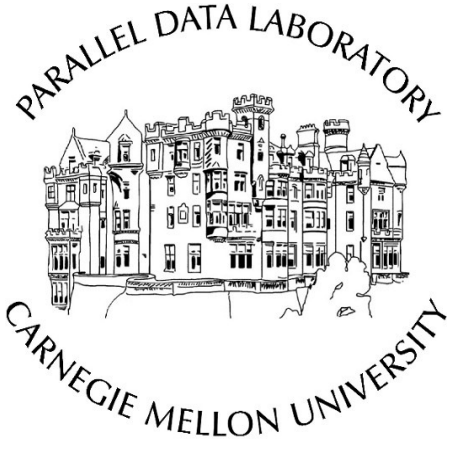

Caveat-Scriptor: Write Anywhere Shingled Disks

Saurabh Kadekodi, Swapnil Pimpale, Garth A. Gibson

CMU-PDL-15-101
April 2015

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Acknowledgements: This research was supported in part by the DOE and Los Alamos National Laboratory, under contract number DE-AC52-06NA25396 subcontract 153593-1 (IRHPIT2), the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE, www.nmc-probe.org), and Seagate through the CMU Data Storage Systems Center. We also thank the member companies of the PDL Consortium - Actifio, American Power Conversion, EMC Corporation, Facebook, Google, Hewlett-Packard Labs, Hitachi, Huawei Technologies Co., Intel Corporation, Microsoft Research, NetApp Inc., Oracle Corporation, Samsung Information Systems America, Seagate Technology, Symantec Corporation and Western Digital.

Keywords: Shingled Disks, Filesystem, Log-Structured, SMR, Segment Cleaning, Aging

Abstract

Magnetic disks, under pressure from solid state flash storage, are seeking to accelerate the rate that they lower price per stored bit. Magnetic recording technologists have begun to pack tracks so closely that writing one track cannot avoid disturbing the information stored in adjacent tracks. Specifically, the downstream track will be at least partially overwritten, or shingled by each write, and the upstream track will tolerate only a limited number of adjacent writes. Some data that was stored in the downstream track will be lost, forcing firmware or software to ensure that there was no necessary data in parts of that track.

In order to avoid deployment obstacles inherent in asking host software to change before shingled disks are sold, the current generation of shingled disks follow the model established by flash storage: a shingled translation layer of firmware in the disk remaps data writes to empty tracks and cleans (read, move, write) fragmented regions to create empty tracks. Known as Drive-Managed Shingled Disks, host software does not need to change because the disk will do extra work to cope with any write pattern that could lose data. To reduce or eliminate this extra work, changes in the hard disk API have been proposed to enable Host-Managed management of shingled disks.

This paper explores two models for Host-Managed Shingled Disk operation. The first, Strict-Append, breaks the disk into fixed sized bands and compels disk writes to occur strictly sequentially in each band, allowing only per-band-truncate-to-empty commands to recover space. This is approximately a physical realization of the classic Log-Structured File System (LFS), and shares the need for the file system to schedule and execute cleaning of bands. The second model, Caveat-Scriptor, exposes a traditional disk address space and a few shingled disks parameters: a distance in the downstream block address space that is guaranteed to never experience shingled overwrite data loss and a distance in the upstream block address space that cannot tolerate multiple adjacent writes. Host-Managed software for Caveat-Scriptor shingled disks is allowed to write anywhere, but if it fails to respect these distance parameters, it may lose data. We show in this paper that Caveat-Scriptor enables reuse of previously written and deleted data with far less cleaning than Strict-Append, enabling the potential for high-density Shingled Disks to perform almost as well as lower-density non-Shingled Disks.

