TrackLace: Data Management for Interlaced Magnetic Recording

Fenggang Wu, Bingzhe Li, Baoquan Zhang, Zhichao Cao, Jim Diehl, Hao Wen, and David H.C. Du, *Fellow, IEEE*

Abstract—Interlaced Magnetic Recording (IMR) is a promising technology which achieves higher data density and lower write amplification (WA) than Shingled Magnetic Recording (SMR). In IMR, top tracks and bottom tracks are interlaced so each bottom track is partially overlapped with two adjacent top tracks. Top tracks can be updated without any WA, but bottom track updates require reading and rewriting of affected valid data on the two neighboring top tracks. There are few published studies discussing WA in IMR drives. We propose *TrackLace* to reduce WA for IMR. TrackLace consists of three techniques: *Z-Alloc* allocates user data to the tracks in alternating directions and spreads unallocated tracks among allocated tracks; *Top-Buffer* opportunistically utilizes unallocated top tracks to buffer bottom track updates; and *Block-Swap* progressively swaps bottom track hot data with top track cold data during high space utilization. To further optimize TrackLace performance, we propose a *virtual frame* design that can keep the relocated block (due to Top-Buffer or Block-Swap) close to its original location and an *adaptive buffering* mechanism that can avoid unnecessary redirections depending on the write locality. Evaluations show that TrackLace can reduce WA by 45 percent and lower average latency by 31percent compared with baseline schemes.

Index Terms—Interlaced magnetic recording, hard disks, data layout, allocation strategies, storage management

1 Introduction

The rapid growth of digital content from cloud, mobile computing, social media, big data, and other emerging applications calls for low cost, but large capacity, storage systems [1]. Energy-assisted technologies such as Heat-Assisted Magnetic Recording (HAMR) [2], [3] and Microwave-Assisted Magnetic Recording (MAMR) [4], [5] enable further growth of the areal data density of hard disk drives. Recently, a promising track layout, namely Interlaced Magnetic Recording (IMR), has been proposed [6], [7] and tested in HAMR systems [8], [9] that shows further increase in the data density. IMR can also be applied to MAMR drives.

In heat-assisted IMR, as shown in Fig. 1, track layout is in an interlaced fashion with alternating bottom tracks (lighter color) and top tracks (darker color). Compared with top tracks, bottom tracks are wider and written with higher laser power. The top (narrower) track is written on top of the boundary of two adjacent bottom (wider) tracks. That is, each bottom track is partially overlapped with two neighboring top tracks. Thus, top tracks can be updated without penalty, but updating a bottom track (*bottom updates*) may require reading and rewriting the two affected top tracks (*write amplification*). If *in-place updates* are used (meaning data is written to its native location with reading/rewriting of neighboring top tracks when necessary), in the worst case, an update to data in a bottom track may require two reads and three writes. Such extra reads and writes on the top tracks seriously affect the I/O performance and hence limit the practical use cases of IMR drives. Here we define *write amplification* (WA) as the amount of actual writes to the disk, including these extra writes, compared with the originally requested amount of writes. For example, in one real-world workload we tested, using IMR causes a WA of 2.6× and a 4× latency increase compared with the CMR case. Note that if we can reduce write amplification, the extra reads to the top tracks will also be reduced. Thus, the focus of this paper is to answer: *how can we reduce write amplification and improve the performance of IMR drives*?

To answer this question, we first make the key observation that if the top tracks do not contain any valid data, no rewrites are required when bottom tracks are updated. We define a bottom track with both the overlapping tracks containing no valid data as a *free* bottom track as it can be updated free of amplification. When the space utilization is low, user data can be stored in bottom tracks with top tracks left open. In this case, all bottom tracks are free, thus no updates will have write amplification. As the capacity utilization grows, data starts to be allocated on top tracks, bottom tracks become non-free, and some of the update operations will begin to experience write amplification. Therefore, the performance of IMR depends on its space utilization and how data is placed in its track layout.

Then, based on this observation, to handle this write amplification problem of IMR disk drives, we propose TrackLace, a data management design for IMR that is comprised of three key techniques: *zigzag allocation* (Z-Alloc), *top track buffering* (Top-Buffer), and *block swapping* (Block-Swap), as well as two optimizations: *virtual frames* and *adaptive buffering*. In TrackLace, the memory usage is capped by

• *The authors are with the Department of Computer Science, University of Minnesota, Twin Cities, Minneapolis, MN 55455. E-mail: {wuxx0835, lixx1743, zhan4281, caoxx380, jdiehl, wenxx159, du}@umn.edu.*

Manuscript received 4 July 2019; revised 2 Jan. 2020; accepted 5 Apr. 2020. Date of publication 20 Apr. 2020; date of current version 10 Feb. 2021. Recommended for acceptance by A. Louri EIC. Digital Object Identifier no. 10.1109/TC.2020.2988257

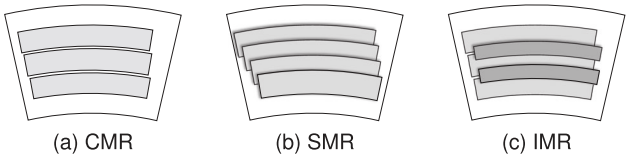


Fig. 1. Track layout for CMR, SMR, and IMR.

a pre-defined size. This enables us to adjust and implement TrackLace for use in different layers in the I/O stack, e.g., the host system, storage controller, or inside the drive, depending on the available memory space.

Z-Alloc numbers (i.e., assigns addresses) and allocates disk space in a zig-zag pattern based on phases of space usage of the IMR drive. Top-Buffer opportunistically takes advantage of unallocated top tracks (top tracks that have not yet been allocated for user data) and uses them to buffer updates to the non-free bottom tracks. When the space usage is high and there are fewer available unallocated top tracks to be used as the Top-Buffer, Block-Swap can progressively swap hot (frequently updated) bottom track blocks with cold (infrequently updated) top track blocks to reduce the update overhead.

Finally, to further improve the performance of TrackLace, we propose two optimizations: *virtual frames* and *adaptive buffering*. To reduce seek time, a *virtual frame* sets the maximum number of tracks that data blocks can be relocated from their original track due to buffering or swapping. The *adaptive buffering* scheme adapts Top-Buffer to the write hit ratio of the workload. If the hit ratio is near zero, buffering and writing back only introduce data redirection overhead without bringing any benefits of accumulating bottom track updates. In this case, the scheme will temporarily "close" the Top-Buffer to avoid the unnecessary Top-Buffer redirection.

We implement TrackLace in an IMR simulator that we built and validated against DiskSim [10]. Evaluation results show that TrackLace is able to reduce the average latency by 31 percent and reduce write amplification by 45 percent compared to the baseline in a high space utilization scenario.

2 Background and Assumptions

2.1 Background

CMR and SMR. There are two common and established track layouts for magnetic recording technologies: Conventional Magnetic Recording (CMR) and Shingled Magnetic Recording (SMR). In CMR (Fig. 1), tracks do not overlap with each other, thus they can be written in any order and updated in place without a write amplification penalty. CMR is the oldest and most common track layout method for hard drives. Demands for increased capacity spurred the development of SMR as a newer and higher-density track layout method. In SMR (Fig. 1), each track is intentionally overlapped on one side by a subsequent track like the layering of shingles on a roof [11], [12], [13]. This process allows a wide write head and a narrower read head to be used to obtain higher density than CMR. SMR drives are typically written in the shingled direction and suffer from expensive write amplification when an earlier track needs to be updated. While there is ongoing research into methods that attempt to mask or delay the poor performance of SMR in update-intensive workloads (see Section 7), production SMR drives are typically reserved for backup, archive, or other applications where updates are rare once data is written.

IMR. A novel third track layout method, IMR (Fig. 1) [6], [7], is currently under development. HAMR-based testing shows that IMR achieves much higher areal density than CMR and slightly higher density, but with significantly fewer update constraints, than SMR [6], [8], [14]. In IMR technology, there are two types of tracks: bottom tracks and top tracks. Ideally, bottom tracks are written first, and then top tracks are written afterwards between two bottom tracks. Bottom tracks are partially overlapped with the two neighboring top tracks leaving a narrower center part of the track that can still be read. Note that "bottom" and "top" here is only conceptual and does not imply physical layers [8], [15]. IMR works well with HAMR where wide bottom tracks are made by writing with higher laser power, and narrow top tracks are written with lower laser power [8], [9]. In such heat-assisted IMR, the linear data density in the bottom tracks is greater than that of the top tracks due to the increased laser intensity. As a result, a bottom track will have higher capacity than a neighboring top track. According to Granz *et al.* [8], the average linear density of the bottom tracks is about 27 percent higher than that of the top tracks.

