Managing Storage Space in a Flash and Disk Hybrid Storage System

Xiaojian Wu, and A. L. Narasimha Reddy
Dept. of Electrical and Computer Engineering
Texas A&M University

Abstract

This paper considers the problem of efficiently managing storage space in a hybrid storage system employing flash and disk drives. The flash and disk drives exhibit different performance characteristics of read and write behavior. We propose a technique for balancing the workload properties across flash and disk drives in such a hybrid storage system. We consider various alternatives for managing the storage space in such a hybrid system and show that the proposed technique improves performance in diverse scenarios. The presented approach automatically and transparently manages migration of data blocks among flash and disk drives based on their access patterns. We implemented a prototype storage system employing our ideas. This paper presents the design and an evaluation of the proposed approach through realistic experiments.

I. Introduction

Novel storage devices based on flash memory are becoming available with price/performance characteristics different from traditional magnetic disks. Many manufacturers have started building laptops with these devices. While these devices may be too expensive (at this time) for building larger flash-only storage systems, storage systems incorporating both flash-memory based devices and magnetic disks are becoming available.

Traditionally, storage systems and file systems have been designed considering the characteristics of magnetic disks such as the seek and rotational delays. Data placement, retrieval, scheduling and buffer management algorithms have been designed to take these characteristics into account. When both flash and magnetic disk drives are employed in a single hybrid storage system, a number of these policies may need to be revisited.

Flash based storage devices exhibit different characteristics than magnetic disk drives. For example, writes to flash devices can take longer than magnetic disk drives while reads can finish faster. The flash drives have no seek and rotational latency penalties unlike their magnetic counterparts, but have a limit on the number of times a block can be written. Flash drives also typically have more uniform performance (especially for reads) depending on file size, where magnetic disks typically perform better with larger file sizes. The write performance of flash drives can experience much larger differences in peak to average completion times due to block remapping done in the Flash Translation Layer (FTL) and necessary erase cycles to free up garbage blocks. These differences in characteristics need to be taken into account in hybrid storage systems in efficiently managing and utilizing the available storage space.

Fig 1 shows the time taken to read, write a data block from a disk drive and two flash drives considered in this study. We read blocks of specific size from a random offset in the drive to obtain this performance data. As can be seen from the data, the flash and disk drives have different performance for reads and writes. First, flash drive is much more efficient than the magnetic disk for small reads. While flash drive read performance increases with the read request size, the magnetic disk performance improves considerably faster and at larger request sizes, surpasses the performance of the flash devices. Small writes have nearly the same performance at both the devices. As the write request size grows, the magnetic disk provides considerably better performance than the flash device. the disk drive is more efficient for larger reads and writes. These characteristics are observed for a 250GB, 7200 RPM Samsung SATA magnetic disk (SP2504C), a 16GB Transcend SSD (TS16GSSD25S-S) and a 32GB MemoRight GT flash drives. While different devices may exhibit different performance numbers, similar trends are observed in other drives.

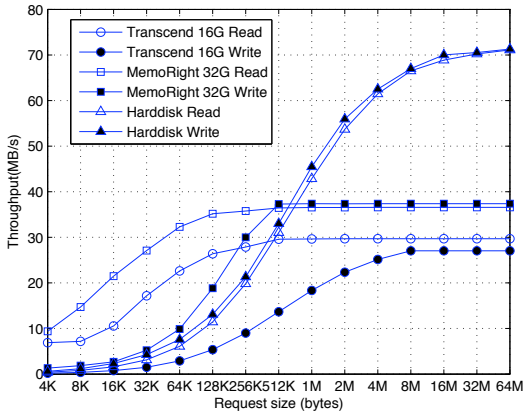


Fig. 1. Read, write performance of different devices

As can be seen from this data, requests experience different performance at different devices based on the request type (read or write) and based on the request size (small or large). This work is motivated at managing storage space to maximize performance in a hybrid system that utilizes such devices together.

Various methods have been proposed to compensate and exploit diversity in device characteristics. Most storage systems use memory for caching and manage memory and storage devices together. Most of these approaches deal with devices of different speeds, storing frequently or recently used data on

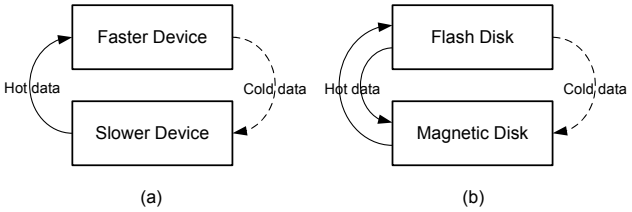


Fig. 2. (a)Migration paths in traditional hybrid storage systems. (b)Migration paths in space management layer.

the faster devices. However, the flash and disk storage devices have asymmetric read/write access characteristics, based on request sizes and whether the requests are reads or writes. This asymmetry makes this problem of managing the different devices challenging and interesting in a flash+disk hybrid system. In order to accommodate these characteristics, in this paper, we treat the flash and disk devices as peers and not as two levels in a storage hierarchy.

Our approach is contrasted with traditional approach of a storage hierarchy in Fig. 2. In a traditional storage hierarchy, hot data is moved to the faster (smaller) device and cold data is moved to the larger (slower) device. In our approach, the cold data is still moved to the larger device. However, the hot data may be stored in either device because of the performance asymmetry. For example, hot data that is mostly read may be moved to the Flash device while large files that are mostly written may be moved to the disk drive.

Flash technology is advancing rapidly and device performance characteristics differ from manufacturer to manufacturer, from one generation to the next. This warrants that a solution to managing the space across the devices in a hybrid system should be adaptable to changing device characteristics.

