2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

Emulating a Shingled Write Disk

Rekha Pitchumani[1], Andy Hospodor[1], Ahmed Amer[2], Yangwook Kang[1], Ethan L. Miller[1], and Darrell D. E. Long[1]

[1]Storage Systems Research Center, University of California, Santa Cruz, CA
[2]Santa Clara University, Santa Clara, CA

***Abstract*—Shingled Magnetic Recording technology is expected to play a major role in the next generation of hard disk drives. But it introduces some unique challenges to system software researchers and prototype hardware is not readily available for the broader research community. It is crucial to work on system software in parallel to hardware manufacturing, to ensure successful and effective adoption of this technology.**

In this work, we present a novel Shingled Write Disk (SWD) emulator that uses a hard disk utilizing traditional Perpendicular Magnetic Recording (PMR) and emulates a Shingled Write Disk on top of it. We implemented the emulator as a pseudo block device driver and evaluated the performance overhead incurred by employing the emulator. The emulator has a slight overhead which is only measurable during pure sequential reads and writes. The moment disk head movement comes into picture, due to any random access, the emulator overhead becomes so insignificant as to become immeasurable.

I. Introduction

Any further significant improvements to the capacity of hard disk drives demands some major changes to the currently employed techniques, as they are reaching their limitations imposed by the laws of physics. Of the new technologies being explored, Shingled Magnetic Recording (SMR) promises an areal density increase of about 2.3x [13], and is particularly appealing as it requires minimal physical changes to the manufacturing process. A disk employing SMR technology, a Shingled Write Disk (SWD), shingles (layers) newly written tracks on top of preceding tracks, and hence new writes destroy old data on any such previously written shingles that are overwritten. This forces the SWD to be a largely sequential write device, but it remains an unrestricted random access device when dealing with read operations. Without careful management of the data layout on disk, random writes are destructive and so simple in-place block updates are no longer possible. Although it is possible to treat a SWD like a virtual tape (albeit with better random read performance), exploring its potential to replace Hard Disk Drives in their traditional roles is essential and is the topic of ongoing research. To enable such efforts, we therefore present a novel shingled write disk emulator that captures the key functional characteristics of such devices, while offering the flexibility to easily adjust drive design parameters.

Recent research [1], [4], [5] has explored the design issues in a shingled write disk system and proposed solutions ranging from data layout management to system software changes. But further development and assessment of the proposed solutions are hindered by both the limited availability of prototype devices, and the relative difficulty of physically adjusting prototype device parameters. In this work, we aim to solve this problem by emulating SWDs atop existing hard disk drives.

Hard disk drive manufactures are already producing shingled write drive prototypes, but the technology is not yet ready to enter production, and prototypes are not readily available for the broader research community. Even when the prototypes are ready to be distributed to researchers, it is not easy to alter the disk parameters and reconfigure new prototypes as desired without continually resorting to the manufacturer. But with our emulator, altering drive parameters is very simple, which is highly desirable for both researchers and disk manufacturers. Shingled Disks might not replace traditional hard disks for applications demanding good random write performance (like databases), but with appropriate remapping and firmware, they may be highly suitable for the wide array of applications that demand increased capacity at ever lower costs (e.g., archival systems, data logging, referential databases and data warehouses). We believe the introduction of SWDs will motivate research resulting in new hybrid storage architectures, and that offering a SWD emulator would be of great benefit to such research efforts.

The basic approach behind our emulator is to mimic the effect of individual writes on multiple tracks, maintaining information about the affected tracks and using this information when responding to subsequent read operations. Producing realistic behavior from the drive requires some prior knowledge about the underlying disk's physical geometry. This is specifically to tell us to which track a write is destined, and thereby to determine the subsequent tracks and sectors affected by that operation. We make use of the state of the art in disk performance profiling to shed light on hard disk drive internals and this information is used to configure our emulator for a specific disk.

We have implemented a pseudo device driver in the linux kernel to work in the block layer and perform the above operations. We used a single platter 160GB Seagate SATA drive as a test drive to create a virtual shingled block device using our emulator driver. This shingled block device was evaluated to measure the overhead incurred by the emulator, and it was determined that the emulator overhead is negligible and the shingled block device's performance is in line with the underlying disk. This shows that the emulated device can be used not just for functional experiments (to verify and validate SWD system software solutions), but can also be used for

1526-7539/12 $26.00 © 2012 IEEE
DOI 10.1109/MASCOTS.2012.46

performance comparisons reliably, as it does not mask the physical behavior of the underlying disk.

Our main contribution is a novel solution that can be used to test shingled disk management schemes, and shingled disk layouts and parameters, atop a real disk. Since the underlying medium and the read and write mechanics of a shingled write disk is expected to be very similar to existing hard disk drives, the read/write performance measurements on our emulated disk can serve as a good SWD performance indicator.

