Performance Boosting and Workload Isolation in Storage Area Networks with SANCache

Ismail Ari, Melanie Gottwals, Dick Henze

ismail.ari@hp.com, melanie.gottwals@hp.com, dick.henze@hp.com

Hewlett Packard Laboratories

Abstract

Consolidation of server and storage resources in data centers results in interference of workloads with different characteristics and different performance requirements. It is difficult to isolate workloads and make performance guarantees on demand.

SANCache automates hot data selection and migration for selected logical units (volumes) on storage. It automatically allocates a SAN-wide cache resource across workloads in order to meet their performance goals. Our goal is to eliminate some of the tedious tasks handled by Storage Area Network (SAN) administrators for performance tuning. SANCache can handle hot data management at a finer granularity, faster response and higher accuracy than what could be achieved manually. This paper presents SANCache design features, demonstrates benefits of online adaptation, and presents performance results using industry standard SPC and TPC-C benchmarks over two commercial disk arrays. We also contribute a new data placement technique called Most Recently Frequently Used (MRFU) to the literature.

1 Problem Statement

The competitive, fast and dynamic business environment continuously demands higher performance and availability from IT infrastructures at lower costs. Enterprises consolidate their server and storage resources into data centers to meet the performance demands while also reducing management costs and increasing utilization. However, consolidation of storage in SANs can create performance problems due to workload interference, since disk-based storage is sensitive to changes in I/O traffic patterns (*e.g.* randomness, sequentiality and locality). Unpredictable storage performance leads to unpredictable application performance. Furthermore, due to increased adoption of utility computing initiatives [11] these applications can belong to different clients from different organizations. It is challenging to isolate workload streams or guarantee performance in SANs. For example, in a 24×7 enterprise the database storage can concurrently serve Online Transaction Processing (OLTP), decision support, and backup loads, which have different I/O characteristics.

Performance demands can also be quite bursty. While CPUs can cope with relatively high data access rates, disk-based storage is overwhelmed by I/O-intensive workloads. To quantify the intensity of data accesses we use the IOPS/GB metric, or *access density*, which measures I/Os per second (IOPS) access rate over a fixed storage address region. Note that this metric will have higher values for localized access patterns and lower values for sequential and uniformly-random accesses, since the latter span large address ranges. Storage administrators use rules of thumbs such as "~250 IOPS/spindle" while architecting disk arrays based on experience with common workload types to address the throughput and access density limits of mechanical disks. Therefore, disk capacity is usually over-provisioned to meet IOPS goals. Architecting SANs out of disk arrays to meet performance goals under mixed and dynamic sets of workloads is simply a black art [1].

Our method called SANCache has two main goals: to make networked storage fast on-demand and to provide isolation for workloads consolidated in SANs. We will demonstrate results with real systems and show improvements for both performance and isolation dimensions. SANCache augments disk array caches and automatically allocates a SAN-wide cache resource across workloads in order to meet their quality of service (QoS) requirements. For example, a client can request "guaranteed 5000 IOPS from selected logical units (LUNs) for the next 2 hours for a given OLTP load". SANCache automates "hot" (*i.e.* with high access density) data selection and migration process for selected LUNs. We call this process *boosting*. A software module at the SAN fabric virtualization layer observes I/O and migrates frequently accessed data chunks into a high-throughput, low-latency Solid-State Disk (SSD) attached to the SAN. Hot data is served from fast SSD after migration. Since policies are implemented at the virtualization layer, existing hosts and arrays remain unchanged. Boosting can be enabled and disabled when the hosts and storage are online and clients are accessing data.

2 Motivations and Challenges

Our biggest motivation is to automate some of the routine and manual tedious tasks handled by SAN administrators. These tasks include moving data around in the SAN for a variety of reasons such as load balancing, resolving hot spots, meeting reliability goals, upgrading systems, and archiving data. Automation of these tasks will spare expert time, thus reducing SAN management costs that are reported to be several times the cost of hardware [1].

Information Lifecycle Management (ILM) addresses 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

This paper was published in the *Proceedings of the 23rd IEEE / 14th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2006)*, College Park, MD, May 2006. Pages 263-273.

applications and highly dynamic environments. It deals with the migration of less frequently used reference or archival data to less-expensive online, near-line, or offline storage resources. SANCache addresses hot data management issues in a SAN.

Today, SAN administrators will try to observe system load and manually configure hot data onto a cache partition such as a RAMDisk. This process has several drawbacks. First, the number of software tools required for hot data detection can increase exponentially by the number of applications, operating systems (file systems, databases) and device drivers (or storage management interfaces), which can be formulated as [#Applications] × [#OS] × [# H/W interfaces]. Beyond the expertise that needs to be developed just for this task, the configuration process has to be done continuously to become efficient and will require quick response to changes for optimal performance. Second, manual configurations are done at a coarse granularity (*i.e.* entire LUNs, database tables, or files), which are not equally hot at a finer granularity and therefore don't justify the cost of using cache space. We automate hot data detection and migration at a finer granularity.

The first challenge is to design policies that will select the hot data across varied workloads and storage configurations. The next challenge is to adapt policy parameters when workloads, storage configurations, or client demands change to maximize cache access density and justify price/performance. Another challenge is to achieve exclusivity with existing disk array caches, so that SAN resources are utilized effectively. This requires understanding multi-level caching issues that we will discuss in the next section. Other challenges are discussed as future work. These include using SANCache with decentralized storage systems [21] and providing dynamic cache allocation among multiple LUNs.

There are also other design requirements, since SANCache will integrate into an online enterprise SAN. First, migration overheads should be minimized to prevent adverse effects on foreground I/O process. By controlling the migration rate we can avoid saturation of disks, array controllers, and fabric resources. Second, all changes should be done when the system is online, since there is little or no maintenance window for 24×7 data centers. SANCache does not modify client hosts or backend storage and can work in a heterogeneous host-array environment. Finally, we should be able to boost selectively for hosts, LUNs or target storage devices, to facilitate workload isolation and differentiation at different levels.

3 SANCache Design Features

SANCache design was driven by the above motivations and goals, which we can summarize as automation of hot data management tasks in SANs. The design is identified by the following features:

- *SANCache policies are implemented at a layer above the disk arrays (e.g. virtualization layer)* to observe all SAN I/O traffic. An analogy can be made between the SANCache resource shared by disk arrays and the array controller cache shared by disks in that array. However, array cache is shared by all LUNs in the array, whereas only enabled LUNs are allowed to use SANCache. The problem with array cache is that the cache is allocated among the LUNs according to their relative I/O rates. In addition, the SANCache SSD resource is provisioned independently of servers and disks. Therefore, expanding the resource itself and the amount allocated for a LUN is relatively easy.

