Larger, Cheaper, but Faster: SSD-SMR Hybrid Storage Boosted by a New SMR-oriented Cache Framework

Chunling Wang, Dandan Wang, Yupeng Chai*, Chuanwen Wang, and Diansen Sun
Key Laboratory of Data Engineering and Knowledge Engineering, MOE
School of Information, Renmin University of China, Beijing, China
*Corresponding author: ypchai@ruc.edu.cn

***Abstract*—By utilizing the new Shingled Magnetic Recording (SMR) technique, the emerging SMR disks achieve higher storage density and lower costs. Together with Flash-based SSDs, SMR disks can be used to construct a new hybrid storage system that is larger, but cheaper than the widely used conventional disk-based storage systems. It is appropriate for the fast growing big data applications which have strict requirements on both capacity and cost, e.g., the scientific computing in the fields of meteorology and astronomy. However, the most serious challenge of the SSD-SMR hybrid storage lies in its weak fine-grained random write performance caused by the write amplification in SMR disks, especially when I/Os are spread randomly across the Logical Block Address (LBA) range of the entire disk. Traditional cache algorithms like LRU usually lead to tens of or even hundreds of times more data written into magnetic plates than the original requests, thus resulting in poor I/O performance, because they do not consider and limit the LBA distribution of the eviction victims at all. Therefore, the cache algorithms for SSD-SMR hybrid storage systems should take this new dimension into account except for the data popularity. In this paper, we propose a new SMR-oriented cache framework, i.e., Partially Open Region for Eviction (PORE), which restricts the LBA range of evicted data from SSD caches appropriately to promote the overall performance of the hybrid storage by making a better tradeoff between the cached data popularity and the SMR write amplification. The experiments based on real-world traces reveal that PORE can boost the SSD-SMR hybrid storage performance by a factor of 3.26 $\sim$ 10.2 with only a 2% SSD cache compared with a conventional disk-based storage on average, while the hybrid storage coupled with LRU is slower than the conventional storage sometimes. PORE makes the hybrid storage much faster than traditional storage besides the advantages on capacity and cost.**

I. Introduction

The global data are estimated to reach 44 ZB in 2020 according to IDC [1], and an increasing proportion of them are being involved in various kinds of data analyses in big data applications, such as scientific computing (e.g., in the fields of meteorology, astronomy, or biology), Internet web pages and user logs, medical images, etc. For instance, the storage scale of the European Centre for Medium-Range Weather Forecasts (ECMWF) has reached 100 PB in 2015 and grows about 45% every year [2]. The Large Synoptic Survey Telescope (LSST), a large-aperture, wide-field, ground-based optical survey system to image the sky, is expected to generate 15 TB data per night [3]. As a result, these fast accumulating data raise high requirements on all the aspects of scalability, performance, and costs for big data storage systems.

Faced with such strict requirements on data storage, how should we choose an appropriate storage solution?

1) All Flash arrays [4], DRAM-based storage like RAMCloud [5], and in-memory databases [6] have been used in some practical environments to manage some high-value enterprise data, but they are not suitable for the fast accumulating big data due to their high costs, reliability, or limited capacity.

2) So far, the Conventional Magnetic Recording (CMR) are still the most widely used disk technique at present, with the benefit of large capacity, low costs, and high reliability, but its magnetic recording density is approaching the limit (approximately 1T bit/in^2) [7]. Thus CMR disks cannot follow the growth rate of the big data volume.

3) Shingled Magnetic Recording (SMR) [8], [9] is a new technology that breaks through the upper limit of disk density by squeezing more tracks onto each platter like the overlapped shingles on a roof. SMR disks have lower prices and larger capacities compared with CMR disks, and there have been some practical SMR drive products released [10], [11]. For example, Pelican [12] packs 1,152 SMR disks in one 52U rack, reaching a total capacity up to 5 PB. Moreover, a series of new technologies like BPMR [13], HAMR [14], MAMR [15] are also likely to be combined with SMR, promoting the magnetic recording density further [16].

Motivation. SMR disks have a performance challenge when handling small random writes compared with CMR drives, because they have an inherent SMR write amplification problem for small random writes [17], as a side effect of their high storage density (see §II-A for more details about the SMR technique). Naturally, we can deploy a fast Flash-based SSD as a cache in the block layer under the file system to boost SMR disks. Thus many writes can be absorbed when they hit the cache, leading to fewer of them flushed to SMR disks. But the cache scheme will encounter a new challenge due to SMR disks' characteristic:

According to our experiments, the random write performance of SMR disks declines rapidly by 15.1 times when we increase the LBA range of written data from 56GB to 1TB, because of enlarged SMR write amplification rates (see §II-B). Traditional cache replacement algorithms (e.g., LRU, LIRS [18], ARC [19], etc.) only consider the popularity of data blocks when selecting eviction victims. Thus the victims may have a wide LBA range, leading to serious SMR write amplification and poor I/O performance of both SMR disks

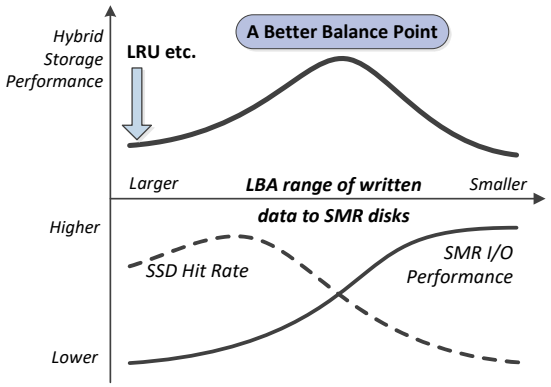


Fig. 1. The SSD-SMR hybrid storage performance trend along with different LBA ranges of evicted cache blocks. **The best performance can be achieved by making a good tradeoff between the cache hit rate and the I/O performance of SMR disks.**

and the SSD-SMR hybrid storage.

Fig. 1 illustrates the approximate trends of the performance of an SSD-SMR hybrid storage system, the SSD cache hit rates, and the SMR I/O performance, when the dirty data blocks that the SSD cache writes back into SMR disks are distributed in different sizes of the LBA range. Along with the decrease of the LBA range of written data, we can achieve a higher I/O performance of SMR disks due to lower write amplification rate and a declining SSD cache hit rate on the whole. Note that restricting the range moderately may lead to a higher hit rate than LRU (see §V-D). Compared with traditional cache algorithms which do not have any limitation about the evicted data's locations, we can achieve a higher performance of the hybrid storage by making a better tradeoff between hit rates and the I/O performance of the underlying SMR disk.

Basic Idea. Motivated by the above discussions, we propose a new SMR-oriented cache framework, i.e., Partially Open Region for Eviction (PORE), which restricts the LBA range of evicted dirty blocks appropriately to promote the overall performance of the SSD-SMR hybrid storage by taking into account both the hit rate and the SMR write amplification. According to PORE, only the cached dirty blocks that are located in a specified region of disk address are open for eviction, while the other dirty blocks are locked in SSDs temporarily. Note that PORE does not have any restriction about clean blocks' eviction, because they will not affect SMR disks' I/O performance much. The open region for eviction is not fixed forever, but is changed periodically. Along with the PORE cache framework, we also propose three schemes to determine the open regions periodically in this paper.

The experimental results based on some typical real-world enterprise traces exhibit that our proposed PORE cache framework can boost the SSD-SMR hybrid storage significantly with a 2% SSD cache, 3.26 times higher than a CMR-only system and 2.86 times higher than the hybrid storage coupled with LRU on average in the read-write cache mode; the values in the write-only cache mode are larger, i.e., 10.2 times and 4.60 times, respecitvely. Note that under half of the real-world traces used in the evaluations, the hybrid storage managed by LRU is very close to or slower than the CMR-only storage, while PORE can always achieve much higher performance than CMR-only.

As a consequence, SSD-SMR hybrid storage systems not only have larger capacities compared with traditional CMR-only storage systems, but also can achieve higher performance by the boost of SSD caches optimized for SMR disks coupled with PORE. In addition, the costs of the hybrid storage will be lowered compared with the CMR-only storage, because larger capacities of SMR disks lead to reduced count of server nodes, and thus the reduced costs of server nodes, server rooms, and cooling devices will be much larger than the additional SSD costs. Therefore, the SSD-SMR hybrid storage is an attractive solution to supply larger, cheaper, but faster storage systems for big data applications.

The rest of the paper is organized as follows. We first introduce the SMR technique and analyze the performance of SMR disks in §II, and then discuss how to optimize the cache schemes to improve the performance of the SSD-SMR hybrid storage in §III. §IV describes our proposed PORE cache framework. The evaluations about PORE and existing solutions are given in §V, and §VI presents the related work. Finally, §VII concludes this paper with a summary of our contributions.

II. Performance Analysis of SMR Drives

Shingle Magnetic Recording (SMR) is a new disk technique with distinctive features. It trades performance, especially the random write performance, for higher storage density and larger capacities. In this section, we will introduce the SMR technique in detail (§II-A), discover SMR disks' most important performance feature based on some measurements (§II-B), and finally give an random-writing performance model of the off-the-shelf drive-managed SMR disks (§II-C).

A. The Shingled Magnetic Recording Technique

Shingled magnetic recording adopts a brand new way of track organization, overlapping adjacent tracks to increase the storage density on the platters like the overlapped shingles on a roof. Some following new technologies like BPMR [13], HAMR [14], and MAMR [15] are also likely to be combined with SMR, promoting the magnetic recording density to 10T bit/in^2 and higher [16].

As Fig. 2 shows, in SMR disks, the read head is much thinner than the write head, so it can read data effectively through the narrow overlapped disk tracks. However, when writing data into a track in SMR disks, the data on the following adjacent tracks will be destroyed at the same time.

