area into SMR to create surplus capacity for a CMR-format on-disk cache, and use block LRU to manage the cache space, with the format configuration fixed;
- `format-adapt`: On top of `+cache-fix`, add adjustment format conversion (Section 5);
- `fluid-smr`: The final FluidSMR scheme that improves from `format-adapt` by using the loop-back caching policy and the occupancy-based zone-swapping;
- `pure-cmr`: The workload is running against a pure CMR partition, which is free from the SMR update overhead. Note that this is an ideal case and is not possible in high storage usage.

7.1.1 Overall Performance under Different Workloads. We test the performance of the competing schemes under different traces, with the storage usage fixed at 90%. The result is shown in Figure 12. We can see although `swap-only` remedies the in-place SMR update overhead by migrating more frequently updated zones to CMR space, the average latencies are still higher than 100 ms. Such a high latency is due to the high overhead of the zone-swapping operation. Besides, the gain of swapping a zone is that further writes to this zone will not suffer from any SMR update overhead. However, in high storage usage, the benefit of swapping a zone may not be able to negate the swapping cost, making the performance even worse. In `+cache-fix`, by introducing CMR cache, more user data have to reside in SMR zones due to that some CMR area is converted to the SMR area to reserve extra CMR space for caching, but the performance is increased from `swap-only`. The performance improvement is because the cache space stores the hot data (frequently-updated) more efficiently, especially when hot data are distributed in a large number of zones where `swap-only` has no way to fit all the zones in the limited CMR space. `format-adapt` applies the adjustment format conversion scheme where the format of each zone is no longer fixed and will be dynamically converted according to the current situation of workload. We can see for most traces `format-adapt` outperforms `+cache-fix`, because the adaptive algorithm can find a better format configuration that fits the workload. We note that for some of the traces, e.g., `prn_1`, `format-adapt` has longer latency than `+cache-fix`. This is because the adaptive algorithm involves expensive format conversions and data migration, which may hurt the performance.

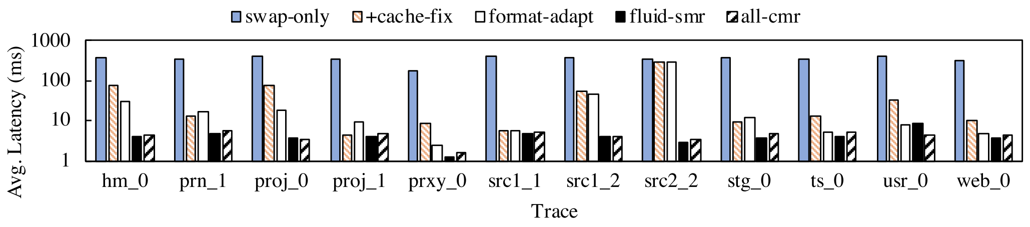

Fig. 12. Overall performance of FluidSMR compared with baselines. Note that the log scale in the y-axis.

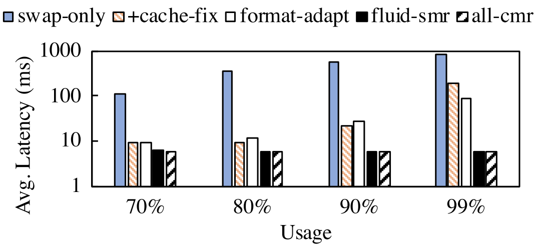

Fig. 13. Performance of FluidSMR compared with baselines under different utilization (using trace prn_1). Note that the log scale in the y-axis.

Besides, the adaptive algorithm takes time to converge to a better configuration, so during the convergence process, the disk configuration may be unfavorable to the performance. Finally, fluid-smr outperforms the format-adapt scheme because of the more efficient loop-back caching algorithm and the fine-tuning by the occupancy-based zone-swapping scheme in between the format conversion period. Sometimes fluid-smr can perform even better than the ideal pure-CMR case (e.g., in trace hm_0). This is because fluid-smr has the log-structured Cache that can convert random writes into sequential ones.

We compared the average latency as the top-line metrics and did not measure the indirect factors such as write amplification. Write amplification focuses on write performance but cannot capture the read performance. Besides, write amplification comes with different flavors (number of IOs versus data amount), emphasizing different factors but not a combined effect. Instead, we choose to use average latency to show the combined, end-to-end effects of all these factors. Still, the write amplification can be reflected by the results of write-dominate workloads. For example, in Figure 12, the workload prxy_0 has a write ratio of 99.94% and still outperforms other schemes. The average latency of fluid-smr is close to that of all-cmr, the write amplification free baseline, indicating a very small write amplification.

7.1.2 Overall Performance under Different Usages. We compare the performance of the schemes with different usages using the trace prn_1 (Figure 13). The performance of swap-only, +cache-fix, and format-adapt gets degraded as the usage increases. For swap-only, at higher storage usages, there is less CMR space for swap-only to swap. One zone may get only a few write hits before it is swapped out of CMR again. For +cache-fix and format-adapt, the performance decreases at higher usages, since more NSWs are happening but relatively less cache space to operate, so more Cache replacement occurs. By contrast, the average latency of fluid-smr stays low as the usage increases, because its loop-back cache replacement algorithm and the occupancy-based zone-swapping are more efficient. Therefore, FluidSMR can introduce a small overhead during eviction even when CMR cache space is relatively small in high storage usages.

