Design and Implementation of a Hybrid Shingled Write Disk System

Dan Luo, Jiguang Wan, Yifeng Zhu, *Member, IEEE*, Nannan Zhao, Feng Li, and Changsheng Xie

Abstract—Disk data density improvement will eventually be limited by the super-paramagnetic effect for perpendicular recording. While various approaches to this problem have been proposed, Shingled Magnetic Recording (SMR) holds great promise to mitigate the problem of density scaling cost-effectively by overlapping data tracks. However, the inherent properties of SMR limit Shingled Write Disk (SWD) applicability since writing data to one track destroys the data previously-stored on the overlapping tracks. As a result, various data layout management designs have been proposed. In this paper, we present a hybrid wave-like shingled recording (HWSR) disk system, which can improve both the performance and the capacity of a shingled write disk. We propose a novel segment-based data layout management and a new wave-like shingled recording that overlaps adjacent tracks from two opposite radial directions. This new scheme can not only efficiently reduce the write amplification, but also double the areal density of conventional circular log-based shingled recording. A new replacement policy based on least write amplification is also devised to manage the hybrid system to effectively eliminate the performance degradation. Our measurements on HWSR implemented in Linux kernel 2.6.35.6 show that it provides superb performance. For example, HWSR reduces the average I/O response time by an order of magnitude compared to S-block for *Financial1* trace, and provides up to 3.7 speedup over standard hard disks without using shingled magnetic recording technology.

Index Terms—Shingled recording, SSD, hybrid system, replacement policy, data layout

1 Introduction

Currently, the areal density of magnetic disks is reaching its length-scale limitation. The capacity of a magnetic disk has increased 30-50 percent per year for almost 50 years. Disk areal density is quickly approaching to 1Tbit/in^2, a limit caused by superparamagnetic effect [6]. The magnetic direction of a sufficiently small particle can be randomly flipped under the influence of ambient thermal energy. Areal density scaling in magnetic hard drives is in jeopardy as magnetic particles become unstable when they are sufficiently small. New recording technologies have been proposed to scale up the areal densities, such as Bit-Patterned Media Recording (BPMR) [19], Microwave Assisted Magnetic Recording (MAMR) [9], and Heat Assisted Magnetic Recording (HAMR) [5]. Of the new technology being explored, Shingled Magnetic Recording (SMR) exposed as the most promising one to achieve high areal density increase and little changes to the manufacturing process. By partially overlapping tracks to reduce track width, the areal density can be improved to 2–3Tb/inch^2 [4]. Combined with 2-D readback and new signal processing techniques [7], [11], the areal density can be further enhanced 10Tb/inch^2 [4].

- *D. Luo, J. Wan, N. Zhao, F. Li, and C. Xie are with the Wuhan National Laboratory for Optoelectronics, Department of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, Hubei, P.R. China. E-mail: {luodan860514, lf921227}@gmail.com, {jgwan, cs_xie}@mail.hust.edu.cn, nnzhaocs@hotmail.com.*
- *Y. Zhu is with the Department of Electrical and Computer Engineering, University of Maine, Orono, Maine 04469. E-mail: zhu@eece.maine.edu.*

Manuscript received 30 Sept. 2014; revised 12 Feb. 2015; accepted 17 Apr. 2015. Date of publication 21 Apr. 2015; date of current version 16 Mar. 2016. Recommended for acceptance by M. M. Hayat.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2425402

The inherent weakness of shingled recording is the poor random write performance, because writing to a given data track requires rewriting its subsequent tracks. The amount of data actually written is extremely larger than the write request size. The inferior performance for small random writes, also called write amplification, is one of major factors restricting its widespread deployment in real systems. Therefor, the key challenge of extending the application of shingled writing and integrating it into magnetic disk systems is to lower the write amplification. Cassuto et al. [14] constructed an indirection system for shingled recording disks and introduced a data layout management, called S-block architecture that organized data into circular log. However, the circular log-based data layout has a high data immigration overhead during garbage collection (GC). Moreover, it has to maintain a large amount of metadata for tracking the dynamic mapping between a logical address and its physical location.

In this paper, we propose a hybrid wave-like shingled recording disk system (HWSR) to improve both the performance and the capacity of a shingled recording disk [1]. HWSR contains three different storage media: memory, SSD, and hard disk. The memory has a very small capacity, such as 64 MB, in our design to reduce the overall cost. It is used to buffer hot writes. The SSD is used as a read cache to improve the read performance, because SSDs have higher sequential and random read performance than HDDs, particularly random reads.

HWSR consists of three key components: (1) a new data layout based on segmentation for shingled recording disks to reduce random write amplification; (2) a new shingled track layout named wave-like shingled recording (WSR) to further improve its capacity; (3) a new replacement policy

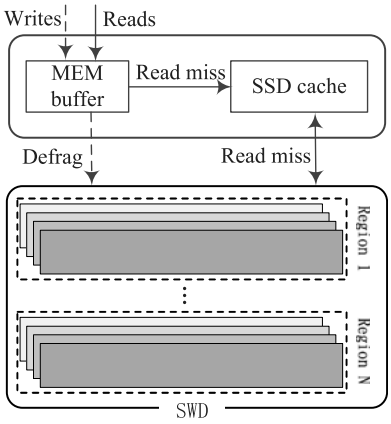


Fig. 1. HWSR system architecture.

based on least write amplification that effectively reduces the miss rate and the amount of rewritten data.

The key contributions of this paper are as follows.

- We have proposed a novel segment-based data layout management which limits random write amplification to a single segment by breaking a region into segments. A segment is much smaller than a region.
- We have devised a wave-like shingled recording that reduces half of wasted space and effectively improves disk utilization rate.
- We have also designed a new replacement policy based on least write amplification that greatly reduces the miss rate and data immigration.
- We have implemented HWSR in the Linux kernel 2.6.35.6 as a stand-alone kernel module and comprehensively evaluated its performance. The experimental results showed that HWSR provided superior performance.

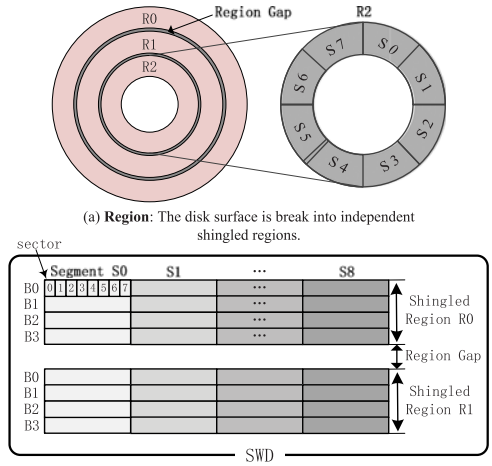
The rest of this paper is organized as follows. Section 2 gives an overview of HWSR system and presents our data layout, address mapping structure, and replacement policy. Section 3 discusses three key issues: write amplification, disk utilization, and metadata amount. We discuss our prototype implementation in Section 4. Performance evaluation is presented in Section 5 followed by related work in Section 6. Section 7 concludes this paper.

2 The Design of HWSR

In this section, we describe our system model and give an overview of HWSR.

2.1 Design Overview

Fig. 1 shows the HWSR system architecture consisting of a SSD cache, a MEM buffer and a shingled-recording disk. While the MEM buffer mainly stores write requests, the SSD cache mainly stores read requests. When a write request arrives, the MEM buffer stores incoming data and updates the address mapping table. When MEM buffer is full, MEM buffer evicts out some cold data to make space for new data. All evicted data is written to the disk. When a read request arrives, MEM buffer looks up the address mapping table. The read request accesses SSD if MEM buffer does not hold the target data. The shingled disk serves all misses of the SSD cache.



(a) **Region**: The disk surface is break into independent shingled regions.

(b) **Segment**: A shingled region is further divided into segments in the radial direction. In each segment, the sectors in the same track constitute one block.

Fig. 2. Data layout based on segmentation.

In the practical application, in order to avoid data loss, we can use NVRAM as the write buffer. NVRAM retains the non-volatile characteristics of secondary storage such as Flash memory or disks and has comparable performance to DRAM. In Section 5.3, we see that HWSR can obtain enormous performance improvement with a small size of MEM buffer (64 MB) and SSD cache. Therefore, by using small NVRAM as write buffer, HWSR adds little hardware cost. Moreover, we can just only use NVRAM as both the write buffer and read cache so that HWSR can be utilized as a drop-in replacement for standard disks.

And there is another method to protect data from loss in practice. In HWSR, we can use a logic control unit to monitor system power and in the case of an unexpected power failure, transfer the cache data from the DRAM to the SSD. A small battery or super capacitor is used to supply power until the transfer is complete. When the power is restored, the logic control unit restores the cache data from SSD to the DRAM. SSD is used as a read cache and all the data in SSD is clean. So, the logic control unit can store the cache data at any fixed location of SSD.