- *Threshold-based hot data selection policy;* note that disk array and server caches use a demand-based placement policy, *i.e.* they place an object (page, block, *etc.*) into the cache as soon as it is requested or used once. Our goal is to complement this recency-based placement policy with a longer-term, more selective frequency-based policy by delaying the migration until an adaptive threshold value is reached.

- *SANCache selects and caches relatively larger chunks* (128KB-1MB) compared to 1-16KB cache lines used by disk array caches. Because chunks are relatively larger than most I/O request sizes, migration results in prefetching based on spatial locality. Larger chunks also have less metadata overhead and they benefit from higher disk bandwidths during transfers.

- *Differentiating between sequential and non-sequential streams to complement disk arrays.* Disk arrays perform better with sequential accesses as a result of seek-time compensation and read-ahead policy. Therefore, we direct longer sequential I/O streams to disks and streams with high access density to SSD. Figure 1 shows how we differentiate between access counts for random and sequential streams.

- *Periodic decrement of access counts*, which we call *cooling*, to increase access density. A detailed discussion of this technique can be found in Section 3.3.

- *Migration rate control.* Currently, we throttle migrations by reducing the number of threads allocated for them. Doing strict bandwidth-control on migration I/O after threads were scheduled caused client I/Os to wait longer, since these threads held locks to chunks being migrated. A better approach is simply to defer some of the migrations if there isn't enough bandwidth.

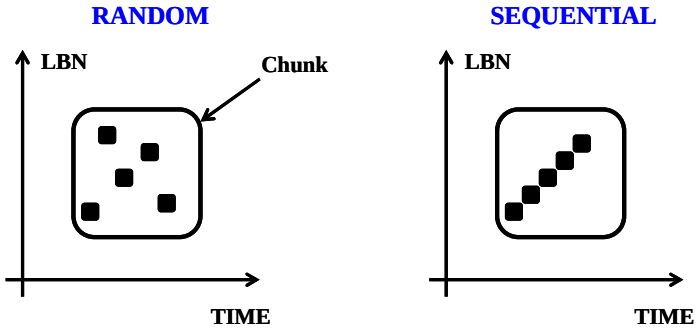

Figure 1: There are five accesses in both cases. We count the sequential run as one access to prevent chunks of this type to reach migration threshold and go to SSD.

3.1 Multi-Level Caching

Today, the focus of caching research is on replacement policies. Improvements over Least Recently Used (LRU) or Least Frequently Used (LFU) replacement algorithms are continuously being made [4,5,6,12,16,18,20]. In this body of work, the cache *placement* decision is usually taken to be demand-based. We can generalize demand-based placement policy by deferring the migration until a threshold value (T) is reached. We call this type of placement policy threshold-based or Most Frequently Used (MFU) placement. Demand-based placement is a special case of MFU where T=1. To keep most frequently used items in the cache, we could also use a combination of demand-based placements and LFU replacement. However, the SANCache environment motivates a combination of MFU-type placement and a low-overhead replacement policy. Demand-based placement of large chunks into SSD would be too aggressive.

Replacement algorithms are about managing one cache space. However, in distributed systems multi-level caching scenarios are very common. It is usually not easy to change cache sizes or policies due to organizational and physical boundaries between levels. Figure 2 illustrates such a scenario in a SAN occurring with the SANCache design. Disk arrays have fixed-capacity, LRU-managed, and demand-based caches. It would be a hard, long process to update policies inside these arrays to achieve our goals. A SAN-level, memory-based cache that is shared by disk arrays provides a solution, but this cache needs to complement array caches for improved utilization of both resources. Note that, this cooperative scenario has a similar goal to that of novel two-list algorithms (ARC [18], LRU-2 [20], 2Q [12]), which is to combine recency and frequency and make two lists exclusive. However, our lists reside on different devices creating a multi-level caching scenario.

The physical boundary that separates multi-level caches creates additional challenges. The first challenge is to provide exclusivity of the two cache levels and second is to minimize network overheads of data migration. There is no easy way to track the contents of a disk array's cache or insure eviction once a data object has been migrated to the SSD. The SANCache design features help address these issues. In comparison, a two-list algorithm merely needs to modify a memory pointer to emulate the effect of moving the object from the first to the second list in a single physical cache.

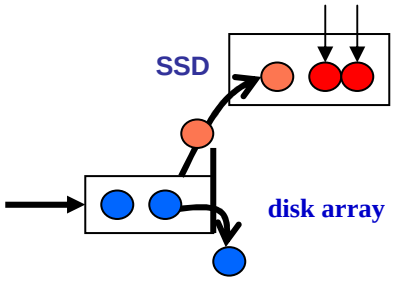

Figure 2: A multi-level caching scenario where the first level is the disk-array cache and the second level is the SAN-level cache implemented on a solid-state disk.

3.2 Operation for Boosted LUNs

Hot data migration operations can be categorized into three stages: before, during, and after data migrations. Before the migration begins, a module on the data path monitors clients' access patterns and makes placement decisions. A placement policy in the migration module determines candidates for migration into the SSD and adds them to a queue. During migrations, the module controls the migration rate and issues locks to assure data consistency. After the migration, some objects reside in the cache potentially providing reduced access latency. The same module monitors cache accesses (hits and misses) and makes eviction decisions based on a replacement policy such as LRU or CLOCK [5].

Some I/O requests span multiple chunks. It is very unusual for a request to span more than two chunks. If both chunks are in the SSD (hit-hit), the result is a cache hit. The response may still have to be constructed from two separate locations on SSD. If one of them misses the cache (Hit-Miss), then the missing chunk is fetched from disk immediately and the I/O completes from SSD. Since the response was slow, the result is considered a cache miss. If both chunks are missed but the migration threshold is not reached, then access counts are updated and the request is completed from disks. If a spanning request also triggers migration, then both chunks are migrated.

3.3 MFU vs. MRFU

Figure 3 illustrates the effect of different filters on workloads. This figure was inspired by a mix of TPC-C and TPC-H (see Figure 1 in [7]) and workload characteristics. By filtering workloads, we can obtain sub-streams with different patterns. Each stream is then handled by the appropriate storage system, either disk-based or solid-state storage. In our architecture, we first apply a sequential filter to sift and direct sequential streams to disks. Next, we pass non-sequential streams from a cooling filter for additional improvement in selection before migration into the cache.

Basic MFU placement has a cache pollution problem similar to its replacement counterpart LFU algorithm, where infrequently used objects occupy cache space and cannot be evicted due to their long access history. Similarly, in MFU relatively cooler (lower IOPS) chunks eventually collect as many accesses as the threshold and get placed into the cache. These chunks only achieve a few hits, which does not justify the bandwidth and cache space costs associated with the migration. Therefore, we "cool" the migration-candidates list by periodically decrementing access counts of the listed candidates. This preferentially selects candidate cache objects with more recent access activity and clears the list from the unwanted, cooler chunks. It retains only the "most-recently frequently-used" chunks as candidates, so called MRFU. The access count is not decremented below zero, since chunks with no access count are not considered active.