1 Introduction

Magnetic disks, under pressure from solid state flash storage, are seeking to accelerate the rate that they lower price per stored bit. Magnetic recording technologists have begun to pack tracks so closely that writing one track cannot avoid disturbing the information stored in adjacent tracks [13]. Specifically, the downstream track will be at least partially overwritten, or *shingled* by each write, as shown in Figure 1, and the upstream track will tolerate only a limited number of adjacent writes. Some data that was stored in the downstream track will be lost, forcing firmware or software to ensure that there was no necessary data in parts of that track.

In order to avoid deployment obstacles inherent in asking host software to change before *shingled disks* are sold, the current generation of shingled disks follow the model established by flash storage: a *shingled translation layer* of firmware in the disk remaps data writes to empty tracks and cleans (read, move, write) fragmented regions to create empty tracks. Known as *Drive-Managed Shingled Disks* [9], host software does not need to change because the disk will do extra work to cope with any write pattern that could lose data. To reduce or eliminate this extra work, changes in the hard disk API have been proposed [1] to enable Host-Managed management of shingled disks.

This paper explores two models for *Host-Managed Shingled Disk* operation. The first, Strict-Append, breaks the disk into fixed sized bands and compels disk writes to occur strictly sequentially in each band, allowing only per-band-truncate-to-empty commands to recover space. This is approximately a physical realization of the classic Log-Structured File System (LFS) [22], and shares the need for the file system to schedule and execute cleaning of bands. The second model, Caveat-Scriptor, exposes a traditional disk address space and a few shingled disks parameters: a distance in the downstream block address space that is guaranteed to never experience shingled overwrite data loss and a distance in the upstream block address space that cannot tolerate multiple adjacent writes. Host-Managed software for Caveat-Scriptor shingled disks is allowed to write anywhere, but if it fails to respect these distance parameters, it may lose data. We show in this paper that Caveat-Scriptor enables reuse of previously written and deleted data with far less cleaning than Strict-Append, enabling the potential for high-density shingled disks to perform almost as well as lower-density non-shingled disks. Because sophisticated log-structured cleaning algorithms have long been studied, we do not anticipate a huge difference in throughput. Instead we expect the probability of very long response times to be the primary difference between a little and a lot of cleaning.

In this paper we will present a simple model for Host-Managed Caveat-Scriptor, describe a simple FUSE-based file system for Host-Managed Caveat-Scriptor, construct and describe preliminary experiments with a file system aging tool on Strict-Append and Caveat-Scriptor and report initial performance comparisons. We will show promising potential for Caveat-Scriptor to help limit heavy tail response times for shingled disks.

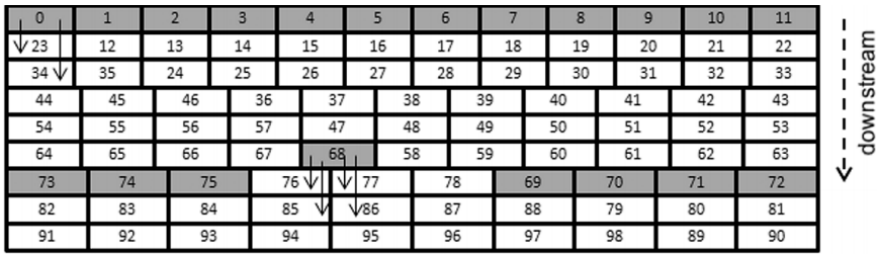

Figure 1: Feldman's [9] Figure 5 showing the effect of two writes, one to logical block address (LBA) 0 and one to LBA 68, on an example shingled disk. Arrows point to downstream LBAs that will be damaged and shading shows logically downstream addresses that will not be damaged, for LBAs 0 and 68.

