ZEA, A Data Management Approach for SMR

Adam Manzanares[1], Noah Watkins[2], Cyril Guyot[1], Damien LeMoal[1], Carlos Maltzahn[2], and Zvonimir Bandic[1]

[1]Western Digital Research
[2]University of California, Santa Cruz

1 Introduction

Digital data is projected to double every two years creating the need for cost effective and performant storage media [4]. Hard disk drives (HDDs) are a cost effective storage media that sit between speedy yet costly flash-based storage, and cheap but slower media such as tape drives. However, virtually all HDDs today use a technology called *perpendicular magnetic recording*, and the density achieved with this technology is reaching scalability limits due to physical properties of the technology [17]. While new technologies such as shingled magnetic recording (SMR) that further increase areal density are slated to enter the market [6], existing systems software is not prepared to fully utilize these devices because of the unique I/O constraints that they introduce.

SMR requires systems software to conform to the shingling constraint. The shingling constraint is an I/O ordering constraint imposed at the device level, and requires that writes be sequential and contiguous within a subset of the disk, called a *zone*. Thus, software that requires random block updates must use a scheme to serialize writes to the drive. This scheme can be handled internally in a drive or an alternative approach is to expose the zone abstraction and shingling constraint to the host operating system. Host level solutions are challenging because the shingling constraint is not compatible with software that assumes a random-write block device model, which has been in use for decades. The shingling constraint influences all layers of the I/O stack, and each layer must be made SMR compliant.

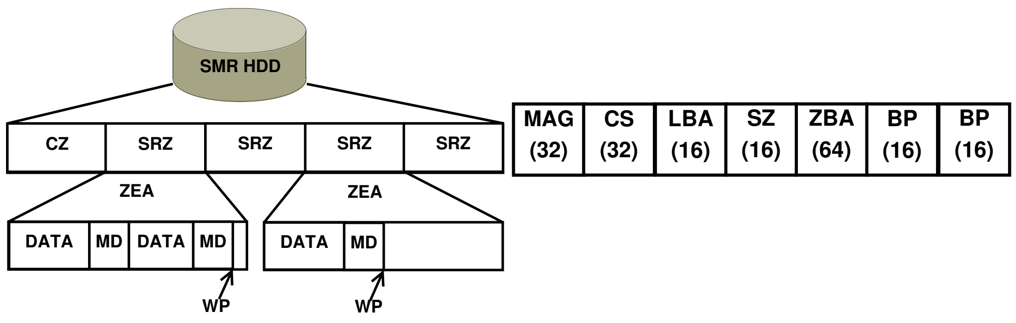
In order to manage the shingling write constraint of SMR HDDs, we have designed a zone-based extent allocator that maps ZEA logical blocks (ZBA) to LBAs of the HDD. Figure 1a depicts how ZEA is mapped onto a SMR HDD comprised of multiple types of zones, which are described in Table 1. ZEA writes logical extents, comprised of data and metadata, sequentially onto the SMR zone maintaining the shingling constraint.

2 Related Work

Previous work has shown that drive-level management of the shingling constraint is a viable approach for SMR HDDs [3, 11]. The main drawback of drive level solutions is that their implementations can lead to unpredictable performance under certain workloads [1]. Some studies look at shingled disk data management from the perspective of the log-structured file system [16, 2]. We were able to adapt the NILFS2 log-structured file system to a shingled disk by mapping file system metadata to a conventional zone. This was a functional approach, but existing problems within NILFS2 reduced space utilization and performance was not acceptable.

HiSMRfs, TableFS, and SFS are file systems that have decoupled metadata and data policies [8, 15, 13]. SMRDB proposes that a KV interface is well suited for SMR devices [14]. WiscKey is another storage system that provides a KV interface and has properties that may work well with an SMR device [12]. Our goal is not to provide a KV or file system interface to the SMR drive, but rather to provide a building block for higher level abstractions in order to provide flexibility. Work related to data placement policies within SMR devices is complementary to our work [9].

The data management challenges on SSDs that require a translation layer is very similar to the challenges faced by drive-managed(DM) SMR devices, although many performance assumptions are different [5]. Our interface has a similar API to Nameless Writes [19], but we intentionally leave garbage collection to the consumer of our API in order to provide predictable performance. In addition we do not tie a logical extent to metadata at ingest, or provide segments in order to maintain a light-weight implementation. Caveat-Scriptor [10] proposes that SMR HDDs do not enforce any write restrictions and instead a host manages all aspects of writing. This is promising work, but there are currently no SMR HDDs that support this model.



