HRAID6ML: A Hybrid RAID6 Storage Architecture with Mirrored Logging

Lingfang Zeng †, Dan Feng † ✉, Janxi Chen †
Qingsong Wei §, Bharadwaj Veeravalli #, Wenguo Liu †

†*School of Computer, Huazhong University of Science and Technology*
†*Wuhan National Laboratory for Optoelectronics*
§*Data Storage Institute, A*STAR, Singapore*
#*National University of Singapore*
✉Corresponding author: dfeng@hust.edu.cn, {lfzeng,chenjx}@hust.edu.cn
wei_qingsong@dsi.a-star.edu.sg, elebv@nus.edu.sg, liuwenguo1985@gmail.com

***Abstract*—The RAID6 provides high reliability using double-parity-update at cost of high write penalty. In this paper, we propose HRAID6ML, a new logging architecture for RAID6 systems for enhanced energy efficiency, performance and reliability. HRAID6ML explores a group of Solid State Drives (SSDs) and Hard Disk Drives (HDDs): Two HDDs (parity disks) and several SSDs form RAID6. The free space of the two parity disks is used as mirrored log region of the whole system to absorb writes. The mirrored logging policy helps to recover system from parity disk failure. Mirrored logging operation does not introduce noticeable performance overhead to the whole system. HRAID6ML eliminates the additional hardware and energy costs, potential single point of failure and performance bottleneck. Furthermore, HRAID6ML prolongs the lifecycle of the SSDs and improves the systems energy efficiency by reducing the SSDs write frequency. We have implemented proposed HRAID6ML. Extensive trace-driven evaluations demonstrate the advantages of the HRAID6ML system over both traditional SSD-based RAID6 system and HDD-based RAID6 system.**

I. Introduction

RAID (Redundant Array of Independent Disks) [18] technology is efficient in improving overall I/O throughput and reliability of storage system by employing parallel I/O and data redundancy mechanisms. However, it suffers high latency of random accesses due to slow mechanical positioning nature of Hard Disk Drives (HDDs). NAND flash based Solid State Drives (SSDs) provide much higher random read performance and lower power consumption than HDD. The steady bit cost reduction of NAND flash memory now makes it economically viable to implement SSD using NAND flash memory [23]. RAID of SSDs is more cost-efficient than PCIe SSD in terms of capacity per dollar and bandwidth per dollar [12].

However, despite SSD's high energy efficiency and high random-read performance, there are many unusual limitations of SSDs that must be addressed. 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 64-128KB must be erased in advance before any part of it can be rewritten, a characteristic feature of SSD known as "erase-before-write" [17], [5], [10]. 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. Above limitations of SSDs must be taken into consideration when designing SSD-based storage systems, especially SSD-based arrays. 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 RAID will be affected [14].

This paper presents a new RAID architecture that exploits the advancement of SSD and HDD. The new RAID architecture is referred to as **HRAID6ML**: A **H**ybrid **RAID6** with **M**irrored **L**ogging. Several SSDs and part of two HDDs form RAID6 where the data disks are SSDs and the *parity disks* are HDDs. The free space of the parity disks is mirrored as log region of whole system to absorb writes and extend lifetime for SSD. Leveraging hybrid management of SSDs and HDDs, proposed HRAID6ML efficiently improves performance, availability and reliability with low power consumption because most read requests are served by SSD and writes are conducted in sequential way by HDDs. In addition, random writes in SSDs are minimized giving rise to longer life time of SSDs. We have implemented a prototype HRAID6ML architecture using software at block device level and carried out extensive performance measurements using file system benchmarks. Our experiments demonstrated that HRAID6ML significantly outperforms either HDD-based or SSD-based RAID6.

The rest of this paper is organized as follows. We review the related work in Section II. We describe the architecture and design of HRAID6ML in Section III. The performance evaluations are presented in Section IV. And we conclude this paper in Section V.

