Shingled Write Disk Emulator

Technical Report UCSC-SSRC-12-05
August 2012

Rekha Pitchumani
rekhap@soe.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

UNIVERSITY OF CALIFORNIA

SANTA CRUZ

Shingled Write Disk Emulator

A project report submitted in partial satisfaction
of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

REKHA PITCHUMANI

June 2012

The Project of Rekha Pitchumani
is approved:

Professor Ethan L. Miller

Professor Darrell D. E. Long

Table of Contents

Acknowledgments

I would like to thank my advisor, Ethan Miller, for his guidance and feedbacks, and Darrell Long for all the helpful hints and encouragement. I would also like to thank Andy Hospodor, Ahmed Amer and Yangwook Kang, for their helpful insights and continued support without which this project would not have been possible.

This work was supported in part by the National Science Foundation under award IIP-0934401 and the industrial sponsors of the Center for Research in Intelligent Storage, including EMC, Hewlett Packard, IBM, NetApp, Northrop Grumman, and Samsung. I thank them all for supporting me and my work.

Finally, I would like to thank my parents, my husband, Pitchai Rajamani, and my son, Vidhush Ganapathy, for always being there for me.

Abstract

Shingled Magnetic Recording (SMR) technology is expected to play a major role in the next generation of hard disk drives. SMR introduces some unique challenges to system software researchers and prototype hardware is not readily available for the broader research community, making it difficult to explore solutions optimized for SMR disks. It is crucial to work on system software in parallel to hardware manufacturing, to ensure successful and effective adoption of this technology. This project is 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.

The emulator was implemented as a pseudo block device driver and the performance overhead incurred by employing the emulator was evaluated. 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. Parts of this report that describes the above has been accepted for publication [1]. In addition, this report also contains a section that serves as a reference on how to setup and use the Shingled Write Disk Emulator.

1 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 [2], 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 (typically, 4–8 tracks are overwritten by a shingled write). 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, this work presents 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 [3, 4, 5, 6] 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. This work aims to solve this problem by emulating SWDs atop existing hard disk drives.

Hard disk drive manufacturers 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 altering drive parameters is very simple with the emulator, allowing easier exploration of the solution space, which is highly desirable for both researchers and disk manufacturers.

Shingled Write 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). The introduction of SWDs will motivate research resulting in new hybrid storage architectures, and offering a SWD emulator would be of great benefit to such research efforts.

The basic approach behind the emulator is to mimic the effect of individual writes on multiple tracks, maintaining information about the affected tracks and using this infor-

mation when responding to subsequent read operations. Producing realistic behavior from the drive requires some prior knowledge about the underlying disk's physical geometry, to tell us to which track a write is destined, and thereby to determine the subsequent tracks and sectors affected by that operation. The emulator makes use of the state of the art in disk performance profiling to shed light on hard disk drive internals and uses this information to configure the emulator for a specific disk.

The emulator has been implemented as a pseudo device driver in the linux kernel to work in the block layer and perform the above operations. A single platter 160 GB Seagate SATA drive was used as a test drive to create a virtual shingled block device using the 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 compared to disk head movement 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 performance comparisons reliably, as it does not mask the physical behavior of the underlying disk.

The main contribution of this work 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 the emulated disk can serve as a good SWD performance indicator.

2 Background

Magnetic data recording technology is fast-approaching the density limit imposed by the super-paramagnetic effect for perpendicular recording. Current drives store 400 GB/in^2, current limit is estimated to be about 1 Tb/in^2 [7]. While shingled writing [8, 2, 9] is 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) [10] technology. This work focuses on emulating the behavior of

