Liu WG, Zeng LF, Feng D *et al.* ROCO: Using a solid state drive cache to improve the performance of a host-aware shingled magnetic recording drive. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(1): 61–76 Jan. 2019. DOI 10.1007/s11390-019-1899-7

ROCO: Using a Solid State Drive Cache to Improve the Performance of a Host-Aware Shingled Magnetic Recording Drive

Wen-Guo Liu[1], Ling-Fang Zeng[1,*], *Senior Member, CCF, Member, ACM, IEEE*
Dan Feng[1], *Senior Member, CCF, Member, ACM, IEEE*, and Kenneth B. Kent[2], *Senior Member, IEEE*

[1]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[2]*Faculty of Computer Science, University of New Brunswick, Fredericton, E3B 5A3, Canada*

E-mail: liuwenguo_hust@163.com; {lfzeng, dfeng}@hust.edu.cn; ken@unb.ca

Received January 8, 2018; revised October 9, 2018.

Abstract Shingled magnetic recording (SMR) can effectively increase the capacity of hard disk drives (HDDs). Host-aware SMR (HA-SMR) is expected to be more popular than other SMR models because of its backward compatibility and new SMR-specific APIs. However, an HA-SMR drive often suffers performance degradation under write-intensive workloads because of frequent non-sequential writes buffered in the disk cache. The non-sequential writes mainly come from update writes, small random writes and out-of-order writes. In this paper, we propose a hybrid storage system called ROCO which aims to use a solid state drive (SSD) cache to improve the performance of an HA-SMR drive. ROCO reorders out-of-order writes belonging to the same zone and uses the SSD cache to absorb update writes and small random writes. We also design a data replacement algorithm called CREA for the SSD cache. CREA first conducts zone-oriented hot/cold data identification to identify cold-cached zones and hot-cached zones, and then evicts data blocks belonging to colder zones with higher priorities that can be sequentially written or written through host-side read-modify-write operations. It gives the lowest priority to data blocks belonging to the hottest-cached zone that have to be non-sequentially written. Experimental results show that ROCO can effectively reduce non-sequential writes to the HA-SMR drive and improve the performance of the HA-SMR drive.

Keywords solid state drive (SSD) cache, host-aware shingled magnetic recording (HA-SMR) drive, zone-oriented block reordering, zone-oriented hot/cold data identification, data replacement algorithm

1 Introduction

Because of the limitation of thermal stability, the areal density of conventional magnetic recording will reach a limit at about 1 Terabit per square inch[1]. Shingled magnetic recording (SMR) can effectively improve the areal density of hard disk drives. SMR enables the usage of write heads with strong fields and high tolerances, and brings no significant cost impact[2]. However, a shortcoming arises when SMR implements data updates. The in-place updates are inefficient. When some data is updated in a track, other data in the subsequent tracks can be overlapped if not read out first.

There are three types of SMR drives: drive-managed SMR (DM-SMR) drives, host-aware SMR (HA-SMR) drives, and host-managed SMR (HM-SMR) drives[3,4]. A DM-SMR drive consists of a disk cache and multiple shingled bands. It hides its internal layout information from the host, and can be deployed without any modifications to the existing storage systems[5]. HA-SMR and HM-SMR drives are mainly composed of zones, which are ranges of consecutive non-overlapping logical block addresses (LBAs). HA-SMR drives can accept non-sequential writes by buffering them in the internal disk caches, and HM-SMR drives only accept sequential writes[6]. Both HA-SMR and HM-SMR drives expose their internal layout information to the host.

HA-SMR drives are expected to be more popular

Regular Paper
This work was supported by the National Natural Science Foundation of China under Grant No. 61472153.
*Corresponding Author

because they can supply higher and more predictable performance than DM-SMR drives, and they are more backward compatible with legacy software than HM-SMR drives. However, HA-SMR drives often suffer performance degradation under write-intensive workloads. The main reason is that when too many non-sequential writes are buffered in the disk cache, frequent data cleaning is triggered to migrate data from the disk cache to target zones through read-modify-write (RMW) operations. These RMW operations can consume the majority of resources in an HA-SMR drive and block the ongoing workload. Non-sequential writes mainly come from update writes, small random writes, and out-of-order writes. Most update writes and small random writes access only a few data blocks of a zone because of low spatial locality of a workload within a short time interval. They can be easily identified as non-sequential writes and sent to the disk cache of the HA-SMR drive. Out-of-order writes have good spatial locality under a workload within a short time interval; however, they arrive in the HA-SMR drive in an out-of-order state and the majority of them are identified as non-sequential writes[7].

Flash-based solid state drives (SSDs) or non-volatile memories (NVMs) have been widely used to improve the performance of conventional hard disk drives (HDDs) or SMR drives[8–10]. However, directly applying these solutions to HA-SMR drives based storage systems is challenging due to the special characteristics of HA-SMR drives[7]. In this paper, we propose ROCO, a hybrid storage system adopting zone-oriented block reordering and a colder-zone first cache replacement algorithm, to use an SSD to improve the performance of an HA-SMR drive. The SSD works as a write-back cache, and the HA-SMR drive works as the main storage. In addition to using the SSD cache to absorb update writes and small random writes, the key idea of ROCO is that it reorders out-of-order writes before they arrive in the HA-SMR drive, and evicts data blocks from the SSD cache to the HA-SMR drive according to the colder-zone first cache replacement algorithm (CREA). CREA first conducts zone-oriented hot/cold data identification to identify hot zones and cold zones, and then evicts data blocks belonging to colder zones with higher priorities that can be sequentially written or written through host-side read-modify-write operations. The main contributions of this paper are as follows.

• *Zone-Oriented Block Reordering.* We propose to implement the zone-oriented block reordering to reorder the out-of-order data blocks belonging to the same zone according to their start LBAs. This can reduce the number of non-sequential writes and increase the number of sequential writes to the HA-SMR drive.

• *Zone-Oriented Hot/Cold Data Identification.* Different cached zones have different access patterns because of their cached data blocks. We propose to use zone-oriented hot/cold data identification to identify hot or cold cached zones according to their respective numbers of cached non-overlapping data blocks and access counts of these data blocks.

• *Cache Replacement Algorithm — CREA.* To further reduce the non-sequential writes written to the HA-SMR drive, CREA gives the highest priorities to the data blocks belonging to colder cached zones that can be sequentially written, and gives higher priorities to the data blocks belonging to colder cached zones that can be written through host-side RMW operations. It gives the lowest priority to data blocks belonging to the hottest cached zone that have to be non-sequentially written.

The rest of the paper is organized as follows. The background and the motivation are presented in Section 2. We describe the design of ROCO in Section 3. The experimental results are presented in Section 4. We review the related work in Section 5 and conclude this paper in Section 6.

2 Background and Motivation

In this section, we provide the necessary background knowledge about HA-SMR drives, including the model of an HA-SMR drive and main operations on a zone. The knowledge motivates our research and facilitates our presentation of ROCO in the following subsections.

2.1 Host-Aware SMR Basics

2.1.1 *Model of a Host-Aware SMR Drive*

Fig.1 shows the model of an HA-SMR drive, which consists of thousands of zones and a disk cache. The entire capacity of the HA-SMR drive is organized into logically contiguous, non-overlapping zones, each of which is a range of consecutive LBAs. The zones can be classified into two types: conventional zones which are composed of conventional magnetic recording tracks, and shingled zones which are composed of SMR tracks. The conventional zones allow in-place updates, while a shingled zone can only be written in a log-structure way[7]. A few shingled zones compose the disk cache and the other shingled zones are used as write-pointer zones[5,6]. A write pointer zone maintains a write pointer which indicates the LBA where the next write should start within this zone. Writes starting at write pointers are seen as sequential writes, or they are seen as non-sequential writes. The majority of zones are write pointer zones; therefore in the rest of the paper we use "zones" to refer to "write pointer zones" unless otherwise stated.

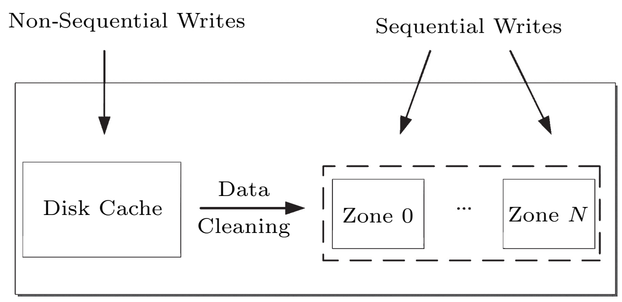

Fig.1. Model of an HA-SMR drive.

When write requests arrive in an HA-SMR drive, sequential writes are directly written into their target zones while non-sequential writes are buffered in the disk cache. When the disk cache is idle or full, data cleaning is implemented. Each time data blocks are selected belonging to the same zone to migrate to the target zone through read-modify-write operations[6]. After data cleaning, the disk cache needs to reclaim enough space for incoming non-sequential writes by implementing garbage collection operations[5,7].