We consider two related issues in this paper for managing space across flash and disk drives in a hybrid system. The first issue is allocation. Where should data be allocated? Should we first allocate data on flash before starting to use the disk drive? Should we employ striping across the two devices until space constraints force us to do something else? Clearly, these decisions will have implications on load on different devices, their utilization and the performance of the storage system.

A second issue we consider in this paper is this: given a block allocation, can we adapt the storage system to better utilize the devices in a hybrid system? We approach this problem through data redistribution or migration of data from one device to another device to match the data access characteristics with the device characteristics. For example, if a data block has a higher number of reads compared to writes, it may be better suited to flash device and moving it from its currently allocated disk drive to flash drive may improve its performance. Similarly, if a file and its blocks are sequentially written and read, it may be better located on the magnetic disk without any loss in performance (compared to the flash device) while preserving space on the smaller flash device for other blocks. In this paper, we focus on the problem of data redistribution (through migration), in storage systems built out of flash and magnetic drives.

We propose a measurement-driven approach to migration to address these issues. In our approach, we observe the access characteristics of individual blocks and consider migrating individual blocks, if necessary, to another device whose characteristics may better match the block access patterns. Our approach, since it is measurement driven, can easily adapt to different devices and changing workloads or access patterns.

In this paper, we study the two related issues of allocation and data migration among the flash and disk drives to manage these diverse devices efficiently in a hybrid system. We make the following significant contributions in this paper: (a) we propose a technique for managing data efficiently in a hybrid system through dynamic, measurement-driven data migration between flash and disk drives, (b) we study the impact of different allocation decisions in such a system, (c) we show that the performance gains possible through flash devices may be strongly contingent on the workload characteristics observed at the storage system and (d) that flexible block management in a hybrid system is possible through a demonstration of results from a Linux-based prototype.

The remainder of this paper is organized as follows. Section II gives details of the related work. Section III describes the the proposed approach to managing the space in a hybrid system. Section IV presents our prototype implementation and its evaluation. Section V presents the results of the evaluation. Section VI provides a discussion of the results and the future work. Section VII concludes the paper.

II. Related Work

Data (file) migration has been studied extensively earlier in systems where disks and tapes are used [3], [7], [8], [18], [19]. HP's AutoRAID system considered data migration between a mirrored device and a RAID device [1]. In these systems, hot[1] data is migrated to faster devices and cold data is migrated to slower devices to improve the access times of hot data by keeping it local to faster devices. When data sets are larger than the capacity of faster devices in such systems, thrashing may occur. Some of the systems detect thrashing and may preclude migration during such times [1]. Adaptive block reorganization to move hot blocks of data to specific areas of disks has been studied in the past [35], [36].

Our work is measurement driven and considers migration among flash and disk drives with different read/write characteristics amid dynamically changing conditions at the devices. Data migration can be decided by read/write access patterns and loads at the different devices. In the work considered here, data can move in both directions from flash to disk and disk to flash for performance reasons. Second, the realized performance in our system depends on read/write characteristics as well as the recency and frequency of access. Characteristics of data migration can be much different in our system compared to earlier hierarchical systems.

Aqueduct [13] takes a control-theoretic approach to data migration among storage devices without significantly impacting the foreground application work. Aqueduct uses I/O request

[1]Data that is currently being accessed or active is called "hot" data and data that is not currently being accessed or inactive is called "cold" data.

response time as a performance measure, but there are significant differences from our work. First, Aqueduct is guided by static policy decisions of the system administrator unlike our dynamic choice of blocks during run time. Data distribution and migration issues are considered in [14], [15], [16] as a result of configuration changes due to additions and removal of disks in large scale storage systems. Adaptive file set migration for balancing the metadata workload among servers in shared-disk file systems is considered in [17]. Observed latencies were used for migration. Our work is different from this work: (a) Our system is dealing with migrating block-level data in contrast to file-level data (e.g., file sets), and (b) we consider read and write requests separately.

Flash storage has received much attention recently. A number of studies have been done on wear leveling and maintaining uniformity of blocks [30], [31], [32]. Studies have been done on block management and buffer management [27], [28], [29], [34]. File system design issues for flash storage have been considered in [33] and file translation layers (FTLs) are employed to mimic log structured file systems in current flash storage devices. Recently, employment of flash in a database workload is considered [26]. This work considered static measurement of device performance in deciding on migration as opposed to our method of employing dynamically measured response times. Dynamic measures are employed in managing power consumption in storage systems in [12]. Flash memory is being used as a cache to improve disk drive performance [e.g., 25]. Flash memory has been studied for reducing the power consumption in a storage system [40].

File system based approaches for flash and non-volatile storage can be found, for example in [38], [39]. These systems can take file level knowledge (data types and their typical access behavior)into account which may not be available at the device level. We consider a device level approach here and we manage storage across the devices at a block or chunk level instead of at the file level. Second, our approach can potentially, leave parts of a file on the flash drive and parts of the file on a disk drive depending on the access patterns, which is not possible in file-based approaches. In addition, the device level approach may obviate the need for new file systems or changes to every file system that employs a flash drive. It is not our intent to argue that the device level approach is better than the file system approach or vice versa, but that they are based on different assumptions and lead to different design points and benefits.

Data migration is considered previously in networked devices [6], in large data centers [4], [5]. This body of work considers optimization of data distribution over longer time scales than what we consider here. This work considers migration of datasets across different systems unlike our focus here on migration of data blocks within a single system. Our work is inspired by much of this earlier work.

III. Measurement-driven Migration

Our general approach is to pool the storage space across flash and disk drives and make it appear like a single larger device to the file system and other layers above. We manage the space across the different devices underneath, transparent to the file system. We allow a file block on one device to be potentially moved later to another device that better matches the access patterns of that data block. We also allow blocks to move from one device to another device as workloads change in order to better utilize the devices.

