Western Digital.

White Paper

Implementing SMR

Exploring options for adapting existing storage applications — or deploying new ones — for use with SMR hard drives.

April 2024

Introduction

As the pace of global data creation continues to accelerate, and as the global demand for devices to store that data climbs as well, there is also a proliferation in the types of applications and usage of that data. The world's data needs are less dominated by databases and corporate email servers; newer usage models such as cloud storage, AI/ML training data sets, and smart video are driving the growth. Below them, increasing needs for nearline/cold/archival storage, whether merely to retain important data or due to regulatory requirements making long-term storage necessary, create workloads that may offer more variety in the storage devices utilized to meet that need.

During this, system designers are also faced with cost pressures, with one of the largest being total cost of ownership (TCO). They often face fixed constraints such as a limited amount of power that a local utility can supply to the datacenter, or limited square footage with which to install storage, and it becomes clear that increasing storage density is one of the key factors in reducing TCO. Higher storage density can both minimize the amount of power (OpEx) and rack space / equipment (CapEx), making the storage as economically efficient as possible.

The proliferation of new applications and the development of shingled magnetic recording (SMR) HDDs can give system designers a new method to increase storage density to meet their storage demand while optimizing TCO. However, SMR HDDs have historically not been a "drop-in" replacement for conventional HDDs, and applications have required significant host software changes to support SMR. The software ecosystem required time to mature and develop support for these changes and is today now well-prepared to enable a much wider range of SMR-compliant applications than in the past.

What is SMR?

HDDs write and read data via the “head” of the drive, flying above the magnetic disk media. Each head will have two components; the write head and the read head. The write head must develop a very strong magnetic field to change the bits on the media, whereas the read head merely senses the magnetic field as the head flies over the spinning media. Based on the tasks each head to achieve its designed purpose, the write head happens to be physically wider than the read head. This disparity is exploited to introduce SMR.

In conventional magnetic recording (CMR), the data is organized in discrete tracks, separated by narrow guard band between them. This allows for random write and rewrite of every sector on the HDD, without corrupting the existing written data on adjacent tracks. However, because the write head is wider than the read head, this track is wider than it needs to be to be successfully read.

This, plus the additional guard band needed between tracks, wastes space and limits the areal density of the disk.

In SMR HDDs, these written tracks are overlapped. Because the track does not need to be the full width of the write head, and because a guard band is not needed, it reduces wasted space and allows for significantly higher areal densities. But this introduces a constraint—data cannot be written randomly without corrupting adjacent tracks. Much like the shingles on a roof, to change the data on one track, the drive must rewrite all the preceding tracks to change one sector.

This introduces a sequential write requirement. To ensure that changing one sector doesn't require a complete overwrite of the *entire* HDD, the drive is divided into smaller zones such that only an individual zone needs to be rewritten, as shown in Figure 1. However, this is still a time-consuming process, so host systems must be optimized to avoid this to still be performant. Note that read performance between CMR and SMR HDDs is similar; it is only writes that are subject to these differences.

For decades, application software, operating systems, and file systems have been built around the ability that an HDD can *randomly* write data. In enterprise storage systems, to avoid confusion, SMR drives are assigned a different *device type* than CMR drives. The SMR drive will identify itself as a host-managed SMR (HM-SMR) drive, also known as a Zoned Block Device. This alerts the host software stack that there are different rules for writing to the drive than a conventional drive.

Each zone will have a sequential write pointer (SWP) which informs the host where the next write in the zone must occur. Any attempted commands to write to a different location than the SWP will be rejected by the drive. Any read commands to an area of the zone beyond the SWP will not return valid data. These rules must be understood at some, often multiple, levels of the software stack to ensure that the storage system behaves as expected.

The software ecosystem built to support SMR is now reaching a level of maturity where HM-SMR drives can be used in an increasing number of applications with fewer software engineering resources than previously required. Some file systems now have native HM-SMR support and make the integration process easier for applications that do not necessarily have access patterns consistent with a sequential write hardware device.

