μCache: a mutable cache for SMR translation layer

Mohammad Hossein Hajkazemi[⋆], Mania Abdi[†], Peter Desnoyers[†]
Department of Electrical and Computer Engineering[⋆], Khoury College of Computer Sciences[†]
Northeastern University
hajkazemi@ece.neu.edu,abdi.ma@husky.neu.edu,pjd@ccs.neu.edu

***Abstract*—Shingled Magnetic Recording (SMR) may be combined with conventional (re-writable) recording on the same drive; in host-managed drives shipping today this capability is used to provide a small number of re-writable zones, typically totaling a few tens of GB. Although these re-writable zones are widely used by SMR-aware applications, the literature to date has ignored them and focused on fully-shingled devices.**

We describe μCache, an SMR translation layer (STL) using re-writable (*mutable*) zones to take advantage of both workload spatial and temporal locality to reduce the garbage collection overhead resulted from out-of-place writes. In μCache the volume LBA space is divided into fixed-sized *buckets* and, on write access, the corresponding bucket is copied (*promoted*) to the re-writable zones, allowing subsequent writes to the same bucket be served in-place resulting in fewer garbage collection cycles.

We evaluate μCache in simulation against real-world traces and show that with appropriate parameters it is able to hold the entire write working set of most workloads in re-writable storage, virtually eliminating garbage collection overhead. We also emulate μCache by replaying its translated traces against actual drive and show that 1) it outperforms its examined counterpart, an E-region based translation approach on average by 2x and up to 5.1x, and 2) it incurs additional latency only for a small fraction of write operations, (up to 10%) when compared with conventional non-shingled disks.

***Index Terms*—shingled magnetic recording, translation layer**

I. Introduction

Shingled Magnetic Recording (SMR) is a modern technology that offers higher storage density compared to conventional magnetic recording (CMR) with the same head and platter technology. It achieves this by overlapping tracks as they are written, for an effective track width narrower than the write head. However, this density improvement comes at a cost: individual disk sectors cannot be over-written, as adjacent downstream tracks will be corrupted [1].

If all tracks on an SMR disk were shingled it would be a purely "write-once" media: once the last track was written, no re-use would be possible without damage to non-overwritten sectors. Instead, the disk is divided into *zones*, separated by "guard tracks" wide enough to prevent adjacent track corruption; each zone may be sequentially written (and rewritten) from the beginning without damage to data in other zones. This approach has been formalized in the SATA and SCSI extensions [2], [3] for zoned block devices, a storage model much like NAND flash: large regions (256 MiB for SMR) must be written sequentially, and the operation to allow a region to be rewritten—"reset zone pointer" for SMR, erase for flash—discards all data in that region.

The SMR restrictions may be addressed in the application or file system [4], using *host-managed* devices which expose SMR write restrictions and provide commands to clear zones for re-write. Alternately, existing file systems may be used over a block translation layer [5], implemented either in the host of a host-managed or in the firmware of a *drive-managed* SMR device. (a third standardized device type, *host-aware* [6], is a hybrid with features of both host-managed and drive-managed.) We focus on block translation layers in this work, exploring algorithms which may be implemented in either the host, via host-managed extensions, or the firmware of drive-managed devices.

Most SMR translation layers (STLs) [1], [7], [8]) are "E-Region-based" [9] translation layers, using a 1:1 mapping from logical block addresses to "data zone" locations, and a small region (the E-region or "on-media cache") to cache exceptions caused by writes, which are written in log-structured fashion to the cache "write frontier". If this cache is shingled the resulting algorithm is similar to the FAST flash translation layer (FTL) [10], with fixed locations for data zones as no wear-leveling is needed. As in the FAST FTL, when the on-media cache fills, the expensive *garbage collection (GC)* operation is performed to make room; space must be reclaimed a zone at a time, merging valid pages from that zone with unmodified pages from the corresponding data zone, and then writing the result back to a data zone.

A fully-shingled host-managed drive presents practical complications. For example, the first sectors of a disk (i.e. LBA 0) are typically used to store partition and file system metadata [11], [12], however, they cannot be updated safely if they are in a shingled zone, but must be "erased" and then re-written. Yet tracks on an SMR drive do not all necessarily need to be shingled, as modern drives are able to vary the track pitch at formatting time [13], and it is possible to format a drive with a combination of shingled tracks and wider re-writable tracks.Thus host-managed drives available today provide a number of re-writable (mutable) zones, starting at LBA zero and totaling 16 to 30 GiB in the drives we have examined.

These data zones are used for hot data storage in both open-source translation layers (i.e. `dm-zoned` [14]) for host-managed SMR and (according to anecdotal statements) in proprietary translation layers in recent drive-managed devices. Yet to date SMR translation layers reported in the literature have either assumed a single fully-shingled disk [1], [5], [7], [15]–[17], or a costly hybrid system comprising e.g. SSD and SMR [8], [18] with no seek overhead for accessing re-writable storage.

In this paper, we introduce and analyze μ*Cache*, a translation layer for SMR disk that relies on both shingled and re-writable zones to operate. Similar to E-region based approaches offered in [5], [16], [17], µCache divides the disk into a large data zone and a small on-media cache to address random writes. However, the on-media cache is formed by re-writable zones. Relying on workload spatial and temporal locality, µCache accommodates the largest possible *write working set* in the on-media cache. This results in improved performance compared to translation layers using only shingled zones as the cached data may be overwritten. To achieve this, on write access, µCache copies a chunk of data (typically larger than the original I/O size) to the on-media cache (if it is already not residing in the cache). Therefore, subsequent write accesses to the same chunk will overwrite this location in cache, rather than consuming new on-media cache space, and thus resulting in fewer multi-second garbage collection cycles.

