2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications

PASS: A Hybrid Storage System for Performance-Synchronization Tradeoffs Using SSDs

Weijun Xiao*, Xiaoqiang Lei†, Ruixuan Li†, Nohhyun Park* and David J. Lilja*

* Department of Electrical and Computer Engineering, University of Minnesota Twin Cities
† School of Computer Science and Engineering, Huazhong University of Sci.&Tech., P.R. China
Email: wxiao@umn.edu, {hustqiang,ruixuan.li}@hust.edu.cn, {parkx408,lilja}@umn.edu

***Abstract*—Recent advances in flash memory show great potential to replace traditional hard drives (HDDs) with flash-based solid state drives (SSDs) from personal computing to distributed systems. However, it is still a long way to go before completely using SSDs for enterprise data storage. Considering the cost, performance, and reliability of SSDs, a practical solution is to combine both SSDs and HDDs together. This paper proposes a hybrid storage system named PASS (Performance-dAta Synchronization - hybrid storage System) to tradeoff between I/O performance and data discrepancy between SSDs and HDDs. PASS includes a high-performance SSD and a traditional HDD to store mirrored data for reliability. All of the I/O requests are redirected to the primary SSD first and then the updated data blocks are copied to the backup HDD asynchronously. In order to hide the latency of copying operations, we use an I/O window to coalesce write requests and maintain an ordered I/O queue to shorten the HDD seek and rotation times. Depending on the charateristics of different I/O workloads, we develop an adaptive policy to dynamically balance the foreground I/O processing and background mirroring. We implement a prototype system of PASS by developing a Linux device driver and conduct experiments on the IoMeter, PostMark, and TPC-C benchmarks. Our results show that PASS can achieve up to 12 times the performance of a RAID1 storage system for the IoMeter and PostMark workloads while tolerating less than 2% data discrepancy between the primary SSD and the backup HDD. More interestingly, while PASS does not produce any performance benefit for the TPC-C benchmark, it does allow the system to scale to larger sizes than when using an HDD-based RAID system alone.**

I. Introduction

Recent development of NAND flash shows great promise to replace traditional hard drives (HDDs) with flash-based solid state drives (SSDs) from desktop storage to enterprise storage systems. For example, Myspace announced that it replaced all server hard disks with PCIe-based SSDs to save 99% power and recently Facebook released a kernel module called flashcache to speed up MySQL by caching data in SSDs. Different from HDDs, SSDs provide much better random I/O performance than rotating HDDs due to the absence of mechanical movements. Moreover, SSDs significantly reduce power consumption and dramatically improve the robustness and shock resistance. However, it is still a long way to go before completely using SSDs for enterprise data storage. First, although cost-per-gigabyte of flash memory drops down quickly in recent years, it is still far more expensive than HDDs. Second, the capacity of a single SSD is much smaller than a hard drive. SSDs cannot provide enough storage space to store huge data sets for enterprise applications, especially for backup or archive storage. In addition, current I/O optimized algorithms on file system level and block level are based on traditional HDDs, such as I/O scheduling, buffer caching, and so on. These algorithms need to be tailored for maximal performance and data relibability by leveraging the physical properties of SSDs [1]–[4]. As a result, it cannot achieve expected high performance by naively plugging SSDs into enterprise storage systems.

In this paper, we propose a Hybrid Storage System called PASS (Performance-dAta Sychronization - hybrid storage System) considering I/O performance and data sychronization for reliability. PASS includes a high-performance SSD and a traditional HDD. It should be noted that currently an HDD can have much larger capacity than an SSD. We can use multiple SSDs to build a RAID0 storage system and then use an HDD with identical storage capacity for mirroring. For simplicity, we use an HDD and an SSD with the same storage capacity for building a hybrid storage system in this paper, although this assumption can be relaxed through the background delayed mirroring process described in this work. Our goal is to trade off I/O performance and data sychronization and look for an optimized solution to construct a hybrid storage system combining SSDs and HDDs by investigating how data sychronization affects I/O performance. We expect the optimized solution can achieve very good I/O performance without compromising reliability too much by minimal data discrepancy between the primary SSD and the backup HDD. To the best of our knowledge, this is the first work to investigate both I/O performance and data sychronization for reliability in a hybrid storage system using SSDs.

In PASS, I/O requests are performed on the high-performance, low-latency SSD first and then use asynchronous mirroring operations to duplicate the latest data from the SSD to the HDD for data reliability. In order to reduce data sychronization operations as much as possible, we use an I/O window to hold outstanding write I/O requests. Inside an I/O window, we coalesce smaller I/O requests into a larger I/O operation and use an ordered I/O queue to reduce disk seek and rotation times for shortening the duplication latency. Depending on the I/O characteristics of different workloads, we develop an adaptive policy to balance normal I/O process-

978-0-7695-4701-5/12 $26.00 © 2012 IEEE
DOI 10.1109/ISPA.2012.59

ing and mirroring operations. When outstanding duplication requests reach a threshold, foreground I/O processing will be blocked for a certain amount of time dynamically. As a result, asynchronous mirroring has more opportunities to perform duplication operations. We have designed and implemented a prototype system by developing a Linux block level driver on top of an HDD and an SSD which is a real implementation based on the source code of the Linux MD module. Experimental results show that PASS is an efficient hybrid storage system for SSDs and HDDs. More importantly, PASS can deliver more than 10 times higher I/O performance when tolerating less than 2% data discrepancy.