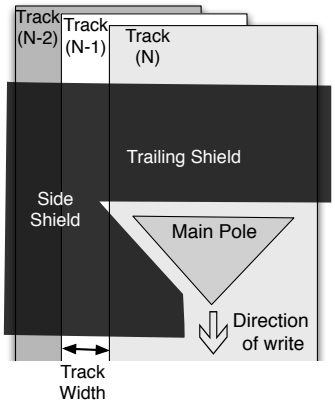


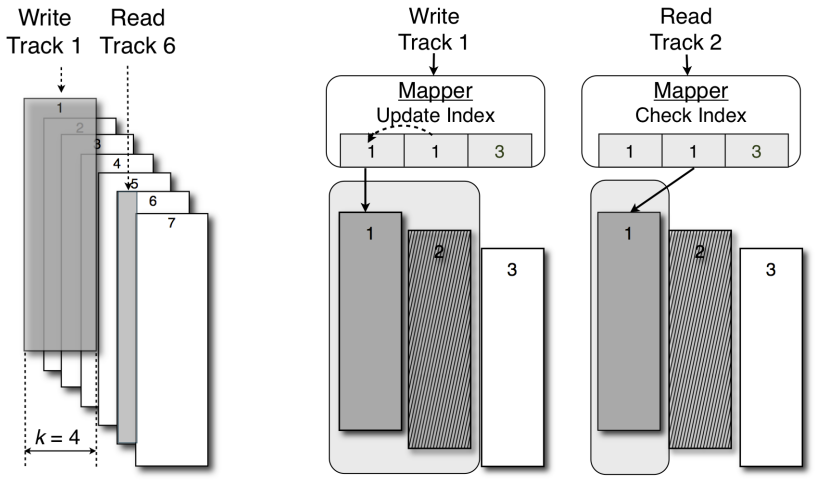
Figure 1: Corner write head for shingled writes [4].

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 Casutto *et al.* [5] and Amer *et al.* [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* [10]. While various approaches to this problem have been proposed, 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). 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 [11]. 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 [2] to 3 [11] times the current theoretical limit of $1\,\mathrm{Tb/in^2}$ with current magnetic recording technology.

3 Emulation

The 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 [12], NANDsim [13] and DRAMSim [14]. But this project seeks to emulate the behavior of a SWD, and allow experimentation using existing physical drives.



(a) SMR drive

(b) Emulating track shingling on a PMR drive using a map table

Figure 2: **Original and emulated track shingling**. Shingling results in wider write tracks and narrower read tracks.

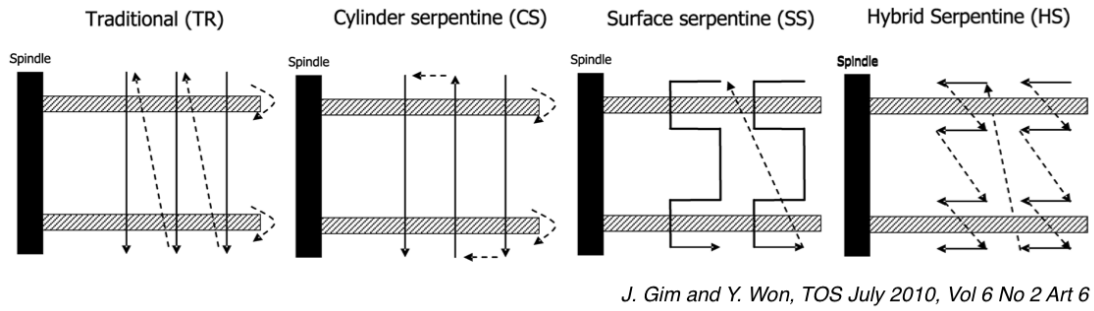
Simulation systems have taken special care to report realistic performance characteristics of systems by studying and simulating the physical timing characteristic of the concerned storage systems. But the big issue with SWDs is not its physical timing but rather the functional behavioral difference.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

