Multi-Cache: Dynamic, Efficient Partitioning for Multi-Tier Caches in Consolidated VM Environments

Sundaresan Rajasekaran, Shaohua Duan, Wei Zhang, Timothy Wood

Department of Computer Science, The George Washington University
{sundarcs, shaohuaduan, zhangwei1984, timwood}@gwu.edu

***Abstract*—Every physical machine in today's typical datacenter is backed by storage devices with hundreds of Gigabytes to Terabytes in size. Data center vendors usually use hard disk drives for their back-end storage as it is cheap and reliable. However, the increase in the I/O accesses to the back-end storage from one or many of the VMs hosted on a physical machine can reduce its overall accesses time significantly due to contention. This may not be suitable for interactive applications requiring low latency that might be co-located with other I/O intensive applications.**

In this paper we present Multi-Cache, a multi-layer cache management system that uses a combination of cache devices of varied speed and cost such as solid state drives, non-volatile memories, etc to mitigate this problem. Multi-Cache partitions each device dynamically at runtime according to the workload of each VM and its priority. We use a heuristic optimization technique that ensures maximum utilization of the caches resulting in a high hit rate.

We use a weighted partitioning policy that improves latency by up to 72% for individual workloads, and a overall hit rate increase of up to 31% for host running several workloads together in comparison to standard LRU caching algorithms.

***Index Terms*—virtualization, flash-cache**

I. Introduction

As hardware increases in power, the number of VMs per host also keeps increasing. A typical host in a datacenter now packs tens to hundreds of VMs in order to maximize resource utilization; the popular VMware ESXi hypervisor has increased the number of VMs supported per host from 32 to 1024 [1] in recent years. To support this, hosts are now equipped with Terabytes of hard disk space and Pettabytes of externally attached storage systems.

While low cost hard drives have allowed data centers to meet this growing capacity, their I/O latency and throughput has stagnated. This is due to the hardware limitations of magnetic hard disk drives with spinning platters. Moreover, the fraction of random I/O operations on these devices tends to increase as more VMs are consolidated to a host. To overcome this limitation, studies have been done to analyze the feasibility of replacing the hard disk devices entirely with solid state drives (SSDs) [2]. Instead, a more practical approach that modern datacenters take to speed up the I/O accesses of the VMs is by using faster drives such as SSDs as caching devices on the hosts.

In the coming years, a deeper hierarchy of storage devices is expected to emerge, each with differing latency, throughput, price, and capacity characteristics [3]. This offers new opportunities for efficiently managing data center storage, but it also complicates the picture by providing a diverse set of options to choose from. The difficulty of building an efficient caching system is further compounded by the varied workload needs of different VMs. As a result, a VM with poor data locality and high random I/O, that cannot benefit from a cache might greedily take all the cache space, despite there being other VMs that could benefit more from it. Also, different VMs can have different priorities, so a cache management system should ensure that interference does not impact the Service Level Agreements (SLA) of the VMs.

In this work we present a multi-tier cache management solution that tries to solve these problems. Multi-Cache dynamically partitions a set of storage devices at runtime based on the VMs' workload and priority. The partitioning is based on workload features such as the long term locality and the intensity of recent bursts. Our contributions include:

- Workload characterization and cache utility models that predict how different VMs will benefit from each tier in the cache.
- An optimization framework and greedy heuristic that partitions the cache layers at runtime to maximize overall performance and account for priority levels.
- A simulation platform to evaluate our cache partitioning algorithm on cache devices of varying throughput and latency.

We have evaluated a range of workloads on our simulation platform to demonstrate the benefits of multiple cache layers, and the importance of intelligently managing how caches are shared by competing VMs. Our results show a latency increase of up to 72% for individual workloads that exhibit high data locality, and a overall hit rate of a host running several VM workloads of up to 31%.

II. Background

Data centers often use network storage devices to store the disks of virtual machines. Host side caching, where a fast local disk on each host is used to cache data for one or more virtual

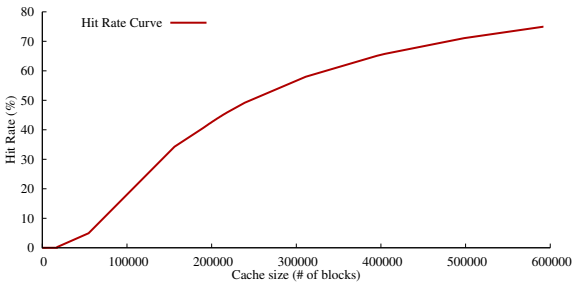


Fig. 1. Hit Rate Curve

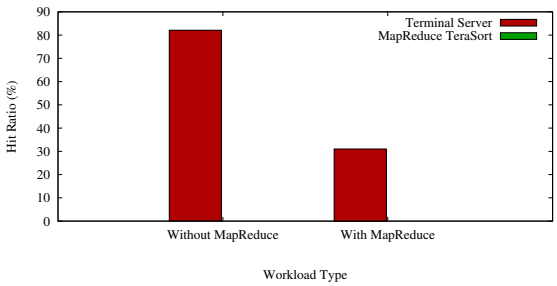


Fig. 2. Hit Rate of interactive VM getting affected due to poor cache management

machines, can improve the performance of I/O accesses since network latency and slow disk lookups are avoided [4], [5], [6]. Depending on the nature of the workloads even the simplest implementation of host side caching can bring significant improvement in performance [6]. One of the major factors affecting the performance is the hardware performance of the caching device itself.

The diversity of available storage devices that can be used for caching is increasing, ranging from SATA and PCIe SSDs to upcoming non-volatile memory (NVM) technologies. Each of these technologies have different throughput, latency, reliability, and cost trade-offs. Thus manually choosing an appropriate caching device and capacity with respect to the expected workload is difficult for several reasons. First, we cannot always predict the nature of the workload. Second, even if we could, workloads tend to be dynamic, so a static assignment of disks to virtual machines will be inefficient in the long term. Lastly, a host may run multiple VMs, so the cache must be effectively shared between these VMs without causing interference.

To mitigate this problem, automated software layer solutions have been proposed that try to divide the cache space of a single disk dynamically among different VMs based on the nature and the priority of the workloads running inside the VMs such that the overall I/O performance is maximized [7], [8], [9]. Our work extends these systems to take advantage of deepening storage hierarchy options, allowing several different caching devices to be partitioned among a set of virtual machines.

