

Quantifying and Improving I/O Predictability in Virtualized Systems

Cheng Li, Íñigo Goiri, Abhishek Bhattacharjee, Ricardo Bianchini, Thu D. Nguyen
Department of Computer Science, Rutgers University, Piscataway, NJ
{chengli, goiri, abhib, ricardob, tdnguyen}@cs.rutgers.edu

Abstract—Virtualization enables the consolidation of virtual machines (VMs) to increase the utilization of physical servers in Infrastructure-as-a-Service (IaaS) cloud providers. However, our experience shows that storage I/O performance varies wildly in the face of consolidation. Since many users may desire consistent performance, we argue that IaaS providers should offer a class of predictable-performance service in addition to existing (predictability-oblivious) services. Thus, we propose VirtualFence, a storage system that provides predictable VM performance. VirtualFence uses three main techniques: (1) non-work-conserving time-division I/O scheduling, (2) a small solid-state (SSD) cache in front of a much larger hard disk drive (HDD), and (3) space-partitioning of both the SSD cache and the HDD. Our evaluation shows that VirtualFence improves predictability significantly, while allowing cloud providers to reach any desired compromise between predictability and performance.

I. INTRODUCTION

With the advent of cloud computing, virtualization has become the primary strategy to consolidate diverse workloads (encapsulated in virtual machines or VMs) to ensure high utilization of physical machines (PMs). Many IaaS cloud providers, such as Amazon EC2 and Rackspace, use virtualization and consolidation in offering their services. However, as we demonstrate in this paper, VM performance may vary significantly in the face of consolidation. In fact, VM performance is essentially unpredictable, since the number of co-located VMs and their workloads may change each time the VM runs, or even during a single run. For example, researchers have shown a single run of a fixed-load VM on Amazon EC2 to exhibit wild performance swings due to consolidation [1]. Since many users may desire consistent performance, we argue that IaaS providers should offer a new class of predictable-performance service *in addition to* (and using different resources from) their existing (predictability-oblivious) services.

Along these lines, our research seeks to create virtualized systems that exhibit *performance predictability*. This property implies that the average throughput and response time experienced by each VM should be unaffected by any other VM executing on the same PM or the overall utilization of the PM.

Importantly, note that our notion of performance predictability differs from *performance isolation* [2], [3], [4], [5], [6], [7], [8]. The goal of isolation is to ensure that each co-located VM achieves at least a minimum desired level of performance. This is one of the goals of predictability. However, in performance isolation it is typically acceptable to dedicate more resources than this minimum, if those resources are available. In contrast, dedicating any available resources beyond a fixed amount will likely ruin predictability. One may see isolation as a less strict form of predictability.

In this paper, we address the specific case of *storage I/O performance predictability* (or simply *I/O predictability*).

Compared to processing, memory, or networking, storage I/O predictability is the most challenging to achieve primarily due to the mechanical limitations of hard disk drives (HDDs). In particular, high disk seek and rotational times make predictability difficult to achieve when requests from multiple VMs to the same HDD are interleaved. Interestingly, the problem becomes even worse when a solid-state drive (SSD) is used as the storage medium; SSD erasures initiated by a VM can interfere with operations from other VMs. To exacerbate the unpredictability problem further, the current work-conserving resource management policies of virtual machine monitors (VMMs) link a VM’s I/O resources to the number of co-located VMs, causing unpredictability as this number changes.

Based on these observations, we propose VirtualFence, a performance-predictable storage system. VirtualFence seeks to produce consistent performance within the range defined by the best and worst performance levels that each VM may experience in a predictability-oblivious service (i.e., in the absence of VirtualFence). Under VirtualFence, each VM runs as if it were alone on a fixed and tightly controlled partition of the resources. To achieve this goal, VirtualFence couples a small persistent SSD cache with a much larger HDD. It also implements a non-work-conserving I/O scheduling algorithm, partitioning time into a fixed number of relatively coarse-grained slots. Each I/O slot can only be given to one active VM, but a single active VM may receive multiple slots (the number of slots depends on how much the VM’s owner is willing to pay the cloud provider). I/O accesses from a VM are only serviced during the VM’s assigned slot(s). A static allocation of time slots while a VM is active ensures consistent resource allocation for predictable performance. The SSD cache and I/O time partitioning together minimize the impact of HDD head movement due to consolidation, whereas I/O time partitioning minimizes the impact of SSD block erasures. Finally, VirtualFence partitions both the SSD cache and the HDD, which is a non-work-conserving space allocation scheme that again ensures a consistent resource allocation.