3.1 Track Shingling

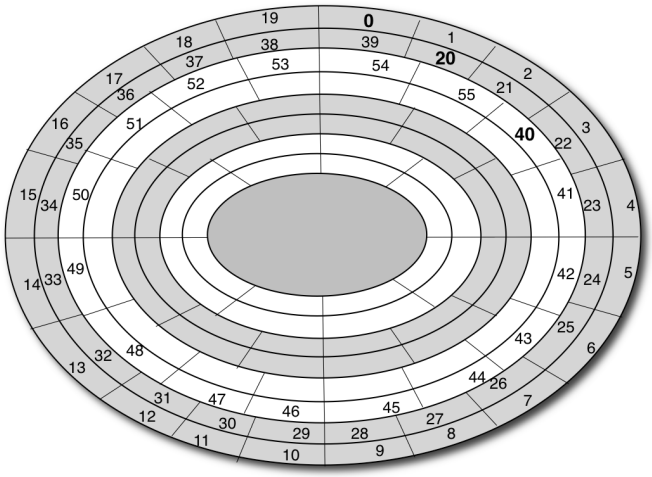
The difference between a SMR hard disk and a Perpendicular Magnetic Recording (PMR) hard disk is track shingling. Thus, any modern hard disk can be made to look like a SMR disk by viewing track shingling as using multiple (k, where k is the number of tracks affected by shingling) tracks for writing and 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.



(a) Sector layout mechanisms.



(b) Surface layout is determined by zoning and track skew.

Figure 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.

3.2 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.

3.2.1 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 [15]. 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 work [16]. Obtaining accurate values is neither easy nor necessary. 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. Hence, even if the extracted LBA mapping is very accurate, it is subject to changes due to the above remapping. Here, approximation of the physical geometry was sufficient and ignoring the intended sector remapping is justifiable, as the emulator focuses on the mapping indicated by the disks observed performance when carefully benchmarked. The disk drive's physical geometry can be parameterized and the extracted values can be fitted 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 [16]. 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.

3.2.2 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

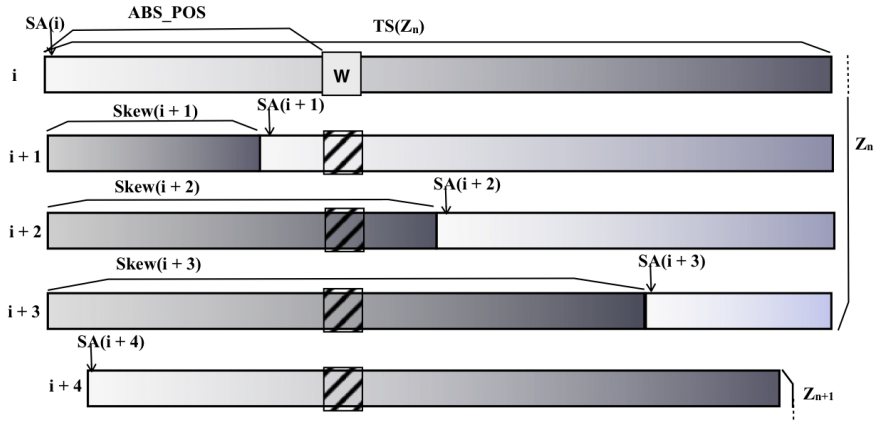


Figure 4: **Emulating Sector Overwrite**. Determining the LBA of the sectors being overwritten by a Write W requires physical geometry information.

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 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 the imaginary line because of track skew and are placed at an angular offset and after a period realigns with the 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 are 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 the 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

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

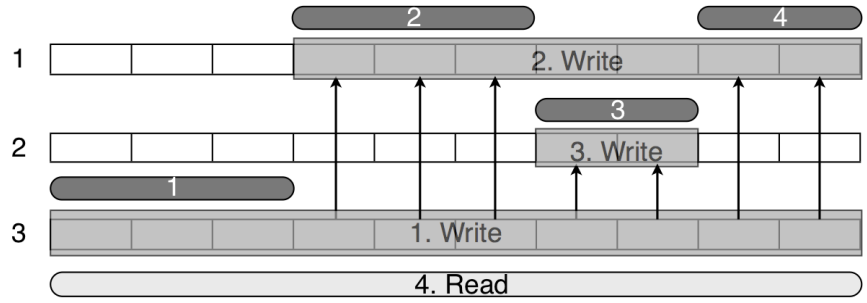


Figure 5: Random writes may result in data corruption. Read (4) after random writes (1, 2, 3) may return corrupted data or error.

