Virtual Guard: A Track-Based Translation Layer for Shingled Disks

Mansour Shafaei
Northeastern University

Peter Desnoyers
Northeastern University

Abstract

Virtual Guard (Vguard) is a track-based static mapping translation layer for shingled magnetic recording (SMR) drives. Data is written *in-place* by caching data from the next track in the shingling direction, allowing direct overwrite of sectors in the target track. This enables Vguard to take advantage of track-level locality, nearly eliminating cleaning for many workloads. We compare performance of Vguard to an available drive-managed SMR drive analyzed and modeled in previous research. Vguard reduces the 99.9% latency by 15× for real-world traces, and maximum latency by 32% for synthetic random write workloads.

1 Introduction and Related Work

Shingled Magnetic Recording (SMR) offers the opportunity for significant increases in disk drive density without radical changes in drive design and manufacturing. This comes at a cost—when writing track T, data on another track (i.e. track $T+1$) may be lost. This limitation may be addressed on the host, using new SCSI and SATA extensions; however many SMR drives sold today are drive-managed SMR (DM-SMR) devices, using an internal translation layer for plug-compatibility with non-SMR disks.

A number of shingling translation layer (STL) mechanisms have been proposed to date [6, 9, 5, 2, 8]; however based on measurements of shipping drives, the dominant ones may be termed E-Region STLs, where writes are made to an exception region (or persistent cache), and later migrated to a permanent location elsewhere on disk. In a common variant [1] the drive is divided into bands separated by unused tracks or *guard tracks*, allowing a band to be completely overwritten by the cleaning process without damage to "downstream" tracks, and LBAs are statically mapped to "home locations" in these bands.

With one exception, STLs proposed to date have used LBA mapping, leaving details of the disk geometry to lower layers of drive firmware. In contrast, He and Du [3, 4] have proposed two mechanisms for *track mapping* STLs, where translation is performed *after* mapping an LBA to a track/head/sector location. Their first work describes a number of track mappings which allow efficient writes to low-numbered tracks in cases where the high-numbered tracks have yet to be written; more recently SMaRT [4] is more general, using dynamic mapping of tracks. SMaRT is highly sensitive to utilization, with a large (64 MB) map which requires 100s of milliseconds to persist to disk and must be held in DRAM, and with poor performance at 90% utilization or above[1]. Finally, SMaRT does not handle differences in track size due to varying track length, adaptive formatting [7], or slip sparing [11].

We propose Virtual Guard (Vguard), a novel track-mapped STL. Before modifying track T, Vguard migrates track $T+1$ to cache, allowing multiple *in-place* modifications to track T. It maps each track to a static home location, eliminating most track size issues. The persistent cache is comprised of large, outer-diameter tracks, allowing most cached tracks to occupy a single cache track along with a small metadata header. On-disk overhead is modest, consisting of a small ($<0.5\%$) persistent cache and a single guard track per band, and RAM overhead is almost negligible.

We present the design of Vguard, as well as simulation results showing significant performance improvements over traditional E-region STLs while imposing the same or lower disk space and DRAM overheads. Vguard is seen to take advantage of strong spatial locality in these workloads, avoiding cleaning in all cases except synthetic traces.

[1] The cost to increase RAM from 128MB to 256 MB to accommodate this table ($0.70) is large compared to an estimated 10% profit margin on drives which may sell wholesale at $50; a loss of 10% capacity is also likely to be economically unfeasible.

2 Virtual Guard

Virtual Guard is a track-mapped STL, where LBAs are mapped to a track number, which is then mapped to a physical track location. A small region at the outer diameter is reserved as a persistent cache; the remainder of the drive is divided into bands separated by empty (guard) tracks, and the LBA space of the drive is mapped onto these non-cache tracks. Unlike most STLs (e.g. E-Region) which direct writes to the persistent cache, Vguard copies the *next track* (in the shingling direction) into the persistent cache, forming a "virtual guard" which allows the target track to be modified *in place* without data corruption.

2.1 In-Place Writes

Given a host write to LBA X, Vguard calculates the track and band number (T_X,B_X), and then checks if (a) T_X is the last track in band B_X or (b) track T_{X+1} is already cached.