(a) ZEA SMR Management (b) ZEA Metadata

Figure 1: **ZEA Architecture** *Figure 1a illustrates how a ZEA instance is mapped to one write pointer based SMR zone. Figure 1b represents the per-write metadata that ZEA places on a SMR zone.*

3 SMR Drive Models

The shingling constraint of an SMR device can be managed either at the drive-level, or at the host-level in software. While DM devices are a drop-in replacement for existing HDDs, they suffer from two important limitations. First, the resources required to provide the virtual block device interface introduces cost and complexity in the drive design. Additionally, DM devices can suffer from unpredictable performance due to the read-modify-write cycles used to reclaim space within zones [1]. In contrast, host-managed(HM) drives fully expose the zone abstraction allowing for application-specific management of the shingling constraint.

The *T10 Zoned Block Command* (ZBC) standard [18] introduces terminology and behavior that is compatible with SMR HDDs, but is not specific to SMR. The T10 standard defines three types of zones that can exist within a block device, shown in Table 1. These zone types are *conventional*, *sequential write preferred*, and *sequential write required*.

First, a conventional zone (CZ), an area of the logical block address (LBA) space that is semantically equal to a SCSI Block Command (SBC) HDD, has no restrictions on I/O ordering. A sequential write preferred zone (SPZ) places no restriction on I/O ordering. However, unlike a CZ, the SPZ introduces the concept of a *write pointer*, which is equal to the largest LBA written to within the zone. Applications should write at the write pointer within a sequential write preferred zone, but the drive is capable of dealing with random writes if they arise. A sequential write required zone (SRZ) also exposes a write pointer, but the device will reject writes to any location other than at the write pointer.

A SMR device is HM when it is shingled and contains sequential write required zones. For completeness, we also define a host-aware device to be one containing sequential write preferred zones. Since host-aware SMR HDDs and DM SMR HDDs are preferably written sequentially and contiguously within a zone, systems software designed for HM devices will be compatible with all SMR HDDs. Therefore we have chosen to focus on systems software for HM SMR drives.

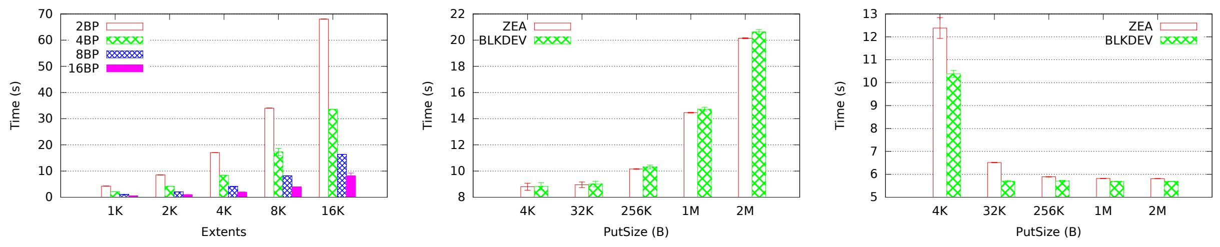
4 ZEA Design & Performance

ZEA focuses on minimizing the metadata overhead of indirected writes that are an artifact of obeying the shingling constraint, while balancing the read/write performance and start-up times of ZEA. ZEA will only return successfully from a put operation if the data and metadata for the extent have been written providing a consistent transaction. In the proposed architecture each zone has a separate instance of ZEA, because each zone has a separate write pointer.

The Put method of ZEA has an argument of a logical extent, which consists of a data buffer and a ZBA, and returns the drive LBA of where the logical extent was placed. During a Put operation, metadata mapping the logical extent to a LBA is placed after the data that is

Type	Write Restriction	Intended Use	Con	Abbreviation
Conventional	None	In-place updates	Increased Resources	*CZ*
Sequential Preferred	None	Mostly sequential writes	Variable Performance	*SPZ*
Sequential Required	Sequential Write	Only sequential writes	Lack of Systems Software	*SRZ*

Table 1: ZBC Zone Types



(a) Start Up Time (b) Write Performance (c) Read Performance

Figure 2: **ZEA vs. Block Device Performance**

contained within the extent. ZEA will append the logical extent at the write pointer of the zone as long as there is space available for the logical extent and its associated metadata within the zone. Within the extent metadata generated by ZEA are back pointers to the metadata of previously written extents. In addition, a checksum of the data contained within the extent and a marker indicating metadata is stored in the extent metadata. The metadata placed by ZEA enables a consumer to discover the logical extents that have been written to a zone on-demand. The Get operation requires that the user supply an argument of an extent that uses a LBA addressing and not a ZBA. The get operation retrieves an entire logical extent if the supplied LBA address and length matches the start LBA and length of a logical extent placed by ZEA. The consumer of ZEA is responsible for preventing/handling ZBA writes to multiple zones and deciding when/how to garbage collect zones.