2 Related Work

The viability of a shingled translation layer, akin to NAND flash translation layers was noted as Shingled Magnetic Recording (SMR) and was proposed for commercialization [11]. Most shingled disk layout research to date pursued Strict-Append variants of LFS [22]. Amer et al. [6, 5] mapped bands LFS segments and treated each band as a circular log. Cassuto et al. [7] proposed two indirections: (1) random-write zones on disk which act as a writeback cache resulting in a read-modify-write band rewrite cycle. A more sophisticated technique (2) involved creating logically contiguous circular writeback buffers on disk, called S-Blocks. Hall et al. [14] built on top of S-Blocks to aid random writes. They defined large sequential shingled runs holding most of the data, I-Regions, each with a small circular buffer called an E-Region. E-regions act as writeback caches for I-Regions and enable background cleaning of I-Regions. Lin et al. [17] exploit the difference between hot and cold data on disks. They segregated hot from cold to prevent costly cleaning operations on hot data which would naturally be deleted more frequently. They also explored temperature aware garbage collection mechanisms.

At least two file systems have been built for banded shingled disks. SFS [17] was designed for video servers, and assumed 64 MB bands, and the presence of both random and sequential shingled zones on the same disk. HiSMRfs [15] supports append-only semantics, relies on unshingled partitions and adds a RAID module to support striping across multiple shingled drives.

Recently, Aghayev et. al [3] reverse engineered a Drive-Managed shingled disk via a series of carefully crafted microbenchmarks and video recordings of the resulting disk arm movement. The disk behavior indicated the presence of an on-disk cache accepting incoming requests that were lazily written back to their final destination. Also, the shingled disk they analyzed seemed to have bands ranging from 15-40 MB across the disk's various zones.

3 Shingled Disk Model

3.1 Strict-Append

Strict-Append is a type of Host-Managed SMR [9] that restricts host writes to only occur at the write cursors of bands. The write cursor is the ending position of the last write to the band. Writes implicitly move the write cursor forward. When a band is cleaned, due to the inter-track interference (see DPID in Section 3.2), the write cursor is reset to the start of the band; it cannot be moved back to any other location except the start of the band. Bands are separated from one another by a band gap.

3.2 Caveat-Scriptor

Caveat-Scriptor is also a Host-Managed SMR [9] model in which no restrictions are enforced. Instead the host is aware of certain drive characteristics that enable it to make safe data placement decisions. Caveat-Scriptor summarizes all layout risks in a few per-drive factory-set parameters:

- **Drive Isolation Distance (DID):** When writing to a certain LBA (k), shingling may result in damage to other LBAs. Call the distance to the largest damaged LBA associated with the chosen LBA (k) the *isolation distance* of k. DID refers to the largest isolation distance observed for any LBA on the disk. Each write damages no more than DID LBAs "downstream" is a safe assumption. [1]

[1] There are some LBAs immediately following a written LBA that are always safely unaffected. The minimum number of these is called Drive No Overlap Range (DNOR) and can be used to dynamically "construct" unshingled space [9]. We do not use DNOR space in Caveat-Scriptor SMRfs yet.

- **Drive Prefix Isolation Distance (DPID):** Absent from Feldman's model from Caveat-Scriptor [9] was a simple parameter to protect upstream inter-track interference degradation of stored data. To remedy this absence, we define DPID as the largest number of preceeding LBAs that may be degraded by a specific LBA write, and require that no more than one write should be done within DPID LBAs downstream of data that may be read in the future.

For example DPID is likely to be about 1 track (on the order of a megabyte) and DID is likely to be on the order of 1-3 tracks (a few megabytes).

4 SMRfs

SMRfs is a FUSE-based file system designed for experimenting with shingled disk Host-Managed APIs. It implements both Strict-Append and Caveat-Scriptor and runs on top of a simple shingled disk model implemented on a (traditional) raw disk partition. SMRfs assumes each Host-Managed shingled disk offers two partitions, one unshingled (traditional) and one shingled. The unshingled partition allows random access and is used to store SMRfs metadata, while the 100x bigger shingled partition stores data.

SMRfs uses a simple design for managing shingled disks: it formats the small unshingled partition as a traditional file system whose files all have no data in the unshingled partition. Instead each file is split into <= 1 MB (mega) blocks. Blocks are allocated using a simple next-fit policy. Each block is written to the large shingled partition with its location recorded as an extended attribute associated with the file on the unshingled partition. We use Ext4 [18] for the unshingled partition.

