ZoneAlloy: Elastic Data and Space Management for Hybrid SMR Drives

Fenggang Wu   Bingzhe Li   Zhichao Cao   Baoquan Zhang
Ming-Hong Yang   Hao Wen   David H.C. Du
University of Minnesota, Twin Cities

Abstract

The emergence of Hybrid Shingled Magnetic Recording (H-SMR) allows dynamic conversion of the recording format between Conventional Magnetic Recording (CMR) and SMR on a single disk drive. H-SMR is promising for its ability to manage the performance/capacity trade-off on the disk platters and to adaptively support different application scenarios in large-scale storage systems. However, there is little research on how to efficiently manage data and space in such H-SMR drives.

In this paper, we present ZoneAlloy, an elastic data and space management scheme for H-SMR drives, to explore the benefit of using such drives. ZoneAlloy initially allocates CMR space for the application and then gradually converts the disk format from CMR to SMR to create more space for the application. ZoneAlloy controls the overhead of the format conversion on the application I/O with our quantized migration mechanism. When data is stored in an SMR area, ZoneAlloy reduces the SMR update overhead using H-Buffer and Zone-Swap. H-Buffer is a small host-controlled CMR space that absorbs the SMR updates and migrates those updates back to the SMR space in batches to bring down the SMR update cost. Zone-Swap dynamically swaps "hot" data from the SMR space to the CMR space to further alleviate the SMR update problem. Evaluation results based on MSR-Cambridge traces demonstrate that ZoneAlloy can reduce the average I/O latency and limit the performance degradation of the application I/O during format conversion.

1 Introduction

With ever-increasing demand on storage capacity, Shingled Magnetic Recording (SMR) technology was introduced where disk tracks are overlapped in a shingled fashion to achieve higher areal density on the same disk platter compared to Conventional Magnetic Recording (CMR). However, the increased capacity of SMR comes with the price that updates to these SMR tracks will cause expensive *read-modify-write overhead* (which we also call *SMR update overhead*).

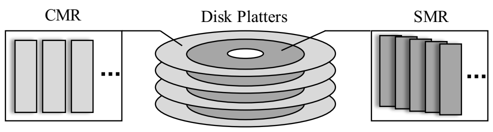

Figure 1: Hybrid SMR Drive

Recently, Google introduced the idea of Hybrid SMR (H-SMR, Fig. 1) drives [1]. A single H-SMR drive can have both CMR and SMR areas, and the format can be changed from one type to the other using the H-SMR APIs [2,3]. H-SMR drives are capable of balancing the IOPS and the capacity potential of the disk platters. Moreover, the configuration of the CMR and SMR areas in H-SMR drives can be adjusted dynamically. Therefore, H-SMR are more flexible than solutions that have a fixed number of CMR/SMR tracks.

H-SMR enables us to better utilize the drives based on the different properties of CMR and SMR. As the H-SMR drive is initially formatted into all CMR [4], an intuitive approach is to store as much data as possible in CMR areas without introducing any SMR update overhead. If the data grows beyond the full CMR capacity, we can gradually convert the disk format from CMR to SMR for larger capacity.

However, there are still many challenges before we can fully utilize these H-SMR drives; e.g., how to arrange the format layout and place data accordingly, how to perform format conversion efficiently, how to reduce SMR update overhead, and how to adapt to dynamic workloads. To the best of our knowledge, there is little investigation on how to deal with these problems in order to use these H-SMR drives efficiently.

In this paper, we propose ZoneAlloy, an elastic data and space management scheme for H-SMR drives to address the aforementioned challenges (Fig. 2). ZoneAlloy hides the H-SMR details and presents upper layer applications with an *elastic data space*, i.e., an address space with extendable size. When data is stored in SMR format, ZoneAlloy can also alleviate the SMR update overhead under dynamic I/O patterns.

