SAC: A Co-Design Cache Algorithm for Emerging SMR-based High-Density Disks

Diansen Sun
Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
School of Information, Renmin University of China
Beijing, China
dssun@ruc.edu.cn

Yunpeng Chai
Key Laboratory of Data Engineering and Knowledge Engineering, MOE, China
School of Information, Renmin University of China
Beijing, China
ypchai@ruc.edu.cn

Abstract

To satisfy the huge storage capacity requirements of big data, the emerging high-density disks gradually adopt the Shingled Magnetic Recording (SMR) technique. However, the most serious challenge of SMR disks lies in their weak fine-grained random write performance caused by the write amplification inner SMRs and its extremely unbalanced read and write latencies. Although fast storage devices like Flash-based SSDs can be used to boost SMR disks in SMR-based hybrid storage, the optimization targets of existing cache algorithms (e.g., higher popularity for LRU, lower SMR write amplification ratio for MOST) are *NOT* the crucial factor for the performance of the SMR-based hybrid storage. In this paper, we propose a new SMR-Aware Co-design cache algorithm called SAC to accelerate the SMR-based hybrid storage. SAC adopts a hardware/software co-design method to fit the characteristics of SMR disks and to optimize the crucial factor, i.e., RMW operations inner SMR disks, effectively. Furthermore, SAC also makes a good balance between some conflicting factors, e.g., the data popularity vs. the SMR write amplification and clean cache space vs. dirty cache space. In our evaluations under real-world traces, SAC achieves a 7.5× performance speedup compared with LRU in the write-only mode, and a 2.9× speedup in the read-write mixed mode.

CCS Concepts • **Information systems → Hierarchical storage management**.

Keywords cache algorithm; SMR; hybrid storage; write amplification

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378474

ACM Reference Format:
Diansen Sun and Yunpeng Chai. 2020. SAC: A Co-Design Cache Algorithm for Emerging SMR-based High-Density Disks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3373376.3378474

1 Introduction

The global data volume is estimated to reach 175 ZB in 2025 according to IDC [15]. The rapidly accumulating amount of data places high demands on the capacity scalability and cost-effectiveness of storage devices, so magnetic recording devices remain strong competitors for maintaining big data. However, conventional magnetic recording (CMR) disks have reached the upper bound of their storage density. The next-generation high-density disks have to adopt some emerging high-density technologies like Shingled Magnetic Recording (SMR) [3], Bit Patterned Magnetic Recording (BPMR) [21], Heat Assisted Magnetic Recording (HAMR) [22], and Microwave-Assisted Magnetic Recording (MAMR) [28], promoting the magnetic recording density to 10T bit/in^2 and more [19].

Among these new techniques, SMR, which squeezes more tracks of platter like the overlapped shingles on a roof to promote the storage density, is the most mature, and all the other techniques are also likely to be applied coupled with SMR. Practical SMR drive products have been released [17, 18]; they have lower prices and larger capacities compared with CMR disks. For example, Pelican [2] packs 1,152 SMR disks in one 52U rack, reaching a total capacity up to 5 PB. Dropbox has deployed hundreds of petabytes of SMR disks in their online storage service [4].

However, as a side effect of high storage density, SMR drives have an inherent problem of *write amplification* when processing random writes, which makes SMR drives exhibit extremely unstable and low performance for random writes. That is why SMR drives are often used as archive disks in many corporations. For example, shown as Fig. 1, according to our measurements of 4KB random writes in a logical block address (LBA) range of 1TB, the bandwidth of a Seagate 8TB SMR HDD severely jittered between 38KB/s and 2000KB/s,

RIGHTSLINK

while the bandwidth of a 5TB CMR in a comparative test swings slightly around 830KB/s.

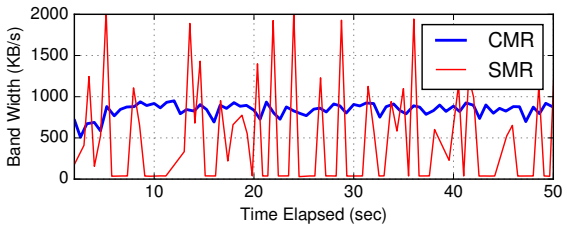


Figure 1. Real-time bandwidth of SMR vs. CMR disks when performing random writes.

Although deploying some fast storage devices (e.g., Flash-based SSDs [6, 13, 16]) as the cache layer upon disks is usually an effective solution to boost the hybrid storage's performance, existing cache algorithms cannot achieve good performance for the SMR-based hybrid storage due to the serious write amplification challenge of SMR disks. For example, traditional cache algorithms like LRU, LIRS [7], ARC [12] aim to optimize the cache hit ratios, but never consider the SMR characteristics at all, usually leading to very large write amplification rates and low performance. On the contrary, the existing algorithms that only aims to reduce SMR write amplification, e.g., MOST [11], have too high cache miss and excessive SMR I/Os due to ignoring the data popularity.

In this paper, therefore, we propose an SMR-aware Co-design (SAC) cache algorithm specially for the SMR-based hybrid storage. SAC achieves 2.9 ∼ 7.5 times higher overall performance compared with the classical LRU algorithm according to the evaluations based on real-world enterprise traces, and SAC also outperfoms MOST for 1.4 ∼ 1.5 times. That is because SAC has three important advantages to promote the performance of the hybrid storage compared with existing cache algorithms:

1) SAC adopts a hardware/software co-design method to fit the characteristics of SMR disks and reduces the SMR write amplification effectively. Furthermore, a universal method of detecting the important hidden features of drive-managed (DM) SMR disks are also given (see Section 5).

2) Different with the algorithms that only focus on optimizing cache hit rates (e.g., LRU) or SMR write amplification rates (e.g., MOST), SAC is designed to lower the number of the heaviest inner operations of SMR, i.e., RMWs, which is the crucial factor of the SMR-based hybrid storage' performance.

3) Aiming at the significant and unstable performance gap between SMR disk read and write, SAC introduces a functional module that dynamically balances cache resources occupied by clean and dirty data blocks. More importantly, the function module can be independent and can be used in combination with other cache algorithms (such as MOST) to improve their performance.

The rest of this paper is organized as following. In Section 2, we introduce the background knowledge of SMR disks and the new challenges brought by them for the cache algorithms in SMR-based hybrid storage. And then we analyze and give three principles for designing efficient SMR-oriented cache algorithms in Section 3. Section 4 presents the detailed design of our proposed SMR-aware co-design (SAC) cache algorithm and the methods of detecting some important SMR hardware specifications are explained in Section 5. Subsequently, we evaluate SAC and some other representative algorithms in Section 6. The related work and the conclusion can be found in Section 7 and Section 8, respectively.

2 Background and Challenges

2.1 Write Amplification Challenge of SMR Disks

In order to increase the storage density, SMR drives reduce the width of disk tracks by squeezing the tracks into a pattern that overlaps each other (similar to roof shingles [3]). However, the write heads of disks are wider than the squeezed track and the read heads, so writing data into an SMR drive will erase the data on adjacent tracks [1]. Therefore, as shown in Fig. 2, to deal with the data interference and avoid updating nearly the whole disk space, modern SMR disks usually contain two separate areas on the disk, i.e., the *band region* and the *persistent buffer (PB) region.*

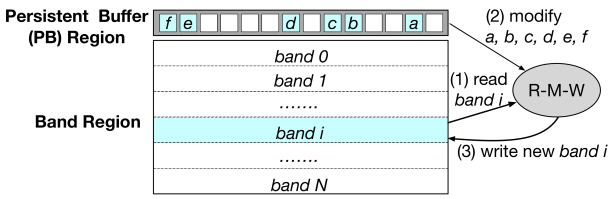


Figure 2. The RMW process in SMR drives.

In the band region, some adjacent squeezed tracks belong to one *band*, and each band is separated from others through some guard space. It means that the data in a band will not be interfered by other band's write operations. However, because the tracks in a band are squeezed, when we want to modify some data within a band, even just one sector, the entire band has to be rewritten, leading to serious *write amplification* (WA) phenomenon and very low performance of writing. This feature is very unfriendly for the fine-grained random writing.

The persistent buffer (PB) region is usually located at the outer tracks, and it temporarily stores the copies of recently written data in the form of circle-logging. When the persistent buffer is almost full, a garbage collection process forces a portion of the buffered data to be written back to the band region to free up some space in PB. The garbage collection process is formed by some ***Read-Modify-Write (RMW)*** operations. Each RMW operation first loads the data of one

RIGHTSLINK

entire band into memory, and then modifies the related parts according to the data blocks loaded from PB which locates in this band. Finally, the new version of this band in memory will be written back into the SMR band region, shown as Fig. 2.

