HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability

Bo Mao†, Hong Jiang*, Dan Feng†, Suzhen Wu†, Jianxi Chen†, Lingfang Zeng†, Lei Tian†*
†Wuhan National Laboratory for Optoelectronics
†School of Computer Science and Technology
†Huazhong University of Science and Technology, Wuhan, 430074, China
*Department of Computer Science & Engineering
*University of Nebraska-Lincoln, USA
Corresponding author: dfeng@hust.edu.cn
{maobo.hust, suzhen66}@gmail.com, {jiang, tian}@cse.unl.edu, {chenjx, lfzeng, ltian}@hust.edu.cn

***Abstract*—A single flash-based Solid State Drive (SSD) can not satisfy the capacity, performance and reliability requirements of a modern storage system supporting increasingly demanding data-intensive computing applications. Applying RAID schemes to SSDs to meet these requirements, while a logical and viable solution, faces many challenges. In this paper, we propose a Hybrid Parity-based Disk Array architecture, HPDA, which combines a group of SSDs and two hard disk drives (HDDs) to improve the performance and reliability of SSD-based storage systems. In HPDA, the SSDs (data disks) and part of one HDD (parity disk) compose a RAID4 disk array. Meanwhile, a second HDD and the free space of the parity disk are mirrored to form a RAID1-style write buffer that temporarily absorbs the small write requests and acts as a surrogate set during recovery when a disk fails. The write data is reclaimed back to the data disks during the lightly loaded or idle periods of the system. Reliability analysis shows that the reliability of HPDA, in terms of MTTDL (Mean Time To Data Loss), is better than that of either pure HDD-based or SSD-based disk array. Our prototype implementation of HPDA and performance evaluations show that HPDA significantly outperforms either HDD-based or SSD-based disk array.**

***Keywords*-RAID; SSD; HDD; Performance Evaluation; Reliability Analysis;**

I. Introduction

The development of multi-core processors in computer systems has further widened the performance gap between CPU and storage devices. Due to the slow mechanical positioning nature of hard disk drives (HDDs), optimizing the performance of HDD-based storage systems has been considered extremely important. A plethora of optimization techniques, such as parallel I/O, caching, and prefetching, have been introduced. For example, the widely used RAID (Redundant Array of Independent Disks) architecture [1] employs the parallel I/O technique and exploits the data redundancy mechanism to improve the performance and reliability of storage systems.

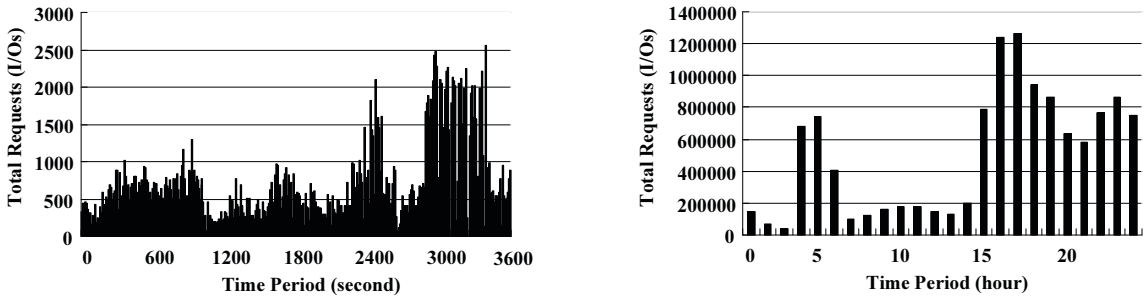
Recently, the flash-based Solid State Drive (SSD) has become an emerging alternative to HDD and received great attentions from both academia and industry [2], [3], [4]. Different from HDDs, SSDs are based on semiconductor chips and provide many benefits, such as low power consumption, high shock resistance, and most importantly, extraordinarily high performance for small random reads. Unfortunately, SSDs also have some disadvantages, such as low performance for small random writes, erase-before-write problem, and the flash wear-out problem [2].

Similar to HDDs, a single SSD can not satisfy the performance, capacity and reliability requirements of storage systems. Moreover, failures of SSDs typically occur in the controller silicon, which renders the whole SSD unusable [5]. Thus applying RAID algorithms to SSDs is necessary and likely promising, not unlike the case for HDDs, to build high performance and highly reliable SSD-based storage systems.

However, straightforwardly applying RAID algorithms to SSDs is challenging due to the special characteristics of flash-based SSDs, such as the erase-before-write and wear-out issues [6], [7]. In this paper, we propose a hybrid SSD-HDD disk array architecture, called Hybrid Parity-based Disk Array (or HPDA), which combines an array of SSDs and two HDDs to improve the performance and reliability of storage systems. In HPDA, the SSDs (data disks) and part of one HDD (parity disk) constitute a RAID4 disk array. The second HDD and the free space of the parity disk are mirrored as a write buffer that temporarily absorbs the small write requests and acts as a surrogate RAID1 set [8] during recovery when a disk fails. The write data is reclaimed back to the data disks during the system idle periods.

Since the parity disk is an HDD, the flash wear-out and erase-before-write problems caused by the parity-

978-1-4244-6443-2/10/$26.00 ©2010 IEEE



(a) OLTP application (Financial1.spc)

(b) HP Labs I/O workloads (Cello99)

Figure 1. Access patterns of applications.

update operations are avoided. Moreover, the mirroring buffer improves the small random write performance. Our reliability analysis shows that the reliability of HPDA, in terms of MTTDL, is better than that of either an HDD-based or an SSD-based disk array of the same capacity. On the other hand, our prototype implementation of HPDA and performance evaluation show that HPDA significantly outperforms HDD-based and SSD-based disk arrays.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. We describe the design of HPDA in Section III. The reliability analysis and performance evaluations are presented in Section IV and Section V respectively. We review the related work in Section VI and conclude this paper in Section VII.

II. Background and Motivation

In this section, we provide the necessary background knowledge that motivates our research and facilitates our presentation of HPDA in the following sections.

A. Workload characteristics

Researchers have extensively collected and analyzed workload traces at the disk-level, and found that burstiness and idleness are common among most applications [9], [10], [11]. Figure 1 plots the access patterns of two applications, i.e., the financial workloads obtained from the Storage Performance Council [12] and the computer research workloads provided by the Hewlett-Packard Labs. We can see that the accesses exhibit a mixed pattern of burstiness and idleness in terms of I/O intensity. With the help of the upper-layer optimizing techniques such as buffer and I/O scheduling, the I/Os seen at the disk level are usually bursty and clustered.