In summary, the main contributions of this work are as follows:

- We describe µCache, an SMR translation layer using re-writable zones for the on-media cache;
- We show that by exploiting both temporal and spatial locality, µCache is capable of accommodating large write working sets in re-writable cache and therefore minimizes the garbage collection overhead;
- We simulate µCache with real-world traces [19], demonstrating the reduction in garbage collection overhead;
- We emulate µCache by replaying its translated traces against actual drive and show that for many of the workloads it makes an SMR drive nearly as performant as a conventional drive, imposing a modest burden of additional seeks and copies. We further show an average 2x performance improvement of µCache compared to its counterpart E-region based translation layer.

II. Motivation

In both E-region-based translation layers and their alternative (`dm-zoned`), as soon as the cache fills, data is evicted from cache region via expensive garbage collection (GC) cycles. Thus, the performance of these translation layers are strongly affected by the size of the *write working set*—i.e. the number of unique locations written during some prior window —and whether it fits in that cache. If the write working set is larger than the cache, a large fraction of writes will result in evictions, each incurring multi-second penalties due to the need for multiple reads and writes of 256 MiB zones. Conversely, if the working set is significantly smaller than the effective cache capacity, the cache would be utilized more highly and thus few or no GC cycles will be needed.

With a shingled on-media cache, however, the effective cache capacity may be significantly smaller than the space it occupies, with space used by outdated (invalid) data. In Figure 1 we see effective cache utilization for several workloads[1] running on a simulation of an E-region STL from [9] with a 16 GiB shingled on-media cache, equivalent to that of a Seagate ST8000AS022 8TB 5900 RPM host-aware drive.

[1]The three shown traces are representative of the range of behaviors seen in the entire 106-trace corpus.

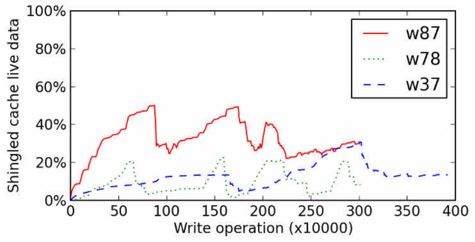


Fig. 1: Fraction of live data in shingled on-media cache vs. time for three representative traces.

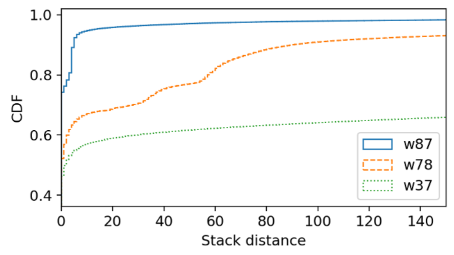


Fig. 2: CDF of 4 MiB-granularity stack distance (number of unique 4 MiB buckets between accesses to the same bucket) [20] for w87, w78 and w37.

As seen, only a small fraction of the cache is used at any time, greatly restricting the write working set which can be held in cache. We see that the high effective cache capacity reaches 50% (for w87), with a mean utilization of 37%, while mean utilization is much lower for the other two: 9% (w78) and 12% (w37). The traces are taken from the CloudPhysics corpus [19] of virtual machine block traces, and are described in more detail later in Section IV.

Workload temporal and spatial localities are two parameters that determine the working set size and therefore caching performance. Temporal locality refers to the tendency of accesses to the same address to cluster in time. In other words, immediately after seeing a reference to address A, the expected time until the next reference to A is lower than the expected time until the next reference to an arbitrarily address. We see this behavior for write operations in Figure 2, showing CDFs of 4 MiB-granularity *stack distance* [20], i.e. the number of unique 4 MiB zones touched between accesses to the same zone, for the same three workloads. As can be seen, among all pair of 4 MiB consecutive write accesses, about 47%, 52% and 78% of the following accesses are to the same 4 MiB region in workloads w37, w78, and w87 showing a high temporal locality for write operations among these traces.

Spatial locality refers to the tendency of references to cluster within the address space; i.e. address $A \pm \epsilon$ is much more likely to be accessed soon after A than some address far from A. Our analysis shows high write spatial locality in most of the traces. For example, for w87 and w78 more than 75% and 33% of the second write accesses are within the range of less than 256 KiB of the preceding write, showing high spatial locality.

These two measures affect different aspects of cache design. Temporal locality determines LRU cache performance, as accesses of stack distance D or less will hit in a cache holding D entries. Spatial locality motivates the use of *cache lines* larger than a single access, on average satisfying requests to more than a single location while needing only one operation to backing storage. For cache line sizes small enough that the fixed cost access dominates, increasing line sizes will (all other things being equal) result in higher performance. Other things are not always equal, however, a larger line sizes result in lower effective cache utilization, with space being used for data that is never accessed.

III. µCache

We describe µCache, an SMR translation layer relying on re-writable zones to more efficiently utilize the on-media cache and thus improve the performance. Taking advantage of on-media cache re-write-ability, µCache exploits both high temporal and spatial locality to perform writes in-place and therefore minimize expensive GC cycles.

A. µCache algorithm

In Figure 3, we see the high-level µCache data layout: the re-writable region is used for checkpoints and on-media cache, while the shingled region holds a temporary zone and the permanent data zones (checkpointing and temporary copies are described more fully in Section III-B). The LBA space of the volume is divided into fixed-sized *buckets*, and on write access a bucket is *promoted* and copied to the on-media cache, reading any necessary data (i.e. that not contained in the triggering write operation) from the data zone. Subsequent writes to that bucket will be performed in-place in the on-media cache until it is evicted by GC. The GC process selects a data zone, reads the data zone and corresponding cached data, merges them, and writes them back.