In order to allow blocks to be flexibly assigned to different devices, an indirection map, containing mappings of logical to physical addresses, needs to be maintained. Every read and write request is processed after consulting the indirection map and determining the actual physical location of data. Similar structures have been used by others [1], [9], [10]. In our system, as the data migrates from one device to another device, there is a need to keep track of the remapping of the block addresses. When data is migrated, the indirection map needs to be updated. This is an additional cost of migration. We factor this cost into the design, implementation and evaluation of our scheme. In order to reduce the costs of maintaining this map and updates to this map, we consider migration at a unit larger than a typical page size. We consider migration of data in chunks or blocks of size 64KB or larger.

We keep track of access behavior of a block by maintaining two counters, one measuring the read and the other write accesses. These counters are a "soft" state of the block i.e., the loss of this data is not critical and affects only the performance, but not the correctness of data accesses. This state can be either maintained in memory or on disk depending on the implementation. If memory is employed, we could maintain a cache of recently accessed blocks for which we maintain this state, in order to limit memory consumption. It is also possible to occasionally flush this data to disk to maintain this read/write access history over longer time. In our implementation, we maintain this information in memory through a hash table. This table is only as large as the set of recent blocks whose access behavior we choose to remember and is not proportional to the total number of data blocks in the system. We employed two bytes for keeping track of read/write frequency separately per chunk (64Kbytes or higher). This translates into an overhead of 0.003% or lower or about 32KB per 1GB of storage. We limited the memory consumption of this data structure by deallocating information on older blocks periodically.

We employ the read/write access counters in determining a good location for serving a block. A block, after receiving a configured minimum number of accesses, can be considered for migration or relocation. This is to ensure that sufficient access history is observed before the block is relocated. We explain below how a relocation decision is made. If we decide to relocate or migrate the block, the read/write counters are reset after migration in order to observe the access history since the block is allocated on the new device.

We took a general approach to managing the device characteristics. Instead of characterizing the devices statically, we keep track of device performance dynamically. Every time a request is served by the device, we keep track of the request response time at that device. We maintain both read and write performance separately since the read, write characteristics can be substantially different as observed earlier. Dynamically

measured performance metrics also allow us to account for load disparities at different devices. It also allows a single technique to deal with diverse set of devices without worrying about any configuration parameters.

Each time a request is sent to the device, a sample of the device performance is obtained. We maintain an exponential average of the device performance by computing average response time = 0.99 * previous average + 0.01 * current sample. Such an approach is used extensively in networking, in measuring round trip times and queue lengths etc [22]. Such exponential averaging smooths out temporary spikes in performance while allowing longer term trends to reflect in the performance measure. For each device i, we keep track of the read r_i and write w_i response times. We consider all request sizes at the device in computing this average response time to reflect the workload at that device in our performance metric. As we direct different types of request blocks to different devices, the workloads can be potentially different at different devices.

Given a block j's read/write access history through its access counters R_j and W_j and the device response times, we use the following methodology to determine if a block should be moved to a new location. The current average cost of accessing this block j in its current device i, C_{ji}= $(R_j * r_i + W_j * w_i)/(R_j + W_j)$. The cost of accessing a block with similar access patterns at another device k, C_{jk} (computed similarly using the response times of device k) are compared. If $C_{ji} > (1+\delta) * C_{jk}$, we will consider this block to be a candidate for migration, where $0 < \delta$ is a configurable parameter. We experimented with different values of δ in this study. A larger value for δ demands a greater performance advantage before moving a block from one device to another device.

In general, when there are several devices which could provide better performance to a block, a number of factors such as the load on the devices, storage space on the devices and the cost of migration etc. can be considered for choosing one among these devices.

Migration could be carried out in real-time while normal I/O requests are being served and during quiescent periods when the devices are not very active. The migration decision could be made on a block by block basis or based on looking at all the blocks collectively at once through optimization techniques. Individual migration and collective optimization techniques are complementary and both could be employed in the same system. We primarily focus on individual block migration during normal I/O workload execution in this paper. Later, we plan to incorporate a collective optimization technique that could be employed, for example, once a day.

There are many considerations to be taken into account before a block is migrated. The act of migration increases the load on the devices. This could affect device performance. Hence, it is necessary to control the rate of migration. Second, a fast migration rate, may result in moving data back and forth, causing oscillations in workloads and performance at different devices. In order to control the rate of migration, we employ a token scheme. The tokens are generated at a predetermined rate. Migration is considered only when a token becomes available. In our current implementation, we experimented with a conservative, static rate of generating tokens.

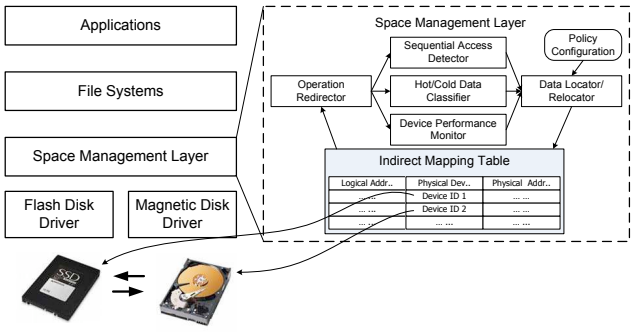


Fig. 3. Architecture of space management layer.

When a block is migrated from one device i to another device k, the potential cost of this migration could be $r_i + w_k$, r_i for reading the block from device i and w_k for writing the block to its new location on the device k. In order to reduce the costs of migration, we only consider blocks that are currently being read or written to the device, as part of normal I/O activity. Migrating non-active blocks is carried out by a background process during quiescent periods.

Depending on the initial allocation, there may be several blocks that could benefit from migration. A number of strategies could be employed in choosing which blocks to migrate. We maintain a cache of recently accessed blocks and migrate most recently and frequently accessed blocks that could benefit from migration. When ever a migration token is generated, we migrate a block from this cached list. The cached list helps in utilizing the migration activity to benefit the most active blocks.

