# SANBoost: Automated SAN-Level Caching in Storage Area Networks

Ismail Ari[†]
*ari@cs.ucsc.edu*

Melanie Gottwals[‡]
*melanie.gottwals@hp.com*

Dick Henze[‡]
*dick_henze@hp.com*

[†]*Storage Systems Research Center*
*University of California, Santa Cruz*

[‡]*Storage Technologies Department*
*Hewlett-Packard Laboratories*

## Abstract

*The storage traffic for different Logical Units (LUs) of a disk array converge at the array's cache. The cache is allocated among the LUs approximately according to their relative I/O rates. In the case of non-uniform I/O rates and sensitivity to storage response times between differing applications in a Storage Area Network (SAN), undesirable cache interference between LUs can result in unacceptable storage performance for some LUs.*

*This paper describes SANBoost, a SAN-level caching approach that can be enabled selectively on a per-LU basis to provide a performance isolation mechanism for response time metrics related to storage quality of service (QoS). SANBoost automates hot data detection and migration processes in block-level storage. The design consists of a migration module implemented in a fabric-based SAN virtualization appliance and a Solid-State Disk (SSD) that acts as a cache resource within the same SAN.*

*Simulation results quantify the impact of a specific static SANBoost caching policy on the SPC-1 benchmark workload and address the relative impact of adapting a threshold in the placement algorithm.*

## 1. Introduction

The storage capacity within Storage Area Networks (SAN) is increasing to petabyte levels. Consolidation of workloads over this large scale storage results in a wide spectrum of usage patterns. Access frequency for a given piece of data can range from very hot, *i.e.* accessed extremely frequently such as the meta-data, to very cold, *i.e.* not accessed for long periods of time such as the archival data.

Typically, the cache within the disk array's storage controller is the primary mechanism to absorb accesses to hot data. As the storage traffic for different *Logical Units* (LUs) of a disk array converge at the array cache, cache resources are allocated among the LUs approximately according to their relative I/O rates. In the case of non-uniform I/O rates to different LUs and non-uniform sensitivity to storage response times between different applications, cache interference between LUs can result in unacceptable storage performance for some LUs.

If hot data can be identified and statically separated from the cold data, it can be assigned to storage technologies with faster response time and higher throughput compared to disk in order to improve the overall performance. This can be implemented with DRAM and system-level solutions for non-volatility and data protection. In the future, non-volatile device technologies such as magnetic RAM (MRAM) [16] or MEMS probe-based storage [12] are expected to provide this function. Today, identification of the hot data and its configuration onto high performance storage resources is done manually by system administrators with the help of access profiling software suites. This manual type of migration typically happens at the file abstraction-level including hot files and database tables [4]. Unfortunately, as the heterogeneity of applications, users, and operating systems (OS) increase in a computing environment, the *manual identification of hot data and its migration in real time becomes overwhelmingly complex* due to the separate profiling tools required by different OSs and applications.

SANBoost is a SAN-level caching approach that *automates hot data detection and migration processes* in block-level storage. The design consists of a migration module implemented in a fabric-based SAN virtualization appliance and a Solid-State Disk (SSD) that acts as a cache resource within the same SAN. Currently the module continuously profiles accesses to fixed-sized *chunks* for the selected, or *boosted*, *Logical Units* (LUs). Chunks are envisioned to be large cache lines in the range of 128 kilobytes (KB) to a few megabytes (MB). The module then makes decisions regarding appropriate data for placement into the SSD using a placement policy, manages the data migration operations from disk arrays to the SSD, and controls replacements from the SSD-based cache.

In this role, the SANBoost cache extends the demand cache within the SAN's backend disk arrays and becomes

a resource to implement the second list of dual-list cache policies such as LRU-2 [17], 2Q [13], or ARC [15]. The frequency-based content of the SANBoost cache makes it similar to the second list of the mentioned algorithms. However, several differences are present in our design compared to previously reported dual-list policy implementations. First, the second list (SANBoost) cache is only available to selected LUs. Second, both the number and characteristics of the enabled LUs, and the capacity of solid-state storage partitioned for this function can change dynamically during system operation. Finally, the cache line size for chunks in the SANBoost cache is substantially larger than the line size implemented in disk array caches, resulting in different cache behavior. Due to the variability of configurations resulting from these differences, adaptation of the policies that control the SANBoost cache contents appears a worthy topic of investigation.

