Next-generation *Magnetic Recording*

Shingled magnetic recording is a low-cost way to increase the areal density of hard disk drives (it is low-cost in the *dollar* sense because it does not noticeably change the drives' manufacturing processes).

The analogy that we see repeated in all SMR descriptions is that SMR tracks are overlapped "like shingles on a roof". Traditionally, HDD density gains have been achieved by reducing both the reader and writer gap sizes. However, we have neared the physical limits of our ability to further reduce the *writer gap width* below its current size (due to the "superparamagnetic limit"), while the *reader gap width* can be further reduced. SMR exploits this asymmetry.

- Consecutive shingled tracks are always written in sequential "downstream" order. For any two tracks *T* and *T+1*, track *T* must be written in its entirety before track *T+1* can be written. Once writing track *T+1* has begun, track *T* cannot be modified.
- Track *T+1* partially overlaps track *T*, but when track *T+1* is written, enough of track *T* is left un-obstructed so that track *T*'s data can still be read. This is possible because the reader gap width is smaller than the writer gap width.

One result of partially overlapping tracks is that any write will clobbers some range of "downstream" data. This limits an SMR drive's support for

- random writes
- overwrites

Types of SMR

- **Host-managed SMR** drives expose important details about the drive's internal organization, giving the user control over data management decisions.
- **Drive-managed SMR** drives do not expose any details about their internal organization. Instead, a drive-managed SMR drive exposes the standard block interface, much like an SSD.

Both types of SMR require *someone* to implement a logic to manage data so that it is not lost. This is often done by maintaining a mapping from the logical blocks to physical blocks in a translation layer (like FTLs on SSDs). At a high level, there are two considerations for implementing an SMR translation solution:

- The drive layout
- The mapping scheme

SMR Drive layout

- The disk is divided into contiguous regions of overlapping tracks called *zones* (sometimes referrred to as *bands*, but after the initial wave of SMR papers, the community has more or less settled on zone). The last track in a zone does not have any tracks that overlap it, and it is often called a *guard track*.

- An SMR disk often has one or more regions where random reads and writes are legal (i.e., no tracks are overlapped). This region is often used as a *persistent cache*.

Zones are typically *O(hundreds of MiB)*. 256MiB is a common zone size among drives I have seen/used.

The persistent cache is typically *O(ones to tens of GiB)*.

Managing LBA -> PBA Mappings

Logical to physical block mappings can be *static* or *dynamic*.

- A static mapping is one where each LBA corresponds to a specific PBA (or a fixed number of PBAs). Static mappings typically require that updates to a PBA be made by reading an entire zone, modifying the zone in memory, and writing the whole contents of the zone as a unit.
- Dynamic mappings can take many forms. In my opinion, this is where the exciting research lies because there is no one "best" approach.

Drive-managed SMR drives typically employ a static mapping, and a persistent cache is used to buffer updates.

Host-managed Drives and Other Interfaces

Host-managed SMR drives are typically accessed using ZBC/ZAC interface (Zoned Block Commands/Zoned-device ATA Commands). These interfaces give the software access to a few key structures.

- The drive may have one or more *conventional zones*. These zones may be randomly written/updated as if it were a "conventional" hard drive.
- There are one or more *sequential write required* zones. These zones must be written sequentially.
    - Seq-write required zones have a "write pointer", and all writes must be performed at the current location specified by the write pointer. Appending at the write pointer advances its value.
    - Seq-write required zones may have their write pointers "reset" to the start of the zone. You cannot "paritally" reset a zone's write pointer.
    - Seq-write required zones are somewhat similar to an SSD's write/erase block: you can write to any page oncec, and you can reset an entire block.
- There are also interfaces to query the status of zones, the layout of the drive, and various other statistics.

One of the readings described an alternate interface called *caveat scriptor*. In this interface, you can write anywhere on the disk. However, doing so "clobbers" downstream data, rendering the disk's contents on subsequent tracks unreliable. The disk provides general guidelines so that the impact of writes can be estimated, but ultimately "writer beware"!

SMR Questions

If you squint your eyes, the "SMR problem" is just like the "SSD problem". However, there are several differences that impact the designs that we can expect solutions to employ.