The effect of cache size on the performance of the applications have been studied rigorously for several decades [10], [11], [12], [13]. Hit Ratio Curves as a function of a cache size shows the hit rate an application has for a given cache size. Figure 1 shows an example of an Hit Ratio Curve(HRC). For a cache size that can hold 200,000 blocks, the given workload will achieve a 42% hit rate if the same workload is replayed. As long as a policy such as Least Recently Used (LRU) is used for eviction, the hit rate increases monotonically with the cache size, but it may level off based on the locality of accesses in the workload.

Mattson et al. [10] pioneered a technique to effectively calculate the Hit Ratio Curve for a working set [14]. Mattson et al. constructed a histogram of *reuse distances* of all the blocks accessed in a given workload to accurately calculate the HRC. A reuse distance of a block is the number of unique block accesses between two consecutive references to that block. Once such a histogram is created with all the reuse distances, a CDF of the histogram would give us the hit ratio as a function of the reuse distance, which is the cache size. Tracking the reuse distances for all the blocks in a trace is computationally intensive and thus it is highly impractical to construct an HRC dynamically at runtime. Recently, a number of techniques have been proposed to optimize the runtime and space constraint of calculating HRCs. These techniques can potentially compute HRCs in sub-linear time with a constant space [15], [16], [17].

Once HRC curves are calculated for several VMs running together on the same host, they can be used to partition the cache among those VMs. Traditionally, system administrators analyze the HRCs and manually allocate and partition the host-side cache. However, this method is labor intensive, and prevents the cache from being dynamically managed as VM workloads change over time. Recently there has been work done for dynamic cache partitioning [9], [7], [5] however, they focus on managing a single shared cache layer and do not take advantage of the deepening storage hierarchy.

III. Motivation

The motivation for this work is two-fold. First, we show why we need a smart partitioning algorithm. Second, we show the benefits of a multi-layer cache.

A. The case for smart partitioning

The need for partitioning host side caches has been studied in [9], [7], [5]. To further illustrate the need, we conduct a simple experiment where two VMs residing in the same host share a caching device that is managed with a global Least Recently Used (LRU) eviction policy, i.e., blocks are stored and evicted from a single cache in Least Recently Used order, with no attention paid to the VM making the disk request.

The left side of figure 2 shows a VM running a Terminal Server benchmark all by itself on a host, and the right side shows the same Terminal Server benchmark running along with a Terasort Hadoop benchmark. Terasort is an I/O intensive job with minimal cache locality. As figure 2 shows, one Terasort is able to significantly lower the hit rate of the

	DRAM	Flash	HDD
Read(ns)	10-50	25,000	5-8*10^6
Write(ns)	10-50	200,000	5-8*10^6
Cost ($/GB)[1]	8-12	1-3	0.05-0.10

TABLE I
COMPARISON OF DATA STORAGE TECHNOLOGIES [20]

Terminal Server from 82% down to 30%, but achieves an essentially 0% hit rate for itself.

The takeaway from this experiment is that I/O hungry jobs can easily congest the cache space and thereby obstruct other workloads that could potentially benefit more from the cache. While it is possible for system administrators to carefully specify which VMs can use a cache to avoid this kind of contention, that is time consuming and difficult to do efficiently if workloads change over time. An ideal solution would be a cache management system that "learns" the behavior of interfering workloads and prevents them from hurting other workloads that have better cache locality.

B. The case for multi-layer cache

An ideal cache provides fast random-read and random-write accesses. Table I shows the characteristic and pricing of several of such storage technologies. There are several other devices such as high performance PCI-express SSDs, and the arrival new faster non-volatile storage devices referred to as the Storage Class Memories (SCM) that constitute NVRAM technologies should also serve as excellent caching devices [18], [19].

However, even with the storage media available today, there is not a single technology that will give the maximum performance with minimal cost. This is because the nature of many storage workloads is that a small set of blocks is typically accessed far more frequently than the rest of the working set, which is in turn accessed more frequently than the remaining blocks. Thus placing the hottest data in a very fast storage device can provide a substantial performance benefit, but putting all data in such a device is too expensive. However, poorly managing these resources can lead to poor performance if the wrong data is kept in the wrong type of device.

To overcome these challenges, we propose MultiCache, a system that optimizes the usage of cache space among different workloads, and transparently scales to use multiple cache devices. MultiCache accounts for the relative speeds of different devices and short/long term behavior of VM workloads. This system allows system administrators to easily combine multiple cache devices and automatically allocate them to diverse workloads.

IV. MULTI-CACHE PARTITIONING

Multi-Cache seeks to partition M cache layers among N different virtual machines in order to derive the greatest overall performance subject to the priorities of different virtual machines. Calculating these partitions in an accurate and scalable way requires new techniques for characterizing the workloads of VM I/O patterns and calculating partitions. Assigning cache space across multiple devices to multiple VMs reduces to the NP-hard knapsack problem, so new heuristic approaches are needed to efficiently find accurate solutions.

[1]Values averaged among various vendors and may vary

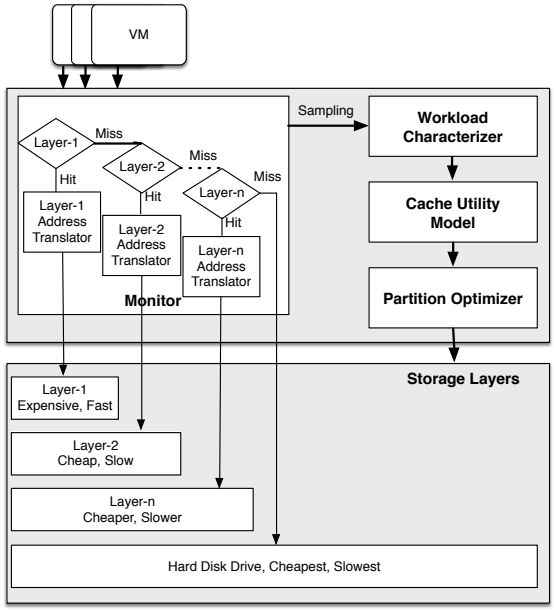


Fig. 3. Multi-Cache System Architecture

Figure 3 represents our system architecture. Our approach consists of three major components–a *Workload Characterizer*, a *Cache Utility Model*, and a *Partition Optimizer*. The Workload Characterizer maintains information about the I/O access pattern of each VM. To capture long-term I/O patterns that affect cache locality we track the reuse distance (RD) for each VM. However, traditional hit rate estimation mechanisms using RD fail to consider the relative access speeds different VMs access the cache, and they can also be disrupted by short-term bursts in I/O which often often come from interactive VMs. To capture these sudden bursts of disk activity, we use Reuse Intensity (RI) which tracks the frequency of unique and reused blocks over shorter time intervals.