To support the extendable address space, ZoneAlloy has a *zone-mapping* that translates the address from the elastic

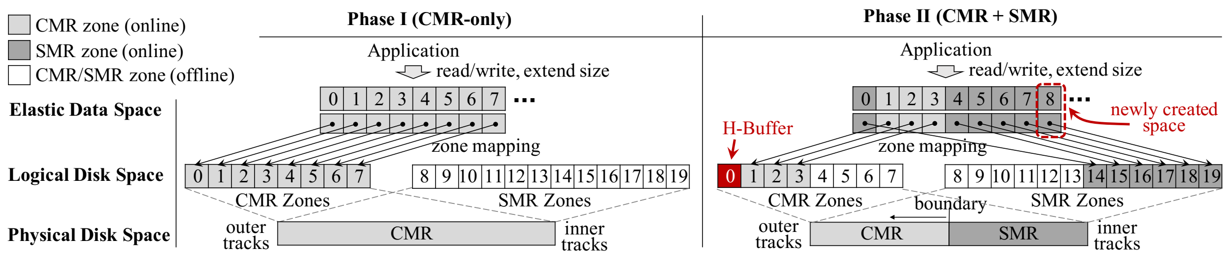

Figure 2: ZoneAlloy Overview: Zone Mapping and Two-phase Allocation.

data space to the disk space. There is also a *two-phase elastic allocation* scheme to fulfill the space extension request by initially allocating CMR space and then converting the disk format from CMR to SMR as necessary (§3.1). Because the conversion operation is expensive for the application due to the intensive *valid data migration*, we propose a *quantized migration* mechanism (§3.2) in ZoneAlloy that is able to control the impact of the conversion operation on the application I/O workload.

In addition, to mitigate the SMR update overhead, we propose *H-Buffer* and *Zone-Swap* in ZoneAlloy. H-Buffer (Host-controlled Buffer, §3.3) is a small CMR space that can buffer the SMR updates and migrate the buffered data back to the SMR space in batches to reduce the SMR update overhead. Zone-Swap (§3.4) can identify and exchange "hot" data in SMR space with "cold" data in CMR space to further alleviate the SMR read-modify-write overhead.

Evaluations using MSR-Cambridge traces [5] show that ZoneAlloy outperforms baseline schemes in average latency and can finish the format conversion efficiently with controllable impact on the application I/O performance.

2 Hybrid SMR Preliminaries

In the hybrid SMR API [1–4,6,7], the same physical media can be formatted into either CMR zones or SMR zones, where a zone is a consecutive LBA space with a size of 256MiB. Each SMR zone has a write pointer indicating where the next write should go to enforce sequential write. Following the most popular Host-Managed SMR model [8,9], the host cannot directly write to the addresses other than the write pointer's location. Therefore, for SMR updates which are not targeted to the write pointer's location, existing data in the SMR zone needs to be read out, combined with the updates, and written back again, causing serious *SMR update overhead*.

Logical disk space of an Hybrid SMR drive is partitioned into the CMR domain (lower address space) and the SMR domain (higher address space) (Fig. 2) [6,7]. According to [2], the host can convert any number of consecutive zones from one format to the other with the H-SMR APIs, which takes an extent of zones from one domain offline and brings the corresponding zones in the other domain online. Here "online" means the zone is backed with the *physical disk space*, and vice versa. Due to the different areal density of CMR and SMR, the number of zones will usually be different from that before the conversion. For example, supposing a hypothetical SMR to CMR areal density ratio of 1.5:1, 2GiB of CMR space (8 zones) could be typically converted into 3GiB of SMR space (12 zones). Note that these APIs will destroy the stored data in the zones being converted, thus the host is responsible for protecting the valid data in these zones.

We define *format conversion*, or *conversion*, as the procedure where the host changes some disk space format from CMR to SMR for more capacity. A conversion comprises of two operations: *valid data migration* and *H-SMR API invocation*. For example, when converting 2GiB of CMR space, valid data will first be read from the CMR zones (∼8 sec, assuming a sequential throughput of 250MB/s), then the space will be changed to SMR format using the H-SMR API (50 ms [1]), resulting in 3GiB of SMR zones. Then the 2GiB of valid data will be written to the newly created SMR zones (∼8 sec), leaving 1GiB of empty SMR space. Note that such conversion is time consuming and the valid data migration dominates the conversion overhead.