To support random writing, expanding the application scenario of SMR disks, a design of *band* appears in SMR drives. A certain amount of consecutive overlapped tracks are packed as a *band*; the last track in a band is not overlapped by others. As Fig. 2 plots, bands are independent of each other without any overlap. In this case, writing one band will not affect any other bands at all. Yet inside a band, writing

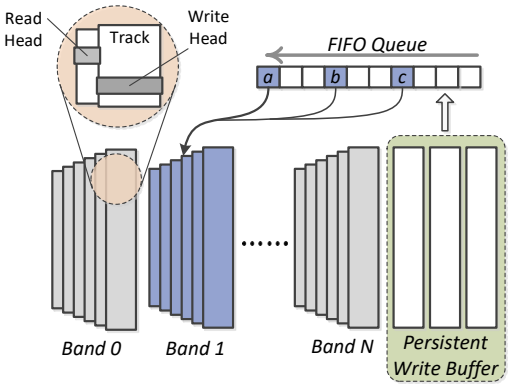


Fig. 2. An example of drive-managed SMR disks' internal structure.

any size of data into the band may also overwrite existing data, so we need to re-write the whole band in this case, i.e., all the data in the band are first copied to memory, modified by updating the newly written contents, and then written back into the band. This behaviour is called the SMR *Write Amplification* phenomenon, which usually becomes their performance bottleneck. Assuming the band size is S_{band}, and the size of the newly written data is S_w, the practical happened I/O amount is enlarged from S_w to $2 \times S_{band}$ including both the reading and writing of the whole band.

In order to reduce the SMR write amplification, a persistent write buffer is usually set in an SMR disk product. The write buffer may be a built-in SSD or some non-overlapped tracks on disks. In some practical SMR products [10], [11], FIFO is usually adopted to manage the persistent write buffer for simplicity. Just as Fig. 2 shows, after the FIFO queue is full and the first entered block *a* will be evicted and written back to its located disk tracks (e.g., in *band 1*). The practical performed operations may include reading the whole *band 1* into memory, merging the latest version of in-FIFO blocks *a*, *b*, and *c*, which are all located in *band 1*, into the memory buffer of *band 1*, and then writing the new version of *band 1* back into its location. The setting of the write buffer can reduce the write amplification of SMR drives. Taking Fig. 2 for example, writing back data blocks *a*, *b*, and *c* only consumes the update of one band thanks to the FIFO queue, instead of independently updating the band for three times.

There are mainly two kinds of SMR disk products currently: host-managed and drive-managed SMR disks [20]. The former means the host knows the exact inner structure and schemes of SMR disks and is responsible for the detailed management of SMR disks, while the latter means the disk drive manages and hides all SMR issues and appears like a CMR disk for the host. Since drive-managed disks are the mainstream products on the market now, in this paper, we discuss the SSD-SMR hybrid storage systems based on drive-managed SMR disks. Although the drive-managed SMR disks hide all the SMR details, there have been already some reverse engineering work like *Skylight* [21] to discover the key properties of practical drive-managed SMR disks.

For example, according to *Skylight*, a Seagate 5900RPM 5TB SMR disk [10] adopts FIFO to manage the persistent write buffer, and the write buffer has 20GB space utilizing some outer tracks that are non-overlapped with each other. Furthermore, the disk uses an aggressive manner to clean the FIFO queue, i.e., data in the persistent write buffer are written back to bands during idle time. The band size ranges from 17 to 36 MB.

B. Performance Features of SMR Drives

In this part, we will introduce our performance measurements based on a Seagate 5TB SMR disk (i.e., ST5000AS0011 [10]) and a typical 7200RPM 500GB CMR disk.

According to our experiments, the most prominent performance characteristic of SMR disks lies in that SMR disks can keep a similar random write performance with CMR disks when the written data are focused in a small LBA range, while a larger LBA range leads to an increasing performance gap between SMR and CMR disks. Fig. 3 shows the throughput of 8-KB random writing performed on the above mentioned SMR and CMR disks respectively. When the written LBA range is enlarged from 56GB to 1TB, the throughput of the CMR disk only drops a bit from 1.46 to 1.05 MB/s. The reason is that the more scattered distribution of written data increases the seek distances of the disk heads.

When it comes to SMR, the throughput decreases from 1.36 to 0.09 MB/s, declining for 93.4%, while that of CMR is only 28.1%. In the experiments, we first fill the 20GB persistent write buffer of the SMR disk, and then write enough data to make sure the throughput is measured during data are actually being written to SMR bands. The fast falling of the SMR disk's throughput along with the written LBA range lies in the SMR write amplification phenomenon. Larger LBA range means that the 20GB data in the persistent write buffer may be located in more SMR bands. We need to flush the dirty data blocks from the 20GB write buffer to more SMR bands.

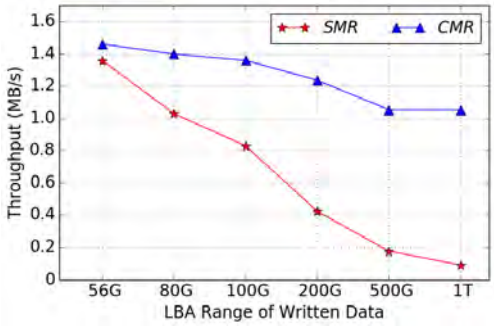


Fig. 3. **The most prominent performance feature of SMR disks**: A larger LBA range of written data leads to a larger SMR write amplification rate and lower I/O performance.

The above performance feature of SMR disks gives us a hint that if we can restrict the written data in a relatively small disk space range, we can get a satisfactory random write performance of SMR disks.

C. Performance Model of SMR disks' Random Writing

To have an explicit understand of SMR disks' random write performance, we will establish a simplified performance model

based on the detailed behaviors happened during the process of random writing in SMR disks. Generally speaking, writing data into SMR drives may gradually trigger the following four steps: (1) writing a data block to the SMR disk persistent write buffer, (2) reading a whole SMR band into memory when evicting a block from the write buffer, (3) reading all the blocks located in this band from the write buffer to update the in-memory buffer of the band, and (4) writing the whole band back.

Assuming S bytes are written into an SMR disk, S bytes of data will be first written into and then read from the persistent write buffer, according to the above analysis. However, when the S bytes of data are evicted from the write buffer, more data will be read from and written into SMR bands. Supposing the write amplification rate is R_{wa}, the read or write bandwidth of the persistent write buffer are respectively B_{rbuf} and B_{wbuf}, and those of the SMR bands are B_{rband} and B_{wband}, We can use the total consumed I/O time to finish all the four above steps of writing S bytes of data to measure the performance of SMR devices, shown as Eq. 1.

$$T = f(\frac{S}{B_{rbuf}} + \frac{S}{B_{wbuf}}, \frac{S \times R_{wa}}{B_{rband}} + \frac{S \times R_{wa}}{B_{wband}}) \quad (1)$$

In Eq. 1, the consumed I/O time of the write buffer and the SMR bands are the two parameters of the function $f()$, because there are two kinds of persistent write buffer: First, if the persistent write buffer is a built-in small SSD in an SMR drive, the I/Os of the write buffer can be performed in parallel with those of the SMR bands. In this case, $f()$ means the maximum value of the two part of time, i.e., the SMR bands in general. Second, if the write buffer is the outer non-overlapped tracks on the disk, they share the same group of magnetic heads with the SMR bands, so the two parts of time should be added.

Although the access granularity of the write buffer is usually small (e.g., 4 $\sim$ 16 KB), the operating system usually combines many of the small requests into large ones because the accesses to the write buffer have close disk locations. Because the write buffer undertakes a small I/O amount compared with the amplified I/Os of SMR bands, the write buffer usually accounts for a small proportion in the total consumed I/O time. In order to support the above analysis, we have made some experiments to measure the consumed I/O time of the write buffer and SMR bands respectively based on an SMR disk emulator developed by us (see §V-A). Different parts of the consumed I/O time of performing the write requests of four enterprise block I/O traces (i.e., *wdev*, *mds*, *ts*, and *usr*, see §V-B and Table I for more) based on the emulated SMR disk are shown in Fig. 4, we can see that SMR bands indeed accounts for the majority of the consumed I/O time due to the write amplification. Note that in the experiments, we use the *O_DIRECT* mode of file operations to measure the actual disk I/O performance accurately.

In a word, the consumed time of writing some data into an SMR disk randomly is mainly determined by two factors, S and R_{wa}. S is the work assigned by the upper applications.

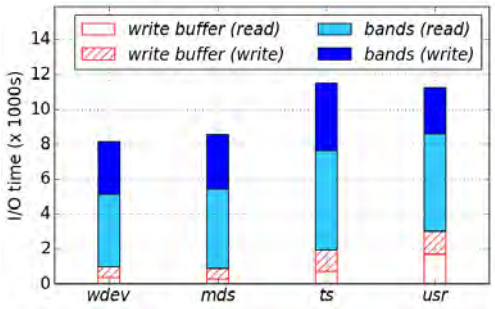


Fig. 4. The SMR bands usually consume the majority of the I/O time when performing random writes due to the write amplification phenomenon.

If there is a write cache layer upon SMR disks, promoting the cache hit rate can reduce the amount of S by merging multiple write requests to the same block. R_{wa} can be reduced by narrowing down the LBA range of data written to SMR disks.

III. Cache Scheme Optimization Objective for SSD-SMR Hybrid Storage

In this section, we will first give a detailed performance analysis about the SSD-SMR hybrid storage. Similar to the SMR-only storage systems (§III-A), the SSD-SMR hybrid storage systems coupled with traditional cache algorithms have the same performance challenge caused by serious SMR write amplification. To solve this problem, a possible better plan of cache schemes is discussed in §III-B.

A. SMR-only Storage Systems

Recall Eq. 1 in §II-C that the written data size (i.e., S) and the write amplification rate (i.e., R_{wa}) are both in positive correlation with the consumed I/O time of SMR disks. As Fig. 5 (a) shows, when writing data directly into SMR disks, we cannot reduce both S and R_{wa}, usually leading to a poor performance.

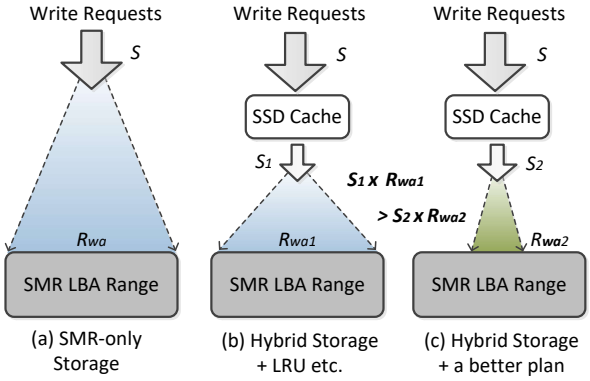


Fig. 5. Accessing SMR disks directly and SSD-SMR hybrid storage coupled with different cache schemes.