If neither is the case, then Vguard reads the content of T_{X+1} and writes it to the next free track in persistent cache (T_{wf}). Track T_X may now be modified in place[2] without risk to data in track T_{X+1}. Assuming the persistent cache is also shingled, Vguard uses the same approach when writing to tracks in cache—if the following track is valid then it is copied to the next free track in cache, and then writes are performed in-place.

Figure 1 demonstrates examples of different write scenarios in Vguard.

2.2 Map Persistence

Vguard maintains an *exception map*, indicating which tracks have been relocated into cache and where. Due to the relatively small size of the persistent cache and the large granularity (one track) of the mapping, this map is extremely small in comparison to the ∼60 MB reported in SMaRT[4] or the ∼1 MB map seen in commercial E-region-based devices [12]. In particular, if N_{MTE} is the number of entries in the mapping table and N_{tr} and N_{trc} are size of the drive and the persistent cache, respectively, in units of tracks, the mapping table size would be roughly:

$$MapSize = N_{MTE} \times (log(N_{tr}) + log(Ntrc)) \qquad (1)$$

For a 5 TB drive with 24 GB of persistent cache and $\sim 4 \times 10^6$ tracks (see Skylight [1]), the mapping table size would be roughly 30K. The map is small enough that a full copy can be appended to each track in persistent cache with minor overhead. (Note that on occasion this, or other track size differences, will make it necessary to allocate two cache tracks; however this will be rare.)

[2]The number of times it may be safely overwritten may be limited due to upstream track interference, but is $\gg 1$.

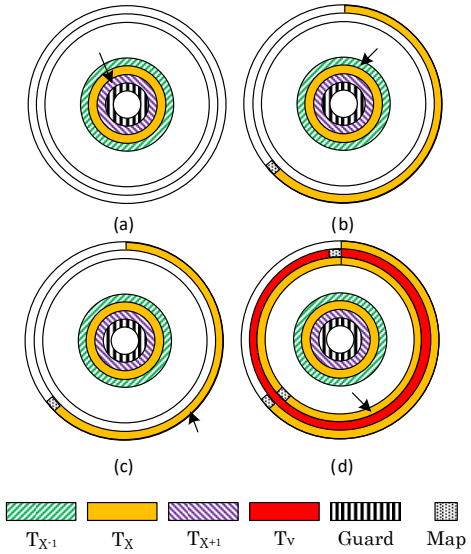

Figure 1: (a) Write in-place on T_{X+1} (the last track in its band with the guard track in front); (b) Write in-place on T_{X-1} after caching T_X; (c) Write in-place on cached track T_X where it is located right before the log head in persistent cache; (d) Write on track T_X after relocating it to the write frontier in cache (as there was a valid track (T_v) in front of it in its previous location).

2.3 Cleaning

The cleaning process moves tracks from the persistent cache back to their home locations. Vguard does not perform background cleaning, but rather triggers cleaning when either:

1. the number of free tracks in the mapping table drops below a threshold α, or
2. the distance between the oldest track in persistent cache and the write frontier (including any invalidated tracks) exceeds a second threshold β.

In a single cleaning episode Vguard will clean two bands, for each band repeating the following process: pick the first two tracks in the tail of the log, and then clean all tracks from cache with home locations in the same band as either of the two tracks. The specific cleaning steps are:

1. Pick the track at the tail of the log, with home location in band B.
2. Read all other tracks in cache with home locations in band B.
3. Read band B.
4. Merge data for band B in memory and write to the *scratch pad*, a reserved section of persistent cache; this prevents data loss if power fails while overwriting the

data band.

5. Overwrite band B with the merged data.

Vguard also adds the following two optimizations to the cleaning process:

1) If the first N tracks are not in cache, they can be skipped by the cleaning process (both the read-data-band-copy-to-scratch and the overwrite-data-band parts). This reduces the worst-case cleaning overhead (when only one track is in cache) by 50% on average. The same optimization is used in DM-SMR drives on the market [1].

2) If multiple tracks from the same band are cached, Vguard may choose to perform a *partial cleaning* of the band. Rather than re-writing to the end of the band, rewriting stops at track $T-1$, where track T is in cache when cleaning starts.

To reduce the buffer memory required and duration for which host I/Os are stalled, a band may be cleaned in multiple stages, with each stage reading, persisting, and then re-writing a single buffer of data; read (and sometimes write) operations are interleaved between stages. Vguard uses a cleaning buffer size of 15 MB, roughly the same that seen in commercial DM-STL drives analyzed to date [12, 1].