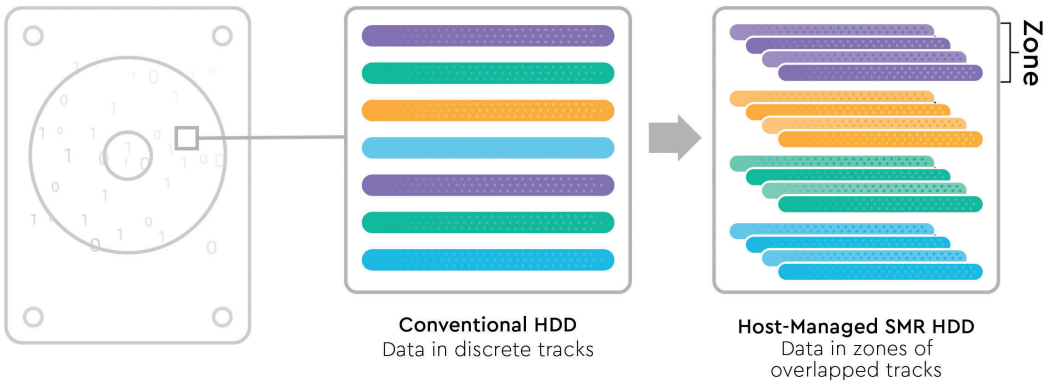

Figure 1: Comparison of Conventional and SMR HDDs (Image source: ZonedStorage.io)

SMR-Aware and SMR-Friendly Applications

The unique sequential write constraints of SMR first require determining whether the behavior of an application is suitable for SMR. With the right application behavior, write performance can be near parity with CMR storage. With the wrong application behavior, write performance may be significantly degraded. The first step any designer must take when determining whether to implement SMR is having a detailed understanding of the application behavior and where (if necessary) it can be changed.

With this, we can introduce the concept of an "SMR-Friendly" application. SMR-Friendly applications are those where the write behavior of the application as it currently exists is consistent with the write pattern constraints of an SMR HDD. In general terms, an SMR-Friendly application is one where the bulk of the writes to the drive are sequential. It is an application where there is no requirement to regularly update data in place, as SMR does not allow for overwrites of existing data in place. In many cases, archival or cold storage applications are inherently SMR-friendly.

Typically, this involves writing data once and keeping it safe for long periods of time without any modification, and so do not require updating in place. Other applications are file systems which use copy-on-write (COW) behavior as these file systems by nature do not update data in place. Certain applications which use versioning of files which are required to keep the original and all history of changes would be similar, as the new versions are written to fresh space on the drive.

By contrast, we can see applications that would be very unfriendly to SMR. One example is any application relying on traditional RAID. RAID is reliant on striping data across multiple disks and is not architected with the SMR zone structure in mind. Another would be a typical database application. In these applications, there is usually significant update of data in place. Most traditional filesystems are not natively friendly to SMR, as they allow for file updates in place instead of rewriting data to unused sectors. Essentially, any application with significant update-in-place or random write requirements will likely not be performant if modified to operate on SMR HDDs.

Beyond SMR-Friendly applications, we can also introduce the category of SMR-Aware applications. The primary difference is that an Aware application will be written specifically with the SMR storage constraints in mind and handle the SMR sequential write rules and data placement manually. Existing applications will need to be rewritten to be Aware. This is an involved software undertaking, but it will provide the most predictable performance characteristics in the system.

An Aware application will most likely be a Friendly application, but a Friendly application is not necessarily Aware. A Friendly application that is *not* SMR-Aware does not know that the storage device is SMR and must rely on the software stack underneath the application for following the SMR write rules and handling data placement. This will be easier to implement, but potentially less predictable in performance.

Examples of SMR-Friendly applications include, but are not limited to, the following:

- **Smart Video:** Video recorders generally write large sequential data streams, well-suited to SMR's sequential write restriction. Most video files are not intended to be overwritten in place. And when video is old and is deleted to make room for new video, the system can free up large blocks of free space which can be re-written sequentially.
- **Archival/Cold Storage:** Data that needs to be stored for long periods of time for financial, government, or regulatory needs, or other archival data that is merely needed to be kept indefinitely, is data that is likely to be written once and then read back, whether regularly or intermittently.
- **Content Delivery Networks:** Edge servers in these networks need the ability to store frequently accessed content to serve to users close to the users themselves but may only periodically update the content on the servers as user preferences change. Because the writes are infrequent, the application can typically work with the SMR write performance even if the write pattern isn't entirely friendly.
- **Artificial Intelligence / Machine Learning Training Datasets:** The training data for these algorithms is rarely changing (although frequently growing) but needs to be re-read each time that the algorithm itself is being tuned. Because this is a write-once read-many (WORM) application, it will usually be SMR-Friendly.
- **Cloud Document/Backup Services/File Storage Services Supporting Versioning:** In these applications, when a file is changed the previous versions and all revision history must be retained, so the changes can be written to a new location on the storage device not requiring updates in place. Because they can be architected around updating in place, these applications would be SMR-Friendly.