II. Related Work

Replacing HDD with SSD is the single most effective way to improve application launch performance [9]. However, simply applying RAID algorithms to SSDs can be nontrivial, as discussed in the previous section. Also, in datacenters, storage subsystems are a major contributor to the power consumption. For a typical datacenter, the HDD-based storage subsystem can consume 27% of the total energy and this fraction tends to increase rapidly as storage requirements rise by 60% annually [25]. In addition, the bursty and clustering characteristics of the I/O workload make people to reconsider the designing of the storage systems [24]. Idle time slots has been exploited to improve performance, reliability, or energy efficiency of storage systems [16], these resources remain to be effectively and fully tapped to optimize the performance and energy efficiency of storage systems with a logging architecture.

The RAID6 architecture is playing an increasingly important role in modern storage systems due to allowing the loss of any two drives. However, its high write penalty, because of the double-parity-update overheads upon each write operation, has been a persistent performance bottleneck of the RAID6 systems. Also, synchronous small writes play a critical role in the reliability and availability of file systems and applications that use them to safely log recent state modifications and quickly recover from failures [6]. For the performance penalty on each write operation because of the overhead associated with double-parity calculations, Jin et al. [8] proposed a log-assisted RAID6 architecture, called RAID6L, to boost the write performance of the RAID6 systems. RAID6L integrates a log disk into the traditional RAID6 architecture, and alleviates its write penalty by simplifying the processing steps to service a write request. Different from RAID6L, HRAID6ML is not required a dedicated disk used as log region, moreover, HRAID6ML provides a mirrored log region to avoid log-data loss.

Gokul Soundararajan et al. [19] proposed a hybrid storage device that uses a 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. Our HRAID6ML is also a hybrid SSD/HDD storage systems but targets at disk array not a single disk. HybridStore [11] provided capacity planning technique and improved performance/lifetime during episodes of deviations from expected workloads through two mechanisms: write-regulation and fragmentation busting. Hystor [3] manages both SSDs and HDDs and achieves a performance improvement by monitoring I/O access patterns at runtime. HybridStor and Hystor are orthogonal and complementary to our work 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 [20] logs the parity updates to overcome the small-write problem of RAID5. Menon [15] proposed exploiting this notion improve RAID5 performance for write requests by turning logically random write requests to physically sequential ones. Logging RAID [4] bundles small writes into large RAID5 stripes using a small non-volatile memory buffer, thus solving the small-write problem of RAID5. Parity Logging and Logging RAID are based on the RAID5 architecture. GRAID [13] and RoLo (rotated logging) [24] combines RAID10 and the logging technique to spin down about half of the disks in the normal mode to improve energy efficiency.

Our work is closely related to HPDA [14] which combines a group of SSDs and two HDDs to improve the performance and reliability of SSD-based storage systems. The difference between them is that HPDA is based on RAID4 architecture and required a dedicated log disk (part of the log disk space is wasted), however, HRAID6ML exploits the parity disks to form a mirrored logging. The mirrored logging in HRAID6ML is similar to the cache disk in DCD (disk cache disk) [7], but the read requests in HRAID6ML can be served effectively by SSDs. HRAID6ML also utilizes the system idle periods to optimize the system performance.

III. Architecture and Design

A. HRAID6ML architecture

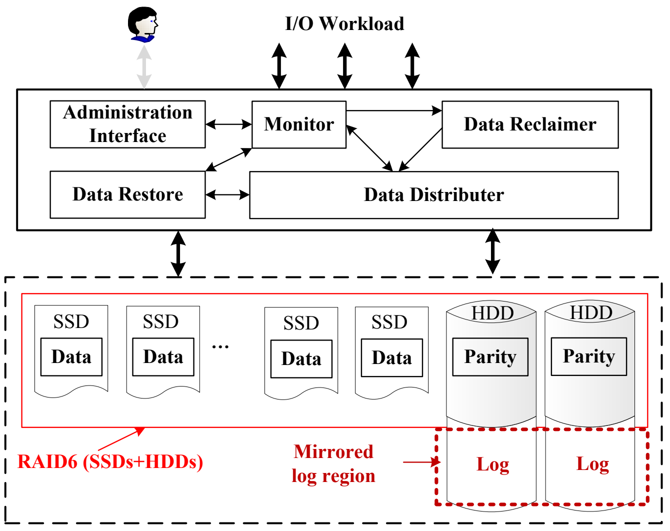


Fig. 1. HRAID6ML architecture.