2.1.2 Main Operations on a Zone

The main operations on a write pointer zone can be classified as three types: sequential-write operations, read-modify-write operations and "synthesized data" generated operations[7]. We use Fig.2 to describe these types of operations. In Fig.2, the disk cache and zone 0 are initially empty, and the write pointer references the starting LBA of zone 0. In the arriving data blocks targeting zone 0 shown in the block queue, blocks *A*–*D* are a set of consecutive data blocks, and blocks *A'*–*C'* are update blocks for blocks *A*–*C*. Block *G*, block *F*, block *E* and block *H* are out-of-order blocks of blocks *E*–*H* with higher LBAs than blocks *A*–*D*, and block *K* and block *M* are small random data blocks. Assume the starting LBA of block *A* is equal to the write pointer, then as shown in Fig.2(a), blocks *A*–*D* can be directly written to zone 0, and the write pointer references the next LBA after block *D*, where block *E* targets. In the following data blocks, the starting LBAs of blocks *A'*–*C'*, block *G* and block *F* are not equal with the write pointer; therefore they are sent to the disk cache. However, block *E* targets the place where the write pointer indicates and it can be sequentially written to zone 0. Hence, the write pointer moves to the next LBA where block *F* targets. The starting LBAs of block *H*, block *K* and block *M* are not equal with the write pointer either, and thus they are sent to the disk cache.

Fig.2(b) shows a read-modify-write operation which happens during the disk cache cleaning which migrates data blocks belonging to the same zone from the disk cache to their target zone. In-place updates are not supported by zone 0; therefore before blocks *A*–*C* are written to zone 0, blocks *D*–*E* have to be read out first, and then written back to zone 0 together with blocks *A'*–*C'*.

Fig.2(c) shows the generation of "synthesized data" on a zone. How "far" the write pointer advances within a zone depends on the largest LBA written by the host. However, from block *H* to block *M*, there are LBA gaps — block *I*, block *J* and block *L* are not written by the host. To guarantee that the write pointer advances to the LBA right after block *M*, synthesized data blocks are generated and written to those LBA gaps[7]. Fig.2(c) also shows a block reordering operation, which reorders out-of-order blocks, block *G*, block *F* and block *H*, into ordered blocks, block *F*, block *G* and block *H*, according to their start LBAs.

2.2 Motivation

As more and more non-sequential writes arrive in the HA-SMR drive, data cleaning and garbage collection need to be performed in the disk cache of the HA-SMR drive. Data cleaning migrates data blocks from the disk cache to their target zones through read-modify-write operations, and garbage collection makes space for incoming non-sequential writes through read-modify-write operations too when the free space in the disk cache is exhausted or nearly exhausted. Too many non-sequential writes cause frequent data cleaning and garbage collection operations, making the disk cache the main performance bottleneck and resulting in severe performance degradation of the HA-SMR drive[6,7].

As shown in Fig.2, non-sequential writes to a zone mainly come from update writes such as blocks *A'*–*C'*, small random writes such as block *F* and block *M*, and out-of-order writes such as blocks *E*–*H*. Update writes appear because of frequent access to a zone. Small ran-

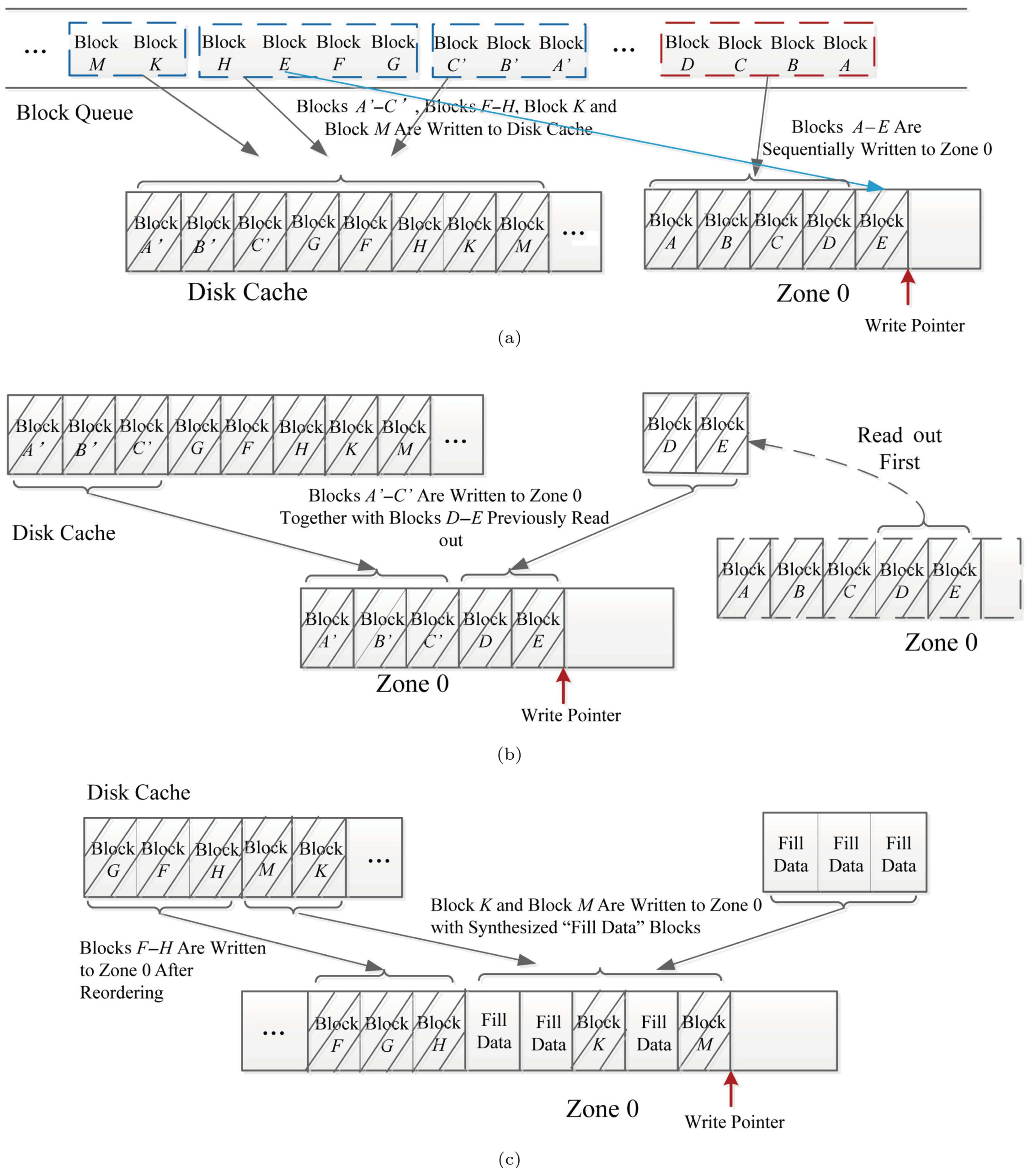
Block Queue
Block M, Block K, Block H, Block E, Block F, Block G, Block C', Block B', Block A', Block D, Block C, Block B, Block A
Blocks A'–C', Blocks F–H, Block K and Block M Are Written to Disk Cache
Blocks A–E Are Sequentially Written to Zone 0
Disk Cache
Zone 0
Write Pointer
(a)
Read out First
Blocks A'–C' Are Written to Zone 0 Together with Blocks D–E Previously Read out
(b)
Fill Data
Block K and Block M Are Written to Zone 0 with Synthesized "Fill Data" Blocks
Blocks F–H Are Written to Zone 0 After Reordering
(c)

Fig.2. Main operations on a zone. (a) Sequential and non-sequential data blocks are written to a zone and the disk cache respectively. (b) Read-modify-write operation on a zone during data cleaning process. (c) "Synthesized data" generated operation on a zone.

dom writes appear mainly because of low spatial locality in a zone within a short time interval, and out-of-order writes appear because they arrive out of order within a short time interval. This motivates us to reorder the out-of-order writes in the host and use a cache (such as flash-based SSDs and NVMs) to absorb update writes and small random writes.

When different zones are cached in the SSD (or NVM), the numbers of non-overlapping data blocks belonging to these zones are usually different, and different non-overlapping data blocks usually have different access counts too. This means that different cached zones may have different "zone heat"s in the SSD cache. This motivates us to conduct the zone-oriented hot/cold data identification, and evicts the data blocks belonging to colder cached zones from the SSD cache to the HA-SMR drive. This can reduce the amount of data blocks written to the HA-SMR drive, and even though the evicted data blocks are buffered in the disk cache, they are prone to triggering the disk cache cleaning less frequently than those belonging to hotter cached zones.

We also propose to reduce non-sequential writes through host-side read-modify-write (RMW) operations. Fig.3 illustrates how a host-side RMW operation is performed. In the blocks stored in the SSD cache, blocks *A'*–*C'*, blocks *E*–*H*, block *K* and block *M* belong to zone 0, and blocks *A'*–*C'* are update blocks for blocks *A*–*C*. When evicted to the HA-SMR drive, they are first read to the host. There are some gaps — block *D*, block *I*, block *J* and block *L* — among these blocks that are not written by the host. To guarantee the cached blocks can be sequentially written to zone 0, these gaps should be filled in the host. Block *D* can be directly read from zone 0 to the host; however, the locations of block *I*, block *J* and block *L* are beyond the write pointer of zone 0. In this situation, when reading data from these locations, the HA-SMR drive returns synthesized data as done in Fig.2(c)[7]. When implementing a host-side RMW operation, data blocks from the SSD cache, and block *D* and synthesized data from the HA-SMR drive belonging to zone 0 are read to the host. Subsequently, they are merged together in the host and sequentially written to zone 0 in the HA-SMR drive. To avoid too much overhead caused by zone read operations, we implement host-side RMW operations when the number of data blocks in the SSD cache belonging to the same zone exceeds a predefined threshold, such as 80% of the zone size.