Our evaluation quantifies I/O predictability using a new metric we call “performance deviation” (or simply “deviation”). Deviation quantifies the percentage I/O performance degradation suffered by a VM that is consolidated with other VMs compared to the I/O performance it achieves in isolation. Our evaluation demonstrates that VirtualFence improves I/O predictability (or, equivalently, that it reduces deviation) significantly, as long as we utilize all of its component techniques at the same time. In fact, we show that simply using an SSD as a cache of HDD data is *not* enough. More fundamentally, our evaluation illustrates the tradeoff between predictability and performance: the more we improve predictability, the worse average response time becomes. The challenge is finding the smallest response time that will produce enough predictability.

II. MOTIVATION

Performance unpredictability in the face of consolidation.

In a longer technical report [9], we present an extensive study of the impact of workload characteristics, the VMM architecture, the approach to virtualizing storage, and storage device characteristics on I/O predictability. The results demonstrate that deviation is endemic across all system configurations for both throughput and response time.

Many users desire predictability. Although most cloud users may not require predictable VM performance, many actually do. For example, streaming and gaming applications typically seek to achieve a consistent rate (e.g., displayed frame rate) rather than the highest performance, if that performance might introduce unpredictability (jitter). There are also many cases where repeatable behavior is important, such as performance tuning, debugging, and diagnosis. In fact, it is impossible to evaluate the impact of changes to an application in the cloud, if its performance may constantly be affected by consolidation. Finally, many applications implement workflows/pipelines, where the performance that can be achieved in each stage depends on the expected performance of a previous stage. Properly designing such applications for the cloud is impossible if the performance of each stage can vary widely due to consolidation.

Predictability would benefit cloud providers and users.

As predictability is important to many users, we argue that IaaS cloud providers should offer a new class of predictable-performance service that uses its own (tightly managed) hardware resources. Current IaaS providers already offer a range of other classes of service, such as the Cluster Compute and Cluster GPU service classes of Amazon EC2.

The tight management of resources would: (1) enable the provider to charge for exactly the pre-defined levels of performance and predictability that its users require; (2) enable the provider to conserve energy when resources are not used to guarantee the performance paid for by its users. (3) create an obvious relationship between the resources that customers pay for and the performance that can be achieved with those resources, i.e. users never complain that the performance of their VMs suddenly got worse (when the provider stopped dedicating more than the minimum set of contracted resources). Gulati *et al.* mention some of these same benefits to limiting maximum allocations [3].

Cloud users can also benefit from predictability because: (1) they can rely on it to implement applications for which predictability is more important than receiving as many resources as are available; (2) predictability can lower their cloud costs when the provider can save money by conserving energy or provisioning their data centers more tightly; and (3) they can predict their cloud costs into the future with the certainty that their VMs' performance will never be affected by changes in provider-side resource allocation.

Client-side throttling does not work. One might think that delays can always be added on the client side to achieve predictable behavior. However, this intuition is incorrect. As the client does not know how bad consolidated VM performance may get in the future, it cannot target a performance level that is guaranteed to be consistent.

III. BACKGROUND AND RELATED WORK

Disk drives and I/O interference. Many recent studies have established that I/O interference prevents VMs from achieving predictable performance [2], [3], [4], [5], [6], [8]. Previous efforts to address this problem have focused primarily on resource scheduling techniques, seeking to provide proportional allocation of I/O resources with strong isolation [10], [11]. Argon [11] shares common techniques with VirtualFence, such as space partitioning of caches (memory caches in the case of Argon) and time partitioning of I/O access time. However, our focus on predictability instead of isolation leads to fundamental differences, including non-work-conserving allocation policies and static configuration parameters.

Other works seek to provide proportional allocation while supporting latency-sensitive applications [12], [13]. mClock [3] provides weighted fair-share to cloud storage, but it does not consider the properties of storage devices and how they impact predictability. VirtualFence does and, hence, combines SSDs and HDDs.