2.2 Data Layout

2.2.1 Segment-Based Data Layout

To deal with the expense of write amplification, SMR breaks the disk surface into smaller pieces called regions, consisting of a set of consecutive tracks. Regions are then separated by a gap called the Region Gap. The width of the Region Gap is just enough to ensure that a write to the last track of a region does not interfere with a write to the first track in the next region. Thus, breaking the disk into regions effectively reduces the write amplification to the size of the region in the worst case. Despite the improvement, this solution is far from ideal, even when region sizes are a modest 64 MB, let alone multi-GB sized regions.

In our scheme, we propose a segment-based data layout management. As shown in Fig. 2, the disk surface is first broken into shingled regions. Then a shingled region is further divided into segments of the same size in the radial

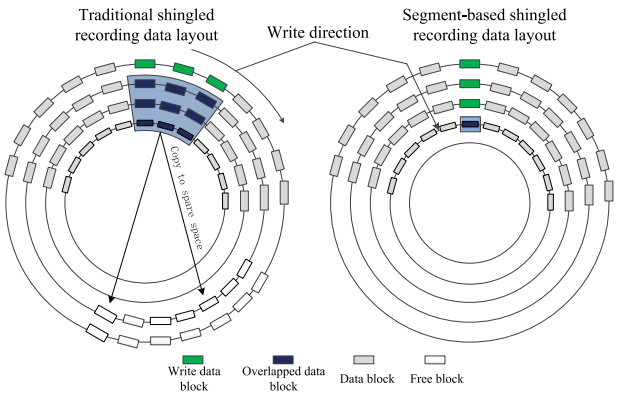


Fig. 3. Comparison of write amplification in traditional layout and our proposed segment-based layout.

direction. In each segment, the sectors in the same track constitute a data block. Fig. 2 shows an example of three regions, with eight segments in each region. Region R0 is divided into eight segments, each segment has four blocks (B0, B1, B2, B3), and each blocks is composed of eight sectors. Within a region, the logical addresses of two adjacent segment are consecutive. Within a segment, the logical addresses of two blocks on the adjacent track are consecutive. When writing sequential data blocks to a segment, the data blocks are laid out in the radical direction. Thus, if data in segment S1 is updated, its neighbor segments S2 and S0 are not affected.

Fig. 3 shows the comparison of write amplification between traditional data layout and our proposed segment-based data layout. In this example, we assume a region consists of four tracks. When three sequential data blocks marked in green are written to a traditional shingled disk, nine data blocks in the adjacent tracks of the same region have to be rewritten, i.e., we need to copy the nine data blocks to some spare space and then rewrite them back. This example shows the inherent properties of SMR: writing data to one track destroys the data previously-stored on the overlapping tracks. In segment-based SWD, the three sequential data blocks are laid out in the radial direction in a segment and only one data block are overwritten. In reality, a region may consist of more than one hundred tracks. The larger the number is, the higher the write amplification of the traditional shingled disk is. However, in segment-based SWD, a write or update operation overwrites the whole segment in the worst case. The segment size is much smaller than a region. Generally, the write amplification of data layout based on segmentation is $1/n$ of traditional data layouts, where n is the total number of segments in a region.

2.2.2 Wave-Like Shingled Recording Disk

Traditional HDDs store data in concentric tracks that are normally separated by a small gap to prevent inter-track cross-talk as shown in Fig. 4a. SMR is introduced to increase the areal density of hard disk. In a traditional shingled disk , the tracks are laid out with partial overlap in the radial direction as shown in Fig. 4b. We assume that the number of tracks that are overlapped by a shingled write is κ, where κ is usually 2-3 in reality. Theoretically, if there were no wasted space (guard band), the maximum capacity of a

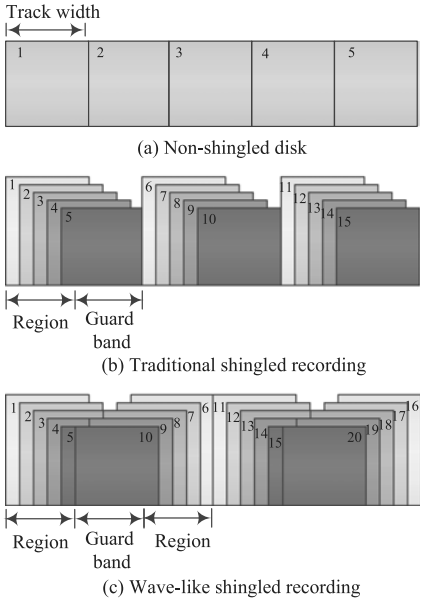


Fig. 4. Track layout.

shingled disk would be κ times of traditional disk as shown in Fig. 4a. Via overlapping tracks, the average track width is significantly reduced.

Data tracks are organized into bands called regions, which are separated by a collection of p following tracks called guard band, where p is at least κ. Guard bands are to prevent the interference between regions and do not store any valid data. Thus guard bands create significant spatial overhead. For traditional shingled recording, each region has its own guard band, which leads to considerable overhead. The tracks in wave-like shingled recording, as shown in Fig. 4c, are laid out with partial overlap in two opposite radial directions like waves. There is only one guard band shared by every two regions, which means that on average only half guard band is wasted for each region. Compared with traditional shingled recording, our Wave-like approach reduces the spatial overhead of traditional shingled disks by half, resulting in significant improvement of the utilization rate. We will discuss in detail the disk utilization ratio of wave-like shingled recording to traditional shingled recording in Section 3.

2.3 Replacement Policy

Upon a miss, cache or buffer must select a block to be replaced. HWSR uses different replacement policies for SSD cache and MEM buffer. Note that MEM buffer is mainly used to buffer writes and SSD cache is only used to cache reads because of its limited write cycles. In addition, evicted blocks out of MEM buffer must be written to the disk to maintain consistence while the replaced blocks out of SSD cache are not written to disk because the evicted data isn't changed.

As the price per byte of SSD continues to decrease, SSD with a relative small capacity (several hundred Mbytes) does not incur a significant cost. However, such a small SSD can effectively capture most hot data and have a high hit rate, as proved by our experimental results presented later. For simplicity, we use LRU replacement policy to manage the SSD cache.

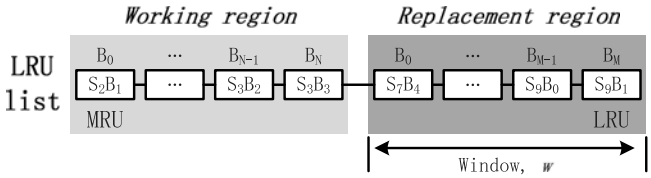


Fig. 5. LWA algorithm based on least data immigration.

There are two LRU queues in MEM buffer: one for blocks and the other for segments. The LRU block queue takes advantage of temporal locality but it replaces only one block at a time. This increases the number of defrag operations and write amplification because HWSR has to rewriting a segment for every defrag operation. The LRU segment queue replaces a segment each time and reduces write amplification. But it wastes significant cache space. For example, some blocks are hot in a segment while the other blocks are cold . Then this segment is in the front of LRU queue and its cold blocks will never be replaced.

We introduce a new LRU algorithm based on the least write amplification for MEM buffer, called LWA (least write amplification). As shown in Fig. 5, LWA divides the LRU list into two regions to reduce write amplification as well as keep the most hot data in LRU. The working region consists of recently used blocks and most of cache hits are generated in this region. The replacement region consists of blocks which are candidates for eviction. LWA selects a segment having the largest number of blocks in it as a victim block for replacement. LWA not only can keep the hot data in working region, but also can obtain maximum free blocks and reduce write amplification.

3 Discussion

In this section, we discuss some key issues existing in our HWSR and S-block architecture. Those key issues have significant impacts on the performance of shingled recording disks.

3.1 Write Amplification

The SSD and MEM work as a two-level hierarchical cache to the shingled disk. Upon a cache miss, it leads to prefetching data from the disk or defraging data to disk. Compared with the S-block architecture [14] that collects garbage when on a cache miss, our HWSR system only rewrites at most one segment. The average number of valid S-blocks that have to be immigrated to the head in garbage collection is ω as shown in Table 1. Note that the S-block in S-block architecture is the same size as segment in HWSR, so we replace S-block with segment for convenience of discussion. For our HWSR, we also need to rewrite some blocks in defrag operation when the request is a miss. However, the amount of rewritten data of HWSR is limited to the size of a segment when defrag occurs. As a result, the ratio of the average data immigration of S-block to HWSR is ω.