Implementation Options for SMR Applications

First and foremost, the Microsoft Windows operating system does not currently support HM-SMR devices. Windows-based applications will be unable to take advantage of HM-SMR. Today the support for SMR is only available within Linux® kernel based operating systems (referred to as Linux for simplicity).

Beyond simply using Linux, however, the level of SMR support is dependent on the kernel version. SMR support was added to the Linux kernel starting with version 4.10.0, with features and additional filesystem support added in later versions. In addition, if using a Linux distribution, it will require that support for HM-SMR (as described in the Linux kernel as "Zoned Block Device Support") is enabled at the time the kernel is compiled. If the kernel version supports HM-SMR devices but it is not enabled, the kernel will need to be recompiled with support enabled.

The ZonedStorage.io web site is a maintained resource where SMR support information on Linux kernel, filesystems, and distributions can be found.

Implementation for SMR-Aware Applications

If the application is SMR-Aware, there are two main paths: direct access, or use of the zonefs file system.

Direct Access: An application may choose to manage data using the SMR HDD directly without the help of a file system. Such direct access design can enable improved performance and can be achieved using either passthrough commands or regular IO system calls for systems that have a Linux kernel with SMR support enabled.

With the passthrough approach, the application directly issues SCSI or ATA commands to the HDD. Zones of HM-SMR drives are managed using a special command in addition to the regular CMR drive command set. For SATA, this is the Zoned Device ATA Command Set (ZAC), and for SCSI (SAS HDDs), it is the Zoned Block Command Set (ZBC). Software libraries such as *libzbc* can be helpful with simplifying the implementation of Aware applications using passthrough access.

For systems using a Linux kernel supporting HM-SMR drives, regular IO systems calls such as *read()* and *write()* can be used. The kernel also provides a set of *ioctl* system calls to manage the zones of a HM-SMR device. This approach generally leads to simpler application software support for HM-SMR and can also be more efficient thanks to its integration with the kernel block IO scheduler. Furthermore, the application also gain the ability to combine SMR support with other features that the HM-SMR drive may provide to improve quality of service. I/O priorities or Command Duration Limits are examples of such features.

Direct access requires that the host software knows where the sequential write pointer is for each zone and only issues writes to the allowed location. In addition, if data within a zone becomes unnecessary to keep, and zones become partially full of unnecessary data while retaining necessary data, the host will need to handle “garbage collection” of the zones to free up space if more writes must occur. Finally, without using a filesystem, the host will be responsible for knowing what data is in each location, likely by storing metadata about what is on the drives elsewhere. However, “elsewhere” may be on the same drive, as HM-SMR drives typically contain a single CMR area equal to 1% of the capacity of the HDD. This CMR area is included to provide an area where random writes are allowed, if needed by the application.

Direct access, because it does not use a filesystem, is the most challenging way to implement SMR from a software perspective. When direct access was the only viable method of implementing SMR, it explains why SMR adoption was very low; the work involved was too much for most. However, because it is dealing with the drive directly without the use of a filesystem, it also gives the host the most control over everything that the application needs to do to control the storage.

Direct access was introduced with the release of Linux kernel version 4.10.0.

zonefs: The introduction of the zonefs file system provides a file-based exposure of the zone structure of the HM-SMR device to a host software. However, this is not a typical POSIX-compliant file system as it does not handle many of the aspects of writing files that a traditional file system would offer. Thus, it is still incumbent on the application to be aware of the SMR architecture and follow the rules of writing.