HRAID6ML is composed of a traditional RAID6 region and a mirrored log region. Fig.1 provides an architectural overview of the HRAID6ML system. Several SSDs and part of two HDDs form RAID6 where the data disks are SSDs and the *parity disks* are HDDs. Since the capacity of HDD is usually much larger than SSD, the free space of the parity disks is used as *a mirrored log region* of whole system to absorb writes. The another purpose of using mirrored logging is to recover system from parity disk failure.

Generally, HRAID6ML may operate using any of the following three methods: (**i**) HRAID6ML delays SSDs writes and parity updates to accelerate the write speed, (**ii**) HRAID6ML re-synchronizes the data between the data disks and the parity disks, (**iii**) the data and parity are reclaimed to the RAID6 region. Note that writing to the *mirrored log region* is fast because data is written to the mirrored logging in sequential way. Rewrite or update is also logged sequentially instead of overwriting the old data. Since serving read requests does not involve the *parity disks*, the contention of the disk head between the parity blocks and log blocks in the parity disks is avoided.

HRAID6ML has five key functional modules: *Administration Interface*, *Monitor*, *Data Reclaimer*, *Data Restore*, and *Data Distributor*. The Administration Interface module provides an interface for system administrators to configure the HRAID6ML parameters. The Monitor module is responsible for monitoring the I/O accesses of applications, identifying the random write accesses and computing the I/O intensity. The Data Distributor module schedules the I/O request accordingly to either RAID6 region, mirrored log region. The Data reclaimer is in charge of reclaiming the written data from log region to the RAID6 region. The Data Restore module supports a typical recovery process upon an outage that results in data loss (no more than two disks failure). We will illustrate the process flow of write/read request of HRAID6ML in details in the following sections.

B. Implementation issues

We have implemented an HRAID6ML prototype in the Linux software RAID framework as an independent module. We mainly modify the *handle_stripe6* function in original RAID6 module and add the hash list structure.

1) Data structure: The main data structure in HRAID6ML is the block-log list (*blk_log_list*) that contains a number of entries. Each entry corresponds to a write that temporarily buffered in the mirrored log region. The main variables in the entry are explained as follows:

- *LBA* indicates the offset of a data block in RAID6 region.
- *buf_log_LBA* represents the offset of a data block in the *mirrored log region*.
- *reclaim_flg* represents a flag. The value of this variable is set 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 *blk_log_list* must be kept in an NVRAM to avoid data loss in case of power failure. HRAID6ML only needs to maintain one hash entry per request rather than per disk sector. The entry size of *blk_log_list* is $n \times \frac{S_{log}}{S_{req}}$, where n is the number of bytes per hash entry, S_{log} is the block-log size, and S_{req} is the request size. Since HRAID6ML tries to improve the RAID6 write/read efficiency, it keeps part of the entries of *blk_log_list* in RAM for the *mirrored log region* storage. Other entries are kept in disk. For example, with a 2GB *mirrored log region* size and 4KB average request size, the total *blk_log_list* size is only around 10MB. Moreover, the extra memory space can be further reduced by periodically reclaiming the write data from the *mirrored log region* back to *RAID6 region* during system idle periods.

2) Metadata refresh and consistency check: We update the HRAID6ML metadata (including the *blk_log_list*) using asynchronous method: the strategy is to periodically refresh or to refresh when the system is idle. For example, if there is no new I/O requests in 5s, the metadata is refreshed, or the system force to refresh the metadata every 30s. We use a timer to judge the system state (idle or not). The timer is reset while the system calls the I/O function. When the time of timer is timeout (e.g. 5s), then the asynchronous refreshing thread is waken up.