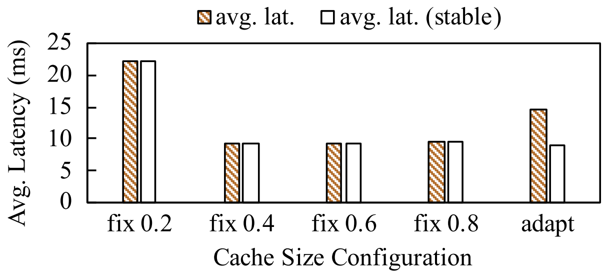

Fig. 14. Comparing fixed cache size vs adaptive cache size via conversion.

7.2 Evaluation of Adaptive Format Configuration

This subsection evaluates the adaptive format configuration's performance by comparing the adaptive format configuration with a series of fixed-format configurations. In the experiment setup, we first calculate the minimum portion of the SMR area to fulfill the usage. Then as baselines of fix size configuration, we convert 20%, 40%, 60%, and 80% of the remaining CMR area to make some extra space for Cache. We denote `fix 0.2` for the case where 20% of the remaining CMR area is converted to the SMR format. Please note that although these baselines' format configurations are fixed, the zone data can still be swapped. For a fair comparison, both the fix and adaptive size configuration schemes use block LRU as the cache replacement policy and adopt the NSW-Range-based zone-swapping scheme. The adaptive scheme (`adapt`) starts from an initial configuration with the minimum SMR area with no Cache space. We set the usage as 90%, and the trace used here is `prn_1`.

In Figure 14, we can see from the left bars that `fix 0.2` has inferior performance than the other fixed configurations (`fix 0.4`, `fix 0.6`, and `fix 0.8`), indicating `fix 0.2` is not a good configuration for this workload. Also we found that configurations ranging from `fix 0.4` to `fix 0.8` all exhibit similar good performance. This is because in some cases, a wide range of configurations can fit the NSW-Range entirely into the cache space. The `adapt` scheme is able to reduce the average latency of `fix 0.2` by 59%. However, it is still higher than those of the other fixed configurations (`fix 0.4`, `fix 0.6`, and `fix 0.8`). The reasons are as follows. First, the initial disk size configuration (i.e., minimum SMR area and no Cache) is not favorable where each NSW triggers one read-modify write of a whole zone, and thus creates a higher SMR update overhead. Second, the configuration adaption involves extra I/Os for format conversion and data swapping. However, when the disk configuration converges to a favorable state, and the adaption operation is less frequently triggered (Section 5.3), the stable performance (right bars in Figure 14 that measures the average latency since the fourth day) shows a low latency similar to that of `fix 0.4`, `0.6`, and `fix 0.8`, indicating the adaptive algorithm can adjust the format configuration of the H-Partition into a favorable state.

To show direct evidence of the size adaption, we monitor the sizes of SMR space, CMR space, and Cache space (in a unit of zones) and plot them in Figure 15. In the first 24 hours, there are more frequent adjustments where CMR space decreases. FluidSMR gradually converts CMR area to SMR area and creates more Cache space. After 36 hours, the adaption converges and the adjustment frequency is decreased to reduce the adjustment overhead. The adjustment in the beginning introduces more overhead, but this is necessary for the disk configuration to move away from the adverse configuration (i.e., no Cache) quickly. Still, there is more research issue on how to pick a good adjustment frequency that considers the fluctuation of the workload, and we leave the investigation to future work.

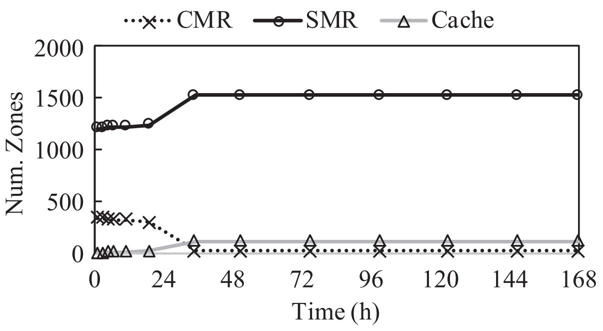

Fig. 15. Real-time size (in #zones) of CMR zones, SMR zones, and CMR Cache during the adaptive process for trace prn_1.

Another observation from this experiment is that, for tested traces, using a CMR cache with a fractional size of the H-Partition can produce a quite good performance, and our adaptive configuration does not show a great advantage over the fixed configurations. Figure 15 shows a wide range of optimal configurations (i.e., `fix 0.4`, `fix 0.6`, and `fix 0.8`). Such a wide range indicates the write traffic is not intensive so that the NSW-Range is relatively small to be totally fit in the CMR cache even with some room to wiggle. Given the trace has a small NSW-Range ratio, the adaptive conversion Algorithm 1 will convert toward the SMR+CMR cache direction. The conversion direction can be verified in Figure 15, where the CMR zones' count is shrinking while the counts of SMR zones and the CMR cache are growing.