The rest of the section explains how we characterize our workload, use long-term behavior and short-term behavior and various other parameters such as priorities of the workload and latency of the target devices to build a Cache Utility Model, and partition the different storage layers.

A. Long Term Behavior

The performance of a cache for a given workload depends on the frequency at which its data blocks are being reused over time. The measure of the frequency at which the data blocks for a workload are being reused is given by Reuse Distance (RD). Thus, RD is an accurate representation of the cache hit rate for a given cache size. We have two goals using RD.

The first is the representation of the workload access history in terms of data locality- the shorter the reuse distances a workload has, better the data locality. Second, we want to calculate this quickly in real time. To achieve this we use the open source implementation of PARDA [21].

A histogram of all RD values gives the distribution of data locality. This distribution is then used to compute *Hit Ratio Curves* (HRC). HRC as a function of cache size gives an estimate of the cache hit rate for a given cache size. We calculate RD values and HRCs in 1 hour intervals. For every 1 hour we calculate RD independently of the previous interval, i.e. we discard the values we obtained from the previous intervals.

A longer interval between RD calculation means it takes longer to obtain HRCs and partition the caches. The interval needs to be short enough to accurately predict the hit rate of the workload for the next interval, but not too long for the prediction to realize the effects. We found in our experiments that usually 1 hour of the trace time was a good value, but this is a tunable parameter.

RD captures the long term behavior of the workload, but cannot respond to short term bursts in I/O accesses. Quickly responding and resizing the cache to accommodate sudden bursts of I/O accesses is a key part of a cache management system. We use *Reuse Intensity* to capture the short-term behavior of the workloads.

B. Short Term Bursts

Different VMs will access the cache at different rates and capturing the relative speed of their accesses is important. Traditional HRC estimation mechanisms using RD fail to consider the relative access speeds with which different VMs access the cache. For instance, for certain web services, such as stock market providers, capturing quick bursts in trend is vital. To capture such small term workload spikes we use *Reuse Intensity* (RI) [7]. We use RI as a measure to capture sudden bursts of hit ratio in a cache.

$$RI = 1 - \frac{\text{\# unique blocks}}{\text{\# total blocks}} \quad (1)$$

Equation 1 represents the formula to calculate RI. It indicates that more the workload is random, the closer the value of RI will be to 0. For instance, if there are only 9 unique blocks from a workload of 10 block requests, the value of RI will be 0.1. On the other hand, if the workload has a lot of repeated requests to the same block, the value of RI will be close to 1. The value of RI is calculated once every 60 seconds, and can also be tuned as a user parameter.

Figure 4 shows an example of RI of 4 different web workloads taken from the MSR Cambridge traces [22]. One can see from the figure that some workloads have spikes such as web_0 where RI will be most useful, and some like web_1 where RI might provide useful information at some instances but not always. Thus for workloads that are very volatile, where hit rates cannot be effectively predicted using RD, RI is used extensively. Finally, for workloads such as web_2 or web_3 RI is close to zero, indicating there are no bursts of localized data accesses for these traces.

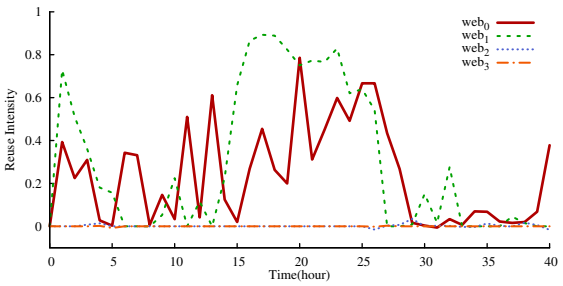


Fig. 4. Reuse intensities of different web workloads

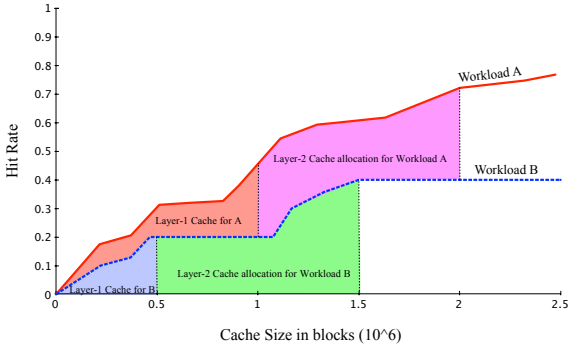


Fig. 5. Split HRC between different Layers according to size

Now that we have the long term behavior captured for every hour, and short term behavior captured for every minute, we use these two values in conjunction with latency of the caching device to build a cache utility model.

C. Cache Utility Model

Our Cache Utility Model predicts the benefit a VM can get from the cache. For simplicity, our exposition only considers a two layer caching model, and we will refer to the faster Layer-1 as *NVM*, and the slower Layer-2 as *SSD*. This model can easily be scaled to n layers.

MultiCache uses all of the layers dedicated to a VM as a single LRU cache—i.e., blocks evicted from a fast cache are demoted to a slower layer, and blocks accessed from the slow cache are promoted to the top of the faster cache. As a result, MultiCache can use the observed RD values to predict the cache hit rate for different *total* cache sizes, as well as for individual tiers. For example, Figure 5 shows two workloads being assigned different amounts of cache space at each level; for Workload A, the predicted overall hit rate will be 78%, of which 45% will come from the 1 million blocks allocated to it from the faster cache. Similarly, of the 40% predicted overall hit rate of Workload B, 20% will come from the 0.5 million blocks allocated from the faster cache. The goal of MultiCache is to determine these split points in order to maximize cache utility.