The cost of disks within Storage Area Networks (SAN) is decreasing, but the cost of SAN management remains high. Storage administration cost is estimated to be several times the cost of the storage hardware [8]. System administrators are spending significant amounts of time migrating data between storage resources. Facets of information lifecycle management (ILM) are beginning to address the automation of data placement related to changing data access needs over data lifetime. However, ILM does not address migration for performance issues in I/O intensive commercial applications and highly dynamic environments. Hence manual data migration to mitigate performance limitations continues to contribute to storage management costs, and should be addressed with automation to address storage quality of service (QoS) issues.

This paper presents simulation results to quantify the impact of SANBoost caching using the Storage Performance Council's SPC-1 benchmark. Section 2 describes the SAN virtualization and solid state storage resources used to implement the SANBoost approach. Section 3 details the structure of the design and introduces the policies that identify hot data and control its migration to and from the SANBoost cache resource. Section 4 describes the SPC-1 workload. Section 5 presents analysis and simulation results of SANBoost cache behavior using a static-threshold placement policy, and then demonstrates the relative impact of adapting the threshold in the placement algorithm. Sections 6 and 7 describe related work and summarize the conclusions, respectively.

## 2. SAN Virtualization

Storage Area Networks (SANs) are composed of a pool of disk arrays serving as backend storage and the fabric such as Fibre Channel (FC) or Gigabit Ethernet (GigE) that connects the storage to the hosts. Recently, usage of fabric-based SAN virtualization appliances [2, 3] have gained popularity, since they simplify the management of large SANs comprised of heterogeneous storage and client operating systems. These appliances [2] can also provide local and remote mirroring and snapshotting of heterogeneous backend storage resources.

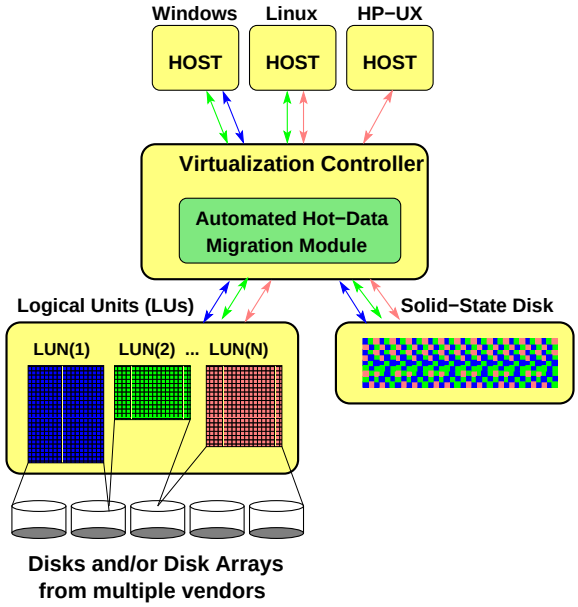



**Figure 1. System topology using SAN virtualization controller and solid-state disk for hot data migration.**

Figure 1 shows a SAN environment with virtualized backend storage. By using the virtualization system, administrators first collect storage capacity from multiple, possibly multi-vendor, disk arrays into a single pool, then partition the pooled storage into Logical Units (LUs), thus "virtualizing" the physical details of the storage resources. Finally, they manage the mapping of LUs to the hosts. This process continues as more hosts and more storage resources are added.

For continuous data availability it is crucial to have multiple paths to the data. Virtualization appliances typically provide a redundant server configuration to avoid single point of failure [2]. Host Bus Adapters (HBAs) are designed to detect failures and switch to alternate paths automatically. HBAs can also provide load balancing between the data paths [2].

### 2.1. Solid-State Disk

A Solid-State Disk (SSD) [6, 4] a is DRAM-based storage resource with a block interface similar to disk-based storage (hence the name "disk"). SSDs speak SCSI [7] or FCP protocols [1]. Random access times of SSDs are measured in tens of microseconds ($\mu$s) compared to milliseconds (ms) for disk drives. Unlike hard disk technology where response time varies with data locality, SSDs have no data

layout sensitivity and exhibit similar performance for random and sequential access patterns. Bus and protocol overheads generally limit SSD response times to the order of a hundred microseconds.

In our design, we add an SSD to the backend fibre channel SAN fabric and partition a desired portion of it as a Logical Unit (LU), similar to configuration of disk-based LUs. This LU on the SSD is our SANBoost cache resource. The capacity of the SANBoost cache resource can be easily expanded or additional SSDs can be added to the SAN using the existing capability of the virtualization appliance.

Modern SSDs achieve fault tolerant operation by the use of redundant controllers with multiple-ports and redundant power supplies, similar to the disk arrays. DRAMs within SSDs incorporate Error Correcting Code (ECC) and are protected with batteries and internal disk systems to ensure safe data retention for extended power failures.