Although the fixed configuration has a good performance on the tested traces, such a fixed configuration is not generic to fit other potential workloads. For write-intensive traces (say, NSW-Range ratio r is way larger than the break-even point r^*, Section 5.2), the adaptive conversion algorithm will convert in the opposite direction of Figure 15, i.e., it will convert toward the CMR direction. Otherwise, converting to the SMR+Cache direction will create a cache space that is not big enough to hold the NSW data, increasing the intensity of the cache eviction and SMR update overhead. Therefore, FluidSMR can fit both the CMR-favorable workload and the Cache-favorable workload. Further, FluidSMR can deal with a changing workload by adapting the format accordingly.

Figure 16 demonstrates how FluidSMR adaptively converts the recording format under a synthesized phase-changing workload. In this experiment, the H-Partition has a total capacity of 428 GB and a usage ratio of 90%, the same as the settings of Figure 15. This workload is write-only with four phases (100 hours each). The first and the third phases are Cache-favorable, where every zone has a 20-MB NSW Range (thus a low NSW Range ratio of 7.8%, given a 256-MB zone size). The second and the fourth phases are CMR-favorable, where the first 80G of zones has an NSW Range of 240 MB per zone (thus a high NSW-Range ratio of 93.8%). Please note that in the second and fourth CMR-favorable phase, the 80 GB of zones can fit into the size of the CMR region (85 GB) if the CMR region is not converted into SMR+Cache. However, even if all the CMR region is converted into SMR+Cache, the total NSW ranges (75 GB) cannot fit into the Cache (30 GB). In the figure, we can see FluidSMR successfully identifies the changes of workload and adapts the sizes of CMR, SMR, and Cache accordingly. In the first and third phases, the H-Partition converts toward the SMR+Cache direction, whereas in the second and fourth phases, the H-Partition is converting in the opposite direction toward the CMR region. To evaluate the performance, we also run the same phase-change workload against fix-sized baselines `fix 0.2`, `fix 0.4`, `fix 0.6`, and `fix 0.8` (fix-sized baselines are defined in the same way as that in Figure 14). The average latency

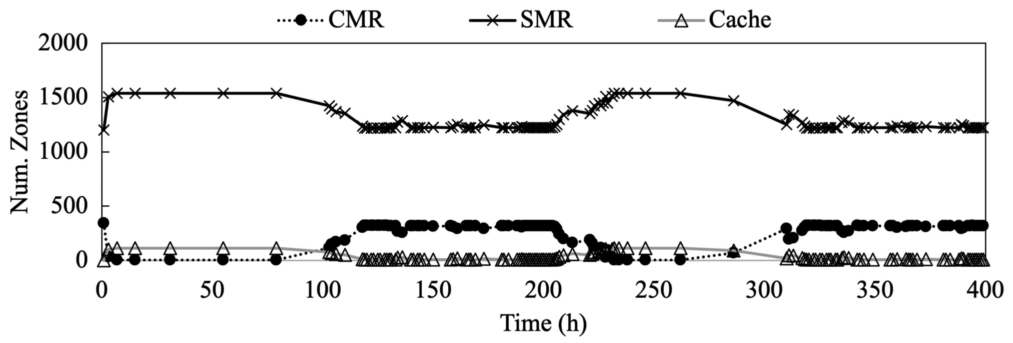

Fig. 16. Real-time size (in #zones) of CMR zones, SMR zones, and CMR Cache during the adaptive process in a four-phase synthesized workload. The first and third phases favor a large cache, and the second and fourth phases favor a large CMR region.

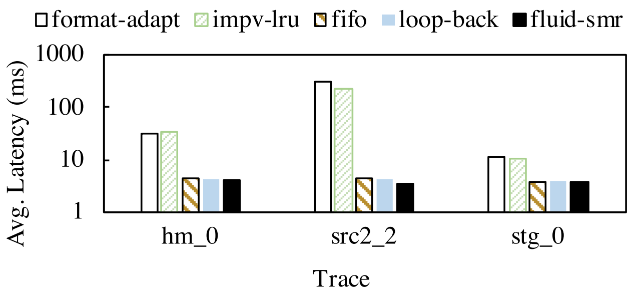

Fig. 17. Cache Optimization performance of FluidSMR compared with baselines. Please note the log scale in the y-axis.

numbers of `fix 0.2`, `fix 0.4`, `fix 0.6`, and `fix 0.8` are all higher than that of FluidSMR, by 11.2%, 5.3%, 17.5%, and 33.8% respectively. This is because a fixed format can fit at most one of the two types—CMR-favorable or Cache-favorable workload—but cannot fit both. Therefore, a fix format configuration leads to sub-optimal performance. On the contrary, FluidSMR can change the recording format with the workload and find the sweet-spot in both cases, and thus can produce better performance than any fix-format configuration.

Besides the benefit of being generic for various Hybrid SMR workloads, for the traditional non-Hybrid SMR drives without the conversion capability, the adaptive conversion algorithm can also help to figure out a reasonable CMR/SMR/Cache configuration for SMR disk vendors when designing the disk product for customers with their particular workload.