3 ZoneAlloy Design

3.1 Elastic Space Mapping and Allocation

Overview and Zone Mapping (Fig. 2). The elastic data space is divided into zones (256MiB). Each zone in the elastic data space is mapped to an online CMR or SMR zone in the drive's logical disk space (§2).

To reduce the SMR update overhead, in ZoneAlloy, we propose to use a small portion of the CMR space as a write buffer, namely H-Buffer (Host-controlled Buffer §3.3), with a size proportional to the SMR partition.

Two-phase Allocation. Disk space is allocated in two phases. In the first phase (Fig. 2 left), the disk is initially all in CMR format [4]. ZoneAlloy allocates new space for the application from outer tracks to inner tracks, because the outer tracks are more performant due to the zone-bit encoding and the short-stroking effect.

In the second phase (Fig. 2 right), the disk format needs to be gradually converted from CMR to SMR when the application needs more capacity (§2). ZoneAlloy performs conversion from inner tracks to outer tracks; i.e., the innermost area is first to convert from CMR to SMR, similar to the pro-

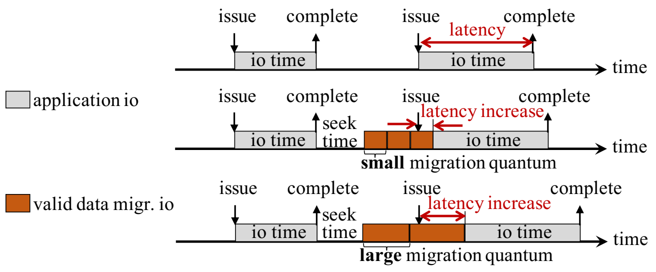

Figure 3: Impact of the size of migration quantum on the application performance. The larger the migration quantum, the greater the increase of latency the application experiences, but the sooner the valid data migration completes.

cedure proposed in [1]. The H-Buffer is located in the outer tracks for performance and will expand proportionally as the SMR partition grows. In each conversion, the zone mapping is updated accordingly and persisted to ensure a linear elastic data space with no LBA gaps.

3.2 Quantized Migration

Conversion is time consuming and intrusive to the application I/Os. For example, to make free space for a requested size quickly, a simple blocking conversion stops all the application I/Os until the requested space has been created. Consequently, the application I/O will be delayed by seconds, or even minutes, depending on the size of the requested space (§2). This is not tolerable for some latency-sensitive applications.

There is an intrinsic trade-off between the performance of the application I/O and the conversion efficiency. Theoretically, the valid data migration job can be broken down into small piecemeal migrations and opportunistically inserted into the time intervals between adjacent application I/Os. In this case, the application I/O will experience little performance degradation. However, it will take a longer time for the conversion to finish, especially when the workload is intense without much time between adjacent I/Os.

Based on this observation, we propose a *quantized migration* mechanism in ZoneAlloy that helps the application find a favorable spot in the trade-off space. We define the *migration quantum* as the smallest-size unit of data to migrate that will carry to completion even after the application I/O arrives (Fig. 3). A small migration quantum has less impact on the application, since the migration can be interrupted at a finer granularity causing less increase of latency to the application I/O. In contrast, a large migration quantum gives priority to the migration I/O, so the conversion will finish sooner, but the application I/O will have a greater latency increase.

