An Out-of-band Approach to SAN-level Cache Management*

Da Xiao
xiaoda99@mails.tsinghua.edu.cn

Jiwu Shu
shujw@tsinghua.edu.cn

Wei Xue
xuewei@tsinghua.edu.cn

Weimin Zheng
zwm-dcs@tsinghua.edu.cn

Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China

Abstract

SAN-level cache is a new caching approach to reduce average response time when accessing storage in Storage Area Networks (SANs). Current SAN-level caching schemes (in-band) have the advantage of flexibility in control but suffer from poor scalability. In this paper, we present an out-of-band SAN-level cache model. It separates cache management decision making from decision executing to enhance scalability, while flexibility is retained through the real-time address mapping mechanism. We also present a cache placement and replacement algorithm called Access and Cache Queue (ACQ) that is suitable for the model. The algorithm uses frequency-based admission control to reduce the frequency of costly cache loading operations with low temporal and spatial complexity. Our simulation results show that ACQ has a hit rate only 1.5% lower than the Frequency-Based Replacement (FBR) algorithm, while the load rate is 90% lower when the cache capacity is one tenth of the total storage capacity. The response time of the model using ACQ increases much more slowly than that of the in-band SAN-level cache with concurrent accesses.

1. Introduction

Caches are commonly used in Storage Area Networks(SANs) to reduce the average response time when accessing storage, such as buffer caches in hosts' operating systems, caches in disk arrays, and caches on SCSI disks. Despite their variety, all these caches are localized to a single component of the SAN and thus can not exploit global storage access information to increase efficiency. Recently a new caching approach for the SAN environment called SAN-level caching has been studied to address this problem. SAN-level caching tries to identify globally hot data blocks and keep them in a dedicated cache to utilize cache resources more efficiently.

In current SAN-level caching schemes [4, 11], the cache controller, which makes cache management decisions (placement and replacement), is usually integrated with a virtualization appliance (also called a redirector) sitting in the data path between hosts and storage. The redirector provides virtual volumes to hosts by redirecting commands and data blocks. It is also responsible for loading data from disks when a cache miss occurs. Cache resources are proprietary to the redirector and transparent to hosts. This SAN-level caching approach will be referred to as the in-band cache model in the rest this paper.

The in-band cache model has the advantage of flexibility in control but a major problem with it is scalability. Due to the fact that the redirector is in the data path and the cache resource is proprietary to it, it is responsible for fetching data from disks on cache misses and loading it into cache when needed, as well as making cache placement and replacement decisions. The heavy load on the redirector makes it a bottleneck as the number of concurrent accesses increases, offsetting potential performance gains through caching.

Out-of-band SAN virtualization has gained popularity recently because of its good scalability and high performance [2, 10]. In an out-of-band virtualization system, a metadata server (MDS) is responsible for managing the mapping from virtual volumes to physical disks, while hosts access storage devices directly through the SAN. This method inspired us to design a new SAN-level cache architecture to address the problem of poor scalability in the in-band cache model. However, because the MDS is not in the data path, it lacks the flexibility of the in-band cache

*This work is partially supported by the National Natural Science Foundation of China under Grant No. 60473101 and 10576018; the National Grand Fundamental Research 973 Program of China under Grant No. 2004CB318205

model. Further study is needed on how to use the MDS to monitor hosts' access to storage and identify hot data to be loaded to cache. It is also necessary to find new cache placement and replacement algorithms that are suitable for the new architecture.

In this paper, we present an out-of-band SAN-level cache model. In the model, cache resource is accessible to all the hosts. Cache management decisions are made by the MDS out of the data path while hosts move the actual data, and thus scalability is enhanced. Hot data identification and redirection is done through a mechanism called real-time address mapping to retain the flexibility of the in-band cache model. We also present a cache management algorithm called Access and Cache Queue (ACQ) that is suitable for the model. The algorithm uses frequency-based admission control to reduce the frequency of costly cache loading operations in the model while maintaining a hit rate comparable to traditional cache management algorithms with low temporal and spatial complexity.