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.

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

3.2.3 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 overwritten sector, *i.e.*, a sector that was overwritten by write to a different sector, is read back. As such, the emulated drive can operate in three modes, depending on the user's requirements, and the modes determine how the emulated drive behaves.

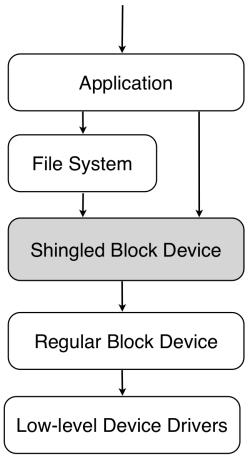


Figure 6: **Emulator in IO stack**. The Emulator is implemented as a pseudo block device driver in the Linux kernel.

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 processed as usual, but the data buffer containing the data from overwritten sectors is 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. It is up to the upper-level layers to determine whether the overwrite was intentional or unintentional.

4 Implementation

The implementation can happen in a pseudo device driver in the kernel that receives block read and write requests and performs the mapping as described above. The emulator uses 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.

The emulator is implemented as *dm-shingle*, a block IO based device-mapper target module. Linux provides a *dmsetup* utility to manage the logical devices that use the device-mapper driver. Once the *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 IO 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. A simple LBA mod number of buckets hash function would be sufficient and would ensure 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 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* like commands to report an overwrite error or the drive physical geometry 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 is crucial to have the flexibility to work on new command sets and interfaces best suited for SWDs.

5 Performance 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 8 GB RAM. To keep it relatively simple, a 160 GB single platter Seagate SATA drive was used as a test drive. A single platter disk was chosen to avoid a complicated sector layout mapping.

Disk Geometry Analyzer (DIG) [16] was used to perform geometry extraction on the test drive and is explained in detail in the next section. *fio* was used to generate desired IO patterns, to test and verify the 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 the kernel block buffers, will give a different result than if read from the disk. This problem 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 overwritten sectors unintentionally, cannot use buffering as-is.

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 aggregate bandwidth reported by *fio* is reported here.

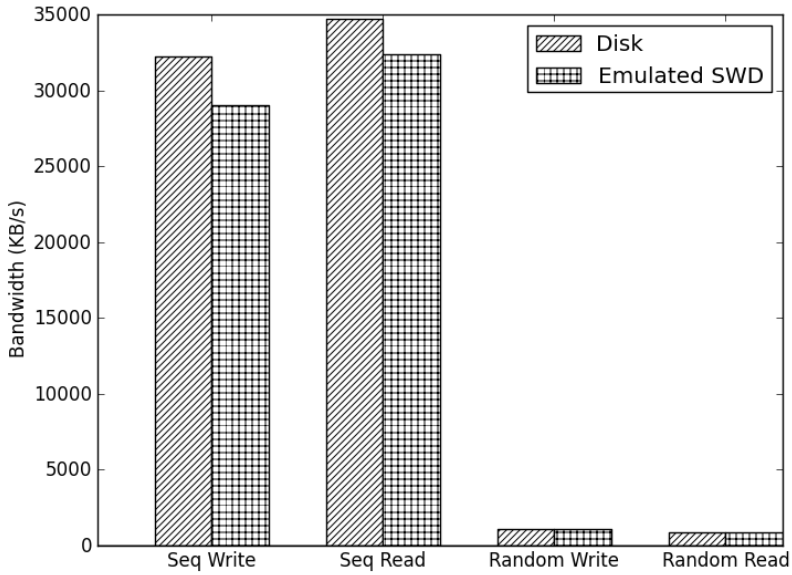


Figure 7: **Emulator performance overhead**. A slight overhead is visible during Sequential IO, but becomes negligible as disk head movement comes into picture.