## 3. Hot Data Migration

The objective for our SANBoost caching solution is to provide performance isolation for selected LUs in the SAN in order to maintain QoS required for satisfactory application-level behavior. These LUs can be identified using application-specified contracts, I/O intensity, or other requirements. Several goals are desired from the policies that control the SANBoost cache contents. These goals are *to maximize the access density of chunk contents in the SSD* and *to drive exclusivity between the contents of the array cache and the SANBoost cache*. If successful, the performance benefits of hot data migrations will be threefold. First, host I/O that is served from the SSD will have a shorter response time compared to the data served from the mechanical disks. Second, since the hot traffic is handled by the SSD, the I/O rate to the disk array will be decreased, thus reducing the queue length and wait time of the disk array and resulting in better access performance for the data on disk. Third, sequential I/O streams which are maintained primarily by the disk array will experience decreased contention with other traffic leading to more efficient sequential runs. Note that, the second and the third benefits mentioned will improve performance for all LUs on the same array. This is especially important for those LUs on heavily utilized arrays.

Our system topology for hot data migration is shown in Figure 1. While this figure shows a single SSD, redundancy mechanisms such as mirroring to two SSDs would be used for data protection in real implementations. A software module is added to the virtualization appliance's OS to make the data placement decisions and to control the migration. This module does bookkeeping of access frequencies to chunks pertaining to boost-enabled LUs. Three major policies govern the migration process and are further addressed in following subsections:

- A *Placement Policy* that decides on which specific chunks to migrate to SSD. It continuously generates a list of candidate chunks from enabled LUs **before** their migrations.
- A *Migration Plan* that determines how the migration of candidate chunks should be managed **during** the migrations. It adjusts the rate of migrations via throttling to avoid over-utilization of system resources. It also selects the appropriate network paths to minimize interference between migrations and user I/O and uses locks to assure the consistency of the migrated data.
- A *Replacement Policy* that identifies the least preferable chunks as candidates for eviction from the SSD **after** their migration. It also checks whether the chunk is dirty or clean to avoid unnecessary flushes back to disk.

Before migration, all requests go to the disk LUs. For boosted enabled LUs, usage statistics are maintained at a chunk level. An optional module can accumulate access statistics for all LUs in order to assist in the selection of candidate LUs for subscription to the SANBoost feature. During migration, threads of I/Os directed to the chunks under migration and the migration threads moving these chunks are serialized. The rest of the I/Os proceed to the disk normally. To minimize the performance impact of the blocked writes, journaling and *resilvering* of writes [14] can be used. These are well-known techniques used for taking snapshots of LUs. Incoming I/Os to boost-enabled LUs trigger a lookup procedure to determine if they fall within a chunk that has already been migrated. I/Os that hit the SANBoost cache contents are served from the SSD and I/Os to regions not migrated are forwarded to the disk similar to normal unboosted operation.

### 3.1. Placement Policy

With demand caching, user's read requests that miss in the cache are fetched from the disk and placed in fixed-sized array cache lines at once. This is representative of policies typically implemented in disk arrays. Writes are also initially placed in the array cache for disk arrays with write-back functionality. Therefore, if the *migration threshold* is defined as *"the number of times the user has to access a chunk of data before it is placed into the cache"*, then demand caching would be the special case of a more generic placement policy with placement threshold equal to one. For a relatively large size unit of management such as our chunks (*e.g.* 128KB to 1MB), a demand-based chunk placement would be too aggressive, since we do not expect numerous accesses to every touched chunk. Therefore, the SANBoost placement policy delays the migration of a chunk by enforcing a higher threshold. However, the migration decision cannot be delayed indeterminately as we

will see in Section 5. In summary, *chunk migration combines frequency-based caching with speculation of spatial locality based on prefetching proximate address regions*.

To avoid migrating data associated with sequential data transfers, a sequential detection algorithm similar to that employed in disk array caches is also used in SANBoost module. A number of discrete transfers that aggregate to a sequential disk access within or across chunks are counted only as a single access in the chunk access statistics. This allows large sequential runs to be handled efficiently by the disk array. Combining this strategy with the frequency-based placement we attempt to provide *exclusivity* [18] between the array cache and the SANBoost cache.

*Static migration policies* count the number of accesses to a chunk and migrate a chunk when the count reaches a predetermined and fixed threshold. An *adaptive policy* is one that can change its threshold value based on an adaptation rule to obtain higher performance compared to static policies. We compare several static and adaptive migration policies in Section 5.