TABLE 1
Key Parameters of Workloads

	Parameters
Ω	The average segment immigration during GC in S-block
κ	The number of tracks that are overlapped by a shingled write
β	The spare capacity rate per region
n	Each region consists of n segments (or S-blocks)
H-S ratio	The disk utilization ratio of HWSR to S-block
S_{reg}	Region size (in tracks)

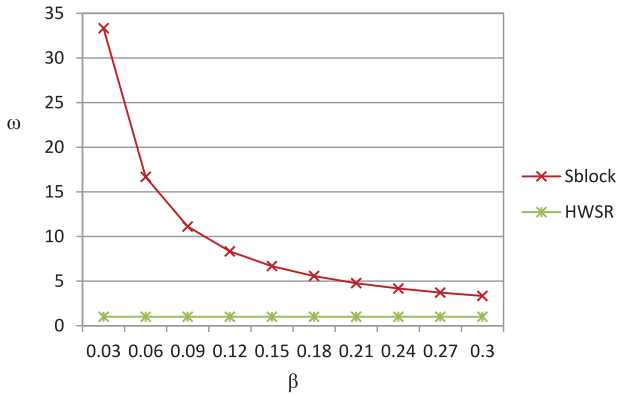


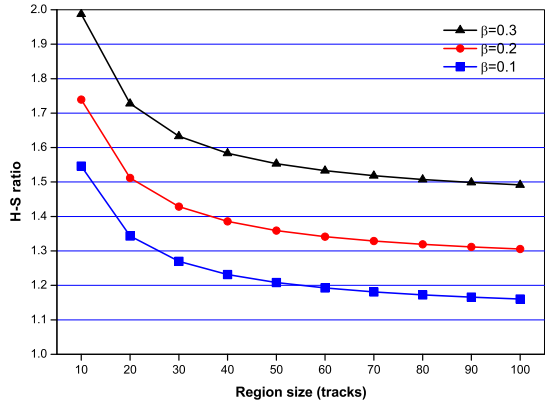
Fig. 6. The average number of segments immigration of S-block architecture during garbage collection.

Assume the number of segments in a region is n, and the spare segments in each region is β*n. The spare segments are used to reduce ω in S-block architecture. Generally, $\beta \neq 0$. If $\beta = 0$, the entire segments of circular log are immigrated to the head to free one invalid S-block when there is only one invalid segment in the head of circular log. Write amplification is limited to the total number of segments in circular log. Because each incoming update write is added to the head of circular log and the head segment are frequently updated, which means that in most cases invalid segments are concentrated to the head of circular log and almost all segments have to be immigrated to the head in order to free the invalid head segment. Approximately, ω can be considered as $\frac{1}{\beta}$, because we write β*n to a region and it will cause approximately n Sblocks immigrated when the whole segment is filled with n S-blocks. Note that there are always β*n invalid segments (spare segments) in each region. Moving all segments in circular log can release at least β*n invalid segments. We neglect the cases that there are some invalid segments in the tail or near the tail of circular log. This is reasonable because after a long term of writing, there are less invalid segments in the tail of circular log in the whole process.

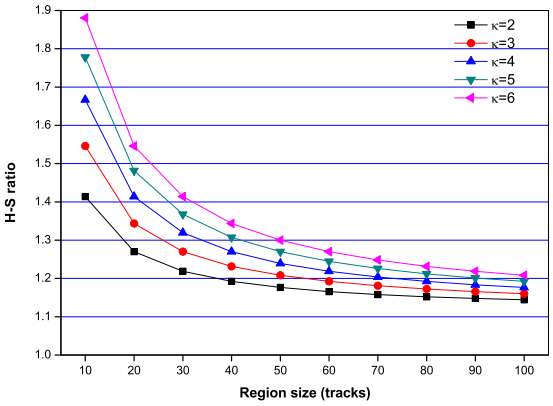
Fig. 6 compares the average number of segments immigration of S-block with different spare capacity rate and the average number of segments rewritten of HWSR. As β increases, the average number of segments immigration of S-block decreases.

3.2 Disk Utilization

In S-block architecture, there are internal guard bands in circular log-based data layout. Internal guard band is used to prevent the written data blocks from affecting existing blocks in each region. There are β*n spare segments that aim to mitigate the data immigration in S-block architecture. The region utilization rate of HWSR is 1 because HWSR doesn't need any spare segments in each region. The region



(a) H-S ratio when κ = 3



(b) H-S ratio when β = 0.1

Fig. 7. Disk utilization of HWSR to S-block (H-S ratio).

utilization rate of S-block architecture is $\frac{S_{reg}-\beta*S_{reg}}{S_{reg}}$. The region utilization ratio of HWSR to S-lock architecture is $\frac{1}{1-\beta}$. We take both the guard band and internal guard band into consideration.

The track utilization rate of S-block architecture which uses traditional shingled recording is $\frac{S_{reg}}{S_{reg}+2\kappa}$. The track utilization of HWSR is $\frac{S_{reg}}{S_{reg}+\kappa/2}$. The track utilization ratio of HWSR to S-block architecture is $\frac{2*(S_{reg}+2*\kappa)}{2*S_{reg}+\kappa}$. So the disk utilization ratio of HWSR system to S-block is $\frac{2*(S_{reg}+2*\kappa)}{(2*S_{reg}+\kappa)(1-\beta)}$.

Fig. 7a compares the disk utilization ratio of HWSR to S-block when κ is three and the region size varies. We observe that the disk utilization of HWSR is almost 2 $\times$ of S-block architecture when S_{reg} is 10 and β is 0.3. In other words, the disk utilization rate of HWSR doubles the traditional circular log-based shingled recording disk. From the g Fig. 7b studies the disk utilization ratio when β is fixed to 0.1. The H-S ratio has a maximum value of 1.88 when S_{reg} is 10 and κ is 6. From this figure, we see that HWSR can improve the disk utilization effectively when the region size is small. As κ decreases or S_{reg} increases, the H-S ratio decreases gradually. When the value of κ is small and the S_{reg} is large enough, HWSR can only increase the disk utilization a little. However, HWSR does not add additional overhead. When an access request requires to read or write one track, HWSR only need to redirect the request to another track by a simple calculation.

3.3 Metadata

HWSR directly uses address translation and it doesn't require a mapping table to map a logical address to its physical location on the disk. The data layout based on circular log requires a translation table to map LBAs to PBAs because their mapping information are not fixed. A large translation table creates significant overhead in a circular log-based shingled disk. According to Ref. [15], 1TB shingled writing disk requires over 24 GB space to store the metadata, which causes significant memory overhead. Data lookup in such a large table is often very slow. In addition, garbage collection is necessary to reclaim invalid blocks to accommodate later requests, which has a negative impact on the overall performance of the circular log-based shingled disk.

3.4 Design Issues

Higher average seek time is a key problem in our segment-based data layout management. Sequential data blocks are distributed on three adjacent tracks in a segment as shown in Fig. 3. If the required data is located on multiple blocks, the traditional sequential data layout shingled recording has less seek time than the data layout based on segmentation. In our design, we make a tradeoff between write amplification and average seek time. There are three reasons. (1) Based on the observation in our experiments, the writes latency contributes to a large proportion of total latency, and the segment-based data layout optimizes the write performance due to greatly reduced write amplification. (2) HWSR uses SSD to cache reads, which effectively reduces the number of random disk reads, resulting in better read performance. (3) Compared with circular log-based data layout, which have to move a great number of valid S-blocks from the tail to the head during garbage collection, resulting in increased write amplification, HWSR exhibits stable performance even under the workload with a lot of random accesses as shown in the following section.

4 Implementation

In this section, we describe the details of the prototype of HWSR.

4.1 Prototype Implementation

We have implemented our HWSR in the Linux kernel 2.6.35.6 as a stand-alone kernel module without any modifications in the Linux kernel. HWSR works as a pseudo block device at the block layer, as shown in Fig. 8. The upper-level components, such as file systems or applications, view HWSR simply as a single block device, despite the complicated internals. Without any system changes, HWSR is easy to be integrated into existing systems.

HWSR has two major components, namely cache management and address translation. Cache management includes SSD cache and MEM buffer management. The cache resources (including SSD cache and MEM buffer) are allocated in chunks which are of the same size as a block in HWSR. In other words, a block is the smallest unit for read and write requests that are sent to hard disk. This brings two benefits. First, when moving data into the SSD, organized writes are more efficient if they are in a reasonably

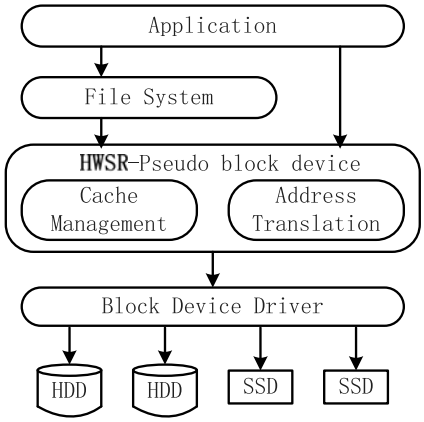


Fig. 8. HWSR in I/O stack.