SMRfs has limited internal parallelism; there is a background thread for reading and writing blocks on the shingled partition, a background thread for cleaning fragmented space on the shingled partition and a foreground thread for executing FUSE-relayed commands. SMRfs also has an internal cache with limited functionality (e.g. currently an open file must be completely instantiated in the cache).

4.1 SMRfs for Strict-Append

SMRfs on a Strict-Append Host-Managed shingled disk implements a variant of LFS [22], made simpler because all metadata is in the unshingled partition. LFS regions are the size of full bands, 32 MB in our experiments [3]. We use a cost-benefit cleaning policy consistent with LFS and use a fixed amount of work per cleaning event, 128 MB, following the advice in Matthews et. al [19]. This cleaner runs continuously in the background if the disk is idle and has less than 35% free space, as needed in the background when the disk has less than 5% free space, and synchronously in the foreground if a disk block cannot be written into an unfragmented space of its size.

4.2 SMRfs for Caveat-Scriptor

SMRfs on a Caveat-Scriptor Host-Managed shingled disk implements a traditional disk layout allocator with the following necessary Caveat-Scriptor safety policy: *protect potentially readable data.* Any run of shingled LBAs containing data that may be read again, Readable (R), must be preceeded by at least DID LBAs that are Not-Writable (NW), and followed by either a run of DPID LBAs that are Write-Once (WO), if the most recent write event to any LBA in the WO region was before the oldest write event in the last DPID LBAs in the R run, or followed by a run of DPID LBAs that are NW (if, for example, the tail section of a sequentially written run was deleted from SMRfs, so the tail of the surviving R run has already seen one nearby downstream write as shown in Figure 2).

The consequence of this safety policy is that deletion of N LBAs from a larger run of readable LBAs does not allow all N LBAs to become writable. Typically there will be DPID LBAs at the start and DID

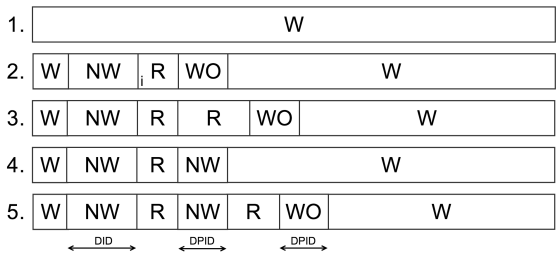

Figure 2: Caveat-Scriptor Operation Example (DID >DPID): Initially the entire disk is writable (row 1). Allocate and write an extent starting at LBA *i*. DID LBAs before *i* are then NW, and DPID LBAs following the extent are WO (row 2). Next (in row 3), allocate and write more data sequentially enlarging the R extent, and shifting the WO safety gap. Now (in row 4), delete the data just written, freeing up some LBAs occupied previously, but also converting the trailing safety gap to NW because the first written extent has been exposed to the allowed single inter-track interference, and must be rewritten before an adjacent downstream write is allowed again. Finally (in row 5), allocate and write as close as possible downstream, fragmenting the disk surface.

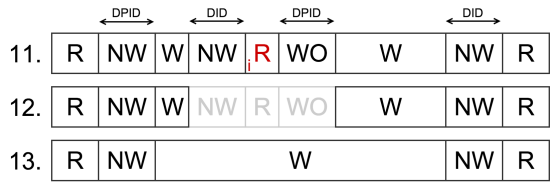

Figure 3: Free Cleaning in Caveat-Scriptor Example (DID = DPID): Starting (in row 11), with an aged and fragmented disk, with 3 runs of R data and 2 runs of W or WO LBAs, delete the entire R run at LBA *i*. Because there is no nearby readable data to protect, the adjacent NW and WO runs can be reclaimed (in row 12) and the entire enclosing writable run coalesced (in row 13).