Migration is carried out in blocks or chunks of 64KB or larger. Larger block size increases migration costs, reduces the size of the indirection map, can benefit from spatial locality or similarity of access patterns. We study the impact of the block size on the system performance. We investigate migration policies that consider the read/write access patterns and the request sizes.

We study several allocation policies since the allocation policies could not only affect the migration performance, but can also affect the system performance significantly (even without migration). These policies include (a) allocation of all the data on flash while it fits; allocating the later data on magnetic disk when flash device becomes full. (b) allocation of all the data on the disk drive, and (c) striping of data across both flash and disk drives.

We also consider a combination of some of these policies when possible. We consider the impact of migration along with the allocation policies in our study.

IV. Implementation

We developed a Linux kernel driver that implements several policies for migration and managing space across flash and disk drives in our system. The architecture and several modules within the space management layer are shown in Fig. 3. The space management layer sits below the file systems and

above the device drivers. We implemented several policies for detecting access patterns of blocks. The sequential access detector identifies if blocks are being sequentially accessed by tracking the average size of sequential access to each block. The device performance monitor keeps track of the read/write request response times at different devices. The hot/cold data classifier determines if a block should be considered hot. The indirection map directs the file system level block addresses to the current location of those blocks on the underlying flash and disk devices. The indirection map is maintained on the hard disk, a memory copy of it allows faster operations. The block access characteristics, in our implementation, are only maintained in memory. The block access counters are initialized to zero, both on bootup and after a migration.

In the experimental environment, the NFS server was a commodity PC system equipped with an Intel Pentium Dual Core 3.2GHz processor, 1GB of main memory. The magnetic disk used in the experiments was one 7200 RPM, 250G SAMSUNG SATA disk (SP2504C), the flash disk drives are a 16GB Transcend SSD (TS16GSSD25S-S), and a 32GB MemoRight GT drive, which were connected to Adaptec SCSI Card 29160 through a SATA-SCSI converter(ADSALVD160). All the NFS clients are simulated by one load generator in the environment.

The operating system on the NFS server was Fedora 9 with a 2.6.21 kernel, and the file system used was the Ext2 file system. The hybrid storage system is connected to the server and a number of other PCs are used to access the server as clients. We cleaned up the file system before each experiment (using file system mkfs command). Device level performance is measured using processor jiffies.

The next section presents performance evaluation and comparison of different policies.

V. Evaluation

A. Workload

We used multiple workloads for our study. The first workload, SPECsfs benchmark, represents file system workloads. The second workload, Postmark benchmark, represents typical access patterns in an email server. We also use IOzone [23] benchmark to create controlled workloads at the storage system in order to study the impact of read/write request distributions on the performance of the hybrid system.

SPECsfs 3.0 is the SPEC's benchmark for measuring NFS file server's performance [20]. This synthetic benchmark generates an increasing load of NFS operations against the server being evaluated and measures its response time and the server throughput as load increases. The operations of the benchmark consists of small metadata operations and reads and writes. A large percentage of the read/write operations, 85%, in this workload, are for small files in the range of 1-15KB.

SPECsfs reports a curve of response time vs. *delivered* throughput (not offered load). The system performance is measured by *base response time*, the *slope* of the response time vs. throughput curve and its *throughput saturation point*. When the offered load exceeds the *throughput saturation point*, the *delivered* throughput will decrease as requests timeout. Therefore, the reported curves usually go backward at the end. The speed of the server and client processor, the size of file cache, and the speed of the server devices determine these measured characteristics of the server [21]. We employed SPECsfs 3.0, NFS version 3 using UDP. At higher loads, the delivered throughput can decrease as requests time out if not completed within the time limit of the benchmark (50ms).

The SPECsfs benchmark exhibits a read write ratio of roughly 1:4 in our experimental environment, i.e., for every read request, roughly 4 write requests are seen at the storage system. In order to study the impact of different workloads, we also employed IOzone. IOzone is a synthetic workload generator. We employed 4 different processes to generate load. Each process can either read or write data. By changing the number of processes reading or writing, we could control the workload read/write ratio from 100%, 75%, 50%, 25% and 0%. We employed Zipf distribution for block access to model the locality of accesses. In this test, we also bypassed the cache to keep a careful control of the read/write mix at the storage system.

We use Postmark [37] as a third workload. Postmark is an I/O intensive benchmark designed to simulate the operation of an e-mail server. The lower bound of the file size is 500 bytes, and upper is 10,000 bytes. In our Postmark tests, we used Postmark version 1.5 to create 40,000 files between 500 bytes and 10 kB and then performed 400,000 transactions. The block size was 512 bytes, with the default operation ratios and unbuffered I/O.

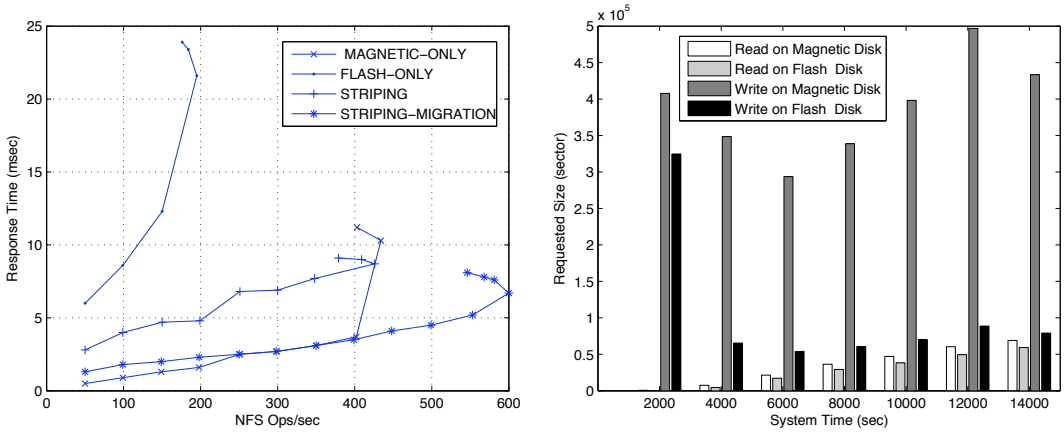
We expect these three workloads generated by SPECsfs, IOzone and Postmark to provide insights into the hybrid system performance in typical file system workloads and in other workloads with different read/write request distributions.