Based on nine real-world enterprise traces (see §V-B and Table I for more), we performed a trace analysis to estimate the write amplification rates of accessing SMR disks directly, as Fig. 6 shows. It is obvious that all the enterprise traces achieve very high SMR write amplification rates in the range of 41.3 $\sim$ 171.5, 113.0 on average. This will lead to poor

I/O performance. The analysis methodology is presented as follows:

In the analysis, we assume adopting a Seagate 5TB SMR disk [10] with a 20GB write buffer (i.e., $1/256$ of disk capacity) and 17 $\sim$ 36 MB bands in a uniform distribution (see §II-A). Only the write requests of the traces are processed and the read ones are dismissed. For a trace, we set a virtual FIFO queue whose size is $1/256$ of the total working set size. Each time the virtual FIFO queue is filled, we accumulate the total size of all the covered bands and then clean up the queue. In this case, we can estimate the write amplification rate of SMR disks as the total band size divided by the capacity of WS.

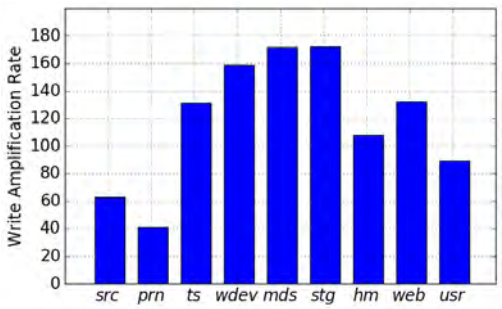


Fig. 6. Real-world traces usually lead to very large SMR write amplification rates when SMR disks are accessed directly.

B. SSD-SMR Hybrid Storage Systems

For an SSD-SMR hybrid storage, because SSD caches are much faster than SMR disks, the overall performance is mainly determined by the SMR disks, especially the I/Os of SMR bands. Therefore, the overall performance of a hybrid storage depends mainly on the multiple of the new write amplification rate R_{wa1} and the reduced write amount S_1 due to cache hits, as Fig. 5 (b) shows.

*1) **Traditional Cache Algorithms**:* Traditional cache algorithms evict data blocks only according to their popularity to pursue as high hit rates as possible and thus as small S_1 as possible. These cache schemes evaluate data popularity according to recency (e.g., LRU), frequency (e.g., LFU), access intervals (e.g., LIRS [18]), or their mixture (e.g., LRFU [22], ARC [19], and MQ [23]). Only promoting hit rates to lower S_1 for high I/O performance is enough when adopting CMR disks as the underlying storage devices.

However, when SMR disks are deployed, their I/O performance is sensitive to the LBA ranage of the written data, thus raising a new challenge for the cache replacement schemes. Therefore, although the SSD cache can reduce the practical written data size to SMR disks from S to S_1 by cache hits, traditional cache schemes do not limit or reduce the accessed space range of SMR disks, i.e., their SMR write amplification rate R_{wa1} is very close to that of the SMR-only storage (i.e., R_{wa}), so they cannot achieve the satisfactory overall performance of the hybrid storage (Please refer to the experimental results in §V-C).

*2) **A Better Plan**:* In fact, there may be a better plan for the SSD-SMR hybrid storage. As Fig. 5 (c) shows, different with the SMR-only storage, the deployed SSD cache is authorized to select data blocks to evict and write back to SMR disks. Traditional cache algorithms usually evict the coldest blocks to pursue the least value of S_1, but we can adjust the scheme to reduce the LBA range of evicted dirty blocks, leading to a much smaller write amplification rate, R_{wa2}. Although S_2 may be a bit larger than S_1, the remarkable reduction of write amplification may lead to a better overall performance of the hybrid storage system by satisfying $S_2 \times R_{wa2} < S_1 \times R_{wa1}$.

In summary, the cache schemes for SMR disks should consider a new dimension of the LBA range of evicted dirty blocks, and aim to reduce the SMR write amplification rate significantly with a modest loss on cache hit rates. Thus the overall performance of the hybrid storage will be much improved.

IV. PORE: A New SMR-oriented Cache Framework

In this section, we propose a new SMR-orientated cache framework, i.e., Partially Open Region for Eviction (PORE), to tradeoff between data popularity and SMR write amplification for better performance. §IV-A will first give an overview of PORE, followed by the workflow of PORE described in §IV-B and the schemes for determining the open region in §IV-C.

A. Overview of PORE

In order to reduce SMR write amplification rates, we can set some parts of SMR LBA range as *Open Region*, the rest as *Forbidden Region*. As Fig. 7 shows, the clean cache blocks are not affected by the setting of *Open Region* and *Forbidden Region*; for the dirty cache blocks, only the ones which are located in the *Open Region* (e.g., block a, b, and c in Fig. 7) are allowed to be evicted; those blocks in *Forbidden Region* will be skipped when selecting a victim (e.g., block d).

In this case, the SMR write amplification rate can be reduced. For example, when the open region covers 200 GB space and the persistent write buffer of an SMR disk is 20 GB, the write amplification rate in this case is around 10 (i.e., 200 GB / 20 GB). Without the setting of *Open Region*, blocks in the write buffer may be located in any bands, so we may need to re-write the whole SMR disk to write them back. This leads to a much larger write amplification rate (e.g., 5 TB / 20 GB = 256).

As Fig. 7 shows, this mechanism is called Partially Open Region for Eviction (PORE), opening part of the SMR disk space for dirty blocks' eviction to reduce the SMR write amplification and periodically choosing an appropriate part of SMR LBA range as *Open Region* to maintain a good cache hit rate. By combining the LBA-aware eviction restriction and the block-level cache replacement manner (instead of evicting all cache blocks located in a large disk region together), PORE can improve the hybrid storage's performance by achieving both low SMR write amplification rates and high cache hit rates.

As a side effect of lowering the write amplification rate effectively, only the cache blocks located in *Open Region* and clean blocks can be chosen for eviction. Yet PORE will not

affect the cache hit rates much due to the following three reasons:

(1) The *Open Region* usually covers a large part of SMR LBA range, so the cache blocks in SSDs with the *Open Region* tag include both hot and cold ones. Among them, the cold ones are evicted gradually, while the hot ones can stay in the SSD cache (e.g., the hot cache blocks a and b in Fig. 7). In this case, PORE can take the performance advantages of both keeping hot blocks in caches for high hit rates and reducing SMR write amplification rates with the eviction limitation of *Open Region* at the same time.

(2) The division of *Open Region* and *Forbidden Region* on SMR disk space is periodically changed. After evicting a certain number of dirty cache blocks in *Open Region*, a new *Open Region* and a new *Forbidden Region* will be set. In this case, the cached cold blocks located in any place of the SMR disk have the chance to be evicted. They will not always be kept in SSD caches to lower the average quality of cached blocks.

(3) The selection of *Open Region* can be optimized for high hit rates. For example, if we set a very cold LBA range in *Open Region*, the eviction choice of PORE is similar to traditional cache replacement, not affecting cache hit rate much. In this paper, we propose different kinds of schemes of the region division to get different optimization objectives (see §IV-C).

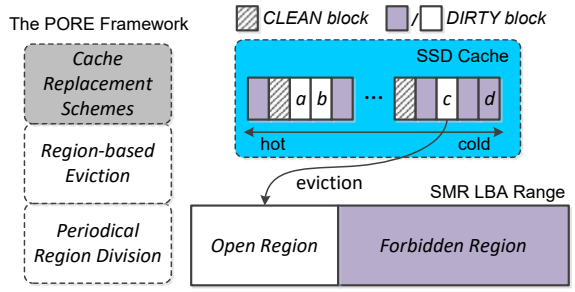


Fig. 7. Overview of the Partially Open Region for Eviction (PORE) cache framework.

B. Workflow of PORE

As Fig. 7 shows, PORE is not a specific cache replacement scheme, but an open cache framework. Any other cache replacement algorithms can be used in PORE to manage the cache blocks in SSD caches. PORE includes two core modules, i.e., *Region-based Eviction* and *Periodical Region Division*. The former is responsible for calling the deployed cache replacement scheme (e.g., LRU) to evict one of the cached blocks, while the latter updates the division of *Open Region* and *Forbidden Region* periodically.

1) ***Region-Aware Eviction****:* In PORE, we add a new tag in the metadata of the cached dirty blocks to identify the region that the block belongs to (i.e., *Open Region* or *Forbidden Region*), consuming 1 bit per block, which will not introduce obvious burden for memory space. When an eviction is required, the coldest clean block or open-region dirty block according to the deployed cache scheme will be chosen. Note that we still adopt a block-level cache replacement, but not evicting the blocks in a band together, because it can protect the hot blocks from being evicted to maintain a high cache hit rate.

However, this may introduce additional time to find a victim block because it may be not the last one in the cache queue. Taking Fig. 7 for example, if LRU is adopted, the original cache replacement without PORE can get the coldest block (i.e., d) immediately from the cache block array or linked list. When it works in PORE, we have to find the victim until the third one (i.e., c) . Yet all of these operations happen in memory, and will not affect the overall performance much in the I/O-intensive storage systems. What is more, for the queue- or queues-based cache algorithms like LRU, we introduce an additional linked list for all the cache blocks in *Open Region* to boost the lookup process.

2) ***Basic Unit of Region Division****:* When setting *Open Region*, the best plan is to align it to the boundaries of SMR bands. However, the bands in an SMR disk are in uncertain sizes (e.g., the Seagate 5T SMR disk [10] has different band sizes in the range of 17 $\sim$ 36 MB), so we do not know the exact band size distribution for the drive-managed SMR products.

Therefore, we first divide the whole SMR disk address space into *zones* with the fixed size, and adopt *zone* as the basic unit to form *Open Region* and *Forbidden Region*, as Fig. 8 plots. The zones in the *Open Region* are called *Open Zones*, while those in the *Forbidden Region* are called *Forbidden Zones*.