Since the SSDs service the read requests more rapidly than write requests, the write requests will dominate disk array's queue and could potentially become a system bottleneck. In addition, small write performance dominates the performance of many applications such as on-line transaction processing and office/engineering environments [13], [14]. Thus, write performance is essential to the overall I/O performance [15], especially for the SSD-based disk array.

B. SSD basics

Unlike mechanical HDDs, SSDs are made of silicon memory chips and have no moving parts (i.e., mechanical positioning parts). Like HDDs, data in SSDs is persistent when they are powered down. Despite SSD's high energy efficiency and high random-read performance, there are two unusual limitations of SSDs that must be addressed [16].

First, the current generation of SSDs suffers from the poor performance of small random writes. The reason is that in the flash storage, each block of size 64–128 KB must be erased in advance before any part of it can be rewritten, a characteristic feature of SSD known as "erase-before-write". Due to the sheer size of a block, an erase operation typically takes milliseconds to complete, one or two order of magnitude higher than a read operation.

Second, the flash wear-out after repeated write-erase cycles impacts the reliability of SSDs. Generally, for the single level cell (SLC) NAND flash memory, the expected number of erasures per block is 100,000 and it is reduced to 10,000 for the multi level cell (MLC) NAND flash memory.

These two limitations of SSDs must be taken into consideration when designing SSD-based storage systems, especially SSD-based disk arrays.

C. Challenges of applying RAID algorithms to SSDs

Table I summarizes SSD-based disk arrays from the viewpoints of reliability, performance and cost. From the table, we can see that straightforwardly applying RAID algorithms to SSDs to achieve high performance and reliability, however, poses three main challenges:

First, RAID0, which does not have data redundancy, is not reliable for SSD-based disk arrays.

Second, RAID1 and its variations that offer duplicative data redundancy are too expensive for SSD-based disk arrays to be cost-effective due to the high cost/GB

Table I
COMPARISON OF DIFFERENT SSD-BASED DISK ARRAYS.

Schemes	Reliability	Performance	Cost
RAID0	low (no redundancy)	medium (small write)	low
RAID1/10	low (double writes)	low (small write)	high (dual redundancy)
RAID5/6	low (frequently parity updates)	low (small write)	low
HPDA	high (parity & mirroring protected)	high (log-based disk buffer)	low

of the current generation of SSDs relative to HDDs. Moreover, since each write request incurs two writes in mirroring arrays, the reliability of the system also degrades due to the flash wear out.

Third and most important, for RAID levels that offer parity redundancy, the parity update operations for small writes exacerbate the flash wear-out problem due to the additional and concentrated erase-before-write operations that occur in the parity blocks [6], [7]. Moreover, the poor performance of small writes to SSD will aggravate the write performance for the parity-based disk arrays. Thus, the the overall I/O performance and reliability of the SSD-based RAID5 will be affected.

D. Motivation

Failures of SSDs typically occur in the controller silicon rather than the flash device as indicated by recent studies [5]. Thus it is necessary to apply RAID algorithms to SSDs for applications that require high reliability, high performance and high capacity [6], [7]. However, simply applying RAID algorithms to SSDs can be nontrivial, as discussed in the previous section. The limitations of SSDs must be addressed when designing SSD-based disk arrays.

On the other hand, the bursty and clustering characteristics of the I/O workload make us to reconsider the designing of the storage systems. If work can be delayed or alleviated from the busy periods to the less busy ones, resource contention and queue length during the busy periods can be reduced, and perceived system performance can be improved [17].

Based on the above observations, we proposed HPDA, which combines the advantages of both SSDs and HDDs to build high performance and high reliable storage systems. Moreover, HPDA takes the workload characteristics into consideration. The small write data in HPDA is temporarily and sequentially stored in the mirroring buffer HDDs and reclaimed back to the SSDs during the lightly loaded or idle periods of the system. Therefor, HPDA improves the performance of the storage systems transparently to the end users.

III. HPDA

A. Architecture of HPDA

Figure 2 shows the architecture of the proposed HPDA. The disk subsystem is composed of both SSDs and HDDs. As shown in Figure 2, several SSDs and one HDD form a RAID4 disk array where the data disks are SSDs and the parity disk is HDD. Since the capacity of HDD is usually much larger than SSD, the free space of the parity disk and the second HDD are mirrored as a RAID1-structured write buffer for the RAID4 disk array. The mirroring buffer temporarily absorbs the small random-write requests. During system idle period, the write data will be reclaimed back to the RAID4 disk array.

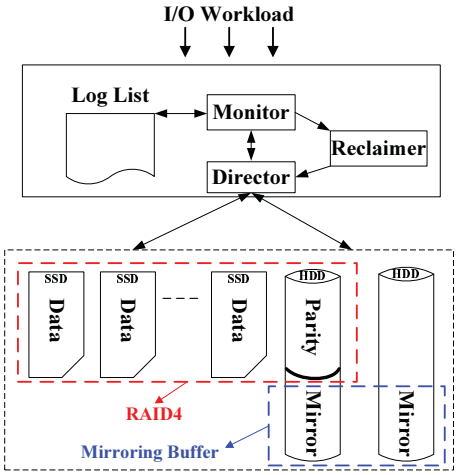


Figure 2. Architecture of HPDA.

It must be noted that the write data is laid out sequentially, much like LFS (Log File System [14]), in the mirroring buffer. Rewrites simply invalidate the old data instead of overwriting the old data. So writing to the mirroring buffer is very fast. On the other hand, since serving read requests does not involve the parity disk, the contention of the disk head between the parity area and the mirroring area in the parity disk is avoided.

HPDA consists of three key functional modules: Monitor, Director and Reclaimer. *Monitor* is responsible for monitoring the I/O accesses of applications, identifying the random write accesses and computing the I/O intensity. *Director* is responsible for issuing the I/O requests to different locations guided by the Monitor, namely, the RAID4 disk array or the mirroring buffer, which will be detailed in Section 3.3. *Reclaimer* is responsible for reclaiming the write data from the mirroring buffer back

to the RAID4 disk array according to the log list that records the redirected write data. Thus, the additional cost to build HPDA is the extra marking memory that stores the log list. Fortunately, the marking memory uses NVRAM that is commonly used in storage controllers.

B. Key Data structure

The main data structure in HPDA is the log list that contains a number of entries, as shown in Figure 3. Each entry corresponds to a write that is temporarily buffered in the mirroring buffer. The main variables in the entry are explained below:

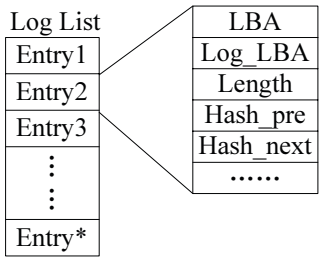


Figure 3. Key data structure in HPDA.

