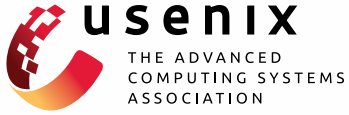

Skylight—A Window on Shingled Disk Operation

Abutalib Aghayev and Peter Desnoyers, ***Northeastern University***

https://www.usenix.org/conference/fast15/technical-sessions/presentation/aghayev

This paper is included in the Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15).

February 16–19, 2015 • Santa Clara, CA, USA

ISBN 978-1-931971-201

Open access to the Proceedings of the 13th USENIX Conference on File and Storage Technologies is sponsored by USENIX

Skylight—A Window on Shingled Disk Operation

Abutalib Aghayev Peter Desnoyers
Northeastern University

Abstract

We introduce Skylight, a novel methodology that combines software and hardware techniques to reverse engineer key properties of drive-managed Shingled Magnetic Recording (SMR) drives. The software part of Skylight measures the latency of controlled I/O operations to infer important properties of drive-managed SMR, including type, structure, and size of the persistent cache; type of cleaning algorithm; type of block mapping; and size of bands. The hardware part of Skylight tracks drive head movements during these tests, using a high-speed camera through an observation window drilled through the cover of the drive. These observations not only confirm inferences from measurements, but resolve ambiguities that arise from the use of latency measurements alone. We show the generality and efficacy of our techniques by running them on top of three emulated and two real SMR drives, discovering valuable performance-relevant details of the behavior of the real SMR drives.

1 Introduction

In the nearly 60 years since the hard disk drive (HDD) has been introduced, it has become the mainstay of computer storage systems. In 2013 the hard drive industry shipped over 400 exabytes [1] of storage, or almost 60 gigabytes for every person on earth. Although facing strong competition from NAND flash-based solid-state drives (SSDs), magnetic disks hold a 10× advantage over flash in both total bits shipped [2] and per-bit cost [3], an advantage that will persist if density improvements continue at current rates.

The most recent growth in disk capacity is the result of improvements to perpendicular magnetic recording (PMR) [4], which has yielded terabyte drives by enabling bits as short as 20 nm in tracks 70 nm wide [5], but further increases will require new technologies [6]. Shingled Magnetic Recording (SMR) [7] is the first such technology to reach market: 5 TB drives are available from Seagate [8] and shipments of 8 TB and 10 TB drives have been announced by Seagate [9] and HGST [10]. Other technologies (Heat-Assisted Magnetic Recording [11] and Bit-Patterned Media [12]) remain in the research stage, and may in fact use shingled recording when they are released [13].

Shingled recording spaces tracks more closely, so they overlap like rows of shingles on a roof, squeezing more tracks and bits onto each platter [7]. The increase in density comes at a cost in complexity, as modifying a disk sector will corrupt other data on the overlapped tracks, requiring copying to avoid data loss [14–17]. Rather than push this work onto the host file system [18,19], SMR drives shipped to date preserve compatibility with existing drives by implementing a Shingle Translation Layer (STL) [20,21] that hides this complexity.

Like an SSD, an SMR drive combines out-of-place writes with dynamic mapping in order to efficiently update data, resulting in a drive with performance much different from that of a Conventional Magnetic Recording (CMR) drive due to seek overhead for out-of-order operations. However unlike SSDs, which have been extensively measured and characterized [22,23], little is known about the behavior and performance of SMR drives and their translation layers, or how to optimize file systems, storage arrays, and applications to best use them.

We introduce a methodology for measuring and characterizing such drives, developing a specific series of micro-benchmarks for this characterization process, much as has been done in the past for conventional drives [24–26]. We augment these timing measurements with a novel technique that tracks actual head movements via high-speed camera and image processing and provides a source of reliable information in cases where timing results are ambiguous.

We validate this methodology on three different emulated drives that use STLs previously described in the literature [20, 21, 27], implemented as a Linux *device mapper target* [28] over a conventional drive, demonstrating accurate inference of properties. We then apply this methodology to 5 TB and 8 TB SMR drives provided by Seagate, inferring the STL algorithm and its properties and providing the first public characterization of such drives.

Using our approach we are able to discover important characteristics of the Seagate SMR drives and their translation layer, including the following:

Cache type and size: The drives use a persistent disk cache of 20 GiB and 25 GiB on the 5 TB and 8 TB drives, respectively, with high random write speed until the cache is full. The effective cache size is a function of write size and queue depth.

Persistent cache structure: The persistent disk cache is written as journal entries with quantized sizes—a phenomenon absent from the academic literature on SMRs.

Block Mapping: Non-cached data is statically mapped, using a fixed assignment of logical block addresses (LBAs) to physical block addresses (PBAs), similar to that used in CMR drives, with implications for performance and durability.

Band size: SMR drives organize data in *bands*—a set of contiguous tracks that are re-written as a unit; the examined drives have a small band size of 15–40 MiB.

Cleaning mechanism: Aggressive cleaning during idle times moves data from the persistent cache to bands; cleaning duration is 0.6–1.6 s per modified band.

Our results show the details that may be discovered using Skylight, most of which impact (negatively or positively) the performance of different workloads, as described in § 6. These results—and the toolset allowing similar measurements on new drives—should thus be useful to users of SMR drives, both in determining what workloads are best suited for these drives and in modifying applications to better use them. In addition, we hope that they will be of use to designers of SMR drives and their translation layers, by illustrating the effects of low-level design decisions on system-level performance.

In the rest of the paper we give an overview of Shingled Magnetic Recording (§ 2) followed by the description of emulated and real drives examined (§ 3). We then present our characterization methodology and apply it to all of the drives (§ 4); finally, we survey related work (§ 5) and present our conclusions (§ 6).

2 Background

Shingled recording is a response to limitations on areal density with perpendicular magnetic recording due to the superparamagnetic limit [6]. In brief, for bits to become smaller, write heads must become narrower, resulting in weaker magnetic fields. This requires lower coercivity (easily recordable) media, which is more vulnerable to bit flips due to thermal noise, requiring larger bits for reliability. As the head gets smaller this minimum bit size gets larger, until it reaches the width of the head and further scaling is impossible.

Several technologies have been proposed to go beyond this limit, of which SMR is the simplest [7]. To decrease the bit size further, SMR reduces the track width while keeping the head size constant, resulting in a head that writes a path several tracks wide. Tracks are then overlapped like rows of shingles on a roof, as seen in Figure 1. Writing these overlapping tracks requires only incremental changes in manufacturing, but much greater system changes, as it becomes impossible to re-write a single sector without destroying data on the overlapped sectors.

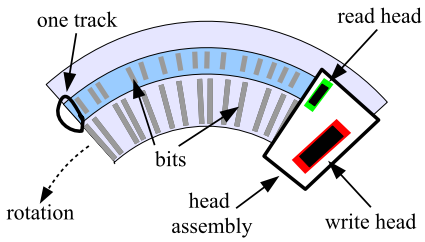

Figure 1: Shingled disk tracks with head width $k=2$

For maximum capacity an SMR drive could be written from beginning to end, utilizing all tracks. Modifying any of this data, however, would require reading and re-writing the data that would be damaged by that write, and data to be damaged by the re-write, etc. until the end of the surface is reached. This cascade of copying may be halted by inserting *guard regions*—tracks written at the full head width—so that the tracks before the guard region may be re-written without affecting any tracks following it, as shown in Figure 2. These guard regions divide each disk surface into re-writable bands; since the guards hold a single track's worth of data, storage efficiency for a band size of b tracks is $\frac{b}{b+k-1}$.

Given knowledge of these bands, a host file system can ensure they are only written sequentially, for example, by implementing a log-structured file system [18,29]. Standards are being developed to allow a drive to identify these bands to the host [19]: *host-aware* drives report sequential-write-preferred bands (an internal STL handles non-sequential writes), and *host-managed* drives report sequential-write-required bands. These standards are still in draft form, and to date no drives based on them are available on the open market.

Alternately the *drive-managed* disks present a standard re-writable block interface that is implemented by an internal Shingle Translation Layer, much as an SSD uses a Flash Translation Layer (FTL). Although the two are logically similar, appropriate algorithms differ due to differences in the constraints placed by the underlying media: (a) high seek times for non-sequential access, (b) lack of high-speed reads, (c) use of large (10s to 100s of MB) cleaning units, and (d) lack of wear-out, eliminating the need for wear leveling.