II. Background

Magnetic data recording technology is fast-approaching the density limit imposed by the super-paramagnetic effect for perpendicular recording. Current drives store $400\,\mathrm{GB/in^2}$, the current limit is estimated to be about $1\,\mathrm{Tb/in^2}$ [12]. While shingled write disks [9], [13], [6] are not the only technology aimed at enabling drives that exceed this limit, it differs from competing approaches by offering an elegant solution that does not require any significant physical changes to the drive mechanics, or to the manufacturing processes and materials used for the disk drives. However, shingled write disks introduce interesting new challenges, as they result in functional differences when compared to existing drives, thanks to the introduction of potentially destructive writes when data is updated.

The elegant solution offered by shingled disks is to use a write head with a stronger, but asymmetric, magnetic field. This approach is made possible by the fact that writes require a much stronger magnetic field than do reads. Shingled writing leverages this property by overlapping the currently written track with the previous track, leaving only a relatively small strip of the previous write track untouched. The remaining track is therefore *narrower* than when it was originally written, but remains readable. In this manner, tracks are ultimately placed closer together, resulting in the capacity gain. Achieving further gains in magnetic hard drives will require a combination of this basic shingled writing technology, and what is known as Two-Dimensional Magnetic Recording (TDMR) [11] technology. In this work we focus on emulating the behavior of basic shingled magnetic recording. Such disks would allow read operations to be performed randomly, but writing tracks must now be done sequentially as long as there is a chance of overwriting subsequent tracks. The number of tracks affected by such a write, k tracks, is a design parameter but is expected to be typically 4–8. This demands very careful interaction on the part of the system software, or the implementation of firmware that masks this risk through remapping data (as proposed by our prior work and Casutto *et al.* [2], [4]).

Disk data density improvements will eventually be limited by the superparamagnetic effect, which creates a trade-off between the media signal-to-noise ratio, the writeability of the media by a narrow track head, and the thermal stability of the media; Sann *et al.* call this the *media trilemma* [11]. While various approaches to this problem have been proposed;

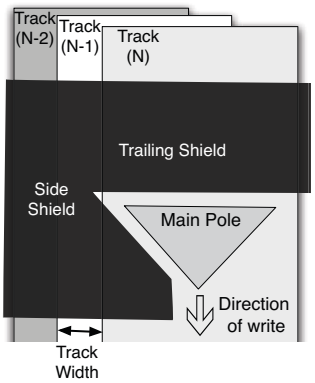


Fig. 1: Corner write head for shingled writes [2].

shingled writing offers perhaps the most elegant solution. Rather than radically altering the makeup of the magnetic layer (as is done by technologies that pattern the media surface, or manipulate it by localized heating using lasers). Shingled writing does this by using a write head that generates an asymmetric, wider, and much stronger field that fringes in one lateral direction, but is shielded in the other direction. Figure 1 shows a larger head writing to track n, as used by Greaves *et al.* in their simulations [8]. Shingled writing overlaps tracks written sequentially, creating effectively narrower tracks after the once-wider leading track has been partially overwritten, and is thereby expected to increase storage densities by a factor of at least 2.5 [13] to 3 [8] times the current theoretical limit of $1\,\mathrm{Tb/in^2}$ with current magnetic recording technology.

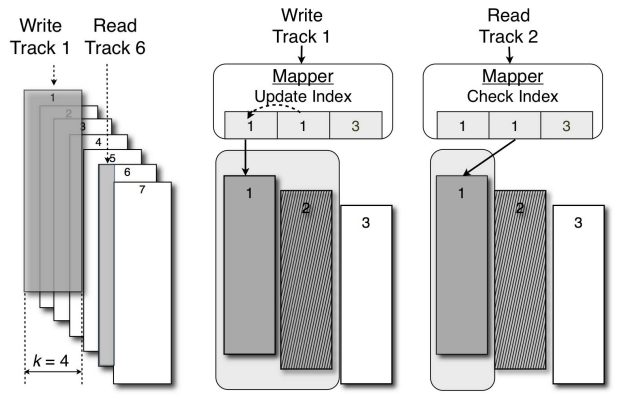
III. Emulation

Our goal here is to make a faithful Shingled Write Disk emulator, one that closely mimics the SWD's functional behavior and can be used to verify and validate proposed disk management schemes. Storage simulation has been widely used by systems researchers as it aids evaluation of proposed storage systems architectures and systems software. Examples of such successful simulation systems include DiskSim [3], NandSim and DRAMSim [14]. But in our work, we seek to emulate the behavior of a SWD, and allow experimentation using existing physical drives.

Simulation systems have taken special care to report realistic performance characteristics of systems by studying and simulating the physical timing characteristic of the concerned storage systems. But the big issue with SWDs is not its physical timing but rather it's the functional behavioral difference. Hence, we leave the physics of the underlying mechanics to a real physical device, thereby emulating a SWD on top of a real hard disk drive. In this section, we detail how to emulate a SWD on a traditional hard disk.