Writing directly to a bottom track will destroy the data blocks already written on neighboring top tracks. To avoid this data loss, the data on the affected top tracks must first be read before the bottom track is updated. After the bottom update, the saved top track data is written back to its original top track locations. This process causes write amplification. Writing to top tracks, however, does not have any penalty.

Seagate Baseline. A three-phase data allocation scheme is proposed by Seagate [7], [16] which numbers and allocates disk space based on three phases of space usage. In the first phase, if the usage is less than the total capacity of the bottom tracks ($0 \sim 56$ percent space usage), all the data is assigned to the bottom tracks sequentially. In the second phase, space will be allocated from every other top track until half of the total top track capacity is used ($56 \sim 78$ percent space usage). In the third phase, the remaining top tracks will be used ($78 \sim 100$ percent space usage). There is no penalty when in-place updating a bottom track during the first phase because no top tracks contain valid data. During the second utilization phase, in-place updates to a bottom track will require one rewrite in one of the adjacent top tracks (or no penalty if neither of the two neighboring top tracks has affected valid data). Similarly, a bottom update in the third phase will require one or two top track rewrites.

2.2 Assumptions

We assume the space manager (e.g., file system, logical volume manager, storage controller, etc.) allocates an interlaced set of consecutive physical top and bottom tracks, named a *Track Group* or *TG* (Fig. 2), to the applications. An IMR disk can have one or more TGs. In our high utilization experiments, we mean that one TG is, for example, 99 percent full, not necessarily the entire disk. How the space manager allocates the space is beyond the scope of this paper since we only focus on the data allocation and management within one TG. We also assume all the allocated space of a TG is

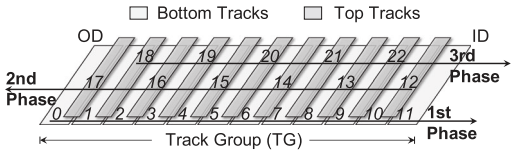


Fig. 2. Zigzag Allocation. Numbers show the addressing and allocation order of the tracks. Note that the even-numbered phases (i.e., the second of three phases in this example) have reversed allocation direction for better data locality. An IMR disk may have one or more Track Groups.

valid and must be protected from being overwritten during a bottom track update. Possible use of a TRIM command to mark invalid data that can be safely overwritten is left for future research.

Although there may be different track capacities within one TG in real production disks due to factors such as zone bit recording and sector defects, we assume the top tracks within one TG have identical capacity and the bottom tracks also have a uniform, but higher, capacity. The IMR configurations that we use are based on testing data by Granz *et al.* [8] (Table 1). Our data management principles can be easily adapted to other IMR production settings.

3 TrackLace Design

TrackLace has three core components: zigzag allocation (Z-Alloc, Section 3.1), top track buffering (Top-Buffer, Section 3.2) and block swapping (Block-Swap, Section 3.3). TrackLace adapts to the changes in space utilization by switching gradually from Top-Buffer to Block-Swap as utilization grows. Both Top-Buffer and Block-Swap can mitigate the write amplification of IMR drives.

3.1 Zigzag Allocation (Z-Alloc)

In the existing allocation scheme described by Seagate [7] (see Section 2.1), the data allocation all follows the same radial direction; i.e., all three phases are allocating from outer diameter (OD) tracks to inner diameter (ID) tracks. This pattern may harm data locality by physically separating adjacent data (e.g., the final track of the first phase and the first track of the second phase). To improve this scheme, we propose our *Zigzag Allocation* or *Z-Alloc* design which reverses the allocation direction of even-numbered phases (making it inner tracks to outer tracks) to preserve data locality between phases (Fig. 2). Block I/O typically exhibits spatial locality, and applications expect logically adjacent data blocks to be physically close to each other. Thus, our improved layout should reduce seek time for data written across phase boundaries.

As we will explain in Section 4.1, there are cases when we wish to extend the addressing and allocation to more than three phases. This same layout pattern can be applied to more than three phases, but we will continue to show three phases in the illustrations to ease visualization of the designs.

TABLE 1
IMR Disk Configuration

Basic Parameters		Derived Parameters	
Median track pitch	820 KTPI	#tracks (N)	1045800
Median top track density	1640 KBPI	Avg. bot. track size	2 MB
Median bot. track density	2030 KBPI	Avg. top track size	1.6 MB
RPM	5400		

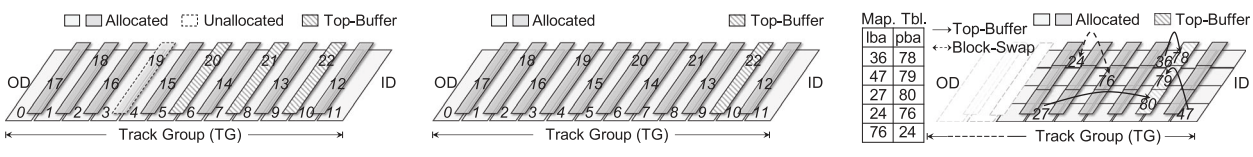
3.2 Top Track Buffering (Top-Buffer)

As top tracks can be updated freely, we propose to buffer *bottom updates* (update operations targeted to bottom tracks) to unallocated top tracks. Then such buffered blocks can accumulate multiple updates before they are migrated back to their original bottom tracks. This ability to amortize and reduce write amplification motivates our design of Top Track Buffering (Top-Buffer).

Fig. 3a shows the design of Top-Buffer in which the highest-numbered (meaning highest addressed, or the last to be allocated, e.g., track 20, 21, and 22 in Fig. 3a) unallocated top tracks are organized as the *Top-Buffer region* (or *Top-Buffer* in short). Top-Buffer utilizes these unallocated top tracks as buffers for updates to non-free bottom tracks when the TG space usage is less than 100 percent.

When the Top-Buffer is full, we evict buffered blocks to reclaim space using a Sequential Cleaning Policy (SCP). SCP cleans a whole Top-Buffer track at a time by sequentially reading the buffered blocks on the target track (*victim blocks*) and writing them back to their original bottom track locations. The track to be cleaned is selected in a round-robin fashion. Compared to other types of data reclamation policies like Least Frequently Used (LFU) or Least Recently Used (LRU), SCP is able to reclaim a continuous space so that the Top-Buffer region will never be fragmented. Thus, further write requests redirected to the Top-Buffer will not be fragmented either. In this way, SCP can reduce the fragmentation of future read/write requests to the buffered data and avoid extra disk seeks caused by fragmented I/O that would harm the overall performance. Also, SCP has less metadata overhead compared with LRU (see Section 5.1 for more discussion). However, some frequently updated data blocks may be cleaned early with SCP thus slightly harming performance. This is addressed by Block-Swap (Section 3.3).

In our design, the LBA to PBA mapping entries for Top-Buffer are fully loaded to the memory when system starts and are persisted in the IMR drive each time updated. We limit the size of the Top-Buffer to be at most a certain portion



(a) Top-Buffer with Max Size = 3 tracks (b) Top-Buffer with two tracks given up (c) Top-Buffer + Block-Swap

Fig. 3. Illustration of Top-Buffer and Block-Swap. Top-Buffer Max Size = 3 tracks in this three-phase example. (a) Even though there is one more unallocated top track available (track 19), only the three highest-numbered unallocated tracks (20, 21, 22) are used as Top-Buffer. (b) Compared to the figure in (a), the TG space utilization has grown, and Top-Buffer gave up two tracks to allocate more user data. (c) Top-Buffer and Block-Swap used together and sharing the mapping table. Bottom tracks have four sequentially numbered blocks and top tracks have three. Three bottom track blocks (36, 47, 27) are buffered in Top-Buffer, and one top-bottom pair has been swapped (24 and 76).

($B_{max}\%$) of the size of the TG, even if there are more unallocated top tracks available. This buffer size limit reduces the maximum size of the mapping table and allows it to be kept in memory (within a *memory table size budget* for fast searching. Besides, we have two ways to reduce the I/O overhead of reading/writing the mapping table entry. 1) For read operations, we cache all the mapping entries in memory to eliminate extra disk reads for the index. 2) For write operations, we put the mapping table entries for each Top-Buffer track at the end of the track (using 1.6 KB per track). By locating the buffered data and the corresponding mapping entries in the same track, the seek time for making mapping entries persistent is minimized during write operations.