In zonefs, each SMR zone is reported to the host as an individual file. The files are named numerically, from 0 to the total number of SMR zones on the disk. The host can write to each file, *but only in append mode*, using ordinary file write commands. The use of append mode ensures that the writes are sequential. When the file reaches the size of the zone, it cannot accept any more writes. If the drive supports a CMR zone, this will also be a file writeable by the host, but not subject to the append or sequential write restrictions.

When a zone is needed to be reclaimed for new writes, it can be done by using a truncate system call on the file representing that zone to set the file size to zero. This effectively tells the zonefs filesystem to reset the sequential write pointer to the start of the zone.

Implementation with zonefs, as shown, still requires that the application be fully SMR-Aware. This would require the same level of data placement management as required by direct access, such as metadata and garbage collection. However, by providing a more common file-based way of working with the zones, it is a simpler way to deal with the architecture, requiring less host work. For applications such as Smart Video, where the writes are sequential and typically already file-based, zonefs can simplify the implementation process.

zonefs support was introduced with the release of the Linux kernel version 5.6.0.

Implementation for SMR-Friendly but not SMR-Aware Applications

An application that writes in an SMR-Friendly manner but isn't written as SMR-Aware can still be used with SMR drives in many cases. Another modern enterprise file system, btrfs (pronounced butter-eff-ess), has native SMR support and some unique characteristics. Applications which can utilize a modern kernel version and btrfs as the file system can handle the SMR architecture with little or no changes to the application. In addition, Linux provides a capability for usage of SMR with legacy filesystems, however it is not recommended for anything but very specialized applications and it will be covered below only for completeness.

btrfs: As mentioned, one of the key aspects to being SMR-Friendly is to not update data in place. btrfs was originally designed prior to SMR to be a copy-on-write (COW) file system. This means that when a file is modified, btrfs does not write the changed file into the same location as the previous file; it writes it to a new fresh location. This behavior naturally made btrfs well-suited for adaptation to support SMR. With other modifications (primarily relating to how the filesystem superblocks and block allocation are handled to ensure sequential writes), btrfs introduced official zoned device / SMR support as of Linux kernel version 5.12. Having native filesystem support for SMR allows a much wider number of applications to support SMR drives.

Applications that rely on a filesystem therefore can run atop btrfs without modification. The application does not need to know the underlying structure of the drive. That said, the inherent rules relating to writing to SMR drives may mean that application behavior needs to be tweaked to ensure consistent performance. In addition, certain filesystem features, most notably software RAID, are not currently supported on zoned/ SMR devices.

When implementing an application that is not SMR-Aware atop btrfs, the biggest question a designer will typically have is “does it perform adequately to requirements?” For an application that is largely archival or cold storage, a WORM application, etc, large block sequential access with deletion (such as smart video), there is unlikely to be significant change relative to CMR. But for an application that does require more “randomized” writes, it is less clear. Here, one area of critical need is to understand that an SMR application atop btrfs may perform on writes much differently when the drive is nearly empty vs when it is nearly full.

This is due, much like an SSD, to the need to perform garbage collection. Since this must be done by the filesystem and the application is not aware of this need, it could impact performance. It is incumbent upon the system designer to ensure that test cases for the application take this into account.

To use btrfs, a minimum of kernel version 5.12, with Zoned Block Device Support enabled, is required. Most applications would likely prefer a more recent kernel version, as various early bug fixes and performance tweaks during the early deployment of SMR support have made SMR support more production-ready.

Device mapper: With the introduction of kernel version 4.13, Linux added support within the device mapper subsystem for two new capabilities, *dm-linear* and *dm-zoned*. These effectively allow both the application AND a legacy filesystem to operate on an SMR drive without being SMR-Aware. Thus, if an application cannot use btrfs, it may still be possible to support an SMR drive.

****DISCLAIMER****: The use of *dm-zoned* can have radically unpredictable write performance. The target applications which would use SMR in this way in any sort of production enterprise/datacenter environment is very narrow. As such, it will not be covered in detail and if additional information is needed, a reader should seek the zonedstorage.io web site for more information. Usage is not recommended unless the application characteristics and the performance implications are well understood by the system designer.

Data Redundancy/Resiliency When Implementing SMR

In addition to merely implementing an application that is either SMR-Aware or using a filesystem like btrfs, there are other needs that a system designer must address. One of the most common at the enterprise/datacenter scale is how to handle data redundancy/resiliency. If data is important, you need either multiple copies or self-correcting architectures that protect against hardware failure.