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 4 KB. Figure 7 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 8 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.

The emulated SWD that overwrites one track is behaviorally the same as the regular disk. Hence, the difference in bandwidth between the 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

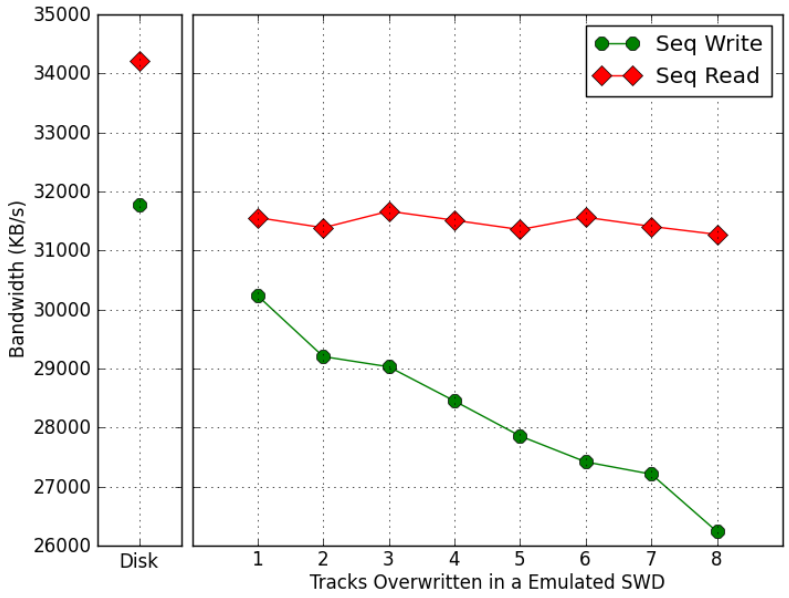


Figure 8: The number of tracks overwritten by a shingled write does not affect reads, but adds a slight overhead for every overwritten track during writes.

the higher overhead. The number of hash table buckets determine the chances of collision and may affect 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 4 KB to 1 MB, and the result can be seen in Figure 9.

Figure 9 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 anomaly where at the 128 KB 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.

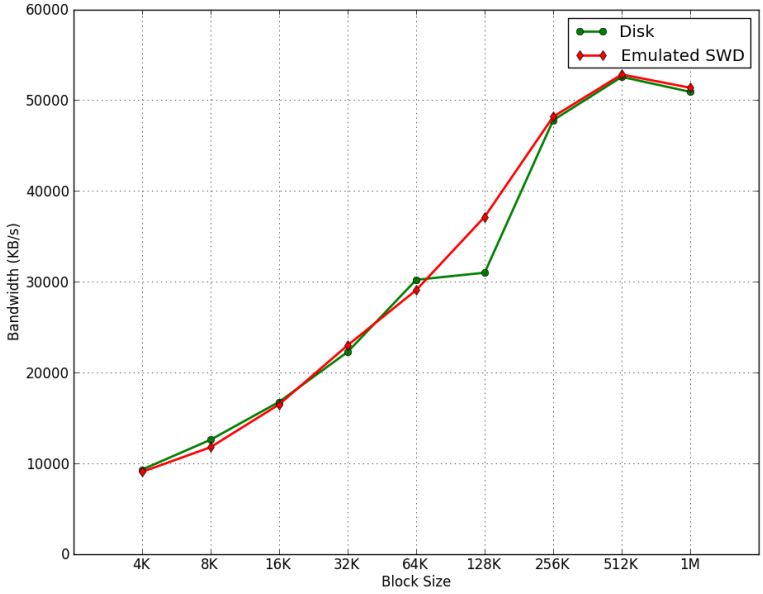


Figure 9: **Banded write performance**. Emulator overhead is negligible and the banded write performance can be improved by increasing the IO block size.