### 3.2. Migration Plan

Before migrations, the user I/O traffic is entirely served from the disks. During migrations, the migration traffic has to compete with the user I/O traffic for the disk, fabric, appliance interface bus and CPU resources. Since the chunk sizes (*e.g.* 128KB to 1MB) are fairly large compared to the user I/O traffic (*e.g.* 4KB to 16KB), long delays could be incurred by the user I/O traffic queued behind migration transfers. The *migration plan* is in charge of coordinating system resources to minimize these delays and avoiding over-utilization of bus and CPU resources of the appliance server node. Various local and distributed locking mechanisms are used to provide consistency of the accessed data. We restrict our discussion on migration plan to the following concepts in use or under consideration and leave the details to other previous work [14].

Migrations can use alternate data paths to backend storage and the SSD, when physically possible. Under conditions of heavy bus and CPU utilization, optimizations used by the migration plan include limiting the number of threads used for the migration service and transferring the migration responsibility for a given chunk to a peer node. Note that, the threshold-based migration policy described above and the adaptation capability on the threshold value inherently control the migration rate. Aqueduct [14] uses feedback loop from control-theory to control the rate of migrations. TCP's slow start and congestion avoidance mechanisms are also good candidates for providing adaptive rate control.

### 3.3. Replacement Policy

Replacement policies provide a priority ordering of migrated chunks and evict the least preferable chunks from SSD to create space for incoming chunks. Least Recently Used (LRU), and Adaptive Replacement Cache (ARC) [15] are simulated and compared in Section 5. ARC dynamically biases its policy between recency and frequency depending on the workload conditions. With respect to replacement issues we ask and answer the following question: How does SANBoost chunk cache compare in hit performance to a demand cache using different replacement algorithms?

Adaptive Caching using Multiple Experts (ACME) [10] is another adaptive framework that can switch among a set of replacement policies to track the best current algorithm. It uses machine learning to evaluate a pool of virtual cache "experts" (replacement policies). Our future work includes testing this algorithm.

### 3.4. Bookkeeping of Migrated Chunks

When a chunk from a backend storage LU is migrated into the SSD, an entry for this chunk is added into a *chunkmap*. The chunkmap is a map of $\langle key, value \rangle$ pairs, where the *key* is efficiently obtained from a combination of the LU and chunk information. An in memory copy of the chunkmap is kept for high-speed access. The chunkmap is also stabilized on a small, non-volatile storage partition for disaster recovery purposes.

Statistics about chunk usage (reads, writes, misses, *etc.*) are recorded in a separate data structure called the *chunk information table*. It is not necessary to stabilize this table to non-volatile storage as the information retained is not crucial for disaster recovery purposes.

## 4. SPC-1 Workload

The Storage Performance Council (SPC) publishes industry standard storage performance benchmarks [5]. "SPC-1 is comprised of a set of I/O operations designed to demonstrate the performance of a storage subsystem while performing the typical functions of a business critical application [5]." We collected the I/O trace of the SPC-1 benchmark on a real disk array storage configuration. Table 1 summarizes the relative I/O distribution among LUs. LU-5, which is representative of an OLTP "data store", has four I/O streams associated with it: one random walk, one sequential scan, and two localized I/O streams. LU-7, representative of an OLTP "user store", has three I/O streams: one random, one sequential and one localized stream. LU-9, representative on an OLTP "Log", and has one sequential I/O stream. Other details can be found in the SPC-1 specifications [5]. Only LU-5 is boosted because it receives the largest portion of the overall traffic (59.6%) and contains all the different types of streams. Further analysis and simulation results presented below are conducted on this portion of the trace.

**Table 1. SPC-1 benchmark properties.**

| Parameter | LU-5 | LU-7 | LU-9 | All |
|---|---|---|---|---|
| Capacity | 54 GB | 54 GB | 12 GB | 120 GB |
| I/O Traffic (%) | 59.6% | 12.3% | 28.1% | 100% |
| Number of Streams | 4 | 3 | 1 | 8 |

## 5. Analysis and Results

We first examine the relationship between the data access behavior in the SPC-1 workload and the benefits obtained from using the SANBoost cache with a placement policy using static thresholds. Then, we illustrate a method to adapt this placement or migration threshold for improved benefits.

### 5.1. Static Migration Threshold

A placement policy with a static migration threshold counts number of accesses to a chunk and migrates a chunk when the count passes a predetermined, fixed threshold. Figure 2 shows the access histogram of the SPC-1 workload on the LU-5 and illustrates how the choice of the migration threshold influences the benefit obtained from migrated chunks. The access histogram in Figure 2(a) is obtained by mapping the user block requests to the closest (256KB) chunk boundaries and then bucketing and plotting the access counts. The count of chunks that were accessed 3 times or less have been concatenated for clarity of the plot, as they went up to tens of thousands in counts. Next, we identify the results of choosing low, medium or high threshold values, respectively.