Although E-region-based STLs typically perform background cleaning, this would be counter-productive for Vguard. By moving tracks into cache and introducing virtual guard tracks, Vguard allows repeated writes to write-hot locations without the need to copy data; cleaning would eliminate those spaces and require additional data movement before the hot locations could be written to again.

3 Evaluation

Vguard is implemented as extensions to an existing and accurate Python simulator for DM-SMR devices [12]. We compare Vguard with the simulated performance of the Seagate 5 TB drive ("DM-SMR"), using parameters and drive settings shown in Table 1. No modification or scaling of trace LBAs was performed; traces were replayed "flat out" without delay between operations and with a queue depth of 1, and the (simulated) write cache was disabled.

3.1 MSR Traces

Sixteen traces from the Microsoft Research trace set [10] were simulated, representing a wide range of read/write ratios and total trace size. In Figure 2 latency CDFs are shown for both Vguard and DM-SMR, with Vguard latency being seen to be dramatically lower. In some cases, e.g. src2_0 and mds_0, the 99^{th} percentile latency for Vguard is significantly lower than the 10^{th} percentile latency for DM-SMR. Details of tail latency may be seen in Table 2. 95% latency for Vguard is seen to be roughly a single rotation (∼10ms) for all traces tested, and maximum latency is reduced from multiple seconds to ∼50ms in many cases[3].

Parameter	Vguard	DM-SMR
Size	5TB	5TB
Form factor	3.5"	3.5"
RPM	5980	5980
Track lengths	1.8-0.9MB	1.8-0.9MB
Mapping type	Static	Static
Band size	20 tracks	20 tracks
Cleaning granularity	2 Bands	2 Bands
Cache location	Outer diameter	Outer diameter
Cache size	13.8K Tracks (24GB)	13.8K Tracks (24GB)
Mapping table size	∼30KB	∼1.3MB
α	9194	22986
β	9194	22986
Write cache	Disabled	Disabled
Read ahead	Disabled	Disabled

Table 1: Simulated drive parameters for Vguard and DM-SMR

Due to spatial locality in the traces examined, the guard track set fits completely within cache and Vguard is able to perform an arbitrary number of writes in place (subject to upstream interference limitations) without cleaning. In contrast, DM-STL consumes space in both persistent cache and the map with every write, entering cleaning for all of the longer write-intensive traces. Figure 3 shows the Vguard cache utilization for various MSR traces, showing both the maximum number of active tracks and the total number of tracks utilized, both as a percentage of total persistent cache size. We see that none of these traces require more than 13% of the 24 GB cache at any instance, and the total number of tracks copied to cache (i.e. the log length) is never more than 38% of the cleaning threshold trigger (β).

3.2 CloudPhysics Traces

Additional experiments were performed on a more modern set of traces courtesy of CloudPhysics [13], representing real workloads on large volumes. In Table 3 we see the total number of writes and the volume capacity for each tested trace. In Figure 4 we see the results of these runs; again Vguard is able to handle the entire trace without entering cleaning.

In fact, analysis of the number of unique tracks modified in each of the 100+ CloudPhysics traces, seen in Figure 5, shows that none of these traces have a write footprint large enough to force Vguard cleaning, even with a small 24 GB persistent cache.

[3] In fact, the 100ms+ maximum latencies seen for rsrch_0 and stg_0 in Table 2 are mostly due to very large (e.g. 6 MB) read operations.