Generally, how should we set an appropriate size for *zone*? Too small size of zones is not a good choice, because it will lead to the large amount of zones, bringing high overhead to manage them. What is more, the too small zone size setting increases the risk of big size difference between a band and its related open zones, just like the case in Fig. 8 shows. In consequence, when the open zone is very small compared with the band, evicting the blocks in this open zone will lead to a large write amplification rate.

On the other side, a too large zone setting is also not good for performance, because it may combine different types of bands together into one big zone. Some of these bands may be suitable for being an open zone, while some others do not. For example, a band containing many hot dirty blocks should not be chosen for *Open Region* by itself, but it may be involved in *Open Region* when the average popularity of its related big zone is lowered by its adjacent bands.

Therefore, we should set a moderate size for zones, generally in the same order of magnitude with the minimum size of SMR bands. More discussions and evaluations about the impacts of different zone size settings can be found in §V-E.

3) ***Periodical Region Division****:* If the region division is fixed, we may have to evict a hot block in *Open Region*, instead of a cold one in *Forbidden Region*, lowering the cache hit rates. In PORE, therefore, we will change the setting of the two kinds of regions periodically. In this case, the cold dirty blocks located in different disk areas all have the chance to be evicted in turns. At the beginning of each *region division period*, we will choose some zones to form the *Open Region* according to the access records of all the zones in the last

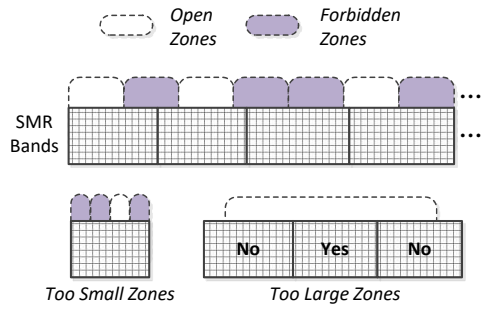


Fig. 8. **Zones with fixed size are the basic unit to construct *Open Region*.** Too small zone settings increase the risk of too large write amplification, while too large zones may combine different types of bands together, bringing some unsuitable bands into *Open Region*.

period, while the other zones compose the *Forbidden Region*. Particularly, the first region division happens when the SSD cache is just filled.

When the the count of dirty blocks written into SSD cache reaches a specified threshold (i.e., the period length), a new period starts to set new *Open Region* and *Forbidden Region*. The write amplification rate of SMR disks can be estimated as Eq. 2, where S_{or} is the size of *Open Region*, S_{wb} is the size of the SMR write buffer, and L_p is the length of PORE's region division period. The write amplification rate is much lower than the original value of S_{whole_disk}/S_{wb}. Note that all the above variables are measured in the block count.

$$R_{wa} \approx \begin{cases} \dfrac{S_{or}}{S_{wb}}, & L_p \geq S_{wb} \\ \dfrac{S_{or}}{L_p}, & L_p < S_{wb} \end{cases} \tag{2}$$

Only when the dirty blocks located in the same SMR band are already in the SMR write buffer (e.g., blocks a_1, a_2, and a_3 in Fig 9 (a)), they can be flushed to the SMR band at the same time to reduce write amplification. When the period length is longer than the write buffer size, the cache blocks that have not been evicted from SSDs (e.g., block a_4) cannot be flushed to SMR bands together with a_1, a_2, and a_3. Therefore, the write amplification rate is equal to S_{or}/S_{wb}, with a constant value of denominator, but the numerator, i.e., the size of the corresponding *Open Region*, has to be enlarged to supply enough eviction victims in the next period, leading to larger write amplification.

On the contrary, when the period length is smaller than the write buffer size, blocks from different rounds of *Open Regions* may be mixed in the persistent write buffer of SMR drives before writing back to bands. As Fig. 9 (b) shows, because *Period B* has different *Open Region* with *Period A*, blocks in *Period B* usually cannot be flushed to SMR bands together with the blocks in *Period A*. In this case, the write amplification rate is S_{or}/L_p. Generally, the open region size increases along with the period length, the value of R_{wa} will not change much when the period length is no longer than the write buffer size.

However, because PORE's zones are not exactly aligned with SMR bands, the total size of the covered bands of an *Open Region* is usually a bit larger than the total size of all open zones. When the period length is too small, the size gap between bands and zones will be nonnegligible, resulting in a bit larger write amplification rates. In addition, too small period lengths will also increase the computing overhead of selecting open zones, so the best setting of period length is exactly the SMR write buffer size.

A good news is that we can discover the exact length of the SMR persistent write buffer through the reverse engineering like *Skylight* [21]. For example, the write buffer of Seagate 5T SMR disks [10] is 20GB and adopts the FIFO scheme. We can choose an appropriate setting for the length of the *region-division period* according to the size of the FIFO queue. The impacts of different period lengths are evaluated in §V-E.

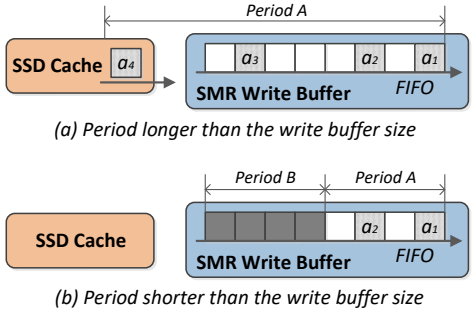


Fig. 9. Different settings of region division period length in PORE.

C. Schemes of Determining Open Region

How to select appropriate zones to form the *Open Region* is very important for the performance of PORE. In this part, we will respectively present the general procedure of the open zone selection, the principle of zone evaluation, and our proposed three schemes of selecting open zones.

1) ***General Procedure of Open Zone Selection****:* In PORE, when a region division is triggered, we first sort all the zones according to a specified method, and then put the front zones in the sorted list into an *Open Zones* set until the total cached dirty block count of all the selected zones in the *Open Zones* set reaches the value of the region-division period length. In this case, we can guarantee there are enough dirty cache blocks from the *Open Region* to be evicted in the following period. Note that both the period length and the open zone selection do not consider clean cache blocks.

In addition, some new write requests during the following period will put some dirty blocks located in these open zones into the SSD cache, i.e., the cached block count in *Open Region* is usually larger than the eviction requirement. Therefore, the best blocks in the *Open Region* will usually not need to be evicted, with the results of more cache hits for better performance.

2) ***Principle of Zone Evaluation****:* In the above procedure of open zone selection, the method to evaluate and sort all the zones is a critical part. As Fig. 10 shows, if we measure all the candidate zones according to both popularity and coverage

rates, i.e., the total size of all the dirty blocks in the SSD cache located in this zone divided by the fixed zone size, all the zones can be put into four quadrants. First, the zones with colder blocks on average should be chosen for open zones, because evicting the cold blocks in them will not lower cache hit rates much. Second, the zones with large coverage rates are more appropriate for the *Open Region*, because we need fewer zones to supply enough eviction victims in the following period. The size of *Open Region* (i.e., S_{or} in Eq. 2) is determined by the zone counts that are selected as open zones, so the high-coverage-rate zones will lead to smaller values of both S_{or} and the write amplification rate.

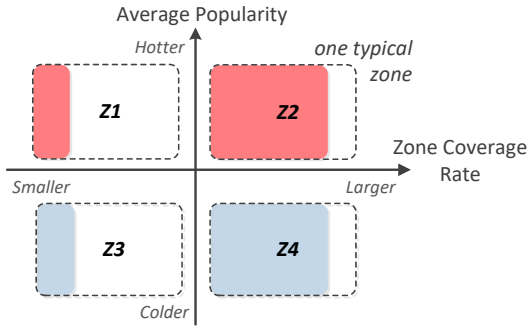


Fig. 10. We should consider both zones' popularity and their coverage rates when selecting open zones.

Based on the above discussion, zones in the top left quadrant (e.g., $Z1$) in Fig. 10 should not be selected for open region, and the zones in the bottom right quadrant (e.g., $Z4$) are the best for being open zones. However, how to set the priority of zones in the other quadrants is an open problem.

*3) **Schemes of Selecting Open Zones**:* Based on different weight settings of popularity and coverage rates of zones, we have proposed three schemes of selecting open zones with different optimization objectives.

Coverage First (*CF*). *CF* is a simple scheme that only considers the zone coverage rate factor to select open zones. We first calculate the coverage rates of all the zones according to their cached block counts in SSDs, and then sort them in a descending order of the coverage rates. The zones with the most cache blocks construct the *Open Region* according to *CF*, so the zone count in *Open Region* is the least, leading to the smallest write amplification rate.

Popularity First (*PF*). In *PF*, we only need to consider the average block popularity of a zone. We sort all the zones by the average access count per block in an ascending order, and the coldest zones are assigned as open zones. *PF* usually selects more zones each time compared with *CF* to have a larger write amplification rate, but it has a better protection for the hot blocks to generate a higher hit rate. In fact, *PF* may lead to a higher hit rate than LRU even with a restricted eviction region, because the warm data blocks with occasional large access intervals in the *Forbidden Region* will be protected in the SSD cache to avoid too early eviction (see §V-D for more).

BaLancing between Zone Coverage and Popularity (*BL*). The above two schemes, *CF* and *PF*, both select open zones only from one perspective. In order to consider the two factors at the same time, we design a third scheme that balances between zone coverage and popularity (*BL*). The characteristic of *BL* is to sort zones according to their values of average block access count divided by the zone coverage rate. A higher zone coverage rate and worse block popularity both contribute to increase the possibility of a zone to be involved in *Open Region*. For instance, according to *BL*, $Z4$ in Fig. 10 usually has the highest priority to be selected as an open zone, while $Z1$ has the lowest priority. For $Z2$ and $Z3$, the values of their average popularity and coverage rate determine the result together. The above schemes will be evaluated in §V-D.

V. Evaluations

In this section, PORE and a series of existing cache algorithms are evaluated based on a prototype SSD-SMR hybrid storage system constructed by us (see §V-A). After introducing the experimental setup in §V-B, the overall results under typical configurations are shown in §V-C. Then the three schemes of selecting open zones proposed by us are evaluated in §V-D, and the impacts of different parameters of PORE and different SSD cache size settings are respectively evaluated in §V-E and V-F. Finally, §V-C1 introduces the experiments performed based on a real SMR disk.