2. Model Design

2.1. SAN Virtualization Architecture

The virtualization architecture on which the out-of-band cache model is based consists of two cooperating components: a metadata server (MDS) which is responsible for metadata management, and the virtualization agent on each host (VA) which provides virtual volumes to the host file system. Hosts access storage devices directly through the SAN for data with the help of the VA, while the VA contacts the MDS for metadata about virtual to physical address mapping through another network (such as an Ethernet) for control. The VA is a kernel module in the host's operating system. When it receives an I/O request from the upper layer (file system/database), it relays the request to the MDS, where the virtual address in the command is translated into the physical address. Note that only the commands are relayed, not the actual read/write data. Hence the MDS is not in the data path. The modified command is then sent back to the VA and the new request is sent to the appropriate disks for execution.

This process differs from the typical block-level out-of-band virtualization approach in [2] in that each I/O request is sent to the MDS for address mapping rather than being mapped locally by the VA on the host. Through this real-time address mapping mechanism, though the MDS is not in the data path, virtually every I/O command passes through it except for the actual data. So it has a better overall view of the storage access situation and thus can make better decisions on which data blocks are global hot spots and should be loaded to cache. The mapping from virtual to physical address can also be changed dynamically by the MDS, preserving the flexibility in control in the in-band approach.

2.2. Out-of-band Cache Model

We incorporated a caching mechanism into the aforementioned virtualization architecture by adding another component - one or more cache disks that are attached directly to the SAN and are accessible to all the hosts. We call our cache resource cache disks because we use them in exactly the same way as plain disks. Unlike the cache resources in the in-band cache model that are only available to the redirector, our cache disks are shared among all the hosts. This offers the opportunity of data loading by hosts to avoid the bottleneck of MDS in the in-band cache model.

These cache disks can be implemented using DRAM-based storage technologies such as Solid-State Disk (SSD) [1, 3]. SSD has random access time measured in tens of microseconds and can achieve large storage capacity (to tens or hundreds of gigabytes). Moreover, SSD speaks SCSI or FCP protocols. The capacity of our cache resource can be easily expanded by adding additional cache disks to the SAN. These features make it an ideal medium for implementing a SAN-level cache.

Besides maintaining metadata about virtual to physical address mapping, the MDS also does bookkeeping of storage access statistics. It uses this information to identify hot data blocks to be cached. Once a hot block is identified, the MDS loads the block to the cache disk and updates its virtualization metadata. On receiving subsequent requests to the already cached blocks, the MDS maps the virtual address to the cache address, instructing the VA on the host to contact the cache disk for data.

A problem with the model is the communication overhead. A round trip time for sending an I/O request between the VA and MDS is introduced. However, the average round trip time is round 50 sec, while the disk service time is 8 msec on the average. Although this round trip time is nontrivial on a cache hit, it is acceptable on average considering the relatively low cache hit rate in second-level caches. This will be further evaluated in Section 5.

In the above operating mode (Mode 1), the loading of hot data blocks to the cache disk is done by the MDS. In fact, the burden on the MDS can be further relieved by offloading this task to the VA, because the cache disks are accessible to all the hosts. We present another operating mode (Mode 2), in which the VA performs data loading instead of the MDS. In Mode 2, when the MDS identifies a hot block, it sends a command to the VA that have sent the last request, instructing it to load a particular data block to a specified cache disk position. The VA replies to the MDS after finishing the loading task. On receiving such a reply, the MDS updates the virtualization metadata it maintains

and all subsequent requests to this block will be redirected by the MDS to the cache disk. All commands and replies can be piggybacked on address mapping requests and responses to reduce the number of messages passed between the MDS and the VA.

Mode 2 removes the MDS completely from the data path. It separates caching decision execution from decision making, which are done by the VA and the MDS respectively. Mode 2 is more scalable than Mode 1 because less work is done by the MDS. However, metadata consistency must be handled when multiple VAs load data simultaneously, so management complexity is increased.