LBAs at the end of the N LBAs deleted that are NW. This reduces writable space, and this reduction becomes worse as the disk is fragmented into smaller R and W runs.

When a deletion causes all data in a readable run of LBAs to never be readable again, SMRfs for Caveat-Scriptor does not need to reserve safety gaps at the beginning and end of the deleted run. In fact, it can reclaim adjacent NW and WO runs. As shown in Figure 3, reclaiming short NW or WO safety gaps is *free cleaning*. Strict-Append SMRfs can only do this when all LBAs before the write cursor in a band have been deleted, so free cleaning in Caveat-Scriptor SMRfs is much more likely to happen.

5 Preliminary Evaluation

Our experiments use the PRObE Marmot cluster [10] whose nodes have dual core 64 bit, 1.6 GHz AMD Opteron-242 processors along with 16 GB of RAM and a single 2 TB WDC SATA 7200rpm disk.

Because the behavior of shingled disks is driven by allocation and fragmentation, we built and use an aging tool before running benchmarks [23]. Our aging tool is parameterized to achieve a target fraction of the disk containing user data (70% of a 100 GB disk) using as inputs a distribution of file sizes (Figure 4 b) taken from a study of a Yahoo! Hadoop Cluster [8] and a distribution of file ages (Figure 4 c) accelerated (by a factor of 1 sec = 6.08 hrs of the reference distribution) to match an observed pattern [4]. Long runs of creates and deletes (with probability 0.9) are performed rather than simple alternation while maintaining the