These translation layers typically store all data in bands where it is mapped at a coarse granularity, and devote a small fraction of the disk to a *persistent cache*, as shown in Figure 2, which contains copies of recently-written data. Data that should be retrieved from the persistent cache may be identified by checking a *persistent cache map* (or

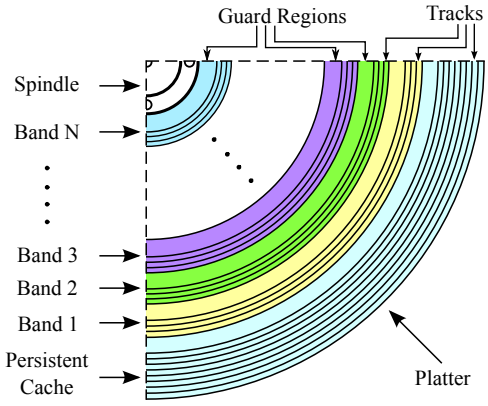

Figure 2: Surface of a platter in a hypothetical SMR drive. A persistent cache consisting of 9 tracks is located at the outer diameter. The guard region that separates the persistent cache from the first band is simply a track that is written at a full head width of k tracks. Although the guard region occupies the width of k tracks, it contains a single track's worth of data and the remaining k-1 tracks are wasted. The bands consist of 4 tracks, also separated with a guard region. Overwriting a sector in the last track of any band will not affect the following band. Overwriting a sector in any of the tracks will require reading and re-writing all of the tracks starting at the affected track and ending at the guard region within the band.

exception map) [20, 21]. Data is moved back from the persistent cache to bands by the process of *cleaning*, which performs read-modify-write (RMW) on every band whose data was overwritten. The cleaning process may be *lazy*, running only when the free cache space is low, or *aggressive*, running during idle times.

In one translation approach, a static mapping algorithmically assigns a *native location* [20] (a PBA) to each LBA in the same way as is done in a CMR drive. An alternate approach uses coarse-grained dynamic mapping for non-cached LBAs [20], in combination with a small number of free bands. During cleaning, the drive writes an updated band to one of these free bands and then updates the dynamic map, potentially eliminating the need for a temporary staging area for cleaning updates and sequential writes.

In any of these cases drive operation may change based on the setting of the *volatile cache* (enabled or disabled) [30]. When the volatile cache is disabled, writes are required to be persistent before completion is reported to the host. When it is enabled, persistence is only guaranteed after a FLUSH command or a write command with the flush (FUA) flag set.

3 Test Drives

We now describe the drives we study. First, we discuss how we emulate three SMR drives using our implementation of two STLs described in the literature. Second, we describe the real SMR drives we study in this paper and the real CMR drive we use for emulating SMR drives.

3.1 Emulated Drives

We implement Cassuto et al.'s *set-associative* STL [20] and a variant of their *S-blocks* STL [20, 31], which we call *fully-associative* STL, as Linux device mapper targets. These are kernel modules that export a pseudo block device to user-space that internally behaves like a drive-managed SMR—the module translates incoming requests using the translation algorithm and executes them on a CMR drive.

The *set-associative* STL manages the disk as a set of N iso-capacity (same-sized) data bands, with typical sizes of 20–40 MiB, and uses a small (1–10%) section of the disk as the persistent cache. The persistent cache is also managed as a set of n iso-capacity cache bands where $n \ll N$. When a block in data band a is to be written, a cache band chosen through $(a \bmod n)$; the next empty block in this cache band is written and the persistent cache map is updated. Further accesses to the block are served from the cache band until cleaning moves the block to its native location, which happens when the cache band becomes full.

The *fully-associative* STL, on the other hand, divides the disk into large (we used 40 GiB) zones and manages each zone independently. A zone starts with 5% of its capacity provisioned to free bands for handling updates. When a block in logical band a is to be written to the corresponding physical band b, a free band c is chosen and written to and the persistent cache map is updated. When the number of free bands falls below a threshold, cleaning merges the bands b and c and writes it to a new band d and remaps the logical band a to the physical band d, freeing bands b and c in the process. This dynamic mapping of bands allows the fully-associative STL to handle streaming writes with zero overhead.

To evaluate the accuracy of our emulation strategy, we implemented a pass-through device mapper target and found negligible overhead for our tests, confirming a previous study [32]. Although in theory, this emulation approach may seem disadvantaged by the lack of access to exact sector layout, in practice this is not the case—even in real SMR drives, the STL running inside the drive is implemented on top of a layer that provides linear PBAs by hiding sector layout and defect management [33]. Therefore, we believe that the device mapper target running on top of a CMR drive provides an accurate model for predicting the behavior of an STL implemented by the controller of an SMR drive.

Table 1 shows the three emulated SMR drive configurations we use in our tests. The first two drives use the set-associative STL, and they differ in the type of persistent cache and band size. The last drive uses the fully-associative STL and disk for the persistent cache. We do not have a drive configuration combining the fully-associative STL and flash for persistent cache, since the fully-associative STL was designed for a drive with a disk cache and uses multiple disk caches evenly spread out on a disk to avoid long seeks during cleaning.

Drive Name	STL	Persistent Cache Type and Size	Disk Cache Multiplicity	Cleaning Type	Band Size	Mapping Type	Size
Emulated-SMR-1	Set-associative	Disk, 37.2 GiB	Single at ID	Lazy	40 MiB	Static	3.9 TB
Emulated-SMR-2	Set-associative	Flash, 9.98 GiB	N/A	Lazy	25 MiB	Static	3.9 TB
Emulated-SMR-3	Fully-associative	Disk, 37.2 GiB	Multiple	Aggressive	20 MiB	Dynamic	3.9 TB

Table 1: Emulated SMR drive configurations.

To emulate an SMR drive with a flash cache (Emulate-SMR-2) we use the Emulate-SMR-1 implementation, but use a device mapper `linear` target to redirect the underlying LBAs corresponding to the persistent cache, storing them on an SSD.

To check the correctness of the emulated SMR drives we ran repeated burn-in tests using `fio` [34]. We also formatted emulated drives with `ext4`, compiled the Linux kernel on top, and successfully booted the system with the compiled kernel. The source code for STLs (1,200 lines of C) and a testing framework (250 lines of Go) are available at `http://sssl.ccs.neu.edu/skylight`.

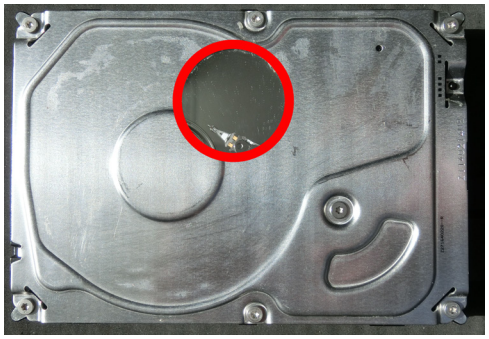

Figure 3: SMR drive with the observation window encircled in red. Head assembly is visible parked at the inner diameter.

3.2 Real Drives

Two real SMR drives were tested: Seagate ST5000AS0011, a 5900 RPM desktop drive (rotation time ≈ 10 ms) with four platters, eight heads, and 5 TB capacity (termed Seagate-SMR below), and Seagate ST8000AS0011, a similar drive with six platters, twelve heads and 8 TB capacity. Emulated drives use a Seagate ST4000NC001 (Seagate-CMR), a real CMR drive identical in drive mechanics and specification (except the 4 TB capacity) to the ST5000AS0011. Results for the 8 TB and 5 TB SMR drives were similar; to save space, we only present results for the publicly-available 5 TB drive.

4 Characterization Tests

To motivate our drive characterization methodology we first describe the goals of our measurements. We then describe the mechanisms and methodology for the tests, and finally present results for each tested drive. For emulated SMR drives, we show that the tests produce accurate answers, based on implemented parameters; for real SMR drives we discover their properties. The behavior of the real SMR drives under some of the tests engenders further investigation, leading to the discovery of important details about their operation.

4.1 Characterization Goals

The goal of our measurements is to determine key drive characteristics and parameters:

Drive type: In the absence of information from the vendor, is a drive an SMR or a CMR?

Persistent cache type: Does the drive use flash or disk for the persistent cache? The type of the persistent cache affects the performance of random writes and reliable—volatile cache-disabled—sequential writes. If the drive uses disk for persistent cache, is it a single cache, or is it distributed across the drive [20, 31]? The layout of the persistent disk cache affects the cleaning performance and the performance of the sequential read of a sparsely overwritten linear region.

Cleaning: Does the drive use aggressive cleaning, improving performance for low duty-cycle applications, or lazy cleaning, which may be better for throughput-oriented ones? Can we predict the performance impact of cleaning?

Persistent cache size: After some number of out-of-place writes the drive will need to begin a cleaning process, moving data from the persistent cache to bands so that it can accept new writes, negatively affecting performance. What is this limit, as a function of total blocks written, number of write operations, and other factors?

Band size: Since a band is the smallest unit that may be re-written efficiently, knowledge of band size is important for optimizing SMR drive workloads [20, 27]. What are the band sizes for a drive, and are these sizes constant over time and space [35]?

Block mapping: The mapping type affects performance of both cleaning and reliable sequential writes. For LBAs that are not in the persistent cache, is there a static mapping from LBAs to PBAs, or is this mapping dynamic?

Zone structure: Determining the zone structure of a drive is a common step in understanding block mapping and band size, although the structure itself has little effect on external performance.