7.3 Performance Analysis Comparing FluidSMR and FormatAdapt

In this section, we evaluate the effectiveness of each of the optimization in FluidSMR (`fluid-smr`) compared to `format-adapt`. First, on top of `format-adapt` we apply the improved block-based LRU policy, denoted as `imprv-lru`). Then we optimize the improved LRU to in-place FIFO policy (`fifo`), and then to loop-back log policy (`loop-back`). Finally, we adopt the more efficient occupancy-based zone-swapping to get the final scheme `fluid-smr`. The the usage is set to 99.9% for a stress test. In such usage, the majority of the H-Partition is going to be in SMR format to hold the user data. The representative results (trace `hm_0`, `src2_2`, and `stg_0`) are plotted in Figure 17.

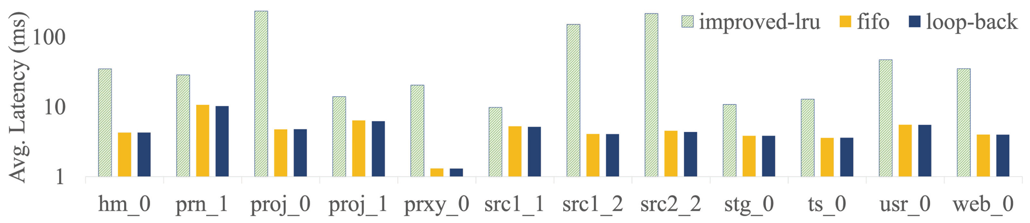

Fig. 18. Caching policy evaluation of FluidSMR. Zone-swapping and format conversion are disabled to rule out irrelevant influencing factors. Please note the log scale in the y-axis.

As seen in the figure, `imprv-lru` has a better performance for the workloads of `src2_2` and `stg_0` (27% and 7% reduction in average latency). This is because the improved block-based LRU reclaims more space using one zone re-write. However, `imprv-lru` has a slightly inferior performance in trace `hm_0`, because the evicted blocks are accessed again in a short time and get fetched back to the cache. So, it does not create extra free cache space but increases the migration cost. `fifo` reduces the average latency from the improved block-based LRU by 1.9×, 2.0×, and 1.7× in the traces `hm_0`, `src2_2`, and `stg_0` respectively, because it does not fragment the on-disk caching space. While the loop-back log (`loop-back`) performs similar to `fifo` in workloads `hm_0` and `stg_0`, it is able to reduce the latency by 3.8% for trace `src2_2`. Finally, although `fluid-smr` has a similar performance with `loop-back` for `stg_0`, it reduces the latency by 3.9% and 20.5% for traces `hm_0` and `src2_2` respectively, because it leverages the occupancy-based zone-swapping for fine-tuning the data allocation during the time between two format conversion processes.

7.4 Caching Policy Evaluation

In this section, we evaluate the loop-back log caching policy (`loop-back`) against two caching baselines: the improved block-based LRU (`impr-lru`), and the in-place FIFO policy (`fifo`). Zone-swapping and format conversion are disabled to rule out irrelevant influencing factors. The usage is set to 99% for stress test (same as Section 7.3), and the Cache size is set to 0.02% of the SMR region size. The average latency is measured and summarized in Figure 18.

As seen in the figure, compared with `impr-lru`, `fifo` reduces the average latency by from 1.4× (`rsrch_0`, write footprint 10 GB) to 45× (`proj_0`, write footprint 144 GB). The difference in performance improvement is due to the difference in write footprint: The larger the write footprint, the more data are redirected to Cache, and the greater potential for the Cache to improve the performance. Comparing `loop-back` and `fifo`, `loop-back` shows positive overall impact (average latency reduction of 1.1% among all traces). `loop-back` is able to reduce the latency from `fifo` in 75% of the traces (with max improvement at `prn_1`, a 4.4% reduction of average latency and 7% decrease in the zone read-modify-writes). There are three (out of twelve) traces show a slight regression in performance, because the loop-back policy requires more resource to predict the hot zones than the simple FIFO scheme and the extra cost offsets the benefit in these cases. However, the greatest regression is still small—an 0.75% increase in average latency (`ts_0`). Overall, loop-back policy shows positive impact, and further investigation on better ways to distinguish hot zones in loop-back policy and how to adaptively fall back to standard FIFO according to the workload characteristics are left as future work.

7.5 Quantized Migration Evaluation

In this subsection, we evaluate the quantized migration's effectiveness by comparing the performance of the application IOs with and without concurrent valid migration IOs using trace `proj_1`.

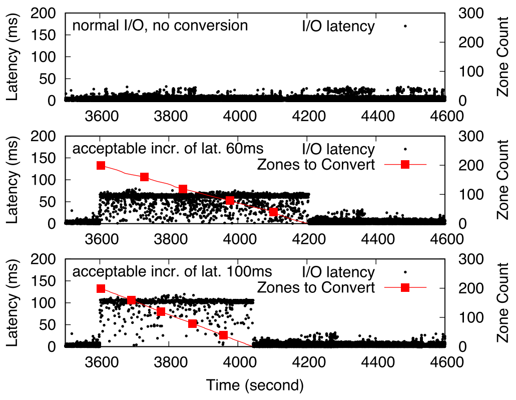

Fig. 19. Evaluating the quantized migration mechanism. Larger acceptable increase of latency leads to shorter conversion time.