A. The Prototype Hybrid Storage

We have designed and implemented an SSD-SMR prototype storage [25], with 3,000+ LOC in *C*. The prototype system performs and measures the practical I/O accesses and cache replacement behaviors based on real SSDs and disks according to some real-world block I/O traces. There are four main modules in our prototype:

*1) **Trace replay module**:* This module reads a specified trace file and generates the corresponding I/O requests to the assigned disk address. It will access data on disks or the SSD cache by calling the interfaces of the following SSD cache module. The trace files that our system supports are the block-level I/O traces, such as the Microsoft Cambridge I/O traces [26] used in our experiments.

*2) **SSD cache module**:* The cache replacement behaviors, a group of existing cache schemes (e.g., LRU), and our proposed PORE cache framework are all implemented in our prototype system. This module manages the metadata of cached blocks in memory and performs I/Os on a file residing in a practical SSD when doing cache block accesses or cache replacements.

*3) **SMR disk emulator module**:* Although we can perform I/O requests directly on practical SMR disks (e.g., the Seagate 5TB SMR disk introduced above [10]), the measured performance of SMR disks driven by some real-world I/O traces may be not accurate. The reason is that the written LBA range of these real-world traces are far different with each other, but the inner persistent write buffer is fixed as 20 GB. In this case, when the written LBA ranges of some traces are much smaller than 5 TB, the capacity of the SMR disk, the practical SMR write amplification rate will be much lower than the practical situation of enterprise storage systems, leading to a meaningless high performance of SMR disks. In fact, it is not

TABLE I
REAL-WORLD TRACES USED IN THE EVALUATIONS.

Trace	Server Function	Total Requests	Write Percent	Written LBA Range (GB)	Accessed LBA Range (GB)
src	Source control	14,024,860	83.2%	3.80	3.93
prn	Print server	17,635,766	80.2%	20.22	20.26
ts	Terminal server	4,216,457	74.1%	9.80	9.81
wdev	Test web server	2,654,824	72.7%	4.71	4.73
mds	Media server	2,916,662	70.4%	3.58	3.73
stg	Web staging	6,098,667	68.2%	7.29	7.58
hm	Hardware monitoring	8,985,487	67.3%	9.07	9.19
web	Web/SQL server	9,642,398	46.4%	7.11	8.35
usr	User home directories	12,873,274	27.9%	6.42	6.92

fair to use the same in write buffer size for traces with different written LBA ranges.

Therefore, based on the detected SMR disks' inner structure and schemes [21], we realize an SMR disk emulator based on a CMR disk in our prototype. In the emulated SMR disk, the persistent write buffer and the emulated SMR bands are both located in the same CMR disk, just as practical SMR disks do. Similarly, we adopt the FIFO scheme to manage the persistent write buffer. The emulated SMR bands have the sizes ranging from 17 MB to 36 MB, and the band sizes are in a uniform distribution. The behavior of updating contents into a band is just the same as that of the practical SMR disk products (see §II-A for more). Thus, we can set the size of the persistent write buffer to 1/256 (i.e., 20GB/5TB) of the written LBA range of the running real-world trace, so as to get the reasonable experimental results similar to practical enterprise environments.

*4) **Statistics module**:* The final module is used to record important metrics to evaluate the performance of the hybrid storage system, including the cache hit rates for read or write requests, I/O time and amounts of different levels (i.e., the SSD cache, the persistent write buffer and the bands of emulated SMR disks), etc.

B. Experimental Setup

We conduct experiments based on our SSD-SMR hybrid storage prototype driven by a series of real-world enterprise I/O traces. All the experiments in this section except for §V-G are performed in a Linux 2.6.32 environment coupled with 8 GB DRAM, a 7200RPM 500GB CMR disk, and a 240GB PCIe SSD.

Performance Metrics. In our experiments, the timestamps in the traces are all dismissed; instead, the block-level I/O requests are performed continuously. Thus the overall performance of the hybrid storage can be reflected by the consumed total I/O time. In order to evaluate SMR-oriented cache schemes comprehensively, except for the classical metric, i.e., the cache hit rates for read or write requests, we also calculate the write amplification rate of SMR disks, i.e. the I/O amount written to SMR bands divided by that evicted from the SMR persistent write buffer.

Competitors. Except for the widely used LRU scheme, which only considers data popularity for eviction, we also design and implement an intuitive optimized cache strategy based on LRU, i.e., LRU-band, which evicts all the cache blocks located in the same SMR band together to reduce SMR write amplification rate when evicting the least recently used block. Note that LRU-band is set to know the exact band distribution on disks for an ideal performance, while our proposed PORE does not. The three schemes of selecting open zones, i.e., *CF*, *PF*, and *BL* (see §IV-C for more) are all evaluated in the experiments, and *BL* is the default scheme of PORE.

Traces. Our experiments are driven by nine real-world enterprise I/O traces from the storage traces released by Microsoft Research in Cambridge [26]. The information of these traces has been summarized in Table I. Because we aim to optimize the random write performance of SMR disks in this paper, most of the selected traces belong to write-intensive ones. Note that the written LBA range of one trace in this table is the total capacity of its accessed SMR bands; the accessed LBA range is the union of the written LBA range and the LBA range of all read requests in each trace.

Default Settings. All the file accesses are in the *O_DIRECT* mode because all the above traces are the I/O request records of block devices, i.e., the trace files do not include the requests that had already hit by the DRAM cache in enterprise servers [26].

C. Overall Results

PORE can be deployed to manage a read-write cache, adding the restriction of dirty cache blocks' eviction (i.e., the dirty blocks located in the *Forbidden Region* cannot be evicted) to improve the write performance of the underlying SMR disks, while not affecting the eviction of clean cache blocks. In fact, PORE only has effects on boosting write requests, and is not designed to accelerate the read ones, so we can also deploy PORE to manage a write-only cache, while leaving the independant read-only for other cache schemes. The above two cases are both evaluated below, while the read-only cache scheme is not discussed in this paper.

The experiments are performed based on the emulated SMR disk, and we measure the overall performance of our proposed PORE coupled with the *BL* scheme along with LRU, LRU-band, the SMR-only and the CMR-only solutions. By default, the SMR persistent write buffer is set as 1/256 of the total capacity, maintaining the same ratio between the 20 GB write buffer and 5 TB storage capacity of the practical Seagate SMR product (i.e., 20GB/5TB = 1/256) [10]. In the read-write cache mode, the SSD cache size is set to 2% of the accessed LBA

range in each trace, and the SSD cache size is set to 2% of the written LBA range in the write-only cache mode.

*1) **Mananging a Read-Write Cache**:* The overall performance can be reflected directly by the consumed total I/O time to accomplish all the requests in the traces. Considering the long test time of performing large amounts of I/Os on disks, we choose four typical traces among the above nine traces listed in Table I, i.e., *wdev*, *mds*, *ts*, and *usr*, to do the experiments. Among the four traces, the first three belong to write-intensive applications with the write request ratios ranging from 70.4% to 74.1%, while the *usr* trace has only 27.9% write requests.

Fig. 11 reveals the total consumed I/O time of SMR-only storage, CMR-only storage, hybrid storage coupled with LRU, LRU-band, and PORE, respectively. The SMR-only storage is the slowest solution, especially for the write I/O time. For example, the write I/O time of SMR-only is always the dominate part no matter for write-intensive traces or read-intensive traces (i.e., *usr*), which is 3.65 $\sim$ 6.86 times slower than CMR-only.

All the cache solutions including LRU, LRU-band, and our proposed PORE can all achieve smaller read I/O time obviously through SSD cache hits compared with the CMR-only solution, but the write I/O time are not always boosted by the cache for these cache schemes. For example, except *ts*, LRU's write I/O time is longer than CMR-only (i.e., 1.04 $\sim$ 2.01 times of CMR-only), because it does not solve the SMR write amplification problem well. The total I/O time of LRU is also longer than the CMR-only solution for half of the traces (i.e., *wdev* and *mds*).

Compared with LRU and LRU-band, our proposed PORE signinficantly shortens the write I/O time, and thus reduces the total consumed I/O time to boost the entire hybrid storage. Compared with SMR-only, CMR-only, LRU, and LRU-band, the total I/O time reductions of PORE are 11.83, 3.26, 2.84, and 1.63 times, respectively.

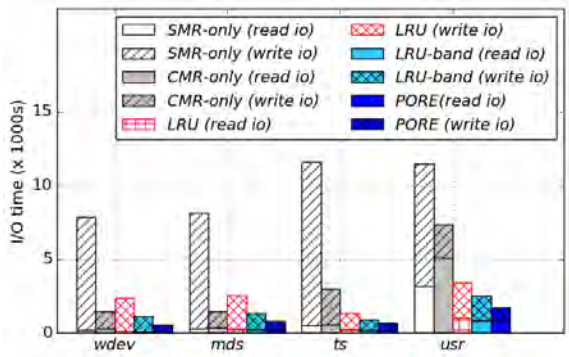


Fig. 11. PORE reduces the total consumed I/O time significantly.

The hit rate and write amplification rate are two important factors to influence the overall performance of the hybrid storage. PORE can achieve much lower I/O time compared with other solutions because it has similar cache hit rates and much lower SMR write amplification rates.

Fig. 12 (a) and (b) respectively give the read and write hit rates of LRU, LRU-band, and PORE based on all the nine traces in Table I. For read cache hit rates, PORE achieves similar or a bit lower values than LRU and LRU-band; write hit rates of PORE are closer to the others. The reason lies in that PORE reduces the choices of eviction victims by setting *Open Region*. But the reduction of cache hit rates is not much, e.g., PORE's read hit rate reduction compared with LRU is 16.15% on average, and the mean write hit rate reduction is only 4.90%.