Given the user I/O pattern, there are only two ways to increase the access density metric, *i.e.* IOs/(GB×sec). The sequential filter improves the *spatial* dimension by eliminating sequentially accessed chunks and the cooling filter improves the *temporal* dimension by selecting chunks with time-clustered accesses for migration.

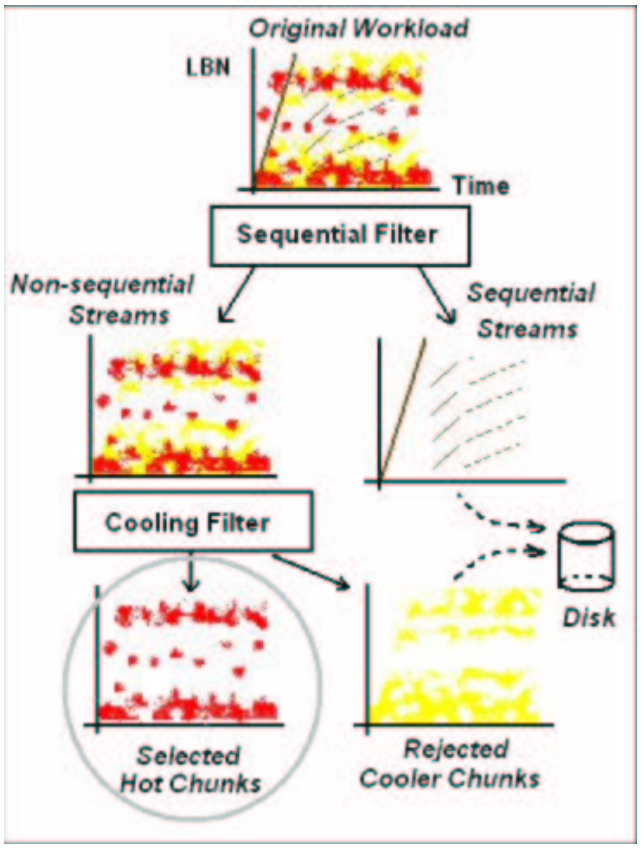

Figure 3: SANCache uses sequential and cooling filters to select hot chunks for SSD migration.

Compared to MFU, MRFU does not diminish the overall benefits (cache hits) obtained from cached objects, since data objects that are allowed in the cache by MFU are relatively cooler. MRFU objects achieve a higher hits/migration (benefit/cost) ratio compared to those achieved by MFU objects. Therefore, MRFU is not just a cache space management technique, but a better hierarchy resource management strategy. The importance of an efficient placement algorithm increases with decreased network bandwidth, decreased cache capacity and increased migration rate. Our technique can also be used in content distribution networks. The choice of replacement strategy to complement MRFU is beyond the scope of this paper, but we suggest usage of policies with low lock overhead such as CLOCK.

MRFU has two tunable parameters: the cooling period and the amount of cooling (reduction in access counts) per period. For example, we can reduce counts by 1 every 30 seconds. One may conceive of combining these two parameters by defining a new metric called *cooling rate*, which is equal to CoolingAmount/Period. However, the effects of the same cooling rate (*e.g.* 1/30, 20/600) with different decrements or periods may not be the same. Therefore, we are currently investigating the sensitivity of results to the cooling period and the relation between the period and the migration threshold via prototypes and simulations. Too much cooling could clear all candidates and stop migrations, while too little cooling would not be able to filter cold chunks. We can increase cooling by decreasing the period or by increasing the CoolingAmount.

Since the operation consists of a simple atomic decrement, thousands of chunks can be cooled in microseconds. In our implementation, cooling is applied by a separate thread and doesn't stall the foreground operation. Cooling is done once per a relatively-infrequent period (*e.g.* every 30 seconds). Furthermore, it is only applied to active chunks that maintain positive access counts under cooling. Increasing the cooling rate reduces the number of active chunks, hence the processing needs.

4 Evaluation

This section describes the SANCache prototype environment and workloads used for system evaluation. It also compares the static and adaptive threshold schemes and analyzes effects of the cooling method. Results related to performance and workload isolation are presented in the next section.

4.1 Prototype

We used two disk arrays in our prototype as disk-based backend storage. The first is a relatively smaller scale HP Modular SAN Array (MSA) 1000 with a 256MB cache and 8 18.2GB and 15,000 rpm disks. We used RAID0 (*i.e.* stripe-no-mirror) to maximize throughput and meet capacity goals of the benchmarks described below. The second array was a high-end HP XP1024 disk array with 125 disks and 32 GB of shared cache. We used an isolated disk group with 32 spindles and configured 3 LUNs with ~100GB total usable capacity as RAID1 (*i.e.* stripe-and-mirror).

To connect the backend to hosts and to implement SANCache policies, we used HP's Continuous Access Storage Appliance (CASA) [10], which is an in-band fabric-based SAN virtualization platform. SAN virtualization simplifies SAN management by pooling storage resources from heterogeneous backend devices and mapping partitions of storage to heterogeneous hosts (*e.g.* Windows, Linux, HP-UX, etc.). It was also easier to add our SANCache software module into the virtualization layer as a new policy than alternatively modifying the array firmware. Other tasks handled by virtualizers include local and remote mirroring, snapshots, and multi-path data availability. This appliance has 2 HP Proliant ML370 servers with 2.8GHz Intel Xeon processors. We used one of these servers to generate the benchmark traffic for some experiments and used other similar Intel-based servers for others. Most hosts have QLogic 2300 series dual-port Fibre Channel (FC) Host Bus Adapter (HBA) cards. Fabric virtualization can be *in-band* (on the data path) or *out-of-band* (distributed and coordinated). An out-of-band implementation of SANCache over decentralized disk arrays is currently in progress. We used the LRU replacement policy for the SSD.

A Solid-State Disk (SSD) is a DRAM-based storage device with a block interface and protection for non-volatility. For our prototype, we attached a Texas Memory Systems' (TMS) RamSan-320 SSD with a 16GB capacity to the backend FC SAN fabric and partitioned portions of it as a SANCache resource. This device has 4 FC controllers and 2 FC ports per controller. We measured ~130-150µs access latency with 512 byte random I/O on this SSD, whereas disks generally have millisecond latencies. SSD presents similar performance for

random and sequential loads, since it is memory based. TMS recently published impressive results for a similar box demonstrating ~120,000 IOPS and 960MB/sec bandwidth measured with the SPC storage benchmark described next.