The key insight here is that the maximum latency increase is proportional to the size of a migration quantum; therefore, adjusting the size of the migration quantum provides a mechanism that helps the application create free space as quickly as possible while providing a controllable performance degradation. The application can specify the value of the *acceptable increase of latency*. Then ZoneAlloy will determine a proper

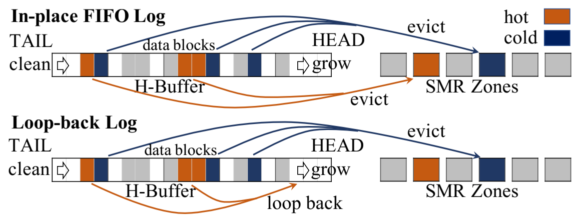

Figure 4: H-Buffer Policies: In-place FIFO log allocates redirected SMR blocks to the head and cleans from the tail by evicting every block belonging to the same zone as the tail block; Loop-back log distinguishes hot and cold SMR zones and only evicts data of cold zones out of H-Buffer while keeping hot data in H-Buffer by "re-queuing" it to the head of the log (loop back).

migration quantum size based on the disk parameters to ensure the increase of latency is bounded by that value. To ensure reliability during the migration, a persistent spare copy of the first two CMR zones of a migration quantum is created (e.g. in reserved SMR zones or non-volatile memory), then the two CMR zones are converted into three SMR zones that can store data copied from the two CMR zones to be converted next, so on and so forth. The data of the first two zones will be copied to the SMR zones at last.

3.3 H-Buffer Management Policies

Updating an SMR zone directly using read-modify-write introduces significant performance overhead. Therefore, we propose to redirect the data into a small CMR buffer space (Host-controlled Buffer, or H-Buffer) to accumulate multiple updates and then migrate the data back to the SMR zone in a batch to amortize the read-modify-write overhead. Note that if an SMR write targets the write pointer, it will be directly issued to the SMR zone without entering the H-Buffer.

Improved block-based LRU. A first intuition is to leverage existing caching policies designed for main memory (e.g., LRU) to "cache" frequently updated data in the H-Buffer. As a result, subsequent updates to those data will happen in the CMR zones without introducing further SMR update overhead. However, directly adopting those caching policies results in poor performance because cache eviction happens frequently at the block granularity and each eviction reclaims one block of free space but triggers one expensive zone read-modify-write. Therefore, we design an *improved block-based LRU* policy that evicts the whole zone containing the victim block out of the H-Buffer instead of only evicting the victim block. Here *evicting a zone* means evicting every block from the H-Buffer that belongs to that zone. In this case, we pay the same price to read-modify-write one zone but reclaim more space, thus the eviction happens less frequently. However, this improved block LRU policy will fragment the free space of the H-Buffer, causing random I/Os which are not friendly to disk drives.

In-place FIFO Log. To avoid fragmenting the free space, we design an *in-place FIFO log* policy (upper figure in Fig. 4) that organizes the redirected data in a log structure. Redirected

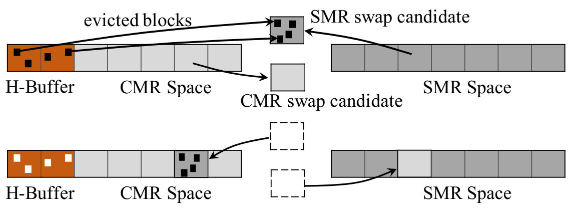

Figure 5: Zone-Swap: CMR and SMR swap candidate zones are read out, combined with the updated blocks evicted from H-Buffer, and written back to each other's location.

data, if already buffered in the log, will be in-place updated. Otherwise, it will be allocated to the log head pointer. Free space is reclaimed from the log tail pointer by *log cleaning*, which evicts the zone containing the tail block (the block at the tail pointer) and advances the tail pointer. Note that if the tail block has already been evicted, the tail pointer will be advanced without any eviction. However, the drawback of an in-place FIFO log is that it strictly follows the FIFO ordering and does not distinguish different types of zones. We observe that frequently updated data blocks will come back to the H-Buffer soon after they were evicted and take up the space that was just been freed. In this case, the time consumed to read-modify-write the original SMR zone is wasted. Therefore, it is beneficial to distinguish different zones when making eviction decisions.