We use $h_i(c_i)$ to indicate the predicted hit rate for cache layer i with capacity c_i for a given workload. Since we use NVM as our first layer cache, it's expected hit rate is simply:

$$h_{nvm}(c_{nvm}) = RD(c_{nvm})$$

Where RD is the percentage of RD values less than c_{nvm} (the size of the NVM cache). Once a size, c_{nvm} is selected for the NVM layer (as described below), the hit rate on the SSD layer can be modeled. For the second and subsequent layers, we need to adjust for the size and hit rate of the previous caches since we only want to predict the hit rate seen in the space used at this layer. For our second layer SSD cache, this gives:

$$h_{ssd}(c_{ssd}) = RD(c_{ssd} + c_{nvm}) - h_{nvm}(c_{nvm})$$

These hit rates are the first terms in our cache utility function and represent the expected hit rate over the long term. We multiply the hit ratio of each layer by the latency of the caching device to the back-end storage device. We use l_b, l_{nvm} and l_{ssd} as the latencies of the backup device, NVM and SSD respectively. This causes cache layers with faster access speeds to provide greater cache utility for an equivalent size cache. We add to this the RI value calculated for each workload, adjusted by a term α:

$$CU_{nvm}(c_{nvm}) = \frac{l_b}{l_{nvm}} \times h_{nvm}(c_{nvm}) + (\alpha_{nvm} \times RI)$$

$$CU_{ssd}(c_{ssd}) = \frac{l_b}{l_{ssd}} \times h_{ssd}(c_{ssd}) + (\alpha_{ssd} \times RI)$$

Here α_{ssd} and α_{nvm} represents the expected "benefit" a particular application/workload can get from the RI values and is calculated for each layer as follows using the total capacity of each layer:

$$\alpha_{ssd} = \frac{cap_{nvm}}{cap_{ssd} + cap_{nvm}}$$

$$\alpha_{nvm} = 1 - \alpha_{ssd}$$

Thus if the layer 1 NVM cache has a relatively smaller size, it will cause alpha to become larger. As a result, RI will have more impact on the cache utility of that layer. This is desirable because RI is used by MultiCache to handle workload bursts, which are likely to cause frequent accesses in the faster, but smaller cache layers. If the first layer cache is very small, then it makes sense to place greater weight on RI instead of RD, because the cache will be too small to cache accesses for the long term behavior of RD, and instead its effectiveness will mainly be dominated by whether short local bursts can be served.

D. Optimization Solver

For each workload (i.e., VM), MultiCache determines a cache utility function for each cache storage layer. The goal of MultiCache is then to determine how to set the cache sizes in order to maximize the sum of utility functions. For the two-layer cache case and n different VMs, this becomes:

$$maximize: \sum_{v=1}^{n} p^v * (CU^v_{nvm}(c^v_{nvm}) + CU^v_{ssd}(c^v_{ssd}))$$

subject to:

$$\sum_{v=1}^{n} c^v_{nvm} = cap_{nvm} \; and \; \sum_{v=1}^{n} c^v_{ssd} = cap_{ssd}$$

This maximization problem includes a priority for each VM, p^v, which can be tuned by administrators to give extra cache weight to specific VMs. Since optimizing for multiple VMs using multiple caching layers is an NP-Hard problem, we use a heuristic algorithm for our optimization. MultiCache uses simulated annealing, and further simplifies the problem by greedily selecting sizes for each layer iteratively. That means that simulated annealing is first used to maximize the $CU^v_{nvm}(c^v_{nvm})$ term across all VMs, and then the selected cache sizes are used to calculate the best $CU^v_{ssd}(c^v_{ssd})$ for all VMs. This substantially reduces the search space compared to attempting to solve the maximization problem across all VMs and all layers simultaneously.

Putting it all together, in our experiments we reallocate cache sizes every hour but this is a tunable parameter. In our setup, the Workload Characterizer continually collects statistics, and for every hour we run our Cache Utility Model, and resize the partition using Partition Optimizer that uses the results from Optimization Solver.

V. Evaluation

A. Simulation Platform & Workload Analysis

To evaluate MultiCache's partitioning algorithms we have implemented a Python-based cache simulator. Our simulator accepts a trace of disk requests and simulates how the requests will be stored into the different cache layers. Similar to prior work [7], we ignore write requests and focus on read performance. We compare our MultiCache partitioning algorithm against Global LRU–a simple LRU policy that treats all layers of the cache as a single partition shared by all disks.

In our work, we calculate the Reuse Distance values using the open source implementation of PARDA [21]. PARDA uses a hash table that maps a LBA to its most recent reference, and a splay tree that holds the values of the number of distinct locations referenced since this LBA's most recent reference. PARDA is extremely efficient as its design philosophy resembles a map-reduce [23] approach. It computes the RD in parallel by partitioning the trace into different sections, processing them independently, and finally merging them back together.

We chose PARDA over other existing techniques [15], [16] because, *a)* Since we are running a simulation, we didn't need the online cache optimization decisions that are provided by Counter Stack and SHARDS. *b)* PARDA provides accurate Hit Ratio Curves as opposed to approximating them, and *c)* PARDA's implementation is open source, readily available to implement, and could be easily integrated within our system.

PARDA calculates the HRC curves for both read and write requests. Since, we are interested only in read requests here, we pre-processed the input trace to discard all write requests.

We use the block I/O traces from MSR Cambridge, hosted by SNIA [22]. It includes the following workloads:

Server	Function	#Disk
usr	User home directories	3
proj	Project directories	5
prn	Print server	2
hm	Hardware monitoring	2
rsrch	Research projects	3
prxy	Firewall/web proxy	2
src1	Source control	3
src2	Source control	3
stg	Web staging	2
ts	Terminal server	1
web	Web/SQL server	4
mds	Media server	2
wdev	Test web server	4

In our simulator we consider each volume of a particular Server as an individual virtual machine, i.e. in the case of *web*, there will be 4 VMs issuing requests using the traces *web_0*, *web_1*, *web_2* and *web_3* respectively. This means that these 4 VMs will be sharing/competing for space among different cache layers—in our case two layers, NVM and SSD.

We run our cache partitioning algorithm for every 1 hour of the trace time. We measure the hit rate, estimated latency, cache space and several other metrics for each interval.