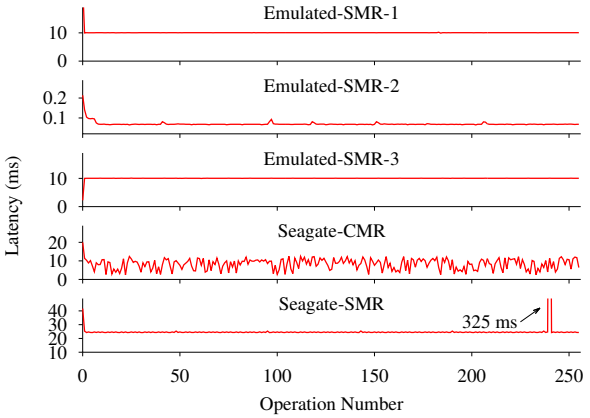

Figure 4: Discovering drive type using latency of random writes. Y-axis varies in each graph.

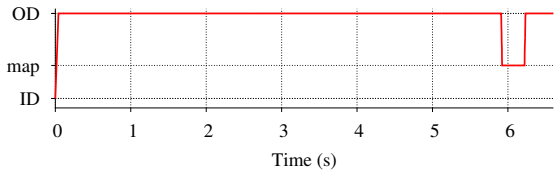

Figure 5: Seagate-SMR head position during random writes.

4.2 Test Mechanisms

The software part of Skylight uses `fio` to generate micro-benchmarks that elicit the drive characteristics. The hardware part of Skylight tracks the head movement during these tests. It resolves ambiguities in the interpretation of the latency data obtained from the micro-benchmarks and leads to discoveries that are not possible with micro-benchmarks alone. To make head tracking possible, we installed (under clean-room conditions) a transparent window in the drive casing over the region traversed by the head. Figure 3 shows the head assembly parked at the inner diameter (ID). We recorded the head movements using Casio EX-ZR500 camera at 1,000 frames per second and processed the recordings with `ffmpeg` to generate head location value for each video frame.

We ran the tests on a 64-bit Intel Core-i3 Haswell system with 16 GiB RAM and 64-bit Linux kernel version 3.14. Unless otherwise stated, we disabled kernel read-ahead, drive look-ahead and drive volatile cache using `hdparm`.

Extensions to `fio` developed for these tests have been integrated back and are available in the latest `fio` release. Slow-motion clips for the head position graphs shown in the paper, as well as the tests themselves, are available at `http://sssl.ccs.neu.edu/skylight`.

4.3 Drive Type and Persistent Cache Type

Test 1 exploits the unusual random write behavior of the SMR drives to differentiate them from CMR drives. While random writes to a CMR drive incur varying latency due to random seek time and rotational delay, random writes to an SMR drive are sequentially logged to the persistent cache with a fixed latency. If random writes are not local, SMR drives using separate persistent caches by the LBA range [20] may still incur varying write latency. Therefore, random writes are done within a small region to ensure that a single persistent cache is used.

Test 1: Discovering Drive Type

1 Write blocks in the first 1 GiB in random order to the drive.
2 **if** latency is fixed **then** the drive is SMR **else** the drive is CMR.

Figure 4 shows the results for this test. Emulated-SMR-1 sequentially writes incoming random writes to the persistent cache. It fills one empty block after another and due to synchronicity of the writes it misses the next empty block by the time the next write arrives. Therefore, it waits for a complete rotation resulting in a 10 ms write latency, which is the rotation time of the underlying CMR drive. The sub-millisecond latency of Emulated-SMR-2 shows that this drive uses flash for the persistent cache. The latency of Emulated-SMR-3 is identical to that of Emulated-SMR-1, suggesting a similar setup. The varying latency of Seagate-CMR identifies it as a conventional drive. Seagate-SMR shows a fixed $\approx$ 25 ms latency with a $\approx$ 325 ms bump at the 240th write. While the fixed latency indicates that it is an SMR drive, we resort to the head position graph to understand why it takes 25 ms to write a single block and what causes the 325 ms latency.

Figure 5 shows that the head, initially parked at the ID, seeks to the outer diameter (OD) for the first write. It stays there during the first 239 writes (incidentally, showing that the persistent cache is at the OD), and on the 240th write it seeks to the center, staying there for $\approx$ 285 ms before seeking back and continuing to write.

Is all of 25 ms latency associated with every block write spent writing or is some of it spent in rotational delay? When we repeat the test multiple times, the completion time of the first write ranges between 41 and 52 ms, while the remaining writes complete in 25 ms. The latency of the first write always consists of a seek from the ID to the OD ($\approx$ 16 ms). We presume that the remaining time is spent in rotational delay—likely waiting for the beginning of a delimited location—and writing (25 ms). Depending on where the head lands after the seek, the latency of the first write changes between 41 ms and 52 ms. The remaining writes are written as they arrive, without seek time and rotational delay, each taking 25 ms. Hence, a single block *host write* results in a 2.5 track *internal write*. In the following section we explore this phenomenon further.

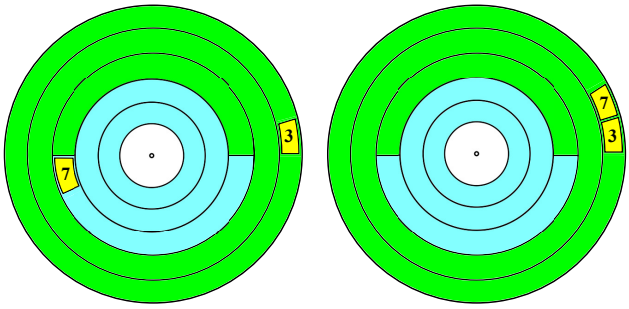

Figure 6: Surface of a disk platter in a hypothetical SMR drive divided into two 2.5 track imaginary regions. The left figure shows the placement of random blocks 3 and 7 when writing synchronously. Each internal write contains a single block and takes 25 ms (50 ms in total) to complete. The drive reports 25 ms write latency for each block; reading the blocks in the written order results in a 5 ms latency. The right figure shows the placement of blocks when writing asynchronously with high queue depth. A single internal write contains both of the blocks, taking 25 ms to complete. The drive still reports 25 ms write latency for each block; reading the blocks back in the written order results in a 10 ms latency due to missed rotation.

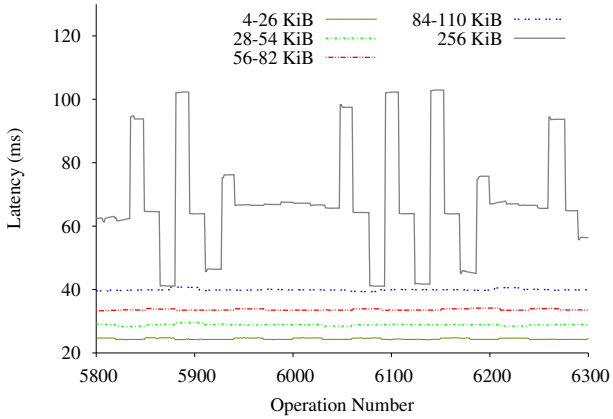

Figure 7: Random write latency of different write sizes on Seagate-SMR, when writing at the queue depth of 31.

4.3.1 Journal Entries with Quantized Sizes

If after Test 1 we immediately read blocks in the written order, read latency is fixed at ≈5 ms, indicating 0.5 track distance (covering a complete track takes a full rotation, which is 10 ms for the drive; therefore 5 ms translates to 0.5 track distance) between blocks. On the other hand, if we write blocks asynchronously at the maximum queue depth of 31 [36] and immediately read them, latency is fixed at ≈10 ms, indicating a missed rotation due to contiguous placement. Furthermore, although the drive still reports 25 ms completion time for every write, asynchronous writes complete faster—for the 256 write operations, asynchronous writes complete in 216 ms whereas synchronous writes complete in 6,539 ms, as seen in Figure 5. Gathering these facts, we arrive at Figure 6. Writing asynchronously with high queue depth allows the drive to pack multiple blocks into a single internal write, placing them contiguously (shown on the right). The drive reports the completion of individual host writes packed into the same internal write once the internal write completes. Thus, although each of the host writes in the same internal write is reported to take 25 ms, it is the same 25 ms that went into writing the internal write. As a result, in the asynchronous case, the drive does fewer internal writes, which accounts for the fast completion time. The contiguous placement also explains the 10 ms latency when reading blocks in the written order. Writing synchronously, however, results in doing a separate internal write for every block (shown on the left), taking longer to complete. Placing blocks starting at the beginning of 2.5 track internal writes explains the 5 ms latency when reading blocks in the written order.