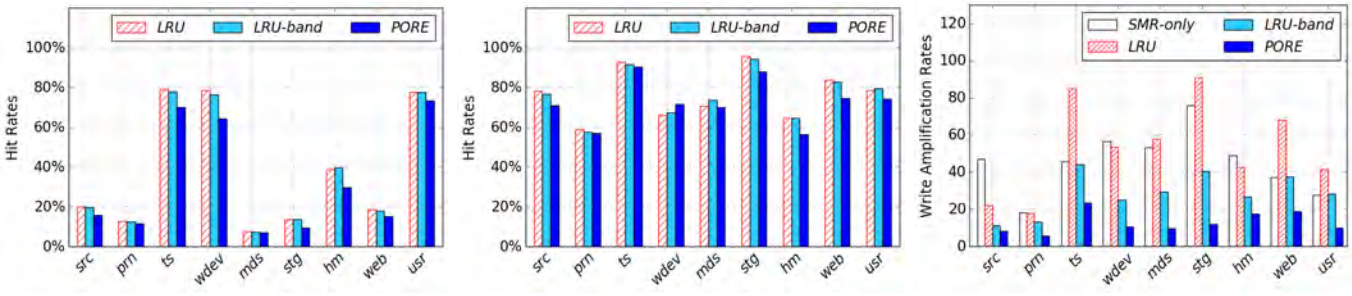
In the aspect of SMR write amplification shown in Fig. 12 (c), PORE achieves significant improvement compared with SMR-only, LRU, and LRU-band. PORE's average reductions on write amplification rates compared with SMR-only, LRU, and LRU-band reach 3.92, 4.82, and 2.31 times, respectively. The results prove that limiting the LBA range of evicted dirty cache blocks in PORE is very effective in reducing write amplification. Note that LRU has larger SMR write amplification rates than the SMR-only solution sometimes, because it does not control the LBA distribution of evicted dirty blocks, leading to unexpected SMR write amplification results.

*2) **Mananging a Write-Only Cache**:* PORE is designed to optimize SMR disks' random write performance, so when the read and write cache are split and managed by different cache schemes independently, we can deploy PORE to manage the write-only cache. In the experiments of this part, we dismiss the read requests in the trace files and use the whole deployed SSD as a write-only cache. In fact, the read requests can be boosted by another independent SSD-based read cache, but it is out of the focus of this paper.

The experiments about consumed I/O time were also performed based on the four traces, i.e., *wdev*, *mds*, *ts*, and *usr*. The results are shown in Fig. 13 (a), including the total consumed I/O time of SMR-only storage, CMR-only storage, hybrid storage coupled with LRU, LRU-band, and PORE, respectively. The results are similar to the read-write cache case above, but the improvement of PORE compared with the other solutions are more remarkable. PORE achieves higher performance than LRU-band and all the other competitors, reducing the I/O time by a factor of 34.9, 10.2, 4.60, and 2.02 compared with SMR-only, CMR-only, LRU, and LRU-band, respectively. Coupled with PORE, the SSD-SMR hybrid storage system becomes much faster than the widely used CMR-only storage systems.

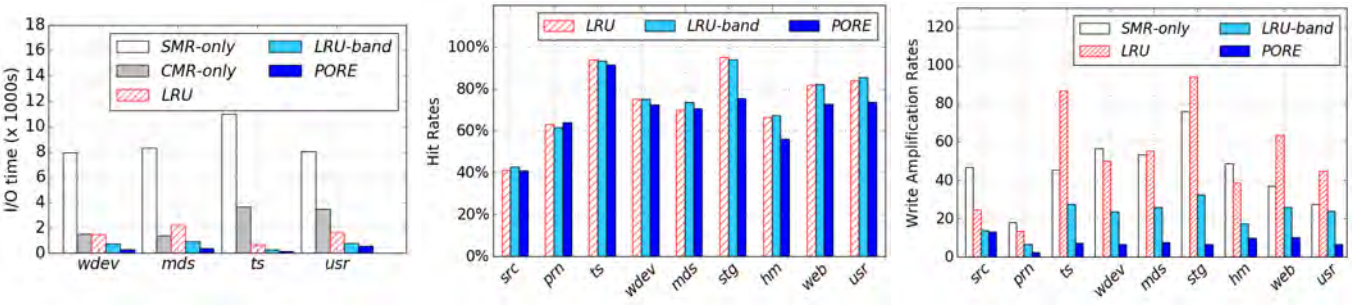
In the SSD cache hit rate aspect, as Fig. 13 (b) shows, LRU-band achieves similar results with LRU, i.e., higher than LRU under six traces while lower under the other traces. It implies that although some hot blocks are evicted ahead of time according to LRU-band compared with LRU, sometimes judging blocks' popularity with the help of their disk location may have positive effects on promoting hit rates. The SSD hit rates of PORE are usually a bit lower but close to LRU, and under two of the traces (i.e., *prn* and *mds*), PORE has higher hit rates than LRU. Although PORE reduces the choices of eviction victims by setting *Open Region*, it does not have many influences on cache hit rates.

For the SMR write amplification rates shown in Fig. 13



(a) Read Hit Rates (b) Write Hit Rates (c) Write Amplification Rates

Fig. 12. PORE achieves satisfactory hit rates and obviously lower write amplification rates compared with other solutions in the read-write cache mode.



(a) Total I/O Time (b) Hit Rates (c) Write Amplification Rates

Fig. 13. Experiment results when cache schemes manage a write-only SSD cache to boost write requests only.

(c), PORE achieves a significant advantage compared with the competitors. We can find that LRU does not improve the SMR write amplification rate obviously, and under five traces (i.e., *ts*, *mds*, *stg*, *web*, and *usr*) it has larger write amplification rates than the SMR-only storage. The average write amplification rate of *SMR-only* is 45.6, while that of LRU is larger, reaching 52.41. The LRU-band reduces the average write amplification rate to 21.99, but PORE achieves a much smaller one, i.e., 7.76, even though PORE does not know the exact SMR band size distribution like LRU-band. The reason lies in that LRU-band is a passive scheme to reduce write amplification by evicting the cache blocks in one band together, but PORE is an active solution to limit the write amplification rate by setting the *Open Region*. PORE reduces the write amplification rate significantly by a factor of 5.88, 6.75, and 2.83 on average compared with SMR-only, LRU, and LRU-band, respectively.

3) ***Comparison Between the Two Above Cases****:* Comparing the results of the above two kinds of cache modes, the cache hit rates do not have many differences, but PORE achieves larger SMR write amplimification rate reduction and performance improvement in the write-only cache mode. The reason lies in two aspects:

(1) PORE can only boost the process of write requests, so according to Amdahl's law, the improvement of the read-write cache mode should be lower than the write-only cache.

(2) Because SMR disks have imbalanced read and write performance, not like the read-write-balanced traditional disks, the penalty for evicting a dirty block is much larger than evicting a clean block. In our implement, we only adopt a simple LRU queue to manage all the dirty and clean blocks together. How to appropriately adjust the eviction priority for dirty and clean blocks in an SSD-SMR hybrid storage is an interesting and open problem, but it is out of the focus of this paper. Therefore, because PORE is desgined to boost the random write performance for SMR disks, we adopt the write-only cache mode in the following experiments.

D. Open Zone Selection Schemes

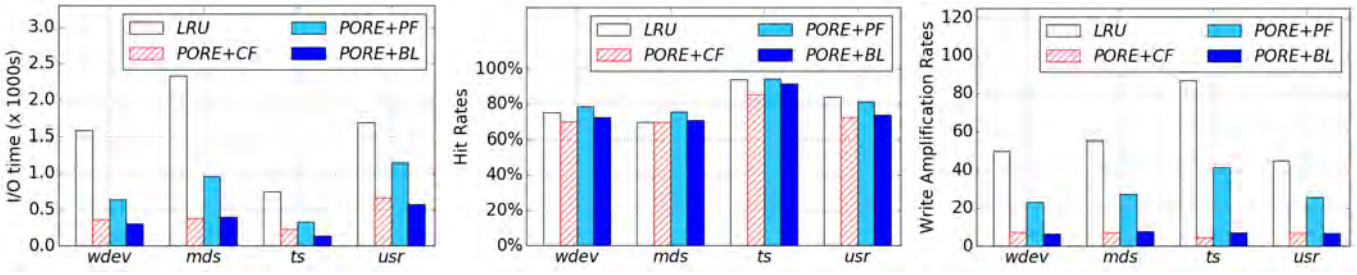
Selecting the best open zones in PORE is an open problem. In this part, we evaluate the three open-zone selecting schemes we proposed, i.e., *CF*, *PF*, and *BL*, by comparing them with LRU, under the same four traces in the above experiments, i.e., *wdev*, *mds*, *ts*, and *usr*, illustrated by Fig. 14.

Among the three schemes, the Popularity First (*PF*) scheme can achieve much higher SSD hit rates than the other two, and its hit rates are higher than LRU in most cases except *usr*. The reason lies in that *PF* restricts that the eviction victims can only be chosen from the coldest zones in SMR disks, with the benefit of protecting blocks in hot zones from eviction. This result gives us a hint to design a cache scheme with high hit rates, not only in the SMR-oriented environment.

In the aspect of write amplification, *PF* is better than LRU, but *CF* and *BL* achieve much lower write amplification rates. For all the four traces, the *BL* scheme consumes the least I/O time, because it achieves similar write amplification rates and higher hit rates compared with *CF*. Therefore, *BL* is chosen as the default open-zone selecting scheme in PORE.

E. Impacts of Different Zone Sizes and Period Lengths

In this part, the comparisons among different parameter settings of PORE, including the size of zones and the length of the region division period, are performed. Fig. 15 gives the total I/O time, the hit rate, and the write amplification



(a) Total I/O Time (b) Hit Rates (c) Write Amplification Rates

Fig. 14. Comparison among different open-zone selecting schemes (i.e., *CF*, *PF*, and *BL*) in the cache framework of PORE.

rate in each case driven by the *wdev* trace. Among the three open-zone selecting schemes evaluated in the experiments, *BL* achieves the best performance in most cases.

As Fig. 15 (b) shows, the zone size setting does not affect the cache hit rate much, so the overall performance reflected by the total consumed I/O time is mainly determined by the write amplification rate.

In the aspects of SMR write amplification and total I/O time shown in Fig. 15 (a) and (c), we find that too large zone settings lead to larger write amplification rate and longer I/O time; too small zone settings also have negative effects on the two metrics (see §IV-B2 for discussions about zone sizes), especially for the *PF* scheme. Too small zone size may increase the overhead of PORE, so we adopt 20MB as the default zone size for *BL*, although we can reduce the zone size appropriately to achieve higher performance further.

The second important parameter of PORE is the region division period length. By default, we set it as the size of the persistent write buffer in SMR disks. Fig. 15 (d)-(f) give the results when the period length ranges from 10% to 10 times of the default value driven by the *wdev* trace as well.