Details of the µCache translation strategy are described as below. Given a disk with N shingled and M re-writable zones of size Z ($M < N$), and buckets of size b:

- we divide the LBA space into $N \cdot Z/b$ *logical buckets*
- we divide the N shingled zones into the same number of buckets, the *home locations* for each logical bucket

Given a write access to address A, we:

- determine its $\underline{l}$ogical $\underline{b}$ucket $\underline{n}$umber $Lbn = \lfloor \frac{A}{b} \rfloor$
- if Lbn is cached in some $\underline{p}$hysical $\underline{c}$ache $\underline{b}$ucket Pcb, perform the write to offset $A \mod b$ in bucket Pcb; otherwise:
- allocate a physical cache bucket Pcb, *promote* the bucket to cache location Pcb, and again perform the write at the appropriate offset.

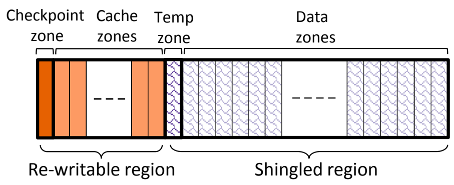


Fig. 3: µCache On-disk data structure: checkpoint zone, re-writable cache zones, temporary zone and data zones.

There are two options for bucket promotion on new writes: *copying* and *mapping*. Copying reads the contents of the bucket from the data zone, merging it with the new write, and writes the entire bucket in cache. Mapping allocates a bucket in cache but does not read from the data zone; a bitmap is instead used to track which portions of the logical bucket are in cache and which in the data zone. Mapping eliminates a seek and two bucket transfers during promotion, but incurs overhead to persist mapping information for every write, as well as read seeks due to fragmentation between cache and data regions. We argue below that copying is the better approach, due to its exploitation of spatial locality and potential to eliminate map persistence overhead, and focus on this option in our evaluation.

Given a read access, we:

- determine its Lbn
- if Lbn is cached in some physical cache bucket Pcb, perform the read to offset $A \mod b$ in bucket Pcb; otherwise:
- perform the read to address A from its home location

A garbage collection cycle is used to make room in cache for new writes:

- select a data zone D and read its contents
- read all buckets from cache holding data from zone D
- merge contents of data zone and data from cache
- save a copy (to prevent data loss in next step)
- write back to the data zone

We note that µCache is in fact a generalization of several existing translation approaches. With minimum-sized buckets, only incoming writes are sent to cache and the two promotion behaviors are equivalent; the behavior is that of an E-region translation layer with re-writable on-media cache, allowing full cache utilization and more efficient GC. At the other end of the range, `dm-zoned` [14] is equivalent to µCache with a bucket size of one zone and bucket promotion by mapping.

Can promotion-by-copy with a modest bucket size out-perform both extremes, avoiding low utilization and wasted cache space due to promoting entire zones, while taking advantage of spatial locality? The answer is yes; with a modest bucket size, almost the entire working set could fit in the on-media cache resulting in high cache utilization (see Table III). Support for this approach maybe seen in Figure 4, showing the write footprint [2]of several workloads when accesses are rounded up to varying bucket sizes.

As seen, footprint increases with bucket size for all work-loads; however the increase is gradual for some, and rapid for others. Moreover, with a reasonable cache size (e.g. 32 GiB) combined with a modest bucket size (e.g. 256 KiB) the entire working set of many of the workloads fits in the cache.

B. Implementation factors

A number of implementation factors that are important to the value of µCache are listed below, but some are not addressed in our evaluation:

[2]Alternately this can be described as the cache size needed to hold the entire workload without GC, when promoting buckets of the specified size.

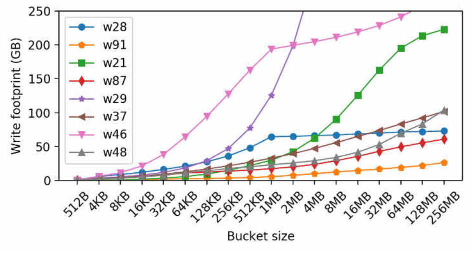


Fig. 4: Write footprint of several workloads vs. bucket size.

Bucket allocation: µCache uses a simple arbitrary bucket allocation policy: it keeps a pool of free physical cache buckets (Pcbs), and assigns them arbitrarily to Lbn on promotion operations.

Memory usage: Memory usage is a significant issue for drive-resident translation layers, and even host-resident ones are limited in the resources they may demand. (e.g. when deployed on specialized storage servers housing as many as 60 or more drives) µCache only needs to map as many buckets as are held in the on-media cache, keeping its memory demands modest. (e.g. with a very small bucket size of 128 KiB, a 16 GiB on-media cache would require a map of 128 K entries, taking less than 10 MiB of memory if implemented with one of several sparse map data structures.)

Map persistence and checkpointing: Translation layers using out-of-place writes must reliably record map updates, as a write is not truly durable until the information needed to locate the new data has been persisted safely in a way which will survive e.g. power failure. In our evaluation we do not implement map persistence; however we note that copy-based promotion eliminates map updates for overwrites, and at bucket promotion they may be logged (as is done in FSTL [5]) in a bucket header, eliminating additional seeks to persist them at a fixed location. Figure 5 shows an example where µCache performs logging when Pcb_{n+2}, Pcb_{n+4}, Pcb_n and Pcb_{n+3} are allocated to Lbn 1 to 4.