In our current implementation, we set the default $B_{max}\% = 2\%$, which only needs a memory table budget of 0.008 percent of the TG size (assuming 4 KB sectors and 64-bit addressing) but can still capture a good amount of trace locality. Supposing a future IMR drive has a capacity of 20 TB, the mapping table size will be roughly 1.6 GB (based on 64-bit addressing, but this number will be roughly 25 percent smaller if 48-bit addresses can be used). However, the value of $B_{max}\%$ in a real environment is adjustable depending on the available memory space.

In Fig. 3a, the Top-Buffer is located at the highest numbered top tracks (however, we have an optimization discussed in Section 4.1 to enable bottom data to be buffered in nearby top tracks that are not necessarily the highest numbered ones). When the TG utilization increases and the user data tracks reach the Top-Buffer tracks, Top-Buffer will clean using SCP and give up tracks for user data (as shown in Fig. 3b). As TG space usage further increases, the Top-Buffer size keeps decreasing. An extremely small Top-Buffer will have very limited benefits or even worse performance than the three-phase Z-Alloc scheme with no Top-Buffer. If the Top-Buffer is too small, buffered data is evicted too soon to bring any benefit from accumulating bottom updates but still introduces redirection overhead. We propose our next technique, Block-Swap, to supplement Top-Buffer by helping avoid this increased overhead as Top-Buffer shrinks and reduce write amplification at very high space utilization.

3.3 Block Swapping (Block-Swap)

As mentioned above, the benefits of Top-Buffer diminish as the TG utilization grows close to 100 percent and Top-Buffer gives tracks back to user data. To improve performance when the size of the Top-Buffer decreases, we propose block swapping (Block-Swap) which progressively swaps *hot* (frequently updated) bottom track blocks with *cold* (infrequently updated) top track blocks. There are three benefits of Block-Swap. 1) Unlike Top-Buffer, Block-Swap is not limited by the availability of unallocated top tracks, so it is able to reduce the write amplification when the utilization is too high for Top-Buffer to perform effectively. 2) Block-Swap can aid Top-Buffer when used together in that the data on bottom blocks that is very hot can be swapped to top tracks, so fewer updates will go to the buffer and the cleaning cost is reduced. 3) Hot blocks identified and selected to be swapped by Block-Swap will stay in top tracks longer without being evicted prematurely from the Top-Buffer by the indiscriminate SCP. More details of this design are presented in the remainder of this subsection.

Block-Swap and Top-Buffer will be used simultaneously during very high space utilization. In such situations, Block-Swap occurs during SCP eviction of the Top-Buffer. Hot bottom blocks chosen from the victim blocks evicted by SCP are swapped with cold top blocks selected from the TG. In our design, a block evicted by SCP will not be swapped with a top track block if the evicted block was last written before all the currently swapped hot blocks were last written. Compared to Top-Buffer, Block-Swap has a higher overhead, which requires eight I/Os (one read and one write each for the top block, the bottom block, and its two neighboring top tracks). In this case, two block hits on the hot block swapped to the top track can offset the overhead, because one bottom update introduces four extra I/Os on the two neighboring top tracks when they both contain valid data. If only one neighboring top track has valid data, four block hits are needed to offset the Block-Swap overhead.

During the swapping, a hot bottom block and cold top block pair will be read and then written to each other's location using in-place update. Two new mapping entries will be created to record the physical locations of the two blocks. It is unrealistic to construct an all-to-all block level mapping for the TG. We make a design decision that the total size of the Block-Swap mapping table combined with the Top-Buffer mapping table will be bounded by a memory table budget size so that it can be cached in the memory (see Section 3.2).

Block-Swap consumes more mapping table entries (two entries) when relocating a hot block to a top track compared with Top-Buffer (one entry). For better mapping table efficiency, we favor Top-Buffer over Block-Swap when consuming the mapping table budget. Therefore, when the unallocated space is greater than $B_{max}\%$ of the TG, Top-Buffer takes all of the mapping table budget and Block-Swap is not used. Block-Swap will start when TG space usage goes beyond $(100 - B_{max})\%$, which is when Top-Buffer starts to give up tracks to user data and cannot make full use of the mapping table size budget. Fig. 3c shows a simplified example where each of the the bottom tracks has four sequentially numbered blocks and the top tracks have three (recall that bottom tracks are roughly 27 percent more dense than top tracks). As shown, Top-Buffer and Block-Swap share the mapping table and its memory budget. When usage is 100 percent, the Top-Buffer is completely disabled and only Block-Swap is operational. Data will be directly written using in-place updates. Here, as there is no space for Top-Buffer, Block-Swap will maintain a virtual Top-Buffer that only collects update statistics to aid swapping decisions.

To keep the mapping simple, we only select blocks to swap that have not been swapped before; i.e., a block that has already been swapped once will not be chosen to swap with another block. If a previously swapped pair of blocks both become hot, they will first be unswapped, and then the hot bottom one will be swapped with a different cold top block.

The Block-Swap mapping entry is updated and synchronized to the disk after the swapping of the two data blocks is completed. In our design, the persistent locations for the Block-Swap mapping entries are distributed into the TG at the end of some of the top tracks. Similar to Top-Buffer mapping entries, the Block-Swap mapping entries are

loaded to memory at system start time. If the Top-Buffer size limit is set to be 2 percent of the TG, using the information in Table 1 and assuming a block size of 4 KB, we can calculate that at most one top block is needed for every 24 tracks (12 top and 12 bottom). Once this "meta-block" that stores the mapping entries for a set of tracks is full, the top data blocks in the corresponding 12 top tracks will not be selected as cold blocks for swapping.

One open issue in our current design is that writes that trigger SCP have higher I/O latency, especially when the TG is nearly full and the expensive Block-Swap occurs during SCP. How to bound this tail latency is left for future work.

4 Optimizations

4.1 Virtual Frames

Top-Buffer and Block-Swap help reduce write amplification by relocating hot bottom data to top tracks. However, such data relocation breaks the physical locality of logically adjacent data; i.e., logically nearby data blocks may be physically far away from one another, possibly across the whole TG, if only some of them are buffered. This causes long seek time and affects the performance, which motivates our virtual frame design as a way to limit the seek distance in Top-Buffer and Block-Swap.

4.1.1 Relocation Distance

Here we define *block displacement* or *relocation distance* as the number of tracks between the original track and the track where a block is relocated. Keeping the relocation distance small will reduce seek distance and improve performance in the following two situations. First, it will reduce seek time during the Top-Buffer write-back operation where we evict buffered data and migrate it back to the original location. The same seek distance reduction advantage is true for Block-Swap and its un-swapping operation. Second, a smaller seek distance is also favorable when reading a group of logically nearby, or clustered, blocks includes tracks near the original physical location and also more distant buffered/swapped locations if some of the logically close blocks being read have been physically relocated by Top-Buffer or Buffer-Swap. Again, it is preferable to limit the seek distance when Top-Buffer and Block-Swap potentially break the physical locality of logically local blocks.

4.1.2 Virtual Frames Explained

A virtual frame is defined as a logical partition of the TG with a user-defined frame *width*, i.e., maximum relocation distance. Top-Buffer or Block-Swap is only allowed to relocate bottom track data within the frame. Virtual frames do not slide or overlap. The whole TG is divided into many equal-width virtual frames, each individually running Top-Buffer and Block-Swap but with a shared mapping size budget the same as defined in Section 3.2. Using such virtual frames can bound the relocation distance for both Top-Buffer and Block-Swap, hence reduce the seek time.

When selecting the width of the virtual frames, there is an intrinsic trade-off between the relocation distance and write amplification. A small virtual frame width has the benefit of reducing the relocation distance and hence reduces the seek time penalty. However, the downside is that a small frame size limits the number of available unallocated top tracks in each virtual frame. For some workloads with concentrated updates in some particular bottom tracks, buffered data may quickly fill the Top-Buffer of the corresponding virtual frame and trigger an expensive SCP even though neighboring virtual frames may have underutilized unallocated top tracks. We found that an intermediate virtual frame width of 32K (K = thousand) tracks works well with most of the tested workloads. Investigation into an adaptive and/or heterogeneous virtual frame width is left for future work.

4.1.3 Multi-phase Allocation

In the initial three-phase Z-Alloc design in Section 3.1, when the utilization goes beyond 78 percent, i.e., enters the third and final phase, virtual frames of the outer tracks will have no Top-Buffer tracks to allocate because the highest numbered tracks are inner tracks within a different set of virtual frames. Consequently, Top-Buffer cannot operate, even if there are still unallocated top tracks far away, and only the more expensive Block-Swap is used in these outer virtual frames when handling the bottom writes. This causes performance inferior to the case when unallocated tracks are available within the virtual frame and Top-Buffer is used together with Block-Swap. This motivates us to distribute the available unallocated top tracks across the TG and into each virtual frame.