Comparing Fig. 15 (d) and (f), we can find that the trend of I/O time is similar to that of write amplification rate, indicating that the overall performance is mainly determined by the SMR write amplification rate due to the similar cache hit rates under different period lengths shown in Fig. 15 (e).

The default setting, i.e., the period length that is equal to the size of SMR persistent write buffer, usually achieves the best performance for all the three schemes. The reason is that when the period length is longer or shorter than the SMR write buffer size, the write amplification rates will be enlarged (see §IV-B3 for more analysis). Note that when period length is larger than 5 times of the write buffer size, the amplification rates have reached the same level of LRU, so increasing the period length to 10 times of the default value will not promote the write amplification rates further.

F. Impacts of Different SSD Sizes

In this part, Fig. 16 exhibits experimental results when the SSD size ranges from 1% to 5% of the accessed bands' total size of the *wdev* trace. Because PORE is designed to work for a cost-efficient big data storage system, we mainly focus on the case of small SSD capapcities.

It is obvious that the hit rates become higher as the size of SSD cache increases because more hot data can be kept in SSD caches. When the SSD cache is 1% and 5% of the storage capacity, PORE achieves higher cache hit rates, and its hit rates are a bit lower in the cases of 2%, 3%, and 4% compared with LRU and LRU-band.

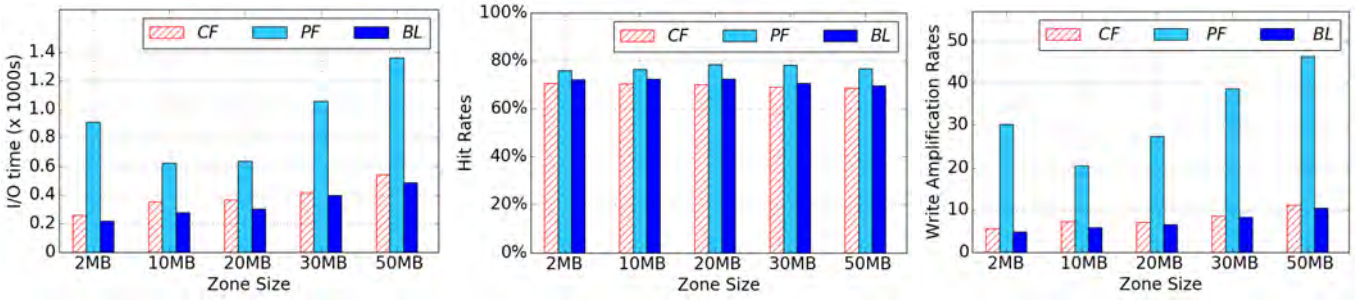
As Fig. 16 (c) shows, for LRU, the write amplification rates are in fluctuation under different SSD cache sizes, because it does not control the disk space distribution of evicted victims at all. For LRU-band and PORE, more cache blocks are located in the same SMR bands when the SSD cache is larger, so fewer SMR bands are needed to be flushed when evicting the same amount of data from SSD caches, resulting in smaller write amplification rates. For example, when the SSD cache increases from 1% to 5%, the SMR write amplification rate reduction of PORE increases from 4.79 to 20.65 times compared with LRU.

Therefore, based on the benefits of both cache hit rate and write amplification rate, the total I/O time drops along with the increase of SSD sizes (see Fig. 16 (a)). In all the cases, PORE achieves better performance than LRU and LRU-band. For example, the I/O time of PORE is only 24.5% and 43.7% of LRU and LRU-band, respectively, for the 1% SSD size; the values are shortened to 5.84% and 18.1% for the 5% SSD size.

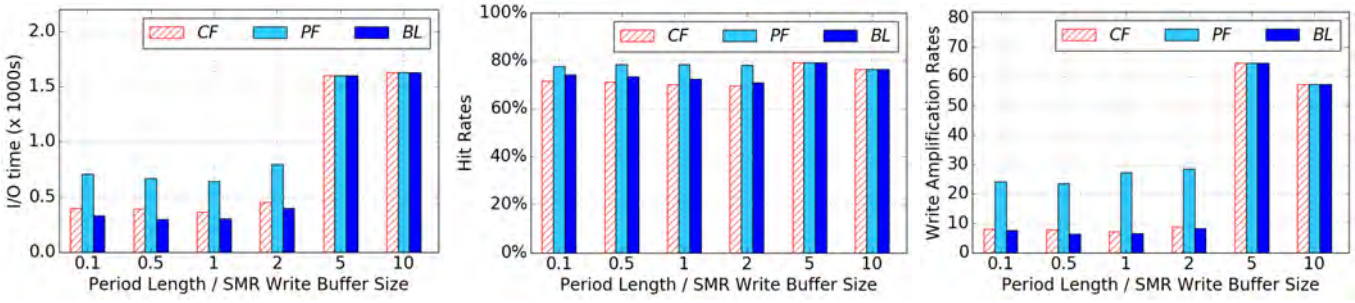
G. Experiments on Real SMR disks

Finally, we compare our proposed PORE with LRU and the CMR-only solution based on a hybrid storage consisted of a Memblaze PBlaze IV 800GB SSD and a 5900RPM Seagate 5TB SMR disk with a 20GB inner persistent write buffer [10], and a 5900RPM CMR disk.

The difficulty of performing experiments on real SMR disks lies in that the written LBA range of requests has to be large enough; otherwise the SMR write amplification rate (i.e., about the written LBA range / 20GB) turns to be small and the performance of SMR disks appears high, much different with practical industry environments. In order to construct a large enough trace with at least a TB-level written LBA range, we first shift the nine traces in Table I into adjacent LBA ranges and shuffle all their requests to contruct a large trace. And then, the merged new trace are duplicated for 16 times; the 16 traces are also be shifted to adjacent LBA ranges and shuffled to form a new trace *mix*, whose written LBA range of *mix* reaches 1.15 TB. Due to the huge number of request records in *mix* and long experiment time, we only perform 50 million

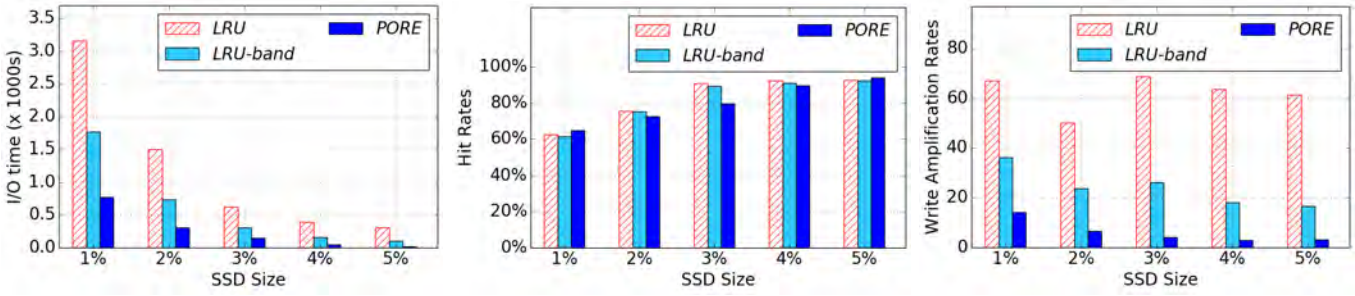


(a) Zone Size: Total I/O Time (b) Zone Size: Hit Rates (c) Zone Size: Write Amplification Rates



(d) Period Length: Total I/O Time (e) Period Length: Hit Rates (f) Period Length: Write Amplification Rates

Fig. 15. Impacts of different settings of the zone size and the period length in PORE.



(a) Total I/O Time (b) Hit Rates (c) Write Amplification Rates

Fig. 16. Impacts of the SSD cache size.

write requests after the 20GB SMR write buffer is filled in the experiments.

As Fig. 17 plots, PORE achieves much higher performance compared with both CMR-only and LRU, although its hit rate is a bit lower than LRU. The I/O time of finishing the same 50 million write requests for CMR-only, LRU, and PORE is repectively 33.4, 13.0, and 5.69 hours. Although the practical written LBA range is much lower than the device capacity (i.e., 5TB), PORE achieves much higher performance than LRU and CMR-only. If the written data scatter in the whole disk space, the advantage of PORE will be more notable.

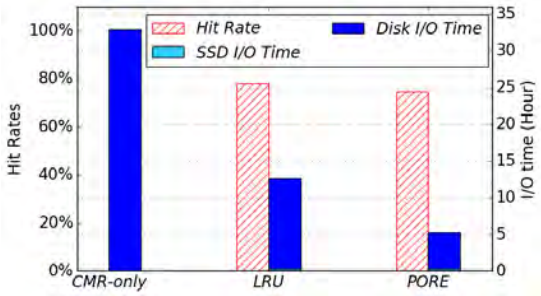


Fig. 17. Experimental results based on real SMR disks.

VI. Related Work

Optimization of SMR Devices. The main challenge of constructing SMR devices lies in that the contents in the overlapped tracks will be overwritten when performing random writing according to the original design of shingled magnetic recording. Y. Cassuto et al. [28] came up with a new concept of indirection system for SMR devices. They proposed a storage unit named Shingled Block (S-block) that is similar to the concept of SMR bands between which are safe gaps in physical level. S-blocks are managed in a circular buffer; the logical addresses of newly written data are mapped to the head of the circular buffer. With the address mapping scheme, this indirection system can support random writing. A garbage collection algorithm recycles the S-block in the tail of the circular buffer by moving the valid blocks to the head. Based on the indirection system, D. Hall et al. [29] proposed a scheme that divides an E-region out of the shingled area as a cache buffer to accelerate random writes. E-region is also arranged as a ring buffer; an enough distance is ensured between the two ends of the ring buffer to ensure data in E-

region cannot be overwritten.

To improve the efficiency of garbage collection, D. Park et al. [30] brought a novel design to classify data into hot ones and cold ones, which makes the out-of-date blocks more concentrate in the hot bands. S. Jones et al. [31] proposed a band compaction algorithm to reduce data movement during the band compaction process, which not only focuses on compacting the emptiest bands, but also considers the write frequency of data in the bands. In the field of address mapping, W. He et al. [17] proposed a novel mapping method to locate data first in tracks that are not overlapped with each other, sacrificing space efficiency to gain higher performance. In addition, Ma et al. [32] proposed 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.