A. Track Shingling

The difference between a SMR hard disk and a PMR hard disk is track shingling. Thus, any modern hard disk can be made to look like a SMR disk by viewing track shingling as



(a) SMR drive (b) Emulating Track Shingling on a PMR drive using a Map Table

Fig. 2: **Original and Emulated Track Shingling**. Shingling results in wider write tracks and narrower read tracks.

using multiple (k, where k is the number of tracks affected by shingling) tracks for writing and single track for reading. Figure 2a illustrates track shingling in a SWD with $k = 4$.

Figure 2b illustrates how to emulate multiple track writes and single track reads in a modern disk drive using a map table. A write to a track is written to only one track, but the mapping table is modified to indicate that subsequent (k - 1) tracks were written with data from the track that was written to. Every read request first checks the map table and is redirected to a different track if required.

For example, in Figure 2b a write to track 1 is written to track 1 and track 2 is marked as overwritten by track 1 (here, k is 2). Every read operation first checks the map table to determine whether the track was overwritten earlier.

B. Sector Level Mapping

Disk *Reads* and *Writes* do not happen at track level, they happen at sector level. Hence, the mapping also has to be performed for individual sectors and has to take the underlying hard disk's physical geometry into account.

1) Hard Disk Physical Geometry: As hard disks have become more complex, they hide detail behind Logical Block Addressing. Today, host operating systems see data written to a disk as being written to consecutive sectors and address the sectors by their Logical Block Addresses (LBA) and the disk takes care of mapping the LBA to the sector's physical location on the disk. Hence, determining in which track a sector with a given LBA resides and determining the LBAs of the sectors that lie in the subsequent tracks (and will be affected by a shingled write) requires an understanding of the underlying disk's true physical geometry.

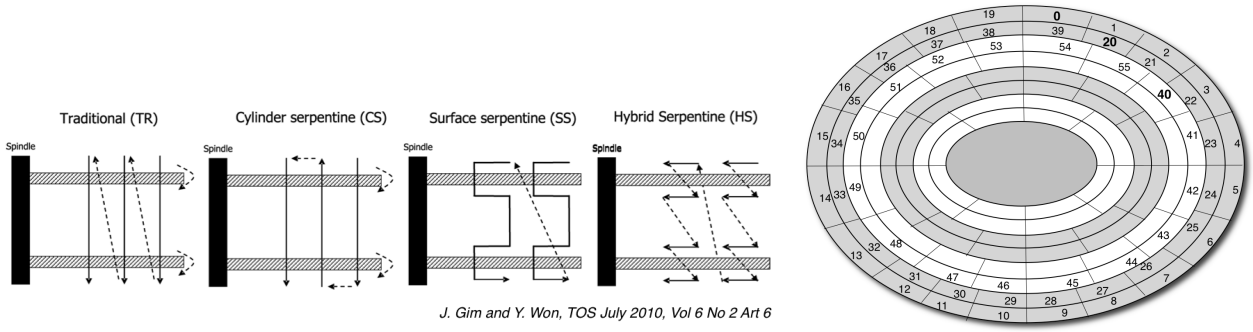
Modern hard disk drives have a complex physical geometry that is tailored to individual drives. The geometry of a particular drive is determined by a combination of the disk surface characteristics and the drive's write head characteristics. Hence the geometry of two disks from the same manufacturer with the same specification can differ [10]. Even the disk's controller *learns* the drive's geometry during the final manufacturing process, meaning that the final geometry is established post-production.

Disk drive physical geometry can be extracted by leveraging existing research [7]. Obtaining accurate values is neither easy nor necessary. It is a known fact that when the HDD firmware detects a physical sector as not usable anymore, it remaps the LBA of the sector to one of the sectors in its spare locations. We are aware that even if the extracted LBA mapping is very accurate, it is subject to changes due to the above remapping. For our purposes, approximation of the physical geometry was sufficient and hence, ignoring the intended sector remapping is justifiable, as we are focusing on the mapping indicated by the disks observed performance when carefully benchmarked. Hence, we parameterize the disk drive geometry and fit the extracted values into the following parameters:

- **Number of Heads** The number of heads gives the number of writable surfaces in a disk drive and can be obtained from the drive's specification.
- **Sector Layout Mechanism** Different hard disk manufacturers use different sector layout mechanisms [7]. Figure 3a illustrates the most commonly used schemes, surface serpentine, cylinder serpentine and hybrid serpentine schemes. A complex sector layout mechanism can make the mapping scheme quite complicated.
- **Number of Zones** A Zone is a set of adjacent tracks that have the same number of sectors per track. For example, in Figure 3b there are 4 zones.
- **Zone Size** The Zone Size is measured in terms of number of tracks. In Figure 3b, there are two tracks in every zone and hence the zone size is 2 for all 4 zones.
- **Track Size per Zone** Denotes the number of sectors per track in a zone. In Figure 3b, the outermost zone has 20 sectors per track and the zone next to it has 16 tracks per track.
- **Track Skew** The start LBA of a track is placed at an angle past the start LBA of the previous track. This angle is given by the track skew.