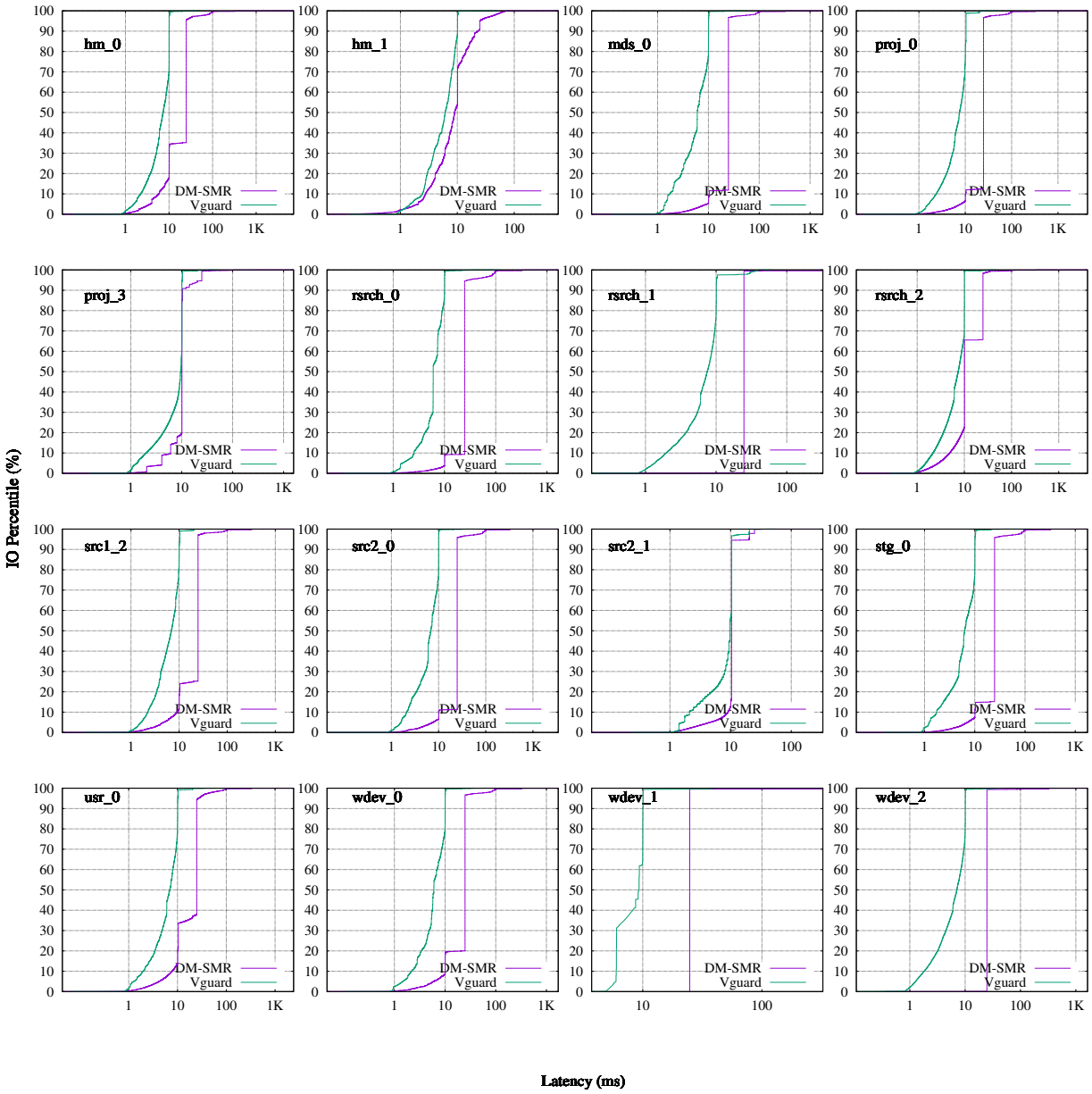

Figure 2: Latency CDFs for MSR traces on Vguard vs DM-SMR

3.3 Random Writes

To evaluate Vguard's cleaning performance, we generate 4 synthetic traces with 100K random writes spanned over 500GB, 1TB, 2TB, and 4TB address spaces and run them on Vguard and DM-SMR. Latency CDFs for these experiments are shown in Figure 6. Vguard shows moderate improvements of about 30% in 99^{th} percentile and maximum latency, while 90^{th} percentile latency improves drastically for the smaller-footprint trace (500 GB) but is slightly worse than DM-SMR for the other cases. The similarity of results between Vguard and DM-STL for the larger traces is expected, as in each case each write forces one band to be cleaned, with only modest differences in overhead between the two STLs. (Note also that random write performance is thus limited by band size—larger band sizes with lower space overhead will result in poorer random write performance.)