Based on the above analysis, we can see that reducing non-sequential writes to the disk cache can fundamentally improve the performance of the HA-SMR drive, and there are three basic ways to reduce non-sequential writes: absorbing update writes, improving the spatial locality in a zone, and reordering out-of-order data blocks belonging to the same zone. We propose to use an SSD to cache writes for the HA-SMR drive. Even if the data writes to a zone have low spatial locality within a short time interval, they can have high spatial locality within a long time interval[11]. We create a block list for each cached zone. In this list, all the data blocks cached in the SSD belonging to the same zone are sorted in the ascending order of their start LBAs. When the SSD cache needs to evict data to the HA-SMR drive, it first conducts zone-oriented hot/cold identification, and then selects sequential writes belonging to colder cached zones as victims. If there are no sequential writes, it selects writes belonging to colder cached zones with the number of data blocks exceeding a predefined threshold, such as 80% of the zone size as victims, and evicts them sequentially to the corresponding zone through host-side read-modify-write operations. If there are no writes belonging to a zone meeting the above two conditions, it selects data blocks belonging to the coldest cached zone and evicts them to the disk cache of the HA-SMR drive.

3 Design and Implementation

In this section, we first present an overview of ROCO, and then describe the key data structure designed for ROCO. After that we present the zone-oriented hot/cold data identification, followed by the data replacement algorithm for the SSD cache.

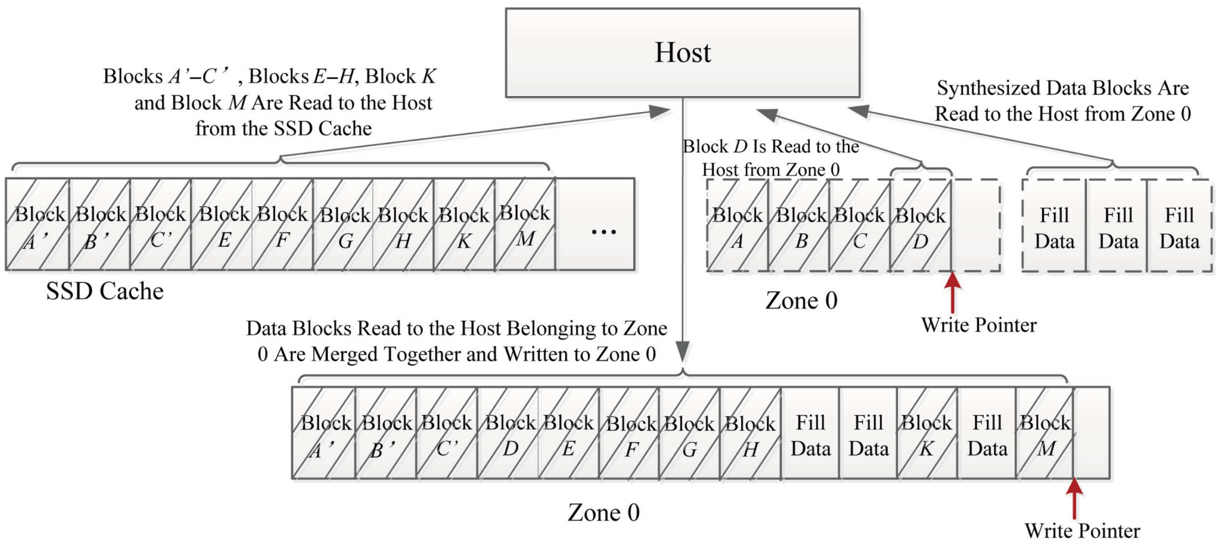
Host
Blocks A'–C', Blocks E–H, Block K and Block M Are Read to the Host from the SSD Cache
Synthesized Data Blocks Are Read to the Host from Zone 0
Block D Is Read to the Host from Zone 0
Block A' Block B' Block C' Block E Block F Block G Block H Block K Block M ...
SSD Cache
Block A Block B Block C Block D
Zone 0
Write Pointer
Fill Data Fill Data Fill Data
Data Blocks Read to the Host Belonging to Zone 0 Are Merged Together and Written to Zone 0
Block A' Block B' Block C' Block D Block E Block F Block G Block H Fill Data Fill Data Block K Fill Data Block M
Zone 0
Write Pointer

Fig.3. Host-side read-modify-write operation.

3.1 Overview

As shown in Fig.4, in ROCO the SSD serves as a write-back cache, and the HA-SMR drive serves as the main storage. ROCO mainly consists of four functional modules: zone-oriented data reordering module, zone-oriented hot/cold data identification module, data replacement module, and host-RMW operation module. The zone-oriented data reordering module sorts data blocks belonging to the same zone in the ascending order of their start LBAs in the SSD cache. The zone-oriented hot/cold data identification module identifies cold cached zones and hot cached zones according to the behaviors of cached data blocks. The data replacement module evicts data blocks from the SSD cache to the HA-SMR drive according to the algorithm described in Algorithm 1. The host-RMW operation module implements host-side read-modify-write operations on the zones in the HA-SMR drive selected by the data replacement module.

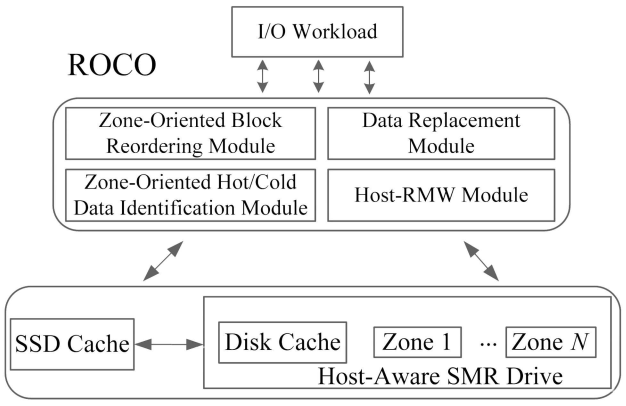

Fig.4. Overview of ROCO.

3.2 Key Data Structure

The key data structure used by ROCO is shown in Fig.5. The cached_zone_list is used to maintain the information of all the zones that are currently cached in the SSD, which consists of a number of *zone_entries*. In addition to the five variables including *zone_no*, *zone_block_count*, *zone_low_LBA*, *zone_access_count* and *zone_heat*, each *zone_entry* consists of a *zone_block_list*. The *zone_block_list* consists of several block_entrys, which represent non-overlapping data blocks that are currently cached in the SSD. Two non-overlapping data blocks have different *blk_start_LBA*s. The main variables in the key data structure are explained as follows.

Algorithm 1 . CREA: Data Replacement Algorithm for the SSD Cache

```
1: //Assume zone 0, zone 38, zone 153, ..., zone $N$ are cached
   in the SSD.
2: //$zone\_heat\_sort()$ : sort cached zones in the ascending order
   of their $zone\_heats$.
3: //$write\_pointer[]$ : indicating the LBA in zone $k$ where the
   next write should target in the HA-SMR drive.
4: //$seq\_write()$ : write data blocks belonging to zone $k$ sequen-
   tially from the SSD cache to zone $k$ in the HA-SMR drive.
5: //$reset()$ : reset the write pointer of a zone to its initial value.
6: //$host\_rmw\_write()$ : write data blocks belonging to zone $k$
   from the SSD cache to zone $k$ by host-side read-modify-write
   operations.
7: //$non\_seq\_write()$ : write data blocks belonging to zone $k$
   from the SSD to the disk cache in the HA-SMR drive.
8: //$seq\_state$: if there is a zone that can be sequentially writ-
   ten, the value is set to 1.
9: //$host\_rmw\_state$: if there is a zone that can be written by
   host-RMW operations, the value is set to 1.
10: $seq\_state = 0$, $host\_rmw\_state = 0$;
11: for $k \in \{0, 38, 153, ..., N\}$ do
12:     $zone[k].zone\_heat = \frac{zone[k].zone\_access\_count}{zone[k].zone\_block\_count}$;
13: end for
14: $zone\_heat\_sort(\{0, 38, 153, ..., N\})$;
15: for $k \in \{38, 42, 16, ..., N\}$ do
16:     if ($zone[k].zone\_low\_LBA == write\_pointer[k]$) &&
        ($zone[k].zone\_block\_count < zone\_size$) then
17:         $seq\_write(zone[k])$;
18:         $seq\_state = 1$;
19:         break ;
20:     end if
21:     if ($zone[k].zone\_block\_count == zone\_size$) then
22:         $reset(write\_pointer[k])$;
23:         $seq\_write(zone[k])$;
24:         $seq\_state = 1$;
25:         break ;
26:     end if
27: end for
28: if ($seq\_state == 0$) then
29:     for $k \in \{38, 42, 16, ..., N\}$ do
30:         if ($zone[k].zone\_block\_count \geqslant zone\_size \times 0.8$) then
31:             $host\_rmw\_write(zone[k])$;
32:             $host\_rmw\_state = 1$;
33:             break;
34:         end if
35:     end for
36: end if
37: if ($seq\_state == 0$ && $host\_rmw\_state == 0$) then
38:     $non\_seq\_write(zone[38])$;
39: end if
```

cached_zone_count: indicates the number of zones which are cached in the SSD.

zone_size: is a constant indicating the total number of non-overlapping data blocks a zone in the HA-SMR drive can hold.

zone_no: indicates the zone number of a zone cached

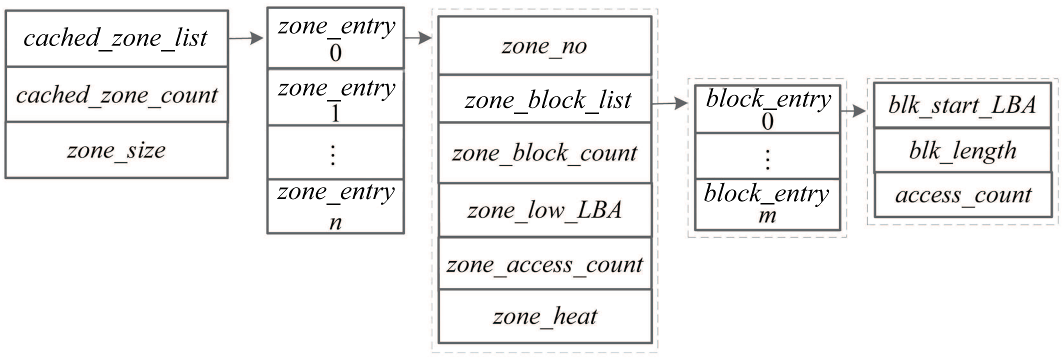

Fig.5. Key data structure in ROCO.

in the SSD.

zone_block_count: indicates the number of non-overlapping data blocks of a zone cached in the SSD.

zone_low_LBA: indicates the starting LBA of the data block with the lowest LBA among data blocks belonging to the same zone that are cached in the SSD.

blk_start_LBA: indicates the starting LBA of a data block.

blk_length: indicates the length of a data block.

access_count: indicates the access count of a data block when cached in the SSD.

zone_access_count: indicates the total number of access counts of non-overlapping data blocks of a zone that are currently cached in the SSD.

zone_heat: indicates the access frequency of a zone reflected by the data blocks that are currently cached in the SSD.