Therefore, we design a *multi-phase allocation* scheme that extends the three-phase Z-Alloc design to have any number of phases and can spread unallocated top tracks evenly among the used top tracks during allocation. When at the end of the second phase and starting the third phase, this approach skips every other empty top track during the allocation. In other words, in the second phase, every other top track is allocated for user data, and in the third phase, every fourth track is allocated, leaving an empty track every four top tracks, and so on for the subsequent phases. If the number of phases is too large, the last few phases will have logically adjacent top tracks that are physically separated too far. We found that a default number of six phases, when working with virtual frames, can distribute unallocated top tracks effectively during very high space usage without introducing too much overhead due to the separation between two logically adjacent top tracks.

4.2 Adaptive Buffering

Quantitatively, Top-Buffer's extra overhead for one bottom write is one write to and one read from the Top-Buffer track. The I/O saved during a Top-Buffer write hit is the read and write on the neighboring top track(s) of the target bottom track (saving one read and one write if one neighboring top track has valid data, two reads and two writes if both neighboring tracks have valid data). So, one Top-Buffer hit will offset the extra I/O of the Top-Buffer redirection and migration. On the other hand, if one block has no write hit during its stay in the Top-Buffer, the redirection causes extra I/O but saves nothing.

We define *write hit ratio* as the ratio of write requests, among all bottom track write requests, which find that the data to be

updated is already buffered in the Top-Buffer. In our evaluation some workloads are found to have a very low write hit ratio where most data blocks are updated once but never again (e.g., the `src1_0` write hit ratio is only 1.3 percent). Such data fills up the Top-Buffer quickly and triggers Top-Buffer cleaning (i.e., eviction) without bringing any benefit. In this case, Top-Buffer performance is even worse than the bufferless three-phase Z-Alloc and Seagate approach because data redirection and migration introduce extra I/O.

Our solution is to "open" and "close" the Top-Buffer according to the workload. Such an adaptive buffer scheme allows the write requests to be redirected into the Top-Buffer when the write hit ratio is high and "close" it when the ratio is low. The difficult part is to flag when to "open" or "close" the Top-Buffer. We address this issue using a heuristic where if the Top-Buffer write hit ratio is zero over a period of time, we consider that the workload write locality has become poor and reduce the rate at which write requests enter the Top-Buffer.

Specifically, during a SCP (Top-Buffer Sequential Cleaning Policy, see Section 3.2), if no write hit has happened on the buffered data since the last SCP, we halve the rate at which write requests are allowed into the Top-Buffer. The rate is initialized as 1; i.e., all of the bottom writes will be buffered. From there, an SCP will halve the Top-Buffer admission rate to 1/2 if there has been no hit since the last SCP. This means one out of every two bottom write requests will enter the Top-Buffer. If there has still been no hit when the next SCP comes, the Top-Buffer fill rate is cut in half again to 1/4 of the bottom write requests entering the buffer. There is a lower limit of the buffer fill rate, which we set to be 1/128 by default. If there is ever a write hit, the buffer admission rate is immediately reset to 1. Here we do not really "close" the Top-Buffer completely by setting the entry rate to 0. This gives a chance for hot data to enter the Top-Buffer if the write hit ratio suddenly becomes high. Note that by using a dynamic buffer admission rate we do not alternate between buffering and directly writing back too frequently so as to avoid excessive seek operations.

There are many benefits of this hit ratio adaptive scheme. 1) When the workload enters a low hit ratio phase and causes frequent eviction, e.g., in a sequential write workload where each block is only written once, the admission rate will be quickly reduced to prevent the I/O from entering the Top-Buffer. Buffering such data will not contribute to reducing write amplification and will only add to the redirection and migration cost. 2) When there is a phase change in the workload and the write locality becomes better, the buffer admission rate will recover quickly to 100 percent to allow hot blocks to enter the Top-Buffer. This facilitates more write hits and reduces write amplification. Our evaluation in Section 6.5.2 shows this adaptive buffering works well in reducing unnecessary buffering but is still sensitive enough to detect workloads with good write locality.

5 Practical Considerations

5.1 Reduce Computation and Memory Cost

While we primarily focus on improving the I/O performance by reducing write amplification, CPU overhead and memory consumption are also considered in our design.

We have made the design choice to limit the mapping table size by imposing a strict memory space budget (Section 3.2, Section 3.3). This and other parameters of our design can be adjusted based on available hardware. The remainder of this subsection discusses how Top-Buffer and Block-Swap are designed to lower the CPU and memory footprint.

In Top-Buffer, as previously mentioned, SCP is chosen over other fine-grained cache replacement policies (such as LRU) because it prevents the Top-Buffer space from becoming fragmented (Section 3.2). Additionally, the LRU data structure takes much more memory space than SCP. LRU needs to maintain a separate list element for every buffered data block and to move each element around independently. By contrast, SCP is dealing with the coarser granularity of tracks, and the total metadata space is smaller by three orders of magnitude. Moreover, in LRU each data block update will reorder the LRU list, which triggers a list element lookup and relocation. Such operational overhead is non-trivial as it sits in the write path, and it is even worse when considering that a single request may consist of hundreds of data blocks.

In Block-Swap, when a hot bottom data block (selected by the criteria specified in Section 3.3) is to be swapped onto a top track, finding a cold block to swap is not a simple procedure. It is impractical to keep an access count statistic for all of the physical blocks in each of the top tracks as this amount of metadata would be prohibitively large. To select a cold block to swap, we design a heuristic random selection algorithm. First, we reduce (coarsen) the statistic granularity from the block level to the track level; i.e., instead of selecting a cold *block*, we select a cold *track* and sequentially choose the existing blocks in this cold track as the cold blocks. An extra benefit of selecting a cold track instead of cold block is that it reduces the I/O fragmentation similarly to SCP. Second, even though we use a track-level granularity, the computation cost to find the coldest track is still too high. For example, if we organize all top tracks as a minimum heap, each access to a physical block of a top track will potentially trigger a heap adjustment, and such heap maintenance occurs in the critical I/O path. Our solution is to distribute the tracks into several buckets with disjoint ranges of update counts. Instead of choosing the *coldest track*, we randomly choose one track from the *coldest bucket*. Then data blocks from this track will be selected sequentially for swapping. With this approach, we sacrifice the accuracy of finding the absolute coldest track but reduce the computation overhead by instead returning a random track in the coldest bucket.

5.2 Consistency and Reliability

Both Top-Buffer and Block-Swap use a block-based address mapping table (logical block address to physical block address) to remember the new location of the data. If this mapping is not consistent with the actual data after a system crash, there will be the potential for data loss. The rest of this section describes how our design protects against such a situation.

In Top-Buffer, our algorithm first writes the redirected data to the Top-Buffer track before making the mapping entry persistent. In this way, if the system crashes after the

data is redirected but before the mapping entry is written, there will be no problem since the existing mapping entries as well as the Top-Buffer write pointer (the log head pointer) can be reconstructed from the existing mapping entries stored in the disk. Redirected data without a mapping entry is discard automatically. The mapping entry and the Top-Buffer write pointer are made persistent using one atomic block write.

In Block-Swap, first the algorithm reads the two data blocks to be swapped and calculates checksums for them. Then the mapping entry, block checksums, and the cold track swap pointer are made persistent with one atomically written block I/O. Finally, the data blocks are swapped. If the system crashes with the metadata persisted but the two data blocks not yet swapped, the checksums can detect it and the recovery process will correctly complete the data swap.

There are two more reliability issues regarding the swapping operation. First, during the swapping procedure when one block is already written to the space of the other block, the other block should be well protected against a system crash as we do not have it on disk anymore. Second, the swapping process needs to ensure that the in-place update of the bottom track during the swap does not allow data loss from either of the two adjacent top tracks while waiting for the completion of the bottom update. Here we assume disk vendors have a way to ensure that the data that has been temporarily read to avoid track overwrites will be well protected from untimely crashes until it is rewritten in its correct location. Regular IMR drives without any TrackLace swapping will still have to offer protection from such top track data loss in the middle of the normal bottom track update process. Similarly, in current SMR production drives (the host-aware and drive-managed models, see Section 7), when the drive migrates data from the media cache back to the middle of the original shingled zone, there is a read-modify-write procedure. During this SMR update procedure, the old data of the zone is read first and the zone is rewritten with the new and old data combined without worrying about the zone data that was temporarily read being lost [17], [18], [19], [20]. We believe there is some non-volatile media organized in either a log or a temporary buffer to hold the data to be protected during this process and that a similar concept will be used in IMR drives.