To understand how the internal write size changes with the increasing host write size, we keep writing at the maximum queue depth, gradually increasing the write size. Figure 7 shows that the writes in the range of 4 KiB–26 KiB result in 25 ms latency, suggesting that 31 host writes in this size range fit in a single internal write. As we jump to the 28 KiB writes, the latency increases by ≈5 ms (or 0.5 track) and remains approximately constant for the writes of sizes up to 54 KiB. We observe a similar jump in latency as we cross from 54 KiB to 56 KiB and also from 82 KiB to 84 KiB. This shows that the internal write size increases in 0.5 track increments. Given that the persistent cache is written using a "log-structured journaling mechanism" [37], we infer that the 0.5 track of 2.5 track minimum internal write is the journal entry that grows in 0.5 track increments, and the remaining 2 tracks contain out-of-band data, like parts of the persistent cache map affected by the host writes. The purpose of this quantization of journal entries is not known, but may be in order to reduce rotational delay or simplify delimiting and locating them. We further hypothesize that the 325 ms delay in Figure 4, observed every 240th write, is a map merge operation that stores the updated map at the middle tracks.

As the write size increases to 256 KiB we see varying delays, and inspection of completion times shows less than 31 writes completing in each burst, implying a bound on the journal entry size. Different completion times for large writes suggest that for these, the journal entry size is determined dynamically, likely based on the available drive resources at the time when the journal entry is formed.

4.4 Disk Cache Location and Layout

We next determine the location and layout of the disk cache, exploiting a phenomenon called *fragmented reads* [20]. When sequentially reading a region in an SMR drive, if the cache contains newer version of some of the blocks in the region, the head has to seek to the persistent cache and back, physically fragmenting a logically sequential read. In Test 2, we use these variations in seek time to discover the location and layout of the disk cache.

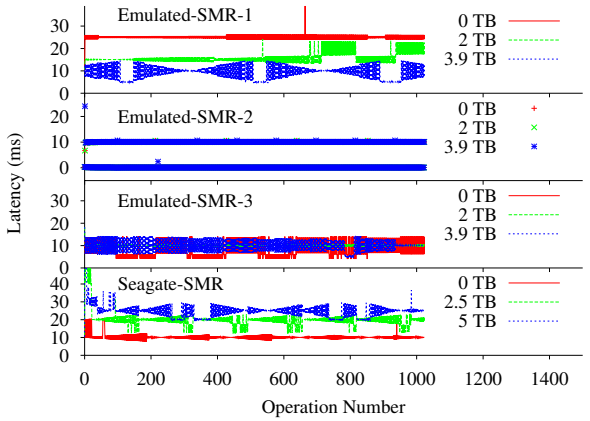

Figure 8: Discovering disk cache structure and location using fragmented reads.

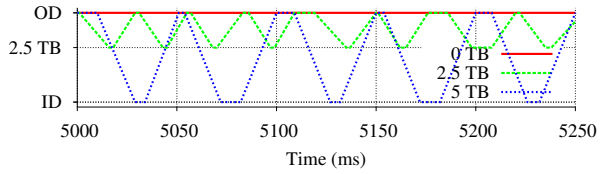

Figure 9: Seagate-SMR head position during fragmented reads.

Test 2: Discovering Disk Cache Location and Layout

1. Starting at a given offset, write a block and skip a block, and so on, writing 512 blocks in total.
2. Starting at the same offset, read 1024 blocks; call average latency lat_{offset}.
3. Repeat steps 1 and 2 at the offsets *high*, *low*, *mid*.
4. **if** $\text{lat}_{high} < \text{lat}_{mid} < \text{lat}_{low}$ **then**
   There is a single disk cache at the ID.
   else if $\text{lat}_{high} > \text{lat}_{mid} > \text{lat}_{low}$ **then**
   There is a single disk cache at the OD.
   else if $\text{lat}_{high} = \text{lat}_{mid} = \text{lat}_{low}$ **then**
   There are multiple disk caches.
   else
   assert($\text{lat}_{high} = \text{lat}_{low}$ **and** $\text{lat}_{high} > \text{lat}_{mid}$)
   There is a single disk cache in the middle.

The test works by choosing a small region and writing every other block in it and then reading the region sequentially from the beginning, forcing a fragmented read. LBA numbering conventionally starts at the OD and grows towards the ID. Therefore, a fragmented read at low LBAs on a drive with the disk cache located at the OD would incur negligible seek time, whereas a fragmented read at high LBAs on the same drive would incur high seek time. Conversely, on a drive with the disk cache located at the ID, a fragmented read would incur high seek time at low LBAs and negligible seek time at high LBAs. On a drive with the disk cache located at the middle diameter (MD), fragmented reads at low and high LBAs would incur similar high seek times and they would incur negligible seek times at middle LBAs. Finally, on a drive with multiple disk caches evenly distributed across the drive, the fragmented read latency would be mostly due to rotational delay and vary little across the LBA space. Guided by these assumptions, to identify the location of the disk cache, the test chooses a small region at low, middle, and high LBAs and forces fragmented reads at these regions.

Figure 8 shows the latency of fragmented reads at three offsets on all SMR drives. The test correctly identifies the Emulated-SMR-1 as having a single cache at the ID. For Emulated-SMR-2 with flash cache, latency is seen to be negligible for flash reads, and a full missed rotation for each disk read. Emulated-SMR-3 is also correctly identified as having multiple disk caches—the latency graph of all fragmented reads overlap, all having the same 10 ms average latency. For Seagate-SMR[1] we confirm that it has a single disk cache at OD.

Figure 9 shows the Seagate-SMR head position during fragmented reads at offsets of 0 TB, 2.5 TB and 5 TB. For offsets of 2.5 TB and 5 TB, we see that the head seeks back and forth between the OD and near-center and between the OD and the ID, respectively, occasionally missing a rotation. The cache-to-data distance for LBAs near 0 TB was too small for the resolution of our camera.

4.5 Cleaning

The fragmented read effect is also used in Test 3 to determine whether the drive uses aggressive or lazy cleaning, by creating a fragmented region and then pausing to allow an aggressive cleaning to run before reading the region back.

Test 3: Discovering Cleaning Type

1. Starting at a given offset, write a block and skip a block and so on, writing 512 blocks in total.
2. Pause for 3–5 seconds.
3. Starting at the same offset, read 1024 blocks.
4. **if** latency is fixed **then** cleaning is aggressive **else** cleaning is lazy.

Figure 10 shows the read latency graph of step 3 from Test 3 at an offset of 2.5 TB, with a three second pause in step 2. For all drives, offsets were chosen to land within a single band (§ 4.7). After a pause the top two emulated drives continue to show fragmented read behavior, indicating lazy cleaning, while in Emulated-SMR-3 and Seagate-SMR reads are no longer fragmented, indicating aggressive cleaning.

Figure 11 shows the Seagate-SMR head position during the 3.5 second period starting at the beginning of step 2. Two short seeks from the OD to the ID and back are seen in the first 200 ms; their purpose is not known. The RMW operation for cleaning a band starts at 1,242 ms after the last write, when the head seeks to the band at 2.5 TB offset, reads for 180 ms and seeks back to the cache at the OD where it spends 1,210 ms. We believe this time is spent

[1] Test performed with volatile cache enabled with `hdparm -W1`.

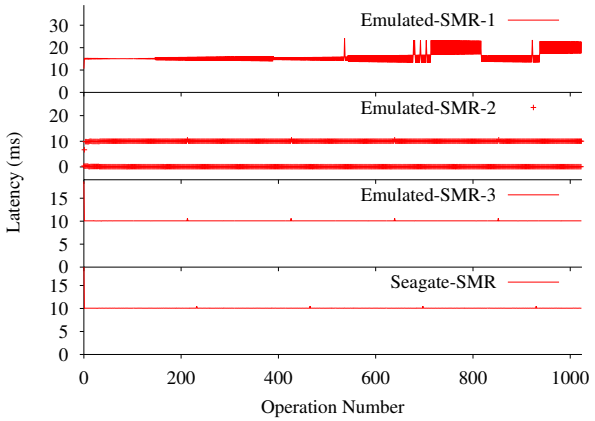

Figure 10: Discovering cleaning type.

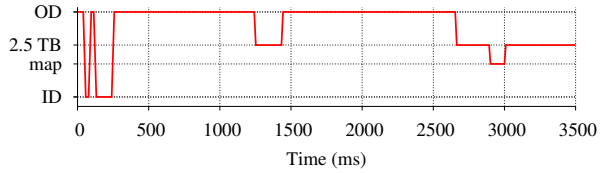

Figure 11: Seagate-SMR head position during the 3.5 second period starting at the beginning of step 2 of Test 3.

forming an updated band and persisting it to the disk cache, to protect against power failure during band overwrite. Next, the head seeks to the band, taking 227 ms to overwrite it and then seeks to the center to update the map. Hence, cleaning a band with half of its content overwritten takes $\approx$1.6 s. We believe the center to contain the map because the head always moves to this position after performing a RMW, and stays there for a short period before eventually parking at the ID. At 3 seconds reads begin and the head seeks back to the band location, where it stays until reads complete (only the first 500 ms is seen in Figure 11).