We use a checksum algorithm to guarantee a very low failure rate for aforementioned HRAID6ML metadata. Checksum calculation is as follows: $\Sigma(\widetilde{A_i}+1)$, where A_i is the i-th block-log, if the sum of check code is 0, then it is right, and vice versa, indicating an error. For example, we consider the case of 1 bit first, if there are happen to have an even number of data errors, the sum of the check code and data of the block-log is 0, then the parity bit is failure. We assume the probability of error of each data is p. $P(m)$ denotes the probability of error, m is the number of data, then, for 2 data, there are $P(2) = C_m^2 p^2$. In HRAID6ML, each checksum block is 4 bytes (32 bits), if the checksum indicates a failure of the metadata, there must have an even number of data and each bit of 32-bit is wrong. Assume there are two data consistent with the above condition, the failure probability $P(2)$ is $(C_m^2 p^2)^{32}$. If there are 100 data, then $P(2)$ is $2.89E$-31. Similarly, $P(4)$, $P(6)$ etc. also can be calculated, but all are under the orders 10^{-31} of magnitude.

C. Process flow of write/read request

Upon receiving a write request, the *Monitor* first determines whether the request is sequential with its prior requests. If yes, the *Monitor* merges the request with its prior requests and the *Data Distributor* directs the write data to the *mirrored log region* of the *parity disks*. At the same time, a new block-log entry of the request should be added into the block-log list. If the request is random, the data will be written to the *mirrored log region*. And a new block-log entry of the request will be created according to the request and inserted into the block-log list.

When receiving a read request, the *Monitor* first checks whether there is an entry corresponding to the request in the block-log list (*blk_log_list*). If yes, the data is read from the *mirrored log region*. Otherwise, the request will be processed by the RAID6 region and served exclusively by the SSDs.

An additional operation in HRAID6ML is the reclaiming operation that reclaims the write data from the *mirrored log region* back to the data disks and re-synchronizes the parity data in the parity disks. 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 *mirrored log region* is full, the *log region* can not continue to absorb the write requests and the write data must also be reclaimed.

D. Recovery

Disk failures can occur either in the SDDs or in the HDDs. **(i)** If one parity disk fail, the *Data Reclaimer* is triggered to reclaim the write data from the mirrored logging (in the normal log region) to the RAID6 region according to the block-log list. At the same time, the parity part can be recovered through the RAID6 recovery algorithm. After the reclaim processing completes, the free space of newly added parity HDD and part of another parity HDD are combined again to act as a mirrored log region.

(**ii**) If a SSD (data disk) and a parity disk (HDD) fail, each parity stripe loses one data block and one parity block. If the failed data block has a entry in the block-log list, it can be directly recovered from the *mirrored log region*; otherwise, it can be recovered through the RAID6 recovery algorithm. After the failed data blocks are recovered, HRAID6ML starts the parity re-synchronization operations, and the HRAID6ML returns to the consistent state.

(**iii**) If two SSDs (data disks) fail, in this case, each parity stripe in the RAID6 region loses two data blocks. For each failed data block, if it has a entry in the block-log list, its current value can be directly copied from the *mirrored log region*. On the other hand, if the failed data block does not have a entry in the block-log list, it must not have been updated. For each of the surviving data block in the parity stripe, its original value can be read out from the RAID6 region. Since the parity stripe is originally in the consistent state, the original value of the failed data block can be re-computed by the original value of all the surviving data blocks and the parity blocks through the RAID6 parity algorithm.

E. Scalability

The unique features of the HRAID6ML architecture have the following beneficial impacts on system scalability.

Alleviated performance bottleneck: Since the peak bandwidth of parity disks can be interleaved with the I/O workloads by exploiting short idle time slots and sequentially writing the *mirrored log region*, the performance bottleneck imposed by random write can be significantly alleviated.

