Pesto: Online Storage Performance Management in Virtualized Datacenters

Ajay Gulati
VMware Inc.
agulati@vmware.com

Ganesha Shanmuganathan
VMware Inc.
sganesh@vmware.com

Irfan Ahmad
VMware Inc.
irfan@vmware.com

Carl Waldspurger
carl@waldspurger.org

Mustafa Uysal
VMware Inc.
muysal@vmware.com

ABSTRACT

Virtualized datacenters strive to reduce costs through workload consolidation. Workloads exhibit a diverse set of IO behaviors and varying IO load that makes it difficult to estimate the IO performance on shared storage. As a result, system administrators often resort to gross overprovisioning or static partitioning of storage to meet application demands. In this paper, we introduce *Pesto*, a unified storage performance management system for heterogeneous virtualized datacenters. Pesto is the first system that completely automates storage performance management for virtualized datacenters, providing IO load balancing with cost-benefit analysis, per-device congestion management, and initial placement of new workloads.

At its core, Pesto constructs and adapts approximate black-box performance models of storage devices automatically, leveraging our analysis linking device throughput and latency to outstanding IOs.Experimental results for a wide range of devices and configurations validate the accuracy of these models. We implemented Pesto in a commercial product and tested its performance on tens of devices, running hundreds of test cases over the past year. End-to-end experiments demonstrate that Pesto is efficient, adapts to changes quickly and can improve workload performance by up to 19%, achieving our objective of lowering storage management costs through automation.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; C.4 [**Performance of Systems**]: Measurement techniques; C.4 [**Performance of Systems**]: Performance attributes; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction*; D.4.8 [**Operating Systems**]: Performance—*Measurements*; D.4.8 [**Operating Systems**]: Performance—*Operational analysis*; D.4.8 [**Operating Systems**]: Performance—*Queuing theory*

General Terms

Algorithms, Design, Experimentation, Management, Measurement, Performance

Keywords

Storage, Virtualization, VM, QoS, Device, Modeling

SOCC'11, October 27–28, 2011, Cascais, Portugal.

1. INTRODUCTION

Over the past decade, many IT organizations deployed virtualized data centers in an effort to consolidate workloads, streamline management, reduce costs, and increase utilization. Despite success in these areas, overall costs for *storage management* remain high. Over its lifetime, managing storage is four times more expensive than its initial procurement [23]. The annualized total cost of storage for virtualized systems is often three times more than server hardware and seven times more than networking-related assets [28].

Virtualization offers unprecedented dynamic control over storage resources, allowing both VMs and their associated virtual disks to be placed dynamically and migrated seamlessly around the physical infrastructure. While the basic mechanisms for quickly migrating storage workloads exist [20, 34], making higher-level decisions regarding the mapping of virtual disks to physical storage devices is much harder. Virtualized environments can be extremely complex, with a diverse set of mixed workloads sharing a collection of heterogeneous storage devices. Such environments are also dynamic, as new devices, hardware upgrades, and other configuration changes are rolled out to expand capacity.

Simplistic approaches for dealing with performance problems, such as gross overprovisioning and strict partitioning of resources between applications, are common in many physical deployments. Such techniques are even less appealing for virtualized environments, where they are fundamentally at odds with the goal of reducing costs through workload consolidation and efficient device utilization. Storage administrators are still grappling with the consequences, making most storage decisions in an ad-hoc manner. Administrators typically rely on rules of thumb, or risky, time-consuming trial-and-error placements to perform workload admission, resource balancing, and congestion management.

The fundamental challenge in effective storage performance management is estimating IO performance accurately and robustly in a dynamic environment. Estimates must be updated to adapt to changing workload mixes, varying device conditions and configurations, and active storage components that attempt to optimize their own local performance. Relying on static parameters, extensive offline measurements, or frequent user intervention is impractical. Even if parameters could be tuned correctly at a given point in time, most environments are too complex to ensure that they would remain reasonable. Moreover, mistakes are costly, affecting many applications due to high consolidation ratios.

Existing approaches for storage provisioning and modeling, such as Hippodrome [5], Minerva [3], IRONModel [30], and relative-fitness models [24], mostly rely on offline measurements and are not designed to work in a completely online manner. Analytical approaches [22, 27, 32] rely on detailed device information which is not available in most real environments, and on configurations which may change over time. BASIL [13] supports online device modeling and IO load balancing, but it lacks cost-benefit analysis leading to poor decisions. We also found its passive online modeling impractical, frequently resulting in long convergence times. PARDA [11] manages congestion on a single device, but requires a key latency-threshold parameter to be set manually.

Toward automated storage management, we introduce *Pesto*, the first practical, automated system designed to manage storage performance in dynamic, heterogeneous virtualized environments. Pesto consists of three key components: 1) a continuously-updated storage performance model to estimate performance; 2) a decision engine that uses this model to place and migrate storage workloads as conditions change; and 3) a congestion management system that automatically detects overload conditions and reacts by throttling storage workloads. Our key contributions include:

- A simple yet useful analytical relationship between peak throughput and the slope of latency *vs.* outstanding IOs.
- The design and implementation of an online workload injector that creates device models in tens of seconds.
- Leveraging these device models to perform robust IO load balancing with cost-benefit analysis, initial placement of new virtual disks, capacity planning and threshold determination for congestion management.
- A fully-automated system that integrates these components to manage storage devices in a virtual datacenter.

We have implemented Pesto as part of a commercial product, within the *Storage DRS* component of VMware's vSphere management software [36]. We report results from end-to-end experiments demonstrating that Pesto computes effective device models and supports powerful automated storage management capabilities for virtual datacenters, including storage load balancing, congestion management, and capacity planning. Pesto-recommended placement improved overall throughput by more than 10% and reduced peak-load latency by up to 19%.

The next section discusses existing approaches for storage modeling and automated management. Section 3 provides a high-level overview of the Pesto system. Section 4 presents our new analytical results, and shows how they can be leveraged to construct practical performance models. The design and implementation of Pesto is described in Section 5, including online model creation and its applications to IO load balancing, congestion management, and capacity planning. Section 6 evaluates Pesto using end-to-end experiments in a datacenter environment. Finally, we summarize our conclusions and highlight directions for future research in Section 7.