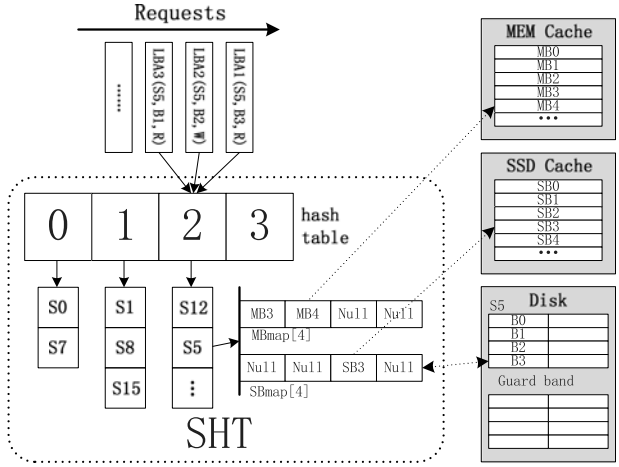


Fig. 9. Address mapping translation.

large request. Second, it avoids splitting a request into several excessively small requests. The mapping table is implemented as a block-level hash table as described in Section 4.2. When the size of the cache is kept constant, the larger the block size is, the fewer number of the block can be contained in the cache. Thus the block size has a significant impact on cache hit rate. Region size is another key parameter affecting system performance. We will discuss these issues later in detail in Section 5.5. When the required data is not contained in the cache, the physical address of the accessed data is calculated by using Equation (5).

We use a two platters 1TB Western Digital SATA drive as a test drive to emulate the shingled disk. In order to simplify the implementation, we assume that the hard disk has a fixed number of sectors per track, and the capacity of each track on the disk is $8 \times 1{,}024$ sectors (512 bytes in one sector).

For comparison, we have also evaluated S-Block Architecture that is also implemented in the Linux kernel 2.6.35.6 as a stand-alone kernel module. We choose the S-Block with the most blocks in the cache buffer for group destage which is the best amortization of S-Block write over invalidated blocks. Except the data layout of shingled disk, the parameters of buffer or cache of S-block architecture are the same as HWSR.

4.2 Address Mapping Structure

To reduce the amount of metadata and ensure consistence, we construct the address mapping structure between SSD cache, MEM buffer, and disk. The logical address space of MEM buffer and SSD cache is divided into independent blocks which have the same size as the blocks of shingled disk. There is one hash table that maps the logical address of requests to the cache or buffer space. If a request has a cache miss, it accesses the disk directly using the following address translation formula:

$$N_{reg} = LBA / S_{reg} \tag{1}$$

$$N_{seg} = (LBA \bmod S_{reg}) / S_{seg} \tag{2}$$

$$N_{blk} = (LBA \bmod S_{seg}) / S_{blk} \tag{3}$$

$$N_{offset} = LBA \bmod S_{blk} \tag{4}$$

$$PBA = N_{reg} * S_{reg} + N_{blk} * S_T + N_{seg} * S_{blk} + N_{offset}, \tag{5}$$

where LBA and PBA are the logical and physical address respectively(in sectors); N_{reg}, N_{seg}, N_{blk} and N_{offset} are the logical section number, segment number, block number and sector number respectively; S_T is the track size (For the convenience of representation, we assume all tracks have the same size); S_{reg}, S_{sec} and S_{blk} are region size, section size and block size respectively (in sectors).

We construct a segment hash table (SHT) in memory that stores segment information in each hash node. Each segment node uses two types of arrays to store block locations. *SBmap* array is used to store the block locations in the SSD cache and *MBmap* array is used to store the block locations in the MEM buffer as shown in Fig. 9.

SHT is used to speed up the lookup in *SBmap* and *MBmap*. If an element in one array is valid, then the requested data content is stored at the location of corresponding devices. If invalid, then the requested data is not in SSD or Mem, and then the disk will be accessed. Fig. 9 gives an example of data lookup operations for two read requests (LBA1 and LBA3) and one write request (LBA2) in HWSR. We assume that the number of blocks per segment is four and the segment hash value is N_s%4, where N_s is the segment number. For read request LBA1 (S5, B3, R), the content inside the parentheses shows that the read request accesses block 3 in segment 5. In the case of reading LBA1, MBmap[3] and SBmap[3] are both NULL, then the request accesses shingled recording disk and the desired data is prefetched to SSD cache. In the case of writing LBA2, the content of SBmap[2] is SB3, which means the required block is stored in block 3 in the SSD cache. In the case of reading LBA3, the content of MBmap[3] is valid, and the target block is stored in block 4 in the MEM buffer.

4.3 LWA Implementation

To implement the LWA replacement algorithm in the Linux kernel, we construct a double circular linked list of blocks as shown in Fig. 10. An incoming write is added to the head of the circular list. During a miss, the victim block is chosen within a predefined lookup window starting from the tail of the circular list. The victim blocks have to belong to the same segment which has the largest number of blocks in the lookup window. For example, MB12, MB13, and MB15 will

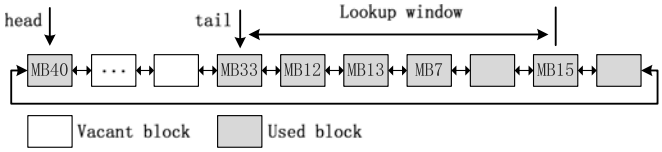


Fig. 10. The implementation of LWA.

be evicted out of the list. MB15 and MB12 belong to the same segment (segment1) as MB13, and segment1 has the largest number of blocks in the lookup window. Because replaced blocks have to belong to the same segment, which limits the rewrites to one segment, this algorithm not only maintains the most hot data and discards the cold data, but also reduces write amplification.

5 Performance Evaluation

5.1 Experimental Setup

The experiments for the evaluation were run in a host with a 2.13 GHz Intel Core Quad CPU and a 8 GB RAM. Our prototype system consists of a 1 TB Western Digital 7200 SATA hard drive, a 120 GB Intel 320 series SSD, and a small portion of main memory. Note that we only use partial SSD space in our experiments to avoid overestimating the performance. We use Fedora Core 14 with the Linux kernel 2.6.35.6 and Ext4 file system with default configuration. In order to minimize the interference, the operating system and home directory are stored in a separate hard disk drive. Table 2 list the detailed description of the experimental setups.

Unless otherwise stated, the sample parameters are now provided as follow. A region consists of 32 contiguous tracks on the same surface, i.e., a segment has 32 tracks. The size of a block is set to 128 sectors. The SSD size and MEM buffer size are 128 and 64 M respectively.

5.2 Workload Characteristics

In order to fairly evaluate the performance of HWSR, we use real world I/O workloads that have meaningful contents as well as access patterns similar to real applications. We have selected three standard benchmarks and 12 real world traces in our performance evaluation experiments.

Postmark is a widely used file system benchmark [26]. It creates 100 directories and 20,000 files, then performs 100,000 transactions to stress the file system, and finally deletes the files.

SysBench is a multi-threaded benchmark tool for evaluating the capability of a system to run a database under intensive load [24]. SysBench runs against MySQL database with a table of size 4,000,000, max requests of 100,000, and 16 threads.

TPC-C is a benchmark modeling the operations of real-time transactions. It simulates execution of a set of distributed and on-line transactions (OLTP) on a number of warehouses. These transactions perform the basic database operations such as inserts, deletes, updates and so on. TPCC-UVA [25] is used on the Postgres database with five warehouses, 10 clients for each warehouse, and 2 hours running time.

We also replay the I/O traces of twelve representative application to evaluate our design. The key characteristics of the 12 traces are summarized in Table 3. The first two traces (*Financial1* and *Financial2*) are collected from OLPT applications which run at two large financial institutions. The other ten traces are collected from enterprise servers at Microsoft Research Cambridge. We use *blktrace* to directly replay the twelve different I/O traces to the block device that is created by inserting our kernel module.