Elimination of single point of failure: Since the log data are stored in the *mirrored log region*, for any log, there are two copies in HRAID6ML, the single point of failure is eliminated.

F. Reliability

For the SSD-based RAID6 region, due to the flash wear-out problem of the parity update operation, the value of failure rate is doubled with respect to the basic reliability value of SSD (with frequently parity update operation). For the write penalty in SSD-based RAID6, the $2\times$penalty is estimated on average. Basically, 80% accesses are small size and the small write requests to SSDs have the "write amplification" phenomenon which can cause much larger penalty. The proposed strategy of logging write data and combining write request (in the *parity HDDs*) reduces the writes for SSDs. And the mechanism of *mirrored logging* avoids the data loss of block-log due to *parity disk* failure.

IV. Performance Evaluations

This section proposes the evaluation methodology that we use to quantitatively study the performance of HRAID6ML as compared to SSD- or HDD-based RAID6.

A. Experimental setup and methodology

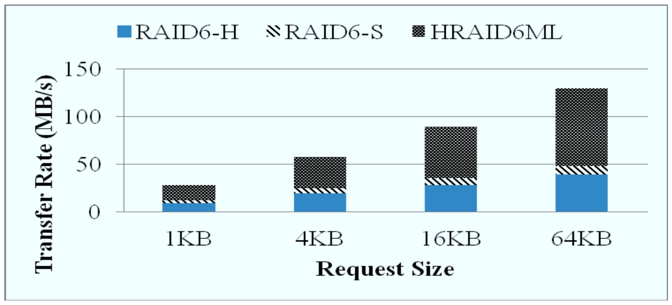
TABLE I
Experimental setup & trace characteristics.

Machine	Intel Xeon 3.0GHz, 2GB RAM
OS	Linux 2.6.21.1 Windows XP Professional SP2
iSCSI	UNH iSCSI Initiator/Target 1.7 [22] Microsoft iSCSI Initiator 2.08
Disk driver	OCZ Core Series V2 120GB SSD WD2500YD SATA 250GB HDD
Benchmark	IOmeter Version 2006.07.27 [1]
Traces	OLTP Application I/O [2]
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 [21]

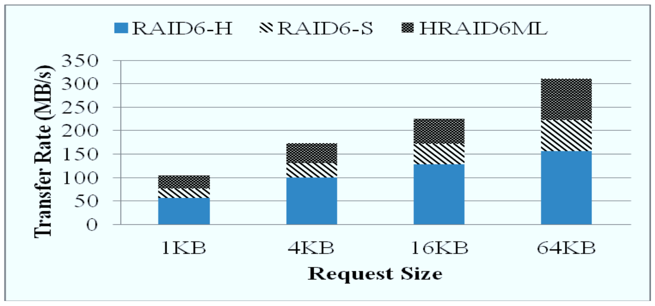
The performance evaluation is conducted on a platform of server-class hardware with an Intel Xeon 3.0GHz processor and 1GB DDR memory. In the system, a Marvel SATA controller card is used to carry 7 SATA disks (SSDs or HDDs). The SSD module is the OCZ Core Serise V2 120GB SSD and the HDD module is the 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 [21]). The experimental setup are shown in Table I. The traces used in our experiments are obtained from the Storage Performance Council [2]. The two financial traces were collected from OLTP (online transaction processing) applications running at a large financial institution, as shown in Table I. Performance evaluation uses the RAIDmeter [21] that is a block-level trace replay software capable of replaying traces and evaluating the I/O response time of the storage device.