The most performance difference between SMR and conventional magnetic recording (CMR) disks lies in the heavy RMW operations, which are usually the most time-consuming operations in SMR. Once the persistent buffer cannot accumulate enough data blocks located in the same band when performing writing back, the write amplification rate for the triggered RMW operation is large, leading to very limited write performance.

2.2 Challenges of Existing Cache Algorithms

Except for some sequential writing-dominated scenes like logging, in many other applications, we cannot directly employ SMR disks due to its extremely poor performance of random writing. Therefore, a common-sense method is to employ a faster storage device (e.g., Flash-based SSDs) as the cache layer of SMR disks, to remedy the shortcomings of SMR. Existing cache algorithms in a hybrid storage system can be divided into two categories: the traditional popularity-driven cache algorithms (e.g., LRU, etc.) and the WA-driven algorithms specially designed for SMR (e.g., MOST [11]).

Traditional Cache Algorithms usually write back blocks from the SSD cache layer to SMR disks according to the popularity which can be quantified by blocks' recency (e.g., LRU), frequency (e.g., LFU), access intervals (e.g., LIRS [7]), or their mixture (e.g., LRFU [9], ARC [12], and MQ [27]). For the SSD-SMR hybrid storage, however, these classical cache algorithms always lead to poor performance, because these evicted unpopular data written to PB may belong to too many bands, resulting in severe write amplification.

In order to exhibit this intuitively, we performed the Microsoft Research Cambridge workloads [14] on an accurate SMR emulator developed by us (see Section 6.1 for more). Fig. 3 gives the SMR write amplification rate results of the experiments when performing the workloads on an SMR only, and an SSD-SMR hybrid storage coupled with LRU and MOST, respectively. The results indicate that the SMR-only solution leads to write amplification rates between 23 ∼ 55 under these practical enterprise I/O workloads. And setting an SSD cache layer coupled with LRU cannot reduce the SMR write amplification rates; on the contrary, LRU causes larger write amplification rates under seven of the ten traces. For example, in *rsrch*, the write amplification rate of SMR-only is 49 whereas LRU aggravates it to 107.

SMR-oriented Cache Algorithms. *MOST* [24] is another extreme cache algorithm specifically designed for SMR. Not considering the data popularity at all, as Fig. 4 depicts, MOST always select the band that contains the most dirty blocks in the cache layer, and then evict all the cached blocks located in this band all at once.

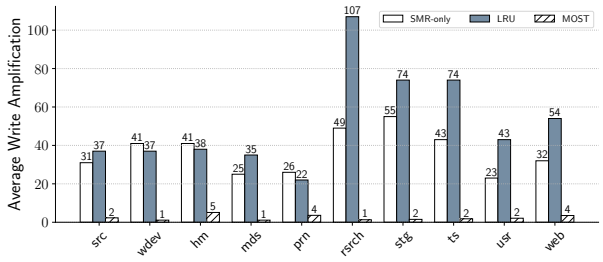


Figure 3. Average write amplification ratios of SMR-only and different cache algorithms under Microsoft traces.

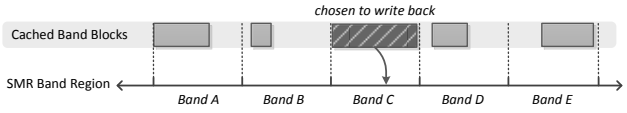


Figure 4. Principle of the *MOST* scheme.

As the previous Fig. 3 shows, *MOST* can effectively decline the write amplification rates to 1 ∼ 5, much lower than SMR-only and LRU. However, the cache miss rate results in Fig. 5 also indicate that *MOST* leads to much more cache misses than LRU, because data popularity is never considered in *MOST*. More cache misses will also lead to more SMR writes and more RMW operations.

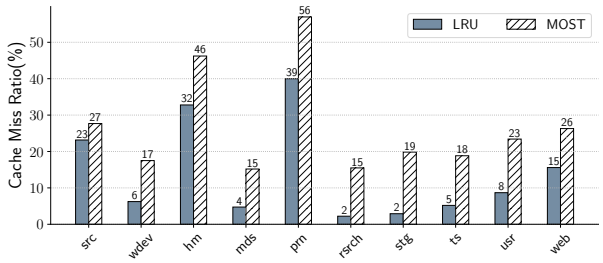


Figure 5. The *MOST* scheme causes more cache misses.

In fact, *MOST* weakens the main function of the cache layer, i.e., reducing I/O accesses by maintaining hot data. It is not advisable to abandon data popularity totally and only consider the write amplification factor. An ideal cache algorithm for SMR should make a good balance between the two factors of the cache hit rate and the write amplification.

3 Principles for Designing Efficient SMR-oriented Cache Algorithms

SMR drives adopt new internal designs to achieve high storage density, but bring the serious write amplification problem. Existing cache algorithms (e.g., LRU, MOST, etc.) cannot make the cache layer support the underlining SMR disks well. They do not make a good balance between the data

RIGHTSLINK

popularity and the SMR write amplification, and do not consider the distinctive hardware design of SMR disks. In order to design appropriate cache algorithms for SMR disks, in this section, we analyze the principles of designing a specific cache algorithm optimized for SMR disks.

Principle #1. Hardware/Software Co-Design to Reduce SMR Write Amplification

The internal hardware design (e.g., the shingled tracks, bands, PB, and RMW operations) of SMR disks is much more complicated compared with CMR disks. Therefore, if we do not consider the SMR hardware specification and put the write requests directly into SMR disks, the write amplification ratios may be up to tens of or even more than one hundred times (recall the results in Fig. 3).

In consequence, an ideal cache algorithm for SMR drives should exactly know the important internal hardware specifications, such as the band size and the persistent buffer capacity. (1) If we are aware of the sizes of all the bands, we can know the located SMR band of each cached block. Similar to MOST, thus, we can put the blocks of the same band together to reduce the additional I/Os when performing RMW operations. (2) The PB capacity is also very important. On one side, PB can accumulate as many cached blocks located in the same band as possible to reduce the write amplification; on the other side, if we cannot utilize PB well, some written data may become the **fragment** to trigger large SMR write amplification rates.

As Fig. 6 (a) plots, in fact, the existence of PB relaxes the time of writing the blocks from the same band into an SMR disk to get a small write amplification rate, unlike MOST, which has to write these blocks all at once into an SMR disk. Therefore, among these blocks, some hot ones can be kept in the upper cache layer for a longer time to gain more cache hits.

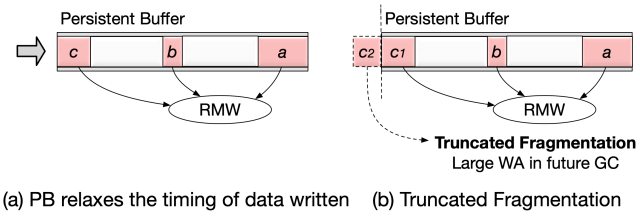


Figure 6. Benefits of PB and Truncated Fragmentation.

However, if we do not know the capacity of PB and make the blocks from the same band too far away from each other, some of them cannot be involved in one round of PB garbage collection (i.e., the RMW operation). Taking the case shown in Fig. 6 (b) as an example, data blocks a, b, c_1, and c_2 locate on the same band of an SMR disk. When c_1 has just been written into PB, PB is full and an RMW operation is triggered to release the space of PB. In this case, the blocks a, b, and c_1 are involved into the RMW operation, but c_2 has not entered PB, which means c_2 will trigger another RMW operation and may lead to large write amplification rate in the future. This phenomenon is called ***Truncated Fragmentation***. The algorithms without the knowledge of PB, including MOST, may trigger great write amplification sometimes due to the truncated fragmentation.

Principle #2. RMW-oriented Tradeoff between Popularity and Write Amplification

Only considering popularity (e.g., LRU) or write amplification (e.g., MOST) cannot lead to satisfactory performance for the SMR-based hybrid storage, because none of them is a decisive factor in the performance of the hybrid storage.

We reasonably assume that the *RMW triggering number* is the most critical measure of the performance as it is always the most time-consuming behavior in SMR regardless of how big the write amplification is. Therefore, if reducing the number of RMW triggers, the total I/O time could be shortened.

For verifying this, we conducted a set of emulation experiments where we run 125 million requests of each Microsoft's workload to detect which factor including miss rates, WA rates, RMW triggering number is more relative to the performance of an SSD-SMR hybrid storage system. In the results analysis, we utilize Pearson Correlation Coefficients (*PCC*) [23] to measure the relativity between the factor and I/O time; higher *PCC* means it is more relative. As Table 1 shows, the miss ratio and the write amplification ratio both have a *PCC* lower than 0.5. However, the number of RMWs triggered in our emulated SMR disk is strongly related to the system performance, i.e., $PCC = 0.82$. This is a good news that the RMW count is a quantitative metric to make a good balance between popularity and write amplification for the SMR-based hybrid storage. An ideal cache algorithm for SMR should reduce the RMW count as many as possible.