Figure 2(b) shows the *benefit* of migrations expressed as hits achieved per migrated chunk (*hits/migratedChunk*) as a function of *threshold*. We see that choosing a threshold smaller than 3 causes the migration of many chunks into SSD that would be subsequently accessed only a few more times; these are the chunks at the far left side of Figure 2(a). The result is a lower benefit (in *hits/migratedChunks*) as seen at the far left side of Figure 2(b). Therefore, a low threshold would have an adverse impact on cache performance due to the resultant high migration volume and low access statistics of migrated chunks.

In Figure 2(a), the hill seen around medium access counts (5-40) leads to an interesting result as it corresponds to the *local* maxima (the first maxima 4) in Figure 2(b). An intermediate threshold of 3-4 avoids migrating the "low value" chunks with few accesses (3 or less in this case), but is able to allow early capture of chunks that will subsequently receive many accesses. This behavior enhances the overall value obtained by speculation.

The higher thresholds wait for longer periods of time and migrate fewer, but possibly more valuable chunks. However, as they wait a longer interval to accumulate accesses up to the migration threshold, they also miss many chances for getting hits in SSD. The benefit (hits/migratedChunks) is therefore lowered again. While this analysis guides our choice of static threshold, the most important result we derive is *the potential for adaptation of the threshold to improve hit performance*. This approach is particularly important for variable workloads and configurations.

Figure 3 shows the effects of migration on overall disk I/O performance. We enable hot chunk migration on LU-5 partition. The disk I/O is reduced as a portion of the workload is served by the SSD. In this simulation, the SSD size was 6 GB, chunk size was 256 KB and the static threshold was 30, which is the global maxima in Figure 2(b). In Figure 3, the migration traffic is plotted in the negative direction to indicate that this operation is a cost.

Figure 3(a) shows the I/Os per second (IOPS) handled by the disks. The disk IOPS is reduced from 700s to 200s after about 20 minutes (1200 seconds) of operation. So at this time approximately 70% of the LU-5 traffic (500/700) is being handled by the SSD. The movement of the hot traffic from disk to SSD is achieved with a minimal increase in total IOPS due to migrations. The IOPS to disk reduced even more, reaching up to 80% by the end of the one hour trace.

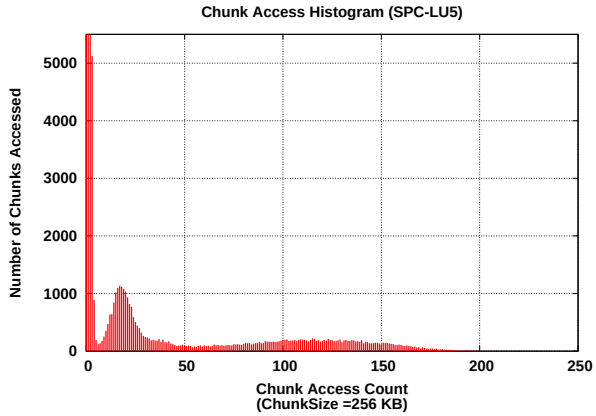
Figure 3(b) shows the same results by weighing each I/O by its size. As the migration I/O size, 256 KB, is much larger than the average user I/O size, 4-8 KB, significant increases in transfer bandwidths were witnessed temporarily for about 20 minutes. However, note that the total transfer bandwidth was already lower than the bandwidth previously used by the workload with no boosting after 30 minutes (1800 seconds). Also, since larger I/Os achieve higher disk throughput, the array queuing time won't increase proportionally. The lesson we learn from this increase is that the *migration plan* has to consider all these aspects of migration and enforce specific controls when necessary.

We do not infer the reductions in user perceived latency at this time, as we are currently doing these measurements on real platforms. Doing latency analysis via simulations require detailed and complex disk and disk-array models [11].