2) Write request handling: On a *Write*, the Logical Block Addresses of the sectors that will be overwritten by the current *Write* has to be determined. Once determined, the overwritten sector information has to be stored in a sector level map table. A simple Hash Table where every entry is a tuple of the form $\langle originalLBA, mappedLBA \rangle$ would be sufficient. On a *Write*, new entries must be added for every overwritten sector, mapping it to the LBA of the current sector being written to. Further, if there exists an entry for the LBA of the current sector, it has to be deleted from the hash table.

Determining Overwritten Sectors Figure 4 illustrates the parameters required to determine overwritten sectors. To make it easier to understand, imagine a line from the disk OD to ID starting at the lowest LBA of the first track on OD. It is known that the sector boundary of subsequent tracks may not align with this imaginary line. But lets take the first sector after this line to be the start and lay the circular track out in



(a) Sector Layout Mechanisms.

(b) Surface Layout is determined by Zoning and Track Skew.

Fig. 3: **Simplified View of Hard Disk Physical Geometry**. Modern hard disk drive's geometry is determined post-production. Hence, the geometry of two disks from the same manufacturer with the same specification can differ.

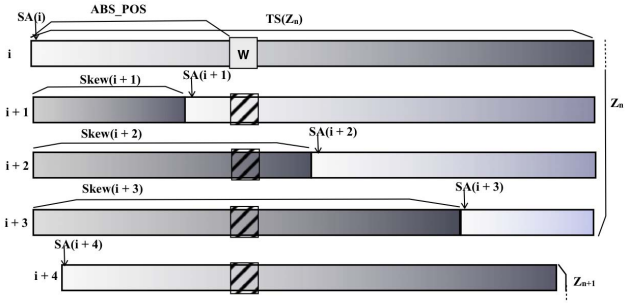


Fig. 4: **Emulating Sector Overwrite**. Determining the LBA of the sectors being overwritten by a Write W requires physical geometry information.

a horizontal manner as shown in Figure 4.

Let **TN(LBA)** be the track number of the track where the sector with the given LBA resides. In the figure, i to $i+4$ are the track numbers of the tracks shown and **TN(LBA)** for any LBA residing in i will be i. **SA(Track)** denotes the starting address of (or the lowest LBA in) the given Track.

As seen in Figure 4, the starting addresses do not align with our imaginary line because of Track Skew and are placed at an angular offset and after a period realigns with our imaginary line. In the figure, this Skew period is 4 tracks. **Skew(Track)** gives the Skew for the Track in terms of number of sectors. Tracks with the same number of sectors per track is grouped into a Zone. The figure shows tracks from two Zones, Z_n and Z_{n+1}. **Z(Track)** gives the Zone number of a given Track and **TS**(Z_n) is the Track Size, the number of sectors per track for the Zone Z_n.

The values obtained from **TN**, **SA**, **Z**, **TS**, and **Skew** are all determined based on the extracted disk geometry information. Hence, the accuracy of the calculations are heavily dependent on the accuracy of extracted disk geometry. Determining the overwritten sectors on a write is a two step process. The first step is determining the absolute position **ABS_POS** of the *Write W* from our imaginary line. The next step is finding the LBA at **ABS_POS** for subsequent tracks.

First step: Let LBA_W be the Logical Block Address of the Write W and $t = TN(LBA_W)$. Then,

$$POS = LBA_W - SA(t) + Skew(t)$$

If $POS > TS(Z(t))$, then

$$ABS\_POS = POS - TS(Z(t))$$

Else,

$$ABS\_POS = POS$$

Second step: This step gives how to determine the LBA of the affected sectors at Track $t + j$. If $Skew(t + j) < ABS\_POS$, then

$$LBA = SA(t + j) + (ABS\_POS - Skew(t + j))$$

Else,

$$LBA = SA(t+j)+(TS(Z(t+j))-Skew(t+j))+ABS\_POS$$

If the affected track falls in the next zone, like track $i + 4$ in the figure, then ABS_POS is first adjusted as below.

$$ABS\_POS = (ABS\_POS/TS(Z(t))) * TS(Z(t + j))$$

3) Read request handling: Read requests have to be checked to determine if the sectors being read would have been (for an SWD) overwritten by a previous write. In other words, the Hash Table has to be checked for the presence of sector entries for all sectors being read. If there exists no entry, then the read is straightforward - forward the read to the underlying real disk and proceed with the read as is done normally.

But, reads could get a little complicated if tracks are not written sequentially. For example, in Figure 5, a write to track 1 overwrites tracks 2 and 3, and a write to track 2 overwrites track 3. Lets consider three writes happening one after another, Writes 1, 2, and 3 in the figure. After all 3 writes, Track 3 has segments pointing to other tracks.