Table 1. Correlation with SMR I/O time

Trace	Time(sec)	Miss ratio(%)	WA	RMWs
src	673	86.1	4.4	1784
wdev	470	43.6	13.2	2403
hm	960	57.6	9.9	5792
mds	354	36.1	9.9	1598
prn	928	66.1	4.4	4377
rsrch	894	47.7	55.5	6677
stg	700	51.7	32.4	4442
ts	684	45.1	27.4	4520
usr	488	36.6	13.1	2778
web	710	37.2	16.3	5022
	PCC	***0.47***	***0.24***	***0.82***

(PPC: Pearson correlation coefficient[23])

Principle #3. Dynamically Balancing Clean and Dirty Cache Segments.

RIGHTSLINK

Traditional cache algorithms usually manage clean and dirty blocks according to the same popularity indicator (e.g., LRU put them in one queue), because the underlying disk has similar performance for the read and the write requests. However, this mechanism does not work for SMR, because the writing overhead in SMR disks is much different compared with reading. On one side, random writes on SMR disks usually cause background RMW operations and significant write amplification, and the write amplification rates are extremely unstable. On the other side, random writes are turned into sequential writes into the persistent buffer, while the random reads still cause the random movements of the disk heads. Therefore, it is hard to tell the practical performance difference between SMR reading and writing operations.

As Table 2 plots, the relative standard deviation (RSD) of write bandwidth of SMR disks is up to more than 800% in the random I/O pattern, while that of CMR drives is small (i.e., 11.1%).

Table 2. SMR's read and write bandwidth (KB/s).

	Random Read			Random Write		
	Avg.	Stdev.	RSD	Avg.	Stdev.	RSD
CMR	247.5	15.2	*6.1%*	588.8	65.3	*11.1%*
SMR	373.7	24.8	*6.6%*	124.2	1080.1	*869.6%*

Workload: uniformly random I/O, 4KB block size, LBA range from 0 to 10GB.

Therefore, we need to dynamically balance the cache capacity occupied by the clean and the dirty blocks, considering both the popularity and the real-time performance difference between SMR writing and reading.

Summary. Among the three principles for designing SMR-oriented cache algorithms, the first and the second ones aim to optimize the dirty cache part management, and the last one is proposed for adjust the cache resource allocation between the clean and the dirty cache appropriately.

4 Algorithm Design of SAC

Motivated by the above three principles, we propose a new SMR-aware co-design cache algorithm called SAC. Section 4.1 gives an overview of our proposed SAC algorithm, and the following three parts present the three key components of SAC, respectively (see Section 4.2, 4.3, and 4.4).

4.1 Overview

The architecture of our proposed SAC algorithm is illustrated as Fig. 7. First, in SAC, the cached clean and dirty data are managed independently according to different schemes. The cached clean blocks are sorted in an LRU queue, but the dirty data management is much more complicated, with some additional rules based on a basic LRU queue. Second, SAC contains three key components, i.e., Cycle-Driven Writeback (CDW), Target Bands Selection (TBS), and Clean/Dirty Comparator (CDC). CDW and TBS are used to manage the dirty cache, while CDC is employed to make a good balance between the clean and the dirty cache.

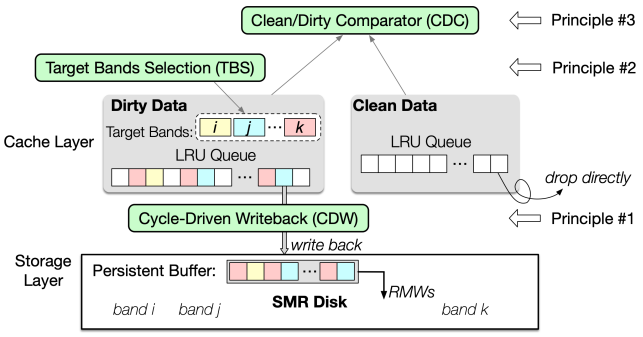


Figure 7. Architecture of SAC.

- Motivated by the above *Principle #1*, CDW fully considers SMR drives' inner structure of PB and bands and reduces the SMR write amplification rates effectively through a cycle-driven writeback manner.
- Cooperated with CDW, the TBS module aims to limit the evicted victims' located bands to a small range, and selects the most appropriate blocks within these bands for eviction, in order to reduce the triggered RMW counts motivated by *Principle #2*.
- According to *Principle #3*, we should dynamically balance the clean and the data cache segments based on real-time SMR write performance and data popularity. The CDC module compares the eviction cost of these two cache parts periodically and determines to evict clean or dirty blocks in a period, resulting to balance the two cache regions.

4.2 Cycle-Driven Writeback

Unshaped SMR written data stream usually makes PB contain data blocks from too many bands and too large write amplification when performing RMWs, or triggers the *truncated fragmentation* problem (see Section 3). We therefore propose a new *Cycle-Driven Writeback (CDW)* approach which can shape the SMR written data's I/O pattern to avoid above problems.

The cycle here means a series of conditional operations on the cache eviction. In one cycle, only a limited number of cache blocks belonging to certain 'qualified' bands are allowed to write back to SMR. The total size of cache blocks that the cycle is allowed to evict is called ***cycle length***, and is typically a fixed size for the workload. The 'qualified' bands referred to as ***Target Bands*** are re-selected at the beginning of each cycle, and the issue about the target bands selection will be handled by another module of SAC, i.e., *Target Bands Selection* (see Section 4.3).

RIGHTSLINK

As Fig. 8 plots, the SMR written data stream enters and gets out of PB. If *cycle B* is the current cycle and the previous cycle is *cycle A*, PB may contain the data written during *cycle A* and *cycle B*. In this case, when the PB garbage collection is triggered, only data blocks located in *Target Bands A* and *Target Bands B* may be involved in the RMW operations. That is to say, limited band number leads to less RMWs and smaller write amplification rates.

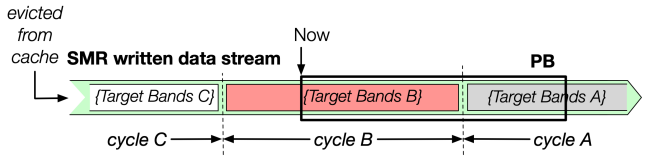


Figure 8. Schematic diagram of *Cycle-Driven Writeback*.

The detailed method of CDW includes the following rules:

Rule 1. In each cycle, only a limited number of target bands are selected for SMR writeback, and the total size of these bands must NOT exceed the PB capacity. The method of how to get the accurate PB capacity for device-managed SMR drives will be discussed later in Section 5.1.

Rule 2. The cycle length should not beyond the PB capacity; otherwise, PB cannot hold all the blocks of the too long cycle, leading to *truncated fragmentation*. The impacts of the cycle length are evaluated in this paper and can be found in Section 6.5.

Rule 3. A cycle should be ended when there are no blocks of the selected target bands in the upper cache layer that can be written back to SMR. Then, start a new cycle.

Rule 4. All active cycles (i.e. not be GC in the PB) should not overlap in the selection of the target bands. Otherwise, the truncated fragmentation will occur across the overlapped cycles.

Rules 1 to 3 prevent truncated fragmentation from occurring in a single cycle, and also the fragmentation will not occur across the cycles according to *Rule* 4.

4.3 Target Bands Selection

The target bands selection (TBS) module is designed to make a good balance between the popularity and the SMR write amplification through restricting the cache victim candidates to the cache blocks located on some appropriate SMR bands. Recall *Principle #2* in the above section and Table 1 that the balance point lies in minimizing the RMW operation counts inner SMR disks. Therefore, what kind of bands should be selected to reduce the RMW triggering time the most?

Illustrated as Fig. 9, if PB can release the as much space as possible when performing an RMW operation, which flushes all the blocks of one band (e.g., B_i) from PB to the SMR band region, less RMW operations are required. Note that some of the blocks in B_i may be accessed and appear in PB again in the near future. So we should pursue the band that can **actually** release the most space, i.e., few blocks of the band should re-enter PB to occupy its space. Furthermore, because the data of PB all comes from the eviction of the upper cache layer, the target bands of the cache layer, which are allowed to evict data to PB, also should be the ones that can **actually** release the most PB space.

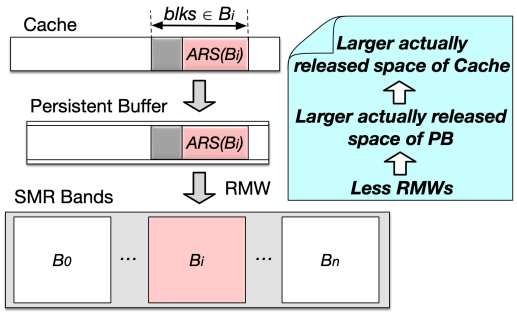


Figure 9. TBS chooses the bands with the largest actually released cache space to minimize RMW counts.