Besides the differences described above, our work identifies predictability, a stricter form of isolation, as desirable, and is the first to study hybrid SSD/HDD systems in this context.

Hybrid SSDs and HDDs. Most research on SSDs has focused on either using them as HDD replacements [14], [15], or using SSDs as a caching layer [16], [17].

In comparison to these efforts, VirtualFence combines the advantages of SSDs (high performance) and HDDs (low cost) to promote a different goal, namely I/O predictability. In addition, VirtualFence minimizes the performance interference produced by SSD block erasures.

IV. MEASURING UNPREDICTABILITY

Since our notion of predictability means achieving the same VM performance in the presence of other VMs as in isolation, we measure performance deviation as the percentage performance degradation when a VM runs in the presence of other VMs compared to when it runs alone on the physical host. Specifically, let P_I be the (initial) performance of a VM when running alone, and P_D be the (degraded) performance of the VM when co-located with other VMs. Then, the amount of deviation is $|\frac{P_I - P_D}{P_I}| \times 100\%$. When multiple VMs executing the same workload are used in an experiment, we report the average deviation across the VMs.

As we show below, performance deviation is often different for throughput and response time. Thus, throughout the paper, we study deviation for both metrics.

V. VIRTUALFENCE

VirtualFence uses three techniques to reduce deviation between VMs whose virtual disks are stored on the same physical disk: (1) a non-work-conserving time-division I/O scheduling algorithm with coarse-grained time quanta, (2) a small persistent SSD cache in front of a much larger HDD, and (3) space partitioning of both the HDD and the SSD cache.

The non-work-conserving time-division I/O scheduling serves two purposes. First, it ensures that the resources allocated to a VM are (mostly) constant regardless of the number

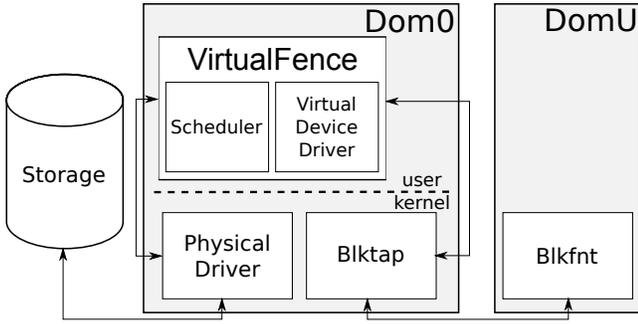


Fig. 1. VirtualFence architecture.

of co-located VMs. Second, it avoids the interleaving of requests from different VMs (inter-VM interference); for an HDD, this reduces seek overheads between operations from the same VM, whereas for an SSD, it reduces the interference of erasures from one VM on accesses from other VMs.

Despite the non-work-conserving policy, a system with only HDDs would still suffer some performance deviation when multiple VMs are co-located; as the system switches from serving one VM to another, the HDD’s head will have to move across partitions, leading to higher seek time for the first HDD operation, and so performance deviation. We limit the impact of this deviation by putting the SSD cache in front of the HDD. With a reasonable hit ratio in the SSD cache, we may eliminate some of these expensive HDD accesses. Moreover, the SSD cache significantly increases the performance of the virtual disk, so that the expensive first HDD operation is amortized across many more operations.

Finally, the space partitioning of the HDD limits the seek overheads between operations from the same VM, while the space partitioning of the SSD cache ensures constant cache space allocation for each VM.

A. Prototype

We have implemented a prototype VirtualFence in the Xen VMM 4.0.1, using the blktap user-level toolkit [18]. As Figure 1 illustrates, the prototype includes a device driver and a scheduler. The driver instances—a separate instance of the device driver is used to service each virtual disk—and the scheduler each runs as a user-level process in Dom0.

The SSD cache holds two types of persistent data: (1) blocks cached from the HDD, and (2) metadata describing the state of each cache block (e.g., valid bit, HDD block number). The driver implements the data structures needed to support an LRU replacement policy in volatile memory, including an LRU list of blocks, a write list that points to dirty blocks that need to be written to the HDD and then evicted, and a free list. At start up, the driver scans the SSD for all metadata, and builds all the in-memory data structures. No “last usage times” are kept across system restarts.