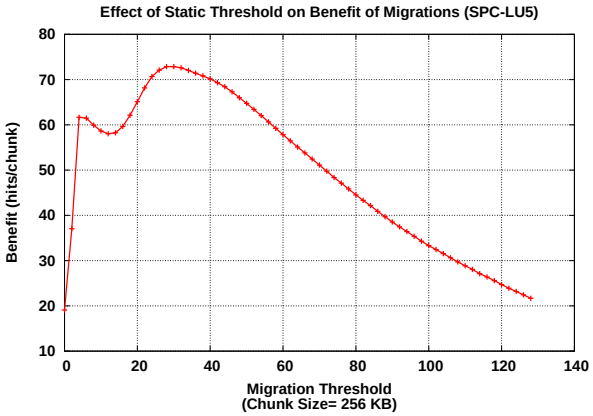
### 5.2. Adaptation of the Migration Threshold

Figure 4 shows a simple adaptation rule for the migration threshold. It tries to adjust the migration threshold such that the change in benefit ($\Delta benefit$) is kept positive. The rule decreases the threshold to migrate more chunks when migrations result in greater benefits and increases the threshold to be more selective when the migrations do not result in a positive change in benefit. The current adaptation rule is elementary and subject to further refinement.

Figure 5 shows the effects of the migration threshold adaptation on SPC-1 LU-5 workload. The threshold which was initialized to 4 stayed around this value for a while and then increased to around 20. We explain this behavior as

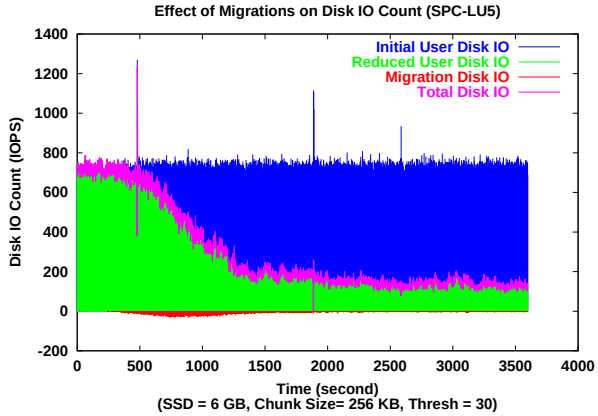



(a) Histogram of chunk access counts for LU-5.

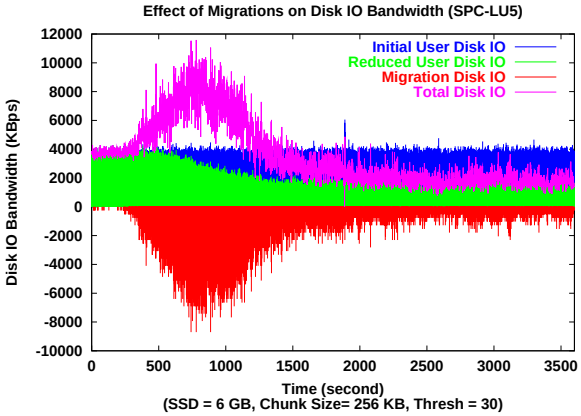



(b) Benefi t of migration expressed as hits achieved per migrated chunk as a function of threshold.

**Figure 2. Choice of threshold value will influence the benefit obtained from migrated chunks.**




(a) Number of IOs seen by the disks is reduced.




(b) Disk IO bandwidth usage is reduced.

**Figure 3. Effects of hot data migration on overall storage system performance.**

follows. Many chunks were quickly accessed more than 4 times and were migrated. Furthermore, among the migrated chunks many (the hill in Fig. 2(a) contains 1000s of chunks) achieved tens of hits. The adaptive threshold value stayed around 4 during this period. The benefit rate ($\Delta benefit$) decreased when significantly less chunks among the migrated were being accessed more than 30 times (the down slope of the hill in Fig. 2(a) and on). Thus, the adaptation rule chose to increase the threshold (by *Count* =1) to be more selective. By making these decisions, the adaptive policy performed better than both of the good static threshold choices, 4 (local maxima) and 30 (global maxima). In Figure 5(b), the comparison of Adaptive to Static-30 policy at x-axis point 75 shows that adaptive policy could achieve up to 100% improvement (25 vs. 12) over the Static-30 policy for the workload in this study.

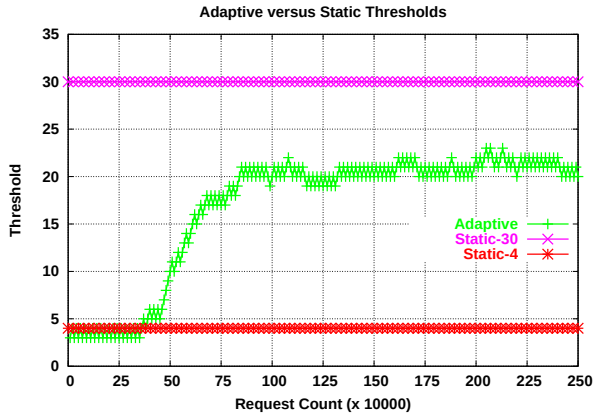
**Initialize:** $prevBenefit = 0$
$prevDelta = 0$