When a write request arrives in ROCO, it is first split into several consecutive data blocks. For each data block, *zone_no* is calculated according to its *blk_start_LBA*, and the data reordering module checks if a corresponding zone entry exists in the cached_zone_list. If not, a zone entry is created for the zone newly cached in the SSD and added to the cached_zone_list in a zone-oriented first-in-first-out (FIFO) order, and the value of *cached_zone_count* increases by 1. After that, the data reordering module checks if an arriving data block exists in the zone_block_list of the corresponding zone according to the *blk_start_LBA* of the block. If not, a block entry is created for this data block and inserted to the zone_block_list in the ascending order of the *blk_start_LBA*, and the value of *zone_block_count* increases by 1. Each time when a data block arrives in the SSD cache, the values of the corresponding *access_count* and *zone_access_count* increase by 1 respectively. When there is no free space in the SSD cache, the data replacement module evicts data blocks belonging to the same zone according to the algorithm described in Algorithm 1 from the SSD to the HA-SMR drive. After data blocks are evicted, the corresponding entries are deleted.

3.3 Zone-Oriented Hot/Cold Data Identification

The zone-oriented FIFO algorithm cleans all the data blocks belonging to the oldest zone buffered in the SSD cache during the cache cleaning process. However, different zones have different access frequencies in the SSD cache. Cleaning cold zones usually brings less data migration than cleaning hot zones and results in better performance. We identify the cold-cached zones and hot-cached zones by computing the *zone_heat* for each cached zone.

The numbers of cached data blocks belonging to different cached zones are often different, and the access counts of different cached data blocks are often different too. We compute the *zone_heat* for a cached zone by combining the number and the access counts of the cached data blocks together belonging to the cached zone. We use Fig.6 to illustrate how to compute the *zone_heat*. Fig.6(a) shows zone 0 which consists of 4 096 non-overlapping LBAs. Each LBA corresponds to a data block. Fig.6(b) shows 3 000 data blocks belonging to zone 0, which are currently cached in the SSD. An *access_count* is maintained for each cached data block which records the access count of each cached data block. The *zone_access_count* denotes the sum of the access counts of all the cached data blocks. Then the *zone_heat* can be calculated as

$$\begin{aligned} zone\_heat &= \frac{access\_count\ 0 + \dots + access\_count\ 3090}{zone\_block\_count} \\ &= \frac{zone\_access\_count}{zone\_block\_count}. \end{aligned}$$

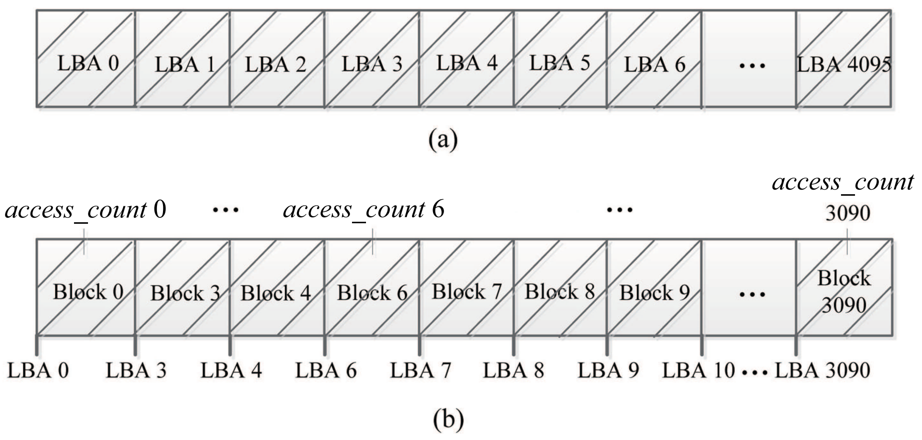

Fig.6. Write pointer zone with 3 000 data blocks cached in the SSD. (a) Zone 0 consisting of 4 096 non-overlapping LBAs. (b) 3 000 data blocks belonging to zone 0 that are currently cached in the SSD.

3.4 Data Replacement Algorithm for the SSD Cache

When the SSD cache needs to evict data, we propose to first implement the zone-oriented hot/cold data identification by computing the *zone_heat* for each cached zone, and then sort all the currently cached zones in the ascending order of their *zone_heat*s. After that we select data blocks belonging to the colder zones that can be sequentially written to the HA-SMR drive. This is because keeping hotter zones cached in the SSD can reduce the data migration from the SSD cache to the HA-SMR drive, and sequential writes do not cause extra overhead to the HA-SMR drive. We identify two types of writes as sequential writes. The first type is that *zone_low_LBA* of a cached zone is equal to the corresponding write pointer and *zone_block_count* is less than *zone_size*. The second type is that *zone_block_count* is equal to *zone_size*, which means that the zone in the HA-SMR drive can be entirely overlapped. If there are no sequential writes, we select data blocks belonging to the colder zones that can be evicted through host-side RMW operations. Host-side RMW operations mainly utilize the host resources and do not cause internal RMW operations of the HA-SMR drive. If the above two kinds of data blocks do not exist in the SSD cache, we select to evict data blocks belonging to the coldest zone to the disk cache of the HA-SMR drive. Our proposed cache replacement algorithm CREA is described in Algorithm 1.

Assume data blocks belonging to zone 0, zone 38, zone 153, ..., and zone N are cached in the SSD. As shown in Algorithm 1, CREA first computes the *zone_heat* for each cached zone according to (1), and then invokes the *zone_heat_sort*() function to sort all the currently cached zones in the ascending order of their *zone_heat*s. After sorting, assume the order of the cached zones is zone 38, zone 42, zone 16, ..., and zone N. CREA checks if there are sequential data blocks from zone 38 to zone N. If zone[38].*zone_low_LBA* is equal to *write_pointer*[38] and zone[38].*zone_block_count* is less than *zone_size*, then all the consecutive data blocks starting from the block with *zone_low_LBA* belonging to zone 38 are sequentially written to zone 38 in the HA-SMR drive. The corresponding entries in the zone_block_list of zone 38 are deleted and the value of *seq_state* is set to 1. If zone[38].*zone_block_count* is equal to *zone_size*, it means that zone 38 can be entirely overlapped. Then the write pointer of zone 38 is reset to its initial value, and all data blocks belonging to zone 38 are sequentially written to the HA-SMR drive. The corresponding block entries are deleted from the zone_block_list of zone 38 and the value of *seq_state* is set to 1. If the *zone_low_LBA* of each cached zone is not equal to the corresponding write pointer and no zones in the HA-SMR drive can be entirely overlapped, CREA checks if there is a cached zone satisfying the host-side read-modify-write condition from zone 38 to zone N. If the *zone_block_count* of zone 38 is greater than 80% of *zone_size*, it first reads data blocks from the SSD cache as well as valid data blocks and synthesized data blocks (if needed) belonging to zone 38 from the HA-SMR drive to the host. The blocks are then merged together in the host, and then written back to zone 38 in the HA-SMR drive. All the entries in the zone_block_list of zone 38 are deleted and the value of *host_rmw_state* is set to 1. If there is not a zone that can be sequentially written or written by host-side RMW operations, CREA evicts data blocks belonging to the coldest zone — zone 38 here — to the disk cache of the HA-SMR drive and all the entries in the zone_block_list of zone 38 are deleted.

4 Experimental Results

In this section, we first describe the experimental setup, and then evaluate the performance of ROCO through trace-driven simulations.