The *Actually Released cache Space* (ARS) of one band B_i can be calculated according to the following Eq. 1, where $S_i = \{x | x \in B_i \text{ and } x \in cache\}$ is the collection of blocks which belong to band B_i and are currently cached in the dirty cache, and the $P(x)$ is the probability of the block x being accessed again after we write back it to SMR. Note that *ARS* of B_i indicates the count of cached blocks, which has the same size.

$$ARS(B_i) = |S_i| - \sum_{x \in S_i} P(x) \quad (1)$$

According to the massive classical cache algorithms (e.g., LRU, LFU, LIRS, etc.), there are many ways to estimate $P(x)$, such as through the recency, the frequency, or the access intervals. In this paper, we adopt a simple and effective one similar to LRU. In SAC, the cached blocks, whose last access time is older than a threshold, are considered as cold blocks that will not be accessed again in the near future. And the other cache blocks are hot ones, and will enter the cache again.

The experimental results (§6.2) indicate that, in cooperation with CDW, the ARS-based target band selection method reduces the RMW triggering number by more than 50% compared to MOST.

4.4 Clean/Dirty Comparator

CDC aims to adjust the occupied cache resources (i.e., the cache space) of the clean and the dirty blocks dynamically by considering both the popularity and the real-time performance gap between SMR reading and writing. The method

RIGHTSLINK

that CDC adopts is to determine evicting a certain number of dirty blocks or clean blocks, by comparing the cost of evicting these clean or dirty blocks periodically. Here we compare two groups of blocks, but not only two blocks, i.e., the worst clean block and the worst dirty one. The reason lies in that the popularity and the SMR performance measurement for a group of blocks is much more accurate than measuring a single block.

Here, the cost of block eviction is defined as the consumed SMR working time for actually releasing the unit of cache capacity. Between the clean and the dirty block groups, we should choose the group with smaller cost as the cache victim, because this means less performance loss to release the same cache capacity. First, the actually released cache capacity, in fact, puts the data popularity into the consideration, because it is equal to the evicted blocks that will not be accessed in a long time. For example, after we have evicted 10 blocks, four of them entered the cache again in a future time window, so the actually released cache space is 6 blocks. Second, we choose the consumed SMR working time as the performance loss indicator because SMR is usually the performance bottleneck of the hybrid storage and it cannot work in parallel due to the disk head limit.

Specifically, the set of the worst clean blocks is denoted as S_{clean} and that of the worst dirty blocks is S_{dirty}. The cost model of each set is generally formulated as the Eq. 2, where S_t is S_{clean} or S_{dirty}, $Time(S_t)$ is the consumed SMR time for processing the evicted blocks, and $ARS(S_t)$ is actually released cache space of S_t (see Section 4.3 and Eq. 1).

$$Cost(S_t) = \frac{Time(S_t)}{ARS(S_t)} \tag{2}$$

Estimation of the consumed SMR time cost. For evicting clean blocks, we just drop them directly, but some of the victims may be accessed again. So the main time cost of evicting clean blocks is the SMR time spent on loading these *hot* blocks back into the cache. According to the definition of $ARS()$, $ARS(S_{clean})$ is the capacity that will not be accessed. The average time spent on each random SMR read can be taken for granted with a pre-measured value of L_{read} because the SMR read performance is stable (recall Table 2). In this case, the clean blocks' eviction overhead $Time(S_{clean})$ can be expressed as Eq. 3.

$$Time(S_{clean}) = L_{read} \times (|S_{clean}| - ARS(S_{clean})) \tag{3}$$

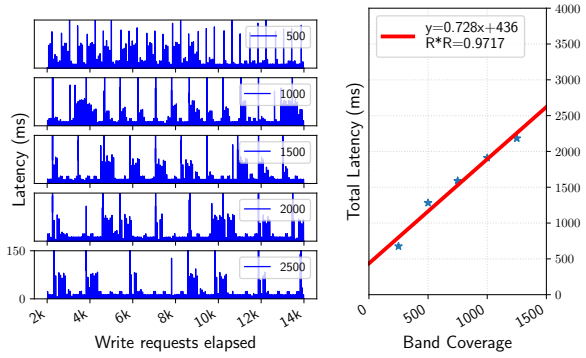
The time cost of evicting a dirty block set S_{dirty} is more intuitive compared with the clean block eviction; it is the consumed time of writing S_{dirty} into the SMR disk. However, the time overhead is not easy to estimate because of the extremely unstable write performance of SMR disks (recall Table 2). Therefore, we use a regression function $R()$, which will be explained in detail in the following part, fitted by multiple sets of experimental results as the estimation of the SMR write overhead.

$$Time(S_{dirty}) = R(S_{dirty}) \tag{4}$$

Estimation of the band eviction time cost. The write latency of SMR is unstable due to the fluctuated write amplification rate in RMW operations. But the total write cost under the same write amplification rate should be stable. If we refer to the number of blocks participating in the RMW operation for a band as the *Coverage* of the band, the total time cost of writing blocks with the same band coverage will be similar to each other.

The measurement results in Fig. 10 (a) are the latencies for all the served requests when the band coverage is 500, 1000, 1500, 2000, 2500, respectively. We can focus on the length of the interval between the "latency" peak cluster and the average continuous time of all the clusters. Each peak cluster is probably made by the GC operation of SMR. The more intensive the latency peak cluster happens, the lower the GC efficiency is. For the 500 coverage, cache evictions are more frequent to trigger the SMR's GC than the 2500 coverage case. Every peak cluster of the 500 coverage case is shorter than that of the 2500 coverage case, because the more coverage in the SMR persistent buffer takes the more time to GC. Figure 10 (b) gives the average write request time cost in each coverage, in which the X axis is the band coverage and the Y axis is the average I/O latency.

Observed from Figure 10 (b), it comes up with a linear regression function from the five samples about the relationship of the band coverage and its write latency. The regression function based on the SMR device we used is $R(x) = 0.728x + 436 (R^2 = 0.9717)$, where x is the write coverage of band.



(a) I/O Latencies **(b)** Regression function

Figure 10. The regressive relationship between the consumed SMR time and the to-be-written block count in one band.

Assuming that S_{dirty} is a collection of some selected $band_s^c$, i.e., $S_{dirty} = \{band_i^c, band_j^c, band_k^c, ...\}$, the SMR overhead of writing S_{dirty}, i.e., $R(S_{dirty})$, can be formulated as Eq. 5,

RIGHTSLINK

where $band^c \in S_{dirty}$, and $Num(band^c)$ is the block number of $band^c$.

$$R(S_{dirty}) = \sum R(Num(band^c)) \quad (5)$$

Generality of CDC. *CDC* is designed to optimize the cache space allocation for the underlying storage devices with unbalance and unstable read/write performance, including but not limited to SMR disks, as long as the cost of evicting clean or dirty blocks can be quantified. In fact, *CDC* is a universal cache space allocation method, not limited to our proposed SAC; it can also be integrated with other cache algorithms (e.g., MOST). Our experiments indicate that *CDC* can adjust the cache resource allocation appropriately when LRU or MOST is adopted for the dirty cache for SMR, with the results of improving the performance by 20.4% for MOST (see Section 6.3 and Table 5 for more).

CDC vs. HRC. Hit Ratio Curve (HRC) is a curve that illustrate the relationship between the cache space and the hit ratio for a given workload. However, the cache resource allocation method based on HRC is not feasible for the SSD cache upon SMR disks. There are mainly two reasons:

(a) The performance of SMR writing is much lower than its reading and it usually fluctuates in a very large range. HRC can only be used to improve the total cache hit ratio; it does not consider the SMR writing factor and cannot make the optimal decision to tradeoff between the hit ratio and the SMR write amplification.

(b) Accurate HRCs can only be gotten for some simple cache algorithms (e.g., LRU). The cache algorithms for SMR should adopt a more complicated scheme (e.g., our proposed SAC). For these SMR-oriented cache algorithms, it is hard to get the accurate HRCs.

5 SMR Hardware Specification Detection

Most of the SMR disk products on the market shield the user from internal technical details as a black box where the information like the PB capacity and the band division that are critical to the algorithm design are hidden. In SAC, the PB capacity gives the upper limit of the cycle length to avoid fragmentation, and the accurate map of band division helps reduce the write amplification. So we provide a set of universal methods to detect these key SMR features in this section; these methods can be applied to most drive-managed SMRs on the market.

5.1 Detection of PB Capacity

For an SMR disk, an interesting characteristic is that its PB capacity does not appear as a fixed value when coupled with different kinds of the workloads and environments (e.g., the block size and the I/O depth settings) [1]. The reason lies in that SMR stores the written data into the PB area, and meanwhile records its information into the PB *metadata* area; as long as any one of the two area reaches the space limit, it triggers the RMW operation (i.e., PB is considered as a full status).