4 Conclusion

Vguard represents a novel approach to shingling translation layers, using persistent cache space for non-written tracks while performing writes in-place. As a result, consumption of cache space is not a function of the volume of data written, but rather of the pattern of LBAs which are written to,

	95%		99%		99.5%		99.9%		Max	
Trace	Vguard	DM-SMR	Vguard	DM-SMR	Vguard	DM-SMR	Vguard	DM-SMR	Vguard	DM-SMR
hm_0	10.08	24.78	10.4	95.01	10.97	97.58	21.57	324.78	53.03	7,234.81
hm_1	10.1	24.78	10.4	56.55	10.4	64.06	20.13	74.62	50.60	592.04
mds_0	10.08	24.78	10.4	95.01	10.4	95.81	30.12	324.78	50.52	1,789.66
proj_0	10.4	24.78	18.7	95.01	20.43	99.78	21.56	324.78	52.29	4,341.17
proj_3	10.38	24.78	10.4	24.78	20.11	30.5	30.38	102.35	53.88	1,573.94
rsrch_0	10.08	28.79	10.31	96.24	10.68	105.24	30.92	324.78	117.81	1,657.22
rsrch_1	10.14	24.78	31.78	24.78	37	41.18	39.46	324.78	62.52	324.78
rsrch_2	10.08	24.78	10.08	31.59	10.1	81.64	30.79	324.78	47.83	4,082.31
src1_2	10.38	24.78	10.4	85.64	20.09	95.63	20.43	324.78	50.62	2,400.70
src2_0	10.08	24.78	10.18	95.09	10.4	98.77	30.21	324.78	60.59	3,428.12
src2_1	10.4	20.18	20.43	24.78	20.43	24.78	20.44	29.72	58.85	332.20
stg_0	10.11	24.78	10.40	95.08	14.89	97.47	30.12	324.78	123.92	1754.54
usr_0	10.16	26.96	10.4	85.3	20.18	95.36	24.85	324.78	158.74	2,396.69
wdev_0	10.08	24.78	10.23	95.01	10.49	97.19	30.14	324.78	56.56	1,710.40
wdev_1	10.08	24.78	10.08	24.78	10.38	24.78	38.04	324.78	39.25	324.78
wdev_2	10.1	24.78	10.19	24.78	11.51	95.01	33.21	324.78	49.32	1,628.12

Table 2: Latency percentiles for MSR traces on Vguard and DM-SMR in milliseconds

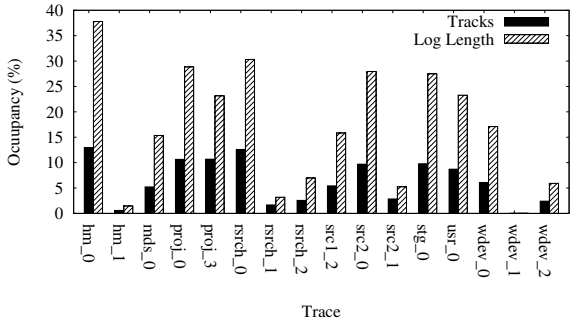

Figure 3: Vguard cache utilization by traces (all MSR traces)

Trace	Number of writes	Drive Size
w27	3,182,636	1.95TB
w29	2,707,559	1.95TB
w53	4,162,497	1.5TB
w54	8,648,118	3.6TB

Table 3: Characteristics of tested CloudPhysics traces

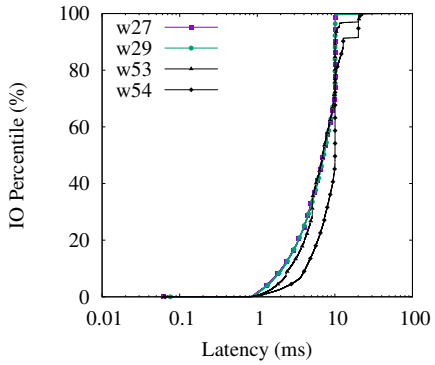

Figure 4: Latency CDFs for CloudPhysics traces on Vguard

without regard to the number of times they are overwritten. In many real-world cases the guard track set is seen to fit comfortably within a rather small persistent cache, potentially offering near-conventional-drive levels of performance as all writes may be performed in-place. Further work is needed

