DVS: Dynamic Variable-width Striping RAID for Shingled Write Disks

Dan Luo, Ting Yao, Xiaoyang Qu, Jiguang Wan*, and Changsheng Xie
Wuhan National Laboratory for Optoelectronics, Department of Computer Science and Technology,
Huazhong University of Science and Technology,
Wuhan City, 430074 China
{jgwan,cs_xie}@mail.hust.edu.cn, luodan860514@gmail.com

***Abstract*—Disk data density improvement will eventually be limited by the super-paramagnetic effect for perpendicular magnetic recording. Of the various new technologies being explored, Shingled Magnetic Recording (SMR) exposes as the most promising one to achieve high areal density increase and little changes to the manufacturing process. At present, high-capacity Shingled Write Disks (SWDs) are available from Seagate and HGST. Since SMR is leading next generation disk technology and more and more SWDs will be used in storage systems, there is a great need to look over the current RAID storage techniques based on HDDs again. In this paper, we proposed a dynamic variable-width striping RAID (DVS-RAID) for SWDs to reduce the parity updating cost. DVS-RAID never overwrites the old data, always constructs a new full or partial stripe (variable-width stripe), and writes to the SWDs through appending. In addition, taking the access characteristics of SWDs into consideration, we present a new write cache management that exploits both spatial and temporal localities. The experiment with six real-world traces demonstrates that DVS-RAID exhibits a slightly lower performance than HDD-based RAID under write-intensive workloads with frequent updates. For some read-dominated or sequential write-dominant workloads, or workloads with infrequent updates, DVS-RAID even shows a better performance than HDD-based RAID.**

I. Introduction

Magnetic hard disk drives have spanned nearly a 60-year product history and this technology continues to be a principal means of storing data in most computer systems today. This is the result of many attractive attributes of HDDs including areal density on the disk media as well as the price per gigabyte. However, the currently employed Perpendicular Magnetic Recording (PMR) is fast approaching its density limit and the industry is eager for the introduction of new technology to overcome the limit [6], [10].

Shingled Magnetic Recording (SMR) is a new HDD recording technology designed to increase density beyond the limits of traditional PMR. In the simplest terms, SMR is a new hard drive technology that allows the tracks on a platter to be layered on top of each other, just like roof shingles on a house, to increase platter density or tracks per inch (TPI). SMR requires minimal manufacturing changes because it retains the use of existing disk head and media technology. This is why SMR is the first technology to reach market: 5TB and 8TB drives are available from Seagate and 10TB drives are available from HGST, while other technologies remain in the research stage, such as Heat-Assisted Magnetic Recording [7], Microwave-Assisted Magnetic Recording [8] and Bit-Patterned Media [9].

SMR increases the areal density by overlapping the neighboring tracks which results in destructive random writes and in-place updates [4]. A fundamental limitation of shingle write disks is their inability to carry out simple update-in-place write operations. That is to say, data has to be written sequentially onto the tracks in order not to destroy the valid data on the subsequent tracks. Alternatively, we have to safely read the impacted valid data in the subsequent tracks out first before writing/updating to the current track and then write those impacted valid data back afterwards. In this way, extra read and write operations are incurred as an extra cost, which is known as the write amplification problem.

The challenge of designing a storage device that employs shingled recording is then to allow an unrestricted write access from the hosts perspective, despite the restriction on the physical write process. Generally, there are two types of SWDs: the drive-managed SWDs and the host-managed/host-aware SWDs. Drive-managed SWDs provide block interface to the upper level applications such as file systems and databases. On the other hand, host-managed/host-aware SWDs are simply raw devices and rely on specific upper level applications to interact with the PBAs directly.

All these new high-capacity SMR disks are designed for cloud datacentres and for cold storage applications (archives). Although SSD prices have been falling drastically in the last few years, compared to hybrid arrays, all-flash arrays are still very expensive. A hybrid storage array uses a combination of spinning disk drives and flash SSDs. Hybrid storage takes advantage of the pros of each type of storage and uses those to minimize the disadvantages of each. A hybrid array typically uses hard disk drive-based storage arrays for storing the majority of the cold data and solid state drives for storing and accessing the most commonly used data.

Since SMR is leading next generation disk technology and more and more SWDs will be widely used in hybrid storage system, there is a great need to look over the current storage techniques based on HDDs again. Because the characteristics, limitations and advantages of SWD are sure to impact the current techniques based on HDDs. RAID system is the one storage technique we investigate here. A RAID5 disk array utilizes a large number of commodity inexpensive HDDs in

978-1-5090-3315-7/16/$31.00 ©2016 IEEE

parallel to achieve higher performance as well as incorporating parity drives to obtain higher reliability with low storage cost. RAID5 which supports concurrent access of small blocks is currently regarded as one of the most promising approaches for providing highly reliable low cost secondary storage systems. Therefore, in this paper, we mainly investigated the challenges and designs of integrating SWD into RAID5 system. Furthermore, we study what impact the introduction of SWDs has on the design of cache for SWD-based RAID system.

We propose a dynamic variable-width striping RAID for SWDs to reduce the parity updating cost. Our contributions are summarized as follows:

- We propose DVS-RAID for SWDs. It dynamically generates a new full or partial stripe and appends new data to the tail of the existing data.
- A write cache management that considers the properties of SWD is designed for DVS-RAID.
- We implement a DVS-RAID simulator and evaluate it under different kinds of workloads. The results show that DVS-RAID shows a better performance than HDD-based RAID under read-dominant or sequential write-dominant workloads.

The rest of the paper is organized as follows. Section II gives the background of segment-based data layout. We present our DVS design in section III and the write cache management in section IV. We describe the experimental setup in section V and evaluate the design in section VI. Related work is presented in section VII, and the conclusions are given in section VIII.

II. Background

Shingled writing takes advantage of the fact that the magnetic field required for a read is smaller than that required for a write. Shingled writing leverages this property by overlapping the currently written track with the previous track, leaving only a relatively small strip of the previous write track untouched. The remaining track is therefor narrower than when it was originally written, but remains readable. In this manner, tracks are ultimately placed closer together, resulting in the capacity gain. The write head is wider than a single track, meaning when data is written to an SMR hard drive, data must be written sequentially so the writer does not destroy data on the overlapping tracks. In other words, while SMR can deliver the much needed capacity gains, it dose so by sacrificing the random write capabilities of the device.