3. Cache Management Algorithm: ACQ

The new model provides potential scalability improvement. However, because of the totally new architecture, a suitable cache management algorithm that is different from the ones used in the in-band cache model has to be devised to fully exploit the potential of the new model.

With demand caching typically used in the in-band cache model, hosts' requests that are missing from the cache are fetched from the disk and placed in the cache at once. However, this placement policy is not suitable for the out-of-band cache model. As the MDS is not in the data path, the operation of loading a block into the cache is more costly in the out-of-band cache model than in the in-band model. In Mode 1, this incurs an additional disk access done by the MDS, while in Mode 2 the cost includes sending a loading command and reply between the MDS and VA and maintaining metadata consistency. Due to the relatively expensive loading operations, a more strict cache admission checking should be enforced to allow only hot blocks to be loaded to cache disks. In other words, the placement algorithm should have a low load rate (the ratio of loading operation count to total access count).

The SAN-level cache belongs to second-level caches [11] in a cache hierarchy in a SAN. It was pointed out in [11] that accesses to a second-level cache exhibit poorer temporal locality than those to a first-level cache, and the access frequency distribution among blocks at the second-level cache is uneven. These facts indicate that instead of a locality-based replacement algorithm such as LRU, a frequency-based replacement algorithm should be employed.

It is also worth noting that the procedure of deciding cache placement and replacement is to be invoked every time a host requests a block. The overall performance of the system will be degraded despite the use of cache if the algorithm is too slow due to high computational complexity. Low storage overhead is also required of the algorithm for it to work in real systems.

```
/* Procedure to be invoked upon an request to block b */
if b is in cache queue CQ {
        b.reference ++;
        adjust blocks' reference in CQ;
        return;
}
if b is in access queue AQ {
        move b to head of AQ;
        b.reference ++;
} else {
        if AQ is full
                remove block at tail of AQ;
        put b at head of AQ;
        b.reference = 1;
}
if CQ is not full {
        remove b from AQ and insert b to CQ
        load content of b into cache disk;
} else {
        c = block in CQ with minimal reference;
        if b.reference >= c.reference {
                remove c from CQ;
                swap out space of c in cache disk;
                move b to CQ;
                load content of b into cache disk;;
        }
}
```

Figure 1. ACQ algorithm

In summary, a good placement and replacement algorithm for the out-of-band cache should have the following properties:

- A low load rate as well as a high hit rate
- Some form of frequency-based replacement algorithm
- Low temporal and spatial complexity

We have proposed an algorithm called Access and Cache Queue (ACQ) to meet the above requirements. Figure 1 gives a pseudo-code outline for the ACQ algorithm.

The main idea of the algorithm is to use a block's reference count as the measurement for its value. When a miss occurs, the block is allowed to enter the cache only if it is considered more valuable than some block in the cache. And the block with the least value is replaced. Here we use a block's reference count in a specific period of time to represent its access frequency.

There are two queues in the algorithm: an Access Queue (AQ), which is an LRU queue, and a Cache Queue (CQ), which is sorted by the blocks' reference count. AQ keeps recently accessed blocks' identifiers as the candidates for placement. CQ keeps already cached blocks' identifiers. Upon a reference to block b, ACQ first sees if b is already cached (in CQ). If so, b's reference is increased by 1 and the procedure returns. Otherwise, if b has recently been accessed (in AQ), b's reference is increased by 1 and b is moved to the head of AQ. If not, b is added to the head of AQ and its reference is initialized to 1. If AQ is full, the

block at the tail is removed. Then ACQ finds the block c in CQ whose reference count is minimal. b's reference is compared with c's and if b wins, b is added to CQ with its current reference and c is evicted from the cache in order to make room for b. Note that when there is still free space in cache, b is placed in the cache without comparison. Compared with the static threshold in SANBoost [4], the threshold changing according to the states of the blocks in the cache ensures that cache space is efficiently utilized, especially when the cache disk capacity is big enough to accommodate the entire working set of the workload.