- *LBA* indicates the offset of a data block in RAID4 disk array;
- *Log_LBA* represents the offset of a data block in the mirroring buffer. The value of this variable increases gradually and restarts from the starting position after the reclaiming operation is completed;
- *Length* indicates the length of a data block;
- *Hash_pre* and *Hash_next* are two pointers used to link the sorted list.

Because of its critical importance in processing the reclaiming operation and the recovering operation (from disk failure), the log list must be kept in an NVRAM to avoid data loss in case of power failure.

HPDA only needs to maintain one hash entry per request rather than per disk sector. The log list size is $n * \frac{l}{r}$, where n is the number of bytes per hash entry, l is the mirroring buffer size, and r is the request size. For example, with a 2GB mirroring buffer size and 4KB average request size, the total log list size is only around 10 MB. Moreover, the extra memory space can be further reduced by periodically reclaiming the write data from the mirroring buffer back to RAID4 disk array during system idle periods.

On the other hand, since every request requires a search of the write buffer log list to determine whether there is a copy of the requested data in the write buffer, the effectiveness of the buffer log list is very important to the system performance. First, the write buffer log is hash indexed [18], [19] and *LBA* (logic block address) is the key. As the write buffer is used to temporarily store the write data, the logged data will be reclaimed to the SSDs during the system idle periods. Moreover, with the current processor speed, the search time penalty (ns) is very small compared with the disk I/O time (us or ms). Thus the search time can be neglected in the request service time. Our previous research experiments [18], [19] and the experiments in section V on the real workloads also validate this.

C. Process flow

When receiving a read request, the Monitor first checks whether there is an entry corresponding to the request in the log list. If yes, the data is read from the mirroring buffer. Otherwise, the request will be processed by the RAID4 disk array and served exclusively by the SSDs.

Upon receiving a write request, the Monitor first determines whether the request is sequential with its prior requests. If yes, the request is processed by the RAID4 disk array. At the same time, if the request is in the log list, the corresponding log entry should be deleted. If the request is random, the data will be written to the mirroring buffer. And if the request already exists in the log list, the previous log entry will be deleted and replaced by the new log entry. Otherwise, a new log entry is created according to the request and inserted into the log list.

An additional operation in HPDA is the reclaiming operation that reclaims the write data from the mirroring buffer back to the RAID4 disk array. The reclaiming operation is usually executed during system idle periods determined by the Monitor based on the I/O intensity. On the other hand, when the mirroring buffer is full, the mirroring buffer can not continue to absorb the random write requests and the write data must also be reclaimed. When the write data is reclaimed from the mirroring buffer to the RAID4 disk array, the corresponding entry in the log list is deleted.

D. Recovery from disk failures

Disk failures can occur either in the SDDs or in the HDDs. If an SSD or the parity disk fails, the recovery operation is initiated in the RAID4 disk array. During recovery, all write requests are directed to the mirroring buffer and read requests are served as usual.

If the other HDD fails, the Reclaimer is triggered to reclaim the write data from the mirroring buffer to the RAID4 disk array according to the log list. After the reclaim process completes, the newly added HDD and the free space of the parity disk are mirrored again to act as a RAID1 buffer. During the reclaiming operation, all requests are served by the RAID4 disk array. Every request must be checked in the log list to keep data consistent, that is, if the write request is in the log list,

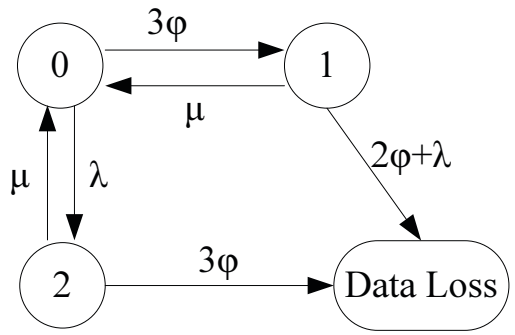


Figure 4. State-transition probability diagram for an HPDA consisting of 3 SSDs and 2 HDDs.

the corresponding entry will be deleted after the data is written to the RAID4 disk array.

IV. Reliability Analysis

To estimate the reliability of our proposed HPDA, we adopt the mean time to data loss (MTTDL) metric that is widely used in the reliability analysis of storage systems. To the best of our knowledge, there is no single commonly acceptable MTTDL in the industry for storage systems. What most people accept is that the higher the MTTDL, the higher the reliability of the storage system. Thus, when we design a reliable storage system, we should pursue as a higher MTTDL as possible [19], [20].

Our system model consists of a disk array with independent failure processes for all disks. When a disk fails, a repair process is immediately initiated for that disk. We assume that HDD failures, SSD failures and the repair process are independent events following an exponential distribution of rate λ, φ, and μ respectively. While some of these assumptions are not true for real systems, they are necessary in order to be able to use stochastic models with finite numbers of states [20], [21] and all the disk array schemes are based on the same assumptions.

According to the conclusion about the MTTDL results of RAID5 [20], the MTTDL of RAID5 consisting of 4 HDDs is:

$$MTTDL_{RAID5-H} = \frac{7\lambda + \mu}{12\lambda^2} \tag{1}$$

and the MTTDL of RAID5 consisting of 4 SSDs is:

$$MTTDL_{RAID5-S} = \frac{7\varphi + \mu}{12\varphi^2} \tag{2}$$

Figure 4 shows the state transition diagram for an HPDA consisting of 3 SSDs and 2 HDDs. State <0> represents the normal state of the disk array when its five disks are all operational. A failure of any of the 3 SSDs would bring the disk array to state <1>. The failure of the parity disk would bring the disk array to state <2>. A failure of a second disk either in state <1> or in state <2> would result in data loss. The repair transition brings the disk array back from state <2> to state <0>, or from state <1> to state <0>. As the failure of the other HDD does not incur data loss in the mirroring buffer, we omit this condition from the state-transition probability diagram, which does not impact the results.