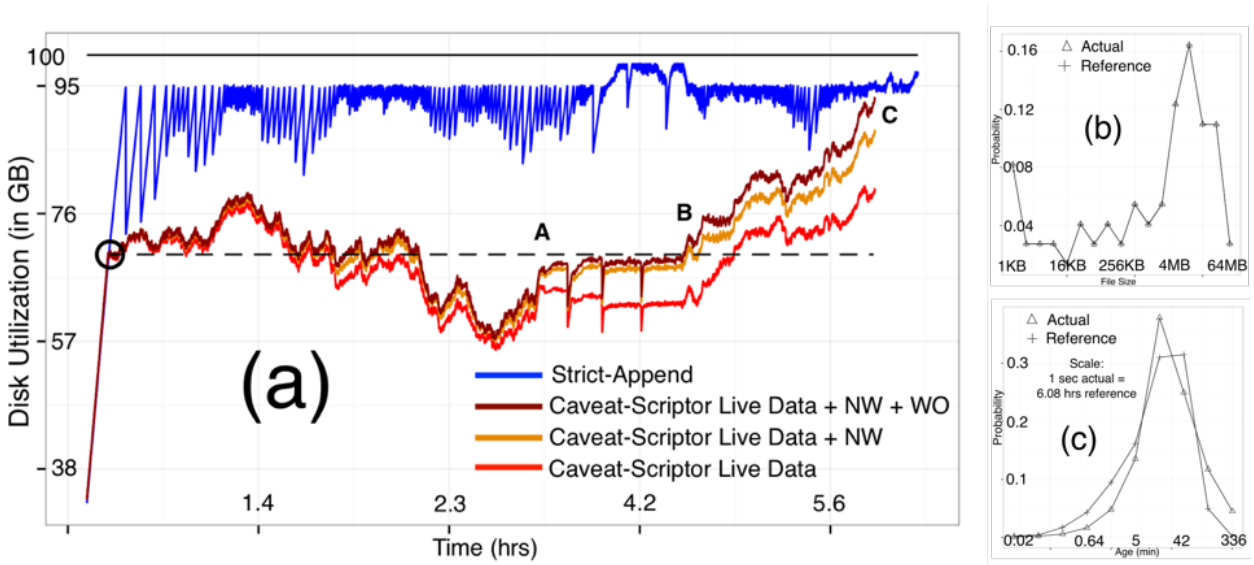

Figure 4: Space availability under test (a) and aging tool file size and file age distributions (b) and (c) (obtained from [4] and [8] respectively). We age a 100 GB disk to 70% utilization (shown by the dashed line) via successive create + write and delete operations (without fsync) imitating 100% cache hits for reads. At time A, 800 GB worth of aging had been done before starting two successive runs of Postmark (file sizes 3-7 MB) then two successive runs of the Filebench fileserver profile (mean file size 9 MB) while the aging tool was slowed down to half its normal speed. Time B shows the end of the second Filebench run which left 4.5 GB additional user data on the disk. The aging tool then resumes full speed until, time C, when Caveat-Scriptor finally runs out of SMRfs block-sized free space. The end point of Strict-Append running the same workload is when it had transfered the same amount of user data as Caveat-Scriptor.

target utilization.

Figure 4 shows a test run of Strict-Append SMRfs versus Caveat-Scriptor SMRfs involving a 5+ hour execution of our aging tool on a 100 GB partition of a traditional hard disk encapsulated by each shingled disk API model. In both runs, the aging tool follows the same pseudo-random sequence, and in both we inject four benchmark runs into the middle, at the same point in the aging tool progress (marked at time A). The four benchmarks are two invocations of Postmark [16] and two invocations of Filebench [20].

In the first few minutes, in Figure 4 a, the aging tool achieves 70% utilization of the disk capacity. Strict-Append SMRfs (blue line) continues to fill the disk, cleaning for the first time when only 5% of the disk is available for new writes. Because the aging tool is running constantly throughout the test, there is never any idle time cleaning, so Strict-Append SMRfs triggers a single 128 MB of work cleaning each time it falls to 5% writable space; only suffering synchronous foreground cleaning during the very busy benchmarking runs.

Caveat-Scriptor SMRfs (red, yellow, brown lines) achieves free cleaning throughout the test. The lowest (red) line shows the total amount of user data being mutated by the aging tool; the middle (yellow) line shows the user data plus NW space and the top (brown) line shows the user data plus NW plus WO space. Because the aging tool emulates a Hadoop style workload, with large file sizes, free cleaning works quite well over hours. During the benchmarks, however, even with large average file sizes, considerable NW space accumulates (and persists because the aging tool will never delete benchmark files).

Figure 5 shows the average throughput of the two runs of Postmark (on the right) for Strict-Append versus Caveat-Scriptor. While there is a small average throughput advantage for Caveat-Scriptor, the use of small (128 MB) cleaning work each time cleaning is needed in Strict-Append limits the overall slowdown

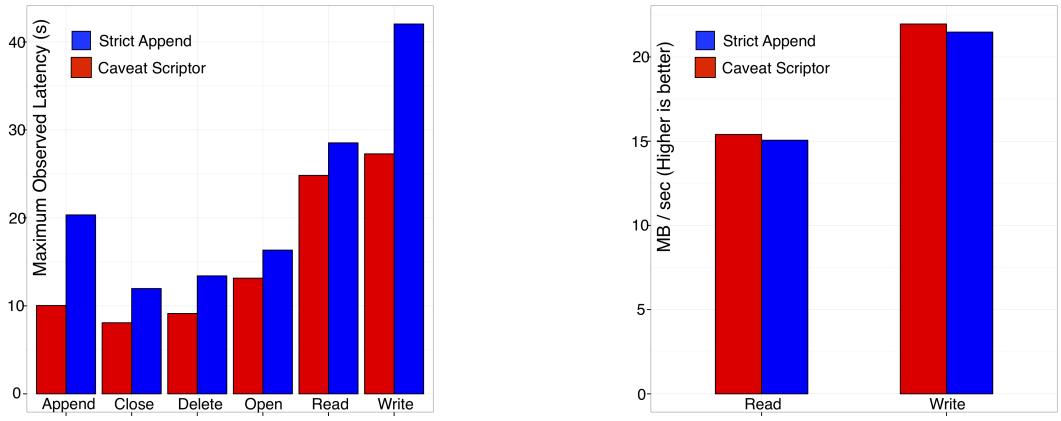

Figure 5: Maximum latency observed by Filebench (left) when executing in parallel with the aging tool as shown in Figure 4. Average read and write performance when running Postmark (right) along with the aging tool as shown in Figure 4.