In addition to the Put/Get methods of ZEA we have implemented metadata iterators over the logical to physical metadata mappings written by ZEA. These iterators can be used at start up to discover all logical extents that have been written and the sequence of how the logical extents have been written. This functionality gives the consumer of ZEA the ability to discover all logical extent writes and their order on demand and this information is used for garbage collection policies.

Figure 2 illustrates the time to read the metadata written by ZEA within a zone and the read/write performance of ZEA when a block size of 4KiB is used. All experiments are an average of 10 runs and include error bars. Figure 2a plots the start up time of ZEA versus the number of extents allocated within the zone for varying numbers of back pointers that ZEA is using. Recall that ZEA leverages pointers to previous metadata entries and that this is a tunable parameter. As expected this time grows linearly with the number of extents that are allocated.

Figure 2b shows the write performance of ZEA vs. writing to a block device. ZEA and the block device are two separate HDDs that are the same model, with the ZEA experiments conducted with SMR compliant firmware. In this experiment we write 1024 extents of sizes varying from 4KiB to 2MiB and compare this to writing the same amount of data directly to a block device. ZEA issues SCSI read and write commands through libzbc [7], a library that implements ZBC compatible SCSI commands, while the block device is using direct I/O. Figure 2b indicates that ZEA has no overhead on write performance as compared to writing to a block device. The extra metadata write is masked by the time taken for the write head to settle over the position of the extent data to be written. When the time to write the extent data is large enough to mask the time taken for the head to settle over the position to write extent data, the amount of metadata written is small, relative to the extent data, to make an impact on performance. In the experiment used to generate Figure 2c we issue sequential reads through ZEA and to the block device using the same parameters used to generate Figure 2b, except we fix the total amount of data read to 1GiB. ZEA is slower than the block device for all put sizes, and this difference is exacerbated by small put sizes. We attribute this difference in performance due to the fact that ZEA is issuing reads through the SCSI layer, which does not perform read-ahead, and the block device is employing read-ahead.

5 LevelDB & ZEA

To demonstrate the usability of ZEA we created a simplified file system, the ZEA file system, *ZEAFS*, which was designed to be compatible with the LevelDB backend interface. ZEAFS has two major goals, mapping extents to files, and maintaining a directory name space. LevelDB databases use a single directory to hold files, and we leverage this flat name space in our file system design. We have chosen to store directory and file allocation information in a CZ located at the start of the SMR drive used for our experiments. This decoupling of metadata used to mange the shingling constraint from metadata used to map ZEA extents onto storage abstractions provides ZEA based architectures flexibility.

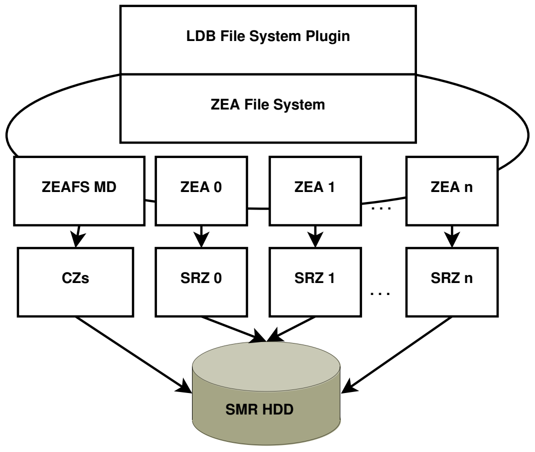


Figure 3: ZEA + LevelDB Architecture

Figure 3 illustrates the modified architecture of LevelDB that is used when an SMR drive is the target media.