2. BACKGROUND

The complexity of storage devices and workloads is reflected in the evolution of modeling techniques, which can be classified into two broad categories: *analytical models* and *sampling-based performance models*. Our technique falls in the empirical, sampling-based modeling category, with the key distinction that models are obtained in a lightweight online manner.

2.1 Analytical Models

Analytical models try to predict the performance of storage arrays using low-level details about their disk drives, cache sizes, and caching policies, along with workload characterizations. Analytical models are often much less expensive and faster to build than empirical models. However, they suffer from two key drawbacks: using simplifying assumptions about the underlying storage device and requiring detailed knowledge of storage array internals for better modeling [33].

The capabilities of analytical models have advanced along with storage technologies. For a single disk drive, Gotlieb and MacEwen [10], Ruemmler and Wilkes [26] and others have proposed detailed seek-distance and drive-performance models. Tools like DIXTrac [9] allow users to extract detailed disk characteristics by running specific workloads. Kim and Tantawi [17] extended models to consider multiple disks. Lee and Katz [18], and Chen and Towsley [6, 7] incorporated queuing, IO segmentation, and RAID architectures.

Later research [21, 40, 31] extended models to include various operating modes, such as normal, degraded, recovery and rebuild while others [8, 27, 32, 33] addressed complexities such as caches, read-ahead and prefetching policies. Some approaches try to understand internal caching and prefetching policies based on empirical measurements. This is mainly done by using offline access to the array, measuring latency and throughput while running specific workloads. For example, random and sequential reads can be used to understand prefetching policies in an array [33].

Our goal is to monitor the array from the outside and predict performance metrics (*e.g.*, expected average latency, peak device IOPS, proximity to peak performance) by utilizing a black-box approach. As such, we prefer simpler models that provide sufficiently high accuracy for automating storage management tasks such as load balancing, workload placement, and admission control.

2.2 Sampling-Based Models

Sampling-based methods build storage models by monitoring a device passively, or by performing measurements actively using micro-benchmarks. Such techniques have become quite popular in recent years, as they do not require understanding increasingly-complex disk array internals.

Based on measurements from actual storage devices, researchers have proposed statistical models [16], table-based models [4], CART [39] and other machine-learning models, and relative-fitness models [24]. Relative-fitness research found that workload performance changes based on the underlying device, making it harder to model closed workloads.

Thereska and Ganger [30] have proposed robust models

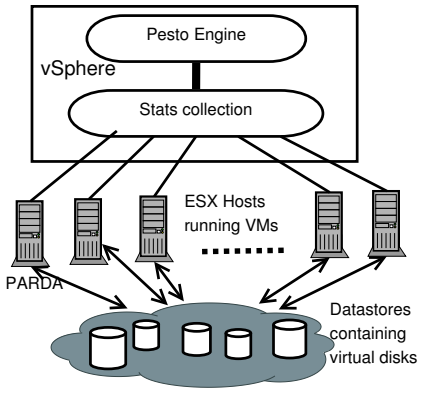


Figure 1: Pesto system architecture.

that can better deal with changes in a storage system over time, as well as modeling bugs. However, their model requires access to the storage system source code, which is not generally available. Other approaches for "what-if" analysis [29] involve very simplistic queuing theory and tend to work only on single-disk systems.

In our previous work, we proposed BASIL, a solution to IO load balancing in virtual environments [13]. BASIL used the linear relationship between IO latency and outstanding IOs as a relative model. However, we encountered several challenges while trying to use BASIL in real production environments. First, a BASIL model is based on passive observations of actual workload IOs, causing different models to be produced for the same device over time. Second, a robust model requires covering a wide range of outstanding IOs, which may not be observed in production deployments, even over long time periods. Finally, BASIL didn't perform any cost-benefit analysis, leading to workload configurations with lower throughput or higher latency.

3. PESTO SYSTEM OVERVIEW

In this section, we describe our system model and provide an overview of Pesto. Figure 1 depicts the Pesto system architecture. Each ESX host [35] runs multiple virtual machines (VMs) containing user workloads. A VM has one or more virtual disks, which are encapsulated by files. Each virtual disk file is placed on a *datastore*, corresponding to a LUN or a filesystem on a shared storage device. Datastores are accessed by all the ESX hosts using NFS or the VMFS clustered file system [37].

Pesto runs in a separate vSphere management server [36] and is responsible for deciding where to place each virtual disk. Pesto makes its decisions based on an online performance model for each datastore. It uses a workload injector that runs in one of the ESX hosts to automatically generate performance models while the system is deployed. These models are updated periodically to reflect any changes in operating conditions.

In order to find a good placement of virtual disks that balances IO load across the available datastores, Pesto collects detailed statistics on the way the virtual disks are accessed at each of the ESX hosts. The management server periodically collects these IO statistics and computes online histograms for various datastore and virtual disk statistics using the P^2 algorithm [15] . The Pesto decision engine uses these statistics and the performance models to recommend actions to balance IO load or to place new virtual disks. The recommendations produced by Pesto are executed on the ESX hosts by moving the virtual disks across datastores automatically, or by creating new virtual disks on the datastores selected by Pesto.

Each ESX host also runs PARDA [11], which performs congestion management on the individual datastores. Pesto uses its performance models to determine the operating parameters of PARDA automatically, such as the latency threshold used to identify congestion on a datastore. As more virtual disks are allocated and IO loads change over time, Pesto determines whether the IO loads are approaching the peak throughput capacity of the storage devices, and helps administrators plan capacity expansion for the storage system.

4. LQ-SLOPE PERFORMANCE MODEL

The performance of a storage device depends on many factors, such as the number of outstanding IOs (OIOs), IO size, read-write ratio, cache hit rate, seek distance among requests and other storage controller optimizations [3, 5, 13, 33]. However, the overall latency for an IO request is the sum of both its service time and its queuing delay. As the number of OIOs increases, queuing delay becomes the main component of overall latency – not the service time for a single IO request [16].

Our model is based on combining an empirical observation – that latency (L) varies linearly with the number of outstanding IOs (Q), as shown by several recent studies [11, 13, 16] – with the well-known Little's Law [19]. We refer to the slope of this linear relationship as the *LQ-slope*.