Loop-back Log. To improve from the in-place FIFO policy, we propose a *loop-back log* policy for the H-Buffer that identifies hot zones and "re-queues" the data of hot zones from the log tail to the log head without evicting them (called "loop back," see bottom figure in Fig. 4). Here we first define an *epoch* as the time for the log head pointer to go through a full cycle and come back to the beginning of the H-Buffer. Then a hot zone is defined as a zone that has all its buffered blocks in the current epoch written again in the next epoch. In this case, evicting this hot zone pays the price of one read-modify-write but gains little space back, so it is better to keep it in the H-Buffer instead of evicting it. Although alternatives exist, we find that block LRU is simple yet effective in predicting hot zones where zones that have the blocks only appearing in the MRU half are considered hot.

3.4 Zone-Swap Scheme

Ideally, "hot" data is favorably allocated to CMR zones, so the H-Buffer cleaning cost will be less and the overall performance is better. However, this is not always the case. For example, after conversion, empty SMR space is created to store new data. However, new data may turn out to be "hot" and causes significant SMR update overhead. Similarly, data stored in the CMR space near the CMR/SMR boundary will be migrated to the SMR space during conversion, and it may also be "hot." Further, the I/O pattern keeps evolving, and existing data in the SMR space may become "hot" too.

To address this, we propose the Zone-Swap mechanism in ZoneAlloy that can dynamically exchange data between SMR and CMR zones to concentrate "hot" data in CMR zones. Although the idea of Zone-Swap is simple, finding good zone candidates to swap is not a trivial job, because swapping is an expensive operation that involves extra effort to read and write the zones. If it is not designed carefully, the overhead of swapping can possibly negate its benefit and bring the overall performance down.

We found that the hot/cold zone classification of the swapping decision is different from that in the H-Buffer eviction design. The *space occupancy* of a zone in the H-Buffer matters more in reducing the H-Buffer cleaning overhead than the *block access recency* as used in the H-Buffer. This is because a zone taking up more H-Buffer space, no matter if it is evicted or looped back in the H-Buffer, reduces the "equivalent" H-Buffer size, and therefore the H-Buffer fills up sooner and log cleaning occurs more frequently.

Therefore, Zone-Swap evaluates all zones in the H-Buffer at the beginning of each epoch and labels the zones with an occupancy above a threshold as "SMR swap candidates." Such SMR zones will be evicted when encountered by the log tail pointer in log cleaning. CMR zones that were not updated during last epoch become "CMR swap candidates." When an SMR swap candidate zone is evicted, it will be paired up with a CMR swap candidate if available. Then the data in the CMR and SMR candidate zones will be read and written back to each other's location after combining the updated data evicted from the H-Buffer (Fig. 5).

4 Evaluation

4.1 Experiment Setup

As there are no H-SMR products available, we built an H-SMR simulator with equations extracted from DiskSim [10]. Microsoft Research (MSR) Cambridge traces [5] are used for evaluation. The disk's full capacity is set to 1TB to match the LBA ranges of the traces. The SMR to CMR areal density ratio is set to 1.5:1.

4.2 Overall Performance

In this section, we evaluate the overall performance of ZoneAlloy (H-Buffer using loop-back log policy with Zone-Swap enabled, denoted by `zone-alloy`) against three H-Buffer baselines without Zone-Swap: H-Buffer using improved block-based LRU policy (`improved-lru`), H-Buffer using in-place FIFO policy (`fifo`), and H-Buffer using loop-back log policy (`loop-back`). Only the traces with a write footprint greater than 10GB are used, because ones with a smaller write footprint do not fill up the H-Buffer and will not show any difference in performance. To stress test the system, the H-Buffer to SMR partition size ratio is set to as low as 0.02%, and the disk usage is set to 99.9% so that the majority of the disk is in SMR format. Average latency is measured and plotted in Fig. 6.