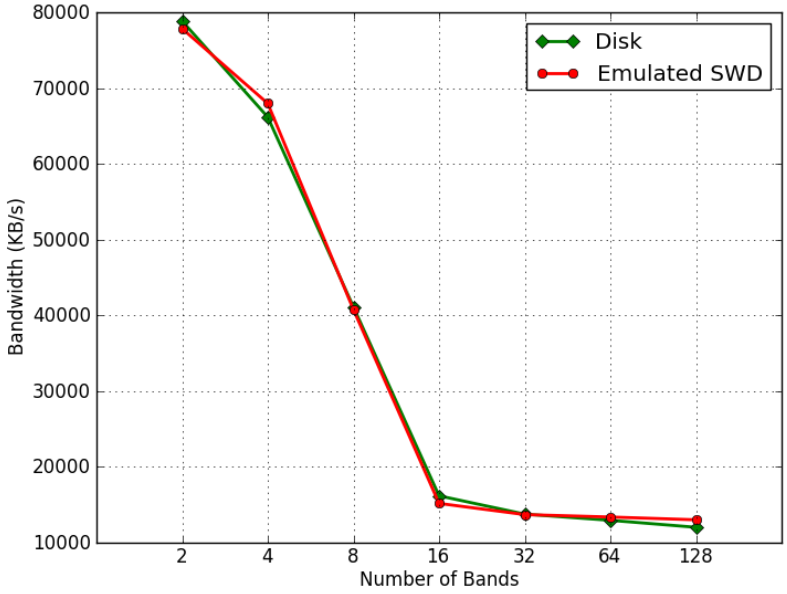


Figure 10: Emulator performance is not impacted by the number of bands.

In the final test, we check the impact of the number of bands on performance. We chose a 32 KB 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 10 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.

6 Setting up a Shingled Write Disk

This section describes how to setup a Shingled Write Disk using the emulator and a spare hard disk to experiment with. Different hard disk manufacturers use different sector layout mechanisms [16], such as surface serpentine, cylinder serpentine and hybrid serpentine schemes. A complex sector layout mechanism can make the mapping scheme quite complicated. Modifications to handle such mechanisms has not been added/tested as of now. Hence, a single platter hard disk drive is recommended with the current version of the emulator.

Since the emulator driver uses the underlying disk's physical geometry information, the emulator driver has to be customized to the test hard disk to be used. This includes the following steps: the test disk drive's geometry has to be first extracted (Section 6.1) and parameterized (Section 6.2), then the emulator driver can be configured with that information (Section 6.3). Section 6.4 shows how to use this emulator driver to create a virtual shingled block device.

6.1 Geometry Extraction

Disk Geometry Analyzer (DIG) [16], a disk characterization tool can be used to extract the disk geometry of the test drive. DIG determines the number of tracks and the size of each track in terms of number of sectors, which can be used to determine disk zoning information. DIG also 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 [16].

DIG is available as an open-source project. Instructions to compile, install and use DIG can be found in the included README file. The emulator software package also contains a version of DIG with some modifications. Since disk cache disturbs finding correct geometry information, DIG recommends disabling the disk cache first using the utility HDPARM. The following steps are required to extract the information required by the emulator.

To turn the disk cache off,

```
hdparm -A0 /dev/diskname
```

To extract the disk track boundary information,

```
./DIG /dev/diskname -d -b -w boundary_output_file
```

Extracting skew requires the output file obtained from the previous step. To extract disk Skew information,