B. Benefit From Migration

In the first experiment, we evaluated the benefit from the measurement-driven migration. We compared the performance of the following four policies:

- **FLASH-ONLY:** Data is allocated on the flash disk only.
- **MAGNETIC-ONLY:** Data is allocated on the magnetic disk only.
- **STRIPING:** Data is striped on both disks.
- **STRIPING-MIGRATION:** Data is striped on and migrated across both disks.

The results are shown in Figure 4(a) for Transcend flash drive. Since there are more write requests in the workload than read requests, and write performance of the flash disk is much worse than that of the magnetic disk, the response times for FLASH-ONLY are longer than those for MAGNETIC-ONLY. Both these systems utilize only one device. The performance of STRIPING is between these two systems even though we get benefit from utilizing both the devices. The performance of the hybrid system is again impacted by the slower write response times for the data allocated on the flash device. Ideally, the addition of a second device should improve performance (which happens with the faster MemoRight flash drive as shown later).



(a) (b)

Fig. 4. (a)Benefit from measurement-driven migration. (b)Request distribution in measurement-driven migration.

As can be seen from Figure 4(a), the measurement-based migration has the throughput saturation point at 600 NFS operations/sec, that is much better than 426 NFS operations/sec in the STRIPING policy, and 434 NFS operations/sec in the MAGNETIC-ONLY policy. This improvement benefits from the data redistribution which matches the read/write characteristics of blocks to the device performance.

Figure 4(b) shows the request distribution in different system periods. At the first, beginning at 2000 seconds, the number of the write requests directed to the magnetic disk and to the flash disk are quite close. However, over time, more and more write-intensive data are migrated to the magnetic disk, resulting in more write requests at the magnetic disk. For example, in Figure 4(b), at the system time between 12000 and 14000 seconds, there are 433343 sectors written to the magnetic disk while only 79043 sectors are written to the flash disk (i.e. nearly 5.5 times as many sectors are written to disk compared to flash), while the read request sizes to the devices are close to each other at 69011 and 59390 sectors respectively. This request pattern at the end of the simulation shows that our approach is succeeding in redistributing write-intensive blocks to the magnetic disk even though the initial allocation of blocks distributes these blocks equally on the two devices.

This experiment shows that the measurement-driven migration can effectively redistribute the data to the right devices and help decrease the response time while improving the throughput saturation point of the system.

C. Sensitivity study

1) Impact of Migration Threshold Parameter δ: As explained earlier, δ is a configurable parameter, that controls how much performance advantage is expected (after migration) for the accesses to the block that is being migrated. In this experiment, we evaluated the impact of using different δ values. Generally, the smaller the δ, the higher the probability that the chunk will be migrated. As a result, too small a value of δ can cause higher migration. On the other hand, if the value is too large, the efficiency of the migration can be weakened as the data can not be remapped to the right device. Based on the results from the experiment shown in Figure 5(a), the value $\delta = 1$ is chosen for the rest of the tests.

2) Impact of Chunk Size: Figure 5(b) shows the impact of using the different chunk sizes. Chunk sizes of 64KB and 128KB had nearly the same performance at various loads while a larger chunk size of 256KB showed worse performance. In all the following experiments, we used the chunk size of 64KB.

D. Comparison with 2-harddisk Device

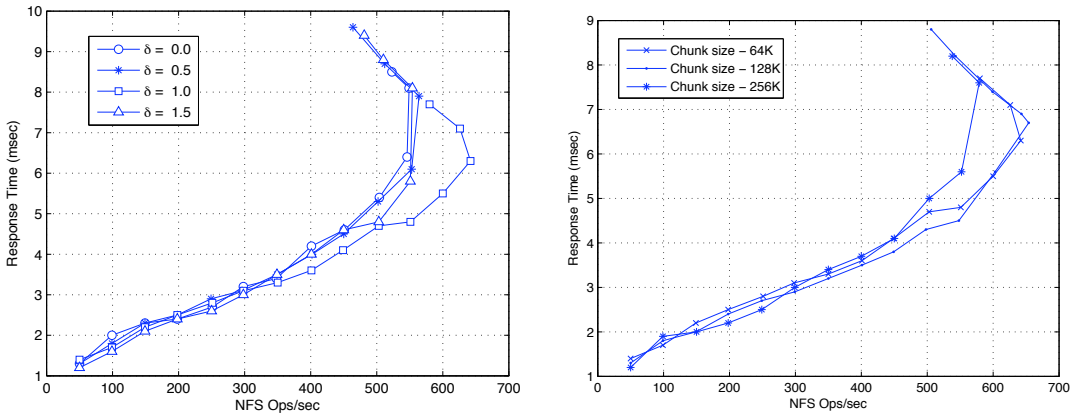
In this experiment, we compared the performances of the hybrid storage system and a 2-hard disk striping storage system (data is striped on two hard disks and no migration is employed). First, we used SPECsfs 3.0 to generate the workload.

Figure 6(b) shows the comparison of the MemoRight-based hybrid system against the 2-disk system. It is observed that the hybrid system outperforms the 2-harddisk striping storage system, achieving nearly 50% higher throughput saturation point. The hybrid storage system delivers higher performance, fulfilling the motivation for designing such a system.