B. Data transfer rate

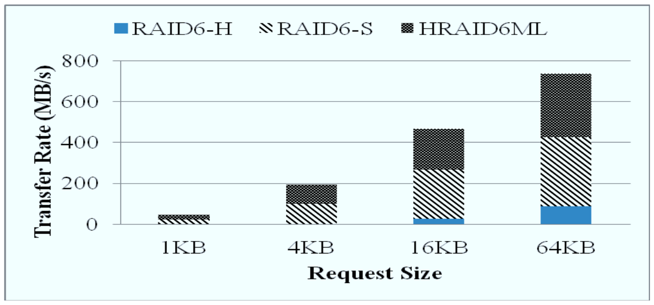
Firstly, we conducted an experiment on HRAID6ML, SSD- and HDD- based RAID6 architectures using IOmeter [1] in different workloads. Different array architectures adopt the same RAID6 volume capacity with a stripe unit size of 64KB. There are four sets of points corresponding to the four different request sizes. Each set contains three points corresponding to the transfer rate of request data for HDD-based RAID6 (referred to as "RAID6-H"), SSD-based RAID6 (referred to as "RAID6-S") and HRAID6ML respectively. From Fig.2(a), we can see that HRAID6ML performs the best for the random write requests. HRAID6ML is better than RAID6-H and RAID6-S by 107.43% and 32.03% on average, respectively. The performance of RAID6-S is worse due to the read-modify-write operations. Since the random writes are sequential order in the *mirrored log region*, the performance of the random writes of HRAID6ML is much higher than the other two strategies.



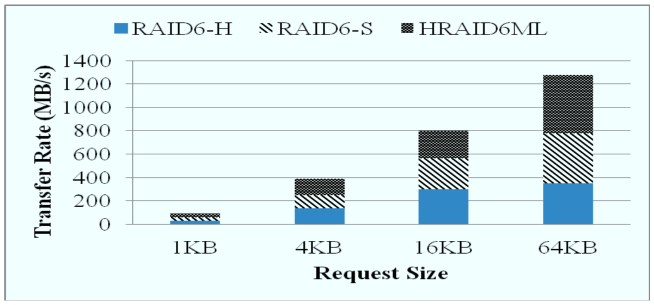
(a) Random write requests



(b) Sequential write requests



(c) Random read requests



(d) Sequential read requests

Fig. 2. IOmeter performance (data transfer rate) results with respect to different requests.

For the sequential write requests, as shown in Fig.2(b), HRAID6ML outperforms RAID6-S by 656.25% on average, but is inferior to RAID6-H by 89.85% on average. The reason is the sequential performance of HDD is comparable or outperforms that of SSD. Thus there is the little superiority of SSD-based arrays over HDD-based arrays under sequential access patterns.

For random read requests, as shown in Fig.2(c), both RAID6-S and HRAID6ML outperform RAID6-H. The reason is the random read performance of SSD is significantly better than that of HDD. For sequential read requests, the performance of the three disk array schemes are comparable, as illustrated in Fig.2(d).

C. Average response time

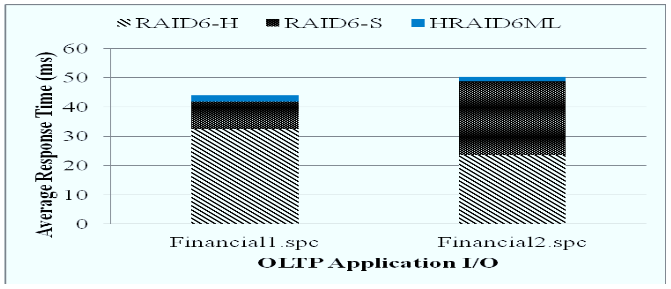
We conduct the second experiment on HRAID6ML, RAID6-H and RAID6-S with the same capacity and stripe unit size (64KB) driven by the two financial traces. Fig.3(a) shows the performance results in the normal mode. We can see that HRAID6ML performs the best. In terms of average response time, HRAID6ML outperforms RAID6-H by a factor of up to 15.29 and 14.84 respectively under the two traces, and outperforms RAID6-S by a factor of up to 4.38 and 15.73 respectively under the two traces. The reason is that for the OLTP workloads, the I/O requests are usually random and small. SSDs are more effective in serving these types of requests than HDDs. For RAID6-S, the access latency for random small write requests is very long, as shown in Fig.3(a). In particular, since most requests of *Financial2.spc* are smaller than the size of a flash page, these write requests incur substantial "erase-before-write" operations for SSDs, thus adversely impacting performance.