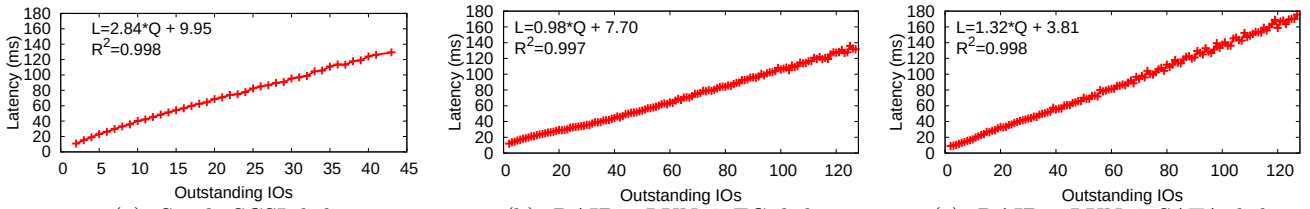
In this section, we first discuss the goals for our model in the context of virtualized environments. We then present our simple analytical result based on the relationship between average latency and the number of OIOs at the storage device. While this result may seem intuitive for simple queuing systems, it is not obvious for storage devices whose performance tends to depend on IO load.Finally, we explain the impact and handling of other workload parameters.

4.1 Modeling for Virtualized Datacenters

Pesto is designed for production use in enterprise environments. In typical deployments, virtualized datacenters are configured to reduce costs by improving efficiency. Multiple workloads are consolidated onto shared devices, driving up both average and peak device utilization. The properties and constraints of such environments are reflected in our design assumptions:

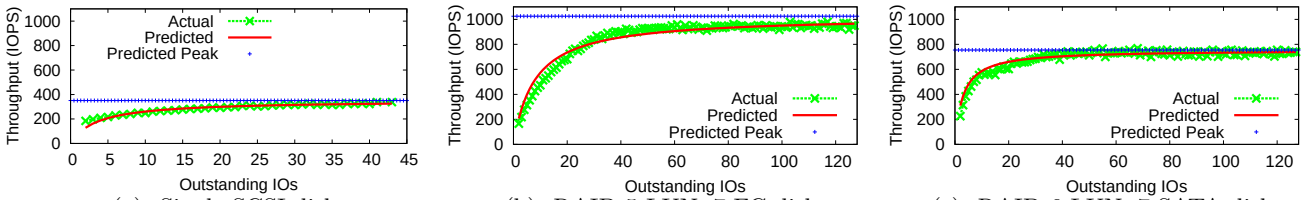
High Device Utilization: As a consequence of workload consolidation, our primary focus is on the behavior of devices during periods of relatively high OIOs, when queuing delay dominates latency. Unlike previous techniques, we do not try to model service times or IOPS for light workloads.

Typical Workloads: Our goal is to obtain device models in terms of peak IOPS and average latency for worst-case workloads during periods of high load. We do not aim to compute precise latencies for given workload configurations. As a result, random IO can effectively approximate aggregated enterprise workloads, despite ignoring the complexities of storage controller optimizations. BASIL [13] found that changes in IO size must be fairly large to affect latency substantially, and that the impact of randomness and read-write ratio is even less significant for typical enterprise workloads.



(a) Single SCSI disk (b) RAID-5 LUN, 7 FC disks (c) RAID-6 LUN, 7 SATA disks

Figure 2: LQ-plots of latency *vs.* outstanding IOs show linear relationships for three different LUNs.



(a) Single SCSI disk (b) RAID-5 LUN, 7 FC disks (c) RAID-6 LUN, 7 SATA disks

Figure 3: Plots of throughput (in IOPS) *vs.* outstanding IOs show non-linear relationships for three different LUNs. As the number of OIOs increases, the increase in throughput slows and ultimately saturates.

Although a few well-known applications, such as backups and database logging, do exhibit more extreme behavior, they are also less likely to suffer from performance bottlenecks.

Concurrency Largely Device-Independent: In many applications, the number of OIOs is an inherent property of the workload, with little device dependence [12, 13]. One study measured workload changes across devices, and found the overall change in OIOs to be within 10-15% [24]. Typical applications are designed to use a certain number of threads to issue IOs, and this number doesn't increase arbitrarily for a slower device. However, in the case of a faster device, the number of OIOs may be lower due to quicker completions. To reduce any error due to overestimation of concurrency, models can be adjusted incrementally based on actual behavior.

No Disruptions: Online modeling must not adversely impact production workloads. As a result, we prefer approximate models that can be obtained online, instead of more nuanced models that require offline or intrusive calibration. Nevertheless, our modeling must handle dynamic, shared and consolidated storage systems robustly.

4.2 Latency vs. OIO: Linear Relationship

Linearity between IO latency and outstanding IOs (OIO) is not very intuitive, given that the performance of storage devices is load-dependent and especially since caches and other optimizations can play a critical role in overall latency. Later, in this section, we explore this relationship in more detail and present a simple model to explain this behavior. For now, we demonstrate empirically that the relationship is linear and thus can be represented using Equation 1, where L and Q denote latency and OIOs, m is the slope of the linear fit line and C its intercept:

$$L = m \times Q + C \tag{1}$$

We validate this empirically using micro-benchmark experiments for three different storage devices with diverse disk types and reliability levels: a single SCSI disk, a RAID-5 LUN on a NetApp array consisting of 7 FC disks and a RAID-6 LUN consisting of 7 SATA disks on the same array. In all cases, we ran a closed-loop workload with 16 KB random reads. Figure 2 plots average IO latency as the number of OIOs is varied for the storage devices.

First, notice that latency increases almost linearly with Q. It is not obvious that latency should vary linearly, especially since device performance improves with OIOs due to improved IO scheduling. A second observation is that IOPS varies in a non-linear manner with Q and reaches saturation after a certain point; this is well-known behavior [11, 13]. Figure 3 plots the throughput for the same three devices and the same workload as Q is increased. The data shows a clear asymptotic throughput saturation point.