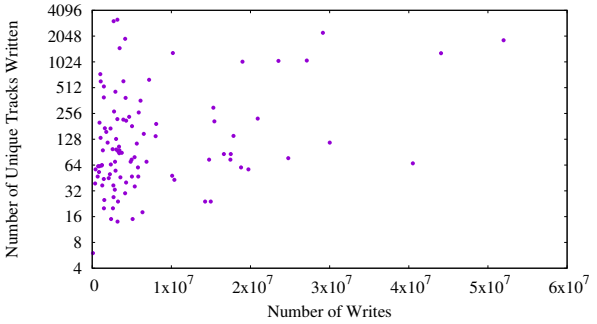

Figure 5: Approximate number of tracks modified in CloudPhysics traces, assuming fixed 1MB track size

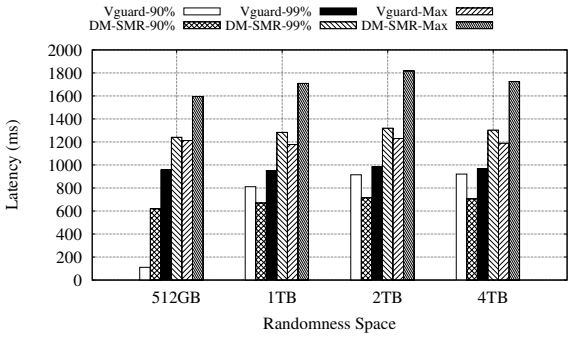

Figure 6: Latency CDFs for random traces on Vguard vs DM-SMR

to compare Vguard with conventional drive performance, as well as to validate performance on real hardware.

References

[1] AGHAYEV, A., SHAFAEI, M., AND DESNOYERS, P. Skylight—a window on shingled disk operation. *Trans. Storage 11*, 4 (Oct. 2015), 16:1–16:28.

[2] CASSUTO, Y., SANVIDO, M. A., GUYOT, C., HALL, D. R., AND BANDIC, Z. Z. Indirection systems for shingled-recording disk drives. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (2010), IEEE, pp. 1–14.

[3] HE, W., AND DU, D. H. Novel address mappings for shingled write disks. In *Proceedings of the 6th USENIX conference on Hot Topics in Storage and File Systems* (2014), USENIX Association, pp. 5–5.

[4] HE, W., AND DU, D. H. Smart: An approach to shingled magnetic recording translation. In *15th USENIX Conference on File and Storage Technologies (FAST 17)* (Santa Clara, CA, 2017), USENIX Association, pp. 121–134.

[5] JONES, S. N., AMER, A., MILLER, E. L., LONG, D. D. E., PITCHUMANI, R., AND STRONG, C. R. Classifying data to reduce long-term data movement in shingled write disks. *Trans. Storage 12*, 1 (Feb. 2016), 2:1–2:17.

[6] KADEKODI, S., PIMPALE, S., AND GIBSON, G. A. Caveat-Scriptor: Write Anywhere Shingled Disks. In *7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 15)* (Santa Clara, CA, 2015), USENIX Association.

[7] KREVAT, E., TUCEK, J., AND GANGER, G. R. Disks are like snowflakes: no two are alike. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems* (2011), USENIX Association, pp. 14–14.

[8] LIN, C.-I., PARK, D., HE, W., AND DU, D. H. H-swd: Incorporating hot data identification into shingled write disks. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on* (2012), IEEE, pp. 321–330.

[9] MANZANARES, A., WATKINS, N., GUYOT, C., LEMOAL, D., MALTZAHN, C., AND BANDIC, Z. Zea, a data management approach for smr. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)* (Denver, CO, 2016), USENIX Association.

[10] NARAYANAN, D., DONNELLY, A., AND ROWSTRON, A. Write off-loading: practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (San Jose, California, 2008), USENIX Association, pp. 1–15.

[11] RUEMMLER, C., AND WILKES, J. An introduction to disk drive modeling. *Computer 27*, 3 (Mar. 1994), 17–28.

[12] SHAFAEI, M., HAJKAZEMI, M. H., DESNOYERS, P., AND AGHAYEV, A. Modeling smr drive performance. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (New York, NY, USA, 2016), SIGMETRICS '16, ACM, pp. 389–390.

[13] WALDSPURGER, C. A., PARK, N., GARTHWAITE, A., AND AHMAD, I. Efficient MRC Construction with SHARDS. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (2015), USENIX Association, pp. 95–110.