4.1 Experimental Setup

We have implemented ROCO in a simulation environment based on Disksim and SSD extension[12]. The SSD is emulated based on SSD extension which emulates SLC NAND flash memory chip operations, and the page/block unit size is 4 KB/256 KB respectively. The total capacity of the SSD is 4 GB. Trying to achieve better performance with lower cost, we use 256 MB of the total capacity as the SSD cache. The HA-SMR drive is emulated based on the disk model of Maxtor Atlas 10K IV disk whose average read and write latency is 4.4 ms and 4.9 ms respectively, and the disk size is 146 GB. The HA-SMR drive uses about 0.3%–0.4% of the total space as the disk cache[7], and the disk cache size is set to 512 MB. The zone size is set to 16 MB. The disk cache implements a zone-oriented FIFO data cleaning policy and greedy algorithm based garbage collection policy[7]. The SSD is used as the write-back cache of the HA-SMR drive and implements CREA replacement algorithm. We implement ROCO at the device driver level, which intercepts all the incoming requests before they reach the underlying disks. In addition, we implement three other solutions: HA, Cache-LRU, and Cache-ZFIFO. HA is implemented on an HA-SMR drive and adopts a zone-oriented FIFO data cleaning policy and greedy algorithm based garbage collection policy in the disk cache. Cache-LRU and Cache-ZFIFO are solutions for the hybrid storage system consisting of an SSD cache and an HA-SMR drive. Cache-LRU implements a block-oriented LRU cache replacement algorithm for the SSD cache, which evicts data blocks to the HA-SMR drive in a least recently used (LRU) order. Cache-ZFIFO implements a zone-oriented FIFO cache replacement algorithm for the SSD, which evicts all the data blocks belonging to a zone to the HA-SMR drive in a zone-oriented FIFO order.

The traces used here are collected from enterprise servers at Microsoft Research (MSR) Cambridge[13]. The MSR traces consist of 36 traces and can be classified into two types: read-intensive traces and write-intensive traces. An HA-SMR drive suffers from performance degradation mainly because of the data migration during the disk cache cleaning when write requests arrive. Because read-intensive traces rarely trigger the disk cache cleaning in an HA-SMR drive, we select write-intensive traces as input traces. We use five write-intensive traces here since the other write-intensive traces show the similar experimental results. The main characteristics of the five traces are listed in Table 1.

Table 1. Characteristics of Traces

Trace	Total Write Size (GB)	Average Write Size (KB)	Write Ratio (%)
proj_0	144.27	34.78	87.52
src2_2	39.28	51.11	69.67
prn_0	45.19	8.50	89.21
stg_0	15.09	9.19	68.02
src1_2	44.15	32.51	83.41

4.2 Results

In this subsection, we first evaluate the impact of the zone-oriented block reordering on the performance of the HA-SMR drive, and then evaluate the impact of ROCO on the performance of the HA-SMR drive.

4.2.1 Impact of Zone-Oriented Block Reordering on the Performance of the HA-SMR Drive

The zone-oriented block reordering module reorders the data blocks belonging to the same zone according to their *blk_start_LBA*s, which can reduce the number of non-sequential writes and increase the number of sequential writes to the HA-SMR drive in varying degrees under different traces. The basic cache replacement algorithm for the hybrid storage system consisting of an SSD cache and an HA-SMR drive is the zone-oriented FIFO algorithm, and if the zone-oriented block reordering is implemented when data blocks are written to the SSD cache, we call this solution Reorder-CZFIFO. We show the impact of the zone-oriented block reordering on the HA-SMR drive by comparing the variations of data migration of the HA-SMR drive when Cache-ZFIFO and Reorder-CZFIFO are implemented respectively.

Using the size of written data of each trace (Trace_WSize) as a baseline, Fig.7 shows the variations of sequential data and non-sequential data written to

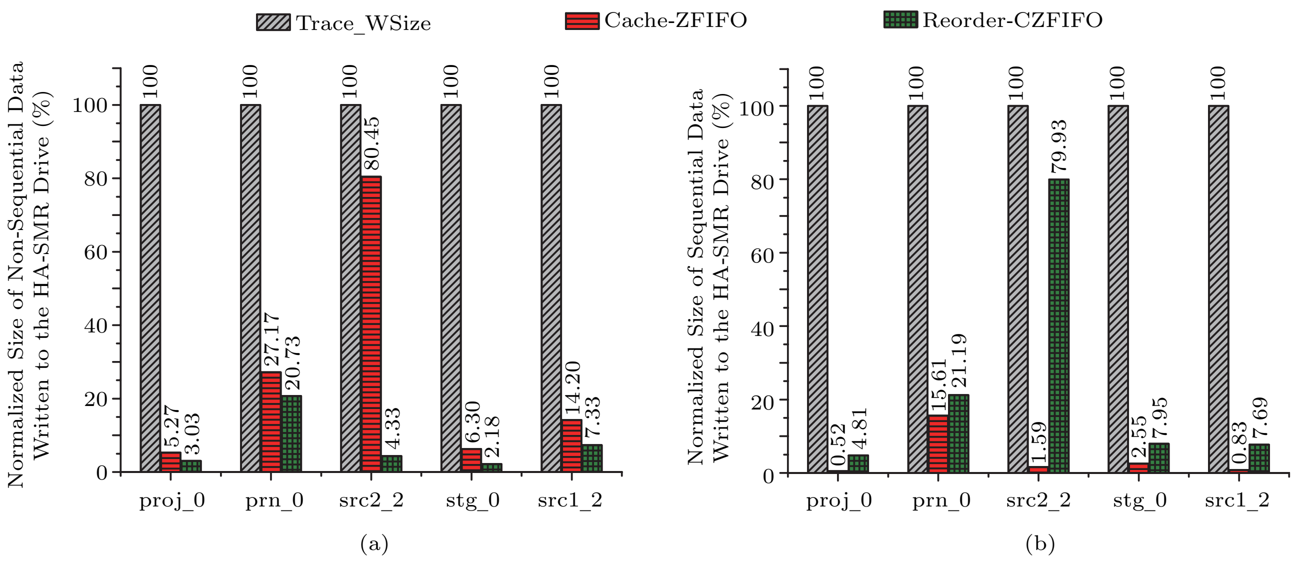

Fig.7. Impact of zone-oriented block reordering on variations of sequential writes and non-sequential writes under proj_0, prn_0, src2_2, stg_0 and src1_2. (a) Variations of non-sequential writes. (b) Variations of sequential writes.

the HA-SMR drive under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. In Fig.7(a) we can see that compared with simply implementing Cache-ZFIFO, implementing the zone-oriented block reordering can further reduce the ratios of non-sequential data from 5.27%, 27.17%, 80.45%, 6.3% and 14.2% to 3.03%, 20.73%, 4.33%, 2.18% and 7.33% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

Among the five write-intensive traces, src2_2 benefits the most from the zone-oriented block reordering. Although src2_2 has good zone level locality, it is less update-intensive than the other four traces, and the majority of data blocks of src2_2 belonging to a zone are out-of-order data blocks. The zone-oriented block reordering can translate the majority of non-sequential writes of src2_2 to sequential writes. Fig.7(b) shows that, compared with simply implementing Cache-ZFIFO, implementing Reorder-CZFIFO can further increase the ratios of sequential data from 0.52%, 15.61%, 1.59%, 2.55% and 0.83% to 4.81%, 21.19%, 79.93%, 7.95% and 7.69% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

The migrated data during the disk cache cleaning in the HA-SMR drive comes from three parts: data read from the disk cache and the zones, data synthesized because of "LBA gaps", and data written to the zones. Because the written data comes from the read data and the synthesized data, we only compare the sizes of the read data and the synthesized data under proj_0, prn_0, src2_2, stg_0 and src1_2 in Fig.8(a) and Fig.8(b) respectively.

From Fig.8(a) we can see that, compared with Cache-ZFIFO, Reorder-CZFIFO can reduce the ratios of read data generated during the disk cache cleaning in the HA-SMR drive from 8.5%, 89.31%, 103.37%, 2.42% and 16.36% to 6.16%, 60.37%, 4.36%, 0.99% and 4.72% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. This is because the zone-oriented block reordering translates more non-sequential writes to sequential writes, reduces the amount of data buffered in the disk cache, and finally reduces the frequency of the disk cache cleaning in the HA-SMR drive.

Because of the same reason, Fig.8(b) shows the similar results to those in Fig.8(a). From Fig.8(b) we can see that, compared with Cache-ZFIFO, Reorder-CZFIFO can reduce the ratios of synthesized data generated during the disk cache cleaning in the HA-SMR drive from 9.1%, 40.87%, 24.33%, 24.09% and 9.32% to 7.94%, 27.67%, 7.82%, 12.36% and 2.99% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

4.2.2 *Impact of ROCO on the Performance of the HA-SMR Drive*

To show the performance improvement of the HA-SMR drive by applying ROCO, we compare it with the other three solutions: HA, Cache-LRU and Cache-ZFIFO. As described in Subsection 4.1, HA is implemented on an HA-SMR drive which implements a zone-oriented FIFO data cleaning policy in the disk cache. Cache-LRU implements a block-oriented LRU cache replacement algorithm and evicts data blocks from the SSD cache to the HA-SMR drive in an LRU order.