6 Evaluation

6.1 Implementation

We built an IMR simulator called IMRSim with equations extracted from DiskSim [10] and the disk model described by Ruemmler and Wilkes [21]. Using the average top and bottom parameters described by Granz *et al*. [8] (see Table 1), we set IMRSim top and bottom tracks with different linear densities and data rates to simulate IMR track width and laser power differences. We validated our disk model by comparing IMRSim with DiskSim (details provided as supplemental material in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2020.2988257). Note that the overhead needed to make mapping table entries persistent on disk is also implemented in our simulator.

TABLE 2
Basic statistics of the traces (M = million)

trace	# of req.	data written	write ratio	trace	# of req.	data written	write ratio
prn_1	11.2M	31 GB	24.7%	src2_2	1.2M	39 GB	69.7%
proj_1	23.6M	26 GB	10.6%	usr_1	45.3M	56 GB	8.5%
proj_2	29.3M	169 GB	12.4%	usr_2	10.6M	27 GB	18.9%
src1_1	45.7M	30 GB	4.8%				

6.2 Experiment Setup and Data Sets

The test system is a Dell PowerEdge R430 1U Server with two six-core Intel Xeon E5-2620 v3 @ 2.40 GHz processors and 64 GB of DDR3 memory. Ubuntu 18.04.1 LTS with Linux kernel version 4.15.0 is installed on the system.

A representative subset of the Microsoft Research (MSR) Cambridge traces [22] (Table 2) are fed into our IMR simulator and used for evaluation. Unless stated otherwise, the TG size is adjusted based on the byte range (workload range) of each trace. If, for example, we want to test a TG that is 50 percent full, we will set the TG size to be twice as large as the byte range of the trace we are testing. The TG is considered full of valid data up to the utilization percent we are testing and therefore all writes are updates. The default virtual frame size is set to 32K tracks. The maximum Top-Buffer size is set to 2 percent of the total TG capacity, and the mapping table size is set in accordance with this Top-Buffer capacity.

6.3 Comparison with Existing Scheme

6.3.1 Performance under Different Workloads

We first compare the average latency and the write amplification of TrackLace (`track-lace`) and the Seagate baseline (`seagate`) when replaying different workload traces (see Fig. 4). The first subfigure shows the average latency, and the second subfigure shows write amplification where a value of one means no additional writes. Here we test with a high TG utilization of 99 percent to evaluate how TrackLace performs under severe space constraints (however, we vary utilization in other tests in Section 6.3.2).

We can see that in six out of the seven workloads TrackLace outperforms the Seagate baseline by reducing the average latency and bringing down the write amplification. For trace `prn_1`, the average latency is reduced by 31 percent and write amplification is reduced 45 percent. We notice one case (`proj_2`) where TrackLace averages slightly more latency than the Seagate baseline (an increase of 3 percent). We found two reasons for this behavior. 1) The trace has poor write locality (only a 4.8 percent buffer/

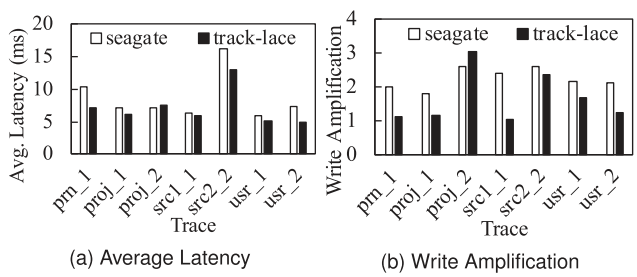


Fig. 4. Evaluation of TrackLace compared with Seagate baseline in different workloads (TG utilization = 99%).

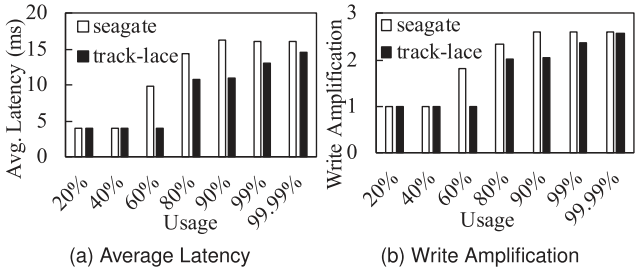


Fig. 5. Evaluation of TrackLace compared with Seagate baseline under different TG utilization (trace src2_2).

swap write hit ratio), so Top-Buffer and Block-Swap will not help much in reducing the write amplification. 2) This particular workload has a large total write size (169 GB) and is concentrated in a small number of virtual frames (three virtual frames experience 76 percent of the bottom writes), causing more write amplification in the "hot" virtual frames. In Section 6.5.1, we have a detailed evaluation and discussion regarding the trade-offs of virtual frames.

6.3.2 Performance under Different Utilization

In this section we compare the performance of TrackLace and the Seagate baseline under different utilizations of the TG, ranging from 20 percent to an extremely high utilization of 99.99 percent. The trace used in this and some of the following tests is src2_2 because it covers a large range of block addresses and is write-dominant, so it can better expose the IMR write amplification. We collect the average latency and the write amplification and plot them in Fig. 5.

We can see from Fig. 5b that at a low utilization (20 and 40 percent), both TrackLace and Seagate baseline have no write amplification (WA = 1). In both schemes, data is not allocated on top tracks until the TG utilization goes beyond 56 percent (the first phase is bottom-only allocation). Thus, there is no need to protect top track data and no write amplification. From Fig. 5a we can see TrackLace and Seagate baseline have very close average latency (less than 0.4 percent difference) while under these lower TG utilizations.

As the TG space usage grows beyond the bottom-only phase (56 percent), the average latency of the Seagate baseline starts to climb. For example, when moving from 40 to 60 percent TG utilization, the Seagate baseline has a 2.43× increase in average latency and an 80 percent increase in write amplification, while TrackLace does not have any noticeable performance loss. From 80 percent to higher utilizations, TrackLace starts to show an increased average latency and write amplification, but the increases are less than those of the Seagate baseline. As the TG space grows nearly full and beyond 98 percent usage, there are fewer available Top-Buffer tracks and Block-Swap starts to take effect (given $B_{max}\% = 2\%$). Although Block-Swap has higher I/O overhead than Top-Buffer, TrackLace still outperforms the baseline, even during extremely high utilization (10.3 percent less average latency and 2.1 percent less write amplification than the baseline at 99.99 percent TG space usage).

6.4 Effect of Design Components

In this section, we take a close look to see the performance impact of each of the design components. We are comparing the following schemes with incrementally more design components: sg is the three-phase Seagate baseline; z-a is the three-phase Z-Alloc; +bf uses Top-Buffer atop z-a; in +ab we apply the adaptive buffering optimization to the Top-Buffer; +sw further adds Block-Swap to the +ab scheme; +vf enables the virtual frame seek distance optimization on top of +sw; finally, our is the complete scheme that includes all the previous components and also changes the number of phases in Z-Alloc from three to six to better spread the unallocated top tracks into more virtual frames as described in Section 4.1.3. We select two representative workloads, src2_2 and prn_1. The average latency and write amplification are plotted on the left and right axis in Fig. 6, respectively. Two sample setups are used: small TG with high usage ($U = 99\%$, top figures), and large TG with intermediate usage ($U = 80\%$, middle and bottom figures).

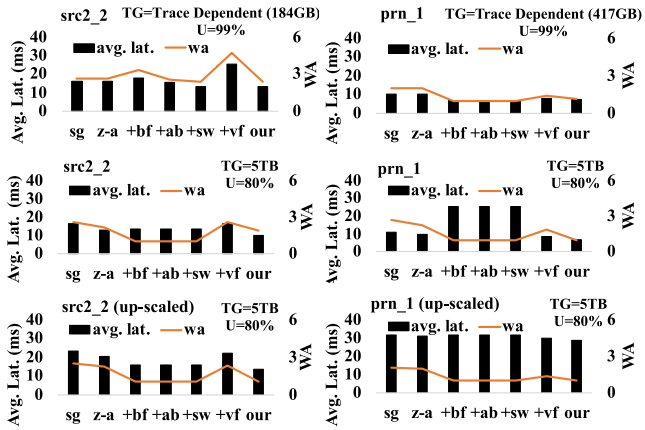


Fig. 6. Breakdown of the effects of each design element. We tested two sample setups: small TG with high usage ($U = 99\%$, top figures), and large TG with intermediate usage ($U = 80\%$, middle and bottom figures).