Compared with demand caching, a block is allowed to enter the cache only when its reference count exceeds a certain threshold (the minimal reference count of cached blocks), so the load rate is reduced effectively. Meanwhile, access frequency is used as the criterion for replacement according to the second-level cache access pattern, and thus the blocks kept in the cache will have a relatively good chance of being referenced again. Hence a high hit rate is achieved.

Low temporal and spatial complexity is obtained by the use of access queue. As only the currently active blocks which have been accessed recently are taken into account when selecting hot blocks, ACQ does not have to maintain the access information of every allocated block. This enables ACQ to put all the information it needs in memory. With a moderate size of access queue the lookup can be done very quickly, which ensures the feasibility of the algorithm in a large-scale storage system.

4. Quantitive Analysis and Comparison

In this section, we present a preliminary analysis of the average response time of the out-of-band cache model as compared with that of the in-band model. For simplicity, we have made the following assumptions when describing system behavior: the time for a disk to serve a block request is a constant value, and all requests are reads. Thus, it is important to emphasize that the analysis is not designed to predict accurately the response times, but rather to comparatively analyze the two models.

- Let S be the round trip time of sending a request to the MDS and getting the response.
- Let D be the time to send a request to disk and retrieve the data for a particular block. Thus, this will be the time for a host to get data from the disk in the out-of-band model and also for the redirector to access data from the disk in the in-band model.
- Let C be the time to get the data for a particular block from the cache. This will be the time for a host to get data from the cache disk directly in the out-of-band model and also for a host to get data of a cached block from the redirector in the in-band model.
- Let Q be the queuing delay at the redirector/MDS due to contention with other simultaneous accesses.

Assume that the placement policy for the in-band model is demand caching. From simple analysis we can get the average response time RT_{IN} and RT_{OUT} for the in-band and out-of-band model respectively (r_1 is the hit rate and r_2 is the load rate):

$RT_{IN} = (1-r_1)*(D+Q_{IN}+C)+r_1*(Q_{IN}+C)$

$RT_{OUT} = (1-r_1-2)*(S+Q_{OUT}+D)+r_2*(S+Q_{OUT}+D+C)+r_1*(S+Q_{OUT}+C)$

Thus the average response times are given by:

$$RT_{IN} = Q_{IN}+(1-r_1)*D+C$$
$$RT_{OUT} = S+Q_{OUT}+(1-r_1)*D+(r_1+r_2)*C$$

We can also get the average processing time of each request, which is the time the redirector/MDS takes to process each request. We only give the processing time of Mode 1 in the out-of-band model because the processing time of Mode 2 is approximate to zero with a very fast lookup of AQ in memory. The processing times PT_{IN} and PT_{OUT} are:

$$PT_{IN} = (1-r_1)*D$$
$$PT_{OUT} = r_2*D$$

At first glance the analysis appears to be in favor of the in-band cache model since an additional round trip time of S is included in the out-of-band model. However, it is important to note that Q_{IN} could be much higher than Q_{OUT} due to the heavier load on the redirector. The work done by the MDS/redirector can be roughly measured as the processing time (PT) of each request. According to the expression of PT, we can make PT_{OUT} for Mode 1 smaller than PT_{IN} by lowering the load rate, which is one of the design goals of ACQ. PT_{OUT} for Mode 2 is even smaller, and therefore is negligible. The less work on the MDS prevents the exponential growing of queuing delay with increasing number of concurrent accesses on the redirector in the in-band cache model so better scalability is achieved.

5. Simulation Results

We have evaluated the feasibility and effectiveness of the proposed model and algorithm using synthetic-workload-based simulation. We used Zipf's workload [6] to simulate the uneven access frequency distribution in a second-level cache with the parameter $\alpha = 0.2$. That is, if there are N blocks, the probability of accessing a block numbered i or less is $(i/N)^{\alpha}$. We first explored the hit rate and load rate characteristic of ACQ and then compared the response time of the in-band and out-of-band cache model with an increasing number of hosts to evaluate the scal-