it experiences. On the left in Figure 5, we show the maximum response times of events in the two runs of Filebench (because this is a timed interactive benchmark). Here the penalty for more frequent cleaning in Strict-Append is more evident.

One interpretation for the behavior in Figure 4 a is that Caveat-Scriptor, although able to defer the continuous cleaning seen in Strict-Append, is not able to avoid building up space lost to safety gaps beyond 5.5 hours in this test. Eventually disk space will be fragmented into enough small safety gaps that continuous cleaning will be needed in both systems. Like solid state disks (SSD), if the data mutation workload does not leave enough idle system opportunities for extensive cleaning (defragmentation), peak write throughput and response times will degrade.

The difference, however, is that after sufficient disk idleness for defragmentation, should an intense mutation period start again, Caveat-Scriptor SMRfs will be able to operate without cleaning for much longer than Strict-Append. Provided the duty cycle for user work in the shingled disk is not 100% (and disk idleness is usually very common [12]), Caveat-Scriptor may be able to hide all cleaning, while Strict-Append will need a very small duty cycle to hide much cleaning.

6 Discussion and Conclusion

While Strict-Append SMRfs has a sophisticated cleaner based on 20+ years of LFS research, cleaning in Caveat-Scriptor is a new topic for storage research. In addition to free cleaning and the resulting deferred cleaning, Caveat-Scriptor cleaning can be "a little at a time", potentially causing much less impact on user response times.

Caveat-Scriptor can provide cleaning flexibility, but it can also enable catastrophic off-by-one software layout bugs. For this reason we expect drive vendors to be slow to offer Caveat-Scriptor APIs for Host-Managed disks. However, even a drive-managed shingled disk needs a disk model for its firmware engineers to program. Perhaps using an internal Caveat-Scriptor API might allow Drive-Managed shingled disks to offer less variable response times.

Finally, Caveat-Scriptor is similar to copy-on-write (CoW) file systems like Btrfs [21] in terms of treatment of in-place updates. With efforts already being made to make Ext4 SMR-friendly [2], incorporating Caveat-Scriptor in a CoW file system seems a natural next step.

7 Acknowledgements

We would particularly like to thank Timothy Feldman, David Anderson and Michael Miller from Seagate Technology for their valuable guidance along with the rest of the members of the SMR project at CMU for their critical contributions to this work, including Anand Suresh, Jainam Shah, Xu Zhang, Chinmay Kamat, Praveen Ramakrishnan, Pavan Kumar Alampalli, Fan Xiang, Aditya Jaltade, Amod Jaltade, Hartaj Dugal, Mukul Kumar Singh, Pratik Shah, Tejas Wanjari, Ravi Chandra Bandlamudi, Fiona Britto and Omkar Gawde.

This research was supported in part by the DOE and Los Alamos National Laboratory, under contract number DE-AC52-06NA25396 subcontract 153593-1 (IRHPIT2), the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE, www.nmc-probe.org), and Seagate through the CMU Data Storage Systems Center. We also thank the member companies of the PDL Consortium - Actifio, American Power Conversion, EMC Corporation, Facebook, Google, Hewlett-Packard Labs, Hitachi, Huawei Technologies Co., Intel Corporation, Microsoft Research, NetApp Inc., Oracle Corporation, Samsung Information Systems America, Seagate Technology, Symantec Corporation and Western Digital.

References

[1] T10 zoned block commands (zbc). *"URL: http://www.t10.org/cgi-bin/ac.pl?t=f&f=zbc-r02.pdf"*, 2013.