TABLE 2
Experimental setups

OS	Linux version 2.6.35.6-45.fc14.x86_64	
CPU	Intel(R) Xeon(R) CPU E5506 @ 2.13 GHz	
Memory	Hynix DDR3 4 GB 2R*4 PC3-10,600 R	
Hard disk	WD 1TB SATA/64 MB Cache 3 Gb/s 7,200 rpm	
SSD	Intel SSDSA2CW120G3 3 Gb/s SATA 120 G	
Parameters	Emulated shingled disk	1TB
	SSD Cache	128 MB
	MEM Buffer	64 MB
	Block Size	128 Sectors
	Region Size	32 Tracks

TABLE 3
Characteristics of Traces

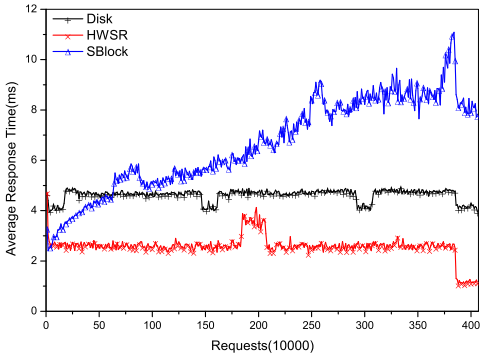
Traces Name	Total Requests	Unique Data Size	Avg. Read Len	Avg. Write Len	# of Updates	Write Percent
Financial1	1,000,000	0.43 GB	3167B	4525B	749,893	78.73%
Financial2	1,000,000	0.24 GB	2166B	3020B	157,481	18.19%
mds0	1,000,000	3.06 GB	26021B	7423B	856,083	86.88%
mds1	1,000,000	51.78 GB	60233B	14057B	37,958	5.22%
prn0	1,000,000	3.99 GB	25019B	10573B	793,929	85.74%
prn1	1,000,000	6.02 GB	13794B	9840B	174,255	28.97%
proj0	1,000,000	1.93 GB	19717B	13262B	758,902	77.43%
proj1	1,000,000	31.27 GB	36394B	23836B	62,869	17.54%
usr0	1,000,000	2.21 GB	42916B	9801B	600,846	61.40%
usr1	1,000,000	33.14 GB	42705B	7154B	54,190	12.67%
prxy0	1,000,000	0.27 GB	5900B	2441B	937,862	95.26%
wdev0	1,000,000	0.49 GB	12861B	8406B	781,367	79.41%

5.3 Evaluation Results

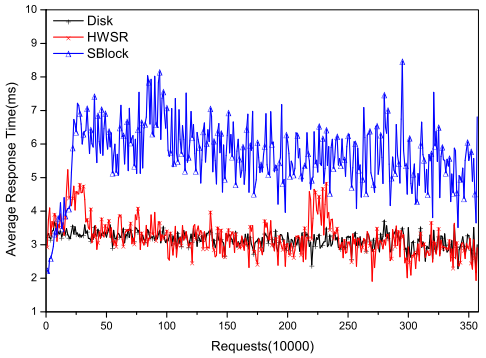
5.3.1 Benchmarks

Our first experiment is on *SysBench*. Fig. 11a shows the average response time per 10,000 requests of SysBench running on three different storage systems, including HWSR, S-block, and standard hard disks. Obviously, HWSR provides superior performance than hard disk and S-block. Among the three storage architecture, HWSR performs the best showing 1.57× faster than hard disk, 2.85× better than S-block as plot in Fig. 12a.

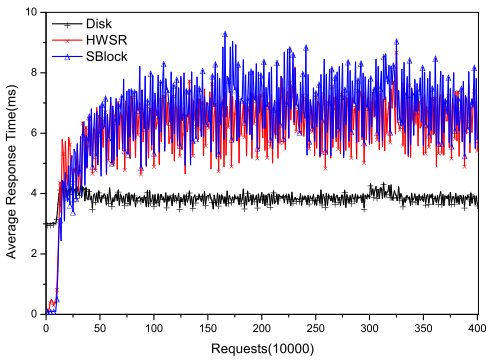
In order to better understand why HWSR performs better than hard disk and S-block, we measured the average read and write response time, the number of segments rewritten, and cache hit ratio as shown in Fig. 12. The reason why HWSR has lower response time than disk is mainly because the cache absorbs the majority of data requests. In Fig. 12c, we see that the measured hit ratios are 48 and 43 percent for HWSR-MEM and HWSR-SSD respectively, i.e., in total, 91 percent of the data accesses are hit in the cache which improves system performance substantially. Another reason is that the number of segments rewritten is fewer in HWSR which benefits from our segment-based data layout management.



(a) Sysbench

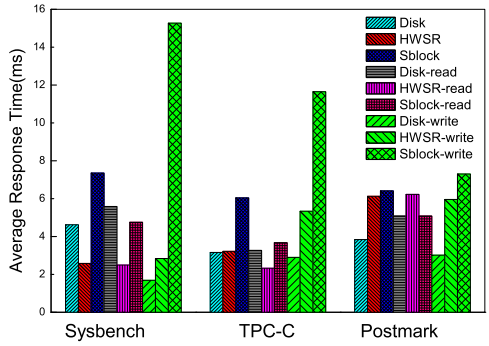


(b) Tpc-c

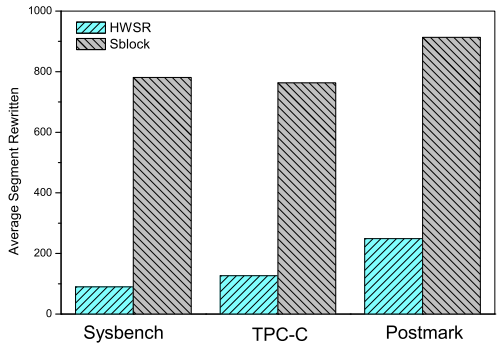


(c) Postmark

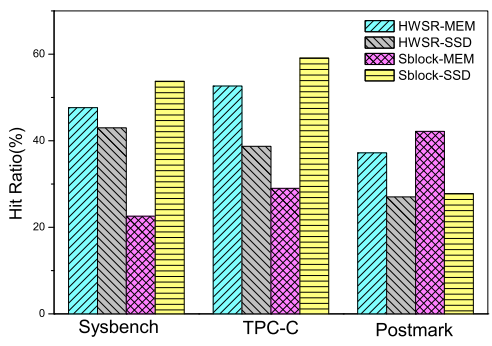
Fig. 11. Average response time per 10,000 Requests.



(a) Average response time



(b) Average segment rewritten per 10000 requests



(c) MEM and SSD hit ratio

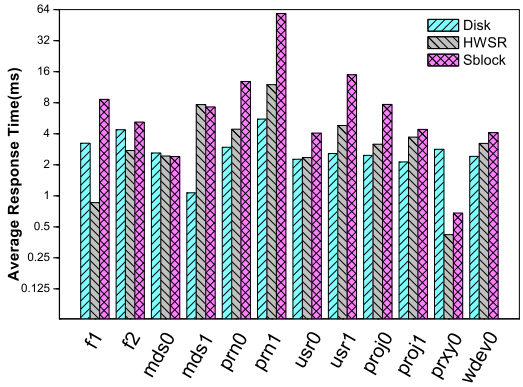
Fig. 12. Average response time, average segment rewritten and Hit ratio.

Fig. 12a shows that the average response time is 2.58 and 7.35 ms for HWSR and S-block respectively. Although S-block architecture also has a high cache hit ratio like HWSR, the average response time of S-block is much higher than HWSR. From Fig. 12b, we can find some clues as to why S-block has such a higher average response time. The average number of segments rewritten per 10,000 requests in S-block is about eight times as many as HWSR. As more segments are immigrated in S-block architecture, the number of disk head movement increases tremendous between the head and the tail of S-block circular buffer. Thus, the seek time and rotate time increase greatly, which degrades the S-block performance severely. As demonstrated in Fig. 12a, the average write response time of S-block is about five times more than HWSR, which also reflects the performance degradation of S-block resulting from high overheads of segment immigrations.

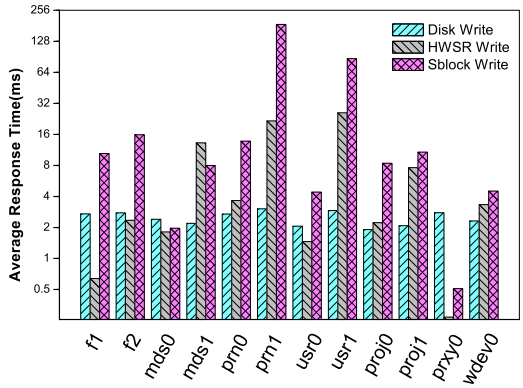
The measured *TPC-C* average response time is shown in Fig. 11b for the three different storage architectures. It is clear that HWSR out-performs S-block with a speedup of 1.9, and shows similar performance to hard disk. The reason why S-block has much longer response time than others is also attributed to tremendous amount of segment immigrations. Fig. 12a shows the detailed read and write response times measured at block I/O level. We see that the average write response time of S-block is much longer than HWSR.

HWSR has a longer write response time in TPC-C than in Sysbench. In TPC-C benchmark, clients commit small transactions frequently generating a large amount of write requests. Correspondingly, more memory defrag operations are generated. As a result, HWSR spends more time in rewriting segment resulting in longer write response time.