As shown, besides the Lbn to which a bucket is allocated, a header contains a sequence number and a CRC to identify whether a particular bucket write completed before a crash. It also comprises a pointer to the next available Pcb header. Pointers are used to reconstruct the map in case of a crash. Depending on where µCache is implemented (either in the firmware or on the host side), a header size will be a sector (512 B) or 4 KiB (to preserve 4 KB alignment) adding a negligible 2.5-5μs or 20-40μs of transfer time to each promotion (bucket write). The resulting space overhead

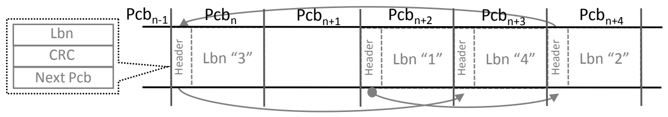


Fig. 5: µCache logging mechanism: buckets may be written in arbitrary order, but are linked through their headers in the order they are written to allow log recovery on failure.

is also negligible: assuming a small bucket size of 256 KiB, the header overhead will be around 0.2% or 0.8% which we consider quite acceptable.

Techniques such as periodic checkpointing may also be used to make the recovery process faster in case of any failure. To do so, the most recent map along with a latest Pcb number is stored at the checkpoint zone. On recovery, first, the most recent map is retrieved. Second, by traversing the header pointers starting at the latest checkpointed Pcb number, map reconstruction is completed.

Compared to copy-based promotion, map-based promotion requires map information to be persisted (at the cost of a seek) for each write to the bucket after promotion. This overhead may be mitigated by deferring writes until the receipt of a write cache FLUSH operation, as is done in `dm-zoned`; however these are very frequent in some workloads.

Garbage collection: The garbage collection process itself in µCache is almost identical to E-region translation layer cleaning (GC) described in Skylight [1] requiring at least 3 full zone transfers: select a data zone to clean, read all data from that zone residing in cache, read the zone itself, merge them and write a backup copy to the temporary zone, and overwrite the original data zone. Selection of the zone to clean is more complex, however, as e.g. maximizing the space freed might in fact evict hot data. In this work we omit discussion of zone selection for GC, and focus on sizing buckets to eliminate the need for GC.

While the cleaning process could affect the throughput significantly, it may not impact the I/O latency necessarily. A garbage collection cycle takes roughly 5 seconds on average, or even more at inner-track LBAs or if reading many extents from cache. The worst-case I/O latency, in turn, depends greatly on how well host I/Os can be interleaved with the operations making up these cycles. We believe that this is primarily an engineering issue, not an attribute of a particular translation layer, and thus in our evaluation we focus on the number of GC cycles incurred and resulting loss in throughput, ignoring the effect on latency.

IV. Trace-driven Evaluation

We evaluate µCache using real-world traces, using simulators implementing the µCache algorithm and the E-region approach, a comparison translation layer. We measure µCache performance by replaying its translated trace (generated by µCache simulator) against a physical drive and compare it with that of the E-region approach as well as a conventional drive; each experiment is run at least five times. This workflow is seen in Figure 6.

Workloads: µCache was evaluated using the CloudPhysics block trace corpus [19], 106 large block traces from a virtualized environment running Windows and Linux with modern file systems. As shown by Hajkazemi et al. [21] these traces are more representative of modern workloads than older traces such as the well-known MSR corpus [22]. A subset of the 106 traces were selected, choosing ones which (1) were long enough to trigger garbage collection, and (2) represented

TABLE I: Statistical summary of selected workloads.

workload	w09	w10	w14	w21	w26	w28	w29	w37	w46	w48	w69	w75	w78	w87	w91
I/O count (M)	49.62	48.34	34.81	29.41	26.53	19.73	19.08	18.84	11.54	14.09	7.87	6.15	5.49	3.78	4.31
write ratio	0.55	0.39	0.68	0.1	0.58	0.33	0.14	0.21	0.62	0.42	0.3	0.79	0.55	0.8	0.27
write mean size (KB)	33.4	26.7	32	21.9	19.1	45.2	14.6	7.1	212	6	17.4	28.5	20.4	18.6	15.4
read mean size (KB)	154	153.7	203.1	99.7	26.6	15.6	10.1	6.1	121.1	153.8	23	20.6	30.3	56.4	17.7
peak IOPS (every 10 sec)	941	2002	681	1321	1063	4489	2327	1414	2560	4520	760	521	1841	3378	3223

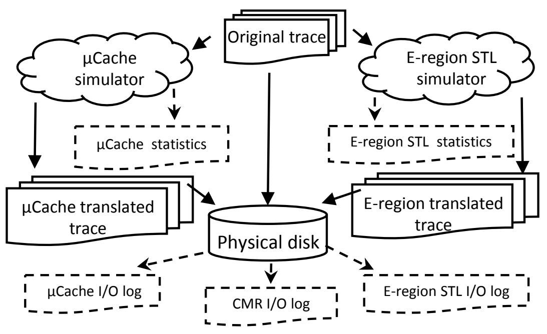


Fig. 6: Methodology overview.

different levels of read/write intensity. The selected workloads vary in size from about 4 to 50 million I/Os, and range from read-heavy (w21, 10% writes) to very write-heavy (w87, 80% writes); details are shown in Table I.

Trace-driven experiment: Trace-driven simulation and emulation of both µCache and the E-region STL [9] were used to measure behavioral statistics as well as performance such as I/O amplification, on-media cache hit ratio, additional seeks incurred, garbage collection cycles, promotion operation latency and throughput.