There are three choices as to what to do when an over-written sector, *i.e.*, a sector that was overwritten by write to a different sector, is read back. As such, the emulated drive can

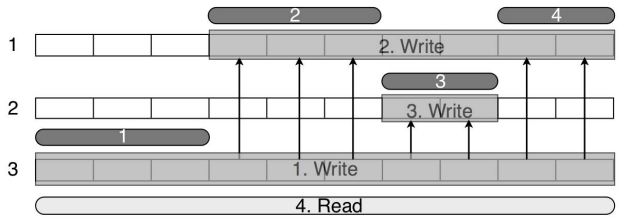


Fig. 5: Random writes may result in data corruption. Read (4) after random writes (1, 2, 3) may return corrupted data or error.

operate in three modes, depending on the user's requirements, and the modes determine how the emulated drive behaves.

Data Centric In this mode, the overwritten data (as per the mapping information) will be retrieved, thereby emulating the behavior of the Shingled Write Disk with no overwrite error checking in-place. For example, in the above scenario, the *Read* request, happening 4th in sequence, is split into four requests, each sent to their track and the read information is returned back. *Read* splits do not happen if tracks are written sequentially.

Performance Centric If retrieving the overwritten data back does not matter, then the emulator performance can be improved further by keeping it simple and maintaining a bitmap structure to indicate overwrites. In that case, reads can be proceeded as usual, but the data buffer containing the data from overwritten sectors is to be filled with garbage data. In other words, invalid data is not retrieved if known to be invalid *a priori*.

Development Centric Since a primary purpose of emulating the shingled disk is to aid system software researchers and developers, a mode that returns an error when an overwritten sector is read back will be beneficial. This mode can be combined with either of the above two modes as needed. And it is up to the upper-level layers to determine whether the overwrite was intentional or unintentional.

IV. Implementation

Our requirement can be best fulfilled by a pseudo device driver in the kernel that receives block read and write requests and performs the mapping as described above. We use the Device Mapper infrastructure available in the Linux 2.6 kernel, a generic framework for constructing new block devices and mapping them to existing block devices.

We implemented *dm-shingle*, a bio-based device-mapper target module. Linux provides a *dmsetup* utility to manage the logical devices that use the device-mapper driver. Once our *dm-shingle* driver module is loaded, *dmsetup* can be used to create a logical shingled block device with desired parameters on top of the block device representing the hard disk. The resulting I/O stack can be seen in Figure 6.

The mapping table is implemented as a sector-level hash table as described earlier. The number of buckets in the hash table can be chosen during device setup and multiple entries in a bucket are linked as a list. The hash function is a simple LBA

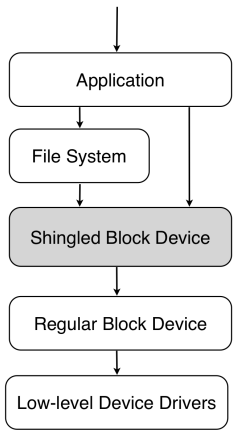


Fig. 6: **Emulator in I/O Stack**. We implemented the Emulator as a pseudo block device driver in the Linux kernel.

mod number of buckets scheme and the number of buckets should be a power of two for faster hash function execution. This ensures even distribution of entries across buckets when writes are mostly sequential.

The emulator driver allows for some customization of the emulated SWD and provides flexibility in deciding suitable parameters. The parameters that are customizable are the number of tracks overwritten by the shingled write (k), extracted underlying disk geometry and the hash table size. The various read behaviors detailed above has not been implemented and can be another such parameter. The read behavior that has been currently implemented and used for the evaluations in the next section is the *Data Centric* behavior.

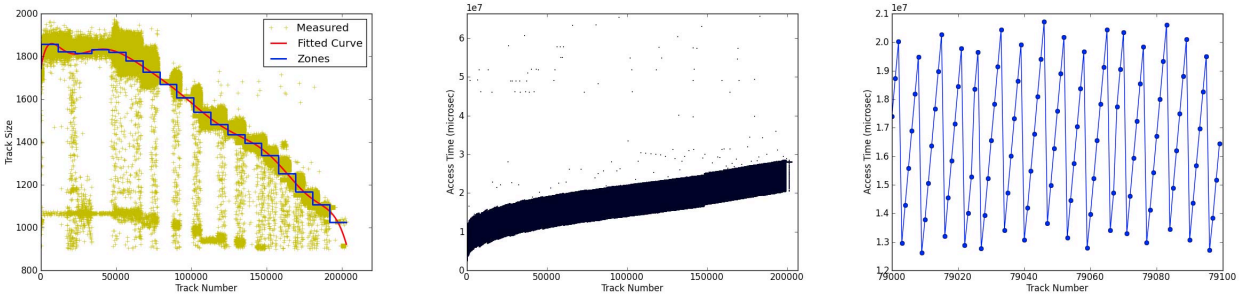
Since the emulator is implemented in the block layer, adding *new commands* is also straightforward. New commands can be easily serviced by implementing new *ioctls*. Since the Shingled Write Disks are still under prototype development and the best interface for it is not yet known, this feature becomes crucial. Researchers can have the flexibility to work on new command sets and interfaces best suited for SWDs.

V. Evaluation

The experiments for the evaluation were run in a host with a dual-core 3.20 GHz Intel(R) Core(TM) i5 processor with hyper-threading and a 8GB RAM. To keep it relatively simple, the test disk drive we used was a 160GB single platter Seagate SATA drive. We chose a single platter disk to avoid a complicated sector layout mapping.