The LRU maintenance is simple. A background thread attempts to maintain the size of the free list above a threshold size by evicting the oldest entries in the LRU list as needed. Dirty blocks to be evicted are moved to the write list while the writes to the HDD are outstanding. If the free list ever reaches a low watermark threshold, processing of incoming requests is halted until the free list grows above the low watermark. The

driver uses asynchronous I/O to read and write data from/to both the SSD and HDD.

B. Space Partitioning

VirtualFence uses a separate partition of an SSD as a cache for each virtual disk co-located on the same HDD. We have also implemented a variation that uses a single SSD partition as a shared cache across multiple virtual disks to quantify the impact of space partitioning on deviation.

C. Time Partitioning

Our I/O scheduler assumes that a physical SSD/HDD pair is used to service at most n simultaneous active virtual disks, and so divides access to the physical disks into n equal-sized time slots. When a VM starts running on a host, its virtual disk is allocated one or more I/O slots (depending on how much of the host’s I/O resources is assigned to that VM). The scheduler then round-robins between the slots, *leaving a slot idle* when it is unassigned or the assigned virtual disk does not have any I/O activities; utilizing these slots would break the non-work-conserving property of the scheduler. On the other hand, this property of the scheduler also impacts performance, as we discuss extensively in Section VII-B.

The driver translates each user I/O request into requests to the SSD and HDD, and adds each type of requests to the appropriate device I/O queue. Each virtual disk has a distinct set of queues that are serviced during the slots assigned to the VM. The scheduler informs a driver instance when its assigned slot is scheduled, at which time it is allowed to forward requests to the SSD and/or the HDD until the slot time expires. A driver can end its slot early (see below), in which case the scheduler will lengthen the slot time appropriately the next time that slot is scheduled. If a driver overruns the slot time, the scheduler will deduct the overrun from the next slot.

To implement accurate time partitioning without losing performance, we need to send as many accesses as possible in a slot without running over the time allocated to the slot. In addition, it is more efficient to batch requests because of effects such as disk scheduling and fixed access overheads. Thus, our approach is to estimate the service times of batches of accesses, and to send the largest batch of accesses that is estimated to complete within the remaining time in the slot.

Our driver dispatches requests to the SSD and HDD in the same manner as follows. If there are no pending requests, then wait until a request arrives or the current slot terminates. If there are pending accesses, and at least the first access is estimated to complete within the remaining time in the slot, find the largest batch that is estimated to fit within the remaining time. After the completion of a batch of requests, if time remains in the slot, then repeat.

The driver ends a slot early if the first pending HDD request is estimated to take longer than the remaining time in the slot. This is because HDD resources are more constrained than the SSD, and thus, when the remaining slot time cannot be used for accessing the HDD, it is better to “credit” the time to the next slot of the same VM instead of wasting it. On the other hand, if a batch of HDD requests overruns the slot, while waiting for the batch to complete, the driver slowly issues any pending

SSD requests. This avoids wasting this time, while not causing even more slot delay by having to wait for the completion of a large batch of SSD requests.

Predicting HDD request service times accurately can be quite complicated. For our purposes, however, it is sufficient to use a simple piece-wise linear function that predicts the access time of a request based on the distance between the block being requested and the block requested by the immediately preceding request. When predicting the service time of a batch, we order the requests using the block addresses under the assumption that the disk scheduling algorithm includes some form of scanning. We parameterize the prediction function for our specific HDD by benchmarking the service time of a large number of random batches of accesses, each batch with a random mix of reads and writes. We use a similar approach for predicting SSD service times. We discuss our service time prediction approaches (and their accuracies) further in [9].

VI. EXPERIMENTAL METHODOLOGY

A. Workloads

We use workloads from Filebench [19], a popular framework for measuring and comparing file system performance. Specifically, we use Fileserver, Mailserver, and Webserver. Fileserver emulates a server hosting directories owned by multiple users; Mailserver focuses on mail operations and has an I/O mix of a read per sync write; and Webserver emulates a server that services a read-only workload.

We configure workloads of four VMs running concurrently, each of which executes the same Filebench application. One of the VMs is configured to produce a low-intensity I/O load that is approximately 8% of the storage system’s saturation load. Each remaining VM is configured to produce approximately 24% of the storage system’s saturation load. Overall, the four VMs reach 80% of saturation, representing an aggressive consolidation scenario. We then compare the low-intensity VM’s performance to when it runs alone, and the performance of each of the three higher-intensity VM to when it runs alone. Note that we scale the load to maintain a constant utilization level (80%) across storage systems (HDD, SSD, and VirtualFence). We call this setup a 4-VM heterogeneous workload and use it as the primary workload for our study. In Section VII, we also study homogeneous workloads and systems with low-intensity VMs only.