4.2 Workloads

The Storage Performance Council (SPC) [22] publishes industry standard storage performance benchmarks. This benchmark emulates On-Line Transaction Processing (OLTP) style data access behavior. It is representative of multi-user OLTP, database and email applications. It uses 3 LUNs: LUN1 called data store with the most I/O activity (~60%) and mixed streams, LUN2 called user store with less activity and a similar mixed I/O pattern and LUN3 called log store with long sequential streams. SPC-1 defines 100% utilization and maximum I/O rate to be at 30ms response time.

Transaction Processing Performance Council's (TPC) [24] TPC-C benchmark is designed to simulate "a complete computing environment where a population of users executes transactions against a database". TPC-C involves a mix of five concurrent transactions of different types and complexity. *Warehouse* is a basic unit of scaling and the number of warehouses determines size of database tables and load applied to the system. The database is comprised of nine types of tables with a wide range of record and population sizes. Most active tables are the Customer and Stock tables. TPC-C results are measured in number of new-order transactions completed per minute. Conventional wisdom suggests that locality is captured in database server buffers, array cache is only a write buffer and storage sees a filtered random access pattern except for the Log. We show that SANCache achieves performance improvements even with this type of load.

4.3 Static vs. Adaptive Threshold

Chunks are migrated into SSD once their random access count reaches the migration threshold. We discuss implications of having a static threshold and motivate the adaptive threshold scheme used in SANCache. We used a 128KB chunk size for the following experiments, since 128KB can be fetched by most HBAs at once without fragmentation.

Figure 4 shows the effects of choosing low to high threshold values (4-32) on hit rates and the number of background migrations done to achieve these hit rates. While lower thresholds generate higher hit rates, ~55%, and reach the steady-state hit rate level quicker, they also have a larger steady-state migration-replacement rate, ~250 IOPS. For example, threshold 4 moves chunks at 5 times the speed of threshold 32 to achieve ~10% more hit rate. Higher thresholds are more conservative in selecting data, thus take longer to fill the cache (*e.g.* +10 minutes in Fig.4). While its saves network bandwidth and cache space, a conservative threshold also results in slower response to workload changes. To combine pros and avoid cons of different static thresholds, we resort to an adapting threshold parameter.

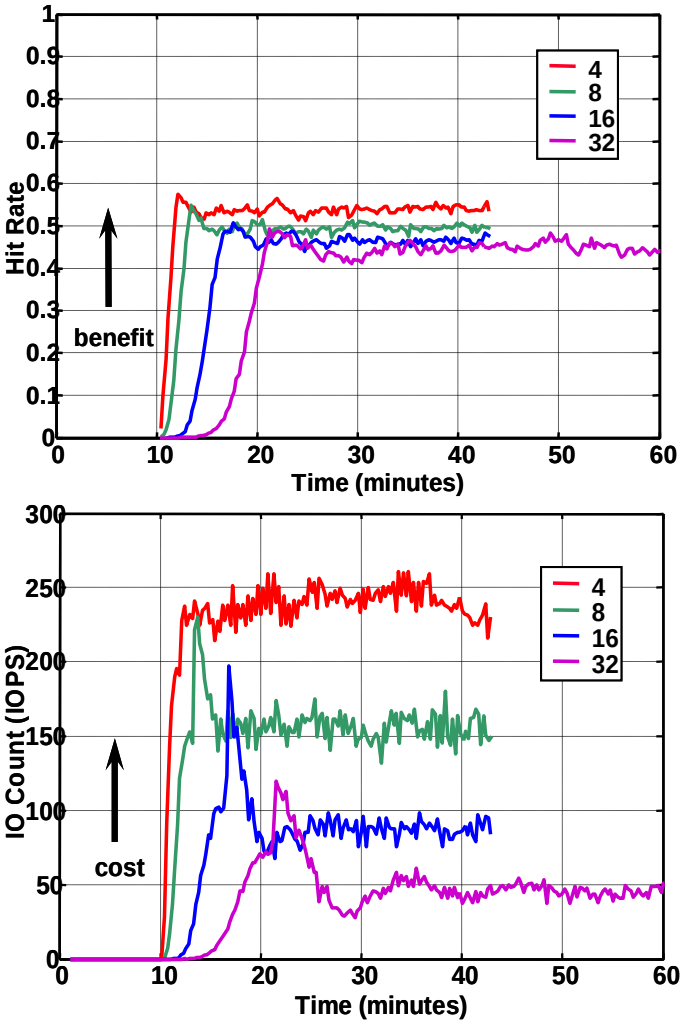

Figure 4: Effects of different static thresholds on cache hit rates (upper) and migration overheads (lower).

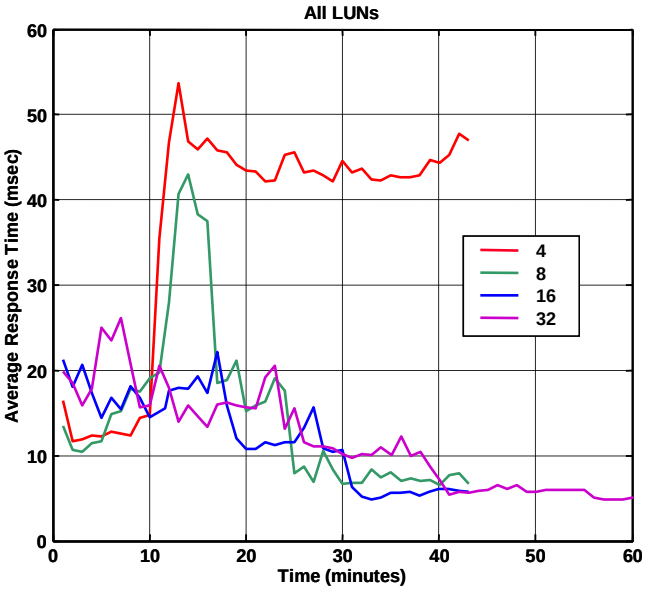

Figure 5: Effect of a poor static threshold choice on average client response time. (view in color)

Figure 5 shows the adverse effects of a poor static threshold choice on foreground I/O latency. In this example, the lower threshold (4) moved more data in favor of higher hit rates, but since most chunks were not hot enough, costs of migration were not justified by the hit rates and the overall response time of the system was crippled. These results were generated with the SPC-1 benchmark over the XP1024 disk array and using 2GB of SSD cache space.

4.3.1 Adaptation Rule

We provide an elementary threshold adaptation rule in Figure 6. Threshold is adapted periodically, every 15 seconds in the following evaluations. More chunks are migrated when the threshold is lowered and vice versa. Since SSD hits are counted periodically and the chunk size is fixed, the dimension of benefit to cost ratio (BCRatio) is equivalent to IOPS/GB, previously defined as the access density. Therefore, by tracking changes in workloads and storage configurations and adapting the migration threshold to these changes we maximize and maintain cache access density meeting one of our design goals.