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 64MB, let alone multi-GB sized regions.

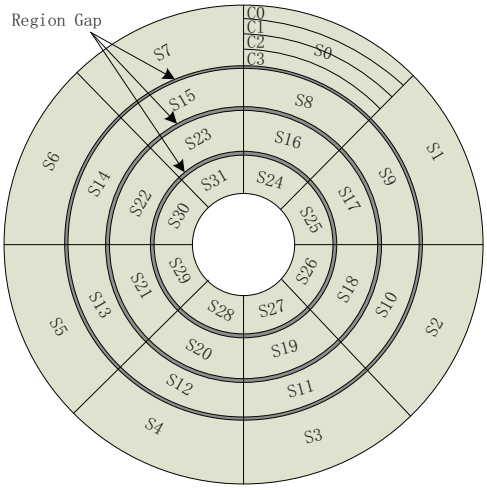


Fig. 1. Segment-based data layout

A. Segment-based data layout

In HWSR [3], we proposed a segment-based data layout management to reduce the write amplification of SWDs.

Figure 1 shows our proposed segment-based data layout management. 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 direction. Each segment has a unique Logical Segment Number (LSN). In each segment, the sectors in the same track constitute a logical data chunk. Figure 1 shows an example of four regions, with eight segments in each region. As shown in the Fig.1, each region is divided into eight segments, each segment (S0-S32) has four data chunks (C0, C1, C2, and C3), and each chunks is composed of several consecutive sectors in the same track. Within a region, the logical addresses of two adjacent segment are consecutive. Within a segment, the logical addresses of two chunks on the adjacent track are consecutive. When writing sequential data chunks to a segment, the data chunks are laid out in the radical direction. Thus, if data in segment S1 is updated, its neighbor segments S2 and S0 are not affected.

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.

III. DVS-RAID for segment-based SWDs

In this section, we first briefly discuss the problem of traditional RAID5 for SWDs. We then describe our proposed DVS-RAID for segment-based SWDs. In this section, we first briefly discuss the problem of traditional RAID-5 for SWDs. We then describe our proposed DVS-RAID for segment-based SWDs. A DVS-RAID is a hybrid storage array that employs high-performance SSD for read caching and write buffering. In section IV, we will elaborate on SSD cache management algorithm for DVS-RAID. DVS-RAID has three

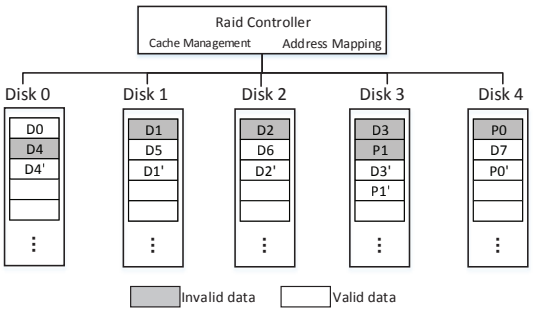


Fig. 2. Traditional RAID5 for SWDs

major functions, namely data organization, maintenance of address mapping table, and garbage collection. We will discuss them in details in the following section.

A. Traditional RAID5 for SWDs

Figure 2 shows the problem of traditional RAID5 for SWDs. At the beginning, data chunks D0-D3 and parity P0 comprise stripe 0 and data chunks D4-D7 and parity P1 comprise stripe 1. Then we assume that D1-D4 are updated. In traditional HDD-based RAID5, this is a simple task as updated data will simply overwrite the old data. However, since overwrites are not possible in shingled write disks, the updated data is written to a new location. Meanwhile, the parity data needs to be updated through either read-modify-write or reconstruct-write. Finally, the old data must be invalidated and the stripe information is updated to reflect the changes of the corresponding stripe.

There are limitations to this approach. First, whether read-modify-write or reconstruct-write is employed, the old data must be read before calculating the new parity. This is also true for traditional HDD-based RAID5 systems that require four disk accesses to write a data chunk. Second, the small write problem is aggravated severely by the inherent property of SWDs that can carry out simple update-in-place write operations, resulting in tremendous write amplification. Third, when writes are not updates to existing data but writes of totally new data, the data cannot be written until the stripe becomes full, leaving open a window of vulnerability. For example, in Figure 2, if new data chunks D8 and D9 arrive, a parity chunk cannot be calculated for these pages that form a partial stripe, and thus these chunks cannot be written until a full stripe is formed, that is, another two data chunk arrive.

B. DVS-RAID

1) Data organization: We propose, **D**ynamic **V**ariable-width **S**triping RAID (DVS-RAID or simply DVS) for segment-based SWDs to reduce the parity updating cost. DVS divides the whole space of the disk array into Segment-based Stripe Groups (SSGs). A SSG is composed of physical chunks that comprise a stripe, which means that these physical chunks are all of the same segment number. For example, as shown in Figure 3, segment S0 of all SWDs comprise SSG0, segment S1 of all SWDs comprise SSG1, and so on.

The key feature of the scheme is that, within a SSG, DVS dynamically constructs a new full or partial stripe as need