B. Experimental Platform

We use a server equipped with a 2.4GHz 4-core Xeon CPU (each core supports two hardware threads), 8GB of RAM, a 60GB SSD, and a 160GB 7200RPM HDD. According to its datasheet, the HDD has an average seek time of 11ms and full stroke time of 22ms. The SSD is spec’ed with random read performance $>20,000\text{op/s}$ and random write performance $>5,000\text{op/s}$. We measured erasures, including garbage collection, to take approximately 3.5ms-4ms in a write-only benchmark. The guest OS in the VMs is always a Debian installation with Linux kernel version 2.6.32.

We choose the Noop disk scheduler in the guest OS to isolate the impact of the VMM’s I/O scheduling. We choose CFQ for the host OS because it minimizes deviation when not using VirtualFence.

Variant	HDD	SSD	Cache	NWC
HDD+NWC	x			x
SSD+NWC		x		x
Hybrid/Shared	x	x	Share	
Hybrid/Shared+NWC	x	x	Share	x
Hybrid/Partitioned	x	x	Partition	
VirtualFence	x	x	Partition	x

TABLE I. VARIANTS OF VIRTUALFENCE COMPRISING DIFFERENT COMBINATIONS OF PREDICTABILITY-ENHANCING TECHNIQUES. THE CACHE COLUMN SHOWS WHETHER THE SSD CACHE IS SHARED OR PARTITIONED. THE NWC COLUMN SHOWS WHETHER NON-WORK-CONSERVING TIME PARTITIONING IS USED.

In all experiments, we allocate 512MB of memory to each VM and pin it to a core to minimize the impact of VMM CPU scheduling. We run at most 4 VMs simultaneously so that each VM can be allocated an entire core.

VII. EVALUATION

We now explore VirtualFence’s effectiveness in providing performance predictability. The SSD cache block size is set to 4KB to match the default 4KB block size of the HDD. We also adjust the SSD cache size to explore the impact of different hit rates. We use the notation $\text{VirtualFence}(X\%, Y\text{ms})$ to denote a VirtualFence system with a time-sharing slot size of $Y\text{ms}$, and the SSD cache empirically sized to achieve a hit rate of $X\%$. We explicitly set the SSD cache hit rate to systematically isolate its impact; in practice, administrators would set the SSD partition size (and the number of time slots) for each VirtualFence virtual disk based on the QoS/resources promised to the disk’s owner and the number of virtual disks to be consolidated on the physical server.

To isolate the contributions of the different features of VirtualFence toward increasing predictability, we also measure deviation for many incomplete variants of VirtualFence. Table I lists these variants. The first two variants, HDD+NWC and SSD+NWC, are designed to isolate the benefits of non-work-conserving time partitioning. The Hybrid/Shared variant uses an SSD cache in front of the HDD, but the entire cache space is shared between multiple virtual disks. Hybrid/Shared+NWC extends this variant with non-work-conserving time partitioning. Hybrid/Partitioned is VirtualFence without non-work-conserving time partitioning, isolating the benefits of space-partitioned SSD caches.

A. Performance Deviation

VirtualFence. We begin by showing VirtualFence’s effectiveness at reducing performance deviation. Figure 2(a) shows the measured deviation when running the 4-VM heterogeneous workloads on $\text{VirtualFence}(50\%, 20\text{ms})$. Figure 2(b) shows the measured deviation when running the 4-VM heterogeneous Fileserver workload, which experiences the highest deviation, on VirtualFence with hit rates ranging from 50% to 100% and a time-sharing slot size of 20ms. Both figures show the deviations of the low I/O VM and the average deviations of the high I/O VMs in the heterogeneous workloads.