The Kolmogorov system of differential equations describing the behavior of this HPDA is expressed in Equation (3):

$$\begin{cases} \dfrac{dp_0(t)}{dt} = -(3\varphi + \lambda)p_0(t) + \mu p_1(t) + \mu p_2(t) \\ \dfrac{dp_1(t)}{dt} = -(2\varphi + \lambda + \mu)p_1(t) + 3\varphi p_0(t) \\ \dfrac{dp_2(t)}{dt} = -(3\varphi + \mu)p_2(t) + \lambda p_0(t) \end{cases} \tag{3}$$

where $p_i(t)$ is the probability that the disk array is in state <i>with the initial condition $p_0(0) = 1$ and $p_i(0) = 0$ for $i \neq 0$.

The Laplace transformation of Equation (3) is:

$$\begin{cases} sp_0^*(s) - 1 = -(3\varphi + \lambda)p_0^*(s) + \mu p_1^*(s) + \mu p_2^*(s) \\ sp_1^*(s) = -(2\varphi + \lambda + \mu)p_1^*(s) + 3\varphi p_0^*(s) \\ sp_2^*(s) = -(3\varphi + \mu)p_2^*(s) + \lambda p_0^*(s) \end{cases} \tag{4}$$

Observing that the mean time to data loss (MTTDL) of the disk array is given by [20]:

$$MTTDL = \sum_i p_i^*(0) \tag{5}$$

Using Equation (5), we solve the disk array of Laplace transformation for $s = 0$ and use the Equation (4) to compute MTTDL of an HPDA consisting of 3 SSDs and 2 HDDs:

$$MTTDL_{HPDA} = \frac{\Delta + 3\varphi(3\varphi + \mu) + \lambda(2\varphi + \kappa + \mu)}{(3\varphi + \lambda)\Delta - 3\varphi\mu(3\varphi + \mu) - \mu\lambda(2\varphi + \lambda + \mu)} \tag{6}$$

where $\Delta = (2\varphi + \lambda + \mu)(3\varphi + \mu)$.

Figure 5 plots MTTDLs as a function of MTTR (mean time to repair) for the different disk array architectures. The disk failure rate λ is assumed to be one failure every fifty thousand hours, which was derived from the recent studies [22]. A failure rate φ is assumed to be one failure every two hundred thousand hours, based on recent studies on SSD failure rates [23]. For the SSD-based RAID5, due to the flash wear out of the parity update operation, the value of φ is doubled with respect to the basic reliability value of SSD (without frequently parity update operation). Disk repair times are expressed in days and MTTDLs are expressed in years. From Figure 5, we can see that MTTDL of HPDA is better

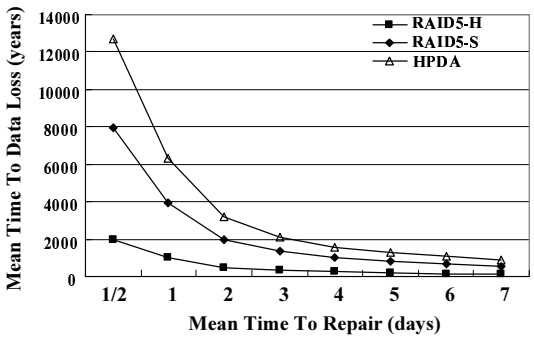


Figure 5. Mean times to data loss achieved by different disk array levels. Note that a higher MTTDL indicates a higher reliability. *RAID5-H* indicates the HDD-based RAID5 disk array and *RAID5-S* indicates the SSD-based RAID5 disk array.

than both HDD-based RAID5 and SSD-based RAID5. As the disk repair time increases, HPDA consistently outperforms the other two schemes in terms of MTTDL. As the total number of disks increases, we observe similar comparative results among the three schemes. In summary, compared with SSD-based RAID5 and HDD-based RAID5, our proposed HPDA architecture is much more reliable.

V. Performance Evaluations

A. Experimental setup and methodology

We have implemented an HPDA prototype in the Linux Software RAID framework as an independent module. The performance evaluation is conducted on a platform of server-class hardware with an Intel Xeon 3.0GHz processor and 1GB DDR memory. In the system, there is a 3Ware 9650 SATA controller card to house different SATA disks, including both SSDs and HDDs. The SSD module is the OCZ Core Serise V2 120GB Solid State Disk and the HDD module is the WD2500YD 250GB SATA Disk. A separate IDE disk is used to house the operating system (Linux kernel 2.6.21.1) and other software (MD, mdadm and RAIDmeter). The experimental setup are shown in Table II.

The traces used in our experiments are obtained from the Storage Performance Council [12]. The two financial traces were collected from OLTP applications running at a large financial institution, as shown in Table II. Performance evaluation uses the RAIDmeter [24] that is a block-level trace replay software capable of replaying traces and evaluating the I/O response time of the storage device.

B. Performance results

1) IOmeter results: We first conduct an experiment on different RAID architectures using IOmeter [25] (Version 2006.07.27) in different access patterns. Different

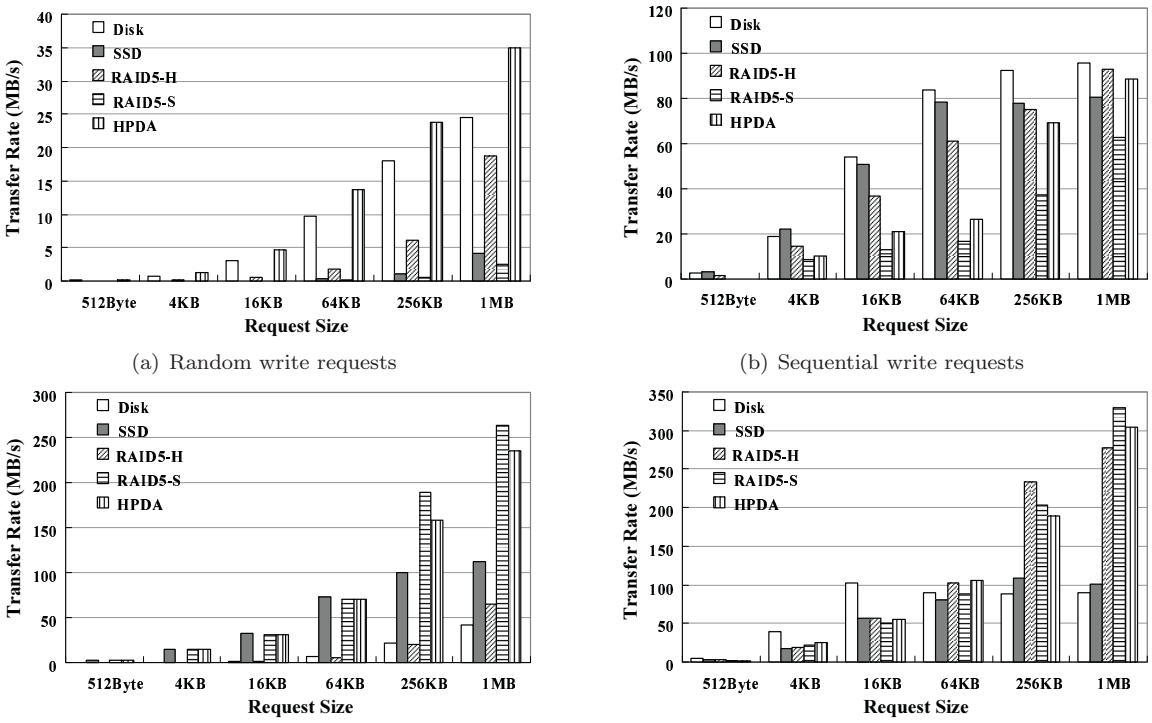
Table II
Experimental setup and trace characteristics