B. MultiCache Performance

In this section, we show how MultiCache can effectively identify the workloads with higher locality and allocate more space for them in the fastest cache layer available. We use our simulator to compare the hit rate achieved with MultiCache against the Global LRU policy. We test each workload trace individually, having the different disks within that trace compete for cache space. We use a first layer cache that can hold $2.5 * 10^5$ blocks and a second layer cache that is ten times larger.

Figure 6 represents the average hit rate of each disk under each policy; we do not show workloads where the hit rate is less than 1%. One can see from the figure that Multi-Cache shows an equivalent or greater hit rate for all the workloads. This is because in global LRU, as explained in section III, the workloads that could benefit from having a bigger cache are restricted from using it due to the neighboring workloads that have random access patterns but higher request rates. In

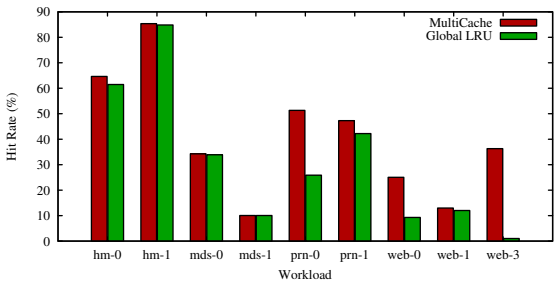


Fig. 6. Hit ratio of all the workloads

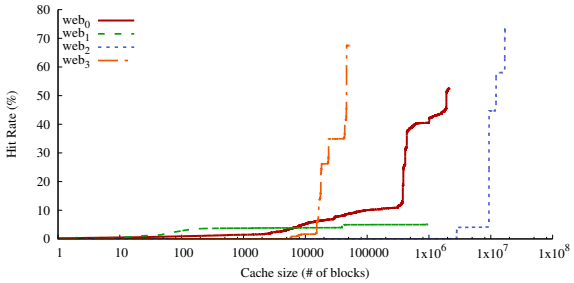


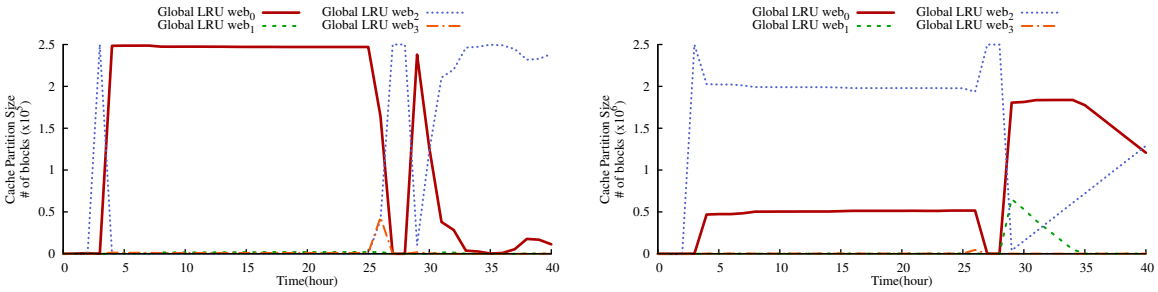
Fig. 7. MSR SNIA web workload

contrast, MultiCache recognizes the workloads that can benefit from more cache and gives them more space and restricts the VMs with random workloads from over utilizing the cache space. Across all of the workloads, Mult-Cache is able to improve the total hit rate by 31% percent.

For the following experiments we focus on MSR's *web* workload, mainly because the 4 volumes in the workload have very different access patterns. We replay each of the volumes' trace from a individual virtual machine. For instance the trace *web_0* will be replayed as if the requests are coming from VM0, and *web_1* from VM1 and so on. Figure 7 shows the hit ratio curves for each of these workloads. Note that *web_1* can never achieve a hit rate above 5%, even with a very large cache, whereas *web_0* and *web_3* are able to achieve reasonably high hit rates with relatively small cache sizes. In contrast, *web_2* sees essentially no hits unless there is an extremely large cache size (note the x-axis is log scale). This suggests that the workload has very poor temporal locality, but that the entire disk contents are read through multiple times over the trace, allowing the hit rate to be large if the cache is big enough. In practice, such a workload is not cacheable– the space dedicated to caching is typically significantly smaller than the size of the backend disks. In our experiments we only consider cache sizes in the range of 10^6 blocks, so *web_2* is not dedicated much cache space.

C. Partitioning Dynamic Workloads

This section illustrates the importance of a dynamic partitioning policy, where cache sizes are automatically adjusted as workloads enter different phases. The comparison of Globral LRU and MultiCache over time are detailed in this section. For clarity purposes, we show only first few hours of the



(a) NVM Layer (b) SSD Layer

Fig. 8. Cache allocation using Global LRU on NVM and SSD caching layers

experiments in our figures; similar behavior is seen in the remainder of the trace.

Figure 8 shows the cache allocation over time for different VMs of the web workload using Global LRU policy. Global LRU policy allocates cache space based on each VM's I/O request rate i.e. the higher the I/O request rate of a VM the more cache space will be allocated, and vice-versa regardless of whether it would benefit from the cache. Furthermore, Global LRU does not explicitly distinguish between NVM or SSD caching layers. As a result, this will aim to fill up the NVM layer's cache space first, and use the SSD if need be.

The figures shows that the first caching layer is used primarily for *web_0*, with occasional spikes where *web_2* crowds it out. The second layer is used primarily for *web_2*, despite that workload having very poor locality for cache sizes below 10 million blocks, as was shown in Figure 7. The other disks receive almost no cache space, despite having greater locality.

Figure 9 shows the cache allocation over time for different VMs of the web workload using MultiCache. Multicache tries to smartly allocate cache space such that the overall hit rate of all the VMs is maximized. Comparing figure 8(a) and figure 9(a), one can see that Multicache tries to give fair space to each of the VMs, but gives priority to VMs on NVM layer that exhibit high hit rate.