1. What are the physical differences between SMR drives and SSDs?
2. What are the target use cases for SMR drives and SSDs?
3. What are the scales of SMR drives and SSDs (in terms of capacity)? You may wish to Google for some high-capacity drives of each type to estimate what is out there.
4. Are there concerns that an FTL must deal with that an SMR drive does not (or vice versa)?
5. What FTL approaches did we discuss that would work well on an SMR drive? Would not work well?
6. Where do you think HM-SMR drives will be used (if at all) moving forward? DM-SMR drives?

Interlaced Magnetic Recording (IMR)

In addition to the SMR hard disk drives that are currently in production (and many DM-SMR drives are widely available for purchase, whether you know it or not), IMR hard disk drives are "on the horizon". IMR stands for Interlaced Magnetic Recording, and the idea is somewhat similar to SMR, but with important distinctions.

- IMR drives are divided into *bottom tracks* and *top tracks*
- Each bottom track has two top tracks that partially overlap it. This creates two distinct cases:
    - To update a bottom track, the drive may need to read both top tracks, update the bottom track, and then rewrite the top tracks.
    - However, any top track may be updated without worry.

So unlike SMR, there is no notion of a "zone" or a "write pointer". However, there is no reason we couldn't retrofit that programming model onto the physical IMR layout.

IMR drives are not yet publicly available. When we do see them, I suspect that they *will* be managed using an interface similar to the "Zoned" interface that we see with HM-SMR. I posit that there is just too much overhead to developing new software that drive vendors would sacrifice backwards compatibility; they've already asked storage developers to scramble to adopt a new interface.

Next-generation SSDs

Multi-streamed SSDs

When using the traditional block interface, the FTL has no way of inferring the application's semantics: WRITE and READ give almost no information. Abstraction is a useful design principle, but it does limit our ability to optimize software across different layers because abstraction hides information. The goal of a multi-stream SSD is to provide applications a mechanism to pass *hints* to the FTL. These hints are optional, so an application can choose to ignore the stream interface entirely. However, using the stream interface may yield noticeable benefits in terms of write amplification, and as a result, performance.

The stream abstraction gives users a way to "tag" data with an identifier that the SSD can optionally use to optimize its placement and GC decisions. For example, a system may annotate data with similar lifetimes with the same tag. The SSD can then use this tag as a *hint* that it should co-locate these logical data objects in the same write-erase blocks because they will likely be accessed together or "deleted" together. Recall that the "cost-benefit" algorithm for LFS garbage collection uses data "hotness" as an input. The stream is a way for applications to directly provide the FTL a hint about hotness.

ZNS

"Zoned Namespaces" (ZNS) are an extension to the NVMe specification. ZNS lets software interact with ZNS SSDs using a similar interface that is used to interact with host-managed SMR drives:

- a ZNS SSD is divided into *zones*
    - applications can append to the write pointer in a zone, and
    - zones can be erased in their entirety. The general idea is that a rich ecosystem of host-managed SMR software already exists; why not take advantage of that ecosystem by creating a similar interface for SSDs?

The overarching goal of ZNS is to give applications a means to reduce their *write amplification* to almost zero. By exposing the zone interface, applications have complete control over two important FTL responsibilities: LBA->PBA mappings and garbage collection.

- A ZNS SSD is still responsible for error correction (typically, error correcting codes are stored in the *page private area*). So page-level error detection and correction is still implemented in firmware (and is completely hidden from us), but the ZNS specification gives drives different ways to communicate failures to the host (e.g., disable the page, disable the zone, etc.). These choices may be drive or vendor dependent.
- Good GC algorithms will also perform wear-leveling. Since this behavior is pushed to the host, ZNS SSDs give us the ability to easily experiment with new and interesting algorithms.
- Unlike SMR HDDs, ZNS SSDs have internal parallelism. So although we certainly have the opportunity to directly port software designed for a HM-SMR HDD onto a ZNS SSD, there may be missed opportunities if this is not done carefully.

The ZNS specification is still undergoing standardization by the NVMe org. Compared to SMR, ZNS is in its early stages, and there are not many examples of ZNS drives "in the wild". However, I posit that ZNS SSDs will become important in the coming years. (At least I hope so! ZNS SSDs have already been included in at least a couple of grants I've submitted...)