To see how effectively HRAID6ML handles *disk failure recovery*, we also conducted experiments on the recovery process of different disk arrays. Fig.3(b) shows the average response time during recovery for the three disk array strategies driven by the two financial traces. Similar to the normal mode, HRAID6ML significantly performs the best in terms of average response time. As shown in Fig.3(b), HRAID6ML is shorter than RAID6-H and RAID6-S by: (96.71%, 98.61%), (73.45%, 93.67%). The first parenthesized pair is the improvement achieved by HRAID6ML over RAID6-H under the two traces respectively. The second parenthesized pair is the improvement achieved by HRAID6ML over RAID6-S under the two traces respectively. The reason is that the reconstruction I/Os and user I/Os compete for disk resources, thus increasing the user response time. In HRAID6ML, the *log region* absorbs all write requests, thus significantly alleviating the contentions between the user I/Os and reconstruction I/Os, thus reducing user response time.

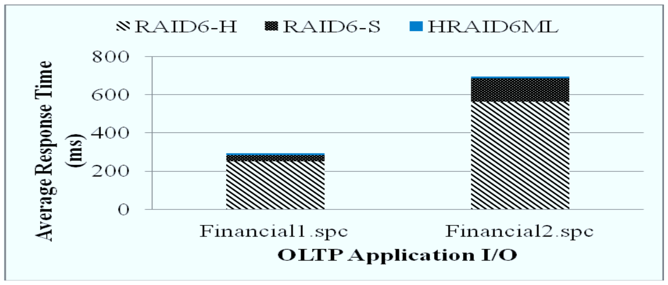
V. Conclusion

With the fast technical improvement, SSD is becoming an important part of the RAID to significantly improve performance and energy efficiency. However, due to its relatively high price and low capacity, a major system research issue to address is on how to make SSD play its most effective role in a high-performance RAID. This paper makes the following main contributions:

- We proposed a new RAID architecture, HRAID6ML, which makes full use of respective advantages of SSDs and HDDs. HRAID6ML effectively exploits the high sequential-write performance and no finite number of



(a) Normal mode



(b) Degraded mode

Fig. 3. Comparison of *average response time* driven by the two OLTP Financial traces.

writes of HDDs, and random-read performance and energy efficiency of SSDs.

- We implemented HRAID6ML in the Linux software RAID system. We conducted comprehensive experiments on our prototype implementation to evaluate the HRAID6ML performance. The results show that HRAID6ML improves the energy efficiency, reliability and performance significantly.

Acknowledgments

This research is sponsored in part by the National 973 Program of China under Grant No. 2011CB302301, China National Funds for Distinguished Young Scientists under Grant No. 61025008, and A-STAR, Singapore under Grant No. 112 172 0010. The authors are grateful to the anonymous reviewers for their valuable comments that helped in improving the paper.

References

[1] IOmeter. [Online]. Available: http://sourceforge.net/projects/iometer

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

[3] F. Chen, D. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. of the 25th International Conference on Supercomputing (ICS)*, Tucson, Arizona, May 2011.

[4] Y. Chen, W. Hsu, and H. Young, "Logging RAID – an approach to fast, reliable, and low-cost disk arrays," in *Proc. of the 6th International Euro-Par Conference on Parallel Processing (Euro-Par)*, Aug. 2000, pp. pp.1302–1311.

[5] K. M. Greenan, D. D. E, L. Ethan, L. Miller, T. J. E. Schwarz, and S. J. A. Wildani, "Building flexible, fault-tolerant flash-based storage systems," in *Proc. of 5th Workshop on Hot Topics in System Dependability (HotDep)*, Jun. 2009.