We have used Disk Geometry Analyzer (DIG) to perform geometry extraction on our test drive and is explained in detail below. We used *fio* to generate desired IO patterns, to test and verify our emulator. The results shown in this section were measured using *fio*.

All tests were run on block devices using Direct IO, bypassing the kernel buffering, because we did not want the buffering to interfere with our mapping. For example, a read call to sectors overwritten by an earlier write, if serviced by



(a) Zone Determination. (b) Track Skew - Entire Disk. (c) Track Skew - Smaller Scale.

Fig. 7: **Geometry Extraction**. Physical Geometry of a 160 GB single platter test disk extracted using Disk Geometry Analyzer (DIG).

the kernel block buffers, will give a different result than if read from the disk. But it has to be noted that this is a problem that will be faced even when using a real Shingled Write Disk. A shingled disk management scheme, that is not absolutely sure that it will not read sectors that was overwritten unintentionally, cannot use buffering as-is.

A. Geometry Extraction

We used Disk Geometry Analyzer (DIG), a disk characterization tool to extract the disk geometry of our test drive. DIG determines the number of tracks and the size of each track in terms of number of sectors. Figure 7a shows a plot of the DIG output. Even after ignoring some of the variations in the measured result as noise, the results only show a zoning pattern and do not give clear zoning information, as expected.

Krevat *et.al.* [10] call this behavior as *adaptive zoning*, where track sizes are determined by the capabilities of disk surfaces and head combination post-production. Since we are only modeling the Shingled Disk behavior, approximate zoning is sufficient and the desired level of approximation can be chosen. Figure 7a shows how we can fit a curve to the measured values and obtain zoning information based on that. One is free to choose the number of zones and hence the level of approximation they desire to have, by varying the curve parameters or the number of zones.

DIG measures the time taken to reach the first sector of individual tracks from the first sector of the outermost track, to calculate Skew as explained in [7]. Figure 7b shows the plot of this result for the entire disk and 7c shows the plot at a smaller scale for 100 tracks. Once again there is some noise in the results obtained, but a pattern can be observed.

Access time gradually increases with track number and then drops significantly after a certain number of tracks. If this number of tracks is n tracks, then track skew corresponds to an angle of $2\pi/n$. For our test disk, this period alternates between 6 and 7 tracks.

B. Verification

We used *fio* to run Write-focused workloads on the emulated SWD and then to read the written data back and perform MD5 checksum verification on them. We wrote 64MB of data in each case and hence totally 16,384 4KB blocks were written in each case. Figure 8 shows the result of the verification.

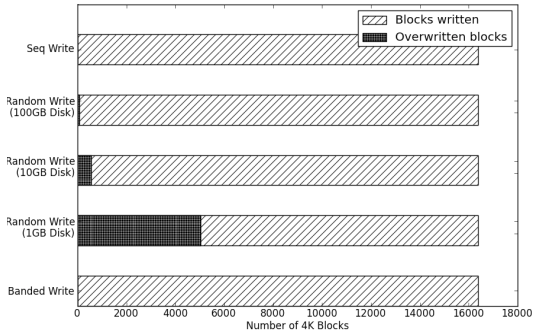


Fig. 8: **Write Verification**. As expected, random writes on the emulated SWD destroy data, while sequential and banded writes don't.

As expected, sequential write does not result in any data loss. Next, we verified random writes in 3 cases, one where the random write is performed in a total space 100GB, another in a space of 10GB and the last in a space of 1GB. As expected, all three random writes result in some blocks being overwritten and the number of blocks overwritten grows as the space decreases.

Finally, we performed banded write, where tracks are grouped together into bands, with some tracks serving as inter-band gap between them. The bands are written to in a log manner. As expected, the results show that there are no overwritten blocks in case of banded write. Banding requires some geometry information, and careful evaluation at the end of bands to ensure they do not affect subsequent bands is essential.

C. Performance Evaluation

The goal of the evaluations presented in this section is to measure the performance overhead incurred by the SWD emulator. We have used *fio* utility to generate the desired IO workload and measured the performance. For all scenarios below, the tests were run multiple (5-10) times and the average

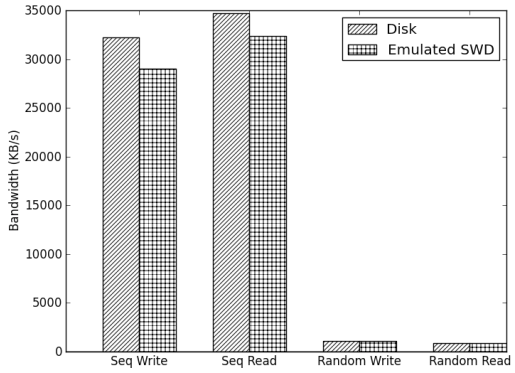


Fig. 9: **Emulator Performance Overhead**. A slight overhead is visible during Sequential IO, but becomes negligible as disk head movement comes into picture.