In the top figures (where TG size is based on the trace's address range as described in Section 6.2), for workload src2_2, we find that changing from the sg scheme to the z-a scheme does not change the performance too much because this trace does not have much localized I/O at the boundaries of different phases. We also observe that adding only Top-Buffer with none of the other components (+bf) performs worse than z-a. This is because src2_2 has poor write locality, with a total write hit ratio as low as 4.9 percent, and buffering a block that is never hit introduces redirection and migration overhead without bringing any benefits. After applying the adaptive buffer scheme (+ab), the performance becomes better than z-a as the adaptive scheme can turn off the Top-Buffer when the write hit ratio of the workload is low (we have a detailed evaluation of the adaptive buffering scheme in Section 6.5.2). Further, on top of an adaptive Top-Buffer, Block-Swap (+sw) brings extra benefit as it fully utilizes the remaining mapping table capacity budget to swap additional hot blocks to top tracks. With 99 percent of the TG space utilized, there is only 1 percent left as unallocated top tracks for the Top-Buffer, leaving the mapping entry budget underutilized if Block-Swap is not used. Moreover, we find that adding virtual frames alone without changing the number of phases (+vf) increases the average latency and the write amplification from +sw. The reason is that in three-phase Z-Alloc, the Top-Buffer tracks are all on the inner diameter side of the

TG (recall Fig. 3a). Other than the innermost virtual frames, most of the virtual frames have no available unallocated top tracks within range to be used as Top-Buffer. Therefore, even though Block-Swap is used to deal with the bottom writes in the virtual frames without Top-Buffer, the performance is still inferior to the case when unallocated tracks are available and the less expensive Top-Buffer scheme can be used together with Block-Swap (Section 4.1.3). Finally, changing the number of phases from three to six (our) successfully distributes unallocated top tracks among virtual frames and thus achieves a lower average latency than +sw by enabling shorter seek distance.

For workload prn_1, the performance of z-a is slightly better than sg with a decrease of 2.1 percent in the average latency as Z-Alloc better preserves the spatial locality around the phase boundaries. Adding Top-Buffer alone further decreases the average latency by 33.4 percent and the write amplification by 50.3 percent. This is because prn_1 has a good write hit ratio (81.1 percent), so the Top-Buffer can absorb a majority of the bottom writes. As the write amplification is almost optimal (WA = 1.003 here, and WA = 1 is minimum), subsequent optimization can hardly improve the performance. Therefore, the average latency is nearly the same as the buffer-only scheme when adding more design components.

In the middle figures (where we use a TG size of 5 TB), we observe that z-a reduces the average latency from sg by 20 percent (src2_2) and 12 percent (prn_1), and our reduces that of +sw by 25 percent (src2_2) and 72 percent (prn_1). This is in contrast with the upper figures where the benefits of Z-Alloc and virtual frames are not obvious. This is because the address range coverage of the MSR traces is small (413 GB for prn_1 and only 182 GB for src2_2). Locality preserving optimizations such as Z-Alloc and virtual frames will not help much because the seek time is already small within a small TG. In addition, virtual frames work better when TG usage is not extremely high leaving only a few free top tracks available for the whole TG. When virtual frames are applied, the available free top tracks become even more limited, and any benefit may be offset by increased write amplification. Both traces in the middle figures also show that although applying Top-Buffer and Block-Swap reduces WA, the average latency is worse (+bf, +ab, +sw). This is because the longer seek time due to the relocated blocks negates the benefits of the reduced WA. However, virtual frames combined with multi-phase allocation successfully mitigates the negative impact of longer seek time and achieves better performance (our). Additionally, we test with the trace's LBAs scaled up proportionally to fill a 5 TB address space (bottom figures), and observe the same, but smaller, benefit: z-a reduces the average latency from sg by 11 percent (src2_2) and 2 percent (prn_1), and our reduces that of +sw by 16 percent (src2_2) and 9 percent (prn_1).

6.5 Effectiveness of Optimizations

6.5.1 Evaluating Virtual Frames

In this test, we evaluate how the width of the virtual frames affects the performance improvement. The TG space usage is set to 95 percent to mimic a high utilization workload,

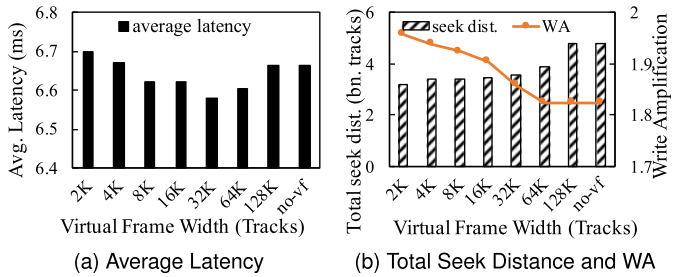


Fig. 7. Evaluation of varied virtual frame widths compared to TrackLace without virtual frames (no-vf).

and the number of Z-Alloc phases is set to six to spread the unallocated top tracks across more virtual frames in the TG (Section 4.1.3). We range the virtual frame width from 2K to 128K tracks (K = thousand), each time doubling the virtual frame size. For comparison, we also run the same trace using TrackLace without any virtual frames (no-vf). The trace replayed is src2_2, and the result is summarized in Fig. 7 (note that the y-axis does not start at zero and is zoomed to show detail in the average latency and the write amplification plots).

The patterned bars in Fig. 7b (left axis) show the total seek distance collected from the simulator for varied frame widths. This illustrates that a smaller virtual frame has a shorter total seek distance than a larger virtual frame. As the virtual frame width increases, both local reads and write-back operations have to travel longer distances as the data block is displaced farther from the original location, which will cause a longer seek time. When the virtual frame width is 128K, the total seek distance is the same as the case without virtual frames (no-vf) (resulting in the same average latency and write amplification for both). This is because a virtual frame width of 128K tracks is larger than the range the workload covers, so it is equivalent to TrackLace without any virtual frames.

Having a larger virtual frame width means there are more unallocated tracks to choose from for buffering, especially when there is some small range of bottom tracks that get more updates. If the amount of updates exceeds the buffer capacity of the local virtual frame, higher write amplification will result (as found in workload proj_2 in Section 6.3.1). The WA line of Fig. 7b (right axis) shows that the write amplification is decreasing as the virtual frame width doubles. When the virtual frame size is small, some of the virtual frames get concentrated bottom track updates and trigger frequent eviction, but nearby virtual frames may have a small number of bottom updates and their resource of unallocated top tracks is wasted. By increasing the width of the virtual frames, the bottom updates of a heavily updated virtual frame can be spread into a larger range of top tracks thus have a better chance to find an unallocated top track for buffering.

In essence, increasing the virtual frame width increases the total seek distance penalty but can reduce the write amplification penalty. The combination of these two effects results in Fig. 7a. It shows that an intermediate width (32K) produces lower average latency than other widths by providing a good balance between the seek distance and the write amplification.

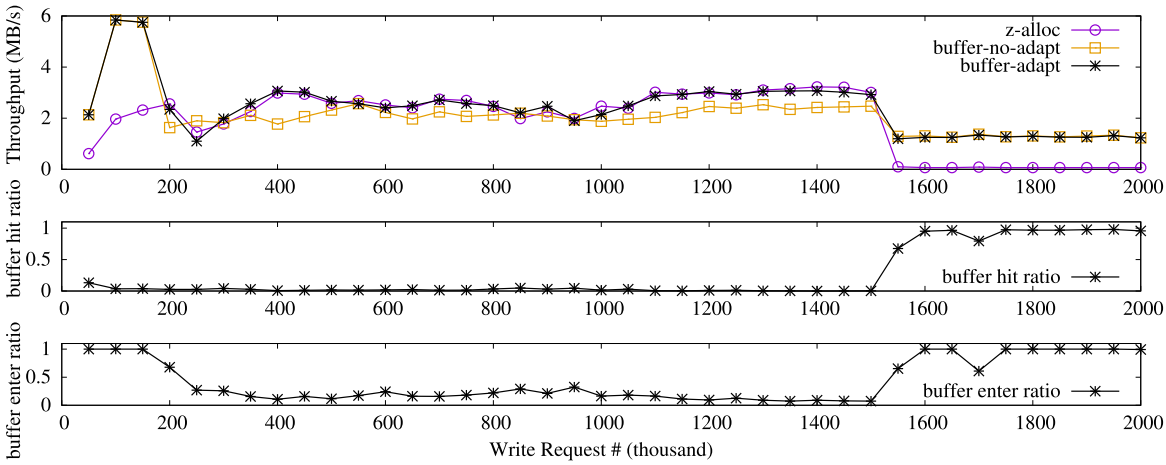


Fig. 8. In a workload mixed with good and poor write locality, Top-Buffer with adaptive buffering (`buffer-adapt`) is able to adapt to the write locality and performs better overall than either the Z-Alloc baseline `z-alloc` or Top-Buffer without the adaptive scheme (`buffer-no-adapt`).