As seen in the figure, in-place FIFO reduces the average latency from the improved block-based LRU by 1.4×

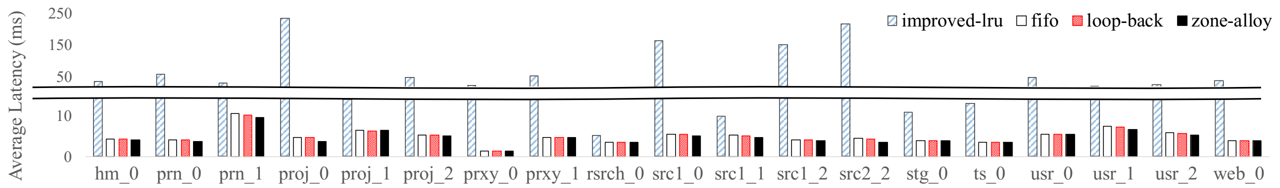

Figure 6: Overall performance of ZoneAlloy compared with three H-Buffer-only baselines. Note that the break of axis in the middle.

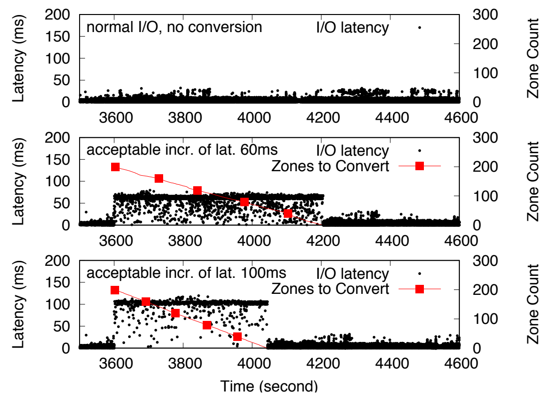

Figure 7: Evaluating the quantized migration mechanism. Larger acceptable increase of latency leads to shorter conversion time.

(rsrch_0, write footprint 10GB) to 45× (proj_0 write footprint 144GB). The difference in performance improvement is due to the different write footprint: the larger the write footprint, the more data is redirected to H-Buffer, and the greater the performance gap will be. Comparing loop-back log and the in-place FIFO log, loop-back log performs similarly or better in all the 19 traces (max improvement at prn_1, a 4.4% reduction of average latency with 7% decrease in the zone read-modify-writes). Zone-Swap further improves from the loop-back log by up to 29% at proj_0, where the count of zone read-modify-writes is reduced from 2989 to 507 at a cost of only 55 zone-swaps. We notice only one case where Zone-Swap makes the performance worse (trace proj_1). This is because 89% of the operations in this trace are read, and swapping causes the logical adjacent data to be far apart, introducing large seek latency. Swapping algorithms considering the seek distance are left as future work.

4.3 Conversion Scheme Evaluation

In this section we evaluate the effectiveness of the quantized migration using a real-world workload (proj_1). In Fig. 7, the latency of every request (left y-axis) is depicted as a black dot. The top figure shows the normal I/O performance without a space extension request (hence no conversion happening), and the middle and bottom figures show the cases where the application is requesting 25GiB of new space (200 zones to convert) with acceptable increase of latency set to 60 ms and 100 ms, respectively. We also plot the number of remaining zones to convert (red line, right y-axis) to indicate the start and the end of the conversion.

The results show that ZoneAlloy is able to perform the conversion efficiently while controlling the increase of latency within the given bound. With a lower acceptable increase of latency (60 ms, middle figure), ZoneAlloy finishes the conversion in a longer time (601 seconds). In contrast, the conversion time reduces to 432 seconds when the the acceptable increase of latency rises to 100 ms. This is quite efficient given the fact that the blocking conversion also takes ∼400 seconds to convert the same number of zones but blocks the application I/O for over 6 minutes.

5 Related Work

Track-based solutions for SMR such as [11–19] leverage the track/sector overlapping information to help with the data placement designs. The eventual establishment of a standard zoned block device interface [8, 9] enabled investigation of three SMR models; namely, drive-managed (DM), host-managed (HM), and host-aware (HA). Skylight [20, 21] studies the internals of the DM-SMR model, while Wu et. al evaluate and characterize the HA-SMR model [22, 23]. ZEA [24], SMORE [25, 26], and GearDB [27] explore the data management and application design of the HM-SMR model in storage systems. Interlaced Magnetic Recording (IMR) has update issues similar to SMR as investigated in [28–30]. Our work differs from the existing work in that we focus on the unique conversion characteristic and I/O handling issues of the incipient hybrid SMR technology.