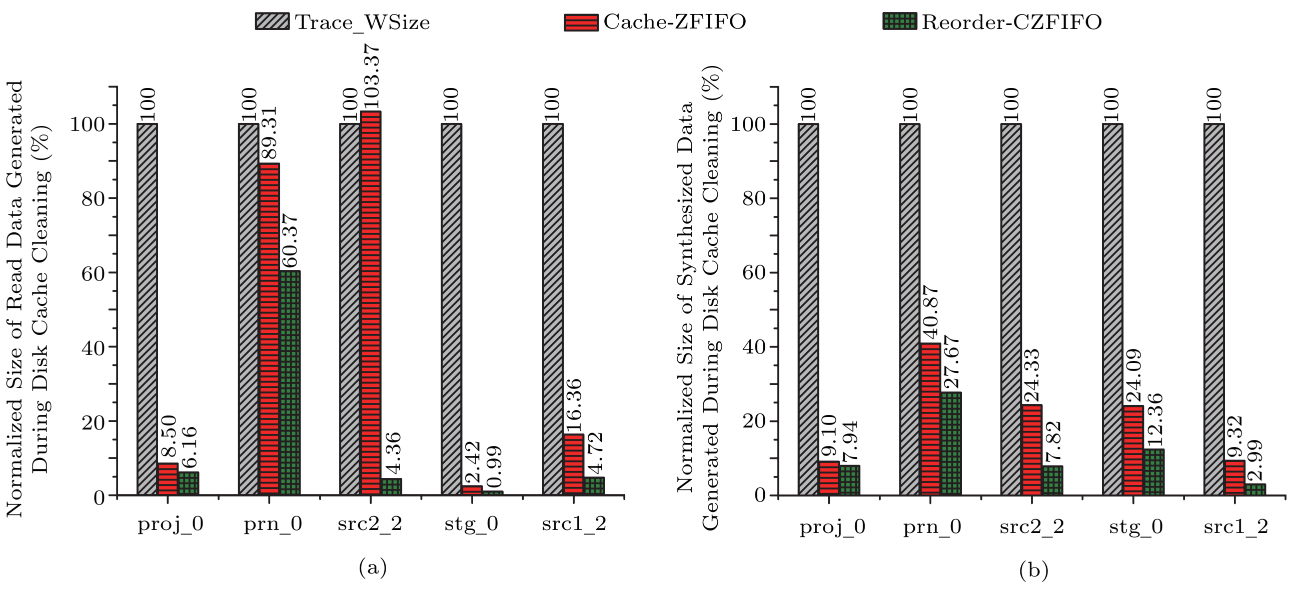

Fig.8. Impact of zone-oriented block reordering on data migration during disk cache cleaning under proj_0, prn_0, src2_2, stg_0 and src1_2. (a) Read data generated during disk cache cleaning. (b) Synthesized data generated during disk cache cleaning.

Cache-ZFIFO implements a zone-oriented FIFO cache replacement algorithm and evicts all the data blocks belonging to a zone from the SSD cache to the HA-SMR drive in a zone-oriented FIFO order.

Using the size of written data of each trace (Trace_WSize) as a baseline, Fig.9 shows variations of sequential data and non-sequential data written to the HA-SMR drive under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. From Fig.9(a) we can see that without an SSD cache, 99.68%, 96.48%, 99.2%, 98.97% and 99.72% of the written data are buffered in the disk cache of the HA-SMR drive under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. The main reason is that proj_0, prn_0, stg_0 and src1_2 have a lot of update writes and small random writes, and have poor spatial locality on zones within a short time interval. Although src2_2 has good spatial locality on zones within a short time interval, the majority of the arriving data blocks belonging to the same zone are out-of-order blocks. With an SSD cache and implementing a block-oriented LRU replacement algorithm, Cache-LRU reduces the ratios of non-sequential data to 13.07%, 38.94%, 96.71%, 8.59% and 20.4% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. This is because the SSD cache absorbs a lot of update writes and small random writes for proj_0, prn_0, stg_0 and src1_2. Cache-LRU does not work well for src2_2. This is because src2_2 is mainly composed of out-of-order data blocks and its update requests have poor spatial locality in the SSD cache.

Unlike Cache-LRU which evicts data blocks at the block level, Cache-ZFIFO evicts data blocks at the zone level, which each time evicts the data blocks belonging to a zone from the SSD cache to the HA-SMR drive when it is implemented. Cache-ZFIFO can gain better zone locality than Cache-LRU, and it reduces the ratios of non-sequential data buffered in the disk cache of the HA-SMR drive to 5.27%, 27.17%, 80.45%, 6.3% and 14.2% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

ROCO performs the best among the four solutions. It reduces the ratios of non-sequential data buffered in the disk cache of the HA-SMR drive to 0.86%, 8.68%, 0.02%, 0.52% and 2.01% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. In addition to the zone-oriented block reordering which can translate more non-sequential writes to sequential writes, the proposed cache replacement algorithm CREA first conducts zone-oriented hot/cold data identification for the cached zones, and then evicts data blocks belonging to colder zones that can be sequentially written or written through host-side read-modify-write operations. Combined with the zone-oriented hot/cold data identification, CREA not only can reduce the number of data blocks written to the HA-SMR drive, but also can reduce the frequency of disk cache cleaning by buffering data blocks belonging to colder zones in the disk cache of the HA-SMR drive.

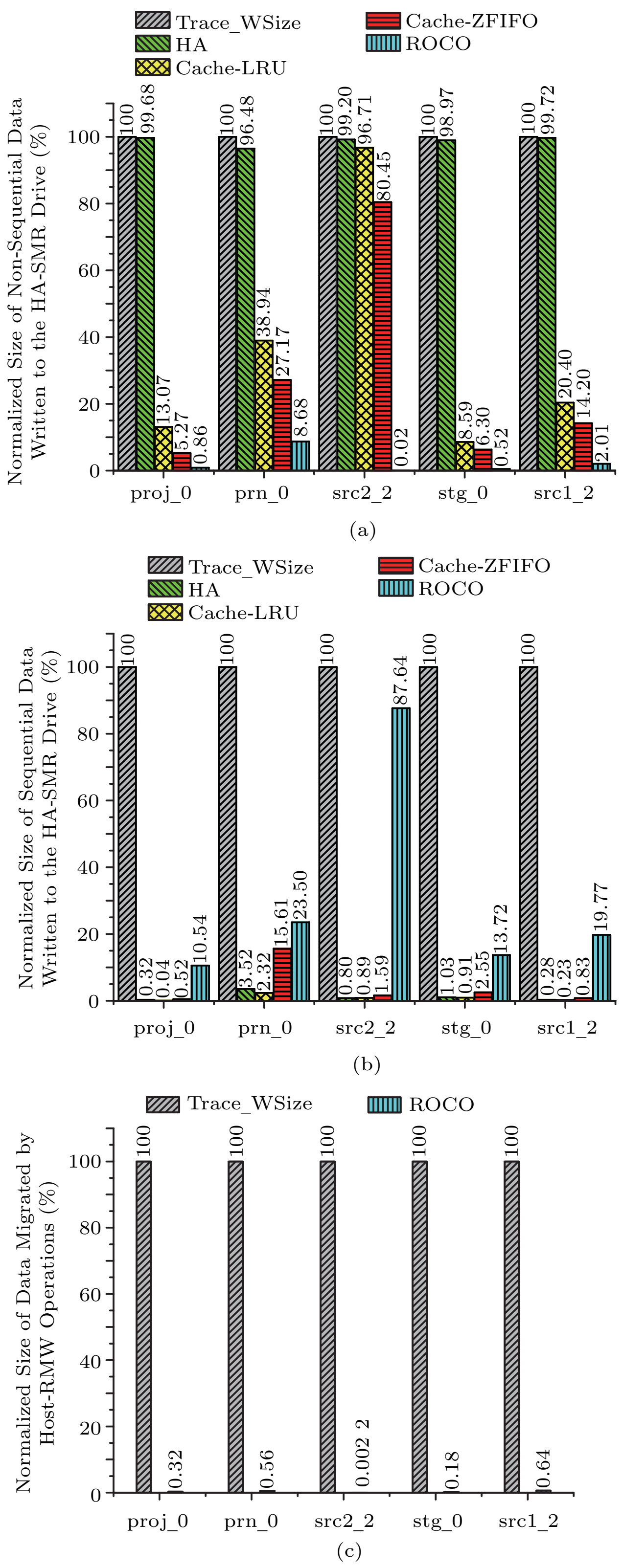

Fig.9. Impact of ROCO on variations of sequential writes and non-sequential writes under proj_0, prn_0, src2_2, stg_0 and src1_2. (a) Variations of non-sequential writes. (b) Variations of sequential writes. (c) Overhead of host-RMW operations.

Because of the same reasons analyzed above, from Fig.9(b) we can see that ROCO writes more sequential data to the HA-SMR drive than HA, Cache-LRU and Cache-ZFIFO. It increases the ratios of sequential data written to the HA-SMR drive from 0.32%, 3.52%, 0.8%, 1.03% and 0.28% to 10.54%, 23.5%, 87.64%, 13.72% and 19.77% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

Fig.9(c) shows the overheads of host-side RMW operations under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively when implementing ROCO. In addition to the zone-oriented block reordering and hot/cold data identification, the host-side RMW operations can further reduce the amount of non-sequential data written to the HA-SMR drive. When necessary, host-side RMW operations need to read extra data blocks from the HA-SMR drive and later write them back to their original locations. From Fig.9(c) we can see that the host-side RMW operations only migrate extra 0.32%, 0.56%, 0.002 2%, 0.18% and 0.64% of the total written data under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. The overhead is the least under src2_2. This is mainly because src2_2 has much better spatial locality and much fewer small random writes than the other four traces.

Fig.10 shows the variations of data migration during the disk cache cleaning in the HA-SMR drive when HA, Cache-LRU, Cache-ZFIFO and ROCO are implemented respectively.

From Fig.10(a) we can see that ROCO generates the least read data during the disk cache cleaning in the HA-SMR drive. It reduces the ratios of generated read data from 24.72%, 82.58%, 109.18%, 55.8% and 32.69% to 0.68%, 13.65%, 0.94%, 0.45% and 2.06% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively. Cache-LRU generates more read data than ROCO not only because it writes more non-sequential data to the HA-SMR drive, but also because its block-oriented eviction results in more frequent disk cache cleaning in the HA-SMR drive. Cache-ZFIFO generates more read data than ROCO mainly because ROCO effectively reduces the amount of non-sequential data by implementing the zone-oriented block reordering and CREA replacement algorithm.