Assuming that the arrival rate is equal to the completion rate, and applying Little's Law [19], we know that:

$$Q = L \times T \tag{2}$$

Here T is the throughput (in IOPS) and L is again the average latency of IO requests (in seconds). Using Equations 1 and 2, throughput can be obtained directly as a function of OIOs by substituting for L:

$$T = Q/(m \times Q + C) \tag{3}$$

Equation 3 models the non-linear relationship between the total number of outstanding IOs, Q, and throughput, T. Figure 3 shows that the predicted throughput matches the empirical observations closely. Using Equation 3, we can estimate the peak value of throughput, T_{peak}, as Q is increased to a large value:

$$T_{peak} = \lim_{Q \to \infty} \frac{Q}{m \times Q + C} \approx 1/m \tag{4}$$

Equation 4 shows that peak throughput is simply a function of LQ-slope. Devices with lower slope have higher peak throughput, such that the peak value matches the inverse of the slope. Intuitively, this seems reasonable: more powerful devices will have relatively smaller increases in average la-

tency in response to increasing load, as well as higher peak throughput.

The key result is that the LQ-slope of a workload can predict the device throughput of that workload closely. This relationship is well-known for a simple queuing server with a fixed service time per request. Nevertheless, it is surprising that such a simple result holds for complex storage systems. By using both the LQ-slope and the intercept C, Equation 3 can be used to predict throughput as the number of OIOs is varied.This is extremely useful, since these parameters can be obtained easily in a live system, either by running a short-lived workload injector (discussed in Section 5.1), or by collecting data pairing the number of OIOs with observed latency using passive online measurements.

Alternatively, peak IOPS could be obtained by running a high-OIO workload and observing throughput directly, but this would have several drawbacks. First, the workload must saturate the device, negatively impacting other workloads competing for array-level resources. Second, we can estimate the entire IOPS *vs.* OIOs curve, and use that information for various purposes, as detailed in later sections. Finally, LQ-slope can be computed for a workload in an online manner, even though the workload may never generate enough load to saturate the device.

4.3 Why is Latency vs. OIO Linear?

Earlier we presented the linear relationship between latency and outstanding IOs as an empirical observation. We now revisit this issue and present a model to explain how this behavior is possible, despite the load-dependence property of storage devices.

From a queuing-theory perspective, the performance of storage devices varies with the number of IOs queued at them. For sufficiently large queue depths, service times do not decrease as more load is added, due to diminishing returns from IO scheduling. Thus, latency is dominated by queuing delay, making it linear in terms of queue depth.

This is consistent with a well-known result: average seek distance decreases with increasing OIO. For a single random IO, the average seek distance is one-third of the maximum seek distance [14]. As the number of OIOs increases, average seek distance decreases according to Equation 5 [10, 25]:

$$AvgSeekDistance = \frac{MaxSeekDistance}{Q+2} \quad (5)$$

Although average seek time decreases with increasing queue depth, there are diminishing returns. The overall service time can be expressed as a fixed latency L_{fixed} and a variable seek-time latency based on queue depth (Q):

$$L_{service} = L_{fixed} + \frac{MaxSeekTime}{Q+2} \quad (6)$$

For Q IOs (including the one in service), the total latency for the current request is $Q \times L_{service}$:

$$L_{total} = Q \times (L_{fixed} + \frac{MaxSeekTime}{Q+2}) \quad (7)$$

Thus, for disks, the overall response time approaches a linear function as Q increases, assuming worst-case service time with random IO requests.

We have verified this empirically for many different storage devices using closed-loop workloads while varying the number of OIOs. Figure 4 graphs LQ-plots for a wide variety of logical devices exposed out of storage arrays from different

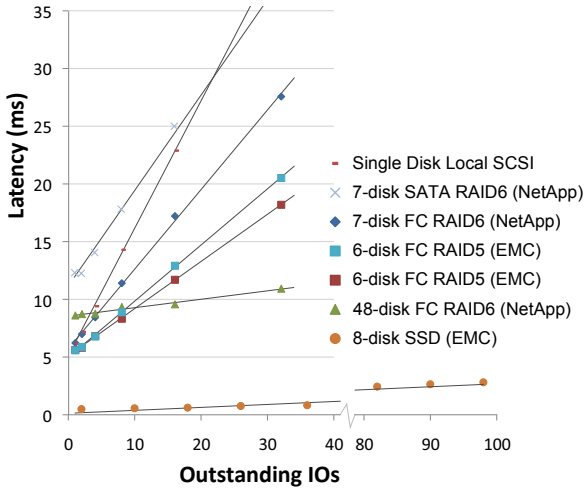


Figure 4: LQ-plots for several different devices.

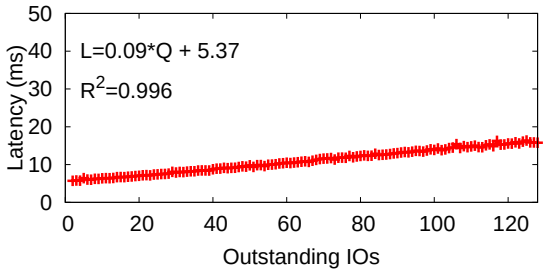


Figure 5: LQ-plot for 48-disk LUN on NetApp FAS.

vendors. These include LUNs backed by varying numbers of Fibre Channel, SATA, and solid-state disks (SSD). In each case, the LQ-plot is linear, with the LQ-slope varying based on device performance. For the SSD device, we continued out to 100 OIOs to show that linearity continues to hold.

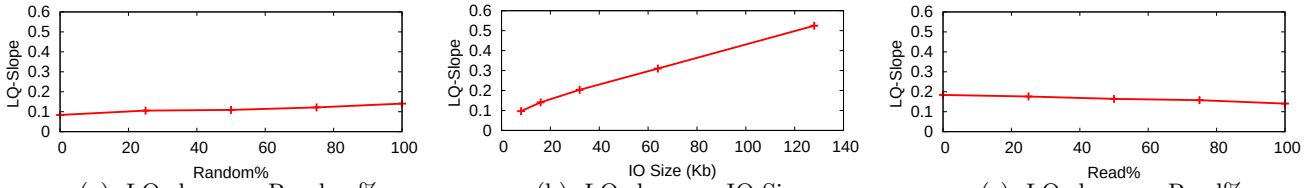
As the number of disks backing a datastore increases, one might reasonably expect more complex behavior that could not be captured by a single LQ-slope. For instance, random IOs may result in a skewed distribution of OIOs across disks. To validate our result for such cases, we ran an experiment using a 48-disk RAID-6 LUN on a NetApp FAS disk array. Figure 5 shows the slope for this LUN as Q was increased. We do not observe any specific abnormal behavior. Of course, the LQ-slope itself is much smaller due to higher parallelism.