In this case, we provide a method to measure the accurate PB capacity for a given SMR product and representative application workloads. The detailed method includes two steps:

Step 1. Forcing to Empty PB. PB is usually occupied by some unknown size of dirty data, so we need to initiate the PB space first to ensure it is empty. We assume C is an empirical value of PB capacity upper bound; C may be equal to the capacity of the whole SMR disk if we cannot get an upper bound. This requires two operations: ***1) Write the LBA range from 0 to*** C ***in random order.*** This is to squeeze out the old data in the PB, only leaving the data located in LBA 0 to C. ***2) Write this LBA range in sequential order.*** SMR will let these sequential data directly write back to the band region and drop the old data in the PB, then the PB is cleaned.

Step 2. Random Writing. After emptying PB, we do random writing in the range from 0 to C again to refill the PB. Meanwhile, we monitor the IOPS and the writing data amount. The IOPS initially stays high at the beginning, i.e., in the stage of sequential writing to PB. After a while, when the PB is full and it begins to clean up the data, a sudden drop of IOPS will be observed, e.g., 54th second in Fig. 11. The written data amount before the sudden IOPS drop is the PB capacity.

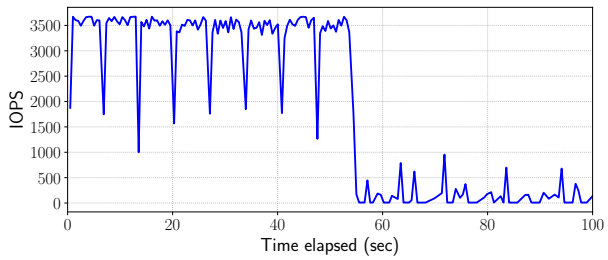


Figure 11. A sudden IOPS drop appears when PB is full during performing random writes to a SMR disk.

Different written block size and the I/O depth setting lead to different PB capacity of SMR drives, so this detection method has to be re-conducted in the specific practical environment. As an example, we have made a series of experiments covering the 4KB and 8KB block size and the I/O depth from 1 to 32 based on a Seagate 8TB SMR drive. The observed IOPS during Step 2 are recorded in Table 3. Under the setting of 4KB block size and 1 I/O depth, IOPS stayed at 3156/s from the beginning and lasted for about 54 seconds, then plummeted to 41/s. The total amount of blocks written during the high throughput period is approximately 720 MB, which means the PB capacity is about 720 MB in this case.

RIGHTSLINK

Table 3. PB capacity detection results with different workload characteristics and system settings.

Blksize	I/O Depth	IOPS	PB Cap.
4KB	1	3156 ⟶ [54s] ↘ 41	720MB
8KB	1	3268 ⟶ [58s] ↘ 9	1450MB
4KB	32	2716 ⟶ [71s] ↘ 225	723MB
8KB	32	2684 ⟶ [75s] ↘ 229	1557MB
256KB	1	452 ⟶ [268] ↘ 21	30.3GB

5.2 Detection of Band Division

Detecting the SMR band division is a bit complex because the band size is usually not a constant in many SMR products. For example, the band size of the SMR drive ST5000AS0011 decreases from 36 MB to 17 MB as the LBA grows [1]. In the future, as SMR capacity continues to expand, there will be a wider band range. It requires us to detect the band size distribution of any given SMR drive.

Band Size Detection. Inspired by the band detection method of Skylight [1], we can estimate the band size for a given LBA. The detailed detection method is illustrated as follows.

Band Size Detection for a Given LBA

1. We first select a target LBA range, which size is 2× PB size, around the given LBA.
2. Randomly write the target LBA range in 4KB granularity, and the total data amount is equal to the PB size to fill PB.
3. Write some data out of the target LBA range to trigger some RMW operations for a few seconds.
4. Read all blocks with 1MB block size in this LBA range in the sequential order.

After PB is filled with the data in the 2× PB size target LBA range in step 2, the new written data in step 3 can trigger some RMW operations and clear the data of some bands to the SMR band region. In this case, the latency distribution of the sequential reading in step 4 can identify a cleaned band. For a band that is cleared in step 3, all its data will not appear in PB and all located in the band region, so the read latency will be constantly low. Otherwise, for a band that is not cleared, the read operations cause the disk head to swing back and forth between the PB and the band region and thus exhibit high latencies due to the head seek time.

Taking the measured results in Fig. 12 as an example, the band size near the space LBA = 0 of is observed to be about 40 MB. According to a series of measurements for different target LBAs, we know that the band size range of the drive ST8000AS0011 is about 15 ∼ 40MB.

Alternative Bands Division. In the practical implementation of SAC, the SMR band size is considered as a constant value (e.g., 20 MB in our implementation), but not following the varying size distribution (e.g., 40 ∼ 15 MB). The reason lies in the following aspects:

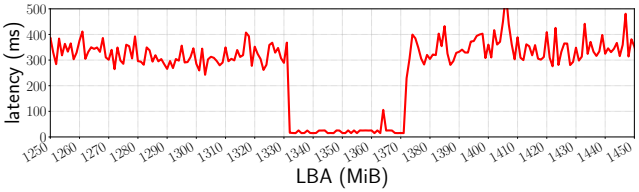


Figure 12. The interval of low-latency during sequential reading indicates the size of a cleared band from PB.

1) The measurement of SMR band size distribution has some errors, because the first and the last low latency requests may be from the neighbor bands, and the prefetching mechanism interferes with latency.

2) It brings heavy overhead to maintain an uneven band size distribution in algorithms like SAC, and importantly, the performance loss of the even band size distribution assumption is not too much compared with the practical varying band size distribution. If we evict the cached blocks of a 20-MB band, but not according to the practical band size, the maximum write amplification rate is to be 4 (= $40MB \times 2 \ / \ 20MB$), that occurs when this zone spans the two largest bands. And in most case, the write amplification rate is smaller.

6 Evaluation

This section will give a comprehensive evaluation for our proposed SAC algorithm. After the experimental setup introduced in Section 6.1, we conduct performance comparison experiments between SAC and other typical cache algorithms for the write-only and read/write-mixed scenarios, respectively in Sections 6.2 and 6.3. Then the experiments to evaluate the impacts of SAC's inner modules including the strategy of *TBS*, the *cycle length*, and the *alternative band size* in Sections 6.4, 6.5, and 6.6, respectively.

6.1 Experimental Setup

Our experiments were driven by ten real-world enterprise I/O traces released by Microsoft Research in Cambridge [14] summarized in Table 4. The last one, i.e., the *long* trace, is a large trace that consists of previous traces and extends the LBA range to 4TB. To this end, we copied each of these traces for 18 times with shifted LBA offsets and then mixed them all.

The experiments were performed in a CentOS Linux release 7.6.1810 (Core) based on the Kernel 4.20.13-1 environment, coupled with 4 cores Intel i7-3770 CPU @ 3.40GHz and 14GB DRAM. All the measurements are based on the Seagate 8TB SMR drive model ST0008AS0002 [18], of which we have detected the hardware specification as described in Section 5, and an 800GB PCIe SSD as the cache layer of the hybrid storage in which the available cache size is set as 5‰ of the total capacity of the accessed SMR bands.

RIGHTSLINK

Table 4. Real-world trances used in the evaluations.

Trace	Server Function	Total Requests ($\times 10^6$)	Write Percent	Written LBA Range (GB)	Accessed LBA Range (GB)
src	Source control	14.0	83.2%	3.80	3.93
prn	Print server	17.6	80.2%	20.22	20.26
ts	Terminal server	4.2	74.1%	9.80	9.81
wdev	Test web server	2.6	72.7%	4.71	4.73
mds	Media server	2.9	70.4%	3.58	3.73
stg	Web staging	6.0	68.2%	7.29	7.58
hm	Hardware monitoring	8.9	67.3%	9.07	9.19
web	Web/SQL server	9.6	46.4%	7.11	8.35
usr	User home directories	12.8	27.9%	6.42	6.92
rsrch	Research projects	3.2	88.8%	17.7	17.7
long	mixed	1481.4	65.7%	3894.3	4332.8

Moreover, based on the CMR disk hardware, we implemented an accurate SMR Emulator, in which we integrated the known SMR working mechanisms and the specifications (e.g., band size distribution and PB size), for the convenience of observing how a cache algorithm affects the inner behavior of the SMR drive. It can provide SMR' inner information including the triggered RMW operation count and the write amplification rates. Note that in this section, all the experimental result of consumed I/O time were based on the real SMR device, while the RMW counts and write amplification rates were obtained from the SMR emulator.