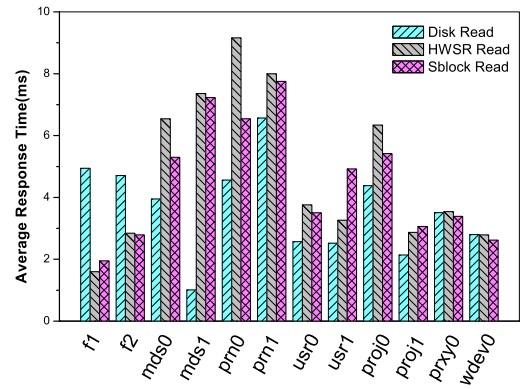
Postmark benchmark results are shown in Fig. 11c. Apparently, hard disk performs better than HWSR and S-block. The average response time per 10,000 requests is 3.84, 6.13, and 6.41 ms for hard disk, HWSR, and S-block, respectively. Two reasons can explain this. First of all, postmark benchmark features intensive small random data accesses. The cache hit ratio is very low as shown in Fig. 12c



(a) Average response time



(b) Average write response time



(c) Average read response time

Fig. 13. Average response time on different traces.

because of the completely random characteristic of data accesses. As a result, the cache hit ratio is relatively low and many requests must be served by disk. Second, as the cache hit ratio decreases, the number of segments rewritten increases substantially which results in severe performance degradation. Although HWSR performs worse than hard disk, the overall performance is still quite satisfactory.

5.3.2 Traces

Fig. 13 shows the read, write, and total average response time for 12 different traces. It is clear from Fig. 13a that for almost all traces HWSR out-performed S-block with speed-ups ranging form 1.2 to 32.8, except that HWSR exhibited a slightly lower performance than S-block for *mds*. For most

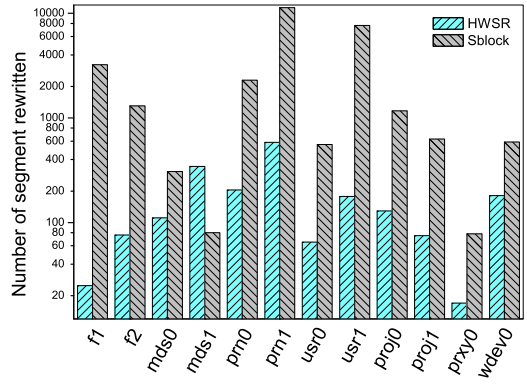


Fig. 14. AVG segment rewritten per 10,000 requests.

traces, HWSR clearly showed superb performance which is comparable to hard disk. Even for some traces with high data locality, HWSR provided much better performance than hard disk. Take *prxy0* for example, the total average response time for hard disk and HWSR is 2.82 and 0.42 ms, respectively.

Mds traces are generated from media servers and thus almost all requests are sequential reads or sequential writes. For HWSR, reading/writing two blocks on adjacent tracks takes more seek time than reading two sequential blocks on one track for S-block. This is why S-block performs a little better than HWSR for *mds*.

As demonstrated in Fig. 13b, the average write response time of HWSR is much lower than S-block for most traces except *mds1*. For *financial1*, the average write response time for HWSR and S-block is 0.65 and 10.43 ms, respectively. HWSR is 16× faster than S-block. The reason why S-block has such a high average write response time is mainly because of high overheads incurring by garbage collection. From Fig. 14, we see that the average number of segments rewritten in S-block is extremely larger than HWSR. For *prn1*, the average number of segments rewritten per 10,000 requests reaches up to 11,310. This means that disks are busy in moving data from the tail to the head during garbage collection of S-block architecture. Due to our segment-based data layout management, the write amplification of HWSR is reduced greatly.

HWSR exhibits a slightly lower performance on reads compared with S-block architecture as shown in Fig. 13c. Reading two consecutive blocks requires an extra seek time for HWSR according to our segment-based data layout. This is the main reason why HWSR has a slightly higher read response time. Because of the exist of SSD cache and MEM buffer, many request accesses are served directly by cache. The read performance gap between HWSR and S-block is small. In almost all cases, HWSR has similar read performance to S-block. Even under some workloads, HWSR provides better performance than S-block. More importantly, the write latency accounts for most proportions of the total latency. Thus HWSR often achieves a better performance than S-block due to our write optimization.

From above discussions, we see that with a small size of MEM buffer and SSD cache, HWSR can obtain stable and superior performance. In almost all cases, HWSR shows much better performance than S-blocks. In some cases, HWSR even performs better than standard hard disks

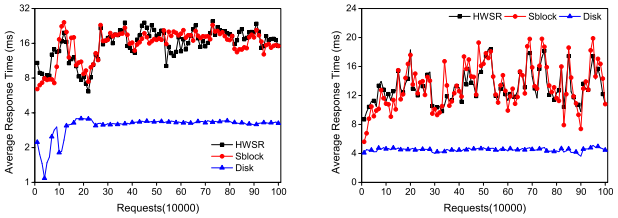


Fig. 15. I/O performance without SSD read cache and MEM write buffer.

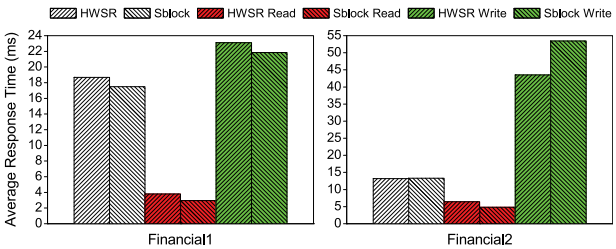


Fig. 16. Average response time without SSD read cache and MEM write buffer.

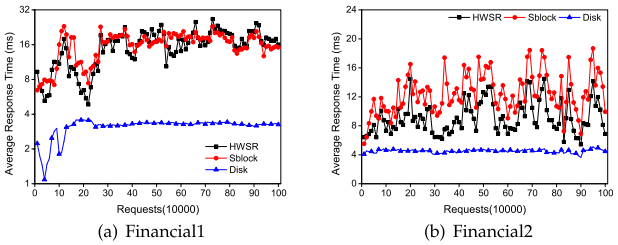


Fig. 17. I/O performance with SSD read cache.

without using SMR. Even in the worst-case scenario, the performance of HWSR is still quite satisfactory.

5.4 The Impact of Caching on HWSR Performance

In order to better characterize the behavior and performance of HWSR, we took a close look at how the SSD cache and MEM buffer affect HWSR performance.

Fig. 15 shows the I/O performance of HWSR and S-block without SSD cache and MEM buffer. As shown in Fig. 15, disk performance is much better than S-block and HWSR provides a slightly lower performance than S-block. The average response time of HWSR and S-block is more than five times higher than standard disk for *Financial1*. The result is in accordance with our expectation because SWDs have the write amplification problem. As shown in Fig. 16, the average response time of HWSR is higher S-block by 1.5 ms for *Financial1*. For HWSR, every write request requires to rewritten the whole segment, which results in high I/O latency. For S-block, every S-block circular buffer has an associate cache circular buffer that is used to store incoming writes. This is why the performance of S-block is a little better than HWSR. However, high overhead of garbage collection also degrades S-block performance substantially.

Fig. 17 shows the I/O performance of HWSR and S-block with SSD read cache. From Figs. 16 and 18, we observe that the performance of HWSR and S-block are both improved and HWSR performs a little better than S-block. It is interesting to observe that the write performance of HWSR is improved greatly. Compared to HWSR without SSD cache

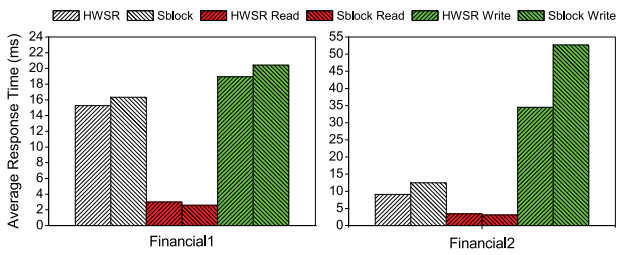


Fig. 18. Average response time with SSD read cache.

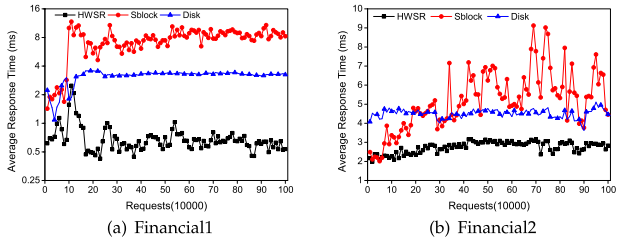


Fig. 19. I/O performance with SSD read cache and MEM write buffer.