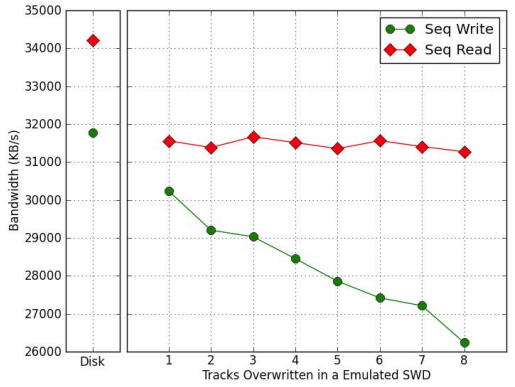


Fig. 10: The number of tracks overwritten by a shingled write does not affect reads, but adds a slight overhead for every overwritten track during writes.

aggregate bandwidth reported by *fio* is reported here.

As mentioned earlier, all IOs are Direct IOs without kernel buffering and the IO bandwidth of the virtual shingled block device is compared to that of the underlying raw block device. A virtual shingled block device was created to mimic a Shingled Disk that overwrites 3 tracks on a write and the IO block size was 4K. Figure 9 shows the performance of sequential and random reads and writes.

The emulator incurs a slight overhead, but it is observed only during pure sequential reads and writes, as seen in the figure. The bandwidth obtained from the virtual device is very similar to that of the underlying block device in case of random reads and random writes.

To understand the overhead incurred during sequential reads and writes better, we varied the number of tracks overwritten by the shingled device and measured the sequential read and write bandwidth. Figure 10 gives the result of the above experiment. The first subplot shows the raw underlying disk bandwidth and the second bigger subplot shows the bandwidth of the emulated SWD as the number of tracks overwritten increases.

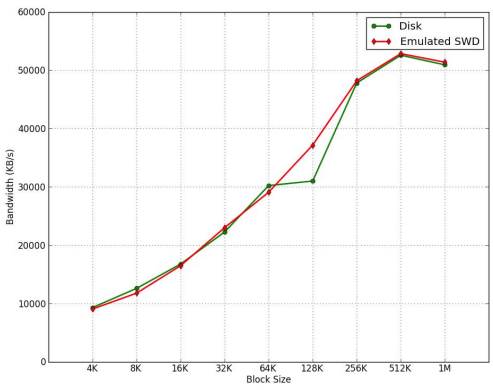


Fig. 11: **Banded Write Performance**. Emulator overhead is negligible and the banded write performance can be improved by increasing the IO block size.

The emulated SWD when the number of tracks overwritten is 1 is behaviorally the same as the regular disk. Hence, the difference in bandwidth of these two is pure overhead, irrelevant of how the emulated device is being used. The pure overhead for reads is higher than that of writes, but it is not affected by varying the number of tracks overwritten by shingled write, whereas write bandwidth decreases as number of tracks overwritten increases.

The above behavior is expected, because for reads every sector has to be verified against the hash table to check whether it was overwritten by a previous write, which explains the higher overhead. Lower the number of hash table buckets, higher the chances of collision and the decrease in bandwidth, as the number of entries in the hash table grows. Having a hash table with high number of buckets reduces collision, and hence the reads are unaffected by the number of tracks overwritten in our results.

During writes, the emulator checks the hash table for every sector being written to, in order to delete the entries if present. This explains the pure overhead for writes. Further, the work to be done increases as the number of tracks overwritten by a write increases. Hence, each additional overwritten track incurs roughly 2% overhead.

We believe some form of banded write is the most likely write scenario to occur with a Shingled Write Disk and hence evaluate the emulated SWD performance overhead for banded writes. We divided the disk into 16 bands and wrote randomly to all 16 bands. In each band, the write is sequential, and hence, we varied the block size of the writes from 4KB to 1MB, and the result can be seen in Figure 11.

Figure 11 shows three things. First, the emulator overhead becomes negligible with banded write. Hence, our emulated SWD can be a really good performance indicator, since writes in a real SWD are most likely to follow some form of similar banded writing. Second, emulator performance is unaffected by IO block size. Third, banded write performance can be improved greatly if write block size is increased. There is an

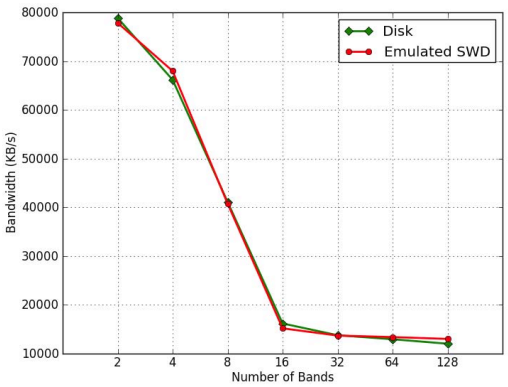


Fig. 12: Emulator Performance is not impacted by the Number of Bands.

anomaly where at the 128K block size, the emulated SWD performs better than the real disk. We believe this is due to buffer size mismatches in the regular block IO path in the Linux kernel.