6.2 Performance of Write-only Workloads

In this part, we evaluate the performance of the SSD-SMR hybrid storage managed by SAC and the competitors (e.g., LRU and MOST), respectively, under the write-only scene which is critical for SMR disks. The consumed I/O time of running the first 100 million write requests of the *long* trace on real SMR device of the three cache algorithms is shown in Fig. 13. For the three algorithms, their I/O time spent on SSD layer are quite close, but their I/O time on SMR is far different. Specifically, SAC achieves up to a 11.4× speedup on the SMR I/O time and 7.5× speedup of total I/O time compared with LRU; compared with MOST, the speedups of SAC are 1.6 and 1.4 times for the SMR I/O time and the total I/O time, respectively.

Note that the reason that SAC reduce the SMR I/O time compared with MOST lies in two aspects: 1) SAC achieves a lower cache miss rates (i.e., 14.5% vs. 21.7%) to reduce the request count of SMR processing. 2) The average throughput of the SMR disk for SAC is measured to be $15.4MB/s$, while that of MOST is only $9.5MB/s$. This is because SAC leads to a lower SMR write amplification rates (see the below emulator experimental results in Fig. 14 (b)).

In addition, we made some further experiments based on the SMR emulator to observe the inner write amplification rates and RMW counts. When the coupled SSD cache size ranges from 16 GB to 40 GB, the cache miss rates, the SMR write amplification rates, and the trigger RMW operation counts when performing the same write-only workload are given in Fig. 14.

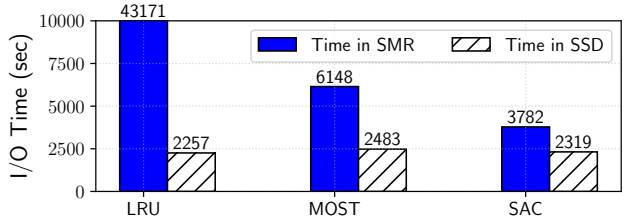


Figure 13. The consumed I/O time for 100 million write requests.

The cache miss rates of SAC under all kinds of cache size settings are between LRU and MOST though, SAC's write amplification rates and RMW counts are always the least, making SAC achieve the best performance among the three algorithms. The RMW count which is highly related to the consumed I/O time (less is better), SAC has a number nearly half of MOST; this is why SAC has much better write performance at the SMR end.

6.3 Performance of Read/Write-mixed Workloads

In this part, as Table 5 shows, we evaluated the performance of the SSD-SMR hybrid storage based on practical SMR products under a read/write-mixed workload coupled with different cache algorithms. The workload contains the first 100 million read and write requests of the *long* trace; the ratio of the read and the write requests is 3.5:6.5. The competitors of SAC include LRU, MOST, and *MOST+CDC*.

MOST+CDC means a cache scheme in which MOST is adopted to manage the dirty cache region, LRU is used to manage the clean cache region, and the Clean/Dirty Comparator (CDC) model of SAC are utilized to balance the cache resources of these two regions. *MOST+CDC* was evaluated here to exhibit CDC's acceleration on other cache algorithms (e.g., MOST).

First of all, according to Table 5, SAC outperforms all the other algorithms in terms of the overall performance, i.e., having the shortest I/O time to finish the all the requests in $40.8K$ seconds, which is 66% less than LRU ($119.3K$ seconds) and 33% less than MOST ($60.7K$ seconds).

Second, *MOST+CDC* helps the original MOST to reduce 17% total time cost, indicating the *CDC* module can improve

RIGHTSLINK

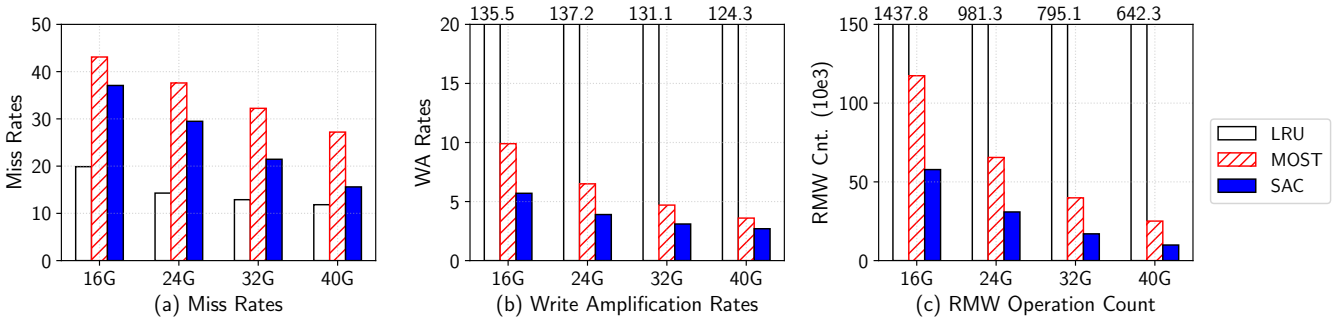


Figure 14. Results based on the emulated SMR disk for write-only workloads.

Table 5. Cache scheme performance in R-W hybrid mode.

Metrics / Cache Scheme	Cache Hit Ratio(%)			Evicted Blks. ($\times 10^6$)		Time Consuming($\times 10^3 sec$)			
	total	read	write	clean	dirty	total	in ssd	read smr	write smr
SAC	**60.7**	**29.9**	**77.7**	**27.4**	**14.6**	**40.8**	**2.8**	**34.8**	**3.2**
LRU	63.8	32.7	81.1	24.4	14.4	119.3	2.8	46.9	69.6
MOST	51.5	0	80.3	38.3	11.4	60.7	1.8	57.1	1.8
MOST + CDC	58.7	29.9	74.8	27.5	16.6	50.4	2.9	44.5	3.0

other band-based cache algorithms by fitting in the SMR characteristics. Compared with MOST, *CDC* cuts down the number of clean block eviction from 38.3*M* to 16.6*M*, but slightly increasing the dirty blocks eviction count from 11.4*M* to 16.6*M*. This result indicates that *CDC* intentionally protects the read cache according to the eviction costs estimated in real time to promote the overall performance.

6.4 Impacts of Target Band Selection Schemes

In this part, we will evaluate the impacts of different target band selection strategies including *P-Decided (PD)*, *A-Decided (AD)*, and *Actual Released Space (ARS)*, which aims to reduce the RMW count and is SAC's default strategy. *PD* means selecting the band with the lowest popularity in each cycle; and *AD* is to select the band containing the most cached blocks number, similar to MOST.

As Fig. 15 illustrates, *PD* achieves the lowest cache miss rates, but the largest write amplification rate; its performance is the worst due to the most RMW operation count. *AD* is almost the opposite to *PD* in terms of both miss rates and write amplification rates. Among the three strategies, ARS is the best one because it is designed to reduce RMWs and it actually achieves a much less one compared with the others.

6.5 Impacts of Cycle Length

The cycle length is critical for the Cycle-Drive Writeback (CDW) module of SAC; it has great impacts on the SMR write amplification rates. We set the default cycle length the same as the SMR PB capacity. Fig. 16 shows the results driven by the first 250 million of the *long* trace, where the cycle length ranges from 50% to 10 times of the default value.

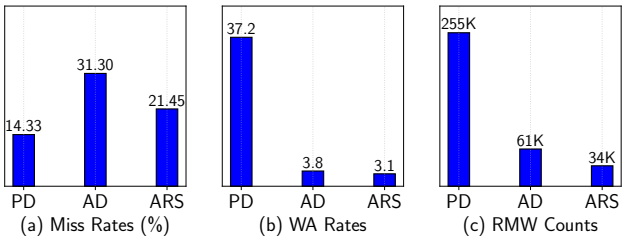


Figure 15. Impacts of target band selection strategies.

We can see from Fig. 16 (b) that the write amplification rates are close for the 0.5x and 1x default cycle length settings, and it has a significant rising steep slope at the 2x length. The reason lies in that the fragmenthation appears more frequently when the cycle length is bigger than PB capacity (recall Section 3). Observed from Fig. 16 (c), we know that the lowest RMW count is obtained when the cycle length is 1x PB size. It means the cycle length close to the PB capacity is the best for performance, which also reflect the importance of detecting the hardware PB size (see Section 5.1).

6.6 Impacts of Alternative Band Size

Recall Section 5.2 that we use the alternative band division method (i.e., the even band size distribution assumption) to deal with the lack of accurate band distribution information of drive-managed SMR disks and avoid high overhead. In this part, we conducted experiments based on the SMR emulator with different alternative band sizes, ranging from 10 MB to 600 MB. As Fig. 17 shows, we see the lines of each metrics are relatively flat in the range of 20 MB ~ 100 MB. The results

RIGHTSLINK

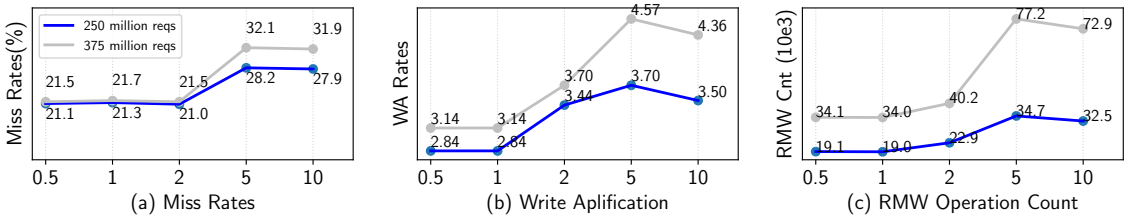


Figure 16. Experimental results of SAC underdifferent cycle lengths ranging from 0.5x to 10x PB size.