6 Conclusion and Future Work

The emerging hybrid SMR drive with a mix of CMR and SMR zones provides the flexibility to convert the disk format on demand. We design and implement ZoneAlloy, an elastic data and space management scheme that hides the H-SMR details and presents an elastic data space to the application. Such elastic data space has an extendable size, which is supported by the two-phase allocation design. ZoneAlloy has a quantized migration mechanism to help applications make a good trade-off to perform conversion efficiently while bounding their performance degradation. ZoneAlloy also leverages the proposed H-Buffer and Zone-Swap to reduce the SMR update overhead. Evaluations show that ZoneAlloy can improve the I/O performance from the baseline schemes and can perform format conversion with bounded performance degradation.

Future work includes investigation into other possible CMR/SMR format layout alternatives and the application-aware design.

7 Discussion Topics

We would like to ask for some feedback about what the current pain points are in the cloud storage industry that can be alleviated by the H-SMR drives. What are the new application scenarios that can be supported by such drives? What is the typical way to expand storage? What is the current solution to handle the trade-off between price, space, and performance and what is enabled by the advent of the Hybrid SMR drives? Does re-purposing a storage system often happen? Is the on-line conversion really necessary/helpful? What are the possible/potential use cases for H-SMR?

What are the other possible abstractions and application models of the Hybrid SMR drive? This paper has provided one possibility, the elastic data space abstraction. Other abstractions/models also exist, for example, there could be a physical space manager to carve out one piece of physical SMR space and let the application decide how to use that physical space. Although we do not assume any knowledge about the application in this paper, involving application awareness is another promising direction. For example, when giving out space to an archival application, we may directly provide SMR space instead of starting from all CMR space. For other applications, maybe a mix of CMR and SMR is more desirable, with a user-defined or system-inferred space ratio between them. What are the other possibilities?

Another important question to ask is which layer in the I/O stack we should use to solve those H-SMR data management issues. There are many alternatives, e.g., disk firmware, block layer, RAID, LVM, file systems, or databases (relational databases, key-value stores, etc.). Managing such drives in a lower layer can hide the hybrid SMR details without requiring the higher layers to be changed. In contrast, making the higher layers hybrid SMR-aware can make better decisions about the management as the higher layer has more information about the workload. However, managing the Hybrid SMR at a higher level limits the impact to a specific application and may not benefit other higher-layer applications.

There are also some open design issues for Hybrid SMR. We have covered some of the possibilities in this paper. However, there is still vast open space for research and investigation. One example is the *Format Layout* problem. What are the alternatives to arrange CMR/SMR partitions? Could an interleaved layout be any better? What are the trade-offs? What are the proper criteria to determine the location and size of the H-Buffer?

Additional challenges may include scalability and heterogeneity. In a large-scale storage system, what are the issues if using tens of thousands of hybrid SMR drives? Do we format them at once or gradually? How do we handle the data migration? In a mixed cluster with high-end and low-end SSDs, non-volatile memory, etc., how do we fit the hybrid SMR drive in to make a good trade-off among price, capacity, and performance?

Acknowledgments

We thank the anonymous reviewers and our shepherd, Bill Bolosky, for their feedback. This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439662, 1525617, 1536447, 1708886, 1763008, and 1812537.

References

[1] Google. Dynamic hybrid-smr: an ocp proposal to improve data center disk drives. https://blog.google/products/google-cloud/dynamic-hybrid-smr-ocp-proposal-improve-data-center-disk-drives/.

[2] Timothy Feldman. Flex overview. http://t13.org/Documents/UploadedDocuments/docs2018/f17156r0-Flex_Overview.pdf.