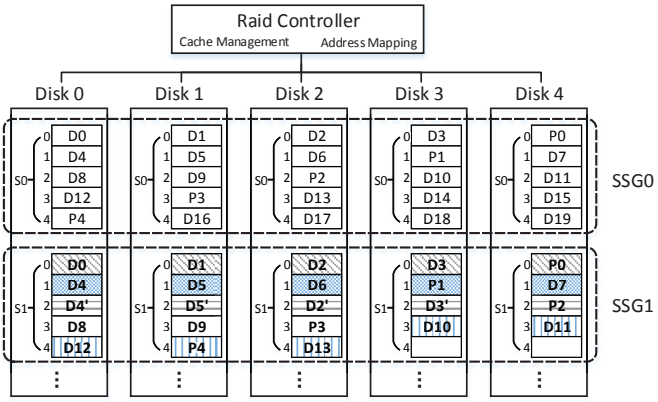


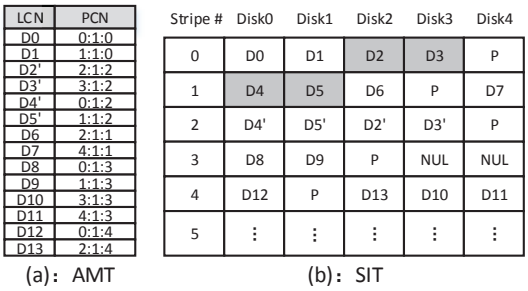
Fig. 3. Data organization in DVS-RAID

be. The new stripe is composed of the newly written or updated chunks from different SWDs. The data chunks of the new stripe are written to the tail of their respective segment through appending. The stripe size of the new stripe may vary (up to the full stripe size) and not be fixed. That is to say, whenever the data chunks of the same segment number (i.e. those data chunks belong to the same SSG) are destaged from storage controller's write buffer to SWDs, DVS generates a new stripe no matter whether the data chunks are newly written or updated. DVS never overwrites the old data and always creates a new stripe.

Figure 3 shows how data chunks are organized in DVS-RAID for segment-based SWDs. SSG0 in Fig. 3 shows a data mapping like a traditional RAID5 with a total of 5 disks, where Di represents data block *i* and Pi represents parity block *i*. P0 contains the parity of data blocks D0, D1, D2, and D3.

Now let us take the example given in SSG1 to illustrate the rationale behind DVS-RAID. At the beginning, with valid user data, D0-D7, DVS generates two new stripe 0 and 1 and stores them like in SSG0. And then, we assume that data chunks D2-D5 are modified. With DVS, the controller dynamically constructs a new stripe with updated data D2'-D5', simply calculates the new parity for these data, and writes them at the tail of segment S1 of their respective SWDs along with the parity value. After writing, the controller simply marks the old data as obsolete. There is no need to read the old data. The old data is implicit redundant data that can be used to data recovery in case of SWDs failures.

Then, we consider the case where only part of the stripe is written/updated. Continuing the previous example, assume that least recently accessed data D8 and D9 are being destaged. At this point, DVS has three choices. First, DVS may choose to wait for data to form a full stripe. In this case, many least frequently accessed data, such as D8 and D9, would reside in the cache for very long time, which pollutes the buffer cache. Second, DVS may choose to read the old data D10 and D11 from the disk. And then DVS generates a new stripe and write the stripe to disk. However, this method incurs for four extra disk accesses - two to read D8 and D9 and two to



LCN	PCN
D0	0:1:0
D1	1:1:0
D2'	2:1:2
D3'	3:1:2
D4'	0:1:2
D5'	1:1:2
D6	2:1:1
D7	4:1:1
D8	0:1:3
D9	1:1:3
D10	3:1:3
D11	4:1:3
D12	0:1:4
D13	2:1:4

(a): AMT

Stripe #	Disk0	Disk1	Disk2	Disk3	Disk4
0	D0	D1	D2	D3	P
1	D4	D5	D6	P	D7
2	D4'	D5'	D2'	D3'	P
3	D8	D9	P	NUL	NUL
4	D12	P	D13	D10	D11
5	⋮	⋮	⋮	⋮	⋮

(b): SIT

Fig. 4. Maintenance of metadata in DVS-RAID

write them. Therefor, DVS chooses the third method. That is, DVS chooses to write the data as-is forming a partial stripe as shown in Fig. 3.

At last, we assume that D10-D13 are destaged to SWDs array. DVS just simply constructs a new stripe and writes the data chunks of the new stripe sequentially to the tail of segment S1. D10 and D11 have been allocated to chunk index 3 of segment S1 in disk 3 and 4, respectively, while D12, P4 and D13 has been written to chunk index 4 of segment S1 in disk 0, 1, and 2, respectively. As we see in Fig. 3, data chunks are always sequentially written to the segment, which conforms to the access characteristics of SWDs.

2) Maintenance of metadata: Figure 4(a) and Figure 4(b) respectively show the corresponding Address Mapping Table (AMT) and Stripe Information Table (SIT) of SSG1 as shown in Fig. 3. AMT is used to translate the logical address to physical address. SIT records the stripe information of SSG that is used for garbage collection and data recovery. Every SSG has an Address Mapping Table and Stripe Information Table. A Global Translation Table (GTT) is used to record the physical address of all the AMT and SIT. The Global Translation Table is small and entirely resident in the cache.

In a DVS-RAID with a total of N+1 disks, when a request is received with the LBA of requested data, we can calculate its LSN and Logical Chunk Number (LCN) by using Eqs. 1 and 2. Using LSN as the index, we can obtain the SSG's AMT by looking up the GTT. Then, we can use the LCN as the index to look up the AMT and thus get the Physical Chunk Number (PCN) of the request. As shown in Fig. 4, a PCN is broken into three components. For PCN (P1:P2:P3), P1 represents the Disk ID, P2 represent the Logical Segment Number, and P3 represent the actual chunk index within segment P2.

$$LSN = LBA/(S_{seg} * N) \quad (1)$$

$$LCN = (LBA \bmod (S_{seg} * N))/S_{chk} \quad (2)$$

where S_{seg} and S_{chk} represent the segment size of the segment-based SWD and the chunk size of DVS-RAID, respectively.

In terms of managing the AMT and SIT, typically, the RAID controller would load these tables partially or in their entirety onto SSD cache during boot-up. When new data is written, the two tables are updated to reflect this change. To maintain consistency, the two tables are periodically destaged to disk array. DVS-RAID can store the address mapping table and stripe information table at fixed location, such as the middle tracks of the disks. Because these metadata information is frequently updated, a good choice is to put the two tables in the Random Access Zone (RAZ).

3) Garbage collection in DVS: Because out-of-place-update method is adopted in DVS-RAID, free stripes are guaranteed to exist over which a newly generated stripe can be stored. If there is no space to write to, a cleaning process, called garbage collection (GC), is invoked to reclaim free space. In DVS-RAID, garbage collection process is performed by the unit of a Segment-based Stripe Group.