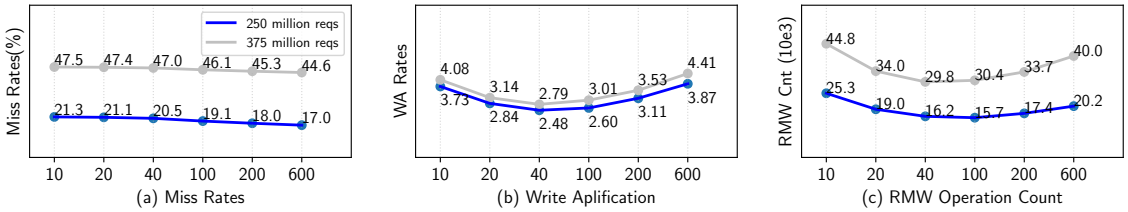


Figure 17. Impacts of different alternative band sizes ranging from 2MB to 600MB.

exhibit that the alternative band size setting does not affect the system performance much in a wide range.

7 Related Work

A. Aghayev et al. [1] proposed a methodology to reveal key properties of drive-managed SMR drives including the details about the persistent write buffer, the band size, etc., which helps a lot in SMR-related system optimization.

Optimization of SMR-based Hybrid Storage. C. Wang et al. [20] designed a write-only cache scheme called *PORE* based on the *MOST* algorithm for SSD-SMR hybrid storage, but did not support the common read/write-mixed scenarios and did not consider the fragmentation issues. W. Xiao et al. [24] implemented three caching algorithm in Flashcache including MOST, LRU, FIFO. The cache layer does not actively cache read request data. X. Xie et al. [25] proposed "Ghost Cache" module to filter out less frequently used data and prevent them from writing to the cache which can improve life of SSD device.

Optimization of SMR Drive. Ma et al. [11] designed to use Flash-based SSDs to replace the persistent write buffer located on tracks and put forward an cache algorithm to manage the write buffer based on the knowledge of band distribution, named *MOST*. T. Yang et al. [26] proposed an improved STL rule for persistent buffer inner SMR drive, which allow dirty blocks whose destination address is the last track of the band to be written back directly from the persistent buffer without RWM operation. C. Ma et al. [10] proposed a built-in flash-cache with fast cleaning for SMR which is aimed to speed up the drive-manage SMR device and to optimize I/O latency by replacing the magnetic recording media of persistent buffer to a flash media. W. He et al. [5] brought a novel mapping method to locate the data first in tracks that are not overlapped with each other, sacrificing space efficiency to gain higher performance. Kadekodi et al. [8] raised an interface scheme named *Caveat-Scriptor* that supports to write anywhere based on static mapping.

8 Conclusion

For the emerging SMR-based high-density disks which are in the trouble of write amplification, we propose a new software/hardware co-design cache algorithm called *SAC*, considering three important facts including the SMR inner hardware structure and schemes, the tradeoff between popularity and SMR write amplification, and the clean/dirty cache regions' self-adjustment according to the real-time SMR read/write performance and workload characteristics. The new features of SAC make it achieve much higher performance for the SMR- based hybrid storage compared with classical cache algorithms like LRU and MOST. Also, the *Clean/Dirty Comparator (CDC)* module in *SAC* is an independent clean/dirty cache balancer for SMR-based high-density disks; it can effectively work well with other cache algorithms (e.g., MOST) to achieve better performance through adjusting the eviction tilt in real-time.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004401), National Natural Science Foundation of China (No. 61972402, 61732014, and 61972275), Beijing Natural Science Foundation (No. 4172031), and open research program of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science (No. CARCH201702). The corresponding author of this paper is Yunpeng Chai (ypchai@ruc.edu.cn).

RIGHTSLINK

References

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

[2] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, David Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony IT Rowstron. 2014. Pelican: A Building Block for Exascale Cold Data Storage.. In *OSDI*. 351–365.

[3] Yuval Cassuto, Marco AA Sanvido, Cyril Guyot, David R Hall, and Zvonimir Z Bandic. 2010. Indirection systems for shingled-recording disk drives. In *Mass Storage Systems and Tec hnologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 1–14.

[4] Dropbox 2018. Extending Magic Pocket Innovation with the first petabyte scale SMR drive deployment. https://blogs.dropbox.com/tech/2018/06/extending-magic-pocket-innovation-with-the-first-petabyte-scale-smr-drive-deployment/.

[5] Weiping He and David HC Du. 2014. Novel Address Mappings for Shingled Write Disks.. In *HotStorage*.

[6] Intel P3500 2017. Intel Solid-State Drive DC P3500 Series: Product Specification. http://www.intel.com/content/www/us/en/solid-state-drives/ssd-dc-p3500-series-product-specification.html.

[7] S. Jiang and X. Zhang. 2002. LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceeding of 2002 ACM SIGMETRICS*.

[8] Saurabh Kadekodi, Swapnil Pimpale, and Garth A Gibson. 2015. Caveat-Scriptor: Write Anywhere Shingled Disks.. In *HotStorage*.

[9] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27. ACM, 134–143.

[10] Chenlin Ma, Zhaoyan Shen, Lei Han, Renhai Chen, and Zili Shao. 2019. FC: Built-in flash cache with fast cleaning for SMR storage systems. *Journal of Systems Architecture* 98 (2019), 214–220.

[11] Wenjian Ma, Liuying abd Xiao, Huanqing Dong, Zhenjun Liu, and Qiang Zhang. 2016. MOST: A High Performance Hybrid Shingled Write Disk System. In *Proceedings of the 22nd National Conference of Information Storage*. CCF.

[12] N. Megiddo and D. Modha. 2003. ARC: a Self-tuning, Low Over-head Replacement Cach. In *Proceedings of the 2nd USENIX Symposium on File and Storage Technologies (FAST'03)*. San Francisco, CA.

[13] Micron 9100 2017. Micron 9100 PCIe NVM SSD. https://www.micron.com/~/media/documents/products/product-flyer/9100_ssd_product_brief.pdf.

[14] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. 2008. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS)* 4, 3 (2008), 10.

[15] David Reinsel, John Gantz, and John Rydning. 2018. The digitization of the world: from edge to core. *Framingham: International Data Corporation* (2018).

[16] Samsung PM1725 2017. Samsung Enterprise SSD MZPLK3T2HCJL (PM1725). http://www.samsung.com/semiconductor/products/flash-storage/enterprise-ssd/MZPLK3T2HCJL?ia=832.

[17] Seagate 2015. Seagate Archive HDD Product Manual: ST5000AS0011,ST5000AS0001. http://www.seagate.com/files/www-content/support-content/enterprise-servers-storage/nearline-storage/archive-hdd/_shared/masters/archive-sata-hdd-%20100743737-product-manual.pdf.

[18] Seagate 2016. Seagate Archive HDD Product Manual: ST6000AS0002,ST800AS0002. http://www.seagate.com/www-content/product-content/hdd-fam/seagate-archive-hdd/en-us/docs/100757960h.pdf.

[19] Christoph Vogler, Claas Abert, Florian Bruckner, Dieter Suess, and Dirk Praetorius. 2016. Heat-assisted magnetic recording of bit-patterned media beyond 10 Tb/in^2. *Applied Physics Letters* 108, 10 (2016), 102406.

[20] Chunling Wang, Dandan Wang, Yiran Cao, and Chao Wang. 2017. Partially Open Region for Eviction (PORE): A SMR-oriented Cache Framework. https://github.com/wcl14/smr-ssd-cache.

[21] Yao Wang and RH Victora. 2013. Reader design for bit patterned media recording at 10 Tb/in^2 density. *IEEE Transactions on Magnetics* 49, 10 (2013), 5208–5214.

[22] Dieter Weller, Gregory Parker, Oleksandr Mosendz, Eric Champion, Barry Stipe, Xiaobin Wang, Timothy Klemmer, Ganping Ju, and Antony Ajan. 2014. A HAMR Media Technology Roadmap to an Areal Density of 4 Tb/in^2. *IEEE transactions on magnetics* 50, 1 (2014), 1–8.

[23] Wikipedia contributors. 2019. Pearson correlation coefficient — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=908824482 [Online; accessed 3-August-2019].