In Figure 19, the latency of every request (left y-axis) is depicted as a black dot. The top figure shows the normal IO performance without a space extension request (hence no conversion or data migration happening), and the middle and bottom figures show the cases where the application IO is affected by the zone conversion and valid data migration. Specifically, the application has requested 25 GB of new space and the new 25 GB of empty guard range has to be created by converting 200 CMR zones into SMR format and migrating the valid data along. The application specifies *acceptable increase of latency* for every application IO based on their requirement of quality of services (Section 4.5). We test out two cases with the acceptable increase of latency set to 60 ms and 100 ms by the user. We also plot the number of remaining zones to convert and migrate (red line, right y-axis) to indicate the start and the end of the conversion.

The results show that FluidSMR performs the conversion and migration efficiently while controlling the increase of latency within the given bound. With a lower acceptable increase of latency (60 ms, middle figure), FluidSMR finishes the conversion and migration in a longer time (601 s). In contrast, the conversion time reduces to 432 s when the acceptable increase of latency rises to 100 ms. This is quite efficient given the fact that a blocking guard range repletion (meaning the drive blocks application IOs and exclusively performs conversion and valid data migration) also takes ~400 s to convert and migrate the same number of zones but blocks the application IOs for over 6 minutes.

8 RELATED WORK

8.1 Device-level Design for SMR

Caveat-Scriptor [29] exploits the interference model of SMR update in a finer-granularity. Amer et al. [2, 3] focused on various design dimensions to consider regarding SMR data layout management such as band usage and cleaning options. H-SWD [32, 38] introduces hot/cold data identification in data placement decisions using a circular log layout. Besides sector level solutions, static [24] and dynamic [25] track-level mapping are also exploited to improve SMR performance. The eventual establishment of a standard zoned block device interface [26, 27] enables the investigation of three SMR models, namely, **drive-managed (DM)**, **host-managed (HM)**, and **host-aware (HA)**. Skylight [1] studies the internals of the DM-SMR model using a unique hole-drilling technique to observe the disk arm movements inside the SMR drive. ZEA [35] explored the data management and application design of the HM-SMR model in storage systems and created a simplified file system to interface with LevelDB to demonstrate the effectiveness of the design. FluidSMR differs from these existing studies in that it handles the unique H-SMR issues, such as the conversion between CMR and SMR format in a hybrid SMR drive.

8.2 Caching Schemes in Magnetic Recording and Flash

Several studies use write cache inside the drive to reduce the SMR write overhead. An indirection system [10] proposed to use a shingled region as a set-associative cache for SMR writes and used a collection of algorithms to maintain an LBA-to-PBA mapping. Hall et al. [23] designed an *E-region* to serve as a buffer for accepting non-sequential writes that were originally targeted to the larger *I-region*. The Seagate translation layer manages a persistent cache as a log structure and each time evict the FIFO block and all the other block from the same SMR zone. Host Controlled Buffer [51, 54] and Virtual Persistent Cache [58, 59] leverage SMR zones to buffer and reform the write workload into SMR friendly write stream for HA-SMR drives. FluidSMR differs from these works in that it uses the CMR area instead of the SMR area as the cache. This has the benefit of updating in-place during write-hits, and has more flexibility when making eviction decisions. In the Flash setting, FASTer FTL [31] exploited the write requests' skewness and used the second-chance policy to retain hot data in the log instead of merging them back to their corresponding data blocks. FluidSMR's CMR cache differs in that one NSW can be repetitively updated in the same CMR location to reduce the SMR update overhead, while Flash's block-based log in FASTer cannot leverage such benefit of in-place updates. **Interlaced Magnetic Recording (IMR)** has update issues similar to SMR, and caching schemes for IMR has been investigated in References [17, 18, 22, 53, 55]. For example, TrackLace [53] leveraged unallocated top tracks as a write cache to indirect the updates aiming at the bottom tracks. Hajkazemi et al. [22] designed track-based translation layers for IMR drives and used a small range of non-interlaced bottom-only tracks as a persistent cache. Our work differs from the existing work in that we focus on the unique conversion characteristic and IO handling issues of the incipient hybrid SMR technology. Besides, existing caching schemes do not consider an on-disk cache with dynamic sizes.

8.3 Designs of SMR-based Applications

Filesystem-based solutions like HiSMRfs[28] use **Solid State Drives (SSDs)** to store frequently updated file system metadata and stores data in SMR drives. `ext-lazy` optimizes the `ext4` file system by reducing the NSWs due to metadata updates. There are also investigations about using SMR drives in key-value stores. SMRDB [39] is a filesystem free, direct-on-disk solution that manages the underlying disk in an SMR-friendly manner. Yao et al. designed GearDB [60] that leverages the gear compaction technique to achieve a GC-free Key-Value Store. SMR drives are also used in object stores such as SMORE [33, 34], which writes data and metadata in a log-structured format across multiple SMR drives, and stores object index on a flash device. FluidSMR can be adapted to those SMR-based applications if they use the emerging H-SMR drives.