In the final test, we check the impact of the number of bands on performance. We chose a 32KB block size and divided the test disk into fixed sized bands. The data to be written was split across the bands and written to the bands randomly. Figure 12 shows the number of bands does not affect the emulator performance. Further, as expected, the bandwidth decreases as the data gets spread across many bands. But after 16 bands, the number of bands does not affect write performance.

VI. Conclusion

Shingled Magnetic Recording technology can increase the areal density and hence the storage capacity of hard disks at least two-fold. Its successful adaption may be the key to meeting the ever-growing storage demands. Since SMR drive changes the way a disk behaves from the host operating system perspective significantly, it requires system software support, or complex firmware, to aid its integration into existing storage systems. To further research in this area, and provide access to realistic SWD behavior, we have implemented a novel emulator that can be easily deployed as a linux driver atop existing physical disks.

We have presented a novel method to readily emulate SWDs on any existing hard disk. We also implemented the proposed method as a pseudo device driver and evaluated the overhead incurred. We have shown that there is a slight overhead in the cases of pure sequential reads and writes, but the overhead incurred is so negligible as to be imperceptible in cases of random IO and banded IO (the latter being the most likely configuration for future SWDs). We will be making our emulator available for use by the research community.

Our emulation approach provides us a unique opportunity to do a little more than Shingling. Since the best interface or command sets for a Shingled Write Disk are as-yet unknown, the emulator can be used to experiment with those as well. A few of the interesting ways in which this can be used include:

- Report an Overwrite error when overwritten sectors are read back.
- Add additional *Banding* commands by means of ioctls and present a Banded device to the upper layers.
- Add commands to report the disk's geometry information to the upper layers, say, to perform dynamic banding as desired, or to implement a simple read-modify-write mechanism.

References

[1] Amer, A., Holliday, J., Long, D. D. E., Miller, E. L., Paris, J.-F., and Thomas Schwarz, S. Data management and layout for shingled magnetic recording. In *IEEE Transactions on Magnetics* (2011).

[2] Amer, A., Long, D. D. E., Miller, E. L., Paris, J.-F., and Schwarz, T. Design issues for a shingled write disk system. In *26th IEEE Symposium on Mass Storage Systems and Technology* (2010).

[3] Busy, J. S., Schindler, J., Schlosser, S. W., Ganger, G., and Contributors. The disksim simulation environment version 4.0 reference manual. Tech. Rep. CMU-PDL-08-101, Carnegie Mellon University, May 2008.

[4] Cassuto, Y., Sanvido, M. A., Guyot, C., Hall, D. R., and Bandic, Z. Z. Indirection systems for shingled-recording disk drives. In *Proceedings of the 26th IEEE Conference on Mass Storage Systems and Technologies* (May 2010).

[5] Gibson, G., and Ganger, G. Principles of operation for shingled disk devices. Tech. Rep. CMU-PDL-11-107, Carnegie Mellon University, 2011.

[6] Gibson, G., and Polte, M. Directions for shingled-write and two-dimensional magnetic recording system architectures: Synergies with solid-state disks. Tech. Rep. CMU-PDL-09-014, Carnegie Mellon University, May 2009.

[7] Gim, J., and Won, Y. Extract and Infer Quickly: Obtaining Sector Geometry of Modern Hard Disk Drives. *ACM Transactions on Storage 6*, 2 (2010).

[8] Greaves, S., Kanai, Y., and Muraoka, H. Shingled recording for 2–3 Tbit/in^2. *IEEE Transactions on Magnetics 45*, 10 (Oct. 2009), 3823–3829.

[9] Kasiraj, P., New, R., de Souza, J., and Williams, M. System and method for writing data to dedicated bands of a hard disk drive. United States Patent 7490212.

[10] Krevat, E., Tucek, J., and Ganger, G. R. Disks are like snowflakes: No two are alike. In *13th Workshop on Hot Topics in Operating Systems* (May 2011).

[11] Sann, C. K., Radhakrishnan, R., Eason, K., Elidrissi, R., Miles, J. M., Vasic, B., and Krishnan, A. R. Channel models and detectors for two-dimensional magnetic recording (TDMR). *IEEE Transactions on Magnetics 46*, 3 (Mar. 2010), 804–811.

[12] Shiroishi, Y., Fukuda, K., Tagawa, I., Takenoiri, S., Tanaka, H., and Yoshikawa, N. Future options for HDD storage. *IEEE Transactions on Magnetics 45*, 10 (Oct. 2009).

[13] Tagawa, I., and Williams, M. High density data-storage using shingle-write. In *Proceedings of the IEEE International Magnetics Conference* (2009).

[14] Wang, D., Ganesh, B., Tuaycharoen, N., Baynes, K., Jaleel, A., and Jacob, B. Dramsim: a memory system simulator. *SIGARCH Comput. Archit. News 33*, 4 (Nov. 2005), 100–107.