Machine	Intel Xeon 3.0GHz, 1GB RAM
OS	Linux 2.6.21.1
Device adapter	3Ware 9650 SATA Controller
Disk driver	OCZ Core Series V2 120GB SSD Active Power = 2W Idle Power = 0.5W WD2500YD SATA HDD Active Power = 8.5W Idle Power = 7.25W Standby Power = 1.1W
Benchmark	IOmeter Version 2006.07.27 [25]
Traces	OLTP Application I/O [12]
Trace Characteristics	Financial1.spc: Read Ratio = 32.8% Average Request Size = 6.2KB Average IOPS = 69 Financial2.spc: Read Ratio = 82.4% Average Request Size = 2.2KB Average IOPS = 125
Trace replay	RAIDmeter [24]

RAID architectures use the same RAID volume capacity (30GB) with a stripe unit size of 64KB.

From Figure 6(a), we can see that HPDA performs the best for the random write requests. Speifically, HPDA outperforms HDD-based RAID5 and SSD-based RAID5 by 512% and 4558% on average, respectively. In Figure 6, we also plot the performance of a single HDD and a single SSD. We can see that the random write performance of a single SSD is very low. Thus, the performance of SSD-based RAID5 is even worse due to the Read-Modify-Write operations [26]. Since the random writes are kept sequential in the mirroring write buffer, the performance of the random writes of HPDA is much higher than the other two schemes.

For the sequential write requests, as shown in Figure 6(b), HPDA outperforms SSD-based RAID5 by 44.8% on average, but underperforms HDD-based RAID5 by 38.4% on average. The reason is that the sequential performance of HDD is comparable or better than that of SSD (Figure 6(b)). Thus there is little performance advantage of SSD-based disk arrays over HDD-based arrays under sequential workloads. Moreover, the larger sequential write request in Figure 6(b) are a mix of full stripe writes and small writes for a 4-disk RAID5. For example, a 256KB request is split into a full stripe write request (192KB) and a small write request (64KB). Consequently, each large write (larger than 192KB) is divided into some full stripe writes and one small write. So the performance of SSD-based RAID5 is limited by the small write request performance that is included in the larger write request. Our extensive experiments on a full-stripe write request size (192KB) access on HDD-based RAID5 show that the performance is 174.61MB/s, which is the highest among all the request sizes.



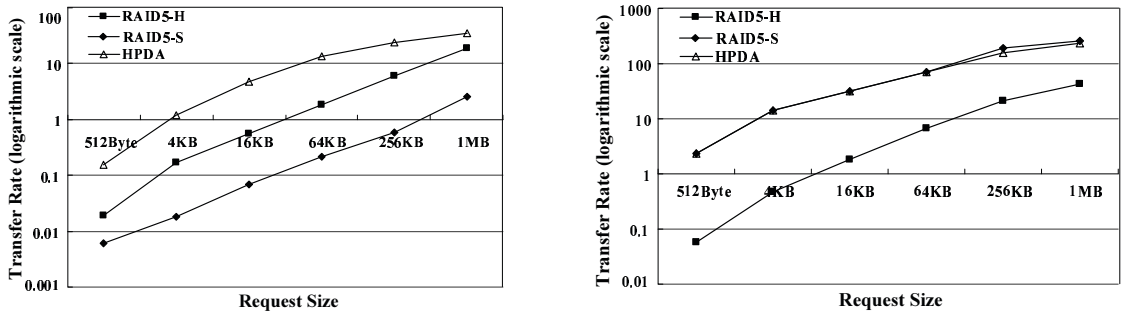
(a) Random write requests

(b) Sequential write requests

(c) Random read requests

(d) Sequential read requests

Figure 6. IOmeter performance results with respect to different access patterns.



(a) Random write requests

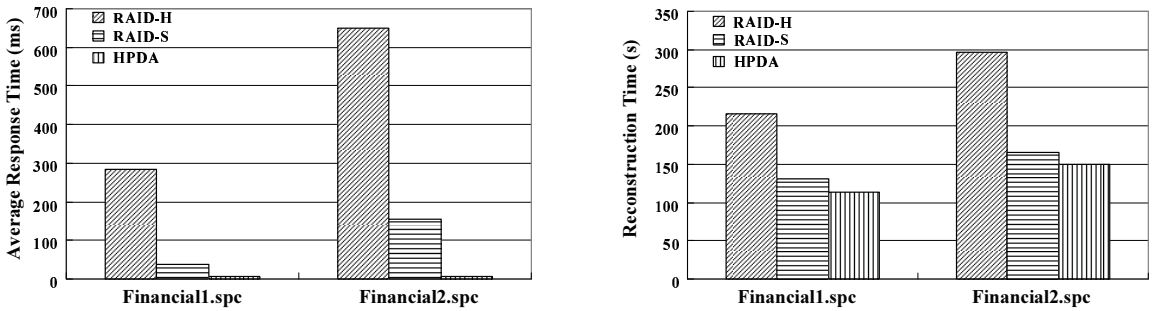
(b) Random read requests

Figure 7. IOmeter performance results with respect to different access patterns for random accesses using logarithmic scale.

For random read requests, as shown in Figure 6(c), SSD-based RAID5 performs the best due to its higher parallelism than HPDA (4:3). On the other hand, both SSD-based RAID5 and HPDA outperform HDD-based RAID5. The reason is that the random read performance of SSD is significantly better than that of HDD, as shown in Figure 6(c). For sequential read requests, the performance of the three disk array schemes are comparable, as shown in Figure 6(d).