Unlike Global LRU, MultiCache gives minimal cache space to *web_2*, since it predicts it will not have a high hit rate. Instead, MultiCache dedicates most space to *web_0* and *web_1*. The choice of *web_1* is at first surprising, since Figure 7's hit ratio curve predicted a relatively low hit rate potential. The explanation is that the HRC is generated from the entire 1-week trace, during most of which *web_1* has quite poor locality. However, for the first 30 hours of the trace, the workload is quite different, and is capable of achieving a hit rate as high as 80%. This is shown in Figure 10, where *web_1* is able to achieve a very high hit rate for the first portion of the trace; this drops after hour 30 since a workload shift reduces the locality. As a consequence, MultiCache reduces the partition size for *web_1* for the remaining hours of the experiment. This illustrates the importance of a dynamic cache partitioning model that responds to workload changes.

A second workload of interest in figures 8 and 9 is the web_3 VM. web_3 has very low disk reads but they have high locality. This kind of workload is similar to real world interactive VMs, that often have low request rates but high data locality. Global LRU completely neglects this type of workload in the presence of competing high request workloads, but MultiCache smartly allocates sufficient space for web_3 in NVM layer for faster access and the rest in SSD. The latency improvements of these allocations will be discussed in the later sections.

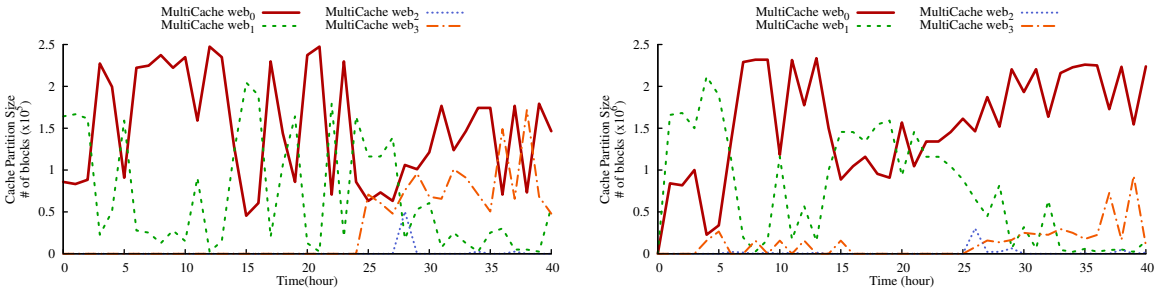
The effect of cache space allocations on hit rate can be seen in figure 10. Initially when the cache is empty, both methods perform the same. Once both the cache layers are warmed up with requests, we can see that the hit rate of the VMs using Global LRU levels up at the bottom while MultiCache reserves more space for VMs that can have higher hit rate and thus maximizes the overall hit rate. The hit rate of the VMs in Global LRU stays at the bottom for two reasons. One, Global LRU allocates almost all of its cache space in the NVM layer to web_0 and web_2 uses up all of the SSD's space with its high number of cold requests i.e. requests with no data locality thus cannot benefit from cache. MultiCache realized this about web_2 and smartly doesn't any cache space at all.

D. Cache partitioning performance

We next examine how using a multi-layer cache can improve performance. We use the disk latency data from table I so our simulator can report estimated response times for each read request depending on whether it is a cache miss or a hit in a particular layer. We use the latency data from the DRAM column for our Layer-1 cache in anticipation of upcoming non-volatile memory technologies that are expected to have response times in a similar range [24], and use Flash for the Layer-2 cache.

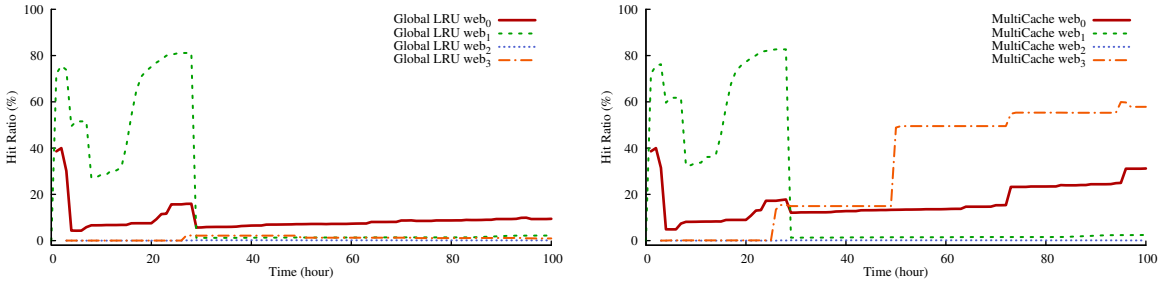
Figure 11 shows that MultiCache provides a significant latency reduction. We can see that the latency of the web_3, the high locality workload, out competed in the Global case, is the lowest for MultiCache since its small working set can easily be kept in the faster Layer-1 cache. The web_0 workload also sees a significant latency improvement. It is important to note that none of the workloads see worse performance in MultiCache, even though they are allocated less space with that policy than they can achieve with Global LRU.

The benefits of MultiCache can be explained by examining the average cache partition size for each of the workloads



(a) NVM Layer (b) SSD Layer

Fig. 9. Cache allocation using MultiCache on NVM and SSD caching layers



(a) GLobal LRU (b) MultiCache

Fig. 10. Hit rate of VMs for web workload using Global LRU and MultiCache

on the two layers. Figure 12 shows what percentage of each cache tier is allocated to each workload over the course of the 1 week trace. This shows how Global LRU ends up dedicating a large portion of both caches to web_2, which doesn't benefit. On the other hand, MultiCache uses about half of each layer for web_0, but also reserves sufficient space in the faster layer to satisfy most of web_3's requests.

This performance can be further exemplified by the hit rates of the workloads on the individual layers. Figure 13 divides the achieved hits up by cache layer. For MultiCache, web_0 about 10% of the hits come from NVM and 20% of the hits come from SSD layer. Here, one can see that the latency of web_3 is very low because all the hits of web_3 comes from NVM layer.

Figure 12 and figure 13 together represent the benefit of our partitioning algorithm. For instance, allocating 55% of the NVM layers cache on average yields about 10% hit rate on web_0 while allocating about 25% of the same cache space yields about 50% of hit rate on web_3. Intuitively one might think that more cache space must be allocated to web_3, but MultiCache predicts that web_3 cannot gain anymore from extra cache space, but other workloads might be able to do so. This is the justification behind allocating more cache space to a web_0 instead of web_3.