We confirmed that the operation starting at 1,242 ms is indeed an RMW: when step 3 is begun before the entire cleaning sequence has completed, read behavior is unchanged from Test 2. We did not explore the details of the RMW; alternatives like *partial read-modify-write* [38] may also have been used.

4.5.1 Seagate-SMR Cleaning Algorithm

We next start exploring performance-relevant details that are specific to the Seagate-SMR cleaning algorithm, by running Test 4. In step 1, as the drive receives random writes, it sequentially logs them to the persistent cache as they arrive. Therefore, immediately reading the blocks back in the written order should result in a fixed rotational delay with no seek time. During the pause in step 3, cleaning process moves the blocks from the persistent cache to their native locations. As a result,

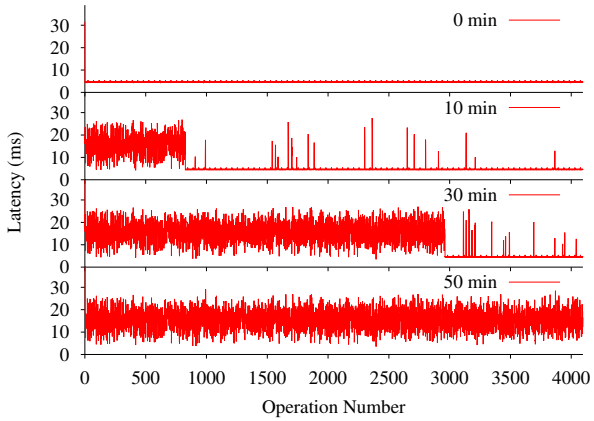

Figure 12: Latency of reads of random writes immediately after the writes and after 10–20 minute pauses.

reading after the pause should incur varying seek time and rotational delay for the blocks moved by the cleaning process, whereas unmoved blocks should still incur a fixed latency.

Test 4: Exploring Cleaning Algorithm

1. Write 4096 random blocks.
2. Read back the blocks in the written order.
3. Pause for 10–20 minutes.
4. Repeat steps 2 and 3.

In Figure 12 read latency is shown immediately after step 2, and then after 10, 30, and 50 minutes. We observe that the latency is fixed when we read the blocks immediately after the writes. If we re-read the blocks after a 10-minute pause, we observe random latencies for the first $\approx$800 blocks, indicating that the cleaning process has moved these blocks to their native locations. Since every block is expected to be on a different band, the number of operations with random read latencies after each pause shows the progress of the cleaning process, that is, the number of bands it has cleaned. Given that it takes $\approx$30 minutes to clean $\approx$3,000 bands, it takes $\approx$600 ms to clean a band whose single block has been overwritten. We also observe a growing number of cleaned blocks in the unprocessed region (for example, operations 3,000–4,000 in the 30 minute graph); based on this behavior, we hypothesize that cleaning follows Algorithm 1.

Algorithm 1: Hypothesized Cleaning Algorithm of Seagate-SMR

1. Read the next block from the persistent cache, find the block's band.
2. Scan the persistent cache identifying blocks belonging to the band.
3. Read-modify-write the band, update the map.

To test this hypothesis we run Test 5. In Figure 13 we see that after one minute, all of the blocks written in step 1, some of those written in step 2, and all of those written in step 3 have been cleaned, as indicated by non-uniform

Test 5: Verifying the Hypothesized Cleaning Algorithm

1. Write 128 blocks from a 256 MiB linear region in random order.
2. Write 128 random blocks across the LBA space.
3. Repeat step 1, using different blocks.
4. Pause for one minute; read all blocks in the written order.

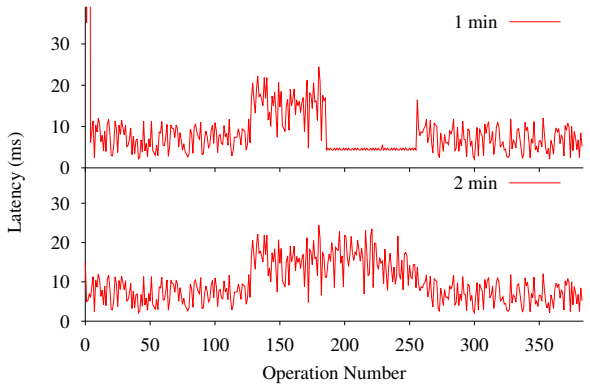

Figure 13: Verifying hypothesized cleaning algorithm on Seagate-SMR.

latency, while the remainder of step 2 blocks remain in cache, confirming our hypothesis. After two minutes all blocks have been cleaned. (The higher latency for step 2 blocks is due to their higher mean seek distance.)

4.6 Persistent Cache Size

We discover the size of the persistent cache by ensuring that the cache is empty and then measuring how much data may be written before cleaning begins. We use random writes across the LBA space to fill the cache, because sequential writes may fill the drive bypassing the cache [20] and cleaning may never start. Also, with sequential writes, a drive with multiple caches may fill only one of the caches and start cleaning before all of the caches are full [20]. With random writes, bypassing the cache is not possible; also, they will fill multiple caches at the same rate and start cleaning when all of the caches are almost full.

The simple task of filling the cache is complicated in drives using extent mapping: a cache is considered full when the extent map is full or when the disk cache is full, whichever happens first. The latter is further complicated by journal entries with quantized sizes—as seen previously (§ 4.3.1), a single 4 KB write may consume as much cache space as dozens of 8 KB writes. Due to this overhead, actual size of the disk cache is larger than what is available to host writes—we differentiate the two by calling them *persistent cache **raw** size* and *persistent cache size*, respectively.

Figure 14 shows three possible scenarios on a hypothetical drive with a persistent cache raw size of 36 blocks and a 12 entry extent map. The minimum journal entry size is 2 blocks, and it grows in units of 2 blocks to the maximum of 16 blocks; out-of-band data of 2 blocks is written with every journal entry; the persistent cache size is 32 blocks.

Part (a) of Figure 14 shows the case of queue depth 1 and 1-block writes. After the host issues 9 writes, the drive puts every write to a separate 2-block journal entry, fills the cache with 9 journal entries and starts cleaning. Every write consumes a slot in the map, shown by the arrows. Due to low queue depth, the drive leaves one empty block in each journal entry, wasting 9 blocks. Exploiting this behavior, Test 6 discovers the persistent cache raw size. In this and the following tests, we detect the start of cleaning by the drop of the IOPS to near zero.

Test 6: Discovering Persistent Cache Raw Size

1. Write with a small size and low queue depth until cleaning starts.
2. Persistent cache raw size = number of writes × (minimum journal entry size + out-of-band data size).

Part (b) of Figure 14 shows the case of queue depth 4 and 1-block writes. After the host issues 12 writes, the drive forms three 4-block journal entries. Writing these journal entries to the cache fills the map and the drive starts cleaning despite a half-empty cache. We use Test 7 to discover the *persistent cache map size*.

Test 7: Discovering Persistent Cache Map Size

1. Write with a small size and high queue depth until cleaning starts.
2. Persistent cache map size = number of writes.

Finally, part (c) of Figure 14 shows the case of queue depth 4 and 4-block writes. After the host issues 8 writes, the drive forms two 16-block journal entries, filling the cache. Due to high queue depth and large write size, the drive is able to fill the cache (without wasting any blocks) before the map fills. We use Test 8 to discover the *persistent cache size*.

Test 8: Discovering Persistent Cache Size

1. Write with a large size and high queue depth until cleaning starts.
2. Persistent cache size = total host write size.

Table 2 shows the result of the tests on Seagate-SMR. In the first row, we discover persistent cache raw size using Test 6. Writing with 4 KiB size and queue depth of 1 produces constant 25 ms latency (§ 4.3), that is 2.5 rotations. Track size is ≈ 2 MiB at the OD, therefore, 22,800 operations correspond to ≈ 100 GiB.

In rows 2 and 3 we discover the persistent cache map size using Test 7. For write sizes of 4 KiB and 64 KiB cleaning starts after ≈ 182,200 writes, which corresponds to 0.7 GiB and 11.12 GiB of host writes, respectively. This confirms that in both cases the drive hits the map size limit, corresponding to scenario (b) in Figure 14. Assuming that the drive uses

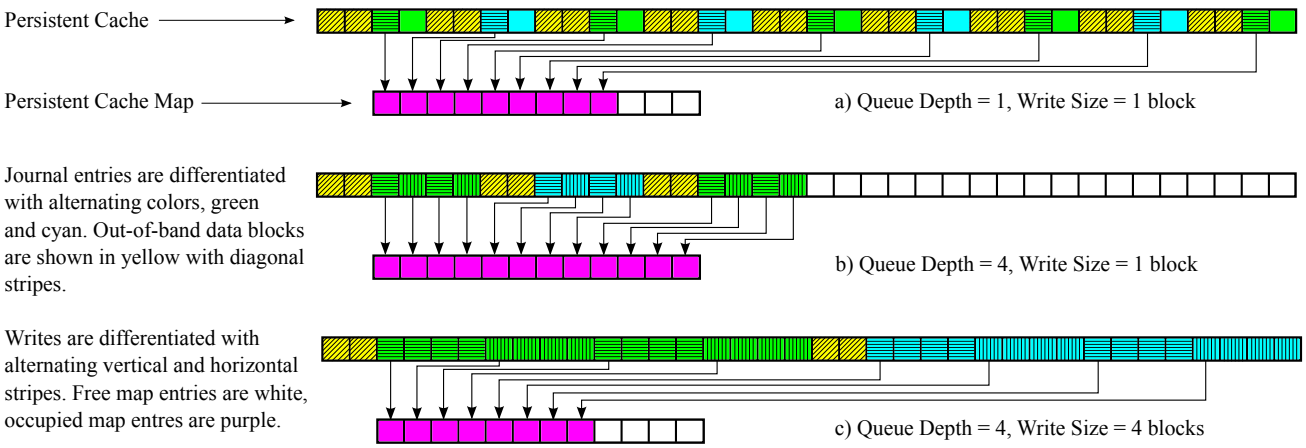

Figure 14: Three different scenarios triggering cleaning on drives using journal entries with quantized sizes and extent mapping. The text on the left explains the meaning of the colors.