As shown in Figure 6, µCache was emulated by replaying its translated traces (generated by µCache simulator) against the actual device to measure latency and throughput. These traces combined remapped host I/Os, bucket promotion reads and writes for µCache, and garbage collection operations for both µCache and E-region STL. Promotion and GC I/Os were split into 512 KiB operations, a common limit for SCSI reads and writes. Trace replay was performed by `fio`, an I/O testing tool [23], using `libaio` I/O engine with `iodepth` of 31. A summary of experimental parameters may be seen in Table II.

A. *µCache performance*

As described in section III, the µCache goal is to hold the largest possible write working set in the on-media cache to maximize the number of in-place writes, and thus reduce garbage collection cycles. To measure its success, we report 1) the µCache throughput with bucket size of 256 KiB compared to that of both E-region STL and a conventional drive (CMR) in Figure 7 and, 2) the cache hit ratio for write accesses with different bucket sizes ranging from 0.125 MiB to 4 MiB in Table III.

TABLE II: Experimental parameters and drive specification.

µCache and baseline E-region STL simulation parameters	
on-media cache size &zone size	16 GiB & 256 MiB
max and min I/O size	512 KiB, 4 KiB
µCache & E-region eviction policy	LRU & FIFO
Physical drive specification	
drive model	ST4000DM000-1F2168
drive capacity & rpm	4 TB & 5900
Wcache & read-ahead & look-ahead	off & on & on

As seen in Figure 7, µCache performance is on average twice as high as that of the E-region STL, and in the best case (w75) µCache outperforms E-region STL by 5x. For two of the workloads (i.e. w69 and w48) µCache performance is marginally lower, however, we note that by selecting a slightly larger bucket size, (e.g, 512 KiB for w69) µCache is able to beat the E-region STL in these traces.

As seen, CMR performance is higher than that of µCache for all workloads, however, this performance gap is marginal for several workloads (e.g., w91, w87) indicating the low overhead and high performance of µCache. Note that the large performance gap between both translation approaches and the CMR for w46 is due to the large footprint of this trace (see Figure 4). This large footprint results in a working set larger than the cache size, and therefore leads to more cache write misses (see Table III), more GC cycles (see Table IV), and consequently lower performance.

As seen in Table III, all but two traces (w10 and w46) show hit ratios over 85% at all bucket sizes, and nine out of ten traces reach a hit ratio of 99% with the proper bucket size. Even the cache-unfriendly w46 trace (54% hit ratio at 128 KiB) improves to 93% when the bucket size is increased to 4 MiB.

We see three different patterns in hit ratio vs bucket size:

1) for the majority of traces (w21, w26, w28, w46, w48, w69, w87, w91) the hit ratio increases monotonically as bucket size is increased;
2) for three of the traces, w09, w10, and w14 the hit ratio first goes up and then starts decreasing at a certain bucket size (2 MiB, 0.5 MiB, and 0.5 MiB for w09, w10 and w14 respectively);
3) for a single trace, w37, the hit ratio decreases with increasing bucket size.

We note that by increasing the bucket size we exploit the spatial locality of the traces more effectively resulting in a higher on-media cache hit ratio for the traces in case 1. However, if a workload has poor spatial locality, this increase in bucket size reduces the effective cache capacity (by caching unneeded data), in turn resulting in more capacity misses and an overall lower on-media cache hit ratio in cases 2 and 3.

B. *µCache garbage collection*

Garbage collection (GC) is the primary factor reducing throughput for shingled drives [7], as it is a multi-second

TABLE III: Write access hit ratio in µCache with different bucket sizes captured in simulations.

	workload														
bucket size	w09	w10	w14	w21	w26	w28	w29	w37	w46	w48	w69	w75	w78	w87	w91
128 KiB	0.88	0.74	0.86	0.95	0.98	0.91	0.86	0.96	0.54	0.97	0.98	0.99	0.95	0.96	0.98
256 KiB	0.91	0.76	0.9	0.97	0.99	0.95	0.89	0.96	0.65	0.99	0.99	0.99	0.98	0.98	0.99
512 KiB	0.92	0.77	0.91	0.98	0.99	0.97	0.91	0.95	0.76	0.99	0.99	1	0.99	0.99	0.99
1 MiB	0.92	0.75	0.91	0.99	0.99	0.98	0.92	0.94	0.84	0.99	0.99	1	0.99	0.99	1
2 MiB	0.90	0.73	0.90	0.98	1	0.99	0.92	0.92	0.90	1	1	1	0.99	1	1
4 MiB	0.88	0.70	0.87	0.98	1	1	0.91	0.92	0.93	1	1	1	0.99	1	1

TABLE IV: GC cycles for µCache (bucket sizes 128 KiB to 4 MiB) and E-region shingled cache captured in simulations.

	workload														
bucket size	w09	w10	w14	w21	w26	w28	w29	w37	w46	w48	w69	w75	w78	w87	w91
128 KiB	30462	29753	30293	10	337	573	5238	46	16828	0	0	0	0	0	0
256 KiB	43942	51798	43531	96	377	570	8585	323	17592	41	0	0	0	0	0
512 KiB	76670	96607	71034	463	441	567	15517	1259	18813	102	0	0	0	1	0
1 MiB	144459	189933	137555	954	625	577	24335	3693	21059	261	0	0	57	53	0
2 MiB	328535	377267	300922	2412	843	592	37921	9206	25674	482	0	0	189	110	0
4 MiB	763952	724891	664792	6670	1483	715	59216	18466	32687	1035	0	0	401	224	0
E-region STL	94597	30183	78849	2310	3459	1917	6967	637	29596	724	208	760	630	538	44

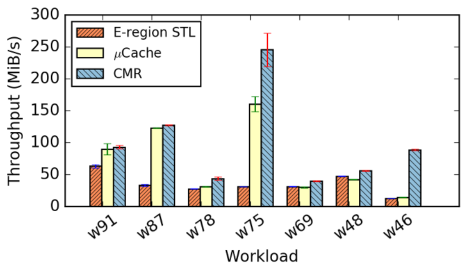


Fig. 7: Throughput of µCache (with bucket size of 256 KiB) compared to E-region STL and CMR captured in emulations.