The rest of the paper is organized as follows. Section II presents the background and related work. The architecture of PASS is presented in Section III and detailed design and implementation are described in Section IV. In Section V, we evaluate PASS and discuss numerical results for multiple benchmarks. Finally, we conclude the whole paper in Section VI.

II. BACKGROUND AND RELATED WORK

Traditionally, data is protected by RAID technology. For example, RAID1 can tolerate single disk failure by data mirroring. For every write I/O request, it will be cloned into multiple identical write requests performed on the mirrored disks, respectively. In other words, cloned write requests need to be synchronized at the system level. As for a read I/O request, it will be forwarded to a close component disk by *read balance* algorithm [5]. In RAID1 architecture, we assume using two identical disks to construct a disk array. When one of the component disks fails, 100% of the data can be recovered by the other good disk. The overall I/O performance of RAID1 approximates the performance of single component disk. However, if component disks have different I/O response times, i.e. we use one SSD and one HDD to build a RAID1 storage system, the performance will be close to that of the slower disk. In other words, RAID1 cannot achieve good I/O performance for heterogeneous disks due to synchronous mirroring (called RAID1-SM thereafter).

Besides synchronous mirroring, write behind is another technique of RAID1 implementation to improve system performance (called RAID1-WB). In RAID1-WB, a memory buffer is allocated to keep a certain amount of outstanding write requests. A write request can be completed early as long as one of the acknownledgements from the component disks is received. Theoretically this kind of lazy update for the second component disk will improve RAID1 performance [6]. However, it is still not clear how much performance improvement can be achieved for a hybrid storage system by using RAID1-WB. In order to understand the performance chacteristics of RAID1 for a hybrid storage system, we have conducted an experiment to build a RAID1 storage system on top of an Intel SSD and a Seagate hard drive and measured the PostMark benchmark. As shown in Table I, the performance of both the RAID1-SM and RAID1-WB implementations is close to that of the HDD, which is much slower than that of the SSD. The preliminary results show that the naive implementations of RAID1 storage do not leverage the high-performance of SSD. We need to come up a new design for the hybrid storage to solve asymmetric performance charateristics of SSDs and HDDs.

TABLE I
PERFORMANCE COMPARISON OF HDD, SSD, RAID1-SM, AND RAID1-WB FOR POSTMARK

Types	Transaction Rate(/s)	Read(MB/s)	Write(MB/s)
HDD	277	7.73	9.44
SSD	2380	71.23	86.91
RAID1-SM	288	8.78	10.71
RAID1-WB	313	9.54	11.64

Considering the cost, performance, and reliability of SSDs, several efforts in the research community have been made to propose a hybrid storage system combining SSDs and HDDs. Kim et al [7] developed MixedStore to investigate performance and SSD lifetime for hybridizing SSDs and HDDs. Experiments show that it can reduce response time for Financial Trace by 71% as compared to an HDD-based system and extend by 33% the lifetime of SSD using adaptive wear-leveling. Wu et al [8] proposed a hybrid storage system by exploiting concurrency to improve latency and throughput. Payer et al [9] proposed Combo Drive to concatenate a smaller-capacity SSD with a larger-capacity HDD. Depending on file types or file access patterns, Combo drive redistributes files between the SSD and HDD to optimize I/O performance. Jo et al [10] presented a hybrid copy-on-write storage system that combines solid-state disks and hard disk drives for consolidated environments. They derived the performance benefits from fast random reads of SSD by storing read-only data in SSD, while written data are stored in HDD. Similarly, Koltsidas [11] et al advocated organizing SSDs and HDDs together, but redirecting read-intensive data to the SSDs and write-intensive data to the HDD for optimizing the SSD write performance. Chen et al [12] designed Hystor to integrate both low-cost HDDs and high-speed SSDs as a single virtual block device. By the prototype development, they uncovered many performance characteristics such as data layout between SSDs and HDDs, SSD write buffer management and proper chunk size for SSDs. However, most of these research works focus on achieving high performance for a hybrid storage system. Data backup and reliability in a hybrid system are still not well-investigated.

In addition to academic efforts for integration of SSDs and HDDs, there are also several storage products in industry that combine SSDs and HDDs. Typical products include SilverStone's HDDBoost [13], Seagate's Momentus XT [14], and RAIDON's Hybrid RunneR [15]. HDDBoost works like a static read cache to preload a portion of HDD data to the SSD when initialization. Because the SSD has good read performance, preloading can improve storage performance if future accesses are hit in the SSD [13]. However, HDDBoost is a simple implementation of a hybrid storage system. It does not consider data reliability. Moreover, it does not handle write requests. The performance improvement of HDDBoost

comes from static prefetching. If data blocks are not stored in the SSD or outdated, HDDBoost does not obtain any benefit from the high-performance SSD. The Seagate Momentus XT drive is a solid state hybrid with adaptive memory technology, enabling the drive to deliver SSD-like performance along with higher capacity. It uses 4GB NAND Flash memory as a cache for a high-capacity HDD [14]. Again, it also does not provide data protection capability in case of damage. In addition, Hybrid RunneR is a combination of 2.5 SSD and 2.5 HDD. It utilizes the advantage of SSD technology for a faster system performance and also uses the HDD for backing up the identical data from SSD regularly or simultaneously to prevent possible data loss caused by SSD failure [15]. However, it is still unknown when data blocks need to be synchronized between the SSD and the HDD and how much performance and reliability we can achieve. Our work can help to understand performance-reliability tradeoffs and pave a way to build an efficient hybrid storage system by utilizing high-performance SSDs.