Drive	Write Size	QD	Operation Count	Host Writes	Internal Writes
	4 KiB	1	22,800	89 MiB	**100 GiB**
	4 KiB	31	**182,270**	0.7 GiB	N/A
Sea-SMR	64 KiB	31	**182,231**	11.12 GiB	N/A
	128 KiB	31	137,496	**16.78 GiB**	N/A
	256 KiB	31	67,830	**16.56 GiB**	N/A
Em-SMR-1	4 KiB	1	9,175,056	35 GiB	35 GiB
Em-SMR-2	4 KiB	1	2,464,153	9.4 GiB	9.4 GiB
Em-SMR-3	4 KiB	1	9,175,056	35 GiB	35 GiB

Table 2: Discovering persistent cache parameters.

a low watermark to trigger cleaning, we estimate that the map size is 200,000 entries.

In rows 4 and 5 we discover the persistent cache size using Test 8. With 128 KiB writes we write ≈17 GiB in fewer operations than in row 3, indicating that we are hitting the size limit. To confirm this, we increase write size to 256 KiB in row 5; as expected, the number of operations drops by half while the total write size stays the same. Again, assuming that the drive has hit the low watermark, we estimate that the persistent cache size is 20 GiB.

Journal entries with quantized sizes and extent mapping are absent topics in academic literature on SMR, so emulated drives implement neither feature. Running Test 6 on emulated drives produces all three answers, since in these drives, the cache is block-mapped, and the cache size and cache raw size are the same. Furthermore, set-associative STL divides the persistent cache into cache bands and assigns data bands to them using modulo arithmetic. Therefore, despite having a single cache, under random writes it behaves similarly to a fully-associative cache. The bottom rows of Table 2 show that in emulated drives, Test 8 discovers the cache size (see Table 1) with 95% accuracy.

4.7 Band Size

STLs proposed to date [15, 20, 31] clean a single band at a time, by reading unmodified data from a band and updates from the cache, merging them, and writing the merge result back to a band. Test 9 determines the band size, by measuring the granularity at which this cleaning process occurs.

Test 9: Discovering the Band Size

1. Select an accuracy granularity a, and a band size estimate b.
2. Choose a linear region of size $100 \times b$ and divide it into a-sized blocks.
3. Write 4 KiB to the beginning of every a-sized block, in random order.
4. Force cleaning to run for a few seconds and read 4 KiB from the beginning of every a-sized block in sequential order.
5. Consecutive reads with identical high latency identify a cleaned band.

Assuming that the linear region chosen in Test 9 lies within a region of equal track length, for data that is not in the persistent cache, 4 KB reads at a fixed stride a should see identical latencies—that is, a rotational delay equivalent to $(a \bmod T)$ bytes where T is the track length. Conversely reads of data from cache will see varying delays in the case of a disk cache due to the different (and random) order in which they were written or sub-millisecond delays in the case of a flash cache.

With aggressive cleaning, after pausing to allow the disk to clean a few bands, a linear read of the written blocks will identify the bands that have been cleaned. For a drive with lazy cleaning the linear region is chosen so that writes fill the persistent cache and force a few bands to be cleaned, which again may be detected by a linear read of the written data.

In Figure 15 we see the results of Test 9 for $a = 1$ MiB and $b = 50$ MiB, respectively, with the region located at the 2.5 TB offset; for each drive we zoom in to show an individual band that has been cleaned. We correctly identify the band size for the emulated drives (see Table 1). The band size of Seagate-SMR at this location is seen to be 30 MiB; running tests at different offsets shows that bands are iso-capacity within a zone

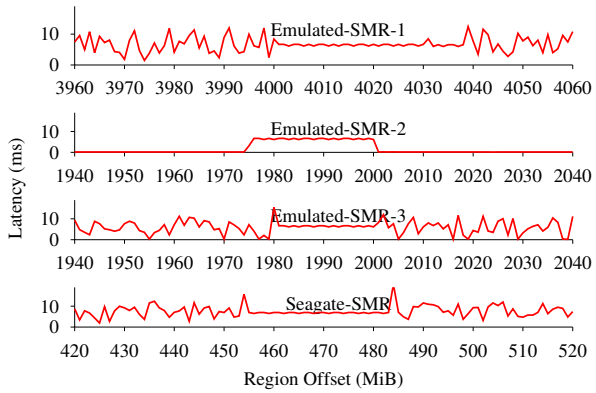

Figure 15: Discovering band size.

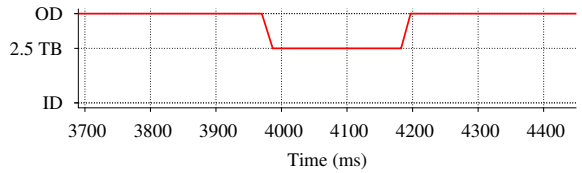

Figure 16: Head position during the sequential read for Seagate-SMR, corresponding to the time period in Figure 15.

(§ 4.9) but vary from 36 MiB at the OD to 17 MiB at the ID.

Figure 16 shows the head position of Seagate-SMR corresponding to the time period in Figure 15. It shows that the head remains at the OD during the reads from the persistent cache up to 454 MiB, then seeks to 2.5 TB offset and stays there for 30 MiB, and then seeks back to the cache at OD, confirming that the blocks in the band are read from their native locations.

4.8 Block Mapping

Once we discover the band size (§ 4.7), we can use Test 10 to determine the mapping type. This test exploits varying inter-track switching latency between different track pairs to detect if a band was remapped. After overwriting the first two tracks of band *b*, cleaning will move the band to its new location—a different physical location only if dynamic mapping is used. Plotting latency graphs of step 2 and step 4 will produce the same pattern for the static mapping and a different pattern for the dynamic mapping.

Adapting this test to a drive with lazy cleaning involves some extra work. First, we should start the test on a drive after a secure erase, so that the persistent cache is empty. Due to lazy cleaning, the graph of step 4 will be the graph of switching between a track and the persistent cache. Therefore, we will fill the cache until cleaning starts, and repeat step 2 once in a while, comparing its graph to the previous two: if it is similar to the last, then data is still in the cache, if it is similar to the first, then the drive uses static mapping, otherwise, the drive uses dynamic mapping.

Test 10: Discovering mapping type.

1. Choose two adjacent iso-capacity bands a and b; set n to the number of blocks in a track.
2. **for** $i \leftarrow 0$ **to** $i < 2$ **do**
   for $j \leftarrow 0$ **to** $j < n$ **do**
   Read block j of track 0 of band a
   Read block j of track i of band b
3. Overwrite the first two tracks of band b; force cleaning to run.
4. Repeat step 2.

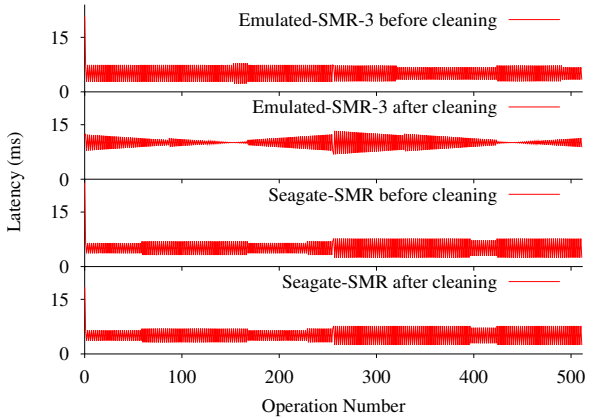

Figure 17: Mapping Type Detection.

We used track and block terms to concisely describe the algorithm above, but the size chosen for these algorithmic parameters need not match track size and block size of the underlying drive. Figure 17, for example, shows the plots for the test on Emulated-SMR-3 and Seagate-SMR, using 2 MiB for the track size and 16 KiB for the block size. The latency pattern for the Seagate-SMR does not change, indicating a static mapping, but it changes for Emulated-SMR-3, which indeed uses dynamic mapping. We omit the graphs of the remaining drives to save space.