Synthesized data is generated mainly because of small random writes. ROCO reduces the most small random writes by SSD absorbing, zone-oriented block reordering, and the CREA replacement algorithm, and as can be seen in Fig.10(b), it reduces the ratios of generated synthesized data from 13.08%, 65.3%, 30.17%,

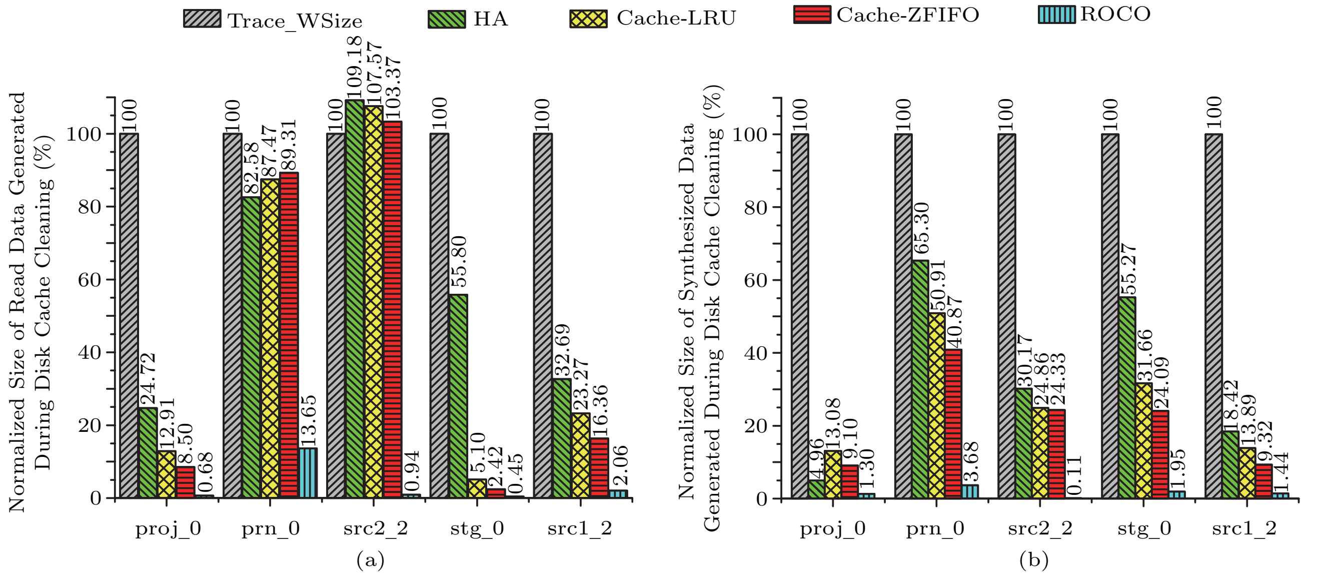

Fig.10. Impact of ROCO on data migration during disk cache cleaning under proj_0, prn_0, src2_2, stg_0 and src1_2. (a) Read data generated during disk cache cleaning. (b) Synthesized data generated during disk cache cleaning.

55.27% and 18.42% to 1.3%, 3.68%, 0.11%, 1.95% and 1.44% under proj_0, prn_0, src2_2, stg_0 and src1_2 respectively.

Fig.11 shows the comparison of average response time of the HA-SMR drive under proj_0, prn_0, src2_2, stg_0 and src1_2. From Fig.11 we can see that ROCO improves the performance of the HA-SMR drive the most among the four solutions. Without an SSD cache, the HA-SMR drive performs 52.89 times worse under proj_0, 5.36 times worse under prn_0, 26.41 times worse under src2_2, 18.42 times worse under stg_0 and 37.91 times worse under src1_2 respectively than that when ROCO is implemented. With an SSD cache and a block-oriented LRU replacement algorithm, when Cache-LRU is implemented, the HA-SMR drive performs 5.61 times worse under proj_0, 4.97 times worse under prn_0, 14.1 times worse under src2_2, 5.46 times worse under stg_0 and 11.29 times worse under src1_2 respectively than that when ROCO is implemented. In addition to writing more non-sequential data than ROCO, Cache-LRU is prone to triggering more frequent disk cache cleaning of the HA-SMR drive because of the block-oriented replacement algorithm. With an SSD cache and a zone-oriented FIFO replacement algorithm, when Cache-ZFIFO is implemented, the HA-SMR drive performs 3.82 times worse under proj_0, 4.67 times worse under prn_0, 12.77 times worse under src2_2, 3.7 times worse under stg_0 and 8.35 times worse under src1_2 respectively than that when ROCO is implemented.

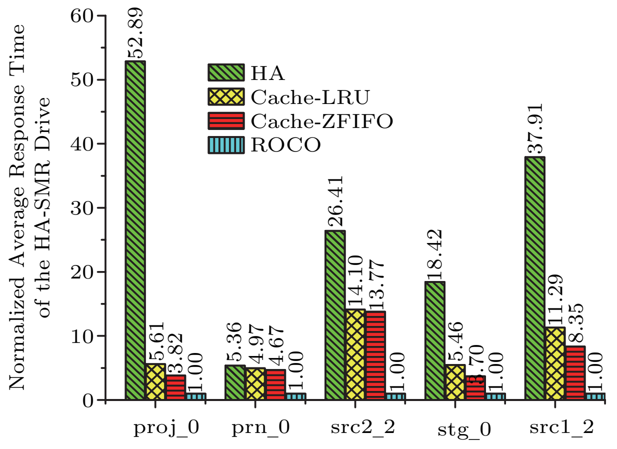

Fig.11. Average response time of the HA-SMR drive.

This is because ROCO implements the zone-oriented block reordering to translate more non-sequential writes to sequential writes, conducts the zone-oriented hot/cold data identification to reduce the amount of data written to the HA-SMR drive and the frequency of disk cache cleaning, and further reduces the non-sequential writes through host-side read-modify-write operations. Among the five traces, we can see that ROCO improves the performance of the HA-SMR drive the most under proj_0. This is because ROCO reduces the largest amount of non-sequential data under proj_0. We can also see that ROCO does not improve the performance of the HA-SMR drive so much under prn_0. This is because prn_0 has the worst spatial locality and the most small random writes.

5 Related Work

The main challenge of SMR technology is that in-place updates cannot be directly supported. Cassuto *et al.* proposed a set-associative STL model which supports in-place updates, and an S-block STL model which supports out-of-place update and intra-band GC[14]. By the set-associative STL, each data update needs a read-modify-write operation which often results in severe performance degradation[5]. By the S-block STL, data is always sequentially written into a band; however, the intra-band GC often brings a large amount of data migration. Lin *et al.* proposed a data layout model called H-SWD which manages data in hot bands and cold bands[15]. Jones *et al.* also classified data as hot data and cold data, and stored them in cold bands and hot bands respectively[16]. To reduce long-term data migration, they also designed a cold-weight algorithm to choose appropriate bands to reclaim. He and Du proposed a hybrid update strategy, which can reduce metadata overhead and write amplification[17]. These above solutions are mainly designed for DM-SMR drives and HM-SMR drives, and cannot be directly used for HA-SMR drives. Wu *et al.* made a thorough investigation of HA-SMR drives by carrying out an in-depth performance evaluation and pointed out the reasons why HA-SMR drives suffered performance degradation when subjected to non-sequential writes[6,7]. They proposed to use zone-oriented reordering to improve the disk cache cleaning efficiency of the HA-SMR drive. Their work is different from ours in that they focused on writing data blocks together belonging to the same zone but not writing data blocks together belonging to different zones, and our work goes a step further to focus on reordering out-of-order data blocks belonging to the same zone to reduce non-sequential writes to the HA-SMR drive.

A lot of work has been done to use SSDs or NVMs to improve the performance of HDDs or SMR drives. Xie and Sun proposed HIT, which was adapted to data-intensive applications by periodically redistributing data between SSDs and HDDs[18]. Chen *et al.* proposed Hystor to achieve performance improvement by using an SSD as a write buffer for an HDD[9]. Mao *et al.* proposed HPDA[19] and Zeng *et al.* proposed HRAID6ML which improves the performance of SSD-based storage systems by combining a group of SSDs and one or two HDDs[20]. Luo *et al.* proposed HWSR to accelerate the performance of SMR drives by using SSD cache and memory buffer[10]. HWSR uses the concept of segment as basic units accessing the SMR drive and alleviates the write amplification to some extent. Wang *et al.* proposed an SMR-oriented cache framework which restricts the LBA range of evicted data from SSD caches to DM-SMR drives[8].

Many cache replacement algorithms have been proposed to improve the performance of the underlying storage devices such as HDDs or SSDs. A widely-used cache replacement algorithm is the LRU algorithm, which takes advantage of recency and evicts the least recently used page with the highest priority. A lot of algorithms are designed based on LRU. Jung *et al.* proposed a buffer replacement algorithm called LRU-WSR[21] which decreases write traffic from the buffer cache to flash storage by reordering writes of non-cold dirty pages. Park *et al.* proposed CFLRU[22] for SSDs which splits the whole cache into a working region and a clean-first region which is one portion of the cache near the end of the LRU position. To take advantage of HDDs' fast sequential access speed and the non-volatile property of NVRAM, Fan *et al.* proposed a buffer cache policy called I/O-Cache[23] to decrease storage writes by regrouping and synchronizing long sets of consecutive dirty pages. Fan *et al.* also proposed a cooperative hybrid NVRAM and DRAM cache policy called Hibachi[24] for storage arrays, which maximizes cache hit rates by treating read cache hits and write cache hits differently and captures workloads' tendencies by adjusting the clean and the dirty cache sizes. Park *et al.* proposed a lookahead read cache[25] designed for a backup application, which exploits future read access patterns during dedupe processes and employs a small log buffer to keep small portions of future reference data chunks.