There are two different modes of garbage collection in DVS, namely background and foreground garbage collection. Foreground GC is always triggered by a write operation to the SSG. For example, when there are data D14-D17 to be written to SSG1 in Fig. 3, a garbage collection must be conducted to make free space. On the other hand, background GC is usually kick-started by a pre-set idle period and by calculating the cost of garbage collection at that time. And the background GC will be aborted by any foreground read or write request. The main advantage of background garbage collection is the gain in write performance.

The generic procedures performed in background GC of DVS are as follows. Firstly, a SSG is chosen by the DVS as the victim SSG. Generally, the SSG with the smallest number of valid data chunks, the frequently accessed SSG, or the recently accessed SSG will be chosen as the victim SSG. Currently, DVS chooses the SSG containing the largest amount of invalid data as the victim SSG. Secondly, valid data chunks are taken from the selected SSG into the SSD write buffer, and then DVS combines these data chunks with the cached data in the SSD of the selected SSG. Thirdly, all the valid data chunks are written to the selected SSG as described in section III-B1. DVS always constructs a new full or partial stripe with the valid data chunks. The old parity chunks are simply discarded and new parity chunks are calculated. Meanwhile, the AMT and SIT of the selected SSG is updated correspondingly to reflect the changes.

Foreground GC is performed only when the free space will be immediately needed in a SSG. Therefore, foreground GC do not need to select a victim SSG, it just executes the last two steps of the background GC process.

IV. Cache management for DVS-RAID

Read cache management is a well studied discipline and there are a large number of cache replacement algorithms in this context, see, for example, LRU, CLOCK, LRU-2, LRFU, LIRS, MQ, ARC, CAR and SARC. In contrast, write caching is a relatively less developed subject. In this section, we shall focus on algorithms for write cache management in the context of a storage controller equipped with fast, non-volatile storage, such as SSDs. Meanwhile, we take the inherent property of SWD into consideration when designing the algorithms.

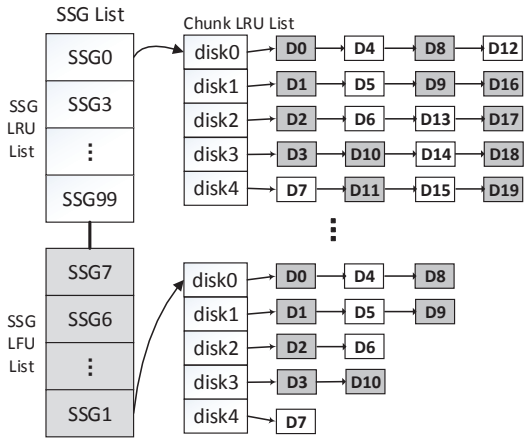


Fig. 5. Write Cache Management

In DVS-RAID, we will manage cache in terms of Segment-based Stripe Groups, which allows a better exploitation of temporal locality by saving seeks and also spatial locality by coalescing writes together. As shown in Fig. 5, each entry of the SSG list represents a segment-based stripe group and has a LRU pointer array. The size of the pointer array equals to the number of disks in the DVS-RAID. We group chunks by their logical segment number and chunks with the same logical segment number are grouped together. For example, chunks with LSN S0 are grouped into the SSG0 entry. In each stripe group entry, chunks belonging to the same disk are maintained through a linked list. In Fig. 5, each SSG entry has 5 Chunk LRU list.

The basic idea of our proposed scheme is to divide the whole SSG list into the SSG LRU list and the SSG LFU list. In the whole SSG list, each entry has a reference count filed to record the access count of the stripe group and a recency bit to indicate whether the stripe group is in the SSG LRU list. The SSG LRU list consists of recently accessed stripe groups, while the SSG LFU list consists of stripe groups that are candidates for eviction. The algorithm now proceeds as follow.

When a chunk is accessed for the first time, a new SSG entry is allocated if its SSG entry dose not exist in the cache. Then the chunk is inserted into the corresponding Chunk LRU list and the newly created SSG entry is inserted into the SSG LRU list. And whenever the chunk in the SSG list (including the LRU list and the LFU list) is accessed, its SSG entry is moved to the MRU position of the SSG LRU list. At the same time, the recency bit of the accessed SGG entry is set to one and the reference count is incremented. When the SSG LRU list is full and a new SSG entry is created (or the SGG entry in the LFU list is accessed), the SSG entry in the LRU position of the list is evicted and inserted into the LFU SSG list, and its recency bit is set to zero.

The LFU SSG list is sorted by the reference count. The evicted entry from the LRU list is always inserted in its correct sorted position. When the cache occupancy reaches the High Threshold, the destage process starts. The SSG entry with the lowest reference count in the LFU list is selected as a victim. In Fig. 5, SSG1 will be selected as the victim. Chunks D0-D3 and D4-D7 constitute two new full stripe, D8-D10 constitute a new partial stripe, and the newly generated stripes will be destaged to SWDs. In practice, the size of chunk LRU list in the victim SSG would be large. Sometimes, only the few chunks in the LRU position of the chunk LRU list would be destaged. One problem with the use of reference count is that certain entry may build up high reference count and never be replaced. To solve this problem, we periodically set every reference count, C, to $\lceil C/2 \rceil$ at fixed interval.

In our proposed write cache management, the LRU list always keeps the recently accessed SSGs in the cache and most of cache hits are generated in this list, which represents temporal locality. Meanwhile, the LRU list avoids destaging the newly created SSG when the destage process starts. If we decided to evict a SSG only depending on reference count, a newly created SSG would be evicted out of the cache, because the reference count of the newly created SSG must be low. As the address space of an SSG is large, the disk head only need to move a small distance between consecutive destages representing spatial locality. Based on above analysis, we incorporate temporal locality, spatial locality, and access frequency into our write cache management. Another advantage is that the algorithm is simple and easy to implement.

V. Experimental Setup

In this section, we describe the design of experimental system, the basic hardware setup, and the workload characteristics.

A. Design of experimental system

A schematic diagram of the experimental system is depicted in Fig. 6. The RAID controller simulator uses the disk I/O traces as an input. For each request recorded in the traces, a corresponding I/O request is issued to the simulator. We evaluate the performance of DVS on raw banded SMR disks without any interference from a drive-managed SWDs's internal remappings. We evaluate the performance of DVS using regular hard disks, by banding them like SMR disks. In the experiment, DVS splits the available LBA range into fixed-sized bands, and reads/writes to the bands with SMR like restrictions, emulating how one would read/write to a SMR disk. We open the device file with O_DIRECT and O_SYNC flags to bypass the cache effect of file system. Thus the simulator can measure the actual response time for every read/write request. For simplicity, we assume that the hard disk has a fixed number of sectors per track, and the capacity of each track is 8×1024 sectors (512 bytes in one sector).