4.4 Dependence on IO Workload

So far we have shown that for a given fixed workload, our model can predict throughput based on load with good accuracy. We now examine how LQ-slope, latency and throughput vary with respect to other workload parameters, such as IO size, read fraction, and request locality.

We ran a series of more than 700 micro-benchmarks, varying just one of these parameters along with OIOs while keeping others constant. We collected data for variable IO sizes from 4 KB to 256 KB, read percentage from 0 to 100%, and randomness from 0 to 100%. These benchmarks were run on 4 different LUNs: three with 3, 6 and 9 FC disks respectively and another with SSD disks.

Figure 6 shows how LQ-slope varies for different values



(a) LQ-slope *vs.* Random% (b) LQ-slope *vs.* IO Size (c) LQ-slope *vs.* Read%

Figure 6: LQ-Slope as a function of IO workload parameters, for a 9-disk FC LUN.

of these parameters for the 9-disk FC LUN; other LUNs displayed similar variations. For IO size, the slope varies from 0.09 to 0.52 as the size increases from 4 to 256 KB. The slope varied from 0.18 to 0.14 as the percentage of reads varied from 0 to 100%. Similarly, the slope varied from 0.08 to 0.14 randomness increased from 0 to 100%.

These results reveal a high variance in LQ-slope based on the actual workload, especially for extreme cases such as sequential workloads, workloads with large IO size, and workloads with a high percentage of writes. The data points corresponding to completely-sequential workloads exhibited a big decrease in slope due to prefetching. The time to service larger IOs increased with IO size, which caused the slope to increase. In LUNs with write-back caching, higher read fractions caused the LQ-slope to be higher, as caches are able to absorb writes.

As a result, one shouldn't use the slope of one workload to predict the performance of others when any of them exhibits extreme behavior. Except for extreme cases where the workloads have very large IO sizes, are completely sequential, or have a large percentage of writes, the variation in LQ-slope was limited to 15%. While a few well-known applications exhibit such extreme behavior, the workload observed by a datastore is typically random due to high degree of consolidation. Even if datastores are read sequentially during backups, virus scanning or other operations, their provisioning will typically be based on more common non-sequential access patterns.

We also measured the error in predicted throughput for the micro-benchmarks. Comparing the actual throughput at 64 OIOs, the error was less than 5% for all four LUNs, for all experiments except the extreme cases of sequential and write-intensive workloads discussed above. For the SSD LUN, the error was less than 5% in all cases. The highest error for the 9-disk LUN was for 128 KB sequential writes, at approximately 7%. The errors for the 3- and 6-disk LUNs were largest for sequential write workloads, near 30%.

For each of these cases, the relationship between the inverse of LQ-slope and peak throughput seems to hold. This shows that when workload characteristics are well known and not very variable, performance estimates can be based on the slope of the micro-benchmark closest to the real workload. For example, if large IO sizes are common in a certain environment, one can construct more accurate device models by using those IO sizes in the workload injector (discussed in Section 5.1)

In practice, most application workloads vary over time. For a more complex IO workload, we can either evaluate the slope based on its actual request mix, or by using a principal component of the workload. In most applications, we observe that these parameters don't change frequently, and

Workload	Size	%Read	%Random
WebServer1	4 KB	95	75
WebServer2	8 KB	95	75
DB Server	1 MB	100	100
DB Log	64 KB	0	0
Exchange	4 KB	67	100
Streaming	64 KB	98	0

Table 1: Iometer settings for simulated workloads.

that they are rarely associated with the extreme ranges. Extreme cases of completely-sequential or write-only workloads, such as log LUNs, are not difficult to identify. In the next section, we examine cases where there are multiple principal components, either within a single application, or due to multiple applications running on the same shared storage.

4.5 Aggregation of IO Workloads

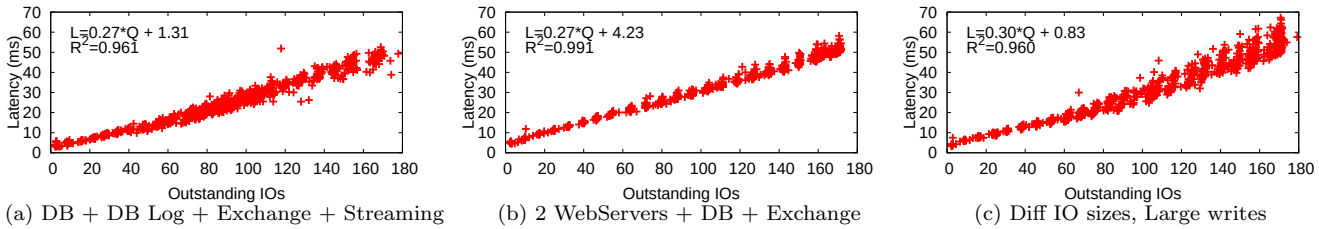
Sharing storage LUNs across multiple applications has become commonplace. For example, deployments of virtualized datacenters and cloud infrastructures require shared storage to facilitate VM migration and management. As a result, there are typically multiple IO streams accessing a single device. We now consider the relationship between throughput, latency and LQ-slope for a mixture of workloads.

We experimented with four VMs, each running a different workload selected from Table 1. The VMs were hosted on a Dell PowerEdge 2950 server running VMware ESX 4.1 [35]. These workloads represent Iometer [2] configurations that have been proposed to closely mimic some well-known real workloads [38]. We scaled the OIOs from the VMs linearly to measure the impact on latency and throughput.

Figure 7 plots the average IO latency for three different workload mixes. Note that latency still increases linearly with outstanding IOs. Figure 8 shows the corresponding throughput obtained as a function of OIOs. Notice again that throughput increases initially, but saturates after a certain load. For each workload mix, we also plot the predicted throughput curve based on our model, which tracks the actual observed throughput closely.