Figure 2(a) shows that VirtualFence is successful at reducing deviation in both throughput and response time, compared to a system without VirtualFence. In fact, VirtualFence produces lower deviations regardless of storage device or approach

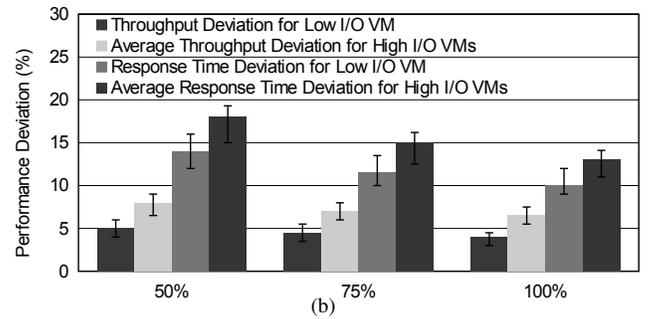
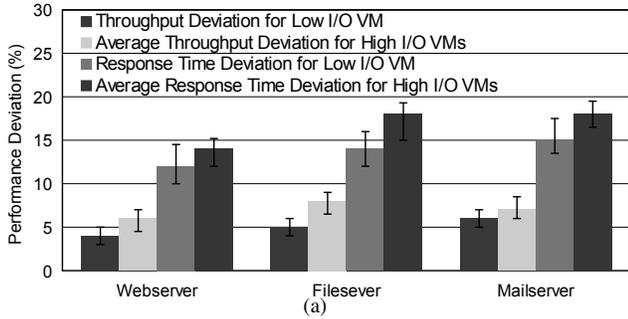


Fig. 2. Deviation when running (a) the 4-VM heterogeneous workloads on VirtualFence(50%,20ms), and (b) the 4-VM heterogeneous Fileserver workload on VirtualFence($X\%$,20ms), with $X \in \{50\%, 75\%, 100\%\}$. The range markers show the minimum and maximum values from three experiments whereas the bar shows the average.

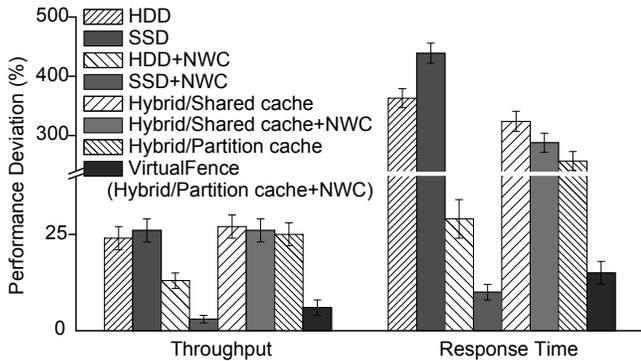


Fig. 3. Deviation when running on VirtualFence(50%,20ms) compared to various incomplete variants of it. Each bar shows results for the low-intensity VM in the 4-VM heterogeneous MailServer workload. The cache size for Shared-cache versions is equal to the sum of the caches in the Partitioned-cache cases.

to virtualizing storage. For the low-intensity VM, all deviations are $\leq 15\%$, compared to throughput deviations of $\geq 31\%$ and response time deviations of $\geq 443\%$ without VirtualFence [9]. Furthermore, deviations are always lower than 19% when the SSD cache affords a 50% hit rate. Figure 2(b) shows that deviation decreases as the SSD hit rate increases.

The raw performance of VirtualFence is also good. For example, Fileserver file accesses (97KB on average) by the low-intensity I/O VM take an average of 19ms, when the VM runs in isolation on the HDD configuration. When the same VM runs co-located with 3 high-intensity VMs, the average file access time increases to 98ms. We increase the I/O intensity of each VM by a factor of 3.3x in VirtualFence(50%,20ms) to achieve the same utilization as in the HDD case. Despite the much higher I/O intensity, the low-intensity VM experiences an average file access time of 59ms when running alone, and just 64ms when co-located with 3 high-intensity VMs, under VirtualFence(50%,20ms). A user who desires better raw performance may purchase multiple slots for its VMs.

Isolating the contributions of different features. Figure 3 plots performance deviation when the Mailserver workload is run on the variants (including the full VirtualFence implementation) listed in Table I. Performance deviations for HDD and SSD are also shown as baselines. These results are representative of all three Filebench workloads.