$benefit = totalHits/migratedChunks$
$\Delta benefit = benefit - prevBenefit$
*if*($\Delta benefit \rangle prevDelta$)
  $threshold$ -= $Constant$
*else*
  $threshold$ += $Constant$
$prevBenefit = benefit$
$prevDelta = \Delta benefit$

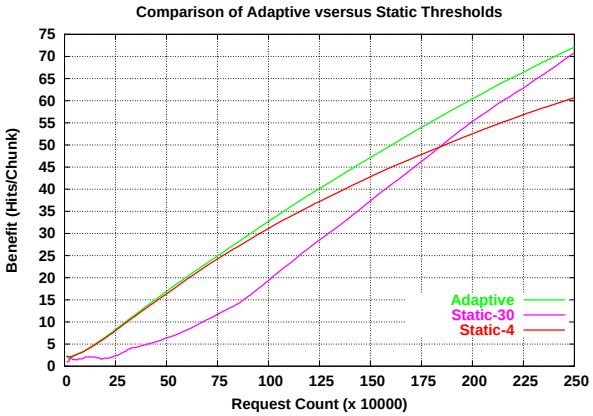
**Figure 4. A simple rule for adaptation of the migration threshold.**

## 5.3. Static and Adaptive Replacement

Figure 6 shows a comparison of cache hit rates for SANBoost and demand caches as a function of cache size. The demand caches are modeled with Least Recently Used (LRU) and Adaptive Replacement Cache (ARC) replacement policies and a cache line size of 4 KB. SANBoost cache results are shown for two static migration thresholds, 10 and 30. ARC achieved up to 3.4% higher hit rates than LRU, since it can dynamically change the cache sizes allocated for the recency (first) and frequency (second) lists




(a) Static and adaptive thresholds.




(b) Benefi t of migrated chunks measured in hits per chunk (top line is adaptive).

**Figure 5. Adaptation of the migration threshold resulted in higher benefits than static thresholds.**

to track the changes in workload behavior and to provide scan resistance. The unique data touched by 4KB-demand caching on SPC-1 LU-5 was smaller than 8GB. So, there were no replacements for cache sizes $\geq$8GB and the algorithms perform the same.

The SANBoost cache used only a fraction (as small as $1/4^{th}$) of the cache sizes used by demand caches to achieve the same hit rates. This can be seen by drawing a horizontal line at 40% hit rate in Figure 6. The SANBoost cache with threshold 10 provided up to 15% improvement (until 8GB) over the demand caches using the same cache sizes. This is due to the speculative prefetching of proximate address ranges into the SSD cache. Around 16 GBs the hit rate difference between SANBoost with threshold 10 and demand caches reached to 20%. However, note that the added cache capacity had diminishing returns. With migration threshold equal to 30, SANBoost achieved higher hit rates than demand caches for cache sizes $<$2GB. For caches $>$2GB and $<$14GB it achieved lower hit rates. The reason is that the migration threshold 30 is not aggressive enough to quickly utilize the added cache space compared to the demand caches. This result suggests "free SSD cache space" as another important parameter that affects the selection of the migration threshold. Therefore, this parameter should also be integrated into the adaptation rule. We leave this as future work.

## 6. Related Work

In demand caching the requested blocks are fetched and placed into the cache the first time they are accessed. Therefore, *demand caching is a special case of our threshold-based placement policy with threshold equal to one*. In the existing two-list cache replacement algorithms [17, 13, 15], the requested blocks are initially demand fetched into the first list and then they are migrated to the second list upon

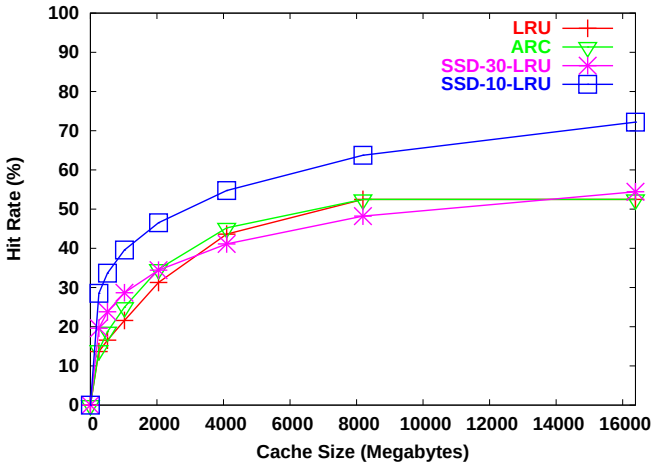



**Figure 6. Comparison of SANBoost cache with static LRU and adaptive ARC demand caches using same cache sizes.**

a second hit. Therefore, in two-list algorithms the placement threshold for both lists is equal to one. Our SAN-level caching design emulates a two-list caching algorithm where the first list consists of the demand-based disk array cache and the second list consists of the SAN-attached SANBoost cache with a migration threshold higher than one.