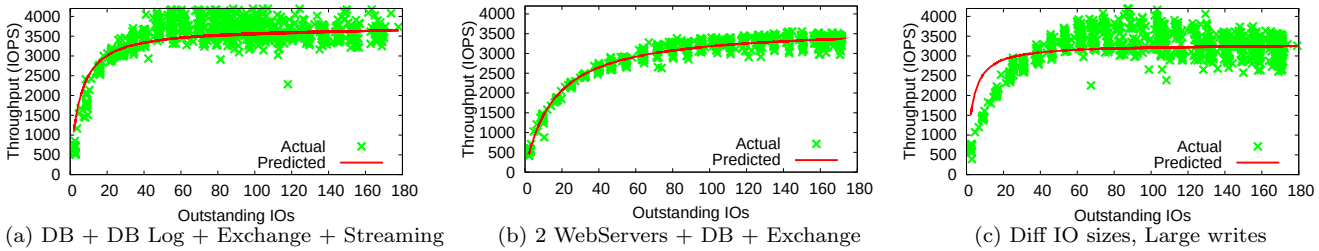
The predicted throughput was compared with the average throughput observed at each OIO. Even when the IO sizes were large and some VMs issued large numbers of writes, while other VMs issued sequential reads, the error in predicted throughput was within 10% of the average value after the throughput saturated, as shown in Figure 8(c).

These results indicate that the LQ-slope is representative of overall throughput, even if the actual workload is a mixture of multiple IO streams. We scaled the workloads uniformly but, in practice, the proportion of workloads may



(a) DB + DB Log + Exchange + Streaming (b) 2 WebServers + DB + Exchange (c) Diff IO sizes, Large writes

Figure 7: Latency *vs.* OIO for three different workload mixes.



(a) DB + DB Log + Exchange + Streaming (b) 2 WebServers + DB + Exchange (c) Diff IO sizes, Large writes

Figure 8: Throughput (in IOPS) *vs.* OIO relationship for three different workload mixes.

vary at different load levels. In such cases, we can use the configuration and slope for a higher range of OIOs.

We have experimented mostly with changes in workload characteristics, but not in array controller settings. One could also evaluate this relationship across different subsets of controller-level optimizations. However, by studying several diverse arrays from different vendors, we believe that we have covered a reasonably broad range of controller-level optimizations and settings.

5. DESIGN AND IMPLEMENTATION

The key to automating storage performance management is obtaining robust and accurate device models in an online manner. In this section, we first describe Pesto's online model creation component and its various challenges. We then explain how to use these models for various tasks, such as congestion management, IO load balancing, handling addition and removal of datastores and capacity planning.

5.1 Online Model Creation

Like Pesto, BASIL [13] also employed online models. However, we encountered several problems while trying to use BASIL in real production environments. First, a BASIL model is based on passive observations of actual workload IOs, causing different models to be produced for the same device over time. Second, a robust model requires covering a wide range of outstanding IOs, which may not be observed in production deployments, even over long time periods.

These shortcomings of BASIL are illustrated in Figure 9, which is composed of wildly-divergent LQ-slope models for the same datastore, depending upon which workloads were placed there while the model was being computed. Similarly, when faced with a new device, BASIL was forced to move some workloads to it before being able to form any model at all. In Pesto, we have attempted to remedy these inaccuracies and long modeling delays by taking a very different approach.

We developed a workload injector that runs for short durations during idle periods to generate performance models periodically. The workload injector generates an LQ-slope model that combines empirical observations of request latency (L) and the number of outstanding IOs (Q). To produce such a model, the injector must gather enough data points to correlate the size of the request queue to the observed latency under a variety of conditions. It needs to identify idle periods, and should detect if its measurements are being disrupted by any ongoing workloads. The models generated by the injector should be applicable to a wide variety of storage devices deployed in the datacenter.

To generate an LQ-slope model, the injector issues random read IOs (of configurable size; by default, 4 KB) over a large fraction of the datastore address space. We collect measurements ranging from 2 to 32 OIOs. For each OIO value, the injector issues IOs for a short period of time and collects the latency measurement for each IO, along with the total injected IO count. Once these measurements are complete, we use a standard least-squares based linear fit to compute the slope in the model. If the R^2 value is smaller than 0.93, we discard the values, assuming that the error may be due to interference or background activity at the storage array.

For block-based storage devices, the injector simply opens the raw device and issues random reads over the full or partial block range of the device. For NFS-based data storage, it isn't possible to open a raw device. Instead, the injector traverses the directory tree, selecting the largest files until it reaches a predetermined file count or total space threshold. It performs a random IO by generating a random offset into the total space consumed by these files, yielding an offset within a particular file to read. Alternatively, one could create a large file, issue random IOs, and delete the file.

By using random read IOs, we aim to measure the physical access characteristics directly. Random reads across the entire block address range ensure that we are not biased by read cache hits. Similarly, reads bypass any write buffers in

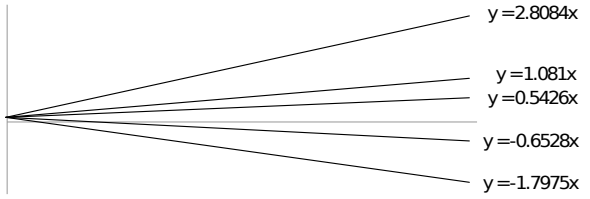


Figure 9: BASIL-generated LQ-slopes for the same NetApp 7-disk RAID-6 datastore, under different workloads. Pesto instead performs active modeling.

the storage device. We are exploring separate write models for storage devices and the impact of write caching as future work.

5.1.1 Idleness Detection

In virtual environments, multiple hosts often access the same storage device using either NFS or a clustered file system such as VMFS [37]. In both cases, a single host can determine idleness only for the VMs it is running locally. In order to detect device idleness globally, we collect statistics at the datastore level by aggregating them from all hosts periodically (by default, every four seconds).