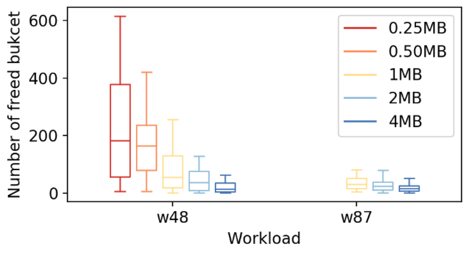


Fig. 8: Buckets gathered per GC cycle vs. bucket size. (no GC cycles observed for 0.25 MiB and 0.5 MiB in w87)

operation (roughly 5 seconds at 160 MB/s). Thus we use the number of GC cycles as a proxy to show the impact on throughput.

In table IV we report the number of GC cycles for trace execution with two translation layers: µCache with bucket sizes from 0.25 MiB to 4 MiB, and the E-region STL [9]. These experiments use a 16 GiB (64-zone) cache, equal to size of re-writable zones in the Seagate ST8000AS022 8TB 5900 RPM host-aware drive[3].

In a few cases, no GC cycles were seen for all or many bucket sizes (e.g., w91, w75 and w69); in all other cases smaller bucket sizes resulted in fewer GC cycles. In 10 and 11 out of 15 cases, µCache with 1 MiB and 256 KiB bucket sizes outperformed E-region cache. Based on the trend we observe for w09, w10, and w14 where baseline E-region STL outperforms µCache, we expect decreasing the bucket size will result in fewer GC cycles in µCache over baseline E-region STL.

To examine the impact of cache size on performance, we repeated the simulation with 32 GiB of cache size for both E-region and µCache (with bucket size of 256 KiB). Our experiments show an average reduction of 2x in number of GC cycles in both approaches if not eliminated completely. w91 is the only new case with zero GC cycles for E-region STL, whereas w37, w28, and w21 are the ones for µCache.

The actual duration of a GC cycle includes time to read data from the cache, which can be high if gathering large numbers of small fragments. To quantify this possible increase in GC cycle, we measure the number of buckets retrieved from cache in each cycle; results for two representative workloads are seen in Figure 8. In the worst case (w48, 256 KiB bucket size) the mean number of buckets evicted per GC cycle was roughly 200; assuming 5 ms each this would add an additional 1 s to the average GC cycle. We note that the use of buckets bounds this overhead, as well, as the maximum number of 256 KiB fragments in a 256 MiB zone is 1024; under worst-case workloads, simple E-region STL may gather far greater numbers of independent extents in a single GC cycle.

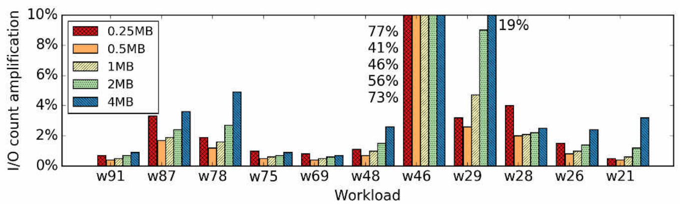
C. µCache bucket promotion

Copying an entire bucket for promotion results in an additional seek and bucket transfer, amplifying both I/O operations and bytes transferred[4]. This may be seen in Figures 9a and 9b, which show I/O amplification in operations and bytes, respectively.

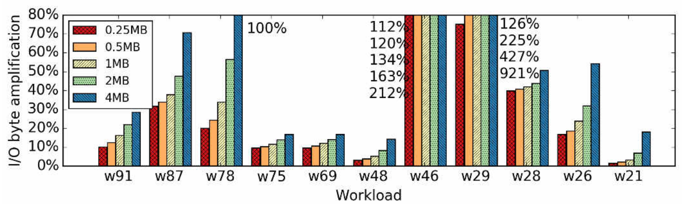
We note that although the I/O amplification in bytes is considerable, reaching a factor of 2 and 9 in two (w46 and

[3]More recent drives have been observed to have slightly larger conventional regions, e.g. 31 GiB in devices available to the authors, and 1% of total capacity in other drives.

[4]Unlike flash, disk has symmetric read and write performance, thus performance impact is better quantified by I/O amplification than write amplification.



(a) I/O operations



(b) bytes

Fig. 9: Background promotion I/O amplification in: a) I/O operations and, b) bytes, captured in simulations.

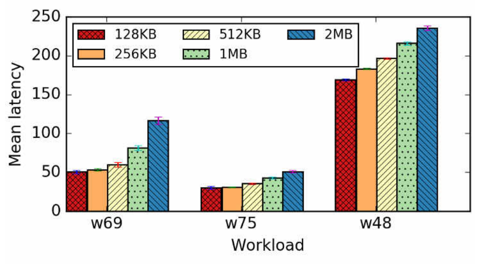


Fig. 10: Promotion mean latency captured in emulations.

w29, 4 MiB bucket size), the total increase in I/O count is modest in almost all cases. The average byte amplification of all cases with the smallest bucket size is 30%. For capacity drives the overhead of a single seek is roughly the same as for a 2 MB transfer, so the overall impact on performance should be modest, especially for bucket sizes of 1 MiB or less. Although I/O volume amplification increases strictly with bucket size, I/O operation amplification varies by trace; in large part due to increased GC (and thus promotion) with larger bucket sizes.