III. Architecture of PASS

PASS works as a virtual block device under the kernel storage layer. It is composed of a flash-based SSD as the primary disk, an HDD as the backup disk, and a PASS I/O controller to manage two component disks. The I/O controller accepts block-level I/O requests from the kernel storage layer and forwards them to the primary disk. Later on it performs asynchronous mirroring to duplicate the latest updates from the primary disk to the backup disk. One may argue that the architecture of PASS will incur a large number of write requests to the primary SSD, which will reduce the lifetime of the SSD. It is true that using an SSD as the primary disk of a hybrid storage will generate more write requests to the SSD and flash-memory has limited erase cycles. Fortunately the issue of lifetime has been addressed by the design of FTL. The basic idea is to update out-of-place [16]. In this paper, we just consider an SSD as a black box and leverage its high performance for building a hybrid storage system.

In order to reduce duplication operations as much as possible, we maintain an I/O window to absorb overwrite requests and delay copying operations until the I/O window is full. Because normal I/O processing and background duplication run with different performance characteristics, an adaptive approach has been developed to balance them by dynamically blocking incoming I/O requests. Particularly when I/O requests arrive in a burst and the I/O window is filled very quickly, the duplication thread will accumulate a large number of outstanding operations. In order to keep data consistent and reduce data differences between the primary disk and the backup disk, we would like to speedup background duplication by blocking foreground I/O processing. The blocking time is proportional to the number of outstanding duplication operations. In other words, the more outstanding operations, the longer the normal I/O thread will be blocked.

On the other hand, foreground I/O processing may have a timeout mechanism from the application point of view. When an I/O request does not complete in an expected time, it will be treated as a failed I/O request. For example, TPC-C requires that at least 90% of transactions to complete within 80 seconds of their being queued for execution [17]. In a way, there is a maximally acceptable I/O response time for every I/O request. Therefore, PASS enforces a maximal blocking time for the I/O thread to prevent some I/O requests from producing a very long response time. This maximal blocking time is a system parameter and can be tuned depending on the I/O charaterization of different workloads.

IV. Design and implementation

PASS is implemented as a Linux block device driver to accept block I/O requests and forward them to the underlying SSD and HDD transparently. The implementation is based on Linux software RAID1 and device-mapper modules [18] including modification of read and write functions, multi-threading management, and I/O coalescing.

A. Modification of read and write functions

In the design of PASS, an SSD is considered as the primary disk. All I/O requests are redirected to the primary disk. Specifically, when a read request arrives, it will be forwarded to the SSD directly instead of choosing a close forwarding disk by the *read balance* algorithm in a RAID1 implementation because the SSD has much better read performance than an HDD and does not have mechanical movements. There is no need to execute the *read balance* function for read requests.

Write requests will also be forwarded to the SSD. When a write request is finished, a callback function will be invoked. In the callback function, we design a window structure to keep all write requests for duplication. When the I/O window is full, the duplication thread will be awakened to submit copying requests from the SSD to the HDD. In order to reduce kernel memory usage in our device driver, we only store the metadata of outstanding write requests including the starting and ending LBA addresses. Because I/O requests can be overlapped or repeated, we can merge and coalesce the write requests inside an I/O window to reduce unnecessary duplication operations. For example, since a data block may be overwritten multiple times, we only need to duplicate the latest update to the HDD and all previous write requests to that block can be ignored. Detailed implementation for I/O coalescing will be discussed shortly.

B. Multithreading management

In our implementation of PASS, there are three kernel threads to run with our device driver. The first one is I/O thread to handle normal I/O requests. Its main functions are to forward I/O requests to the underlying SSD by calling kernel function *generic_make_request* and to setup the callback function. The second kernel thread, duplication thread, is responsible for submitting duplication requests and releasing the I/O window. This thread is awakened by an I/O callback function when the size of the I/O window reaches a predefined threshold. Once all the duplication requests

have been submitted, the duplication thread makes the I/O window available for the I/O thread so that incoming I/O requests can be filled into the window again. Because the I/O window is a critical resource shared by both the I/O thread and the duplication thread, an exclusive lock has been implemented in the PASS driver. When the current I/O is a write request, the I/O metadata is added into the I/O window before finishing the callback function. Once the I/O window is full, the duplication thread will be awakened. During the running of the duplication thread, the number of outstanding duplication operations is checked first. If it exceeds a threshold, the I/O thread will be blocked by barrier synchronization. We implement a timeout mechanism to make sure that the I/O thread comes back to handle normal I/O requests. The blocking time is adjusted dynamically based on the queue length of outstanding duplication operations. In this paper, we use a simple formula to calculate the blocking time as follows:

$$\alpha = \frac{\text{the number of outstanding operations}}{\text{predefined threshold}}$$
$$blocking\_time = base\_time * min(\alpha, N)$$

From above formula, one can see there are *N* levels to dynamically calculate the blocking time of the I/O thread to balance the I/O thread and the duplication thread. *N* and predefined base_time are user-configurable for various workloads. When write I/O requests come in a burst, many outstanding duplication operations accumulate in the system, so α becomes a big number to block the I/O thread in a long time period. On the other hand, if write I/O requests arrive smoothly, the idle time among the I/O requests can be used to copy data blocks from the SSD to the HDD. If there are only a few or no outstanding duplication operations to accumulate, we block the I/O thread in a short time period or do not need to block at al. Depending on the different behaviors of the I/O workloads, the blocking time is adaptively adjusted to make a balance between normal I/O processing and data duplication so that we can make a performance-sychronization tradeoff for PASS.