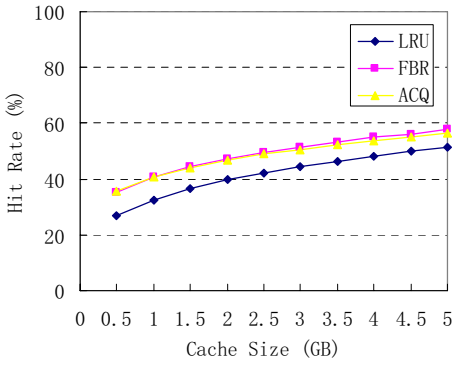

Figure 2. Cache hit rate

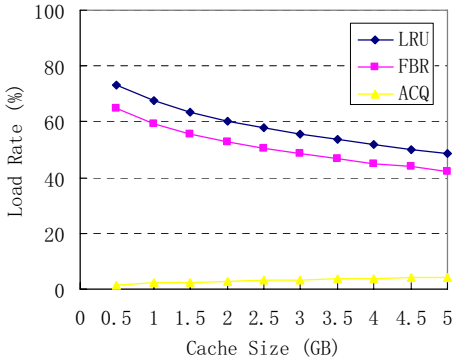

Figure 3. Cache load rate

ability. The parameters are set as follows: $S = 0.05ms$, $D = 8ms$, $C = 0.01ms$, $TotalDiskCapacity = 50GB$.

5.1. Cache Hit and Load Rate

We have implemented ACQ and two existing replacement algorithms LRU and FBR [9] in our simulation, and compared their hit and load rate with varying cache size.

We set AQ's size to be equal to the size of CQ based on the following observation we made during our simulation: the hit rate will grow with AQ's size but the slope of the curve becomes lower gradually. When AQ's size exceeds CQ's size, the hit rate almost stops growing. This indicates that with a moderate size of AQ, a relatively high hit rate can be achieved, which ensures a low computational and storage overhead of the algorithm.

Figure 2(a) shows the hit rate of the three algorithms with varying cache size and Figure 3(b) shows the load rate of the three, assuming on-demand placement is used for FBR and LRU. As shown in Figure 2(a), the ACQ and FBR algorithms have a hit rate very close to each other under all cache sizes, with ACQ being slightly lower (the difference is not more than 1.5 percent), while both of them outperform the LRU algorithm with an 8 percent improvement

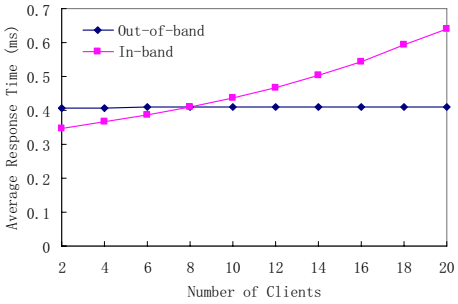

Figure 4. Average RT under varying number of clients

on average. In Figure 3(b), the load rate of ACQ increases slightly with increasing cache size but is much lower than the other two under all cache sizes. The slight increase of the load rate of ACQ can be explained as follows: as Zipf's workload is used, the portion of the top most frequently accessed blocks that should be kept in cache will grow when the cache size increases. So there will be a bigger chance that these blocks are accessed and loaded. With a 5 GBytes cache size, the load rates of FBR and ACQ are 42.2 and 4.2 percent respectively with a 90 percent decrease. We can conclude from the results that with a more strict admission checking, ACQ decreases the load rate significantly while maintaining a hit rate comparable to general replacement algorithms.

5.2. Scalability

We have compared the response time of the in-band and out-of-band cache model to explore the scalability of the new model. We made a rough estimation of queuing delay Q according to queuing theory: $Q = PT * \rho/(1-\rho)$, where $\rho = PT/AT$. PT is the processing time formulized in Section 4 and AT is the time interval between the arrivals of two successive requests. In the experiment, the in-band model used the FBR algorithm for replacement and the out-of-band model operated under Mode 1. A host issued requests at a rate of one request every 12.8 ms on average (thus $AT = 12.8ms/NumberOfClients$). The result is shown in Figure 4.