If the BCRatio does not justify the current heat levels (*i.e.* < JUSTIFY), then the threshold is increased to slow down migrations and be more selective. If the BCRatio justifies, then we maintain the threshold, but also demand higher hit rates for the next period by increasing JUSTIFY. Finally, if there aren't enough migrations we lower the threshold and justify parameters to get data moving into the cache again.

There are no system or workload specific parameters in this formula. The algorithm finds good threshold values and the average heat of the working set in a workload by blindly probing it. This method has worked with a wide variety of workloads over different storage configurations. For larger cache sizes (*e.g.* 8-16GB) the BCRatio was quite high and it took JUSTIFY too long to reach and push the BCRatio higher. Therefore, we modified JUSTIFY as the running average of BCRatio. However, the running average in turn caused migration-rate oscillations for very dynamic workloads with high-threshold sensitivity, which we report here as potential avenues for improvement.

Figure 7 shows the adaptation rule in action. It illustrates that by adjusting THRESHOLD and JUSTIFY parameters the adaptation rule is able to achieve high BCRatio around 110 hits per migrated chunk. The benchmarks used had constant access rates and mixes of load, which caused the adaptation algorithm to reach a steady-state heat level and the threshold to stabilize around 18. This data was collected by running the TPC-C benchmark on the XP1024 and boosting Customer/Stock tables to a 2GB SANCache.

4.3.2 Adapting to Cache Size

We provide no expert hints to the adaptation rule including any hints about the amount of SSD cache space available or the active data set size in the workload. Figure 8 shows how the adaptation algorithm stabilizes at different steady-state values for different cache sizes although the workload and disk array are unchanged (SPC-1 over MSA).

```
Initialize:
BCRatio=0; JUSTIFY=0;

foreach period {
  Benefit = Δhits; //new hits
  Cost = Δmigs;    //new migrations
  BCRatio = Δhits/Δmigs; //trade-off

  if(Δmigs > JUSTIFY) {
    if(BCRatio < JUSTIFY)
     THRESHOLD++
    elseif (BCRatio >= JUSTIFY)
     JUSTIFY++
  } else {
     THRESHOLD--;
     JUSTIFY--;
  }
}
```

Figure 6: Migration threshold adaptation algorithm.

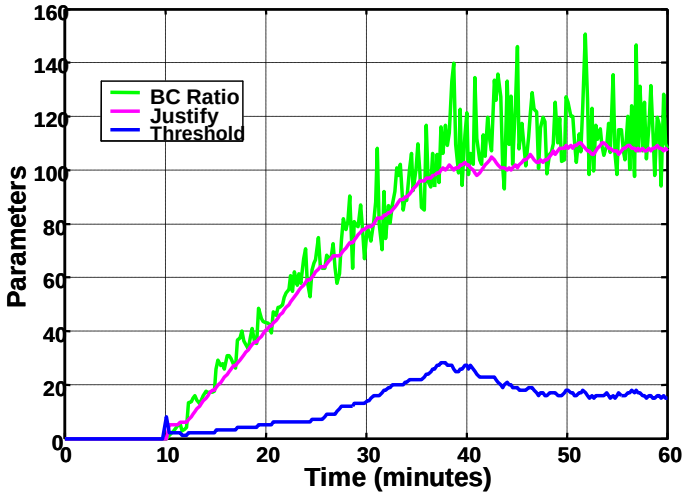

Figure 7: Adaptation rule in action. Benefit-Cost Ratio is improved by adapting the threshold to workload.

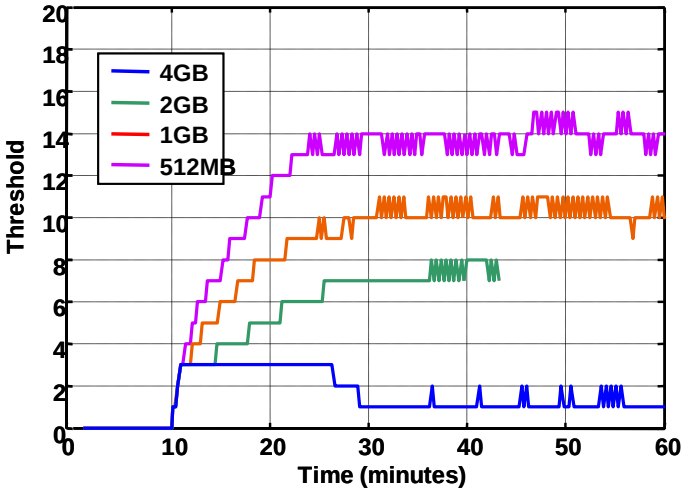

Figure 8: The algorithm stabilizes at different threshold values for different cache sizes. With smaller caches it increases the threshold to be more selective and compensate for the lack of resource and vice versa.

Initially the threshold is reset to 1 to indicate that a chunk will be migrated into SSD on a second hit. The results show that the adaptive algorithm is more selective when there is less cache resource. The threshold is around 1-2 for 4GB and 13-15 for 512MB cache sizes. Basically, due to the shorter residence times there will be more capacity misses and BCRatio will be lower on average pushing the threshold to higher values. The spikes in the graph occur, since the threshold is a discrete value and the BCRatio is a rational number.

4.4 Cooling

Figure 9 and Figure 10 compare MFU and MRFU based on the total amount of migrated chunks (GB) and hit rates, respectively. These results were obtained using the SPC load on the XP1024 disk array. We decrement accesses to active chunks by 1 every 15 seconds. Both placement policies used the same adaptive threshold adaptation rule.

Figure 9 compares MRFU and MFU based on total chunks migrated or their bandwidth usage. Savings achieved by MRFU over MFU placement are clear. After 1 hour MRFU moved about 30% (9.5 GBà 6.5GB) fewer chunks than MFU. Specifically, the cooler chunks were eliminated by MRFU from the candidate list before they used up any disk and network bandwidth for migration or occupied cache space. Figure 10 compares hit rates achieved by MRFU and MFU. MRFU achieved hit rates comparable to MFU (even slightly higher), although it migrated fewer chunks as evidenced by the data in Figure 9. Together, these two plots demonstrate that simple periodic cooling can successfully discriminate among chunks that reach a target access threshold and determine those that are currently hot or "most-recently frequently-used".

5 Results

SANCache has two main goals: to make networked storage fast on-demand and to provide isolation for workloads consolidated in a SAN. This section demonstrates results with real systems and shows improvements for both performance and isolation dimensions using SPC-1 and TPC-C benchmarks.