[2] Smr friendly ext4. *URL: https://github.com/Seagate/SMR_FS-EXT4*, 2015.

[3] Abutalib Aghayev and Peter Desnoyers. Skylight - a window on shingled disk operation. In *FAST 2015*. USENIX, 2015.

[4] Nitin Agrawal, William J Bolosky, John R Douceur, and Jacob R Lorch. A five-year study of file-system metadata. *ACM Trans. on Storage (TOS)*, 2007.

[5] Ahmed Amer, JoAnne Holliday, Darrell DE Long, Ethan L Miller, J Paris, and Thomas Schwarz. Data management and layout for shingled magnetic recording. *Magnetics, IEEE Trans. on*, 2011.

[6] Ahmed Amer, Darrell DE Long, Ethan L Miller, J-F Paris, and SJ Thomas Schwarz. Design issues for a shingled write disk system. In *MSST, 2010 IEEE*. IEEE, 2010.

[7] Yuval Cassuto, Marco AA Sanvido, Cyril Guyot, David R Hall, and Zvonimir Z Bandic. Indirection systems for shingled-recording disk drives. In *MSST 2010 IEEE*. IEEE, 2010.

[8] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. Diskreduce: Replication as a prelude to erasure coding in data-intensive scalable computing. In *SC11*, 2011.

[9] Tim Feldman and Garth Gibson. Shingled magnetic recording areal density increase requires new data management. *USENIX;login issue*, 2013.

[10] Garth Gibson, Gary Grider, Andree Jacobson, and Wyatt Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX; login*, 2013.

[11] Garth Gibson and Milo Polte. Directions for shingled-write and twodimensional magnetic recording system architectures: Synergies with solid-state disks. *CMU, Pittsburgh, PA, Tech. Rep. CMU-PDL-09-014*, 2009.

[12] Richard Golding, Peter Bosch, John Wilkes, USENIX Association, et al. Idleness is not sloth. In *USENIX*, 1995.

[13] Simon Greaves, Yasushi Kanai, and Hiroaki Muraoka. Shingled recording for 2-3 tbit/in2. *IEEE Trans. on Magnetics*, 2009.

[14] David Hall, John H Marcos, and Jonathan D Coker. Data handling algorithms for autonomous shingled magnetic recording hdds. *Magnetics, IEEE Trans. on*, 2012.

[15] Chao Jin, Wei-Ya Xi, Zhi-Yong Ching, Feng Huo, and Chun-Teck Lim. Hismrfs: A high performance file system for shingled storage array. In *MSST, 2014*. IEEE, 2014.

[16] Jeffrey Katcher. Postmark: A new file system benchmark. Technical report, Technical Report TR3022, NetApp, 1997. www. netapp. com/tech_library/3022. html, 1997.

[17] Chung-I Lin, Dongchul Park, Weiping He, and David HC Du. H-swd: Incorporating hot data identification into shingled write disks. In *MASCOTS, 2012*. IEEE, 2012.

[18] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Linux Symposium*, volume 2, 2007.

[19] Jeanna Neefe Matthews, Drew Roselli, Adam M Costello, Randolph Y Wang, and Thomas E Anderson. *Improving the performance of log-structured file systems with adaptive methods*. ACM, 1997.

[20] Richard McDougall and Jim Mauro. Filebench. *URL: http://www.nfsv4bat.org/Documents/nasconf/2004/filebench.pdf*, 2005.

[21] Ohad Rodeh, Josef Bacik, and Chris Mason. Btrfs: The linux b-tree filesystem. *ACM Trans. on Storage (TOS)*, 2013.

[22] Mendel Rosenblum and John K Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. on Computer Systems (TOCS)*, 1992.

[23] Keith A Smith and Margo I Seltzer. File system agingincreasing the relevance of file system benchmarks. In *ACM SIGMETRICS*. ACM, 1997.