As already stated, traditional RAID is not suited to SMR, as it stripes data in fixed locations across sets of disks requiring update-in-place if any data changes. And while btrfs supports SMR, in current implementations the btrfs RAID capabilities do not support SMR. However, SMR's advantages are most well seen in the large-scale datacenter deployments where TCO drives decision-making. In these large-scale deployments, traditional RAID is not in common use, and data resiliency is more typically handled with replication, erasure coding, or both. Both are much more SMR-Friendly than RAID.

Replication is simple—distributing copies of the data across multiple drives (and potentially multiple geographic locations) looks the same on SMR as it would on CMR. Thus, replication on SMR introduces nothing new relative to CMR.

Erasure coding is more complex. The advantage of most erasure coding schemes is that the data is sharded across drives without a fixed stripe or requiring homogenous drive sizes and configurations. There is no requirement for update-in-place. All that is required is that there are enough drives with enough free available space to write every shard. It is very much a "copy on write" architecture for data resiliency. However, this can then also cause previously written data to be invalidated, requiring garbage collection that can impact performance. The erasure coding scheme itself, or the filesystem upon which it sits (if used), must be SMR-Aware to handle these garbage collection tasks.

So much as a system designer must really understand their application workload if employing SMR, the system designer must also understand and test the performance implications of any complexities introduced by erasure coding. Most importantly, they must understand that the performance characteristics of a mostly-utilized drive pool may be very different than a mostly-empty drive pool.

Validation/Qualification Strategies When Implementing SMR

When validation or qualifying HDDs for use in a system, many customers rely on tried-and-true benchmarks to understand drive performance, and understand their own workloads well enough that they can predict how a CMR drive will respond in the application as long as it meets the specified targets in their benchmarks.

When introducing SMR into the picture, the situation changes significantly. While benchmarking software exists for SMR drives, it becomes less clear how drive benchmarking results will map to application performance. Most applications with any level of "randomization" to the write patterns will require garbage collection. Benchmarks of sequential write speeds and random read IOPS are important, but it doesn't easily tell you how garbage collection of partially-utilized zones will impact performance. Some systems could see near-parity to CMR performance during garbage collection, or could see wild aberrations, once the storage is utilized enough that garbage collection routines are invoked.

SMR application performance is state-based, meaning that the performance depends on the state of information already on the storage device. To truly understand application performance, the application should be

"stress-tested" against various initial states, up to and including not simply testing mostly-full storage, but testing storage after multiple rounds of write / delete / garbage collection operations occur. In this, it is similar to SSD performance testing, which often requires the device to be preconditioned before testing. But for host-managed SMR, it is more that the application and the filesystem should be used extensively to understand performance changes that may occur in the various states the application will encounter in the storage pool.

As with most questions involving SMR implementation, validating performance and qualifying drives *is highly application-dependent*, and the system designer must fully understand the application and its needs to determine which benchmarks are relevant and what testing must be adapted from previous systems to ensure that the storage will meet the application performance requirements.

Conclusion

Implementing SMR HDDs into a storage application is not a trivial change. However, the value of increased storage density and improved TCO make it a compelling option. While in the past, the software ecosystem support wasn't mature enough for all but the most resourced companies to handle, modern software opens this up to more applications and potentially many which can be "drop in" replacements or close to it for CMR HDDs.

Learn More

Learn more about Zoned Storage Devices
https://zonedstorage.io/

Ultrastar DC HC680 Data Center HDD Data Sheet
https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/data-center-drives/ultrastar-dc-hc600-series/data-sheet-ultrastar-dc-hc680.pdf

Shingled Magnetic Recording (SMR) HDD Technology
https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/white-paper/white-paper-shingled-magnetic-recording-hdd-technology.pdf

Zoned Storage: Higher Capacities, Lower TCO & Improved QoS
https://www.westerndigital.com/company/innovation/zoned-storage

Technology Brief: UltraSMR
https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/tech-brief/tech-brief-ultrasmr-technology.pdf

Western Digital.

5601 Great Oaks Parkway
San Jose, CA 95119, USA
www.westerndigital.com