Figure 6(a) shows the comparison of the Transcend-based hybrid system against the 2-disk system. It is observed that the 2-harddisk striping storage system works better than the hybrid drive on both the saturation point and the response time.

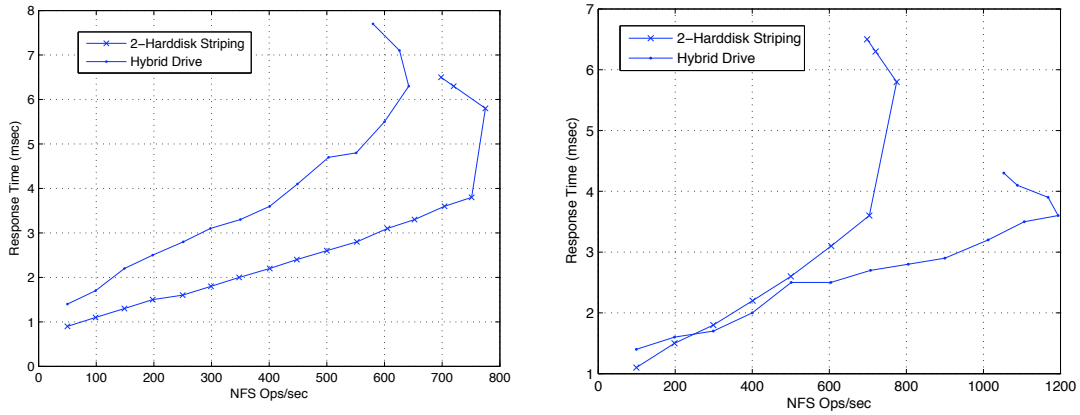
We used IOzone to generate workloads with different read/write ratio to find out what kind of workloads are more suitable for the hybrid storage system. In this experiment, we employed four processes to generate workload at the storage system. By varying the number of processes doing reads vs. writes, we could create workloads that 100% writes, to 75%, 50%, 25% or 0% write workloads (0R4W, 1R3W, 2R2W, 3R1W and 4R0W). We employed a Zipf distribution for data accesses in each process and bypassed the cache to maintain a controlled workload at the storage system.

The results are shown in Figure 7. Each workload name consists of <*number of reading processes* ***R*** *number of writing processes* ***W***>. Each process reads from or writes to a 512MB file according to a Zipf distribution. We adjusted the number of processes to control the ratio of read/write requests to the device. To bypass the buffer cache, we used direct I/O. While the 2-harddisk system did not employ migration, we tested the hybrid system with two policies STRIPING and STRIPING-MIGRATION.



(a) (b)

Fig. 5. Performance impact of design parameters. (a)Impact of different δ values. (b)Impact of different chunk Sizes.



(a) (b)

Fig. 6. (a)Hybrid drive vs. 2-harddisk striping device on SPECsfs 3.0 using Transcend 16G drive. (b)Hybrid drive vs. 2-harddisk striping device on SPECsfs 3.0 using MemoRight 32G drive.

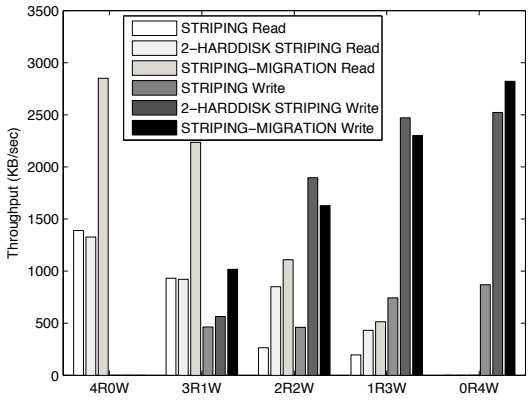


Fig. 7. Hybrid drive vs. 2-harddisk striping device on IOzone.

As we can see from Figure 7, the performance of Transcend-based hybrid drive with STRIPING policy is not as good as that of the 2-harddisk system, especially the writing performance. However, with migration employed, the performance of the hybrid drive achieved significant improvement, even surpassed the 2-harddisk system. The results show that the hybrid drive with migration can get higher performance improvement when the ratio of read/write requests is higher, even with the slower Transcend flash drive. When the ratio was 1:3 (in workload 1R3W), the performances of 2-harddisk system and hybrid drive with STRIPING-MIGRATION policy are almost the same.

These results indicate that read/write characteristics of the workload have a critical impact on the hybrid system. With migration, the hybrid system's performance can be improved greatly and made to offer a performance improvement over a 2-disk system over much wider workload characteristics than it would have been possible.

The results with Postmark benchmark experiment are shown in Table I. In our experiment we employed two simultaneous processes to generate sufficient concurrency, both processes running the Postmark benchmark with parameters mentioned in V-A. We ran our experiments across both the hybrid storage systems with Transcend and MemoRight flash drives and on the 2-disk system. It is observed that migration improved the transaction rate, read/write throughputs in both the hybrid storage systems, by about 10%. It is also observed that the Transcend-based hybrid system could not compete with the 2-hard disk system. However, the MemoRight-based hybrid system could outperform the 2-hard disk system, by roughly about 10-17%.