B. The basic hardware setup

The experiments for the evaluation were run in a host with a 2.13GHz Intel® Core™ Quad CPU and a 8GB RAM. Our prototype system consists of five 1TB Western Digital® 7200 SATA hard drives, a 120GB Intel 320 series SSD, and

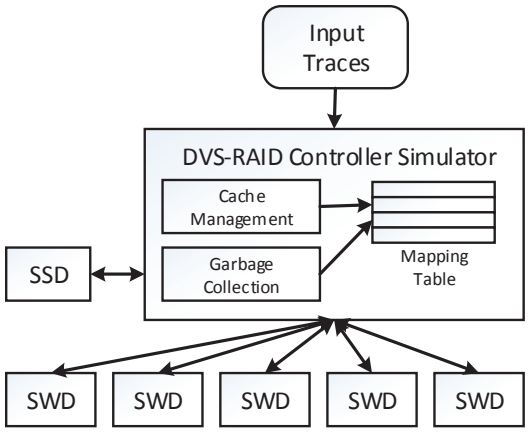


Fig. 6. The experimental system

a small portion of main memory. In order to minimize the interference, the operating system and home directory are stored in a separate hard disk drive. Table I lists the detailed description of the experimental setups.

Unless otherwise stated, the sample parameters are now provided as follow. A region consists of 50 contiguous tracks on the same surface, i.e., a segment has 50 chunks. The chunks size is set to 128 sectors.

C. Workload characteristics

In order to fairly evaluate the performance of DVS-RAID, we use real world I/O workloads that have meaningful contents as well as access patterns similar to real applications. Six real-world traces are utilized for performance evaluations. The first four traces are collected from enterprise servers at Microsoft Research Cambridge [27]. The last two traces are collected from OLPT applications [26]. Table II summarizes the characteristics of these traces.

For performance comparison purpose, we have evaluated four different data organizations on the same testing environment:

1) RAID5 for HDD (HDD RAID5): To establish a baseline, we analyzed the performance of traditional HDD-based RAID5.
2) RAID5 for Sblock-based SWD (Sblock RAID5): We composed a traditional RAID5 storage system that is made up of Sblock Architecture-based SWDs [13].
3) RAID5 for Segment-based SWD (Segment RAID5): We built a traditional RAID5 storage system that is made up of Segment-based SWDs [3].
4) DVS-RAID for Segment-based SWD (DVS-RIAD5): This is our proposal.

VI. Evaluation results

In this section, we demonstrate experimental results and comparative analyse.

TABLE I
Experimental Setups

OS	Linux version 2.6.35.6-45.fc14.x86_64	
CPU	Intel(R) Xeon(R) CPU E5506 @ 2.13GHz	
Memory	Hynix DDR3 4GB 2R*4 PC3-10600R	
Hard disk	WD 1TB SATA/64MB Cache 3Gb/s 7200rpm	
SSD	Intel SSDSA2CW120G3 3Gb/s SATA 120G	
Parameters	Emulated shingled disk	1TB
	Block Size	128 Sectors
	Region Size	32 Tracks

TABLE II
Characteristics of traces

Traces Name	Total Requests	Unique Data Size	Avg. Read Len	Avg. Write Len	# of Writes	# of Updates	Write Percent
usr0	2,000,000	2.41GB	41848B	10559B	1,200,145	1,190,240	60.00%
web0	2,000,000	7.26GB	30778B	8828B	1,395,007	1,383,358	69.75%
prxy0	2,000,000	0.34GB	6296B	2444B	1,930,570	1,913,646	96.53%
mds0	1,211,034	3.09GB	24278B	7411B	1,067,061	1,055,832	88.11%
Financial1	2,000,000	0.45GB	2569B	4082B	1,568,985	1,529,403	78.45%
Financial2	2,000,000	0.33GB	2228B	3033B	352,719	346,169	17.64%

A. Overall performance

Figure 7 shows the average response time for the various RAID schemes under different traces. In the Fig.7, the *x*-axis denotes all the evaluated schemes per workload, while the *y*-axis represents the average response time in milliseconds. The read cache size and write cache size are both set to the 10% of the working size (the number of blocks being accessed during execution). Generally speaking, among the four schemes, traditional HDD-based RAID5 has the best performance for most workloads as hard disks have not the SWDs' disadvantage of inability to serve update in-place directly. However, DVS also provided superior performance and exhibited only a slightly lower performance than HDD-based RAID5. Even for some read-dominated workloads or sequential write-dominant workloads, DVS shows better performance to HDD-based RAID5. Take *mds0* for example, the average response time is 1.33ms, 0.99ms, 1.08ms, and 1.04ms for HDD-based RAID5, DVS, Segment RAID5 and Sblock RAID5, respectively.

As is known to all, traditional RAID5 suffers a performance penalty during data updates, especially for small write-dominated workloads. Among all these traces, *financial1* and *prxy0* workloads feature intensive small random data accesses. Although data in SWDs cannot be updated freely in place without overwriting the valid data in subsequent tracks if any, the average response time is only slightly higher than HDD-base RAID5. The following reasons can explain why DVS can provide satisfactory performance. First, when data chunks are evicted from the write cache, DVS always construct a new full or partial stripe and write data chunks to the tail of their respective segment through appending. However, traditional RAID5 has to read the old data for calculating the new parity and write the new data to the disk along with the new parity. Second, benefitting from segment-base data

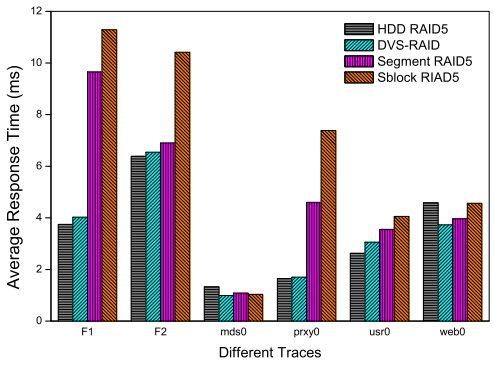


Fig. 7. Average response time under different traces

layout management, DVS reduces the write amplification to the size of the segment that is much smaller than a region, which reduces the write amplification effectively. Third, within a segment, data chunks are always written to the tail of the segment. Compared to HWSR [3] that adopts in-place update method, DVS reduces the number of segment defrag operations, which improves system performance substantially. Lastly, specially-designed cache management is used to solve the problem of what to destage. The write cache management not only can keep the hot data in the cache, but also can decrease the number of GC operations.