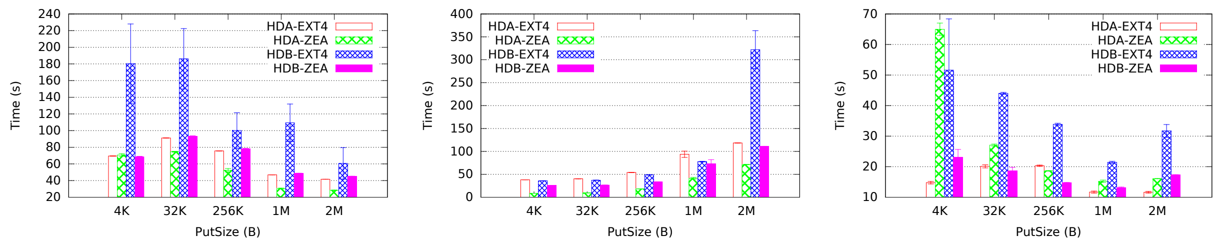
To elucidate the performance of ZEAFS we ran several benchmarks comparing the performance of LevelDB when ZEAFS and ext4 are used for storage. We have chosen to compare against ext4 because it is mature and widely used. The test system that we have used contains a six core 1.9GHZ Xeon CPU, contains 48GiB of memory and we run our experiments on two HDDs. The first HDD is a 6TB VendorA HDD, denoted as *HDA* in experiments. The second drive is an 8TB VendorB DM SMR HDD, denoted as *HDB* in our experiments. In order to run ZEA on HDA we have flashed the firmware of HDA to be a zbc compliant HM HDD, formatting the drive with 1% of the drive as a CZ and the rest of the drive as SRZs of 256MiB in size. To get ZEA to run on HDB we use the emulation mode of libzbc and configured the CZ and SRZ with the same parameters used for the HDA with modified firmware.

In the first experiment a total of 1GiB of data was written and we varied the size of the value of the KV pair inserted into the database from 4KiB to 2MiB. The KV pairs are inserted into the database in integer order (not lexicographically sequential) without the LevelDB sync flag for the first experiment and sync enabled on the second experiment. Data is periodically synced to the disk due to flushing of sstables and compactions when the sync flag is not used. The ext4 configuration of LevelDB leverages the Linux Page Cache and we have added a 5MiB file write buffer to ZEAFS files to provide a fair comparison between ZEAFS and ext4 versions of LevelDB. ZEAFS will flush the file buffer to the disk whenever a sync operation is performed on a file.

The most important trend from Figure 4a is the fact that ZEAFS is able to reduce the variance of performance for the HDB HDD. Ext4 is not strictly sequential and spreads data and metadata across the LBA space of the HDD. Even though LevelDB is writing sequentially to each individual file it uses, the access pattern seen by the HDD is not SMR friendly. DM SMR HDDs require indirection schemes to manage the shingling-constraint and their implementations are fixed and will work well with a limited set of workloads. ZEAFS, being built on top of ZEA, serializes access to zones of the HDD providing purely sequential access within well defined LBA regions, which is an access pattern that matches the physical properties of SMR HDDs. The second trend that we can glean from Figure 4a is the fact that ZEAFS matches or exceeds the performance of ext4 when LevelDB is used.

Figure 4b varies the size of KV pairs inserted into LevelDB, but now issues a sync operation after every KV pair insert. In these experiments we limited the number of put operations to 1024, to highlight the fact that small KV pairs inserts that are synced are much slower as compared to the previous experiment. A key take away from these graphs is that ZEAFS is faster than ext4 for all drives. If we look at the HDA drive we can conclude that ZEAFS has fewer disk operations than ext4 per sync operation. ext4 uses a journal to keep metadata consistency in the face of crashes, which comes at a performance penalty for small file sync operations. ZEAFS writes all metadata updates synchronously, which can limit its performance under batch operations but is superior for small sync operations. Figure 4b shows that HDB HDD behaves much more predictably with synced KV puts until a size of a 2MiB KV pair is used, but once again ZEAFS eliminates the performance variance.

Figure 4c plots the read performance of LevelDB on top of ext4 and ZEAFS when reading 1GiB of KV pairs in integer order. The first trend highlighted in Figure 4c is the fact that ZEAFS has lower read performance as compared to ext4 when the HDA HDD is used (with an exception for 256K KV pairs). This performance discrepancy is most prominent in the case of 4K KV pairs. ZEAFS on HDA has no read-ahead mechanism because I/Os are issued directly through the SCSI layer, limiting its read performance for small requests. The trend seen with the HDA HDD is reversed when we look at the results from the HDB HDD. Read performance is lower with ext4 as compared to ZEAFS. There are two key factors that give ZEAFS an advantage on HDB as compared to HDA. The first is that we emulate a zbc compliant device on HDB using software as opposed to firmware for the HDA. This means the HDB will be able to leverage read-ahead and the page-cache since it is seen as a block device by the OS. Secondly the write time for HDB on ext4 is much larger than the write time of the HDB using ZEAFS, which may indicate that compactions are occurring during read experiments which negatively impacts read times.