We conducted a second experiment with Postmark bench-

	Process 1			Process 2		
	Transactions /sec	Read Throughput	Write Throughput	Transactions /sec	Read Throughput	Write Throughput
Transcend-Hybrid (Striping)	52	211.46	49.22	50	204.70	47.65
Transcend-Hybrid (Migration)	58	237.18	55.21	57	229.68	53.46
MemoRight-Hybrid (Striping)	126	509.17	118.52	125	502.98	117.08
MemoRight-Hybrid (Migration)	137	553.17	128.76	134	538.76	125.41
2-Harddisk	121	487.77	113.54	115	462.86	107.74

TABLE I
PERFORMANCE OF DIFFERENT SYSTEMS WITH POSTMARK WORKLOAD. THROUGHPUT IS IN KBYTES/SECOND.

mark to study the impact of request size. In this experiment again, we employed two simultaneous Postmark processes. The file size was varied from a low of 500 bytes to 500 Kbytes, with each process generating and accessing 4,000 files. Each process conducts 80,000 total transactions in this experiment. The results of these experiment are shown in Table II. We employed two migration policies. In policy labeled Migration-1, only read/write characteristics were considered as earlier. In the second policy labeled Migration-2, request size was considered as detected by our sequentiality detector module. If the observed request size is less than one migration block (64KB in this experiment), we allowed the block to be migrated based on the read/write request patterns of that block. If the request size is observed to be larger, we allowed this data to exploit the gain that can be had from striping data across both the devices. If the block is accessed as part of a request larger than 64KB, it is not migrated.

The results of these two policies are shown in Table II. It is observed that migration policy based on read/write patterns improved the performance over striping. When we considered the request sizes and the read/write access patterns (Migration-2), the performance is observed to be higher. While the performance improved by about 7% for MemoRight based hybrid storage system when read/write patterns are considered, the performance on an average improved by about 20% when both read/write patterns and the request size are considered. The performance for Transcend based storage system also improves in both the policies, the performance improvement is not as substantial. These experiments show that both read/write and request size patterns can be exploited to improve performance.

Memoright-based hybrid storage system provided better performance than a 2 hard disk system, but Transcend-based hybrid system could not match the performance of the 2 hard disk system. The results in this section indicate that the newer Memoright flash drive provides superior performance and exploiting such devices in various scenarios will remain interesting.

VI. DISCUSSION & FUTURE WORK

Typically, storage or memory hierarchies are considered for dealing with different speeds of devices or components. The asymmetric read/write costs or performance of flash and disk drives warranted a more flexible approach to block allocation and redistribution.

In our approach, data is migrated conservatively to other locations only when there is a benefit in terms of performance. To avoid oscillations and to take costs of migration into account, we set thresholds (the δ parameter) to dampen our system. In the future, we plan to make this parameter δ a function of migration cost of moving from one device to another device and the loads on the devices.

Migration among devices of different speeds are considered earlier in AutoRAID [1] and in storage systems based on disk and tape [3], [7], [8], [18], [19].

In this paper, we considered several allocation policies: initially all the data on flash, all the data on disk, striping, and allocation partially determined by the observed device performance characteristics. Random allocation has been proposed by others [2], [11] to achieve load balancing with heterogeneous devices. We will consider random and other allocation schemes that employ file system or application level hints, in the future.

We plan to integrate our online migration with a system level block placement optimization strategies that are implemented less frequently (e.g., once a day). Strategies for seek optimization of blocks on disk device have employed such strategies before [24], [25].

We also plan to evaluate our migration policy in more diverse workloads and test it on a wider set of flash devices and different capacity configurations of flash and disk. We plan to control the token generation rate, in the future, based on the observed loads and utilizations of devices in the system.

Our results indicate that it is possible to detect the read/write access patterns and the request sizes to migrate the blocks to the appropriate devices to improve the device performance in a hybrid system. The measurement-drive approach is shown to be flexible enough to adapt to different devices and different workloads. We expect the hybrid storage systems to provide significant performance improvements as flash drives improve.

VII. CONCLUSION

In this paper, we studied a hybrid storage system employing both flash and disk drives. We have proposed a measurement-driven migration strategy for managing storage space in such a system in order to exploit the performance asymmetry of these devices. Our approach extracts the read/write access patterns and request size patterns of different blocks and matches them with the read/write advantages of different devices. We showed that the proposed approach is effective, based on realistic experiments on a Linux testbed, employing three different benchmarks. The results indicate that the proposed measurement-driven migration can improve the performance of the system significantly, up to 50% in some cases. Our

	Process 1			Process 2		
	Transactions /sec	Read Throughput	Write Throughput	Transactions /sec	Read Throughput	Write Throughput
Transcend-Hybrid (Striping)	17	3.19M	434.88K	17	3.19M	434.69K
Transcend-Hybrid (Migration-1)	18	3.35M	449.43K	17	3.33M	449.75K
Transcend-Hybrid (Migration-2)	19	3.47M	461.86K	18	3.45M	460.23K
MemoRight-Hybrid (Striping)	25	5.39M	610.72K	25	5.33M	602.92K
MemoRight-Hybrid (Migration-1)	26	5.69M	644.48K	26	5.67M	642.35K
MemoRight-Hybrid (Migration-2)	33	5.91M	740.05K	30	6.38M	722.17K
2-Harddisk	24	5.15M	583.10K	24	5.13M	580.32K

TABLE II
PERFORMANCE OF DIFFERENT SYSTEMS WITH POSTMARK WORKLOAD. THROUGHPUT IS IN BYTES/SECOND. THE LOWER BOUND OF THE FILE SIZE IS 500 BYTES, AND THE UPPER BOUND IS 500,000 BYTES.

study also provided a number of insights into the different performance aspects of flash storage devices and allocation policies in such a hybrid system. Our work showed that the read/write characteristics of the workload have a critical impact on the performance of such a hybrid storage system.

REFERENCES

[1] J. Wilkes, R. A. Golding, C. Staelin, T. Sullivan. The HP AutoRAID Hierarchical Storage System. *ACM Trans. Comput. Syst.* 14(1): 108-136 (1996).

[2] IBM Almaden Research Center. Collective Intelligent Bricks (CIB). *http://www.almaden.ibm.com/StorageSystems/autonomic_storage/CIB/index.shtml*, 2001.

[3] B. Gavish and O. Sheng. Dynamic File Migration in Distributed Computer Systems. *Commun. ACM*, pages: 177-189, 1990.