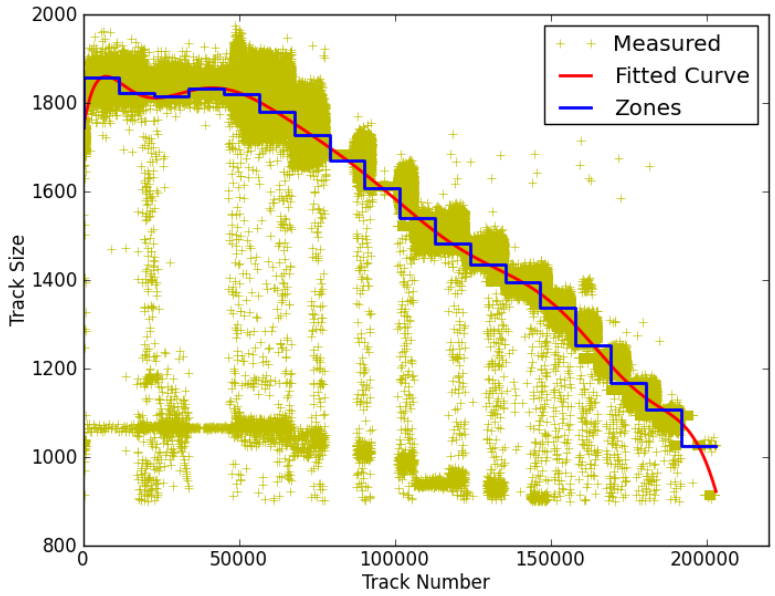


Figure 11: **Zone determination**. Plot obtained for a 160 GB single platter test disk with default values.

```
./DIG_skew /dev/diskname -d -s boundary_output_file
                            -w skew_output_file
```

Once the above steps are completed, the disk cache can be turned back on by,

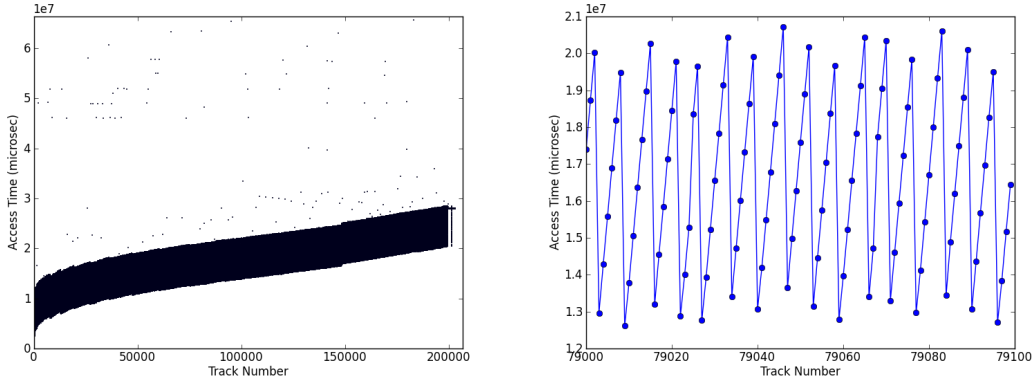
```
hdparm -A1 /dev/diskname
```

6.2 Geometry Parameterization

The results obtained from the previous section do contain some noise. But even after ignoring the noise, the results only show a zoning pattern and do not give clear zoning information, as expected. Krevat *et.al.* [15] call this behavior *adaptive zoning*, where track sizes are determined by the capabilities of disk surfaces and head characteristics post-production. Since we are only modeling the Shingled Disk behavior, approximate zoning is sufficient and the desired level of approximation can be chosen.

The SWD emulator software package also includes a python script (*geometry.py*) to read the DIG output, to fit a curve to the measured values and obtain zoning information based on that. The parameterization can be customized by varying the degree of the polynomial that is fitted and the number of zones. This script also shows a plot of the result. For example, Figure 11 shows the output plot for the 160 GB Seagate test drive.

This script also reads the DIG output and shows plots to indicate the skew measurement, one in larger scale and another in smaller scale for 100 tracks. 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



(a) Entire disk.

(b) Smaller scale.

Figure 12: **Track skew**. Plot obtained for a 160 GB single platter test disk extracted using Disk Geometry Analyzer (DIG).

$2\pi/n$. For example, Figure 12 shows the output plots for the 160 GB Seagate test drive. When satisfied with the level of approximation, the obtained values are to be fed to the emulator driver.