5.1 Performance Boosting

Storage performance is represented with IOPS-Response Time curves. When a disk array reaches its throughput limits the response time will shoot exponentially avoiding further increase. We show the effects of boosting client LUNs on throughput (IOPS) and average response time.

Figure 11 shows the operation of SANCache step-by-step by running the SPC-1 benchmark on the XP1024 disk array and plotting IOPS for LUN1. For 20 minutes SPC-1 runs with a constant rate of 2200 IOPS on LUN1 (total 3500 IOPS over 3 LUNs). After 20 minutes, LUN1 is boosted and the migration of hot chunks to SSD occurs in the background. The algorithm quickly moves chunks to exploit available cache space causing a bump and then reaches a steady-state rate.

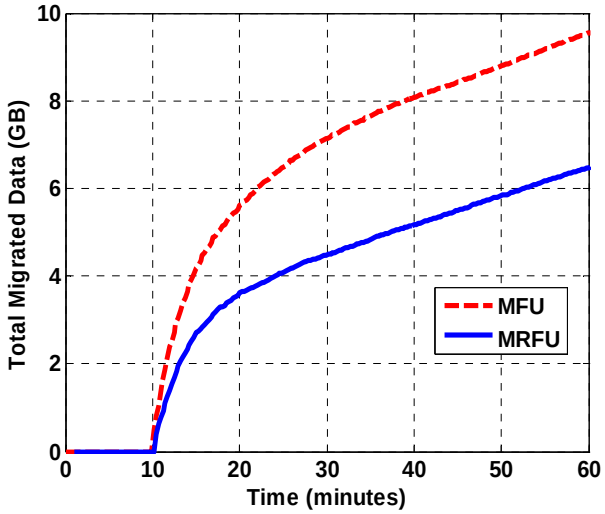

Figure 9: Comparison of MRFU (MFU + Cooling) and MFU policies based on total amount (GB) of migrated chunks, i.e. bandwidth overhead. After 60 minutes MRFU eliminated about 3GB of unnecessary migrations.

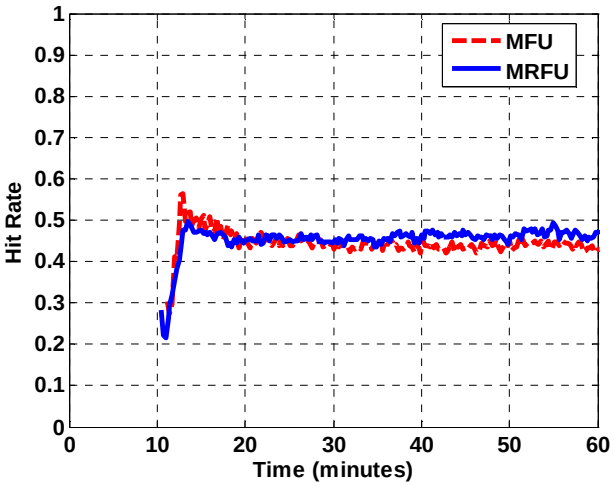

Figure 10: Hit rate comparison of MRFU and MFU. MRFU achieved even slightly higher hit rates with SPC-1 benchmark, while the goal was to perform just as good.

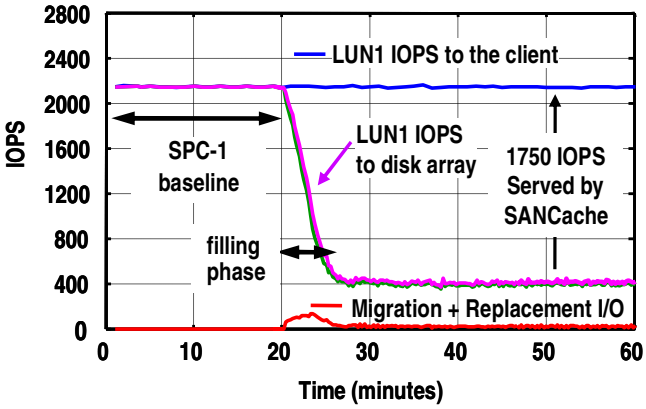

Figure 11: IOPS performance results before and after boosting most active LUN1 in SPC-1 benchmark.

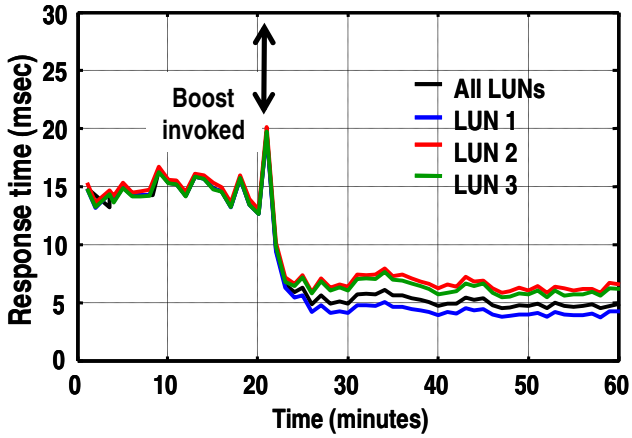

Figure 12: Average client response time before and after boosting most active LUN1 in SPC-1 benchmark.

After ~7 minutes I/O requests going to disk array are reduced to 450 IOPS (or by ~80%) and 1750 IOPS are served by the fast SANCache SSD resource. Overall disk array I/O reduction is about 1750/3500=50%. In this case, the client is not pushing for more IOPS and therefore the benefit is seen in the reduction of average response times from 15ms to 5ms by 3-fold as shown in Figure 12. Although only LUN1 was boosted all LUNs benefited significantly due to reduced utilization of the disk array cache and spindles. The initial bump in response time at boost time is expected as several table allocations and initializations are done at this time.

Figure 13 shows the effect of SANCache on throughput (IOPS) using SPC-1 and the HP MSA array. Maximum IOPS increased from 2000 to 3000 IOPS or by 50%. We do not show the disk IOPS curve for this experiment as it is similar to Figure 11. To summarize, after the boost 60% of LUN1 I/O traffic was being handled by the SSD at steady-state, which maps to 35% of the total I/O traffic.

Figure 14 shows throughput results for the TPC-C benchmark running at 800 warehouses with 20 users on a HP XP1024 disk array. At the 10 minute mark the LUNs that have the Customer and Stock tables are boosted. After another 10 minutes "total" IOPS increases from ~3900 to ~5000 IOPS (C&S+Misc+Log) and transaction per minute increases from ~8000 to ~9500 (see right vertical axis). We would like to emphasize that these improvements are being achieved with workloads believed to be very random (so caching would be useless) and only using a 2GB SSD space for SANCache to complement an array that already has 32 GB cache. The initial dip in performance is again attributed to the initial migrations and initialization routines that create the hash tables and set many parameters.