Using a shared file on the datastore, each host is assigned a single 512-byte block to deposit its statistics. Since block writes are considered atomic, any host can read the file asynchronously, and compute the overall IO count and latency recently observed by other hosts, thereby computing datastore-wide values. To avoid conflicts, we execute the injector only from a single host by electing a leader using the shared file in the datastore: the host assigned to the first block in the shared file is the leader and tasked with the model generation for the datastore. Liveness is detected using a heartbeat mechanism: if a host dies or becomes disconnected, a new block assignment is generated and a new leader is elected.

To detect idleness, the injector monitors the overall statistics for recent periods. A datastore is deemed idle if its overall IO count and outstanding IOs remain below a specified threshold during two consecutive periods. Since thresholds of zero are unrealistic in practice, we used thresholds of 30 for IO count and 0.6 for OIOs in this paper. These numbers were determined based on experimentation, although the approach is not very sensitive to them.

Even after declaring a device idle, it is possible that an IO workload on some host becomes active while the injector is running. The injector continuously applies Little's Law to compute an OIO value using actual IO count and latency measurements. If the measurement differs from the injected value, it is an indication that some other workload must have become active. If there is significant interference (currently defined as more than a 30% OIO discrepancy), the run is terminated and restarted at a later time. When the error is small, we improve accuracy by using the computed OIO value based on empirical measurements, instead of the number of OIOs issued by the injector.

5.1.2 Thin-Provisioned Storage Devices

Thin-provisioned storage devices allocate physical blocks lazily on demand. In some cases, read IOs injected to a thinly-provisioned storage array will return zero-filled data very quickly, because the corresponding block has not even been allocated.

We noticed extremely low slopes for such datastores because most of the IOs had very low latencies. To detect such cases automatically, we check if the latency is smaller than a threshold (by default, 1 ms) and spot-check the data to see if it contains all zeros. If both of these conditions are met, we mark the IO as thin-provisioned and remove it from our measurements. If more than 80% of all points are discarded, the run is marked invalid, and no slope value is computed.

5.1.3 Default Performance Models

If a performance model is not yet available for some datastores, Pesto assigns default models. This is done systematically by assigning either the average slope of the other datastores in the system, or a default slope of 1.0 if none of the datastores have valid models yet. This has proven to be an effective heuristic in practice. For example, with a thin-provisioned datastore, Pesto ends up placing a few workloads on it, so that physical disk blocks will be allocated and subsequent attempts at constructing a real model can succeed.

5.2 Workload Models

In Pesto, fine-grained statistics are measured on a per-virtual-disk level periodically (every 20 seconds) at each of the hosts. These are then sent every 30 minutes to the vSphere management server, which uses the P^2 [15] algorithm to compute online percentiles for important workload metrics. Similar to BASIL [13], our modeling technique primarily characterizes a workload using four parameters: OIOs, IO size, read/write ratio and randomness.

As a workload is moved from one device to another, its workload model may change because of the underlying device characteristics, as described in Section 4.1. For a *closed* workload, new IO requests are generated only when earlier requests complete. Therefore, we expect that the number of OIOs will not change as we move the workload from one storage device to another. For an *open* workload, new IO requests arrive independently of IO completions. For open workloads, we expect a certain degree of change, though drastic changes are rare in practice. This is because the applications running in a VM typically use a certain number of threads to issue IOs, limiting the incoming workload.

In the next three sections, we describe how the automatically generated LQ-slope model and the workload models are used for relieving congestion on a datastore, balancing IO load across multiple datastores, and determining the overall capacity needed to support a given set of workloads.

5.3 Congestion Management

In general, resource allocation policies are needed only for resources that are congested. In the absence of contention, each client should receive the service it requires. When resources are scarce, congestion management can provide differential quality-of-service to clients of varying importance.

In the case of storage devices, which operate more efficiently with higher OIO values, it is not always clear when to start engaging resource management control over the IOs issued by various clients. For example, PARDA [11] throttled IO from hosts accessing shared storage in a distributed virtualized environment when the average observed IO latency exceeded a specified threshold. However, PARDA placed the burden of determining an appropriate latency threshold on the system administrator.

		% of Tests Worse	
Initial Test Setup	**System**	**any**	**>10% regression**
Space balanced	BASIL	13%	5.0%
IO balanced	BASIL	21%	5.3%
Space balanced	Pesto	0%	0.0%
IO balanced	Pesto	0%	0.0%

Table 2: Comparison of BASIL and Pesto results for runs using diverse workloads and 150 test cases.

Using Pesto, an administrator can either specify a desired latency threshold or specify a fraction α of peak throughput. The latency threshold L_α corresponding to throughput of α/m can be computed as:

$$\frac{L_\alpha - C}{m \times L_\alpha} = \frac{\alpha}{m} \tag{8}$$

$$L_\alpha = \frac{C}{1-\alpha} \tag{9}$$

An interesting implication of this result is that the latency corresponding to a certain fraction of peak throughput is independent of the slope of the device. This makes sense, since as the performance of a device increases, it should be able to handle more OIOs for a given latency value. Hence the number of outstanding IOs will vary based on the performance of a device, but not the latency, as long as the intercept is same.

We validated this result by comparing predicted throughput at certain latencies with actual throughput. As shown in Figure 3, the predicted value of throughput matches the actual observed value closely. Similar results were observed for several other devices, including those listed in Figure 4.

5.4 IO Load Balancing

Congestion management addresses contention only at a single storage device. The goal of IO load balancing is to ensure that no single device is overloaded while others are lightly utilized. The basic idea is to equalize the latency observed across devices, similar to BASIL [13].

There are two key differences between BASIL and Pesto load balancing. First, Pesto uses a device model computed by a workload injector, which is more accurate and can be obtained quickly, independent of the actual workload running on the device. Second, Pesto performs additional cost-benefit analysis for each move that is considered. Due to the lack of cost-benefit, BASIL led to bad moves causing performance regression in some cases. We tested against BASIL for two different cases, where the initial placement is done in a space- and IO-balanced manner. Table 2 shows the results for BASIL and Pesto. Next, we explain our cost-benefit functions in more detail.