In order to see more clearly the difference for the small request sizes and compared with the benchmark trace results, we replot the Figure 6(a) and Figure 6(c) using logarithmic scale, as shown in Figure 7. In Figure 7(a) and Figure 7(b), when the request is small (for example 4KB), the performance of HPDA is the best for random write and random read, which corresponds to the benchmark trace replay results in the later section. For the bigger request sizes, the performance varies significantly. For example, for larger reads, HPDA is better than HDD-based RAID5 but worse than SSD-based RAID5. For larger writes, HPDA is significantly better than SSD-based RAID5. HPDA is better than HDD-based RAID5 for random writes but worse for sequential writes.

In a word, for the IOmeter benchmark there is no scheme that is the best for all the access cases, but HPDA is the best for most of the access cases and



(a) Average response time. (b) Reconstruction time.

Figure 9. Recovery performance comparison driven by the two OLTP Financial traces.

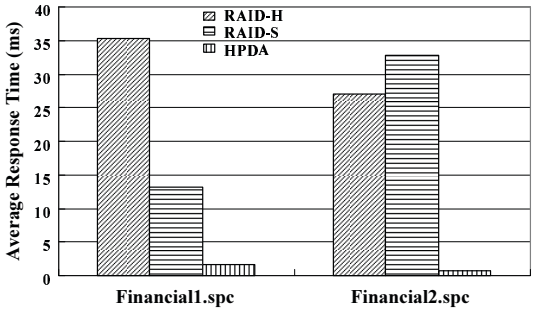


Figure 8. Response time comparison in the normal state driven by the two OLTP Financial traces.

alleviates the disadvantages of the other two schemes: the poor random read/write performance of HDD-based RAID5 and the poor random write performance of SSD-based RAID5.

2) Benchmark trace results: We conduct the second experiment on HPDA, HDD-based RAID5 and SSD-based RAID5 with the same capacity (30GB) and stripe unit size (64KB) driven by the two financial traces.

Figure 8 shows the performance results in the normal mode. We can see that HPDA performs the best. In terms of average response time, HPDA outperforms HDD-based RAID5 by a factor of up to 22.3 and 42.2 respectively under the two traces, and outperforms SSD-based RAID5 by a factor of up to 8.4 and 51.3 respectively under the two traces. The reason is that for the OLTP workloads, the I/O requests are usually small and random. SSDs are more effective in serving these types of requests than HDDs. For SSD-based RAID5, the access latency for random small write requests is very long, as shown in Figure 6(a). In particular, since most requests of Financial2.spc are smaller than the size of a flash page (as shown in Table II), these write requests incur substantial erase-before-write operations for SSDs, thus adversely impacting performance. The benchmark trace results also validate the IOmeter benchmark result as shown in Figure 7.

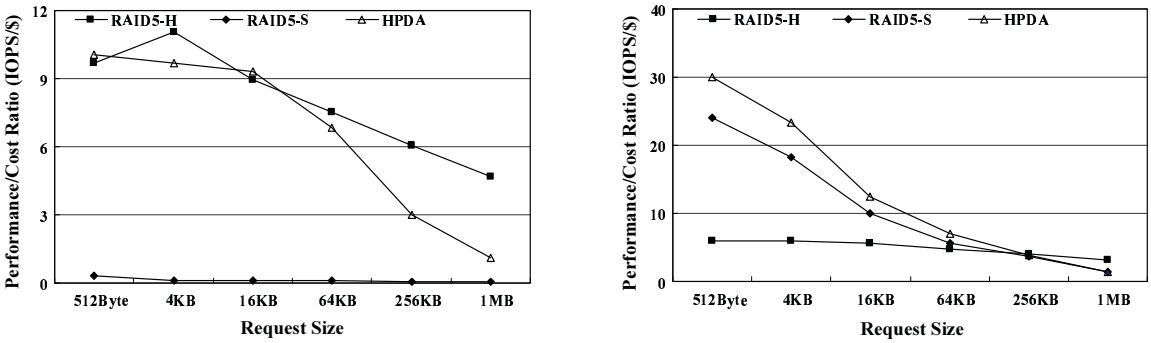
To see how effectively HPDA handles failure recovery, we also conducted experiments on the recovery process of different disk arrays. Figure 9(a) shows the average response time during recovery and Figure 9(b) shows the reconstruction time for the three disk array schemes driven by the two financial traces. Similar to the normal mode, HPDA significantly performs the best in terms of average response time and reconstruction time. The reason is that the reconstruction I/Os and user I/Os compete for disk resources, thus increasing the reconstruction time and user response time. In HPDA, the mirroring buffer absorbs all write requests, thus significantly alleviating the contentions between the user I/Os and reconstruction I/Os, thus reducing the reconstruction time and user response time simultaneously [8].

C. Performance/cost and performance/energy comparison

Besides the evaluation on throughput and latency, we also quantify the comparison of performance/cost and energy efficiency for the different disk array schemes.

For the current generation of flash-based SSDs, the cost/GB is about 10 times than that of HDDs [16]. Figure 10 show the performance/cost comparison with respect to different access patterns for random accesses. When the request size is small (less than 64KB), the performance/cost of HPDA is the best among the three disk array schemes. As the request size increases, the performance of HDD-based RAID5 increases as shown in Figure 6. Thus, the performance/cost of HDD-based RAID5 is better than that of HPDA due to the low cost of HDD-based RAID5 with the same capacity. Compared with the SSD-based disk array, HPDA decreases the cost when the capacity is the same. Moreover, the performance of HPDA is much better than that of the SSD-based disk array as shown in Section V-B. Therefore, the performance/cost of HPDA is better than that of the SSD-based disk array, as shown in Figure 10.

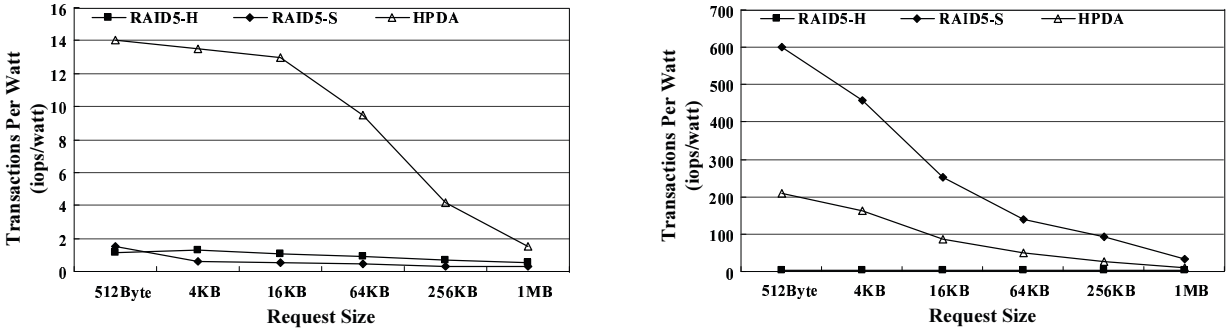
The energy consumed by disks in the active and idle modes is charged on a per-request basis and shown in



(a) Random write access

(b) Random read access

Figure 10. Performance/Cost (IOPS/$) comparison with respect to different access patterns for random accesses.