In addition to the normal I/O processing and the duplication threads, the third thread duplicates data from the SSD to the HDD. In the current Linux kernel, *kcopyd* has already been implemented as a kernel thread providing the ability to copy a range of sectors from one block-device to one or more other block-devices, with an asynchronous completion notification. In this paper, we just use *kcopyd* to perform data copying by calling these provided kernel functions. It should be mentioned here that PASS needs to read the updated data from the SSD first and then write to the backup HDD. This is different from RAID1-WB because we only store the metadata of updated blocks instead of the whole data blocks to save kernel memory space. It is true that we need to pay the overhead for reading data from the SSD. The good thing is that high read performance of the SSD and I/O coalescing can make this additional overhead negligible.

C. I/O coalescing

When the duplication thread is awakened, all the write requests in an I/O window will be submitted to *kcopyd* for copying updated data from the SSD to the HDD. For every copying request, designated data blocks will be read from the SSD and written to the HDD. Because sequential write performance of the HDD is much higher than random write performance, if the submitted duplication requests follow the order of LBA addresses, the performance of writing data to the HDD will be significantly improved. We use an ordered list *tn_list* to keep all outstanding write requests. Each node of the list represents an outstanding duplication request that will be submitted to *kcopyd* thread when the length of the list reaches a predefined threshold. When a new write request comes in, *new_node* is constructed using the starting and ending LBA addresses.

It should be mentioned here that sorting and retrieving are expensive functions in kernel space. We maintain a current node in the I/O coalescing algorithm. When a new write request comes in, we start a search of *tn_list* with the current node instead of the beginning node of the whole list. By using a current node, *tn_list* is split into two segments: one is from the head node to the current node and the other is from the current node to the tail node. We only choose one segment to retrieve by comparing the LBA of the new request with that of the current node. Our experiments show this approach reduces the number of retrieval nodes significantly compared to that of retrieving the whole list because, on average, only the half of the nodes are traversed by the optimization.

V. Performance evaluation

We use an Intel SSD (SSDSA2SH032G1GN) and a Seagate HDD (ST3320418AS) to build a prototype of PASS. The prototype runs as a Linux module under Redhat Enterprise Linux Desktop 5 with the 2.6.18 Linux kernel. Using our developed device driver, we set up three benchmarks, IoMeter, PostMark, and TPC-C, to conduct experiments. For the experimental measurements, we use an I/O window size in terms of the number of sectors to evaluate the data difference between the primary SSD and the backup HDD, which is the size of possible data loss in case of data damage. Since the storage size in our experiments is 30GB, we use the I/O window size to estimate data loss probability approximately. For example, if the I/O window size is 400K sectors (512 bytes per sector), the possible data loss probability is 400K*512/30GB = 0.651%. In other words, if the primary SSD is damaged, we can recover at least 1-0.651%=99.349% of the data from the backup HDD. The percentage of possible data loss is 0.651%.

A. I/O workloads

IoMeter is an I/O subsystem measurement and characterization tool for single and clustered systems. It can generate entirely synthetic I/O workloads by setting I/O system parameters and collecting the performance results of its I/O operations. It is widely used in industry and the research community. In this paper, we generated two IoMeter workloads to test our

prototype by changing the proportion of read I/Os. For both workloads, the buffer size is set to 4KB and the access pattern is 50% random I/Os. Another benchmark we use is PostMark, which is a widely used file system benchmark tool developed by NetApp [19]. It was designed to simulate heavy small-file system loads for email, netnews, and e-commerce classes of applications. In our experiments, we ran the PostMark workload with 10,000 files and 50,000 transactions. The file size ranges from 4KB to 500KB. Read and write buffer sizes are set to 4KB.

In order to investigate how PASS affects database performance, we also use the TPC-C benchmark to test our prototype system. TPC-C is a well-known benchmark used to model the operational end of businesses where real-time transactions are processed [17]. We set up a Postgres database based on the implementation from TPCC-UVA [20]. 50 warehouses with 500 users on a 30GB storage volume are initialized first and then 10, 20, 30, and 40 warehouses are chosen for performance measurements.