In addition to transfer time, promotion-by-copy results in an additional disk seek to read the associated bucket from data zones for each promotion. Promotion-by-mapping incurs no promotion seeks, but fragments the bucket between the data zone and the cache, causing extra read seeks. Table V compares the read seek overhead of these two approaches, for bucket sizes of 0.5 MiB and 4 MiB. When promotion-by-copy is used, for a majority of the traces the read seek overhead is seen to be lower than that of promotion-by-mapping for both examined bucket sizes.

In addition to the read seek incurred to promote a bucket, promotion-by-copy introduces fragmentation at bucket boundaries as well; however the number of additional seeks incurred varies depending on the bucket size. Among the traces, w87 and w21 shows the highest and the lowest fragmentation at boundaries (20% and 1% respectively) with the smallest examined bucket size (0.125 MiB) . However, increasing the bucket size to 4 MiB reduces the fragmentation ratios to 0.8% and 0.1% for the two workloads.

Bucket promotion is a synchronous operation, i.e. the write operation causing the promotion is not completed till the end of promotion process; this latency varies with bucket size. Figure 10 shows mean latency measured by replaying the µCache translated trace on actual device) incurred by bucket promotion for varying bucket sizes for three representative traces. Not surprisingly, interruption time goes up with bucket size; however the interruptions are modest and of bounded duration; mean interruption time ranges from 30 ms (w75 with bucket size of 128 KiB) to 240 ms (w48, bucket size of 2 MiB). However, these interruptions occur only for cache misses, which as we saw in Table III represent between 1 and 10% of writes in almost all cases. This overhead will be small in comparison to the latency already incurred by writes; Our trace analysis results show that depending on workload between 25% and 70% of writes involve seeks of greater than 256 MiB, incurring significant seek and rotational delays.

V. Related work

Host-managed SMR drives have shipped with both shingled and re-writable regions for several product generations [6], with a fixed number of re-writable zones starting at LBA zero. Recently announced plans extend this by allowing the size of these regions to be adjusted dynamically [24]; however no guidance is given for use of these regions. ZoneAlloy is among the very first research that proposes approaches to manage such a technology [25].

Re-writable regions for caching are proposed in recent work on track translation layers for interlaced magnetic recording (IMR) [26], where writes to “bottom” tracks may damage adjacent “top” tracks, while top tracks may be modified without such risk. In that work, however, the re-writable zone is used very differently than in our work: only writes to selected hot bottom tracks are forwarded to the cache. Furthermore, in µCache read-modify-write (RMW) operations are performed infrequently, at a zone granularity, whereas in the IMR work [26] RMW operations are performed at a track granularity and more frequently.

In publicly-disclosed SMR algorithms to date, only `dm-zoned` [14] makes use of re-writable zones; other work either assumes a fully-shingled drive [5], [15]–[17], [27], [28] or a costly hybrid system including SSD [8], [18]. Although `dm-zoned` represents a step in the direction of this work, we demonstrate the clear advantage offered by (a) smaller bucket sizes, allowing more effective use of the re-writable cache, and (b) promotion by copying, reducing fragmentation and additional read seeks incurred.

Cassuto et. al propose a translation layer using a set-associative cache as well as an additional circular buffer cache. FSTL [5] introduces a framework to design new translation

TABLE V: Additional seeks due to promotion (promotion-by-copy) and fragmentation (promotion-by-map).

		workload														
bkt size	seek ovrhd	w09	w10	w14	w21	w26	w28	w29	w37	w46	w48	w69	w75	w78	w87	w91
0.5MB	copy ($\times 10^6$)	2.02	4.13	2.02	0.05	0.11	0.2	0.22	0.17	2.24	0.05	0.02	0.02	0.03	0.03	0.01
	map ($\times 10^6$)	1.89	6.32	1.2	0.71	0.72	0.15	4.05	0.24	0.06	10.41	0.98	0	0.06	0.02	0
4MB	copy ($\times 10^6$)	3.02	5.15	2.7	0.06	0.04	0.03	0.21	0.31	0.5	0.02	0	0	0.02	0.01	0
	map ($\times 10^6$)	0.3	0.83	0.18	0.26	0.43	0.12	3.87	0.14	0.03	4.88	0.9	0	0.04	0.02	0

layers, and use it to explores different garbage collection algorithms for SMR translation layer. Both VGuard [16] and SMaRT [17] propose track-based STL solutions. Tancheff et. al [28] introduce ZDM, a fully page-mapped translation layer implemented as a host-side device mapper for SMR disks, similar to DFTL [29] for NAND flash. In each case µCache differs by taking advantage of conventional regions on the disk, allowing many (and often nearly all) writes to be handled without the overhead of out-of-place writes. Furthermore, with the exception of the track-based SMaRT [17] none of these cases exploit the spatial and temporal locality of workloads, while SMaRT differs in that its caching granularity is fixed to the (location-varying) track size.

Both FC [8] and SMRC [18] use NAND flash for the on-media cache, and thus may outperform µCache, at the cost of adding higher-priced storage to the device or system. µCache relies only on a single magnetic recording device, and thus is simpler and lower-cost.