As shown in Fig. 7, the average response time of DVS, Segment RAID5 and Sblock RAID5 is all lower than traditional HDD-base RAID under *mds0* workload. This result is consistent with our institution. *Mds0* trace is collected on media servers by Microsoft Research Cambridge. *Mds0* workload is a write-intensive trace with large number of sequential write requests. Another characteristics of media server workloads is that the data in media server is not updated almost. The read/write characteristics of SWDs is extreme suitable for this kind of workloads. This is why SWD-based RAID5 schemes perform a little better than HDD-base RAID5 under *mds0* trace.

For *web0* trace, DVS even provided a little better performance than HDD-based RAID5. *Web0* trace is obtained from a web service server and shows a write intensive access pattern. And there are a larger number of sequential read requests. More importantly, data chunks in web service server are almost immutable unless the data is deleted. In other words, once the data has been written, it will not be updated. The reason why DVS perform a little better than HDD-base RAID5 for *web0* trace is similar to the reason for *mds0* trace.

Now, let us consider the read-dominated workloads. As demonstrated in Fig. 7, DVS showed superb performance for *Financial2* trace, which is comparable to HDD-base RAID5. Although *Financial2* trace is featured with intensive small random data accesses, the read latency accounts for most proportions of the total latency. This is also the reason why SWD-base RAID5 schemes show comparable performance to HDD-base RAID5.

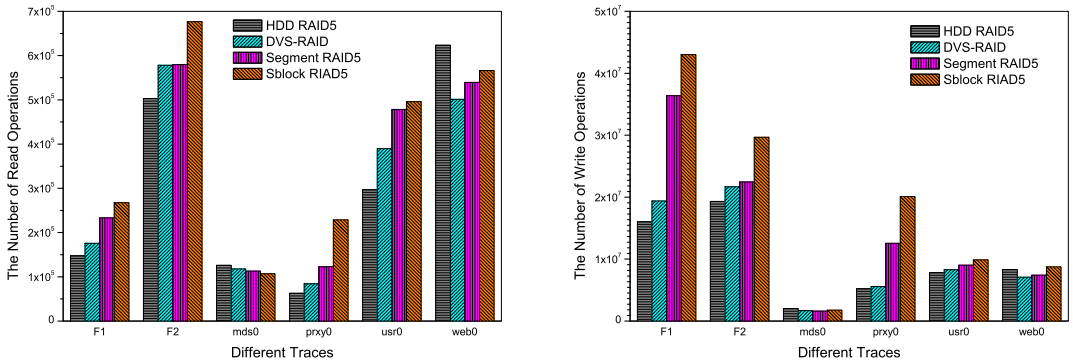
In Fig. 7, we can clearly observe that the average response time of DVS is much lower than Segment RAID5 and Sblock RAID5, except that they show similar performance for *mds0* trace. Like HWSR, the SWDs in Segment RAID5 adopt in-place update method. That is to say, no matter which chunk is updated in a segment, the whole segment of the SWDs in Segment RAID5 must be rewritten. As a result, a lot of time is spent in disk seeking and rotating, which degrades the performance of Segment RAID5 severely. At the same time, Segment RAID5 also faces the small-write problem like traditional HDD-based RAID5 that aggravates the first problem seriously. This is why Segment RAID5 does not perform well. Among the four schemes, Sblock RAID5 has the highest average response time. Sblock RAID5 has such a high average response time is mainly because of high overheads incurring by garbage collection. The garbage collection of Sblock architecture moves tremendous data between the head and the tail of Sblock circular buffer, which degrades Sblock RAID5 performance severely. Meanwhile, Sblock RAID5 also has the small-write problem. Due to segment-based data layout management, the write amplification of Segment RAID5 is reduced greatly, compared with Sblock RAID5. Therefor, Segment RAID5 provides better performance than Sblock RAID5 for most traces.

From above discussions, we see that DVS-RIAD can provide satisfactory performance under various different traces. For some read-dominated workloads, sequential workloads, or workloads with infrequent updates, DVS shows a better performance than traditional HDD-based RAID5. Even for write-intensive workloads with frequent updates, DVS provides only a slightly lower performance than HDD-based RAID5. Under almost all traces, DVS provides much better performance than Segment RAID5 and Sblock RAID5.

B. Disk read/write operations

Figure 8 shows the actual number of read and write operations that is issued to the hard disks or shingled write disks. Note that the *y*-axis scale of the two plot is different. The number of read/write operations reflects the storage system performance from the side. The storage system performance becomes worse as the number of read/write operations become larger. As shown in the Fig. 8, the number of read/write operations for *mds0* is smallest. Correspondingly, the four storage schemes all show the best performance for *mds0* presented in Fig. 7, compared to the other five traces.

From Fig. 8, we can discover two observations. The first observation is that the number of write operations is about two order of magnitude larger than the number of read operations. Two reasons can explain it. First, most traces are write-intensive traces with large number of random write requests. In other words, the number of write requests is larger than that of read requests. Second, the write amplification problem caused by small write problem of RAID5 and inability to directly serve update in-place increase the number of write operation tremendously. Another observation is that the number of write operations is extremely large for workloads with frequent



(a) The number of Read operations

(b) The number of Write operations

Fig. 8. The number of Read/Write operations under different traces

updates. *Financial2* workload is a good example for this situation. Although *Financial2* is a read-intensive workload, the number of write operations is extremely large because of the existence of the write amplification problem.