However, because of the special characteristics of HA-SMR drives, all these solutions cannot be straightforwardly applied to improve the performance of HA-SMR drives.

6 Conclusions

HA-SMR drives are expected to be more popular than DM-SMR and HM-SMR drives in storage systems; however, they often suffer performance degradation under write-intensive workloads. The fundamental approach is to reduce non-sequential writes buffered in the disk cache of the HA-SMR drive. Update writes, small random writes and out-of-order writes compose the majority of non-sequential writes. In this paper, we proposed a hybrid storage solution called ROCO, which reorders the out-of-order writes belonging to the same zone before they arrive in the HA-SMR drive and uses an SSD cache to absorb update writes and small random writes. We also designed a data replacement algorithm called CREA for the SSD cache. It implements zone-oriented hot/cold data identification, and gives the highest priority to sequential writes belonging to colder zones. If there are no sequential writes, it selects data blocks belonging to the colder zones that can be evicted through host-side RMW operations as victims. CREA gives the lowest priority to non-sequential writes belonging to the hottest zone. Experimental results showed that ROCO can effectively reduce non-sequential writes and improve the performance of the HA-SMR drive.

Our current work mainly focuses on improving the performance of HA-SMR drives by reducing non-sequential writes from the host side. However, the data cleaning and garbage collection policies working in the disk cache also influence the performance of HA-SMR drives. In the future, we plan to design more sophisticated data cleaning and garbage collection policies for the disk cache to improve the performance of HA-SMR drives.

References

[1] Wood R, Williams M, Kavcic A, Miles J. The feasibility of magnetic recording at 10 terabits per square inch on conventional media. *IEEE Trans. Magnetics*, 2009, 45(2): 917-923.

[2] Venkataraman K S, Dong G, Zhang T. Techniques mitigating update-induced latency overhead in shingled magnetic recording. *IEEE Trans. Magnetics*, 2012, 48(5): 1899-1905.

[3] Feldman T, Gibson G. Shingled magnetic recording areal density increase requires new data management. *Login: the USENIX Magazine*, 2013, 38(3): 22-30.

[4] Zeng L F, Zhang Z H, Wang Y, Feng D, Kent K B. CosaFS: A cooperative shingle-aware file system. *ACM Trans. Storage*, 2017, 13(4): Article No. 34.

[5] Aghayev A, Shafaei M, Desnoyers P. Skylight — A window on shingled disk operation. *ACM Trans. Storage*, 2015, 11(4): Article No. 16.

[6] Wu F G, Yang M C, Fan Z Q, Zhang B Q, Ge X Z, Du D H. Evaluating host aware SMR drives. In *Proc. the 8th USENIX Workshop on Hot Topics in Storage and File Systems*, June 2016, pp.31-35.

[7] Wu F G, Fan Z Q, Yang M C, Zhang B Q, Ge X Z, Du D H. Performance evaluation of host aware shingled magnetic recording (HA-SMR) drives. *IEEE Trans. Computers*, 2017, 66(11): 1932-1945.

[8] Wang C L, Wang D D, Chai Y P, Wang C W, Sun D S. Larger, cheaper, but faster: SSD-SMR hybrid storage boosted by a new SMR-oriented cache framework. In *Proc. the 33rd Int. Conf. Massive Storage Systems and Technology*, May 2017, pp.1-16.

[9] Chen F, Koufaty D A, Zhang X. Hystor: Making the best use of solid state drives in high performance storage systems. In *Proc. the 25th Int. Conf. Supercomputing*, May 2011, pp.22-32.

[10] Luo D, Wan J G, Zhu Y F, Zhao N N, Li F, Xie C S. Design and implementation of a hybrid shingled write disk system. *IEEE Trans. Parallel and Distributed Systems*, 2015, 27(4): 1017-1029.

[11] Kim H, Shin D, Jeong Y *et al.* SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device. In *Proc. the 15th USENIX Conf. File and Storage Technologies*, February 2017, pp.271-284.

[12] Bucy J S, Schindler J, Schlosser S W, Greger G R. The DiskSim simulation environment version 4.0 reference manual. Technical Report, Carnegie Mellon University, 2008. http://www.pdl.cmu.edu/DiskSim, May 2018.

[13] Narayanan D, Donnelly A, Rowstron A. Write off-loading: Practical power management for enterprise storage. *ACM Trans. Storage*, 2008, 4(3): Article No. 10.

[14] Cassuto Y, Sanvido M A A, Guyot C, Hall D R, Bandic Z Z. Indirection systems for shingled-recording disk drives. In *Proc. the 26th IEEE Symp. Mass Storage Systems and Technologies*, May 2010.

[15] Lin C I, Park D, He W P, Du D H. H-SWD: Incorporating hot data identification into shingled write disks. In *Proc. the 20th IEEE Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, August 2012, pp.321-330.

[16] Jones S N, Amer A, Miller E L, Long D D E, Pitchumani R, Strong C R. Classifying data to reduce long-term data movement in shingled write disks. *ACM Trans. Storage*, 2016, 12(1): Article No. 2.

[17] He W, Du D. SMaRT: An approach to shingled magnetic recording translation. In *Proc. the 15th USENIX Conf. File and Storage Technologies*, February 2017, pp.121-133.

[18] Xie T, Sun Y. Dynamic data reallocation in hybrid disk arrays. *IEEE Trans. Parallel and Distributed Systems*, 2010, 21(9): 1330-1341.

[19] Mao B, Jiang H, Wu S Z, Tian L, Feng D, Chen J X, Zeng L F. HPDA: A hybrid parity-based disk array for enhanced performance and reliability. *ACM Trans. Storage*, 2012, 8(1): Article No. 4.

[20] Zeng L F, Feng D, Chen J X, Wei Q S, Veeravalli B, Liu W G. HRAID6ML: A hybrid RAID6 storage architecture with mirrored logging. In *Proc. the 28th IEEE Symp. Mass Storage Systems and Technologies*. April 2012.

[21] Jung H, Shim H, Park S, Kang S, Cha J. LRU-WSR: Integration of LRU and writes sequence reordering for flash memory. *IEEE Trans. Consumer Electronics*, 2008, 54(3): 1215-1223.

[22] Park S, Jung D, Kang J, Kim J, Lee J. CFLRU: A replacement algorithm for flash memory. In *Proc. the 2006 Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems*, October 2006, pp.234-241.

[23] Fan Z Q, Haghdoost A, Du D H, Voigt D. I/O-Cache: A non-volatile memory based buffer cache policy to improve storage performance. In *Proc. the 23rd IEEE Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, October 2015, pp.102-111.

[24] Fan Z, Wu F, Park D, Diehl J, Voigt D, Du D H. Hibachi: A cooperative hybrid cache with NVRAM and DRAM for storage arrays. In *Proc. the 33rd IEEE Symp. Mass Storage Systems and Technologies*, May 2017, pp.90-101.

[25] Park D, Fan Z Q, Nam Y J, Du D H. A lookahead read cache: Improving read performance for deduplication backup storage. *Journal of Computer Science and Technology*, 2017, 32(1): 26-40.

Wen-Guo Liu received his B.E. degree in information and computing science from Jinan University, Jinan, in 2008, and M.E. degree in computer science and technology from Hunan University, Changsha, in 2011. He is currently working toward his Ph.D. degree in the School of Computer Science and Technology from Huazhong University of Science and Technology (HUST), Wuhan. His research interest includes solid state drives, shingle magnetic recording, and hybrid storage systems.

Ling-Fang Zeng received his B.S. degree in computer application from Huazhong University of Science and Technology (HUST), Wuhan, in 2000, M.S. degree in computer application from China University of Geosciences, Wuhan, in 2003, and Ph.D. degree in computer architecture, from HUST, Wuhan, in 2006. He was a research fellow in Department of Electrical and Computer Engineering, National University of Singapore, Singapore, during 2007–2008 and 2010–2013. He is currently with Wuhan National Laboratory for Optoelectronics, and School of Computer Science and Technology, HUST, as an associate professor. He publishes more than 50 papers in major journals and conferences, including ACM Transactions on Storage, IEEE Transactions on Magnetics, Journal of Parallel and Distributed Computing, Journal of Network and Computer Applications, Software: Practice and Experience, FAST, SC, MSST, IPDPS, CLUSTER, and CCGrid, and serves for multiple international journals and conferences. He is a member of CCF, ACM, and IEEE.

Dan Feng received her B.E., M.E., and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, in 1991, 1994, and 1997, respectively. She is a professor and the dean of the School of Computer Science and Technology, HUST, Wuhan. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has over 100 publications in journals and international conferences, including FAST, ICDCS, HPDC, SC, ICS, and ICPP. Dr. Feng is a member of CCF, ACM and IEEE.

Kenneth B. Kent received his B.Sc. degree in computer science from the Memorial University of Newfoundland, St. John's, in 1996, and his M.Sc. and Ph.D. degrees in computer science from the University of Victoria, Victoria, in 1999 and 2003, respectively. He is a professor in the Faculty of Computer Science at the University of New Brunswick, Fredericton, and the director of IBM Centre for Advanced Studies-Atlantic, Canada. His research interests include hardware/software co-design, virtual machines, reconfigurable computing, and embedded systems. His research groups are key contributors to widely used software such as the J9 Java virtual machine and the VTR (Verilog-To-Routing) FPGA CAD flow.