E. Total Cache Size

In this section we evaluate the effect of total cache size on the hit ratio of different VMs running together. In the following experiment we firstly show that MultiCache gives a higher hit rate using a cache of any size. Secondly, we show that Global

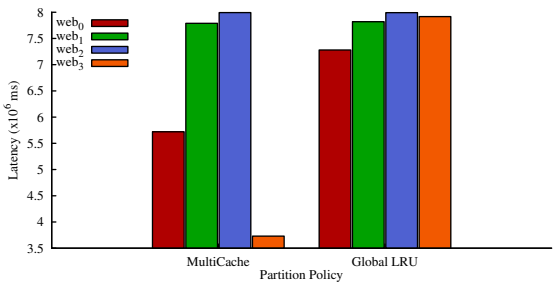


Fig. 11. Latency comparison of web workload between Global LRU and Multicache

LRU plateaus after a certain cache size when there is still room for improvement.

In this experiment we run 6 VMs *web_0* . . . *web_3*, *prn_0* and *prn_1* to create a greater level of contention on the cache, and compare the overall hit rate under each cache management policy. Figure 14 shows the average overall hit rate versus different cache sizes; the x-axis shows the Layer-2 cache size, which is always ten times larger than Layer-1. The average overall hit rate is the total accesses of all the VM workloads in the host divided their total hit rate. For instance if web_0 and web_1 has a total number of hits as 5000 and 5 with accesses 10000 and 5000 respectively, then we report the total hit rate as 33.3% ($\frac{5005}{15000} * 100$).

The figure shows that MultiCache always yields a higher hit rate than Global LRU except for very small cache sizes of less than 1.5 million blocks where both methods yield similar

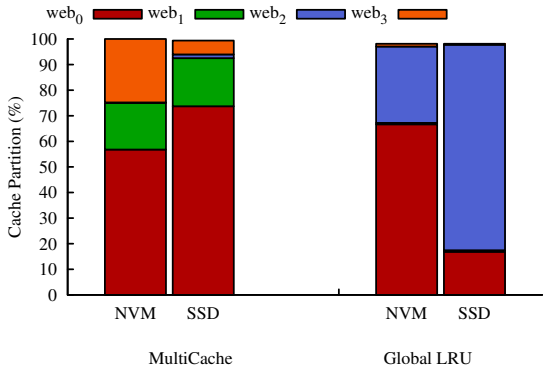


Fig. 12. Cache Partition for web workload

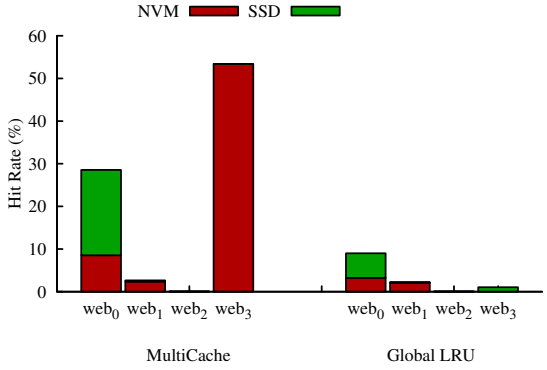


Fig. 13. Hit rate for web workload

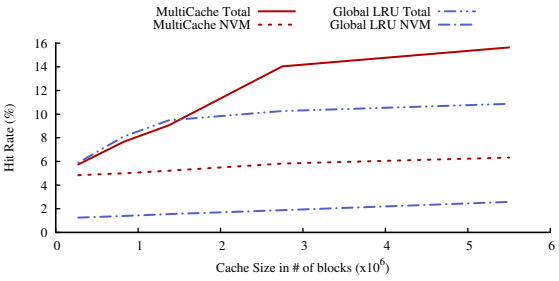


Fig. 14. Total Hit rate across all VMs of the web workload

results. This is because as the cache size grows small and the number of VMs increase, partitioning doesn't help for significant improvement unless there are multiple workloads with high random I/O. In that case, MultiCache will smartly prevent the random workloads from using the cache space. With large cache sizes, MultiCache improves the overall hit rate by 31% and the NVM hit rate by 89%.

VI. Related Work

RD calculation is computationally intensive. A naive implementation of Mattson's algorithm [10] takes $\mathcal{O}(m)$ space and $\mathcal{O}(n.m)$ runtime where n is the size of the workload, and m is the number of unique blocks requests in the workload. Standard implementations of the Mattson's algorithm use a combination of balanced tree to locate the previous reference of a block, and a hash table for quick lookups into the tree. This method still uses $\mathcal{O}(n\log(m))$ time, but the space still remained to be $\mathcal{O}(m)$.

Recently there have been works [21], [15], [16] that calculate RD in sublinear time with constant space. Wires et al. [16], in his CounterStack algorithm, reduced the space complexity down to $\mathcal{O}(\log(m))$ by using Hyperloglog [25] data structure and by various other techniques such as down sampling and pruning. Waldspurger et al. [15] on the other hand, in his Shards algorithm, uses a hash based spatial sampling technique to calculate approximate Hit Ratio Curves using just $\mathcal{O}(1)$ space.

The closest work to ours, from which we extend upon are by Meng et al. [7] and Luo et al. [26]. Meng et. al.'s system *vCacheShare* and Luo et al.'s *S-Cave* dynamically allocates cache-space at runtime for virtualized environments. *S-Cave* optimizes the cache based on identifying cache-friendly and unfriendly workloads while *vCacheShare* monitors the changes in data locality optimizing for long term cache benefits and short term bursts in data reuse. We extend their findings and present a case for the need of a multi-tier cache and show through our simulation how a two tier can improve the throughput and latency.

Studies have been done to evaluate the feasibility of replacing the disks with flash storage [2], and improving its resource utilization with minimal cost [27], but our work focuses on sharing the flash cache amongst different VMs based on their workload. Intel's Turbo Memory [3] takes a similar approach and uses a flash device as an extension of main memory.

Multiple storage devices used in conjunction to provide a flash based caching solution have been studied. Many of these consider the usage of Flash devices along with backup devices to provide a host based caching solution, but our work primarily varies with the following work in the sense that we can use multiple flash devices together for caching and can guarantee maximum utilization/performance benefits from all of them.

Multi tiered flash based storage systems have been recently studied [28], [29], [30]. Wang et. al. [28] proposed a allocation model that identifies bottlenecked workloads per-device in a hybrid storage system. In this model, clients that are bottlenecked on the same storage device receive throughputs in proportion to their fair shares while allocation ratios among clients in different bottleneck sets are chosen to maximize overall system utilization.