4.9 Zone Structure

We use sequential reads (Test 11) to discover the zone structure of Seagate-SMR. While there are no such drives yet, on drives with dynamic mapping a secure erase that would restore the mapping to the default state may be necessary for this test to work. Figure 18 shows the zone profile of Seagate-SMR, with a zoom to the beginning.

Test 11: Discovering Zone Structure

1. Enable kernel read-ahead and drive look-ahead.
2. Sequentially read the whole drive in 1 MiB blocks.

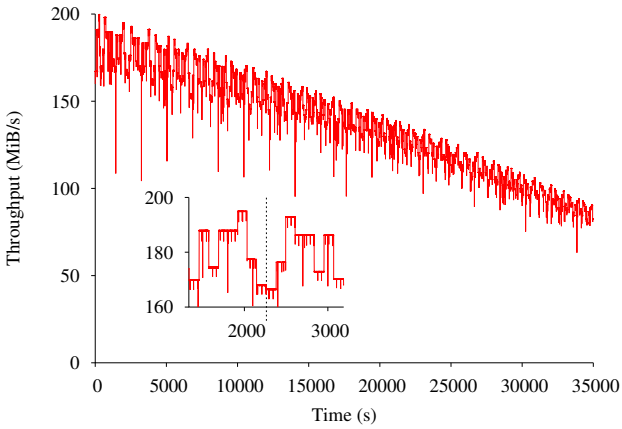

Figure 18: Sequential read throughput of Seagate-SMR.

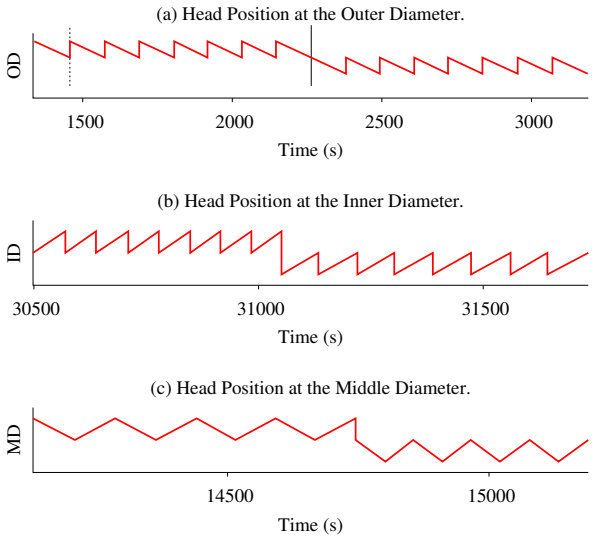

Figure 19: Seagate-SMR head position during sequential reads at different offsets.

Similar to CMR drives, the throughput falls as we reach higher LBAs; unlike CMR drives, there is a pattern that repeats throughout the graph, shown by the zoomed part. This pattern has an axis of symmetry indicated by the dotted vertical line at 2,264th second. There are eight distinct plateaus to the left and to the right of the axis with similar throughputs. The fixed throughput in a single plateau and a sharp change in throughput between plateaus suggest a wide radial stroke and a head switch. Plateaus corresponds to large zones of size 18–20 GiB, gradually decreasing to 4 GiB as we approach higher LBAs. The slight decrease in throughput in symmetric plateaus on the right is due to moving from a larger to smaller radii, where sector per track count decreases; therefore, throughput decreases as well.

We confirmed these hypotheses using the head position graph shown in Figure 19 (a), which corresponds to the time interval of the zoomed graph of Figure 18. Unlike with CMR drives, where we could not observe head switches due to narrow radial strokes, with this SMR drive head switches are visible to an unaided eye. Figure 19 (a) shows that the head starts at the OD and slowly moves towards the MD completing this inwards move at 1,457th second, indicated by the vertical dotted line. At this point, the head has just completed a wide radial stroke reading gigabytes from the top surface of the first platter, and it performs a jump back to the OD and starts a similar stroke on the bottom surface of the first platter. The direction of the head movement indicates that the shingling direction is towards the ID at the OD. The head completes the descent through the platters at 2,264th second—indicated by the vertical solid line—and starts its ascent reading surfaces in the reverse order. These wide radial strokes create "horizontal zones" that consist of thousands of tracks on the same surface, as opposed to "vertical zones" spanning multiple platters in CMR drives. We expect these horizontal zones to be the norm in SMR drives, since they facilitate SMR mechanisms like allocation of iso-capacity bands, static mapping, and dynamic band size adjustment [35]. Figure 19 (b) corresponds to the end of Figure 18, shows that the direction of the head movement is reversed at the ID, indicating that both at the OD and at the ID, shingling direction is towards the middle diameter. To our surprise, Figure 19 (c) shows that a conventional serpentine layout with wide serpents is used at the MD. We speculate that although the whole surface is managed as if it is shingled, there is a large region in the middle that is not shingled.

5 Related Work

Little has been published on the subject of system-level behavior of SMR drives. Although several works (for example, Amer et al. [15] and Le et al. [39]) have discussed requirements and possibilities for use of shingled drives in systems, only three papers to date—Cassuto et al. [20], Lin et al. [40], and Hall et al. [21]—present example translation layers and simulation results. A range of STL approaches is found in the patent literature [27, 31, 35, 41], but evaluation and analysis is lacking. Several SMR-specific file systems have been proposed, such as SMRfs [14], SFS [18], and HiSMRfs [42]. He and Du [43] propose a static mapping to minimize re-writes for in-place updates, which requires high guard overhead (20%) and assumes file system free space is contiguous in the upper LBA region. Pitchumani et al. [32] present an emulator implemented as a Linux device mapper target that mimics shingled writing on top of a CMR drive. Tan et al. [44] describe a simulation of S-blocks algorithm, with a more accurate simulator calibrated with data from a real CMR drive. To date no work (to the authors' knowledge) has presented measurements of read and write operations on an SMR drive, or performance-accurate emulation of STLs.

This work draws heavily on earlier disk characterization

Property	Drive Model	
	ST5000AS0011	ST8000AS0011
Drive Type	SMR	SMR
Persistent Cache Type	Disk	Disk
Cache Layout and Location	Single, at the OD	Single, at the OD
Cache Size	20 GiB	25 GiB
Cache Map Size	200,000	250,000
Band Size	17–36 MiB	15–40 MiB
Block Mapping	Static	Static
Cleaning Type	Aggressive	Aggressive
Cleaning Algorithm	FIFO	FIFO
Cleaning Time	0.6–1.6 s/band	0.6–1.6 s/band
Zone Structure	4–20 GiB	5–40 GiB
Shingling Direction	Towards MD	N/A

Table 3: Properties of the 5 TB and the 8 TB Seagate drives discovered using Skylight methodology. The benchmarks worked out of the box on the 8 TB drive. Since the 8 TB drive was on loan, we did not drill a hole on it; therefore, shingling direction for it is not available.

studies that have used micro-benchmarks to elicit details of internal performance, such as Schlosser et al. [45], Gim et al. [26], Krevat et al. [46], Talagala et al. [25], Worthington et al. [24]. Due to the presence of a translation layer, however, the specific parameters examined in this work (and the micro-benchmarks for measuring them) are different.

6 Conclusions and Recommendations

As Table 3 shows, the Skylight methodology enables us to discover key properties of two drive-managed SMR disks automatically. With manual intervention, it allows us to completely reverse engineer a drive. The purpose of doing so is not just to satisfy our curiosity, however, but to guide both their use and evolution. In particular, we draw the following conclusions from our measurements of the 5 TB Seagate drive:

1. Write latency with the volatile cache disabled is high (Test 1). This appears to be an artifact of specific design choices rather than fundamental requirements, and we hope for it to drop in later firmware revisions.
2. Sequential throughput (with the volatile cache disabled) is much lower (by 3× or more, depending on write size) than for conventional drives. (We omitted these test results, as performance is identical to the random writes in Test 1.) Due to the use of static mapping (Test 10), achieving full sequential throughput requires enabling volatile cache.
3. Random I/O throughput (with the volatile cache enabled or with high queue depth) is high (Test 7)—15× that of the equivalent CMR drive. This is a general property of any SMR drive using a persistent cache.
4. Throughput may degrade precipitously when the cache fills after many writes (Table 2). The point at which this occurs depends on write size and queue depth[2].
5. Background cleaning begins after ≈1 second of idle time, and proceeds in steps requiring 0.6–1.6 seconds of uninterrupted idle time to clean a single band. The duration of the step depends on the amount of data updated in the band. Cleaning a band whose single block was overwritten may take 0.6 seconds (Figure 12), whereas cleaning a band with half of its content overwritten may take 1.6 seconds (Figure 11). The number of the steps required is proportional to the number of bands—contiguous regions of 15–40 MB (§ 4.7)—that have been modified.
6. Sequential reads of randomly-written data will result in random-like read performance until cleaning completes (§ 4.4).