As shown in Figure 4, when few clients issue requests simultaneously (less than 8 under the above simulation settings), the average response time (RT) of the in-band model is shorter than that of the out-of-band model due to the communication overhead included in the latter. However, as the number grows, The RT of the in-band model grows much more rapidly than that of the in-band model. When there are 20 hosts, the former almost doubles while the latter increases by only 3.7 percent, which can hardly be seen in the figure. As analyzed in Section 4, the growing part

in RT is attributed to queuing delay. Therefore, the result indicates that the out-of-band cache model using the ACQ algorithm scales better than the in-band cache model with increasing concurrent accesses. It achieves this by giving the MDS less work so as to reduce the queuing delay.

6. Related Work

SAN-level cache management is a relatively new branch in the area of storage cache management and there has not been much work on it. SANBoost [4] is a SAN-level cache with frequency-based admission control using a static or dynamic threshold. It is a typical in-band caching approach where the appliance is in the data path. Our approach is more scalable than theirs because the MDS is removed from the data path. Additionally, in SANBoost, the rule for determining the dynamic threshold is hard to implement and the authors did not describe it in detail, while in our method, the admission threshold is chosen as the minimal reference count of cached blocks so that the cache capacity can be utilized more efficiently.

Zhou et al. studied access patterns of second-level buffer caches and presented a replacement algorithm for second-level cache called Multi-Queue (MQ) that outperforms most other algorithms [11]. MQ divides cached blocks into multiple queues with different priorities according to their access frequency. When a replacement occurs, the block at the head of the queue with the lowest priority is evicted. While MQ aims at increasing the hit rate by using a good replacement algorithm, our ACQ algorithm aims at reducing the load rate by combining placement and replacement algorithms. Besides, the architecture on which the MQ algorithm is based also falls into the in-band cache model.

Much research has been conducted on general cache management algorithms such as LRU-K [8], FBR [9], 2Q [5], ARC [7] and MQ [11]. Most of these studies have focused on cache replacement algorithms. Designed for different upper-level applications, they make improvements to LRU to enhance the hit rate. The work that comes closest to our ACQ is the Two Queue algorithm [5]. It uses two queues to allow only warm blocks to cache. The idea is similar to ours. 2Q can be considered as a frequency-based admission control algorithm with a static threshold of two. Our ACQ algorithm determines the threshold according to the access state of blocks in cache.

7. Conclusions and Future Work

In this paper, we have presented an out-of-band SAN-level cache model and a corresponding cache management algorithm. To the best of our knowledge, this is the first attempt to design a SAN-level cache with an out-of-band approach. Our approach addresses the scalability problem of the in-band cache by adopting a new architecture, in which cache placement and replace decisions are made by the MDS out of the data path. The decisions are made according to the ACQ algorithm so that the least blocks are loaded to meet the hit rate requirement. The simulation results show that the model is much more scalable than the in-band cache model. We are currently implementing a prototype of the model and plan to test the model and the algorithm on real benchmarks running on the prototype system. In addition, we are also investigating the policies of metadata consistency in Mode 2 in more detail and the cache consistency mechanism.

References

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

[2] StoreAge White Paper. High-Performance Storage Virtualization Architecture. http://www.storeage.com.

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

[4] I. Ari, M. Gottwals, and D. Henze. SANBoost: Automated SAN-level caching in storage area networks. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC '04)*, pages 164–171, 2004.

[5] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 439–450, 1994.

[6] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley, 1973.

[7] N. Megiddo and D. S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST03)*, pages 115–130, 2003.

[8] 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 1993 ACM SIGMOD International Conference on Management of Data*, pages 297–306, 1993.

[9] J. T. Robinson and M. V. Devarakonda. Data cache management using frequency-based replacement. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 134–142, 1990.

[10] G. Zhang, J. Shu, and W. Xue. MagicStore: A New Out-of-Band Virtualization System in SAN Environments. In *Proceedings of IFIP International Conference on Network and Parallel Computing2005 (NPC05), Lecture Notes in Computer Science*, volume 3779, pages 379–386, 2005.

[11] Y. Zhou, Z. Chen, and K. Li. Second-Level Buffer Cache Management. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):505–519, 2004.