B. Performance results

Our first experiment is to evaluate the IoMeter performance of PASS for the workload with 50% read requests. As shown in Figure 1, we plot IOPS (the number of I/O operations per second) with the I/O window size ranging from 256 to 400K sectors (512 bytes per sector) and take three cases of HDD, SSD, and RAID1-SM as references. From Figure 1, one can see PASS has much better performance than a single HDD and RAID1-SM. Specifically, when the I/O window size is 400K sectors, PASS can achieve up to 11 times performance speedup compared to a single HDD and 10 times higher than RAID1-SM. In other words, if we do not need 100% data synchronization and allow the data discrepancy (400K*512B/30GB= 0.651%) between the primary SSD and the backup HDD, we can have more than 10 times better performance than RAID1-SM storage. This performance is close to 77% of the peak IOPS of the SSD. When the primary SSD fails, we can still recover data from the backup HDD up to 1- 0.651%=99.349%. Another observation from Figure 1 is the performance of PASS increases as the I/O window size increases. This is expected because the discrepancy will increase as the I/O window size increases. We pay the price for the performance by compromising the data reliability because the data discrepancy will increase as the I/O window size increases so that the percentage of recoverable data will decrease when a fault occurs in the primary SSD.

The above experiment is based on RAID1-SM implementation. As we discussed before, write-behind is another implementation for RAID1 storage. For synchronous mirroring, every write request can be returned to the file system only after all cloned write requests are finished with a synchronous operation. Different from synchronous mirroring, write-behind RAID1 can come back to the upper layer early once I/O requests are finished on one of component disks. It can improve the performance of RAID1-SM by propagating the updates asynchronously. Since both RAID1-WB and PASS use asynchronous mirroring for the hybrid storage, we have conducted the second experiment to compare their performance. Figure 2 shows I/O throughputs in terms of IOPS for both RAID1-WB and PASS for multiple I/O window sizes ranging from 256 to 16K sectors. The last column in the figure is the reference performance for the HDD and RAID1-SM. As shown in Figure 2, PASS has better performance than RAID1-WB for all the cases. The improvement is up to 13% as the I/O window size increases from 256 to 16K sectors. Both RAID1-WB and PASS can produce twice the performance of the RAID1-SM for this benchmark program.

It should be pointed out here that both RAID1-WB and PASS achieve much better performance than RAID1-SM by trading off the data discrepancy because they perform duplication asychronously. In other words, PASS can have the same level of reliability as RAID1-WB. However, RAID1-WB uses kernel memory to keep outstanding updates including data blocks, and the current implementation can only support up to 16K sectors of the I/O window size due to limited kernel memory space. This is the limitation of the write-behind implementation. PASS does not have this limitation because we only store a very small amount of metadata and will read the data blocks on the fly when performing duplication from the primary SSD to the backup HDD. Although we need to pay the overhead for reading data from the SSD, fortunately high-performance SSD and I/O coalescing can counteract this overhead. As shown in Figure 1, the performance will be significantly increased when the I/O window size is greater than 16K sectors.

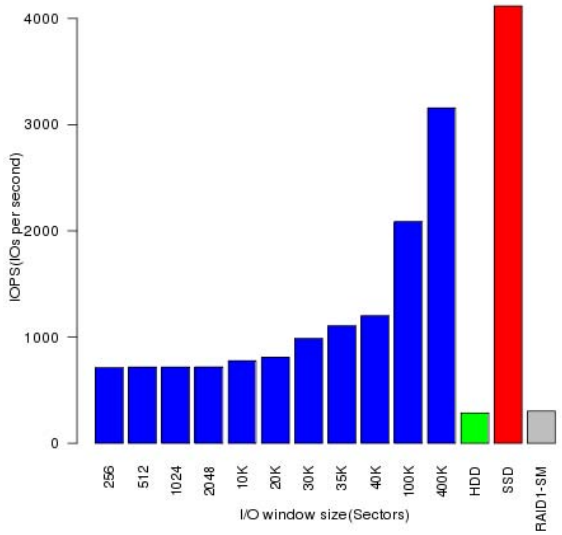

Fig. 1. I/O performance of PASS for 50% read IoMeter workload

Besides setting up experiments on 50% read IoMeter workload, we also ran performance experiments on the workload with 90% read requests. For this IoMeter workload, we have more impressive results. As shown in Figure 3, PASS has up to 12 times higher I/O throughtput than RAID1-SM when

compromising data reliability up to 0.651%. Compared to the workload of 50% reads, the throughtput for RAID1-SM does not increase much when the read ratio increases to 90%. The good thing is that the performance of PASS has significantly increased for this read-intensive workload. Specifically, when the I/O window size is 400K sectors, PASS achieves throughput of 4671.941 IOPS, which is close to 90% of the peak performance of the SSD. The reason is that fewer write requests will reduce duplication operations from the primary SSD to the backup HDD and also will increase the probability of hiding the latency of data copying by overlapping the foreground I/O processing and the background duplication. Again, Figure 3 shows a similar performance trend as Figure 1 so that the I/O performance will increase as the I/O window size increases.