5.2 Workload Isolation

This section demonstrates another dimension of SANCache benefits, which is its potential use for workload isolation. Figure 15 illustrates the scenario.

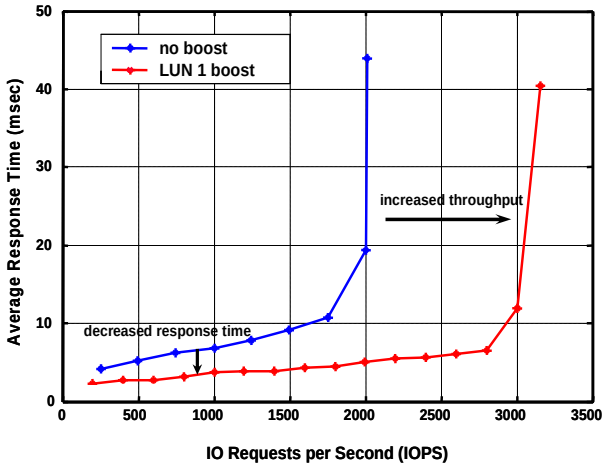

Figure 13: Effects of SANCache boosting on MSA storage throughput. Migrating hot data to a 2GB SSD partition resulted in 50% improvement of overall IOPS.

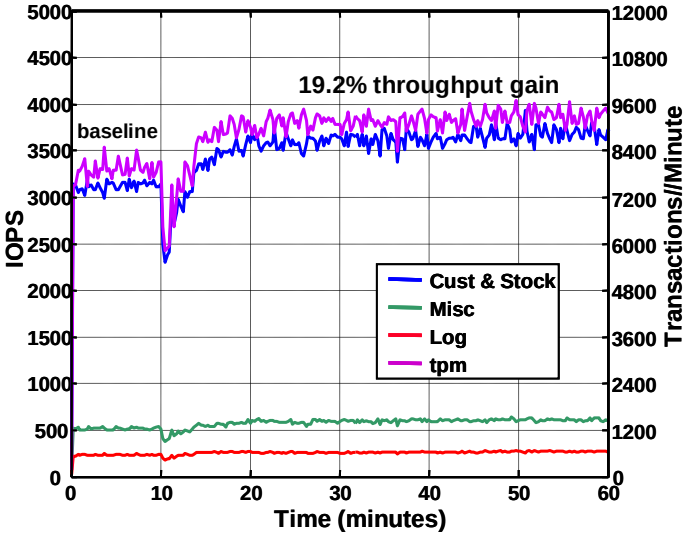

Figure 14: Effect of boosting Customer-Stock database tables to 2GB SSD on overall TPC-C performance.

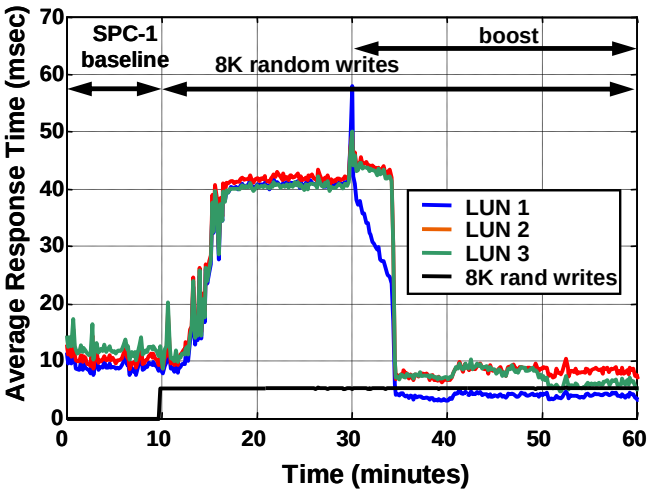

Figure 15: Alleviating cache interference and providing workload isolation using SANCache.

While the SPC-1 workload is operating on the XP1024 array, a write-intensive workload is introduced at the 10 minute mark. The write workload is on a different LUN, port, and set of disk groups, but it shares the array's 32 GB of cache with SPC-1. The cache quickly fills, causing the array to flush to its disks with limited IOPS. This builds queue length, adversely impacting response time seen in the increase from 10 ms to 40 ms. Boosting the SPC-1 LUN1 with 4 GB of SANCache (at 30 minutes) absorbs I/O load from the XP array, restoring acceptable response time to SPC-1 and allowing the interfering load to continue at its initial rate.

6 Related Work

The SANCache design has similarities to some novel and emerging replacement algorithms [4,5,6,12,16,18,20]. For example, the goal of two-list replacement algorithms such as Adaptive Replacement Cache (ARC) [18] is to capture both recency and frequency types of locality and this goal is also reflected in the SANCache design, where the array cache represents the recency component and the SSD contents represent the longer-term frequency component. However, the two lists of SANCache reside on different devices. Another group of replacement algorithms including Unified Buffer Management (UBM) [14] and Program-Counter-based Classification (PCC) [7] split workloads into sequential, looping and other types of patterns and use sub-caches to handle each pattern separately. Workload filters introduced in SANCache also split workloads into sub-streams (sequential, high-access density, others) and direct each stream to the storage device that can handle that type of stream well.

Most replacement policies focus on making local caching decisions and ignore multi-level or global performance implications. Wong and Wilkes investigate exclusive caching in a SAN context [25]. In a two-level cache scenario, where the second level is a disk array cache, they evaluate usage of heterogeneous replacement policies in each level. They find that using different policies (*e.g.* LRU and MRU) is better than using the same policy (*e.g.* LRU-LRU) for different levels. They further improve exclusivity of the two-level cache via demotions, which refers to moving objects from first to second level so that each level has unique set of objects. Demotions increase network bandwidth usage and require modifying the cache management routines in servers and disk arrays.

Our previous work also shows benefits of using heterogeneous policies for multi-level caches and promotes employing adaptive caching using multiple replacement policies as experts [2,8] for collaboration among levels without communication. In another related work [3], we compare the SANCache style cache management with replacement policies such as LRU and ARC [18] using analysis and simulations.

Korupolu and Dahlin define a similar MFU placement algorithm called MFUPlace ([15], page2) as follows: "The distinction between this strategy (LFU replacement) and the MFU placement strategy is that, while the MFU placement strategy picks a set of objects to place in the cache and leaves them undisturbed throughout that period, the LFU replacement strategy can change the contents of the cache after each request." In our definition of MFU placement, objects can enter the cache once they hit the access threshold, *i.e.* caching can occur anytime irrespective of the placement period. Therefore, our cooling strategy can be applied to their MFUPlace algorithm as well, since this algorithm is potentially prone to the cache pollution problem.