(a) Random write access

(b) Random read access

Figure 11. Performance/Power (IOPS/Watt) consumption with respect to different access patterns for random accesses.

Table II. During the IOmeter test, disks are driven to saturation all the time. We compare the transactions per watt (short for iops/watt) measure [19] for energy consumption. The results are show in Figure 11. We can see that the energy efficiency of HPDA is better than SSD-based RAID5 for random write accesses, but worse for random read accesses. The reason is that the significant performance difference between the random write and random read as shown in Figure 7. Due to the larger performance advantage for random write accesses, HPDA can remedy the energy consumed by the two HDDs. So the energy efficiency of HPDA is better than SSD-based RAID5 for random write accesses. However, The energy efficiency of both HPDA and SSD-based RAID5 is better than that of HDD-based RAID5 due to the high energy consumption and low random performance of HDDs.

VI. Related Work

Studies conducted on flash-based SSDs fall into two categories, namely, white-box that optimizes SSDs internally [2], [3], [27], [28], [29], [30] and black-box that trades SSDs as storage devices to boost the performance of applications [6], [31], [32], [33], [34]. Our work belongs to the latter. To the best of our knowledge, ours represents a first attempt at analyzing and evaluating SSD-based disk arrays.

Using the magnetic and the flash disks at the same level of the storage hierarchy, Koltsidas and Viglas [31] proposed a scheme to place the pages with a read-intensive workload on the flash disk and the pages with a write-intensive workload on the magnetic disk. Kim and Ahn [28] proposed BPLRU, a write buffer management strategy, to enhance the random write performance of flash storage. A study on hybrid MEMS/disk arrays [35] evaluates the potential gains in performance and cost by incorporating MEMS-based storage devices in disk arrays with replication redundancy. However, applying similar replication redundancy is too expensive for SSD-based arrays to be cost-effective.

Gokul Soundararajan et al. [30] proposed a hybrid storage device that uses a hard disk drive (HDD) as a write cache for an SSD. By maintaining a log-structured HDD cache and migrating cached data periodically, the hybrid design reduces writes to the SSD while retaining its excellent performance. Their performance evaluations show that the hybrid design extends SSD lifetime by 2 times and reduces average I/O latency by 42%. Our HPDA is also a hybrid SSD/HDD storage systems but targets at disk array not a single SSD.

MixStor [36] employs both SSDs and HDDs in an enterprise-level storage system. In MixStor, the SSD-based arrays and HDD-based arrays are independent and requests are distributed based on their characteristics at the application level. The small-random-write performance problem and flash wear-out problem due to the parity update in SSD-based arrays are not addressed. Our work is orthogonal and complementary to MixStor and can be easily embedded in it to further improve the reliability and performance of enterprise-level storage systems.

Using the logging technique to optimize RAID performance has been well studied. Parity Logging [13] logs the parity updates to overcome the small-write problem of RAID5. Menon [37] proposed a log-structured array (LSA) that combines LFS, RAID, compression and non-volatile cache to improve the write performance of RAID5. Logging RAID [38] bundles small writes into large RAID5 stripes using a small non-volatile memory buffer, thus solving the small-write problem of RAID5. Parity Logging, LSA and Logging RAID are based on the RAID5 architecture. GRAID [19] combines RAID10 and the logging technique to spin down about half of the disks in the normal mode to improve energy efficiency.

HP AutoRAID [39] and Hot Mirroring [40] are two schemes that combine mirroring and parity redundancy to improve RAID performance. However, they focus on improving HDD-based RAID performance. Our HPDA also utilizes mirroring and parity redundancy, but the mirroring space only acts as a temporary write buffer for the parity-based disk array. AFRAID [41] delays the parity update until the system becomes idle between bursts of client activities to eliminate the small update penalty, which plagues traditional RAID5 disk arrays. HPDA also utilizes the system idle periods to optimize the system performance.

Yang et al. [15], [42] presented a new disk storage architecture called DCD that uses a small NVRAM and a small cache disk to form a two-level cache. Write data is first assembled in the small NVRAM and then logged to the cache disk. Destaging write data from the cache disk to the data disk is delayed to idle periods. However, the read performance of DCD is very low as data may have to be read from the cache disk, which may further adversely influence the sequential log writing. The mirroring buffer in HPDA is similar to the cache disk in DCD, but the read requests in HPDA can be served effectively by SSDs.

VII. Conclusion

In this paper, we propose a hybrid parity-based disk array architecture, HPDA, that combines both SSDs and HDDs to improve the performance and reliability of storage systems. In HPDA, the SSDs (data disks) and part of one HDD (parity disk) compose a RAID4 disk array. Moreover, the other HDD and the free space of the parity disk are mirrored to form a RAID1-based write buffer to temporarily absorb the small write requests and act as a surrogate set during recovery when a disk fails. The write data is reclaimed back to the data disks during system idle periods. Reliability analysis shows that the reliability of HPDA, in terms of MTTDL, is higher than that of either an HDD-based or an SSD-based disk array. On the other hand, our prototype implementation of HPDA and its performance evaluation show that HPDA significantly outperforms both HDD-based and SSD-based disk arrays in performance.

Acknowledgments

This work is supported by the 863 project 2008AA01A402, Changjiang innovative group of Education of China No. IRT0725, US NSF support under grant numbers NSF-CCF-0621526, NSF-CCF-0937993, and NSF-IIS-0916859.

References

[1] D. Patterson, G. Gibson and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD'88)*, Jun. 1988.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proceedings of USENIX 2008 Annual Technical Conference (USENIX'08)*, Jun. 2008.

[3] C. Dirik and B. Jacob. The Performance of PC Solid-State Disks as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization. In *Proceedings of 36th International Symposium on Computer Architecture (ISCA'09)*, Jun. 2009.

[4] F. Chen, D. A. Koufaty, and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *Proceedings of 2009 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance'09)*, Jun. 2009.

[5] Samsung defends flash reliability in solid-state drives. http://news.cnet.com/8301-13924_3-9876557-64.html.