Optimization of SMR-based Systems. Another research direction is to do the indirection work in the upper file system layer based on host-managed SMR disks. C. Jin et al. [33] designed a new file system called *HiSMRfs* that can support random writing without the SMR Translation Layer (STL) in SMR products. G. Gibson et al. [34] proposed a new *SMRfs* that works based on a hybrid system consisted of SSDs and SMR disks, and manages metadata separately to improve performance. S. Kadekodi et al. [35] proposed an interface scheme named *Caveat-Scriptor* that supports to write anywhere based on static mapping. SMR file systems for specific usage have also been proposed, such as SFS for video servers [36] and big data applications [37].

Apart from file systems, there are also some studies on database based on SMR disks. For example, P. Rekha et al. [38] proposed a key-value data store for SMR disks leveraging its better sequential write performance.

Moreover, A. Aghayev et al. [21] proposed a methodology to reverse 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.

SSD Cache Optimization in Hybrid Storage. Flash-based SSDs have been widely used in enterprise storage systems due to their high performance and high storage density. The hybrid storage system consisted of SSDs and disks [39], [40], [41] is also an important storage form in modern data centers because of its advantages on cost performance and data reliability. What is more, there are many industrial products or open-source systems to optimize SSD caches in hybrid storage, such as Facebook Flashcache [42], Linux dm-cache [43], EMC FAST cache [44], and Oracle Smart Flash Cache in Exadata Database Machine [45].

Except for some traditional general-purpose cache algorithms like LRU, MQ [23], LIRS [18], ARC [19], and etc., there are also some specially optimized cache algorithms for SSD-based caches. For example, the second-level ARC (L2ARC) technique in Solaris ZFS file system [46], SieveStore [47], and LARC [48] add different types of new data filters to improve the quality of cached data and meanwhile to reduce the frequency of cache updating. WEC [49] and ETD-Cache [50] protect popular cached data by appropriately extending their staying time in SSD-based cache to avoid too early eviction. RIPQ [51], Pannier [52], and SRC [53] pack small blocks into large containers and replace cached data in the unit of large containers. This mechanism can improve the I/O performance of SSDs due to the reduced write amplification in Flash chips.

Besides, M. Canim et al. [24] proposed a cache algorithm for an SSD caching layer between RAM and hard disks in databases systems. This algorithm monitors the disk access patterns to identify hot regions of the disk. Blocks in hot disk regions have higher priority to be cached in SSDs to avoid the cache pollution from cold regions.

In a word, although there have been some optimized schemes specially designed for SSD-based caches, the SMR-oriented SSD cache scheme is still an open research topic.

VII. Conclusion

In this section, we conclude this paper with a summary of our contributions:

(1) For the design of SMR-oriented cache algorithms, we find a new dimension to consider, i.e., the LBA range of evicted dirty blocks, which is a particular feature of the emerging SMR disks and directly affects the SMR write amplification rate and I/O performance.

(2) For the SSD-SMR hybrid storage systems, we propose a new cache framework called Partially Open Region for Eviction (PORE), which not only improves the performance by cache hits, but also boosts the underlying SMR disks by periodically setting the appropriate *Open Region* allowed for eviction. Coupled with PORE, the SSD-SMR hybrid storage can be larger, cheaper, but faster than the conventional disk-based storage systems.

(3) PORE is an open cache framework; the cache replacement and the open region selection schemes are both flexible. In this paper, we have designed three different open region selection schemes, i.e., *CF*, *PF*, and *BL*, for different optimization objectives.

(4) Our proposed PORE improves the performance significantly due to its hardware/software co-design based on the detected inner structure and schemes of practical SMR disk products. If the host-managed SMR products are available, this kind of co-design cache algorithm can achieve higher performance.

Acknowledgment

We appreciate the anonymous reviewers and Brent Welch sincerely for improving the quality of this paper. The authors would also like to thank Yiran Cao and Chao Wang for the contributions to the prototype system, and Zihan Li and Fang Zhou for the trace analysis work.

This work was supported by Beijing Natural Science Foundation (No. 4172031) and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (No. 16XNLQ02).

REFERENCES

[1] Vernon Turner, John F Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. *IDC Analyze the Future*, 2014.

[2] Matthias Grawinkel, Lars Nagel, Markus Mäsker, Federico Padua, André Brinkmann, and Lennart Sorth. Analysis of the ecmwf storage landscape. In *FAST*, pages 15–27, 2015.

[3] Mario Jurić, Jeffrey Kantor, KT Lim, Robert H Lupton, Gregory Dubois-Felsmann, Tim Jenness, Tim S Axelrod, Jovan Aleksić, Roberta A Allsman, Yusra AlSayyad, et al. The lsst data management system. *arXiv preprint arXiv:1512.07914*, 2015.

[4] Pure Storage. Pure storage flasharray. Technical report, FlashArray Technical Specificatinos, 2016.

[5] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, et al. The ramcloud storage system. *ACM Transactions on Computer Systems (TOCS)*, 33(3):7, 2015.

[6] SAP HANA. Sap hana vora. Technical report, SAP Information Sheet SAP HANA Vora, 2016.

[7] Wikipedia. Perpendicular recording/advantages. https://en.wikipedia.org/wiki/Perpendicular_recording, 2017.

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

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

[10] Seagate archive hdd product manual: ST5000AS0011,ST5000AS0001, 2015. 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.

[11] Seagate archive hdd product manual: ST6000AS0002,ST800AS0002, 2016. http://www.seagate.com/www-content/product-content/hdd-fam/seagate-archive-hdd/en-us/docs/100757960h.pdf.

[12] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, David Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony IT Rowstron. Pelican: A building block for exascale cold data storage. In *OSDI*, pages 351–365, 2014.

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

[14] Dieter Weller, Gregory Parker, Oleksandr Mosendz, Eric Champion, Barry Stipe, Xiaobin Wang, Timothy Klemmer, Ganping Ju, and Antony Ajan. A hamr media technology roadmap to an areal density of 4 tb/in^2. *IEEE transactions on magnetics*, 50(1):1–8, 2014.

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

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

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

[18] S. Jiang and X. Zhang. Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proceeding of 2002 ACM SIGMETRICS*, 2002.

[19] N. Megiddo and D. Modha. 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, 2003.

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

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

[22] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 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*, volume 27, pages 134–143. ACM, 1999.

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

[24] Mustafa Canim, George A Mihaila, Bishwaranjan Bhattacharjee, Kenneth A Ross, and Christian A Lang. Ssd bufferpool extensions for database systems. *Proceedings of the VLDB Endowment*, 3(1-2):1435–1446, 2010.

[25] Yiran Cao Chunling Wang, Dandan Wang and Chao Wang. Partially open region for eviction (pore): A smr-oriented cache framework, 2017. https://github.com/wcl14/smr-ssd-cache.

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

[27] Microsoft production server traces, 2017. http://iotta.snia.org/traces/158.

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

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

[30] Dongchul Park, Chung-I Lin, and David HC Du. H-swd: A novel shingled write disk scheme based on hot and cold data identification. In *10th USENIX Conference on File and Storage Technologies (FAST12)*, 2012.

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

[32] Liuying Ma, Wenjian Xiao, Huanqing Dong, Zhenjun Liu, and Qiang Zhang. Most: A high performance hybrid shingled write disk system. In *Proceedings of the 22nd National Conference of Information Storage*, 2016.

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

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

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

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

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

[38] Rekha Pitchumani, James Hughes, and Ethan L Miller. Smrdb: key-value data store for shingled magnetic recording disks. In *Proceedings of the 8th ACM International Systems and Storage Conference*, page 18. ACM, 2015.

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

[40] Taeho Kgil, David Roberts, and Trevor Mudge. Improving nand flash based disk caches. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 327–338. IEEE, 2008.

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

[42] Mohan Srinivasan and Paul Saab. A general purpose, write-back block cache for linux, 2016. https://github.com/facebookarchive/flashcache.

[43] Mike Snitzer Joe Thornber, Heinz Mauelshagen et al. dm-cache, 2016. https://en.wikipedia.org/wiki/Dm-cache.

[44] Emc fast cache: A detailed review, 2011. http://www.emc.com/collateral/software/white-papers/h8046-clariion-celerra-unified-fast-cache-wp.pdf.

[45] Exadata smart flash cache features and the oracle exadata database machine, 2013. http://www.oracle.com/technetwork/serverstorage/engineered-systems/exadata/exadata-smart-flashcache-366203.pdf.

[46] Brendan Gregg. L2arc, 2008. https://blogs.oracle.com/brendan/entry/test.

[47] Timothy Pritchett and Mithuna Thottethodi. Sievestore: a highly-selective, ensemble-level disk cache for cost-performance. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 163–174. ACM, 2010.

[48] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng. Improving flash-based disk cache with lazy adaptive replacement. In *Proceedings of the 29th International Conference on Massive Storage systems and Technology (MSST'13))*, 2013.

[49] Yunpeng Chai, Zhihui Du, Xiao Qin, and David A Bader. Wec: Improving durability of ssd cache drives by caching write-efficient data. *IEEE Transactions on Computers*, 64(11):3304–3316, 2015.

[50] Ningwei Dai, Yunpeng Chai, Yushi Liang, and Chunling Wang. Etd-cache: an expiration-time driven cache scheme to make ssd-based read cache endurable and cost-efficient. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, page 26. ACM, 2015.

[51] Linpeng Tang, Qi Huang, Wyatt Lloyd, Sanjeev Kumar, and Kai Li. Ripq: Advanced photo caching on flash for facebook. In *FAST*, pages 373–386, 2015.

[52] Cheng Li, Philip Shilane, Fred Douglis, and Grant Wallace. Pannier: A container-based flash cache for compound objects. In *Proceedings of the 16th Annual Middleware Conference*, pages 50–62. ACM, 2015.

[53] Yongseok Oh, Eunjae Lee, Choulseung Hyun, Jongmoo Choi, Donghee Lee, and Sam H Noh. Enabling cost-effective flash based caching with an array of commodity ssds. In *Proceedings of the 16th Annual Middleware Conference*, pages 63–74. ACM, 2015.