8.4 Data Migration in Hybrid Storage System and RAID Systems

A hybrid storage system, consisting of both fast yet expensive storage devices and slow but inexpensive ones, can provide cost-effectiveness and high-performance [11, 14, 21, 30, 43]. Ge et al. analyzed enterprise block IO traces and proposed ChewAnalyzer that does data chunk placement across multiple storage pools [19]. TDDFS [9] is a two-tier storage system where data is deduplicated before migrating to the slower tier. As SMR drives are especially sensitive to workloads with updates, people try to team them up with SSDs to alleviate the performance impact of NSWs by relocating hot data to the SSD tier [45, 56]. ZoneTier [57] uses the SSD as both a faster storage tier and a cache to reduce the NSWs to SMR drives. FluidSMR differs in that it leverages H-SMR's unique property to create CMR caching space and does not require additional fast storage devices.

Data migration is also widely used in the RAID systems [47–49, 63, 64]. Zhang et al. [62] designed SLAS the uses both the reordering window and the sliding window to manage data migration. To speed up the data migration during RAID-5 capacity expansion, Mao et al. [36] proposed parity-based migration that only migrates blocks from a special parallelogram with one side consisting of only parity blocks. Code 5-6 [50] focused on the RAID level migration that converts an existing RAID-5 to a RAID-6. Code 5-6 used a novel MDS code to combine a new dedicated parity column with the original RAID-5 layout. FluidSMR differs in that it makes data placement decisions based on the CMR/SMR difference instead of the RAID-specific data layout.

9 CONCLUSION

The emerging hybrid SMR drives with a mixture of CMR and SMR zones provides the flexibility to convert the disk format on demand. We design and implement FluidSMR, an adaptive data and space management scheme that hides the H-SMR details and presents an extendable space to the application. FluidSMR supports a growing usage by a two-phase allocation design. With the proactive and quantized migration mechanisms, FluidSMR can reduce the response time of space allocation and limit the impact on the application IOs. Besides, FluidSMR can adjust the format configuration dynamically by adaptive conversion according to a workload's current situation. Finally, FluidSMR leverages the proposed loop-back caching and occupancy-based zone-swapping scheme to further reduce the overhead of handling NSWs. Evaluations show that FluidSMR can effectively adjust the cache size and reduce the average IO latency compared with the baseline schemes under different scenarios. It also demonstrates that FluidSMR can perform a space allocation request efficiently with a controllable tradeoff between the valid data migration and the application IOs.

REFERENCES

[1] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. 2015. Skylight—A window on shingled disk operation. *ACM Trans. Stor.* 11, 4, Article 16 (Oct. 2015), 28 pages. https://doi.org/10.1145/2821511

[2] Ahmed Amer, JoAnne Holliday, Darrell D. E. Long, Ethan L Miller, Jehan-François Pâris, and Thomas Schwarz. 2011. Data management and layout for shingled magnetic recording. *IEEE Trans. Magn.* 47, 10 (2011), 3691–3697.

[3] Ahmed Amer, Darrell D. E. Long, Ethan L. Miller, J.-F. Paris, and S. J. T. Schwarz. 2010. Design issues for a shingled write disk system. In *Proceedings of the IEEE Conference on Mass Storage Systems and Technologies (MSST'10).*

[4] Timothy G. Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: A database benchmark based on the Facebook social graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data.* 1185–1196.

[5] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426.

[6] Bill Boyle and Curtis E. Stevens. [n.d.]. Realms API. Retrieved from http://www.t10.org/cgi-bin/ac.pl?t=d&f=17-158r1.pdf.

[7] John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. 2008. *The disksim Simulation Environment Version 4.0 Reference Manual* (cmu-pdl-08-101). Parallel Data Laboratory (2008), 26.

[8] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. 2020. Characterizing, modeling, and benchmarking RocksDB key-value workloads at Facebook. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20).* 209–223.

[9] Zhichao Cao, Hao Wen, Xiongzi Ge, Jingwei Ma, Jim Diehl, and David H. C. Du. 2019. TDDFS: A tier-aware data deduplication-based file system. *ACM Trans. Stor.* 15, 1 (2019), 1–26.

[10] Yuval Cassuto, Marco A. A. Sanvido, Cyril Guyot, David R. Hall, and Zvonimir Z. Bandic. 2010. Indirection systems for shingled-recording disk drives. In *Proceedings of the IEEE Conference on Mass Storage Systems and Technologies (MSST'10).*

[11] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2011. Hystor: Making the best use of solid state drives in high performance storage systems. In *Proceedings of the International Conference on Supercomputing.* ACM, 22–32.

[12] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. 1997. Computational geometry. In *Computational Geometry.* Springer, 1–17.

[13] Dropbox. 2018. Extending Magic Pocket Innovation with the First Petabyte Scale SMR Drive Deployment. Retrieved November 2019 from https://blogs.dropbox.com/tech/2018/06/extending-magic-pocket-innovation-with-the-first-petabyte-scale-smr-drive-deployment/.

[14] Ziqi Fan, Fenggang Wu, Jim Diehl, David H. C. Du, and Doug Voigt. 2018. CDBB: An NVRAM-based burst buffer coordination system for parallel file systems. In *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 1.

[15] Timothy Feldman. [n.d.]. Flex Overview. Retrieved from http://t13.org/Documents/UploadedDocuments/docs2018/f17156r0-Flex_Overview.pdf.

[16] Tim Feldman. 2018. Flex dynamic recording. *USENIX ;login:* 43, 1 (2018).