First, this figure shows that VirtualFence achieves performance predictability close to that of SSD+NWC. Specifically,

SSD+NWC achieves 3% and 10% throughput and response time deviation, respectively, whereas VirtualFence achieves 6% and 12%. SSD+NWC represents the best case scenario since it includes space partitioning (each VM is given a separate SSD partition), non-work-conserving scheduling, and storage completely on the SSD. The fact that these two systems achieve almost the same predictability shows that our caching approach is effective, allowing VirtualFence to extend the predictability benefits of (expensive) SSDs to much larger (and cheaper per byte) HDDs with small SSD caches.

Second, results for HDD+NWC suggest that non-work-conserving time partitioning can also be effective in reducing deviation when not using an SSD cache. However, the movement of the disk head between partitions when changing between time slots assigned to different VMs is sufficiently large that HDD+NWC with a 20ms slot size still incurs a 13% throughput deviation and a 33% response time deviation. As we show in Section VII-B2, increasing the slot size to attack this source of deviation also increases the response time observed by a VM running alone on a host. As already mentioned, this source of deviation also exists in VirtualFence but is mitigated by the SSD cache.

Third, at this hit rate, non-work-conserving time partitioning achieves higher predictability than using an SSD cache: HDD+NWC has lower deviations than both Hybrid/Shared and Hybrid/Partitioned. Interestingly, HDD+NWC is also better than Hybrid/Shared+NWC, implying that the interference at the shared cache negates some of the benefits of NWC. Of course, as the hit rate increases, the relative advantage of using NWC vs. an SSD cache will likely change.

Fourth, as expected, a shared SSD cache produces worse predictability than a partitioned cache. A shared cache can produce higher absolute performance; e.g., it may benefit an I/O-intensive VM running by itself. However, it would hurt predictability when the VM is co-located with other VMs and so must share the cache.

Finally, all three techniques used in VirtualFence contribute to increasing predictability; VirtualFence achieves higher predictability than the other configurations, except for the much more expensive SSD+NWC.

B. Performance vs. Predictability

In this section, we explore the impact of two key parameters: the number of VM slots, and the length of each slot.

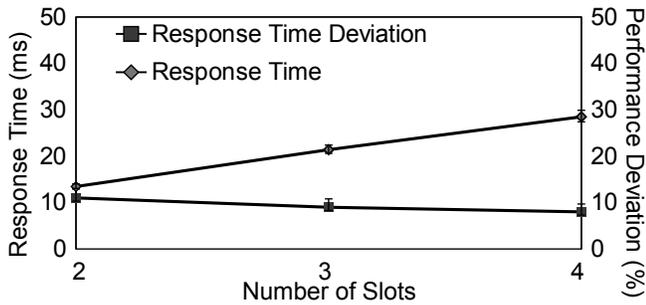


Fig. 4. Number of slots and response time trade-off.

1) *Impact of Number of Slots:* Figure 4 illustrates the impact of the number of slots on response time deviation and raw response time of VirtualFence(50%,20ms) for the Webserver workload (the other workloads show similar trends). For each number of slots n , we assume n VMs, and configure each VM to generate only $\sim 5\%$ of the VirtualFence(50%,20ms) saturation load. Such a low aggregate load is challenging for VirtualFence, raw performance-wise, because it may produce significant average waiting times.

As the figure shows, VirtualFence produces lower response times as we decrease the number of slots (and keep the slot length fixed). Fewer slots also mean lower waiting times, as each VM is allotted a higher fraction of time. Interestingly, response time deviation decreases slowly as we increase the number of slots. With a large number of slots, deviation would approach 0%, because the waiting time would overwhelm the single disk head movement in the first request of each slot.

Clearly, there is a tension between wanting a small number of slots to reduce average response times and wanting to increase the number of slots to improve predictability. Fortunately, VirtualFence produces reasonably good predictability even with only a few slots. Thus, the number of slots should be the smallest that will enable enough consolidation.

2) *Impact of Slot Length:* The key source of remaining deviation in VirtualFence is the need to move the disk head from one partition to another when changing slots assigned to different VMs. Thus, the slot length directly impacts VirtualFence’s predictability: a longer slot better amortizes the inter-partition head movement cost among more requests. However, lengthening the slots also increases response time.

Assuming 4 slots, Figure 5 plots the response time deviation and average response time, as a function of slot length for VirtualFence(50%,10-40ms) for the Webserver workload (the other workloads show similar trends). We use the 4-VM heterogeneous workload, and focus on the low-intensity I/O VM in Figure 5. This setup is challenging, predictability-wise, because it is almost certain that every first access in the low-intensity VM’s slot will cause a disk head movement.