[6] A. Hatzieleftheriou and S. V. Anastasiadis, "Okeanos: Wasteless journaling for fast and reliable multistream storage," in *Proc. of the USENIX Annual Technical Conference (ATC)*, Portland, OR, June 2011.

[7] Y. Hu and Q. Yang, "DCD – disk caching disk: A new approach for boosting I/O performance," in *Proc. of the 23rd Annual International Symposium on Computer Architecture (ISCA)*, May 1996, pp. pp.169–178.

[8] C. Jin, D. Feng, H. Jiang, and L. Tian, "RAID6L: A log-assisted storage architecture with improved write performance," in *Proc. of the 27th IEEE International Symposium on Massive Storage Systems and Technologies (MSST)*, Denver, CO, May 2011.

[9] Y. Joo, J. Ryu, S. Park, and K. G. Shin, "FAST: Quick application launch on solid-state drives," in *Proc. of the 9th USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, USA, February 2011, pp. 259–272.

[10] A. Kadav, M. Balakrishnan, V. Prabhakaran, and D.Malkhi, "Differential RAID: Rethinking RAID for SSD reliability," in *Proc. of Workshop on Hot Topics in Storage and File Systems (HotStorage),*, Oct. 2009, pp. pp.15–26.

[11] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs," in *Proc. of the Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2011.

[12] Y. Kim, S. Oral, G. M. Shipman, J. Lee, D. A. Dillow, and F. Wang, "Harmonia: A globally coordinated garbage collector for arrays of solid-state drives," in *Proc. of the 27th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2011.

[13] 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 *Proc. of 16th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2008, pp. pp.113–120.

[14] B. Mao, H. Jiang, D. Feng, S. Wu, J. Chen, L. Zeng, and L. Tian, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," in *Proc. of International Symposium on Parallel & Distributed Processing (IPDPS)*. Atlanta, GA, April 2010, pp. 1–12.

[15] J. Menon, "A performance comparison of RAID-5 and log-structured arrays," in *Proc. of 4th International Symposium on High Performance Distributed Computing (HPDC)*, Aug. 1995.

[16] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel, "Efficient management of idleness in storage systems," *ACM Transactions on Storage*, vol. 5, no. 2, pp. pp.1–25, Jun. 2009.

[17] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron., "Migrating server storage to SSDs: Analysis of tradeoffs," in *Proc. of 4th European Conference on Computer Systems (EuroSys)*, Mar. 2009.

[18] D. Patterson, G. Gibson, and R. Katz., "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. of the ACM SIGMOD International Conference on Management of Data*, Jun. 1988.

[19] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending SSD lifetimes with disk-based write caches," in *Proc. of 8th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2010, pp. pp.101–114.

[20] D. Stodolsky, G. Gibson, and M. Holland, "Parity logging overcoming the small write problem in redundant disk arrays," in *Proce. of 20th Annual International Symposium on Computer Architecture (ISCA)*, 1993.

[21] 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 *Proc. of 5th USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[22] (2006) iSCSI reference implementation. University of New Hampshire (UNH). [Online]. Available: http://unh-iscsi.sourceforge.net

[23] B. Yoo, Y. Won, J. Choi, S. Yoon, S. Cho, and S. Kang, "SSD characterization: from energy consumption's perspective," in *Proc. of the 3rd USENIX conference on Hot topics in storage and file systems (HotStorage)*, 2011, pp. 3–3.

[24] Y. Yue, L. Tian, H. Jiang, F. Wang, D. Feng, Q. Zhang, and P. Zeng, "RoLo: A rotated logging storage architecture for enterprise data centers," in *Proc. of 30th International Conference on Distributed Computing Systems (ICDCS)*, 2010, pp. 293–304.

[25] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping disk arrays sleep through the winter," in *Proc. of ACM Symposium on Operating Systems Principles (SOSP)*, 2005, pp. 1–14.