Some virtualization appliances provide data caching by using the DRAM embedded within the virtualization servers. In comparison, SANBoost uses a SAN-attached *data* cache and manages the *meta-data* within the virtualization servers. Our solution makes it easier to expand the cache resource. Also, since our solution requires only the synchronization of meta-data (not the data) when multiple servers exist, we believe that our strategy is potentially more scalable than in-memory data caches.

Due to the increased complexities, automation of stor-

age management has become extremely important. Hippodrome [8] automates the assignment of workload streams to different Logical Units (LUs) and reevaluates the performance results to reconfigure the storage resources in a SAN. Ergastulum [9] automatically finds the appropriate RAID levels for different streams.

The migration process has to be done without disrupting the front-end user I/O performance. Aqueduct [14] describes an online migration module that signs a delay contract with the front-end users (or applications) and guarantees a worse case delay performance using control theory. The driving applications for this work are online backup, failure recovery, or load balancing where the data being migrated is relatively "colder" than what SANBoost aims to handle. These type of bulk data migrations can also be distinguished by a start and an end, whereas the SANBoost migration is a continuous process.

## 7. Summary and Conclusions

SAN administration is a complex, costly, continuous, and online process. We described the design of an automated hot chunk migration module, called SANBoost, embedded in a SAN virtualization appliance. SANBoost collects statistics on accesses to fixed-sized chunks on the order of 128 kilobytes to a few megabytes. It uses a migration threshold to choose the most valuable chunks for placement in SSD cache, a migration plan to move chunks from the disk to the SSD in a controlled fashion, and a replacement policy to find the least valuable chunks to discard from the cache or flush to disk if dirty.

We described the details of these three policies and discussed the possibilities for adaptation to achieve higher performance. We compared static and adaptive placement and replacement policies by using the SPC-1 benchmark workload. With the placement policy using best static threshold (30) we showed up to 80% reductions in the disk I/Os traffic. This 80% consists of the hot traffic that was handled by the faster SSD cache. Adaptation of the threshold allowed up to 100% increase in hits/migratedChunk for the same workload. SANBoost cache resulted in 15-20% higher hit rates compared to popular demand caches using the same cache sizes.

## Acknowledgements

Many thanks to John Bates, Scott Marovich, Rich Elder in HP, and Ethan Miller in UC Santa Cruz for their comments on the design of the SANBoost system and the earlier drafts of this paper.

## References

[1] Fibre Channel Protocol (FCP) http://www.t10.org.

[2] HP Continuous Access Storage Appliance (CASA), http://www.hp.com/go/casa.

[3] IBM TotalStorage virtualization family, http://www.storage.ibm.com/software/virtualization/.

[4] SolidData, SD3000 Solid-State Disk and White Papers, http://www.soliddata.com.

[5] SPC benchmark-1 specification, http://www.storageperformance.org/specification.html.

[6] Texas Memory Systems, RamSan Solid-State Disk, http://www.texmemsys.com.

[7] American National Standard for Information systems(ANSI). Small Computer System Interface SCSI-2. Standard X3.131-1994, Jan. 1994.

[8] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[9] E. Anderson, R. Swaminathan, A. Veitch, G. A. Alvarez, and J. Wilkes. Selecting RAID levels for disk arrays. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[10] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Proceedings in Informatics*, volume 14, pages 143–158. Carleton Scientific, 2002.

[11] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment version 2.0 reference manual. Technical report, Carnegie Mellon University / University of Michigan, Dec. 1999.

[12] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle. Modeling and performance of MEMS-based storage devices. In *Proceedings of the 2000 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 56–65, June 2000.

[13] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the 20th Conference on Very Large Databases (VLDB)*, pages 439–450, Santiago, Chile, 1994.

[14] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, Monterey, CA, Jan. 2002.

[15] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2003 Conference on File and Storage Technologies (FAST)*, pages 115–130, San Francisco, CA, Mar. 2003.

[16] E. L. Miller, S. A. Brandt, and D. D. E. Long. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 83–87, Schloss Elmau, Germany, May 2001.

[17] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.

[18] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference*, pages 161–175, Monterey, CA, June 2002. USENIX.