Below usage shows the available options and Table 1 gives the description and defaults for all available options.

```
geometry.py [-h] --dig-skewfile SKEW_FILE [--min-outlier MINOUT]
            [--max-outlier MAXOUT] [-d D] [-n N] [-s S] [--no-graphs]
            [--output-file OUTPUT_FILE]
```

Table 1: **Geometry parameterization options**

Argument	**Description**
`--dig-skewfile SKEW_FILE`	Input file obtained from DIG (*skew_output_file* obtained from the steps in Section 6.1). Required.
`--min-outlier MINOUT`	Track size value below which to considered as outlier. Optional. Default is 1000.
`--max-outlier MAXOUT`	Track size value above which to considered as outlier. Optional. Default is 2200.
`-d D`	Degree of Polynomial to fit. Optional. Default is 13.
`-n N`	Number of zones. Optional. Default is 18.
`-s S`	Track number to check skew from at a smaller scale. Optional. Default is 79000.
`--no-graphs`	Do not show graphs. Optional. Default is False.
`--output-file OUTPUT_FILE`	Output file name. Optional. Default is on-screen output.

6.3 Emulator Configuration

To configure individual virtual shingled block device, in this version of the emulator, the zoning and skew information have to be hardcoded in the driver (*dm-shingle.c*). The parameterization output file from the previous step contains the exact content to replace with and the driver code clearly indicates which line to replace. Now, the driver is ready to be compiled and installed in the system (instructions included in the emulator).

6.4 Emulator Usage

Linux provides a *dmsetup* utility to manage the logical devices that use the device-mapper driver. Once the *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.

To create a Shingled block device,

```
echo 0 `blockdev --getsize /dev/diskname` shingle /dev/diskname k
                    | dmsetup create shingled_diskname
```

Here, `/dev/diskname` is the test disk drive's block device file and k is the number of tracks overwritten/affected by a shingled write. This creates the emulated shingled block device file with the specified name under the directory `/dev/mapper/` (`/dev/mapper/shingled_diskname`).

To list all such emulated shingled block device in the system,

```
dmsetup table --target shingle
```

To remove the created device,

```
dmsetup remove shingled_diskname
```

7 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 ever-growing storage demands. Since SMR drive behaves way differently from a disk, from the host operating system perspective, it requires system software support, or complex firmware, to aid integration into existing storage systems. This project, a novel SWD Emulator furthers research in this area.

The emulator provides a unique opportunity to do a little more than Shingling. Some 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.

The emulator provides access to realistic SWD behavior, and can be easily deployed as a linux driver atop existing physical disks. Since the Shingled Write Disks are still under prototype development and the best interface for it is as-yet unknown, the flexibility offered by the emulator to modify the drive parameters with ease, and to work on new command sets and interfaces best suited for SWDs, makes it very useful for the research community.

References

[1] R. Pitchumani, A. Hospodor, A. Amer, Y. Kang, E. L. Miller, and D. D. E. Long, "Emulating a Shingled Write Disk." Accepted by *IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2012.

[2] I. Tagawa and M. Williams, "High density data-storage using shingle-write," in *Proceedings of the IEEE International Magnetics Conference*, 2009.

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

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

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

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

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

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

[9] G. Gibson and M. Polte, "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.

[10] C. K. Sann, R. Radhakrishnan, K. Eason, R. Elidrissi, J. M. Miles, B. Vasic, and A. R. Krishnan, "Channel models and detectors for two-dimensional magnetic

recording (TDMR)," *IEEE Transactions on Magnetics*, vol. 46, pp. 804–811, Mar. 2010.

[11] S. Greaves, Y. Kanai, and H. Muraoka, "Shingled recording for 2–3 Tbit/in^2," *IEEE Transactions on Magnetics*, vol. 45, pp. 3823–3829, Oct. 2009.

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

[13] "NAND Simulator." http://www.linux-mtd.infradead.org/index.html.

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

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

[16] J. Gim and Y. Won, "Extract and Infer Quickly: Obtaining Sector Geometry of Modern Hard Disk Drives," *ACM Transactions on Storage*, vol. 6, no. 2, 2010.