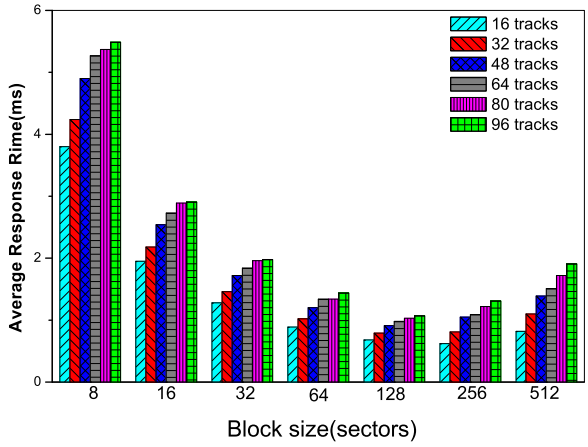
and MEM buffer, the average write response time of HWSR with SSD read cache is reduced from 43.53 to 34.46 ms for *Financial2*. When rewriting data blocks to one segment, HWSR needs to read the other data blocks of that segment from disk. Because of the exist of the SSD read cache, HWSR can read some data blocks directly from high-performance SSD instead of disk, which accelerates the rewriting process. For S-block, SSD read cache improves only read performance, but does not bring any benefit to write performance. This is why HWSR obtains much more performance improvement than S-block when there exists SSD read cache.

Fig. 19 shows the I/O performance of HWSR and S-block with SSD cache and MEM buffer. For *Financial1*, HWSR performs the best showing an order magnitude faster than S-block, 3.7× better than standard disk. The reason why MEM buffer brings much more performance gains to HWSR than to S-block is that HWSR is able to take good advantage of strong spatial locality of the workload. When MEM buffer is full, HWSR uses LWA replacement policy to select victim blocks that belong to the same segment. The logical address of these victim blocks are consecutive in the logical address space. By using LWA replacement policy, HWSR decreases defrag operations dramatically and reduces write amplification efficiently. Therefor, HWSR provides superior performance.

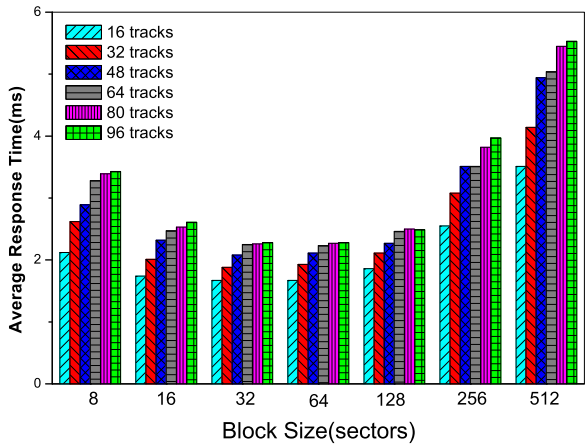
5.5 The Impact of Block Size and Region Size

In this section, we study the impact of block size and region size on the performance of HWSR. Fig. 20 plots the average response time of HWSR under different block sizes and region sizes. In Fig. 20, 16 tracks in the legend means that a region consists of 16 continuous track on the same surface in hard disk. We use the number of tracks to denote the region size. A segment has the same number of tracks as a region.

First, we can clearly observe in Fig. 20 that when the block size is kept constant, the average response time increases monotonically as the region size increases. This result is consistent with our institution. As is known to all, writing data to one track in shingled write disk destroys the data previously-stored on the overlapping tracks. Once the



(a) Financial1



(b) Financial2

Fig. 20. The impact of block size and region size.

incoming update being served, HWSR writes the data back to its original position. If writing the data back further impacts other stored data, the same operation must be performed. Such operation will be continuously executed until no more data will be affected (i.e., no data stored on the subsequent tracks or reach a region edge). Therefore, as the number of tracks increases in a segment, the affected tracks ascend and the write amplification increases resulting in performance degradation correspondingly. S-block architecture has the same problem as the region size increases because the number of valid S-blocks immigrations increases monotonically with increasing region size during garbage collection. Although the write amplification increases as the region size increases, HWSR effectively reduced the write amplification to the size of a segment by breaking a region into segments. From above discussion, we see that the number of segments rewritten in S-block is higher than HWSR by almost two order of magnitude.

Second, Fig. 20 shows that when the region size is kept constant, as the block size increases, the average response time decreases gradually at the beginning, and then increases after reaching the lowest point. In general, a larger block can capture more requests and accordingly less requests are separated to different blocks. For HWSR, the average seek time is reduced because less requested data are divided into different blocks which are mostly located on two adjacent tracks. Thus, the performance improves at the beginning as the block size increases. However, when the block size continues to increase, the number of blocks that can be contained in the cache become less as described in Section 4. Correspondingly, the cache hit ratio decreases because substantial cold data is buffered in the cache. Thus, when the block size continues to increase, the performance degrades. Block size is a key parameter which has a significant effect on system performance. In our future work, block size will be set to a tunable parameter. We can change the block size to suit for different workloads.

6 Related Work

Recently many researches work on high density recording technology.

One approach of improving the areal density of magnetic disks is to change recording medium to avoid the superparamagnetic limit. Examples include Bit-patterned magnetic recording (BPMR), Heat-assisted magnetic recording, and microwave assisted magnetic recording. However, those technologies dramatically change the structure of underlying and mechanical design and disk head sensors of existing magnetic disks, which might introduce significant costs to disk manufacture.

Another approach to achieve high density is shingled recording technology that partially overlaps tracks to narrow the track width. Cross and Montemorra [21] demonstrated that by using a conventional disk head, the areal density of shingled recording disk can exceed 800 Gb/in^2. And with the stronger write field, the areal density of shingled write disk can be increased to around 2Tb/in^2 [4], [8]. Miura [20] pointed out that high density can be attained when the reader is accurately placed on the center line of data tracks by analyzing different heads and media and estimating the maximum track density of shingled writing. Combined with 2-D readback and advanced signal processing (TDMR), SMR can achieve an areal density of 10 Tb/in^2 [7], [11]. Reading from narrower tracks by using a wide reader causes inter-track interference (ITI). To address this problem, Ozaki et al. [23] proposed an ITI canceller to reproduce waveform from a shingled recording disk. In order to evaluate the performance of SWDs, Pitchumani et al. [18] designs a novel SWD emulator that uses a hard disk utilizing traditional Perpendicular Magnetic Recording and emulates a Shingled Write Disk on top of it.

Shingled disks have attracted a lot of attentions due to its density improvement by at least 2T/in^2 [4] and its cost-effectiveness since it does not require to rework the storage media. But its inferiors performance, particularly for random writes, limit its application scope to minimal update workloads, such as archival workloads. Currently most existing research works tackle this problem by designing new data layout for shingled disks [12], [13], [14], [15], [16], [17]. In [12], [13], SWD is divided into log access zones (LAZs) and random access zones(RAZs), where LAZs store user data and RAZs store metadata respectively. The LAZs are organized as circular log structure and logs at each level store different data and take different clean strategies. Cassuto et al. [14] constructed an indirect system which contains two data layout methods. The first one is a set-associative disk cache architecture that divides SWD into data

zones and cache zones, where data zones are used for permanent data storage while cache zones are used for caching incoming write/update requests. Data zones are related to cache zones set-associatively. The second one is S-block architecture which organizes data as a circular log. Park et al. [15] proposed H-SWD to reduce the garbage collection of circular log by using a hot data identification mechanism. In [16], a SWD is divided into I-regions and E-regions. E-regions serve incoming writes/updates while I-regions store data moved from E-region during background garbage collection.

The closest work to ours is the shingled file system (SFS) [17], which is a host-managed design for in-place update SWDs. The disk space is divided into shingled zones separated by gaps (unused tracks). SFS furthermore differentiates shingled zones into sequential-write shingle zones and random-write shingle zones. Although SFS reduces the overhead of random updates, it did not address the write amplification problem.

It is mentioned that SSD and NVRAM can be embedded as cache for delaying updates to shingled recording disks and reduce data rewritten [3], [7], [12]. Gibson and Ganger [2] proposed a shingled translation layer (STL) in an embedded controller to mask the random write restriction and integrated shingled writing into magnetic disks.

7 Conclusions

This paper presents a hybrid wave-like shingled recording disk system to address the problem of poor performance for small random writes on shingled write disks. We design a novel segment-based data layout management and a new wave-like shingled recording that can not only double the disk space utilization of conventional circular log-based shingled disks, but also effectively limit the write amplification to a small segment. Our hybrid system combines shingled disks with fast memory that works as buffer for writes and SSD that works as cache for reads. We also design a new LRU algorithm based on least write amplification to optimize the overall I/O performance. We have prototyped HWSR in the Linux kernel 2.6.35.6 as a stand-alone kernel module. Experimental evaluations on our prototype under a variety of I/O intensive workloads show that HWSR efficiently reduces the write amplification and improves the performance of small writes significantly. While our new data layout slightly increases the average seek time as sequential data blocks are stored on adjacent tracks, such degradation can be effectively masked by the memory buffer and the SSD cache. All in all, HWSR performs much better than S-block and even provides superior performance to disk in some cases.