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

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

[5] 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.

[6] Seagate Technology. New flex dynamic recording method redefines the data center hard drive. https://blog.seagate.com/intelligent/new-flex-dynamic-recording-method-redefines-data-center-hard-drive/.

[7] Western Digital. Dynamic hybrid smr. https://blog.westerndigital.com/dynamic-hybrid-smr/.

[8] INCITS T10 Technical Committee. Information technology - zoned block commands (zbc). 2015.

[9] INCITS T13 Technical Committee. Zoned-device ata command set (zac) working draft.

[10] John S Bucy, Jiri Schindler, Steven W Schlosser, and Gregory R Ganger. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). *Parallel Data Laboratory*, page 26, 2008.

[11] Saurabh Kadekodi, Swapnil Pimpale, and Garth A Gibson. Caveat-scriptor: write anywhere shingled disks. In *Proc. HotStorage'15, Santa Clara, CA, USA.*, 2015.

[12] Ahmed Amer, Darrell DE Long, Ethan L Miller, J-F Paris, and SJT Schwarz. Design issues for a shingled write disk system. In *Proc. MSST'10, Incline Village, NV, USA.*, 2010.

[13] 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.

[14] Yuval Cassuto, Marco AA Sanvido, Cyril Guyot, David R Hall, and Zvonimir Z Bandic. Indirection systems for shingled-recording disk drives. In *MSST'10, Incline Village, NV, USA.*, 2010.

[15] 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.

[16] Chung-I Lin, Dongchul Park, Weiping He, and David HC Du. H-swd: Incorporating hot data identification into shingled write disks. In *20th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS12)*, 2012.

[17] 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 *FAST12, San Jose, CA, USA.*, 2012.

[18] Weiping He and David HC Du. Novel address mappings for shingled write disks. In *Proc. HotStorage'14,Philadelphia, PA, USA.*, 2014.

[19] Weiping He and David HC Du. Smart: An approach to shingled magnetic recording translation. In *15th USENIX Conference on File and Storage Technologies (FAST17)*, 2017.

[20] Abutalib Aghayev and Peter Desnoyers. Skylight–a window on shingled disk operation. In *Proc. FAST'15, Santa Clara, CA, USA.*

[21] Abutalib Aghayev, Mansour Shafaei, and Peter Desnoyers. Skylight – a window on shingled disk operation. *ACM Trans. Storage*, 11(4):16:1–16:28, October 2015.

[22] Fenggang Wu, Ming-Chang Yang, Ziqi Fan, Baoquan Zhang, Xiongzi Ge, and David H.C. Du. Evaluating host aware smr drives. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

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

[24] Adam Manzanares, Noah Watkins, Cyril Guyot, Damien LeMoal, Carlos Maltzahn, and Zvonimr Bandic. Zea, a data management approach for smr. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

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

[26] Peter Macko, Xiongzi Ge, John Haskins Jr, James Kelley, David Slik, Keith A Smith, and Maxim G Smith. Smore: A cold data object store for smr drives (extended version). *arXiv preprint arXiv:1705.09701*, 2017.

[27] Ting Yao, Jiguang Wan, Ping Huang, Yiwen Zhang, Zhiwen Liu, Changsheng Xie, and Xubin He. Geardb: A gc-free key-value store on hm-smr drives with gear compaction. In *17th USENIX Conference on File and Storage Technologies (FAST'19)*, pages 159–171, 2019.

[28] Kaizhong Gao, Wenzhong Zhu, and Edward Gage. Write management for interlaced magnetic recording devices, November 29 2016. US Patent 9,508,362.

[29] Kaizhong Gao, Wenzhong Zhu, and Edward Gage. Interlaced magnetic recording, August 8 2017. US Patent 9,728,206.

[30] Fenggang Wu, Baoquan Zhang, Zhichao Cao, Hao Wen, Bingzhe Li, Jim Diehl, Guohua Wang, and David HC Du. Data management design for interlaced magnetic recording. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.