The figure clearly shows the tradeoff between lowering deviation by lengthening the slots against increased response time. Lengthening the slots from 10ms to 20ms significantly reduces performance deviation. Further lengthening the slots to 40ms reduces deviation much more slowly at the expense of a further, essentially linear, increase in response time. Thus, a slot length of 20ms is the right tradeoff for our particular SSD and HDD devices. We have chosen this length as our a default for all previous experiments based on these results.

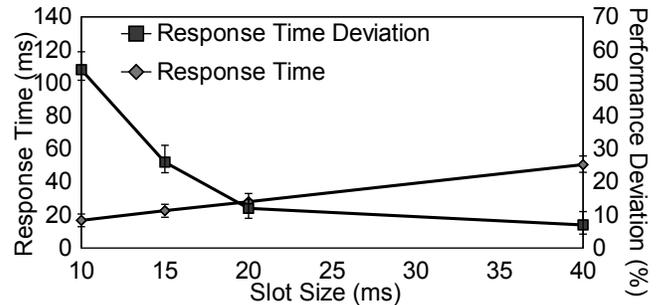


Fig. 5. Slot length and response time trade-off.

Again, there is a tension between wanting shorter slots for lower average response times and longer slots for better predictability. The slot length should be the shortest that will produce enough predictability.

VIII. CONCLUSIONS

We conclude that it is possible to build performance-predictable storage systems with simple software and hardware components, especially for those users that find predictability just as important as (or even more so than) raw performance.

Acknowledgments: This research was partially funded by NSF grant CNS-0916878.

REFERENCES

- [1] D. Novaković *et al.*, “DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments,” *USENIX ATC*, 2013.
- [2] P. Barham *et al.*, “Xen and the Art of Virtualization,” *SOSP*, 2003.
- [3] A. Gulati *et al.*, “mClock: Handling Throughput Variability for Hypervisor IO Scheduling,” *OSDI*, 2010.
- [4] D. Gupta *et al.*, “Enforcing Performance Isolation Across Virtual Machines in Xen,” *Middleware*, 2006.
- [5] H. Kim *et al.*, “Task-Aware Virtual Machine Scheduling for I/O Performance,” *VEE*, 2009.
- [6] Y. Koh *et al.*, “An Analysis of Performance Interference Effects in Virtual Environments,” *ISPASS*, 2007.
- [7] R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds,” *EuroSys*, 2010.
- [8] J. Wang, P. Varman, and C. Xie, “Avoiding Performance Fluctuation in Cloud Storage,” *HIPC*, 2010.
- [9] C. Li *et al.*, “Quantifying and Improving I/O Predictability in Virtualized Systems,” Rutgers, Tech. Rep. DCS-TR-697, 2012.
- [10] W. Jin, J. Chase, and J. Kauer, “Interposed Proportional Sharing for Storage Service Utility,” *SIGMETRICS*, 2004.
- [11] M. Wachs *et al.*, “Argon: Performance insulation for shared storage servers,” *FAST*, 2007.
- [12] A. Povzner *et al.*, “Efficient Guaranteed Disk Request Scheduling with Fahrrad,” *EUROSYS*, 2008.
- [13] J. Zhang *et al.*, “Storage Performance Virtualization Via Throughput and Latency Control,” *ACM TOS*, 2006.
- [14] A. Birrell *et al.*, “A Design for High-Performance Flash Disks,” *SIGOPS OPER SYST REV.*, 2007.
- [15] W. Josephson *et al.*, “DFS: A File System for Virtualized Flash Storage,” *FAST*, 2010.
- [16] T. Kgil, D. Roberts, and T. Mudge, “Improving NAND Flash Based Disk Caches,” *ISCA*, 2008.
- [17] S. Lee *et al.*, “A Case for Flash Memory SSD in Enterprise Database Applications,” *SIGMOD*, 2008.
- [18] D. Meyer, “Virtual Disk Backend Driver for Xen,” <http://wiki.xensource.com/xenwiki/blktap2>.
- [19] Filebench, “Filebench,” <http://sourceforge.net/projects/filebench>.