[17] Kaizhong Gao, Wenzhong Zhu, and Edward Gage. 2016. Write management for interlaced magnetic recording devices. US Patent 9,508,362.

[18] Kaizhong Gao, Wenzhong Zhu, and Edward Gage. 2017. Interlaced magnetic recording. US Patent 9,728,206.

[19] Xiongzi Ge, Xuchao Xie, David H. C. Du, Pradeep Ganesan, and Dennis Hahn. 2018. Chewanalyzer: Workload-aware data management across differentiated storage pools. In *Proceedings of the IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'18)*. IEEE, 94–101.

[20] Google. [n.d.]. Dynamic Hybrid-SMR: An OCP Proposal to Improve Data Center Disk Drives. Retrieved from https://blog.google/products/google-cloud/dynamic-hybrid-smr-ocp-proposal-improve-data-center-disk-drives/.

[21] Jorge Guerra, Himabindu Pucha, Joseph S. Glider, Wendy Belluomini, and Raju Rangaswami. 2011. Cost effective storage using extent based dynamic tiering. In *Proceedings of the USENIX Conference on File and Storage TechnologiesFAST*, Vol. 11. 20–20.

[22] Mohammad Hossein Hajkazemi, Ajay Narayan Kulkarni, Peter Desnoyers, and Timothy R. Feldman. 2019. Track-based translation layers for interlaced magnetic recording. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'19)*. 821–832.

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

[24] Weiping He and David H. C. Du. 2014. Novel address mappings for shingled write disks. In *Proceedings of the HotStorage'14.*

[25] Weiping He and David H. C. Du. 2017. SMaRT: An approach to shingled magnetic recording translation. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17).*

[26] INCITS T10 Technical Committee. 2015. Information Technology—Zoned Block Commands (ZBC). Retrieved from http://www.t10.org/drafts.htm.

[27] INCITS T13 Technical Committee. [n.d.]. Zoned-device ATA Command Set (ZAC) working draft.

[28] Chao Jin, Wei-Ya Xi, Zhi-Yong Ching, Feng Huo, and Chun-Teck Lim. 2014. HiSMRfs: A high performance file system for shingled storage array. In *Proceedings of the 30th Symposium on Mass Storage Systems and Technologies (MSST'14).* 1–6. https://doi.org/10.1109/MSST.2014.6855539

[29] Saurabh Kadekodi, Swapnil Pimpale, and Garth A. Gibson. 2015. Caveat-scriptor: Write anywhere shingled disks. In *Proceedings of the ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'15).*

[30] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *Proceedings of the IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems.* IEEE, 227–236.

[31] Sang-Phil Lim, Sang-Won Lee, and Bongki Moon. 2010. FASTer FTL for enterprise-class flash memory SSDs. In *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI'10).* IEEE, 3–12.

[32] Chung-I. Lin, Dongchul Park, Weiping He, and David H. C. Du. 2012. H-SWD: Incorporating hot data identification into shingled write disks. In *Proceedings of the 20th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'12).*

[33] Peter Macko, Xiongzi Ge, John Haskins Jr, James Kelley, David Slik, Keith A. Smith, and Maxim G. Smith. 2017. SMORE: A Cold Data Object Store for SMR Drives (Extended Version). arXiv:1705.09701. Retrieved from https://arxiv.org/abs/1705.09701.

[34] Peter Macko, Xiongzi Ge, J. Kelley, D. Slik, et al. 2017. SMORE: A cold data object store for SMR drives. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST'17).*

[35] Adam Manzanares, Noah Watkins, Cyril Guyot, Damien LeMoal, Carlos Maltzahn, and Zvonimr Bandic. 2016. ZEA, A data management approach for SMR. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'16).*

[36] Yu Mao, Jiguang Wan, Yifeng Zhu, and Changsheng Xie. 2014. A new parity-based migration method to expand RAID-5. *IEEE Trans. Parallel Distrib. Syst.* 25, 8 (2014), 1945–1954.