[24] 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 *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 64–71.

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

[26] Tianming Yang, Haitao Wu, Ping Huang, and Fei Zhang. 2017. A Shingle-Aware Persistent Cache Management Scheme for DM-SMR Disks. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 81–88.

[27] Yuanyuan Zhou, Zhifeng Chen, and Kai Li. 2004. Second-level buffer cache management. *IEEE Transactions on parallel and distributed systems* 15, 6 (2004), 505–519.

[28] Jian-Gang Zhu, Xiaochun Zhu, and Yuhui Tang. 2008. Microwave assisted magnetic recording. *IEEE Transactions on Magnetics* 44, 1 (2008), 125–131.

RIGHTSLINK

A Artifact Appendix

A.1 Abstract

Our artifact contains a user-mode cache system which includes some cache algorithms including SAC and an SMR emulator module. The artifact supports and validates all the results of this paper. All the source codes are available on Github. This appendix describes the HW/SW dependencies, the dataset, the experiment workflow, and the evaluation.

A.2 Artifact check-list (meta-information)

- **Algorithm: Cache algorithm, e.g. SAC, LRU, MOST.**
- **Program: Source code in C/C++.**
- **Compilation: GCC.**
- **Data set: All trace files are publicly downloaded on Google Drive.**
- **Run-time environment: Verified on CentOS Linux version 7.6.1810 (core) and expected to work correctly in other Linux distributions.**
- **Hardware: SSD, SMR or regular HDD for emulation.**
- **Run-time state: I/O intensive.**
- **Execution:**
- **Metrics: See section A.6.**
- **Output: Real-time print on standard output.**
- **How much disk space required (approximately)?: For testing real disk I/O, 100GB for the small workloads, and 6TB for the big workload; For no-disk I/O, no disk space required.**
- **How much time is needed to prepare workflow (approximately)?: It needs to download 3GB trace files.**
- **How much time is needed to complete experiments (approximately)?: For testing real disk I/O, tens of minutes for the small workloads, and several hours for the big workload; For no-disk I/O, a few minutes required.**
- **Publicly available?: Yes**
- **Code licenses: GNU General Public License Version 3**
- **Artifact DOI: 10.5281/zenodo.3605763**

A.3 Description

A.3.1 How delivered

The entire SAC project including source code, test scripts, and benchmark datasets can be obtained on Github: https://github.com/dcstrange/sac.

A.3.2 Hardware dependencies

- The Seagate 8TB SMR drive model ST0008AS0002 is recommended (but is optional). We still provide the SMR emulator module in the source code to enable experimenters to perform experiments on regular HDDs. Furthermore, if you just need to inspect the specific metrics (e.g. write amplification, RMW count, and cache hit rates) and are careless of the I/O time, it is recommended to use the `–no-real-io` flag option in command line, in which case, you don't need to deploy any HDD.
- An SSD device (or use a memory file or Ramdisk instead) is required to be the cache layer of the SMR, and it is recommended to reserve 40GB of available capacity.

A.3.3 Software dependencies

- The SAC project has been tested on CentOS Linux release 7.6.1810 (Core) based on Kernel 4.20.13-1 environment and is expected to run correctly in other Linux distributions.
- FIO benchmark required.

A.3.4 Data sets

- All trace files are publicly downloaded on the Google Drive, follow the link
  https://drive.google.com/drive/folders/1zwZYwGB9PbuqAs3wlIE4cpIrYkdgUtYB?usp=sharing.
  Download all the trace files to the project directory `traces/`.
  If the link fails, get the latest location from the project Github.
- We provide 11 trace files based on enterprise applications released by Microsoft Research in Cambridge. Users are very welcome to perform other datasets using the `–workload-file [FILE]` option in command line.

A.4 Installation

Clone the SAC project from Github and compile the source code:

```
$ git clone https://github.com/dcstrange/sac.git
$ cd sac/ && make
```

It generates the binary file `sac`, and then you can test with different options without recompiling. See the README.md of SAC Github project for more detail.

A.5 Experiment workflow

Once you have the executable file `sac`, you need to give the option parameters for testing, and the options include the cache and SMR device file, the running workload, emulator (use or not), etc. There are the steps to prepare, decide the test purpose, and run the experiment.

- *Preliminary.* Before the test, you need to ensure that the SSD and SMR device files are present, and set the files path to the option `–cache-dev` and `–smr-dev`. If you don't have an SMR device, you still can use a regular HDD device file instead and use the SMR emulator module. Please for sure you have the permission to access the files, otherwise, you will get the error information like `errno:13`.

- ***Example 1. Big dataset and real disk I/O***
  ```
  $ ./sac –cache-dev FILE –smr-dev FILE –workload 11 –algorithm SAC
  ```
  For the typical test on SSD-SMR hybrid storage, run the this command which will run the workload number 11 (the *long* trace) for 100 million requests in read and write mix mode, and using the SAC cache algorithm.
  Also, we provide other comparison cache algorithms including LRU, MOST, and MOST with CDC. Change the `–algorithm` option to test other algorithms, such as `–algorithm LRU`.

- ***Example 2. Small dataset and write-only mode***
  ```
  $./sac –cache-dev FILE –smr-dev FILE –algorithm SAC –workload 5 –workload-mode W
  ```
  If you need the workload running in write-only mode, you should add the option `–workload-mode W` which will only

RIGHTSLINK

Output list of Emulator	Description
PB_read_blks	Number of blocks read from the PB
PB_write_blks	Number of blocks write to the PB
SMR_read_blks	Number of blocks read from SMR emulator
PB_write_hits	Number of blocks write hit in the PB
CG_collect_blks	Number of blocks get CG in the PB
RMW_cnt	Total RMW trigger count
CG_bandsize (Byte))	Total size of the bands involved in the GC
WA_avg	Average write amplification of the test

execute write requests in the trace file. There are three optional value of –workload-mode, they are W for write-only mode, R for read-only mode, and RW for read-write mix mode.

- ***Example 3. Use emulator instead of the SMR drive***
  `$./sac –cache-dev FILE –smr-dev FILE –workload 5 –workload-mode W –use-emulator`
  We enable you to verify the SAC and other cache algorithms without a SMR drive. With the option –use-emulator, the program will emulate the behavior of the STL (Shingle Translation Layer) on the regular HDD you specify. The SMR emulator module will then output the information about the I/O time, write amplification, RMW counts, etc.
- ***Example 4. Quick verification without real disk I/O***
  `$./sac –algorithm SAC –workload 5 –use-emulator –no-real-io`
  This command can be used to quickly verify the effectiveness of a cache algorithm, and its performance will be reflected from the results of the SMR emulator where the most critical indicator is RMW trigger count. Note that, this command will not generate the real disk I/O due to the option –no-real-io which discards the request before it is sent to the device, while the metadata structure of the cache algorithm and emulator still running in memory. In this case, your test environment does not need to deploy the SSD and the SMR devices, nor does it need to specify the –cache-dev and –smr-dev.

A.6 Evaluation and expected result

All (intermediate) results are printed on standard output. It is recommended to redirect to a local file when testing in batch. The following output list is performance metrics when the test completed.

Output list of SAC project	Description
totalreqNum	Number of requests completed.
read_req_count	Number of read requests.
write_req_count	Number of write requests.
hitnum	Requests hit in cache.
hitnum_r	Read requests hit in cache.
hitnum_w	Write requests hit in cache.
read_ssd_blocks	Number of blocks read from SSD
flush_ssd_blocks	Number of blocks write to SSD
read_hdd_blocks	Number of blocks read from SMR
flush_dirty_blocks	Number of dirty blocks evict from SSD
flush_clean_blocks	Number of clean blocks evict from SSD
total_run_time (sec)	Total running time of the test
time_read_ssd (sec)	Total I/O time to read the SSD device
time_write_ssd (sec)	Total I/O time to write the SSD device
time_read_smr (sec)	Total I/O time to read the SMR device
time_write_smr (sec)	Total I/O time to write the SMR device

If you run the test with SMR emulator module, you will see extra output metrics from the emulator, the following output list is performance metrics given by the emulator, in which the RMW_cnt is the most critical indicator because it is always the most time-consuming behavior in SMR.

A.7 Experiment customization

Uses are allowed to test their customized cache algorithm. It is required to add the algorithm file to the directory strategy/. See more detail in the project user guide UserCustomizeGuide.md.

A.8 Notes

Before testing the real SMR drive, you need to force clean the PB area of the SMR, otherwise the remaining data will affect the performance. We provide the PB cleaning script in the project scripts/smr-pb-forceclean.sh, run

```
$ ./smr-pb-forceclean.sh [DEVICE FILE]
```

But be careful ***NOT*** to use this script in any production environment, it will overwrite the data on the disk.

If any resource link or path fails, get the latest location from the project Github.

A.9 Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging

RIGHTSLINK