C. The impact of cache size

Figure 9 shows the average response time of various cache size over *Financial1* trace. In order to study the impact of cache size on storage performance, we conduct experiments with six configurations of write cache size, namely 5%, 10%, 20%, 30%, 40%, 50% of the working-set size. In all experiments, the read cache size equals to the size of write cache. Generally speaking, all the four schemes exhibit a better performance as the cache size increases. HDD-based RAID5 and DVS both show stable performance on different cache size and DVS provides comparable performance to HDD-based RAID5. However, Segment RAID5 and Sblock RAID5 is more sensitive to the size of cache than HDD-based RAID5 and DVS.

The stable performance of DVS is attributed to the excellent write cache management described in section IV. A good write caching algorithm has to solve two problems regarding destaging: the *destage order* and the *destage rate*. The destage order deals with leveraging temporal and spatial localities, while the destage rate deals with guaranteeing free space and destaging at a smooth rate. Our proposed write caching algorithm not only considers spatial and temporal locality, but also take the access frequency into consideration, which solve the destage order problem efficiently. The destage rate problem has been well studied in [19].

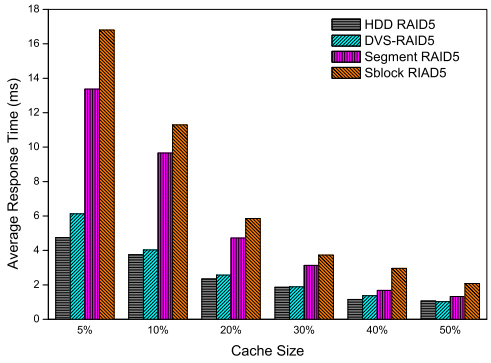


Fig. 9. The impact of cache size

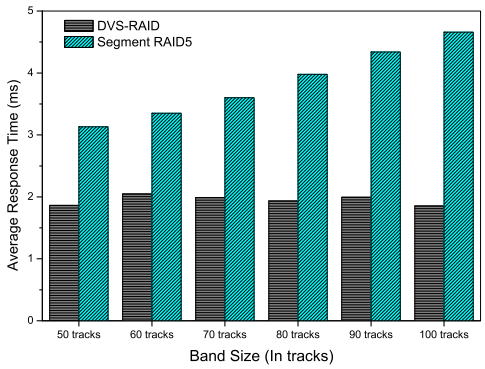


Fig. 10. The impact of band size

D. The impact of band size

Figure 10 shows the average response time of various band size over *Financial1* trace. Our pervious study [25] have demonstrated that the average response time of segment-based SWDs increases monotonically as the band size increases. The performance of Segment RAID5 also becomes worse as the band size increases. However, increasing band size does not necessarily degrade the storage performance of DVS-RAID that is composed of segment-based SWDs. In DVS, data chunks are always written to the tail of the segment through appending, unlike HWSR that adopts in-place update method. This is the primary reason why the performance of DVS does not degrades proportionally as the band increases.

Moreover, as the band size increases, the segment size become larger. As a result, when the SWD space utilization is low, there is more free space for each segment. That

is to say, relative to small segment size, a larger segment can provide more free space for subsequent writes, which postpones the passive garbage collection. Therefor, when the space utilization is low, increasing band size has a positive effect on system performance.

As the band size increases, the latency for GC operation increases proportionally, because the amount of data that need to be read/write during GC becomes large. And more free space in the write cache have to be available. So, the write caching algorithm has to maintain a steady and reasonable amount of free space and manage the destage rate well.

VII. Related Work

A. RAID Technique

Several techniques have been proposed to reduce the overhead for small writes in a HDD-based RAID5. A parity logging approach [22] was proposed to convert a large number of small random writes to parity blocks on disks to a large sequential writes, similar to the idea of log-structured file system. A floating parity scheme [23] was presented that sacrifices disk storage efficiency by relaxing the requirement that modified data parity blocks be written back into their original locations. The floating parity technique remaps dynamically parity blocks within disk cylinders to reduce the rotational latency between reading and writing parity. In both schemes, they attempted to reduce the disk seek time overhead of parity updates, and the old data must be read from the disk to calculate parity. However, DVS-RAID always generates a new partial or full stripe and writes the stripe to the disk array through appending, which complies with the write characteristic of the shingled write disk.

The *fast write* [24] scheme uses a nonvolatile write buffer to reduce write latency. The data in the write buffer and corresponding parity are written at disks in the background. And the parity should be updated at disk when the data is evicted from the write buffer in the fast write scheme. STOW [19] describes a spatially and temporally optimized write caching algorithm that exploits not only temporal and spatial localities, but also manages both the destage rate and destage order effectively in a sigle powerful algorithm. Both schemes do not consider the inherent properties of SMR. However, we take the inherent property of SWD into consideration in our proposed write cache algorithm.

B. SMR Technology

In shingled write disk, writing data to one track will destroy data previously written on the overlapping tracks. The challenge of designing a storage device that employs shingled recording is then to allow an unrestricted write access from the host's perspective, despite the restriction on the physical write process. Two basic strategies exist to handle this constraint effectively. First, we can mask the operational differences of a SWD by introducing a Shingle Translation Layer (STL) as proposed by Gibson and Polte [4], [5]. Another possible solution would be to use a stand-alone SWD with a specialized file system or object store serving as the interface.

In order to improve the performance of SWDs, most existing research works design new data layouts for shingled disks [11]–[14]. In [11], [12], SWD is divided into log access zones(LAZs) and random access zones(RAZs), where LAZs store user data and RAZs store metadata respectively. Cassuto [13] 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. SMRDB [15] is a key-value databese engine for SWDs. SMRDB design and optimize an LSM tree based data layout and management for SMR disks.

Several SMR-specific file systems have been proposed, such as SMRfs [18], SFS [16], and HiSMRfs [17]. SMRfs present a SMR file system for big data application that writes to files only sequentially, never reopen a closed file for a write, and never rewrite a block in the file. SFS is a host-managed design for in-place update SWDs. And SFS is specially design for video recorders/set-top boxes. For other workloads, performance obviously take a bit.

There are some other works on SWDs, which is complimentary to our work. Lin [21] proposed H-SWD to reduce the garbage collection of circular log by using a hot data identification mechanism. Jones [2] proposed using write frequency as a metric to separate blocks in order to reduce data movement during band compaction.