VI. Conclusion

Although a combination of shingled and conventional magnetic recording has been already implemented in host-managed SMR drives, its characteristics when used by a translation layer have not been addressed in the literature yet.

In this work, we introduce µCache, a translation layer that takes advantage of both shingled and re-writable zones on the same device, exploiting both spatial and temporal locality of workloads to reduce the overhead of out-of-place writes.

Simulating µCache against real-world traces, we find that with appropriate bucket sizes the entire write working set of many of workloads can fit in the re-writable cache, resulting in the total elimination of garbage collection overhead. We further, emulate µCache and evaluate its performance by replaying translated traces against actual device and show that it outperforms its counterpart E-region translation layer on average by 2x and up to 5.1x.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1910327 and by a NetApp Faculty Fellowship.

References

[1] A. Aghayev, M. Shafaei, and P. Desnoyers, "Skylight—a window on shingled disk operation," *ACM Transactions on Storage (TOS)*, vol. 11, no. 4, p. 16, 2015.

[2] I. T. T. Committee, "Information technology - Zoned Block Commands (ZBC)," ANSI, Inc., Draft Standard T10/BSR INCITS 536, Sep. 2014.

[3] ——, "Information technology - Zoned-device ATA Command Set (ZAC)," ANSI, Inc., Working Draft American National Standard T13/BSR INCITS 537 Revision 04b, Sep. 2015.

[4] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," in *13th SOSP*, 1991, pp. 1–15.

[5] M. H. Hajkazemi, M. Abdi, M. Shafaei, and P. Desnoyers, "FSTL: A framework to design and explore shingled magnetic recording translation layers," in *MASCOTS '18*, Oct. 2018.

[6] T. Feldman and G. Gibson, "Shingled magnetic recording: Areal density increase requires new data management," *USENIX; login: Magazine*, vol. 38, no. 3, pp. 22–30, 2013.

[7] M. Shafaei, M. H. Hajkazemi, P. Desnoyers, and A. Aghayev, "Modeling drive-managed smr performance," *ACM Transactions on Storage (TOS)*, vol. 13, no. 4, p. 38, 2017.

[8] C. Ma, Z. Shen, L. Han, R. Chen, and Z. Shao, "FC: Built-in flash cache with fast cleaning for SMR storage systems," *Journal of Systems Architecture*, vol. 98, pp. 214–220, Sep. 2019.

[9] D. R. Hall, "Shingle-written magnetic recording (SMR) device with hybrid e-region," Apr. 1 2014, uS Patent 8,687,303.

[10] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, Jul. 2007. [Online]. Available: http://doi.acm.org/10.1145/1275986.1275990

[11] B. J. Nikkel, "Forensic analysis of GPT disks and GUID partition tables," *Digital Investigation*, vol. 6, no. 1-2, pp. 39–47, Sep. 2009.

[12] P. Technologies, *System BIOS for IBM PC/XT/AT computers and compatibles: The complete guide to ROM-based system software*, 2nd ed. Reading, Mass: Addison-Wesley Pub. Co, 1989.

[13] E. Krevat, J. Tucek, and G. R. Ganger, "Disks are like snowflakes: No two are alike," in *Proceedings of the 13th HotOS*, 2011, pp. 14–14.

[14] "dm-zoned," https://www.kernel.org/doc/Documentation/device-mapper/dm-zoned.txt.

[15] Y. Cassuto, M. A. A. Sanvido, C. Guyot, D. R. Hall, and Z. Z. Bandic, "Indirection systems for shingled-recording disk drives," in *Proceedings of the 26th MSST*, Washington, DC, USA, 2010, pp. 1–14.

[16] M. Shafaei and P. Desnoyers, "Virtual guard: A track-based translation layer for shingled disks," in *HotStorage 17*, Santa Clara, CA, 2017.

[17] W. He and D. H. Du, "SMaRT: An approach to shingled magnetic recording translation." in *15th FAST*, 2017, pp. 121–134.

[18] X. Xie, L. Xiao, X. Ge, and Q. Li, "SMRC: An endurable ssd cache for host-aware shingled magnetic recording drives," *IEEE Access*, vol. 6, pp. 20 916–20 928, 2018.

[19] C. A. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient mrc construction with shards," in *13th FAST*, 2015, pp. 95–110.

[20] R. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[21] M. H. Hajkazemi, M. Abdi, and P. Desnoyers, "Minimizing read seeks for smr disk," in *IISWC*. IEEE, 2018, pp. 146–155.

[22] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: practical power management for enterprise storage," in *Proceedings of the 6th FAST*, San Jose, California, 2008, pp. 1–15.

[23] J. Axboe, "fio," https://github.com/axboe/fio.

[24] T. Feldman, "Flex dynamic recording," *; login:*, vol. 43, no. 1, 2018.

[25] F. Wu, B. Li, zhichao Cao, B. Zhang, M.-H. Yang, H. Wen, and D. H. Du, "Zonealloy: Elastic data and space management for hybrid SMR drives," in *HotStorage 19*, Renton, WA, Jul. 2019.

[26] M. H. Hajkazemi, A. N. Kulkarni, P. Desnoyers, and T. R. Feldman, "Track-based translation layers for interlaced magnetic recording," in *USENIX ATC 19*, Renton, WA, Jul. 2019, pp. 821–832.

[27] W. He and D. H. Du, "Novel address mappings for shingled write disks," in *HotStorage 14*, 2014.

[28] S. Tancheff, "Seagate zdm device mapper," https://github.com/Seagate/ZDM-Device-Mapper.

[29] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," *Acm SIGPLAN Not.*, vol. 44, no. 3, pp. 229–240, 2009.