[4] E. Anderson et al., Hippodrome: Running circles around storage administration. *USENIX FAST Conf.*, Jan. 2002.

[5] L. Yin, S. Uttamchandani and R. Katz. SmartMig: Risk-modulated Proactive Data Migration for Maximizing Storage System Utility. In *Proc. of IEEE MSST* (2006).

[6] S. Kang and A. L. N. Reddy. User-centric data migration in networked storage devices. Proc. of IPDPS, Apr. 2008.

[7] E. Miller and R. Katz. An Analysis of File Migration in a UNIX Supercomputing Environment. In *Proc. of USENIX* (1993).

[8] M. Lubeck, D. Geppert, K. Nienartowicz, M. Nowak and A. Valassi. An Overview of a Large-Scale Data Migration. In *Proc. of IEEE MSST* (2003).

[9] R. M. English and A. A. Stepanov. Loge: A Self-Organizing Disk Controller. In *Proc. of USENIX Winter* (1992).

[10] R. Wang, T. E. Anderson, and D. A. Patterson. Virtual Log-Based File Systems for a Programmable Disk. In *Proc. of OSDI* (1999).

[11] S. Ghemawat, H. Gobioff and S-T. Leung. The Google File system. In *Proc. of SOSP* (2003).

[12] S. Sankar, S. Gurumurthi and M. Stan. Sensitivity based power managemenet of enterprise storage systems Proc. of MASCOTS 2008.

[13] C. Lu, G. A. Alvarez and J. Wilkes Aqueduct: online data migration with performance guarantees In *Proc. of FAST* (2002).

[14] A. Brinkmann, K. Salzwedel and C. Scheideler. Efficient, distributed data placement strategies for storage area networks (extended abstract). In *Proc. of SPAA* (2000).

[15] Y. Saito, S. Frølund, A. Veitch, A. Merchant and S. Spence. FAB: building distributed enterprise disk arrays from commodity components. In *Proc. of ASPLOS* (2004).

[16] R. Honicky and E. Miller. A Fast Algorithm for Online Placement and Reorganization of Replicated Data. In *Proc. of IPDPS* (2003).

[17] C. Wu and R. Burns. Handling Heterogeneity in Shared-Disk File Systems. In *Proc. of SC* (2003).

[18] A. J. Smith. Long Term File Migration: Development and Evaluation of Algorithms. *Communications of ACM* 24(8): 521-532 (1981).

[19] V. Cate and T. Gross. Combining the Concepts of Compression and Caching for a Two-level File System. In *Proc. of ASPLOS* (1991).

[20] Standard Performance Evaluation Corporation. SPEC SFS97_R1 V3.0 *http://www.spec.org/osg/sfs97r1*.

[21] R. P. Martin and D. E. Culler. NFS Sensitivity to High Performance Networks. In *Proc. of SIGMETRICS* (1999).

[22] L. Peterson and B. Davies. Computer Networks: A Systems Approach. Morgan Kauffman Publishers (2000).

[23] IOZONE file system benchmark. http://www.iozone.org/, accssessed 07/2008.

[24] R. Wang, T.Anderson and D. Patterson. Virtual log based file systems for a programmable disk. *Proc. of OSDI*, 1999.

[25] S. D. Carson and P. F. Reynolds, Adaptive disk reorganization. *Tech. Rep.: UMIACS-TR-89-4, University of Maryland, College Park, Maryland*, January 1989.

[26] I. Koltsidas and S. Viglas. Flashing up the storage layer. *Proc. of VLDB*, Aug. 2008.

[27] T. Kgil, D. Roberts and T. Mudge. Improving NAND Flash Based Disk Caches. *Proc. of ACM Int. Symp. Computer Architecture*, June 2008.

[28] H. Kim and S. Ahn. BPLRU: A buffer management scheme for improving random writes in flash storage. *Proc. of USENIX FAST Conf.*, Feb. 2008.

[29] H. Kim and S. Lee. An effective flash memory manager for reliable flash memory space management. *IEICE Trans. on Information systems*., June 2002.

[30] S. Baek et al. Uniformity improving page allocation for flash memory file systems. *Proc. of ACM EMSOFT*, Oct. 2007.

[31] J. Lee et al. Block recycling schemes and their cost-based optimization in NAND flash memory based storage system. *Proc. of ACM EMSOFT*, Oct. 2007.

[32] Li-Pin Chang. On efficient wear leveling for large-scale flash-memory storage systems. *Proc. of ACM SAC Conf.*, Mar. 2007.

[33] A. Birrell, M. Isard. C. Thacker and T. Wobber. A design for high-performance flash disks. *ACM SIGOPS Oper. Syst. Rev.*, 2007.

[34] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, June 2005.

[35] S. Akyurek and K. Salem. Adaptive block rearrangement. *Computer Systems*, vol.13, no.2, pages 89-121, 1995.

[36] R. Tewari, R. King, D. Kandlur and D. Dias. Placement of Multimedia blocks on zoned disks. *Proc. of Multimedia Computing and Networking*, Jan. 1996.

[37] Jeffrey Katcher. Postmark: A New File System Benchmark. *http://www.netapp.com/tech library/3022.html*.

[38] A.-I. A. Wang, P. Reiher, G. J. Popek, and G. H.Kuenning. Conquest: Better performance through a disk/persistent-ram hybrid file system. *In Proceedings of the USENIX Annual Technical Conference*, June 2002.

[39] M.Wu and W. Zwaenepoel. envy: A non-volatile, main storage system. *Proc. of ASPLOS*, 1994.

[40] L. Useche, J. Guerra, M. Bhadkamkar, M. Alarcon and R. Rangaswami. EXCES: External caching and energy saving storage systems. *Proc. of HPCA*, 2008.