[6] K. M. Greenan and Darrell D. E. Long and Ethan L. Miller and Thomas J. E. Schwarz S. J. and A. Wildani. Building Flexible, Fault-Tolerant Flash-based Storage Systems. In *Proceedings of 5th Workshop on Hot Topics in System Dependability (HotDep'09)*, Jun. 2009.

[7] A. Kadav, M. Balakrishnan, V. Prabhakaran, and D. Malkhi. Differential RAID: Rethinking RAID for SSD Reliability. In *Proceedings of Workshop on Hot Topics in Storage and File Systems (HotStorage'09)*, Oct. 2009.

[8] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O Workload Outsourcing for Boosting RAID Reconstruction Performance. In *Proceedings of 7th USENIX Conference on File and Storage Technologies (FAST'09)*, Feb. 2009.

[9] C. Ruemler and J. Wilkes. UNIX disk access patterns. In *Proceedings of USENIX 1993 Winter Technical Conference (USENIX'93)*, Jan. 1993.

[10] A. Riska and E. Riedel. Disk Drive Level Workload Characterization. In *Proceedings of USENIX 2006 Annual Technical Conference (USENIX'06)*, Jun. 2006.

[11] N. Mi, G. Casale, L. Cherkasova and E. Smirni. Burstiness in Multi-Tier Applications: Symptoms, Causes, and New Models. In *Proceedings of the 9th ACM/IFIP/USENIX International Middleware Conference (Middleware'08)*, Dec. 2008.

[12] OLTP Application I/O. UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.

[13] D. Stodolsky, G. Gibson and M. Holland. Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays. In *Proceedings of 20th Annual International Symposium on Computer Architecture (ISCA'93)*, May. 1993.

[14] M. Rosenblum and J. K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[15] Y. Hu and Q. Yang. DCD — Disk Caching Disk: A New Approach for Boosting I/O Performance. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA'96)*, May. 1996.

[16] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proceedings of 4th European Conference on Computer Systems (EuroSys'09)*, Mar. 2009.

[17] R. Golding, P. Bosch, and C. Staelin. Idleness is Not Sloth. In *Proceedings of USENIX 1995 Technical Conference (USENIX'95)*, Jan. 1995.

[18] B. Mao, D. Feng, S. Wu, J. Chen, L. Zeng, and L. Tian. A High Performance RAID10 Storage Architecture Based on Logging Technique. In *Proceedings of 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC'08)*, Aug. 2008.

[19] B. Mao, D. Feng, H. Jiang, S. Wu, J. Chen and L. Zeng. GRAID: A Green RAID Storage Architecture with Improved Energy Efficiency and Reliability. In *Proceedings of 16th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'08)*, Sep. 2008.

[20] J. f. Pâris, A. Amer, and Darrell D. E. Long. Using Storage Class Memories to Increase the Reliability of Two-Dimensional RAID Arrays. In *Proceedings of 17th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'09)*, Sep. 2009.

[21] J. G. Elerath and M. Pecht. Enhanced Reliability Modeling of RAID Storage Systems. In *Proceedings of 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, Jun. 2007.

[22] B. Schroeder and G. A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, Feb. 2007.

[23] SanDisk Solid State Driver. http://www.sandisk.com/.

[24] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: A Popularity-based Multi-threaded Reconstruction Optimization for RAID-Structured Storage Systems. In *Proceedings of 5th USENIX Conference on File and Storage Technologies (FAST'07)*, Feb. 2007.

[25] IOmeter. http://sourceforge.net/projects/iometer.

[26] Intel Technical Reporto. Understanding the flash translation layer (FTL) specification.

[27] S. Nath and A. Kansal. FlashDB: Dynamic Self-tuning Database for NAND Flash. In *Proceedings of 2007 International Conference on Information Processing in Sensor Networks (IPSN'07)*, Apr. 2007.

[28] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST'08)*, Feb. 2008.

[29] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *Proceedings of 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*, Mar. 2009.

[30] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber. Extending SSD Lifetimes with Disk-Based Write Caches. In *Proceedings of 8th USENIX Conference on File and Storage Technologies (FAST'10)*, Feb. 2010.

[31] I. Koltsidas and S. D. Viglas. Flashing Up the Storage Layer. In *Proceedings of 34th International Conference on Very Large Data Bases (VLDB'08)*, Aug. 2008.

[32] M. W. Storer, K. M. Greenan, Ethan L. Miller, and K. Voruganti. Pergamum: Replacing Tape with Energy Efficient, Reliable, Disk-Based Archival Storage. In *Proceedings of 6th USENIX Conference on File and Storage Technologies (FAST'08)*, Feb. 2008.

[33] A. Caulfield, L. Grupp, and S. Swanson. Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications. In *Proceedings of 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*, Mar. 2009.

[34] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*, Oct. 2009.

[35] M. Uysal, A. Merchantand, and G. A. Alvarez. Using MEMS-Based Storage in Disk Arrays. In *Proceedings of 2rd USENIX Conference on File and Storage Technologies (FAST'03)*, Mar. 2003.

[36] Y. Kim, A. Gupta and B. Urgaonkar. MixedStore: An Enterprise-scale Storage System Combining Solid-state and Hard Disk Drives. Technical Report 2008, Dept. of Computer Science and Engineering, The Pennsylvania State University.

[37] J. Menon. A Performance Comparison of RAID-5 and Log-Structured Arrays. In *Proceedings of 4th International Symposium on High Performance Distributed Computing (HPDC'95)*, Aug. 1995.

[38] Y. Chen, W. Hsu and H. Young. Logging RAID — An Approach to Fast, Reliable, and Low-Cost Disk Arrays. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing (Euro-Par'00)*, Aug. 2000.

[39] J. Wilkes, R. Golding, C. Staelin and T. Sullivan. The HP AutoRAID hierarchical storage system. *Operating Systems Review*, 29(5):96–108, 1995.

[40] K. Mogi and M. Kitsuregawa. Hot mirroring: a method of hiding parity update penalty and degradation during rebuilds for RAID5. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, Jun. 1996.

[41] S. Savage and J. Wilkes. AFRAID: A Frequently Redundant Array of Independent Disks. In *Proceedings of USENIX 1996 Annual Technical Conference (USENIX'96)*, Jan. 1996.

[42] Y. Hu, Q. Yang and T. Nightingale. RAPID-Cache — A Reliable and Inexpensive Write Cache for Disk I/O Systems. In *Proceedings of the 5th International Symposium on High Performance Computer Architecture (HPCA'99)*, Jan. 1999.