[37] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Trans. Stor.* 4, 3 (2008), 10.
[38] Dongchul Park, Chung-I Lin, and David H. C. Du. 2012. H-SWD: A novel shingled write disk scheme based on hot and cold data identification. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'12).*
[39] Rekha Pitchumani, James Hughes, and Ethan L. Miller. 2015. SMRDB: Key-value data store for shingled magnetic recording disks. In *Proceedings of the 8th ACM International Systems and Storage Conference.* ACM, 18.
[40] Goldwyn Rodrigues. 2014. Flushing out pdflush. Retrieved from https://lwn.net/Articles/326552/.
[41] Chris Ruemmler and John Wilkes. 1994. An introduction to disk drive modeling. *Computer* 27, 3 (1994), 17–28.
[42] Seagate Technology. [n.d.]. New Flex Dynamic Recording Method Redefines the Data Center Hard Drive. Retrieved from https://blog.seagate.com/intelligent/new-flex-dynamic-recording-method-redefines-data-center-hard-drive/.
[43] John D. Strunk. 2012. Hybrid aggregates: Combining SSDs and HDDs in a single storage pool. *ACM SIGOPS Operat. Syst. Rev.* 46, 3 (2012), 50–56.
[44] Timothy Feldman. 2017. Flex Device Interface. Retrieved March 2020 from http://t13.org/Documents/UploadedDocuments/docs2018/f18101r0-Flex_Device_Interface.pdf.
[45] Chunling Wang, Dandan Wang, Yupeng Chai, Chuanwen Wang, and Diansen Sun. 2017. Larger, cheaper, but faster: SSD-SMR hybrid storage boosted by a new SMR-oriented cache framework. In *Proceedings of the IEEE Symposium Mass Storage Systems and Technology(MSST'17).*
[46] Western Digital. [n.d.]. Dynamic Hybrid SMR. Retrieved from https://blog.westerndigital.com/dynamic-hybrid-smr/.
[47] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 1996. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.* 14, 1 (1996), 108–136.
[48] Chentao Wu and Xubin He. 2012. GSR: A global stripe-based redistribution approach to accelerate RAID-5 scaling. In *Proceedings of the 41st International Conference on Parallel Processing.* IEEE, 460–469.
[49] Chentao Wu, Xubin He, Jizhong Han, Huailiang Tan, and Changsheng Xie. 2012. SDM: A stripe-based data migration scheme to improve the scalability of RAID-6. In *Proceedings of the IEEE International Conference on Cluster Computing.* IEEE, 284–292.
[50] Chentao Wu, Xubin He, Jie Li, and Minyi Guo. 2015. Code 5-6: An efficient MDS array coding scheme to accelerate online RAID level migration. In *Proceedings of the 44th International Conference on Parallel Processing.* IEEE, 450–459.
[51] Fenggang Wu, Ziqi Fan, Ming-Chang Yang, Baoquan Zhang, Xiongzi Ge, and David H. C. Du. 2017. Performance evaluation of host aware shingled magnetic recording (HA-SMR) drives. *IEEE Trans. Comput.* 66, 11 (2017), 1932–1945.
[52] Fenggang Wu, Bingzhe Li, Zhichao Cao, Baoquan Zhang, Ming-Hong Yang, Hao Wen, and David H. C. Du. 2019. ZoneAlloy: Elastic data and space management for hybrid SMR drives. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'19).*
[53] Fenggang Wu, Bingzhe Li, Baoquan Zhang, Zhichao Cao, Jim Diehl, Hao Wen, and David HC Du. 2020. Tracklace: Data management for interlaced magnetic recording. *IEEE Transactions on Computers* 70, 3 (2020), 347–358.
[54] Fenggang Wu, Ming-Chang Yang, Ziqi Fan, Baoquan Zhang, Xiongzi Ge, and David H.C. Du. 2016. Evaluating host aware SMR drives. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'16).*
[55] Fenggang Wu, Baoquan Zhang, Zhichao Cao, Hao Wen, Bingzhe Li, Jim Diehl, Guohua Wang, and David H. C. Du. 2018. Data management design for interlaced magnetic recording. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'18).*
[56] Wenjian Xiao, Huanqing Dong, Liuying Ma, Zhenjun Liu, and Qiang Zhang. 2016. HS-BAS: A hybrid storage system based on band awareness of Shingled Write Disk. In *Proceedings of the IEEE 34th International Conference on Computer Design (ICCD'16).* IEEE, 64–71.
[57] Xuchao Xie, Liquan Xiao, and David H. C. Du. 2019. ZoneTier: A zone-based storage tiering and caching co-design to integrate SSDs with SMR drives. *ACM Trans. Stor.* 15, 3 (2019), 19.
[58] Ming-Chang Yang, Yuan-Hao Chang, Fenggang Wu, Tei-Wei Kuo, and David H. C. Du. 2017. Virtual persistent cache: Remedy the long latency behavior of host-aware shingled magnetic recording drives. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17).* IEEE, 17–24.
[59] Ming-Chang Yang, Yuan-Hao Chang, Fenggang Wu, Tei-Wei Kuo, and David H. C. Du. 2018. On improving the write responsiveness for host-aware SMR drives. *IEEE Trans. Comput.* 68, 1 (2018), 111–124.
[60] Ting Yao, Jiguang Wan, Ping Huang, Yiwen Zhang, Zhiwen Liu, Changsheng Xie, and Xubin He. 2019. GearDB: A GC-free key-value store on HM-SMR drives with gear compaction. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST'19).* 159–171.
[61] B. Zhang, M. Yang, X. Xie, and D. H. C. Du. 2020. Idler: I/O workload controlling for better responsiveness on host-aware shingled magnetic recording drives. *IEEE Trans. Comput.* (2020), 1–1. https://doi.org/10.1109/TC.2020.2966194
[62] Guangyan Zhang, Jiwu Shu, Wei Xue, and Weimin Zheng. 2007. SLAS: An efficient approach to scaling round-robin striped volumes. *ACM Trans. Stor.* 3, 1 (2007), 3–es.

[63] Guangyan Zhang, Weiman Zheng, and Jiwu Shu. 2009. ALV: A new data redistribution approach to RAID-5 scaling. *IEEE Trans. Comput.* 59, 3 (2009), 345–357.

[64] Weimin Zheng and Guangyan Zhang. 2011. Fastscale: Accelerate raid scaling by minimizing data migration. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST'11)*. 149–161.

Received May 2020; revised February 2021; accepted May 2021