In summary, SMR drives like the ones we studied should offer good performance if the following conditions are met: (a) the volatile cache is enabled or a high queue depth is used, (b) writes display strong spatial locality, modifying only a few bands at any particular time, (c) non-sequential writes (or all writes, if the volatile cache is disabled) occur in bursts of less than 16 GB or 180,000 operations (Table 2), and (d) long powered-on idle periods are available for background cleaning. From the use of aggressive cleaning that presumes long idle periods, we may conclude that the drive is adapted to desktop use, but may perform poorly on server workloads. Further work will include investigation of STL algorithms that may offer a better balance of performance for both.

Acknowledgments

This work was supported by NetApp, and NSF award CNS-1149232. We thank the anonymous reviewers, Remzi Arpaci-Dusseau, Tim Feldman, and our shepherd, Kimberly Keeton, for their feedback.

References

[1] Seagate Technology PLC Fiscal Fourth Quarter and Year End 2013 Financial Results Supplemental Commentary, July 2013. Available from http://www.seagate.com/investors.

[2] Drew Riley. Samsung's SSD Global Summit: Samsung: Flexing Its Dominance In The NAND Market, August 2013.

[3] DRAMeXchange. NAND Flash Spot Price, September 2014. http://dramexchange.com.

[2] Although results with the volatile cache enabled are not presented in § 4.6, they are similar to those for a queue depth of 31.

[4] S. N. Piramanayagam. Perpendicular recording media for hard disk drives. *Journal of Applied Physics*, 102(1):011301, July 2007.

[5] Terascale HDD. Data sheet DS1793.1-1306US, Seagate Technology PLC, June 2013.

[6] D.A Thompson and J.S. Best. The future of magnetic data storage techology. *IBM Journal of Research and Development*, 44(3):311–322, May 2000.

[7] R. Wood, Mason Williams, A Kavcic, and Jim Miles. The Feasibility of Magnetic Recording at 10 Terabits Per Square Inch on Conventional Media. *IEEE Transactions on Magnetics*, 45(2):917–923, February 2009.

[8] Seagate Desktop HDD: ST5000DM000, ST4000DM001. Product Manual 100743772, Seagate Technology LLC, December 2013.

[9] Seagate Ships Worlds First 8TB Hard Drives, August 2014. Available from http://www.seagate.com/about/newsroom/.

[10] HGST Unveils Intelligent, Dynamic Storage Solutions To Transform The Data Center, September 2014. Available from http://www.hgst.com/press-room/.

[11] M.H. Kryder, E.C. Gage, T.W. McDaniel, W.A Challener, R.E. Rottmayer, Ganping Ju, Yiao-Tee Hsia, and M.F. Erden. Heat Assisted Magnetic Recording. *Proceedings of the IEEE*, 96(11):1810–1835, November 2008.

[12] Elizabeth A Dobisz, Z.Z. Bandic, Tsai-Wei Wu, and T. Albrecht. Patterned Media: Nanofabrication Challenges of Future Disk Drives. *Proceedings of the IEEE*, 96(11):1836–1846, November 2008.

[13] Sumei Wang, Yao Wang, and R.H. Victora. Shingled Magnetic Recording on Bit Patterned Media at 10 Tb/in^2. *IEEE Transactions on Magnetics*, 49(7):3644–3647, July 2013.

[14] Garth Gibson and Milo Polte. Directions for Shingled-Write and Two-Dimensional Magnetic Recording System Architectures: Synergies with Solid-State Disks. Technical Report CMU-PDL-09-104, CMU Parallel Data Laboratory, May 2009.

[15] Ahmed Amer, Darrell D. E. Long, Ethan L. Miller, Jehan-Francois Paris, and S. J. Thomas Schwarz. Design Issues for a Shingled Write Disk System. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.

[16] Garth Gibson and Greg Ganger. Principles of Operation for Shingled Disk Devices. Technical Report CMU-PDL-11-107, CMU Parallel Data Laboratory, April 2011.

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

[18] D. Le Moal, Z. Bandic, and C. Guyot. Shingled file system host-side management of Shingled Magnetic Recording disks. In *Proceedings of the 2012 IEEE International Conference on Consumer Electronics (ICCE)*, pages 425–426, January 2012.

[19] INCITS T10 Technical Committee. Information technology - Zoned Block Commands (ZBC). Draft Standard T10/BSR INCITS 536, American National Standards Institute, Inc., September 2014. Available from http://www.t10.org/drafts.htm.

[20] Yuval Cassuto, Marco A. A. Sanvido, Cyril Guyot, David R. Hall, and Zvonimir Z. Bandic. Indirection Systems for Shingled-recording Disk Drives. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–14, Washington, DC, USA, 2010. IEEE Computer Society.

[21] 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.

[22] Luc Bouganim, Bjorn Jnsson, and Philippe Bonnet. uFLIP: understanding flash IO patterns. In *Proceedings of the Int'l Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, California, 2009.

[23] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 181–192, New York, NY, USA, 2009. ACM.

[24] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, and John Wilkes. On-line Extraction of SCSI Disk Drive Parameters. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '95/PERFORMANCE '95, pages 146–156, New York, NY, USA, 1995. ACM.

[25] Nisha Talagala, Remzi H. Arpaci-Dusseau, and D. Patterson. Microbenchmark-based Extraction of

Local and Global Disk Characteristics. Technical Report UCB/CSD-99-1063, EECS Department, University of California, Berkeley, 1999.

[26] Jongmin Gim and Youjip Won. Extract and infer quickly: Obtaining sector geometry of modern hard disk drives. *ACM Transactions on Storage (TOS)*, 6(2):6:1–6:26, July 2010.

[27] Jonathan Darrel Coker and David Robison Hall. Indirection memory architecture with reduced memory requirements for shingled magnetic recording devices, November 5 2013. US Patent 8,578,122.

[28] Linux Device-Mapper. Device-Mapper Resource Page. https://sourceware.org/dm/, 2001.

[29] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-structured File System. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, SOSP '91, pages 1–15, New York, NY, USA, 1991. ACM.

[30] Serial ATA International Organization. Serial ATA Revision 3.1 Specification. Technical report, Serial ATA International Organization, July 2011.

[31] David Robison Hall. Shingle-written magnetic recording (SMR) device with hybrid E-region, April 1 2014. US Patent 8,687,303.

[32] Rekha Pitchumani, Andy Hospodor, Ahmed Amer, Yangwook Kang, Ethan L. Miller, and Darrell D. E. Long. Emulating a Shingled Write Disk. In *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '12, pages 339–346, Washington, DC, USA, 2012. IEEE Computer Society.

[33] Tim Feldman. Personal communication, August 2014.

[34] Jens Axboe. Flexible I/O Tester. git://git.kernel.dk/fio.git.

[35] Timothy Richard Feldman. Dynamic storage regions, February 14 2011. US Patent App. 13/026,535.

[36] Libata FAQ. https://ata.wiki.kernel.org/index.php/Libata_FAQ.

[37] Tim Feldman. Host-Aware SMR. OpenZFS Developer Summit, November 2014. Available from https://www.youtube.com/watch?v=b1yqjV8qemU.

[38] Sundar Poudyal. Partial write system, March 13 2013. US Patent App. 13/799,827.

[39] Quoc M. Le, Kumar SathyanarayanaRaju, Ahmed Amer, and JoAnne Holliday. Workload Impact on Shingled Write Disks: All-Writes Can Be Alright. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '11, pages 444–446, Washington, DC, USA, 2011. IEEE Computer Society.

[40] Chung-I Lin, Dongchul Park, Weiping He, and David H. C. Du. H-SWD: Incorporating Hot Data Identification into Shingled Write Disks. In *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '12, pages 321–330, Washington, DC, USA, 2012. IEEE Computer Society.

[41] Robert M Fallone and William B Boyle. Data storage device employing a run-length mapping table and a single address mapping table, May 14 2013. US Patent 8,443,167.

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

[43] Weiping He and David H. C. Du. Novel Address Mappings for Shingled Write Disks. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, pages 5–5, Berkeley, CA, USA, 2014. USENIX Association.

[44] S. Tan, W. Xi, Z.Y. Ching, C. Jin, and C.T. Lim. Simulation for a Shingled Magnetic Recording Disk. *IEEE Transactions on Magnetics*, 49(6):2677–2681, June 2013.

[45] Steven W. Schlosser, Jiri Schindler, Stratos Papadomanolakis, Minglong Shao, Anastassia Ailamaki, Christos Faloutsos, and Gregory R. Ganger. On Multi-dimensional Data and Modern Disks. In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4*, FAST'05, pages 17–17, Berkeley, CA, USA, 2005. USENIX Association.

[46] Elie Krevat, Joseph Tucek, and Gregory R. Ganger. Disks Are Like Snowflakes: No Two Are Alike. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.