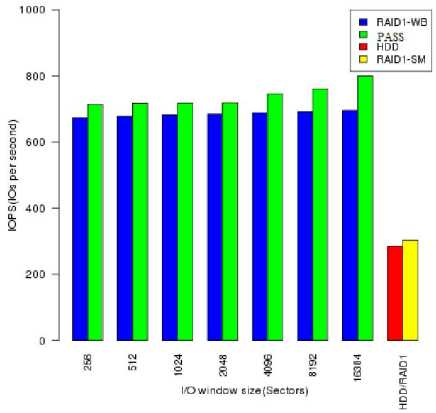

Fig. 2. Performance comparison of PASS and write-behind for the 50% write IoMeter workload

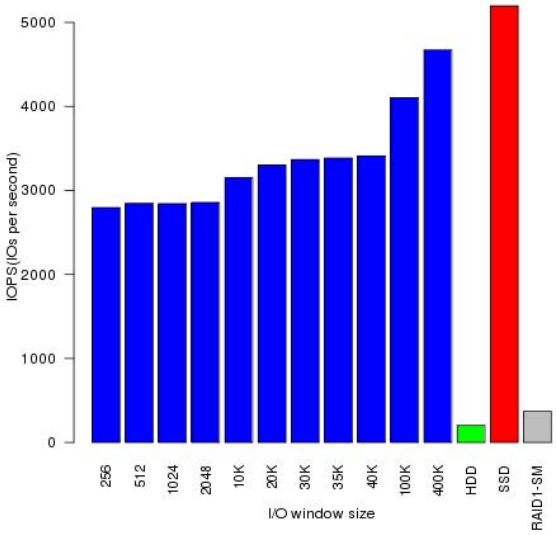

Fig. 3. I/O performance of PASS for the 90% read IoMeter workload

Figure 4 shows the performance results in terms of transaction rate for the PostMark benchmark. One can see that PASS has much better performance than RAID1-SM with up to 6 times higher transaction rate when tolerating data discrepancy up to 800K*512B/30GB=1.3%. Another observation from Figure 4 is that the transaction rate increases as the I/O window size increases from 1K to 800K sectors. This performance trend is consistent with the results of IoMeter benchmark. However, there is an exception when the I/O window size is less than 1K sectors. The possible reason for this abnormal behaviour is that there are fewer repeated or overlapped I/O requests for I/O coalescing if the I/O window size is too small so that we cannot totally hide the latency of synchronization operations from the primary SSD to the backup HDD.

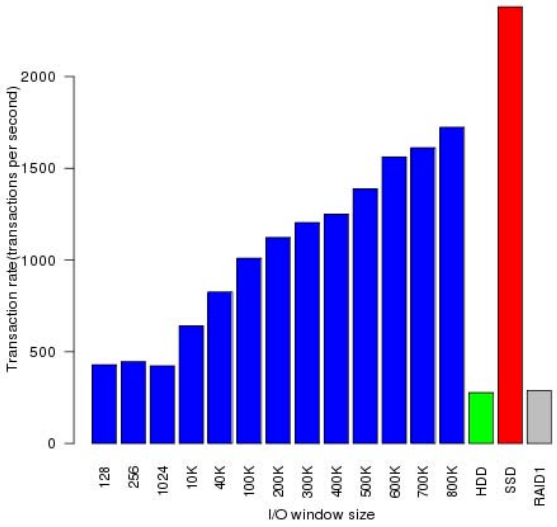

Fig. 4. I/O performance of PASS for PostMark benchmark

TABLE II
PERFORMANCE OF HDD, SSD, RAID1-SM, AND PASS FOR TPC-C

Types \ Warehouses	10	20	30	40
PASS (1K)	127.567	252.9	380.5	505.133
PASS (2K)	128.1	254.5	382.233	508.4
PASS (3K)	128.2	255	382.633	509.599
PASS (4K)	128.4	255.333	383.033	510.266
RAID1-SM	127.6	251.4	Failed	Failed
HDD	127.733	243.233	Failed	Failed
SSD	128.5	255.8	384.166	512.133

In addition to IoMeter and PostMark, we have also conducted experiments on a postgres database with 10, 20, 30, and 40 warehouses using the TPC-C benchmark. Table II shows TPC-C results in terms of tpmC (new-order transactions per minute). Different from the results of IoMeter and PostMark benchmarks, TPC-C does not produce a performance speedup when the I/O window size increases from 1K to 4K sectors for the same number of warehouses. It produces almost the same I/O performance as to PASS and RAID1-SM. The

possible reason for this performance behavior is that the TPC-C benchmark has enough idle time to hide the latency of data duplication from the primary SSD to the backup HDD because there is a period of random thinking time after submitting a transaction to the database. PASS will use this idle time to perform asynchronous mirroring. However, the IoMeter and PostMark workloads continuously issue I/O requests without thinking time. The second possible reason is that buffer caching and flushing policies of the database system reduce the storage load as much as possible and issue I/O requests smoothly to the storage system. As a result, the underlying storage system is not a bottleneck for the TPC-C benchmark so that the SSD has similar TPC-C performance to the HDD as shown in Table II. We will analyze and explain these unexpected performance behaviors by characterizing I/O burstiness in the next section.

Another observation from Table II is that the performance will increase as the number of warehouses increases from 10 to 40. This is because more users and bigger databases will generate many more I/O requests so that more transactions will be performed on the storage system. In addition, when the database increases up to 30 warehouses, the TPC-C tests on single HDD and RAID1-SM will fail because some transactions have very long response times which exceed the maximum threshold for the TPC-C benchmark. Fortunately, PASS can perform these TPC-C tests successfully because we design a timeout mechanism and an adaptive policy to deliver acceptable response time for every I/O request. In other words, PASS can support many more concurrent users and larger databases than the RAID1-SM and a single HDD, although there is no performance benefit for PASS.