(a) Write Time (b) Sync Write Time (c) Read Time

Figure 4: **LevelDB Performance** *The graphs in Figure 4 compare the write/read performance of LevelDB on top of EXT4 and ZEAFS on two HDDs. The first HDD is a 6TB VendorA Drive (HDA) and the second drive is an 8TB VendorB SMR Drive (HDB)*

6 Discussion and Conclusion

In order to provide HM SMR software abstractions the complete I/O stack must be verified/modified to be SMR compatible. In addition, high-level abstractions such as file systems and KV stores that are designed to be SMR compliant lack the flexibility to be reused by other projects. In response to these challenges we designed and evaluated a SMR compatible extent allocator, ZEA. In this work we demonstrate the performance characteristics of ZEA by comparing it to a raw block device. In addition we were able to incorporate ZEA with LevelDB to benchmark a drive managed and emulated HM SMR HDD. ZEA shows promise in the fact that it smooths out performance for a DM HDD and is compatible with HM HDD. This current work demonstrates the utility of ZEA and in our future work we intend to build a SMR compatible KV store based on ZEA.

References

[1] Aghayev, A., and Desnoyers, P. Skylight: a window on shingled disk operation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (2015), USENIX Association, pp. 135–149.

[2] Amer, A., Holliday, J., Long, D. D. E., Miller, E. L., Pris, J.-F., and Schwarz, T. Data management and layout for shingled magnetic recording. *IEEE Transactions on Magnetics 47*, 10 (Oct. 2011).

[3] 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.

[4] Gantz, J., and Reinsel, D. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east.

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

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

[7] HGST. Libzbc.

[8] Jin, C., Xi, W.-Y., Ching, Z.-Y., Huo, F., and Lim, C.-T. Hismrfs: A high performance file system for shingled storage array. In *Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on* (June 2014), pp. 1–6.

[9] Jones, S., Amer, A., Miller, E. L., Long, D. D. E., Pitchumani, R., and Strong, C. Classifying data to reduce long term data movement in shingled write disks. In *Proceedings of the 31st International Conference on Massive Storage Systems and Technology (MSST 2015)* (June 2015).

[10] 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, July 2015), USENIX Association.

[11] Lin, C.-I., Park, D., He, W., and Du, D. 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* (Aug 2012), pp. 321–330.

[12] Lu, L., Pillai, T. S., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Wisckey: Separating keys from values in ssd-conscious storage. In *14th USENIX Conference on File and Storage Technologies (FAST 16)* (Santa Clara, CA, Feb. 2016), USENIX Association, pp. 133–148.

[13] Moal, D. L., Bandic, Z., and Guyot, C. Shingled file system host-side management of shingled magnetic recording disks. In *2012 IEEE International Conference on Consumer Electronics (ICCE)* (Jan 2012), pp. 425–426.

[14] Pitchumani, R., Hughes, J., and Miller, E. L. Smrdb: Key-value data store for shingled magnetic recording disks. In *Proceedings of SYSTOR 2015* (May 2015).

[15] Ren, K., and Gibson, G. Tablefs: Enhancing metadata efficiency in the local file system. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2013), USENIX ATC'13, USENIX Association, pp. 145–156.

[16] Seltzer, M., Bostic, K., Mckusick, M. K., and Staelin, C. An implementation of a log-structured file system for unix. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings* (Berkeley, CA, USA, 1993), USENIX'93, USENIX Association, pp. 3–3.

[17] Shiroishi, Y., Fukuda, K., Tagawa, I., Iwasaki, H., Takenoiri, S., Tanaka, H., Mutoh, H., and Yoshikawa, N. Future options for hdd storage. *Magnetics, IEEE Transactions on 45*, 10 (Oct 2009), 3816–3822.

[18] T10. Zoned block commands.

[19] Yiying Zhang, Leo Arulraj, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau. De-indirection for Flash-based SSDs with Nameless Writes. In *Proceedings of the 10th Conference on File and Storage Technologies (FAST '12)* (San Jose, California, February 2012).