In order to evaluate the performance of SWDs, Pitchumani [20] 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. Skylight [1] was novel and drilled a hole into a SMR drive to understand how Seagate SMR drives work.

VIII. Conclusions

Traditional HDD-based RAID5 has the small write problem. The inherent weakness of shingled magnetic recording is the inability to serve update in-place directly, i.e., writing to a given data track requires rewriting its subsequent tracks. If we used the original RAID5 technique to set up a RAID5 storage system that is composed of shingled write disks, the small write problem would be aggravated severely by the weakness of SMR, resulting in tremendous write amplification. In this paper, we proposed a dynamic variable-width striping RAID for SWDs to reduce the parity updating cost. Meanwhile, we also design a write cache management specially for SWDs. Experimental evaluations under a variety of I/O intensive workloads show that DVS-RAID can provide satisfactory performance and the performance is not sensitive to the cache size. Meanwhile, increasing band size does not necessarily degrade the performance of DVS-RAID. However, we should note that the write caching algorithm has to maintain a steady and reasonable amount of free space and manage the destage rate well as the band size increases.

ACKNOWLEDGMENT

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, and the National Natural Science Foundation of China under Grant No.61432007 and No.61300047.

REFERENCES

[1] A. Aghayev and P. Desnoyers, "Skylight-a window on shingled disk operation, in Proceedings of the 13th USENIX Conference on File and Storage Technologies. USENIX, Feb. 2015, pp. 135C149.

[2] S. N. Jones, A. Amer, E. L. Miller, D. D. E. Long, R. Pitchumani, C. R. Strong, "Classifying data to reduce long term data movement in shingled write disks," in Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1-9.

[3] Jiguang Wan, Nannan Zhao, Yifeng Zhu, Jibin Wang, Yu Mao, Peng Chen, Changsheng Xie, "High Performance and High Capacity Hybrid Shingled-Recording Disk System," in Proceedings of IEEE Cluster, 2012, pp. 173-181.

[4] G. Gibson, and G. ganger, "Principles of Operation for Shingled Disk Devices," Technical Report CMU-PDL-11-107, Carnegie Mellon University, 2011.

[5] 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, Tech. Rep., May 2009, CMU-PDL-09-014.

[6] Y. Shiroishi, K. Fukuda, I. Tagawa, S. Takenoiri, H. Tanaka, and N. Yoshikawa, "Future options for HDD storage," IEEE Transactions on Magnetics, Vol. 45, no. 10, Oct. 2009.

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

[8] J. -G. Zhu, X. Zhu, and Y. Tang, "Microwave assisted magnetic recording," IEEE Transactions on Magnetics, Vol. 44, no. 1, pp. 125-131, Jan. 2008.

[9] 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 Transactions on Magnetics, Vol. 33, no. 1, pp. 990-995, Jan. 1997.

[10] R. Wood, "The feasibility of magnetic recording at 1 terabit per square inch," IEEE Transactions on magnetics, Vol. 36, no. 1, pp. 36-42, Jan. 2000.

[11] 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 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[12] A. Amer, J. Holliday, D. D. Long, E. Miller, J.-F. Paris, and T. Schwarz, "Data Management and Layout for Shingled Magnetic Recording," IEEE Transactions on Magnetics, vol. 47, no. 10, pp. 3691-3697, 2011.

[13] 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 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[14] D. Hall, J. Marcos, and J. Coker, "Data handling algorithms for autonomous shingled magnetic recording hdds," IEEE Transactions on Magnetics, vol. 48, no. 5, pp. 1777-1781, may 2012.

[15] R. Pitchumani, J. Hughes, and E. L. Miller, SMRDB: key-value data store for shingled magnetic recording disks. In Proceedings of the 8th ACM International Systems and Storage Conference, May. 2015.

[16] D. Le Moal, Z. Bandic, and C. Guyot, "Shingled file system host-side management of shingled magnetic recording disks," in Consumer Electronics (ICCE), 2012 IEEE International Conference on, jan. 2012, pp. 425-426.

[17] C. Jin, W.-Y. Xi, Z.-Y. Ching, F. Huo, and C.-T. Lim, "HiSMRfs: A high performance file system for shingled storage array, in 2014 30thSymposium on Mass Storage Systems and Technologies (MSST). IEEE, 2014, pp. 1C6.

[18] Suresh A, Gibson G, Ganger G. "Shingled Magnetic Recording for Big Data Applications, CMU-PDL-12-105, 2012.

[19] B. Gill, M. Ko, B. Debnath, and W. Belluomini, "STOW: A Spatially and Temporally Optimized Write Caching Algorithm," In USENIX ATC, Jun. 2009.

[20] R. Pitchumani, A. Hospodor, A. Amer, Y. Kang, E. L. Miller, and D. D. E. Long, "Emulating a shingled write disk," in Proceedings of the 20th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Aug. 2012.

[21] C.-I. Lin, D. Park, W. He, and D. Du, "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.

[22] D. Stodolsky, G. Gibson, and M. Holland, "Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays, Proc. 20th Ann. Intl Symp. Computer Architecture (ISCA 93), pp. 64-75, 1993.

[23] J. Menon, J. Roche, and J. Kasson, "Floating Parity and Data Disk Arrays, J. Parallel and Distributed Computing, vol. 17, nos. 1/2, pp. 129-139, 1993.

[24] J. Menon and J. Cortney, "The Architecture of a Fault-Tolerant Cached RAID Controller, ACM SIGARCH Computer Architecture News, vol. 21, no. 2, pp. 76-87, 1993.

[25] Dan Luo, Jiguang Wan, Yifeng Zhu, Nannan Zhao, Feng Li, and Changsheng Xie, "Design and Implementation of a Hybrid Shingled Write Disk System", IEEE Transactions on Parallel and Distributed Systems, no. 1, pp. 1, PrePrints, doi:10.1109/TPDS.2015.2425402

[26] Financial1.spc and Financial2.spc. http://traces.cs.umass.edu/index.php/Storage/Storage. 2002.

[27] MSR Cambridge Traces. http://iotta.snia.org/traces/388. 2008