The relationship between MFU and MRFU is similar to the relationship between LFU and LFU with Dynamic Aging (LFUDA) [4]. The latter algorithms, MRFU and LFUDA, avoid the cache pollution problem of the former. However, the "aging" mechanism in LFUDA and the "cooling" mechanism in MRFU operate in completely opposite ways: Cooling decrements access counts of candidate objects to select the hot objects for placement before the cache placement occurs. Aging increments access counts of cached objects to select the cold objects for replacement after the cache placement occurs. LFUDA ages the cache to evict objects (pages, blocks), which cause the cache pollution, whereas MRFU solves a similar problem before the cache placement is made.

Hierarchical Storage Management (HSM) is also related to our work, but the term usually refers to migration of cold data from online to near-line or to offline storage in practice. Our research focuses on moving data from disk-based storage to faster, non-volatile and memory-based storage technologies such as DRAM-based SSDs [23] or other emerging NVRAM technologies [19].

Our work has Quality of Storage Service (QoSS) implications and therefore relates to several projects in that field. Triage [13] provides workload performance isolation and differentiation. It uses control theory to throttle requests from different clients to meet SLAs goals, but differentiation can occur within the performance limits of the reserved storage. Triage delays requests of less delay-sensitive applications. SANCache can increase the performance of a selected client beyond disk storage performance. Aqueduct [17] also meets delay contracts by using control theory. It controls migration of colder data for backups, failure recovery and load balancing. CacheCOW [9], which is inspired by the ARC replacement algorithm, differentiates classes of workloads (COW) by dynamically allocating cache spaces for LUNs in a SAN. It estimates required hit rates, derives new cache allocations and either gives or claims space to meet QoS goals. When all QoS goals are met, it allocates extra space to maximize overall hit rate. It uses demand-based placement and depends on building a hit rate history for accurate estimations of cache space allocations. It is challenging to come up with good workload estimates in todays SAN, where workloads will interfere with each other.

In some high-end disk arrays including the HP XP series, array cache can be partitioned by using the array's management tools. In one use case, this partition will serve as a RAMDisk or a "Cache LUN". In another use case, the partitioned cache will be mapped to a selected set of disks in the array in order to carve out smaller isolated disk arrays from larger ones. While this feature is an improvement over existing disk arrays, it has several disadvantages when

compared to the SANCache method. First, it requires modifying existing disk arrays and would only be useful for that array and not the whole SAN. Next, the cache cannot be provisioned (expanded, protected, purchased) independently from the disks as in network-attached SSD. Finally, when loads are isolated using separate spindles the workloads will not benefit from all available spindles and disk capacity will be under-utilized.

7 Ongoing and Future Work

In this paper, SANCache hot data selection and placement policies were shown to boost performance of traditional disk arrays. However, it is challenging to implement SANCache over decentralized storage systems such as Federated Array of Bricks (FAB) [21]. SANCache requires lots of fine-granularity data migrations and stresses metadata management when the underlying storage is distributed. We are currently designing and implementing a light-weight, distributed metadata management algorithm to run SANCache over FAB to provide performance improvements.

In this paper, we only introduced the hot data management mechanisms and showed preliminary workload isolation results. We haven't fully addressed QoSS issues related to the business aspects (*e.g.* different quality metrics, contracts-SLAs, utility functions, *etc.*) of this topic. In our current prototype, we implemented APIs where users can enter their (minimum) target IOPS and (maximum) latency goals. SANCache automatically allocates cache space to client LUNs to meet their storage performance goals and depends on maximizing global utility when there is resource contention.

8 Conclusions

We addressed performance problems in SANs due to workload interferences and throughput limitations of disk-based storage. Our approach was to place a SAN-level cache device that will complement array caches. All operations were done when the system was online and the disk arrays were not modified.

We prototyped SANCache and implemented its techniques on the HP-CASA virtualization platform. We demonstrated performance improvement and workload isolation results with SANCache by running industry standard SPC-1 and TPC-C benchmarks over two commercial disk arrays. We are continuing our design with guidance from measurement and simulation. We believe that a balanced storage system design with the right amounts of cache and disks for today's complex, 24×7 SAN can only be achieved though automation of data management tasks and adaptation of system parameters to track changes in workloads and storage configurations.

References

[1] 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.

[2] I. Ari, A. Amer, R. B. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: Adaptive Caching Using Multiple Experts. Workshop on Distributed Data and Structures, (WDAS) 2002 143-158.

[3] I. Ari, M. Gottwals, and D. Henze, SANBoost: Automated SAN-Level Caching in Storage Area Networks, In Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC) 2004 164-171.

[4] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In Proceedings of the 2nd Workshop on Internet Server Performance (WISP), Atlanta, Georgia, May 1999.

[5] S. Bansal and D. S. Modha, CAR: Clock with Adaptive Replacement. In Proceedings of the 2004 Conference on File and Storage Technologies (FAST) 187-200.

[6] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS) 193–206, Dec. 1997.

[7] C. Gniady, A. R. Butt, and Y. C. Hu, Program-Counter-Based Pattern Classification in Buffer Caching, In Proceedings of OSDI 2004.

[8] R. B. Gramacy, M. K. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In Neural Information Processing Systems (NIPS) 2002. 1465-1472

[9] P. Goyal, D. Jadav, D. Modha, R. Tewari, CacheCOW: QoS for Storage System Caches, In Proceedings of 11th International Workshop on Quality of Service (IWQoS), Monterey, CA, 2003

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

[11] HP Utility Computing Services, http://www.hp.com

[12] 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) 439–450, Santiago, Chile, 1994.

[13] M. Karlsson, C. Karamanolis and X. Zhu. Triage: Performance Isolation and Differentiation for Storage Systems. In Proceedings of the 12th IEEE International Workshop on Quality of Service (IWQoS). Montreal, Canada, 2004

[14] J. M. Kim, J. Choi, J. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. A low-overhead, high-performance unified buffer management scheme that exploits sequential and looping references. In Proceedings of OSDI 2000.

[15] M. R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No 6, Nov/Dec 2002.

[16] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, C. S. Kim. LRFU (Least Recently/Frequently Used) Replacement Policy: A Spectrum of Block Replacement Policies. In IEEE Transactions on Computers, Vol. 50, No 12, 1352-1360, December 2001.

[17] 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.

[18] 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) 115–130.

[19] 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

[20] 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, 297–306, 1993.

[21] Y. Saito, S. Frølund , A. C. Veitch, A. Merchant, and S. Spence: FAB: building distributed enterprise disk arrays from commodity components. ASPLOS 2004.

[22] SPC benchmark-1 specification, http://www.storage performance.org/specification.html

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

[24] Transaction Processing Performance Council, TPC-C benchmark, http://www.tpc.org

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