6.5.2 Evaluating Adaptive Buffering

To evaluate the effectiveness and sensitivity of the adaptive buffering scheme, we concatenate two traces with different write locality: poor locality trace (`src1_0`, Top-Buffer write hit ratio 1.3 percent) and good locality trace (`prxy_0`, Top-Buffer write hit ratio 99.5 percent). We filter out the read requests from the trace before replaying to isolate other irrelevant factors. The TG is set to be 80 percent full, and the maximum space of the Top-Buffer is $B_{max} = 2\%$.

Three competing schemes are used: `z-alloc` is the three-phase Z-Alloc; `buffer-no-adapt` is `z-alloc` with the Top-Buffer; and `buffer-adapt` adds the adaptive buffering optimization to `buffer-no-adapt`. The throughput of the three schemes is shown in the top plot in Fig. 8. Besides, to show direct evidence of the adaptiveness of the scheme, the buffer write hit ratio and the buffer enter ratio (i.e., admission rate) of the `buffer-adapt` scheme are depicted in the middle and bottom plots in Fig. 8, respectively. Here the buffer enter ratio is the rate at which Top-Buffer permits the write requests to enter the Top-Buffer; e.g., a ratio value of 1 means the Top-Buffer is fully "open" and buffers all bottom update requests, while a value of 0.5 buffers half the bottom updates.

The trace begins with a workload with poor write locality; then from the $1500K$-th request, it switches to a workload with good write locality. The average throughput of `buffer-adapt` is 2.54 MB/s, which is 17.2 percent better than buffer-no-adapt (2.17 MB/s) and is 2.56× that of `z-alloc` (0.99 MB/s).

In the beginning (roughly 0∼200K) when the Top-Buffer is initially empty, all the requests are redirected to the Top-Buffer in both the `buffer-adapt` and `buffer-no-adapt` schemes. Because such request redirection consolidates the scattered write requests into large sequential writes to the Top-Buffer and does not yet causes in-place updates, both of the schemes outperform the `z-alloc` baseline in the beginning.

However, as the Top-Buffer becomes full at about the $200K$-th request, Top-Buffer cleaning, namely SCP (see Section 3.2), is triggered and the throughput of both `buffer-adapt` and `buffer-no-adapt` drops. As the writes have low locality, `buffer-no-adapt` redirects the requests into the Top-Buffer and evicts them with almost no hits. The overhead of such redirection and migration makes the throughput of `buffer-no-adapt` even worse than the `z-alloc` baseline, which is performing in-place bottom track updates. By contrast, `buffer-adapt` is able to adapt to the poor write locality and quickly reduces the entry rate of the Top-Buffer so that the majority of the low locality write requests will not enter the Top-Buffer. As we can see from the bottom figure (roughly between 200K∼1500K), `buffer-adapt` controls the Top-Buffer entry rate resulting in an average enter ratio fluctuating around the 20 percent mark. Therefore, a large portion of the low locality write requests will be directly written back to the destination bottom tracks without causing the redirection and migration overhead. As a result, `buffer-adapt` exhibits a throughput similar to the baseline during the poor write locality duration.

Finally, at the $1500K$-th request when there is a workload phase change and the write locality becomes good (the Top-Buffer write hit ratio grows to near 100 percent), `buffer-adapt` is able to respond and quickly raises the buffer enter ratio up to near 100 percent. Consequently, the throughput of `buffer-adapt` adjusts to match that of `buffer-no-adapt` (with its constantly open buffer) and is far higher than that of the bufferless `z-alloc` baseline.

7 Related Work

7.1 Data Management for SMR

Cassuto *et al.* [23] separate the SMR space into caching space and permanent storage space and define an *indirection system* for SMR drives. Amer *et al.* [24], [25] explore different design spaces for both the data management and the layout of SMR drives. Hall *et al.* [26] propose a data handling algorithm where the larger *I-region* is continuously refreshed while the smaller *E-region* serves as a buffer for accepting non-sequential writes. H-SWD [27] introduces hot/cold data identification in data placement decisions. Kadekodi *et al.* [28] explore a finer granularity of track interference model within the SMR drive. He and Du exploit track-level mapping for SMR drives and propose

static [29] and dynamic [30] track-level mapping schemes to reduce the write amplification and improve performance. As the zoned block command/device API is established, three SMR models, drive-managed (DM), host-managed (HM), and host-aware (HA), foster further SMR research. Manzanares *et al.* design a zone-based extent allocator (ZEA) [31] for HM-SMR drives. Aghayev *et al.* reverse-engineer the DM-SMR model in Skylight [17], [18], while Wu *et al.* carry out performance evaluation to characterize the HA-SMR model [19], [20]. Recently, ZoneAlloy [32] is proposed to address the data management issues in emerging hybrid SMR drives where the CMR and SMR formats can be converted on demand.

7.2 Data Management for IMR

In Gao *et al.* [7], [16], two data management schemes are proposed: a two-phase and a three-phase implementation. In their two-phase implementation, bottom tracks are first allocated and then the top tracks. In their three-phase implementation, while bottom tracks are allocated during the first phase in the same way as the two-phase implementation, in the second phase only alternating top tracks are allocated. There is a final third phase allocating space from the remaining top tracks. In both their two-phase and three-phase implementations, tracks are allocated in the same radial direction. Our work first proposes a Z-Alloc scheme that differs from their three-phase implementation by: 1) reversing the allocation and addressing direction of even-numbered phases to preserve data locality, and 2) extending to multiple phases to spread the unallocated top tracks throughout the TG. Moreover, whereas the Seagate three-phase design maps data blocks statically, our TrackLace can adapt to the access pattern of dynamic workloads by buffering updates to the unallocated top tracks (Top-Buffer) and swapping hot bottom track data with the cold top track data (Block-Swap).

A most recent work by Hajkazemi *et al.* [33] analyzes the in-place update approach to IMR bottom tracks and proposes three track-based translation layers, namely *track flipping*, *selective track caching*, and *dynamic track mapping*, to reduce the write amplification of IMR drives. TrackLace differs in several aspects. 1) TrackLace can adapt to different disk usage and does not have to protect top tracks when updating free bottom tracks in low usage, while the approach by Hajkazemi *et al.* assumes top tracks always contain valid data to protect. 2) TrackLace is designed to be able to be adjusted for implementation in different parts of the I/O stack, while the track-based translation layers in [33] are targeted for in-disk implementation with very tight memory constraints. 3) The translation layer proposed in [33] is based on tracks, while TrackLace has a block-level mapping for more flexibility. 4) The Top-Buffer in TrackLace will give top tracks back to the user data in situations with very high utilization, while the selective track caching scheme in [33] requires reserved space that cannot be reclaimed for allocation of new user data.

8 Conclusion and Future Work

IMR is a recently proposed recording technology that, when used with HAMR or MAMR, can have a good areal data density with much less write amplification than SMR. In this paper, we investigate how IMR can be used with even less write amplification to provide better throughput and lower latency. We propose TrackLace to efficiently manage the data in IMR drives. TrackLace has three main design components: Z-Alloc allocates space according to different utilization phases; Top-Buffer makes use of unallocated top tracks to buffer bottom track updates; and Block-Swap can swap hot data from a bottom track with cold data from a top track. Furthermore, we propose two optimizations: virtual frames to reduce the seek time and adaptive buffering that switches buffering on and off according the workload. Evaluations with Microsoft Research Cambridge traces show that TrackLace can decrease the average latency, increase the throughput, and reduce the write amplification compared with the baseline approaches.

Possible future work is mentioned throughout this paper. It includes adaptive/heterogeneous virtual frame width and use of a TRIM command to mark invalid data or tracks that can be safely overwritten by nearby track updates. Additionally, alternative designs that distribute unallocated tracks on demand can be explored.

Acknowledgments

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

References

[1] E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. Ts'o, "Disks for data centers," Tech. Rep., Google, Feb. 2016. [Online]. Available: https://research.google/pubs/pub44830/

[2] R. E. Rottmayer *et al.*,"Heat-assisted magnetic recording," *IEEE Trans. Magn.*, vol. 42, no. 10, pp. 2417–2421, Oct. 2006.

[3] M. H. Kryder *et al.*,"Heat assisted magnetic recording," *Proc. IEEE*, vol. 96, no. 11, pp. 1810–1835, Nov. 2008.

[4] J.-G. Zhu, X. Zhu, and Y. Tang, "Microwave assisted magnetic recording," *IEEE Trans. Magn.*, vol. 44, no. 1, pp. 125–131, Jan. 2008.