Hystor [30] combines SSDs and HDDs and provisions it as a single block device. It separates the performance critical blocks and redirects the I/O requests for those blocks to the SSD and the rest to HDD, thus improving performance for the critcal data blocks. Similarly, Combo Drive [31] also abstracts SSD and HDD as a single device, and internally redirects the I/O to each of the different devices. Fusion Drive [32] is a comercial implementaion of this model. Differentiated Storage Services [33] classifies the block I/O request from the user-

level based on system policies and matches blocks with storage devices.

VII. Conclusion

The deepening storage hierarchy, composed of non-volatile memory, flash, and traditional hard drives offers new opportunities in how virtual machine data can be cached on end-hosts. We have presented MultiCache, a multi-layer host-side cache that intelligently partitions a set of storage devices to cache data for competing VM workloads. MultiCache gathers workload data and characterizes the short and long-term behavior of each VM to build an optimization framework that maximizes cache utility. We have presented how MultiCache can improve cache hit rates for a wide range of workloads by predicting the locality of each VM's accesses and sizing each layer of the cache accordingly.

Acknowledgements: This work was supported in part by NSF grant CNS-1253575.

References

[1] "Configuration Maximums - vSphere 6.0 - VMware," https://www.vmware.com/pdf/vsphere6/r60/vsphere-60-configuration-maximums.pdf.

[2] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 145–158.

[3] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud, "Intel&Reg; Turbo Memory: Nonvolatile Disk Caches in the Storage Hierarchy of Mainstream Computer Systems," *Trans. Storage*, vol. 4, no. 2, pp. 4:1–4:24, May 2008.

[4] A. Leventhal, "Flash Storage Memory," *Commun. ACM*, vol. 51, no. 7, pp. 47–51, Jul. 2008.

[5] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "Mercury: Host-side flash caching for the data center," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, Apr. 2012, pp. 1–12.

[6] D. A. Holland, E. Angelino, G. Wald, and M. I. Seltzer, "Flash Caching on the Storage Client," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 127–138.

[7] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, ser. ATC '14. Philadelphia, PA: USENIX Association, Jun. 2014, pp. 133–144.

[8] D. Lee, C. Min, and Y. I. Eom, "Effective flash-based SSD caching for high performance home cloud server," *Consumer Electronics, IEEE Transactions on*, vol. 61, no. 2, pp. 215–221, 2015.

[9] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-side SSD Caching for Storage Performance Control," in *Proc. of ICAC*, 2015.

[10] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, Jun. 1970.

[11] C. Chow, "On optimization of storage hierarchies," *IBM Journal of Research and Development*, vol. 18, no. 3, pp. 194–203, 1974.

[12] W. D. Strecker, "Cache memories for pdp-11 family computers," *ACM SIGARCH Computer Architecture News*, vol. 4, no. 4, pp. 155–158, 1976.

[13] A. J. Smith, "Disk Cache-Miss Ratio Analysis and Design Considerations," *ACM Trans. Comput. Syst.*, vol. 3, no. 3, pp. 161–203, Aug. 1985.

[14] P. J. Denning, "The working set model for program behavior," *Communications of the ACM*, vol. 11, no. 5, pp. 323–333, 1968.

[15] C. A. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, "Efficient MRC Construction with SHARDS," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, Feb. 2015, pp. 95–110.

[16] J. Wires, S. Ingram, Z. Drudi, N. J. A. Harvey, and A. Warfield, "Characterizing Storage Workloads with Counter Stacks," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 335–349.

[17] T. Saemundsson, H. Bjornsson, G. Chockler, and Y. Vigfusson, "Dynamic performance profiling of cloud caches," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014.

[18] M. Nanavati, M. Schwarzkopf, J. Wires, and A. Warfield, "Non-volatile storage," *Commun. ACM*, vol. 59, no. 1, pp. 56–63, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2814342

[19] S. Pelley, T. F. Wenisch, B. T. Gold, and B. Bridge, "Storage management in the nvram era," *Proc. VLDB Endow.*, vol. 7, no. 2, pp. 121–132, Oct. 2013. [Online]. Available: http://dx.doi.org/10.14778/2732228.2732231

[20] "Beyond DRAM and Flash," http://www8.hp.com/hpnext/posts/beyond-dram-and-flash-part-2-new-memory-technology-data-deluge.

[21] Q. Niu, J. Dinan, Q. Lu, and P. Sadayappan, "PARDA: A Fast Parallel Reuse Distance Analysis Algorithm," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 1284–1294.

[22] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-loading: Practical Power Management for Enterprise Storage," *Trans. Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008.

[23] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[24] A. M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11.

[25] P. Flajolet, . Fusy, O. Gandouet, and e. al, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *IN AOFA 07: PROCEEDINGS OF THE 2007 INTERNATIONAL CONFERENCE ON ANALYSIS OF ALGORITHMS*, 2007.

[26] T. Luo, S. Ma, R. Lee, X. Zhang, D. Liu, and L. Zhou, "S-CAVE: Effective SSD Caching to Improve Virtual Machine Storage Performance," in *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 103–112.

[27] J. Tai, D. Liu, Z. Yang, X. Zhu, J. Lo, and N. Mi, "Improving Flash Resource Utilization at Minimal Management Cost in Virtualized Flash-based Storage Systems," *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[28] H. Wang and P. Varman, "Balancing Fairness and Efficiency in Tiered Storage Systems with Bottleneck-aware Allocation," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, ser. FAST'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 229–242.

[29] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost Effective Storage Using Extent Based Dynamic Tiering," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, ser. FAST'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 20–20.

[30] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 22–32.

[31] M. A. A. Sanvido, Z. Z. Bandic, and C. M. Kirsch, "Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device," in *In the 1st Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, 2009.

[32] Apple, "Fusion Drive," https://en.wikipedia.org/wiki/Fusion_Drive, Oct. 2012.

[33] M. P. Mesnier and J. B. Akers, "Differentiated Storage Services," *SIGOPS Oper. Syst. Rev.*, vol. 45, no. 1, pp. 45–53, Feb. 2011.