C. I/O trace analysis

As we discussed previously, the TPC-C benchmark has different performance behaviors compared to IoMeter and PostMark. In this section, we collect block-level traces and analyze the I/O burstiness for the three benchmarks. Analysis of I/O burstiness is a widely used method to characterize and explain I/O performance behaviors [21]. Generally, if an I/O trace has stable temporal burstiness and spatial burstiness and spatio-temproal correlation, all the entropy plots are linear [21]. For example, if all the I/O requests come in a burst, all the arrival timestamps will be the same. The slope of the entropy curve is equal to 0. On the other hand, when the I/O requests arrive in an uniform distribution along time, the slope of the entropy curve is equal to 1. In other words, we can use the slope of the entropy curves to measure I/O burstiness. The more burstiness in an I/O trace, the smaller the slope of its entropy plot. For a given I/O sequence, we generate histograms first by designating the number of bins and then use maximum likelihood estimation to calculate the probabilities for every bin. Finally, we apply the classic entropy formula to estimate the entropy. All the I/O trace analyses have been done by developing a simple R program [22].

In this paper, we first use the Linux I/O trace tool, *blktrace*, to capture all time sequences of I/O requests for five different I/O workloads and then calculate the entropy in the time domain to measure temporal burstiness. These five I/O workloads are 50% read requests of IoMeter, 90% read requests of IoMeter, TPC-C with 10 warehouses, TPC-C with 20 warehouses, and PostMark with 50,000 transactions. Figure 5 shows entropy plots with a logarithmic scale for the five different I/O workloads. As shown in Figure 5, when the number of bins is less than 2^{10}, the postmark workload has the most bursty I/O requests. For the number of bins ranging from 2^{10} to 2^{15}, the two TPC-C workloads are more bursty than the other three workloads. When the number of bins exceeds 2^{15}, the PostMark workload shows a uniform distribution in arrival times, but the two IoMeter workloads are a little bit burstier than the two TPC-C workloads.

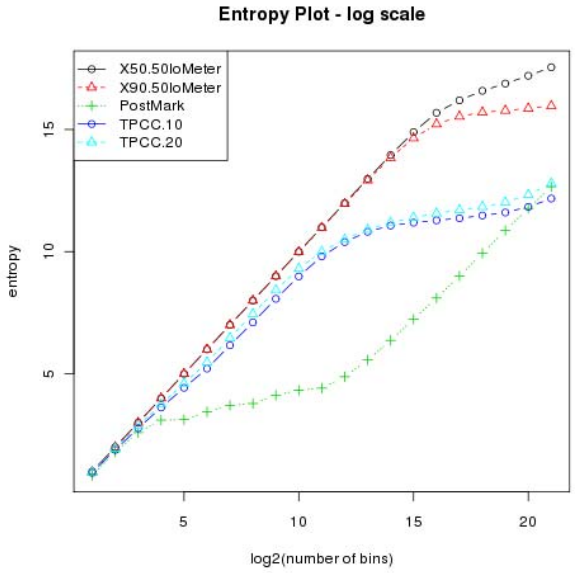

Fig. 5. Burstiness comparison for the five I/O workload using log scale entropy

Based on above results, we observe that the PostMark workload is burstier than the others over long periods of time (ie., with fewer numbers of bins). However, in the short term, the PostMark workload is uniformly distributed (larger numbers of bins). This observation means that the PostMark workload does not have enough idle time to hide duplication latency from the primary SSD to the backup HDD in the short term. Over the longer term, the PostMark workload has a much bigger time gap without any pending I/O requests. This does not help PASS because all pending duplication operations have already been finished. On the other hand, the IoMeter workloads are burstier at smaller time granularities (ie., greater than 2^{15}). The burstiness at this granularity means that the IoMeter workloads are idle more frequently. However, every idle period is so short that it is not enough to finish the I/O duplication from the primary SSD and the backup HDD. To explain this behavior consider that, if the number of bins is 2^{21}, the total experimental time is 30 minutes for the IoMeter workloads. Each idle period will take $30 * 60 * 1000/(2^{21}) = 0.8583$

ms roughly, but the average latency for a 7200RPM HDD is 4.17 ms [23]. In other words, this short idle time is not enough to finish a single I/O operation on the HDD so that the latency of the I/O duplication cannot be hidden. For this simple calculation, we ignore the latency of reading data from the primary SSD.

As for the two TPC-C workloads, they are burstier than the other workloads when the number of bins ranges from 2^{10} to 2^{15}. We make a simple estimate of the duration of an idle period as follows. Suppose the number of bins is equal to 2^{15}, one idle time will take $30 * 60 * 1000/(2^{15}) = 54.931$ ms. This indicates that this period of idle time among the I/O requests is enough to perform data duplication from the primary SSD to the backup HDD. As a result, we can explain that the I/O window size does not affect the performance of TPC-C workloads because the latency of the I/O duplication is already hidden by the idle time. However, PASS still allows the system to scale to large sizes, as described in Section V-B.