[5] J.-G. Zhu and Y. Wang, "Microwave assisted magnetic recording utilizing perpendicular spin torque oscillator with switchable perpendicular electrodes," *IEEE Trans. Magn.*, vol. 46, no. 3, pp. 751–757, Mar. 2010.

[6] E. Hwang, J. Park, R. Rauschmayer, and B. Wilson, "Interlaced magnetic recording," *IEEE Trans. Magn.*, vol. 53, no. 4, pp. 1–7, Apr. 2017.

[7] K. Gao, W. Zhu, and E. Gage, "Interlaced magnetic recording," US Patent 9,728,206, Aug. 8, 2017.

[8] S. Granz *et al.*, "Heat-assisted interlaced magnetic recording," *IEEE Trans. Mag.*, vol. 54, no. 2, pp. 2018, Art. no. 3100504.

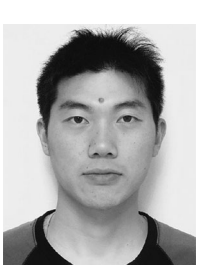
[9] A. Krichevsky, "Heat assisted magnetic recording with interlaced high-power heated and low-power heated tracks," US Patent 9,099,103, Aug. 4, 2015.

[10] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual," Parallel Data Lab, Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-PDL-08–101, 2008. [Online]. Available: https://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf

[11] R. Wood, M. Williams, A. Kavcic, and J. Miles, "The feasibility of magnetic recording at 10 terabits per square inch on conventional media," *IEEE Trans. Magn.*, vol. 45, no. 2, pp. 917–923, Feb. 2009.

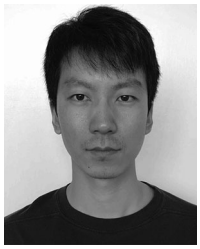
[12] I. Tagawa and M. Williams, "Shingle-write technology and gain estimation," FA-02 presentation in INTERMAG 2009, Sacramento, CA, May 2009. [Online]. Available: http://www.intermagconference.com/intermag2009/src/Program1.pdf

[13] G. Gibson and M. Polte, "Directions for shingled-write and two-dimensional magnetic recording system architectures: Synergies with solid-state disks," *Parallel Data Lab, Carnegie Mellon Univ., Pittsburgh, PA,* Tech. Rep. *CMU-PDL-09–014*, 2009.
[14] K. Gao, "Architecture for hard disk drives," *IEEE Magn. Lett.*, vol. 9, pp. 1–5, 2018, Art no. 4501705.
[15] T. Feldman, "Flex dynamic recording," in *Proc. USENIX Login*, 2018, vol. 43, no. 1, pp. 44–47.
[16] K. Gao, W. Zhu, and E. Gage, "Write management for interlaced magnetic recording devices," US Patent 9,508,362. Nov. 29, 2016.
[17] A. Aghayev and P. Desnoyers, "Skylight—A window on shingled disk operation," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 135–149.
[18] A. Aghayev, M. Shafaei, and P. Desnoyers, "Skylight – A window on shingled disk operation," *ACM Trans. Storage*, vol. 11, no. 4, pp. 16:1–16:28, Oct. 2015.
[19] F. Wu, M.-C. Yang, Z. Fan, B. Zhang, X. Ge, and D. H. Du, "Evaluating host aware SMR drives," in *Proc. 8th USENIX Workshop Hot Topics Storage File Syst.*, 2016, pp. 31–35.
[20] F. Wu, Z. Fan, M.-C. Yang, B. Zhang, X. Ge, and D. H. Du, "Performance evaluation of host aware shingled magnetic recording (ha-smr) drives," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1932–1945, Nov. 2017.
[21] C. Ruemmler and J. Wilkes,"An introduction to disk drive modeling," *Computer*, vol. 27, no. 3, pp. 17–28, 1994.
[22] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, 2008, Art. no. 10.
[23] Y. Cassuto, M. A. Sanvido, C. Guyot, D. R. Hall, and Z. Z. Bandic, "Indirection systems for shingled-recording disk drives," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–14.
[24] A. Amer, J. Holliday, D. D. Long, E. L. Miller, J.-F. Pâris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *IEEE Trans. Magn.*, vol. 47, no. 10, pp. 3691–3697, Oct. 2011.
[25] A. Amer, D. D. Long, E. L. Miller, J.-F. Paris, and S. T. Schwarz, "Design issues for a shingled write disk system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–12.
[26] D. Hall, J. H. Marcos, and J. D. Coker, "Data handling algorithms for autonomous shingled magnetic recording HDDs," *IEEE Transactions on Magnetics*, vol. 48, no. 5, pp. 1777–1781, May 2012.
[27] C.-I. Lin, D. Park, W. He, and D. H. Du, "H-SWD: Incorporating hot data identification into shingled write disks," in *Proc. 20th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2012, pp. 321–330.
[28] S. Kadekodi, S. Pimpale, and G. A. Gibson, "Caveat-scriptor: Write anywhere shingled disks," in *Proc. 7th USENIX Workshop Hot Topics Storage File Syst.*, 2015, Art. no. 16.
[29] W. He and D. H. Du, "Novel address mappings for shingled write disks," in *Proc. 6th USENIX Workshop Hot Topics Storage File Syst.*, 2014, Art. no. 5.
[30] W. He and D. H. Du, "Smart: An approach to shingled magnetic recording translation," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 121—133.
[31] A. Manzanares, N. Watkins, C. Guyot, D. LeMoal, C. Maltzahn, and Z. Bandic, "Zea, a data management approach for SMR," in *Proc. 8th USENIX Workshop Hot Topics Storage File Syst.*, 2016, pp. 26—30.
[32] F. Wu *et al.*, "Zonealloy: Elastic data and space management for hybrid SMR drives," in *Proc. 11th USENIX Workshop Hot Topics Storage File Syst.*, 2019, Art. no. 2.
[33] M. H. Hajkazemi, P. Desnoyers, A. N. Kulkarni, and T. R. Feldman, "Track-based translation layers for interlaced magnetic recording," in *Proc. USENIX Annu. Techn. Conf.*, 2019, pp. 821—832.



Fenggang Wu received the BS and MS degrees in computer science and technology from Shanghai Jiao Tong University, in 2010 and 2013, he is currently working towards the PhD degree in the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. His research focuses on key-value stores, non-volatile memory based storage systems, archival drives (SMR/IMR), and data deduplication.

Bingzhe Li received the BS degree in electrical engineering from Each China Jiaotong University, in 2010, he received the PhD degree in electrical and computer engineering from the University of Minnesota, Twin Cities, in 2018. Currently, he is working as a postdoctoral associate with the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. His research focuses on the intelligent storage system and low-cost computing architecture.



Baoquan Zhang is working toward the PhD degree in computer science at the University of Minnesota - Twin Cities advised by professor David H.C. Du. Currently, he is working on memory/storage systems including persistent memory systems, key-value stores, archiving storage systems and so on.

Zhichao Cao received the BS degree in science in automation from Tsinghua University, in 2013, currently, he is currently working toward the PhD degree in computer science at the University of Minnesota, Twin Cities. His research focuses are on key-value stores, storage systems, file system, tiered storage, data deduplication, and new storage devices (NVM, SMR, IMR).

Jim Diehl received the master's and bachelor's degrees in computer engineering from the University of Minnesota, Twin Cities, he is currently working toward the PhD degree in electrical engineering with a minor in computer science. His research interests include hierarchical storage systems, emerging storage hardware such as Kinetic drives, machine learning, and other aspects of intelligent storage systems.

Hao Wen received the BS degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2013, he received the PhD degree in computer science at the Department of Computer Science and Engineering, University of Minnesota, Twin Cities, in 2019. He is currently a software engineer with Twitter Inc. His research focuses on storage QoS, software defined storage, Docker, Kubernetes and data deduplication.

David H.C. Du (Fellow, IEEE) and received the BS degree from National Tsing-Hua University, Taiwan, in 1974, and the MS and PhD degree from the University of Washington, Seattle, in 1980 and 1981, respectively. He is currently the qwest chair professor of computer science and engineering at the University of Minnesota, Twin Cities, and the director of the NSF Center for Research in Intelligent Storage (CRIS). His current research focuses on intelligent and large storage systems, cyber-physical systems, and vehicular/sensor networks. He serves on editorial boards of several international journals. He was a program director (IPA) at the National Science Foundation (NSF) CISE/CNS Division from 2006 to 2008. He has served as conference chair, program committee chair, and general chair for several major conferences in databases, security, and parallel processing.