Acknowledgments

The authors are grateful to the anonymous reviewers for their constructive comments. This work is sponsored in part by the National Natural Science Foundation of China under Grant No. 61472152, the National Basic Research Program of China (973 Program) under Grant No. 2011CB302303, the Fundamental Research Funds for the Central Universities, HUST:2015QN069, and the National Natural Science Foundation of China under Grant No. 61432007 and No. 61300047. J. Wan is corresponding author.

References

[1] J. Wan, N. Zhao, Y. Zhu, J. Wang, Y. Mao, P. Chen, and C. Xie, "High performance and high capacity hybrid shingled-recording disk system," in *Proc. IEEE Cluster*, 2012, pp. 173–181.

[2] G. Gibson and G. ganger, "Principles of operation for shingled disk devices," *Phil. Trans. Roy. Soc. London*, vol. A247, pp. 529–551, Apr. 1955.

[3] G. Gibson and M. Polte, "Directions for shingled-write and two-dimensional magnetic recording system architectures: Synergies with solid-state disks," Carnegie Mellon University Parallel Data Lab, Pittsburgh, PA, Tech. Rep. CMU-PDL-09-014, May 2009.

[4] S. Greaves. Y. Kanai, and H. Muraoka, "Shingled recording for 2–3 Tbit/in^2," *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3823–3829, Oct. 2009.

[5] M. Kryder, E. Gage, T. McDaniel, W. Challener, R. Rottmayer, G. Ju, Y.-T. Hsia, and M. Erden, "Heat assisted magnetic recording," *Proc. IEEE*, vol. 96, no. 11, pp. 1810–1835, Nov. 2008.

[6] H. Richter, A. Dobin, O. Heinonen, K. Gao, R. veerdonk, R. Lynch, J. Xue, D. Weller, P. Asselin, M. Erden, and R. Brockie, "Recording on bit-patterned media at densities of 1Tb/in^2 and beyound," *IEEE Trans. Magn.*, vol. 42, no. 10, pp. 2255–2260, Oct. 2006.

[7] Y. Shiroishi, K. Fukuda, I. Tagawa, S. Takenoiri, H. Tanaka, and N. Yoshikawa, "Future options for HDD storage," *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3816–3822, Oct. 2009.

[8] I. Tagawa and M. Williams, "High density data-storage using shingle-write," in *Proc. IEEE Int. Magn. Conf.*, May 2009.

[9] J. -G. Zhu, X. Zhu, and Y. Tang, "Microwave assisted magnetic recording," *IEEE Trans. Magn.*, vol. 44, no. 1, pp. 125–131, Jan. 2008.

[10] R. Wood, "The feasibility of magnetic recording at 1 terabit per square inch," *IEEE Trans. Magn.*, vol. 36, no. 1, pp. 36–42, Jan. 2000.

[11] R. Wood, M. Williams, A. Kavcis, and J. Miles, "The feasibility of magnetic recording at 10 terabits per square inch on conventional media," *IEEE Trans. Magn.*, vol. 45, no. 2, pp. 917–923, Feb. 2009.

[12] A. Amer, D. D. E. Long, E. L. Miller, J.-F. Paris, and S. J. T. Schwarz, "Design issues for a shingled write disk system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–12.

[13] A. Amer, J. Holliday, D. D. Long, E. Miller, J.-F. Paris, and T. Schwarz, "Data management and layout for shingled magnetic recording," *IEEE Trans. Magn.*, vol. 47, no. 10, pp. 3691–3697, Oct. 2011.

[14] Y. Cassuto, M. A. A. Sanvido, C. Guyot, D. R. Hall, and Z. Z. Bandic, "Indirection systems for shingled-recording disk drives," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–14.

[15] C.-I. Lin, D. Park, W. He, and D. Du, "H-swd: Incorporating hot data identification into shingled write disks," in *Proc. IEEE 20th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, Aug. 2012, pp. 321–330.

[16] D. Hall, J. Marcos, and J. Coker, "Data handling algorithms for autonomous shingled magnetic recording hdds," *IEEE Trans. Magn.*, vol. 48, no. 5, pp. 1777–1781, May 2012.

[17] D. Le Moal, Z. Bandic, and C. Guyot, "Shingled file system host-side management of shingled magnetic recording disks," in *Proc. IEEE Int. Conf. Consum. Electron.*, Jan. 2012, pp. 425–426.

[18] R. Pitchumani, A. Hospodor, A. Amer, Y. Kang, E. L. Miller, and D. D. E. Long, "Emulating a shingled write disk," in *Proc. 20th IEEE Int. Symp. Model., Anal., Simul. Comput. Telecommun. Syst.*, Aug. 2012, pp. 339–346.

[19] R. L. White, R. M. H. New, and R. F. W. Pease,"Patterned media: A viable route to 50 Gbit/in^2 and Up for magnetic recording?," *IEEE Trans. Magn.*, vol. 33, no. 1, pp. 990–995, Jan. 1997

[20] K. Miura, E. Yamamoto, and H. Muraoka, "Estimation of maximum track density in shingled writing," *IEEE Trans. Magn.*, vol. 45, no. 10, pp. 3722–3725, Oct. 2009.

[21] R. W. Cross and M. Montemorra, "Drive based recording analyses at $>$ 800Gb/in^2 using shingled recording," in *Proc. 9th Perpendicular Magn. Recording Conf.*, 2010, vol. 324, no. 3, pp. 330–335.

[22] N. Jeremic, G. Mühl, A. Busse and J. Richling, "The pitfalls of deploying solid-state drive RAIDs," in *Proc. 4th Annu. Int. Syst. Storage Conf.*, May, 2011, p. 14.

[23] K. Ozaki, Y. Okamoto, Y. Nakamura, H. Osava, and H. Muraoka, "ITI canceller for reading shingled-recorded tracks," in *Proc. 9th Perpendicular Magn. Recording Conf.*, 2011, pp. 83–87.

[24] A. Kopytov. (2004). SysBench, a system performance benchmark. [Online]. Available: http://sysbench.sourceforge.net/

[25] D. Llanos, "TPCC-UVa: An open-source TPC-C implementation for global performance measurement of computer systems," *ACM SIGMOD Rec.*, vol. 35, p. 15, 2006.
[26] (1997). Postmark. A new file system benchmark. [Online]. Available: http://www.netapp.com/tech_library/3022.html
[27] Financial1.spc and Financial2.spc.[Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage, 2002.
[28] MSR Cambridge Traces.[Online]. Available: http://iotta.snia.org/traces/388, 2008.

Dan Luo received the BS degree at Hunan Normal University, Changsha, China, in 2008, the MS degree at the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2011. He is currently working towards the PhD degree in the Computer Science Department at the Huazhong University of Science and Techology. Her research interests include computer architecture, hybrid storage system, and NVRAM storage.

Jiguang Wan received the BS degree in computer science from Zhengzhou University, China, in 1996, and the MS and PhD degrees in computer science from the Huazhong University of Science and Technology, China, in 2003 and 2007, respectively. He is currently an associate professor at the Wuhan National Laboratory for Optoelectronics, the Huazhong University of Science and Technology, China. His research interests include computer architecture, network storage system, embedded system, and parallel and distributed system.

Yifeng Zhu received the BS degree from the Huazhong University of Science and Technology, Wuhan, China, in 1998, and the MS and PhD degrees from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is an associate professor at the University of Maine. His research interests include parallel I/O storage systems, and energy-aware memory systems. He served as the program committee of international conferences, including ICDCS and ICPP. He received the Best Paper Award at IEEE CLUSTER 07. He is a member of the ACM, the IEEE, and the Francis Crowe Society.

Nannan Zhao received the BS degree at the Qilu University of technology, China, in 2009, the MS degree at Wuhan university, China, in 2011. She is currently working towards the PhD degree in the Computer Science Department at the Huazhong University of Science and Techology. Her research interests include computer architecture networked storage system, parallel and distributed system.

Feng Li is currently a undergraduate student in the Huazhong University of Science and Technology. He is now studying for the BS degree in the Computer Science Department. His research interests include computer architecture, hybrid Storage and NVRAM storage.

Changsheng Xie received the BS and MS degrees in computer science from the Huazhong University of Science and Technology (HUST), China, in 1982 and 1988, respectively. Presently, he is a professor in the Department of Computer Engineering at the Huazhong University of Science and Technology. He is also the director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, disk I/O system, networked data storage system, and digital media technology. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China.