VI. Conclusion

In this paper, we have proposed a hybrid storage system named PASS to tradeoff between I/O performance and data sychronization by using SSDs. PASS includes a high-performance SSD and a traditional HDD to store mirrored data for reliability. All the I/O requests are redirected to the primary SSD first and then the updated data blocks are duplicated to the backup HDD asynchronously. In order to hide the latency of duplication operations, we use an I/O window to coalesce write requests and maintain an ordered I/O queue when copying data from the primary SSD to the backup HDD. Depending on different I/O workloads, we have developed an adaptive policy to balance the foreground I/O processing and background mirroring. We have implemented a prototype system of PASS by developing a Linux device driver and conducted experiments on the IoMeter, PostMark, and TPC-C benchmarks with five different configuration settings. Numerical results show that PASS can achieve up to 12 times the performance of the RAID1 storage for the IoMeter and PostMark workloads when tolerationg data discrepancy by less than 2% . More interestingly, although PASS does not have any performance benefit for the TPC-C benchmark, it can still make the system more scalable. In the end, we have explained the unexpected performance behaviors of the TPC-C benchmark by examining the entropy of the I/O traces.

Acknowledgment

This material is based upon work supported by the US National Science Foundation (NSF) under Grant ITR-0937060 to the Computing Research Association for the CIFellows Project. This work is also sponsored in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-0934396, IIP-1127829, and member companies. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF. The authors also would like to thank the anonymous reviewers for the valuable comments that improved the quality of this paper.

References

[1] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drivers," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 295–304.

[2] M. Dunn and A. L. N. Reddy, "A new i/o scheduler for solid state devices," Texas A&M University-College Station, Tech. Rep. TAMU-ECE-2009-02, April 2009.

[3] T. Pritchett and M. Thottethodi, "Sievestore: a highly-selective, ensemble-level disk cache for cost-performance," *SIGARCH Comput. Archit. News*, vol. 38, pp. 163–174, June 2010.

[4] J. Park, H. Lee, S. Hyun, K. Koh, and H. Bahn, "A cost-aware page replacement algorithm for nand flash based mobile embedded systems," in *Proceedings of the seventh ACM international conference on Embedded software*, 2009, pp. 315–324.

[5] R. W. Henderson and P. Blankenbaker, "Linux software raid," http://nst.sourceforge.net/nst/docs/user/ch14.html, 2003.

[6] E. Varki, A. Merchant, J. Xu, and X. Qiu, "Issues and challenges in the performance analysis of real disk arrays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, pp. 559–574, June 2004.

[7] Y. Kim, A. Gupta, B. Urgaonkar, P. Berman, and A. Sivasubramaniam, "Hybridstore: A cost-efficient, high-performance storage system combining ssds and hdds," in *Proceedings of MASCOTS 2011*, Jul 2011.

[8] X. Wu and A. L. N. Reddy, "Exploiting concurrency to improve latency and throughput in a hybrid storage system," in *Proceedings of MASCOTS 2010*, Miami Beach, Floroda, August 2010.

[9] H. Payer, M. A. Sanvido, Z. Z. Bandic, and C. M. Kirsch, "Combo Drive: Optimizing cost and performance in a heterogeneous storage device," in *Proc. Workshop on Integrating Solid-state Memory into the Storage Hierarchy (WISH), co-located with ASPLOS*, 2009.

[10] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "Ssd-hdd-hybrid virtual disk in consolidated environments," in *Euro-Par 2009 Parallel Processing Workshops*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6043, pp. 375–384.

[11] I. Koltsidas and S. D. Viglas, "Flashing up the storage layer," *Proc. VLDB Endow.*, vol. 1, pp. 514–525, August 2008. [Online]. Available: http://dx.doi.org/10.1145/1453856.1453913

[12] F. Chen, "On performance optimization and system design of flash memory based solid state drives in the storage hierarchy," Ph.D. dissertation, The Ohio State University, Columbus, Ohio, 2010.

[13] SilverStone, "SST-HDDBOOST manual," http://www.silverstonetek.com/downloads/Manual/storage/Multi-HDDBOOST-Manual.pdf, 2010.

[14] Seagate, "Momentus xt: Product manual," http://www.seagate.com/staticfiles/support/disc/manuals/notebook/momentus/XT/100610268b.pdf, 2010.

[15] RAIDON, "HyBrid RunneR iH2420-2S-S2: User Manual," http://www.raidon.com.tw/user_file/000518.pdf, 2010.

[16] A. Gupta, Y. Kim, and B. Urgaonkar, "Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '09, 2009, pp. 229–240.

[17] T. P. P. Council, "TPC benchmark C–Standard Specification," http://www.tpc.org/tpcc/spec/tpcc_current.pdf, 2010.

[18] Wikipedia, "Device mapper," 2010, [Online; accessed 9-April-2010]. [Online]. Available: http://en.wikipedia.org/wiki/Device_mapper

[19] J. Katcher, "Postmark: A new file system bench-mark," Network Appliance, Tech. Rep. 3022, 1997.

[20] D. R. Llanos, "TPCC-UVA: An open-source TPC-C implementation for global performance measurement of computer systems," *ACM SIGMOD Record*, December 2006, iSSN 0163-5808.

[21] M. Wang, A. Ailamaki, and C. Faloutsos, "Capturing the spatio-temporal behavior of real traffic data," *Perform. Eval.*, vol. 49, pp. 147–163, September 2002. [Online]. Available: http://dx.doi.org/10.1016/S0166-5316(02)00108-6

[22] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2006, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[23] Wikipedia, "Acess time," 2010, [Online; accessed 17-November-2004]. [Online]. Available: http://en.wikipedia.org/wiki/Access_time