For each candidate virtual disk migration that is evaluated, there is a source datastore *src* and a destination datastore *dst*. Pesto first filters the move if the 50^{th}- to 90^{th}-percentile *src* latencies are lower than the corresponding *dst* values. BASIL performed this check only for the normalized load on the source and destination datastores. If these checks pass, Pesto computes the cost and benefit for a move. A candidate move is selected only if its benefit exceeds its cost.

5.4.1 Cost-Benefit Computation

Cost: Storage migration induces a non-trivial load on the source and destination during the dominant copy phase of the operation [20]. For each move, Pesto again computes the change in source latency and destination latency using the LQ-slope value and the number of OIOs induced by storage migration (typically 16). Given these numbers, we use the following equation to compute the cost:

$$Cost = Q_{src} \times \Delta L_{src} + Q_{dst} \times \Delta L_{dst} \tag{10}$$

Since we maintain percentile values for all these numbers, the cost is computed for each percentile and summed up to obtain the total cost. This cost is paid for the duration of the migration, which is estimated by considering the total IOPS available at source and destination. Available IOPS are computed by using the peak IOPS estimation at each datastore and the existing IOPS already being used. The lower of the available IOPS at source and destination, along with the disk size, is used to estimate the migration time.

Benefit: For each move we compute the change in source latency and destination latency by using the current latency and the fraction of the load that is being moved. If the destination has no load, we use the LQ-slope to compute the estimated latency at the destination using the OIOs corresponding to the move. Once we have these numbers, we use the following equation to compute the benefit:

$$\begin{aligned} Benefit \;=\; & Q_{src} \times \Delta L_{src} - Q_{dst} \times \Delta L_{dst} \\ & + Q_{mig} \times (L_{src}^{before} - L_{dst}^{after}) \end{aligned} \tag{11}$$

As with the cost, the benefit is computed for each percentile value and summed to obtain the total benefit. Pesto conservatively estimates the duration of the benefit to be the invocation frequency of the load-balancing module (set to 16 hours by default).

5.4.2 Other Features Enabled by Pesto

Initial Placement: During a typical initial placement, Pesto does not have a workload model for the new virtual disk being placed. We cannot use all-zero statistics in this case, as that could result in all new virtual disks being placed on only one or a few datastores. Instead, we compute the average workload model for all existing virtual disks and use this average for the incoming disk. Candidate placements for the new disk are considered on each datastore and the one resulting in the most-balanced placement is selected.

Datastore Removal: Administrators may want to remove a datastore from active use, in order to fix some faults or simply to eliminate it. We refer to this operation as putting a datastore into *maintenance mode*. Pesto provides this feature by migrating all of the virtual disks from that datastore to the remaining datastores. This effectively results in a series of initial placement operations, except that valid workload models already exist for the virtual disks.

Datastore Addition: The case of a new datastore being added to the system is handled by regular load balancing, using the newly-created device model. If for any reason the model is not yet available, a conservative default model is used for the datastore (see Section 5.1.3).

5.5 Capacity Planning

Capacity planning is a common storage provisioning task. A typical capacity planning problem is: How many workloads of a certain type can fit on a device? A complete answer must consider both available space capacity (in GB), as well as IO performance in terms of throughput and/or latency.

Space planning is an important problem, but relatively straightforward if the storage device is not using technologies such as thin provisioning or deduplication, which complicate free-space calculations. To handle out-of-space situations, we allow a user to set a space consumption threshold per datastore. If any datastore exceeds this threshold, Pesto considers the eviction of various virtual disks from that datastore and selects a move that reduces space consumption with the best IO load balance. In this paper, we focus on performance-based capacity planning. We allow an administrator to specify the desired performance policy in terms of either latency or a share of peak throughput.

5.5.1 Latency-Based Policy

With a latency-based policy, an administrator specifies a latency threshold, L_{max}, which serves as an upper bound on the desired response time from the storage device. During capacity planning, workloads can be mapped to storage devices until the projected L_{max} is reached. This case is relatively simple, since the latency bound can be converted into the number of outstanding IOs that will keep the actual latency below this bound using Equation 12. New workloads can be added until the total expected OIOs reach Q_{max}.

$$L_{max} = m \times Q_{max} + C \tag{12}$$

Once the Q_{max} limit is reached, it may still be feasible to add more workloads if the latency is smaller than expected, due to workload locality or caching at the array. In such cases, one can add more workloads in an incremental and cautious manner.

5.5.2 Peak-Throughput-Based Policy

Alternatively, an administrator can express a desired threshold as a fraction α of the peak device throughput. During capacity planning, workloads can be mapped to devices until throughput reaches this threshold. We can determine the number of outstanding IOs, Q_α, that correspond to the desired throughput using Equation 13. This policy can be used to maximize the device utilization in terms of overall throughput.

$$\frac{Q_\alpha}{m \times Q_\alpha + C} = \frac{\alpha}{m} \tag{13}$$

For capacity planning, an administrator may also want to know if all datastores are operating close to their peak throughput. If some datastores are operating close to the congestion threshold set by PARDA, and IO load balancing is not able to alleviate the problem, then new datastores need to be added to handle the workload. This situation can be detected easily by setting a threshold alarm on the datastore latency metric.

In order to perform these computations, Pesto needs some estimate of overall number of outstanding IOs for a given workload. This may not be known in many cases. One can obtain OIO values either by profiling similar workloads or by using an approximate value based on average loads from existing workloads. In case of misprediction, Pesto will correct workload placements later, based on actual online measurements.

6. EXPERIMENTAL EVALUATION

For our end-to-end system evaluation, we used an experimental setup consisting of 6 ESX hosts and 8 datastores backed by disks of different types and reliability levels in four storage arrays from three different vendors: EMC CLARiioN, Iomega and NetApp. We used 28 VMs, each with multiple virtual disks, running both micro-benchmarks and real enterprise workloads. The unit of storage migration in Pesto is a virtual disk; our setup included a total of 66 virtual disks. Table 3 shows the experimental setup details. We used a diverse set of workloads, hardware, and datastores in order to represent the evolution of a datacenter over a few years.

Experimental Setup
7 datastores: 1 RAID-0, 6 RAID-5; all have FC disks **6 ESX hosts**: 5 HP Proliant (4-16 cores, 8-16 GB RAM), 1 Dell Poweredge (4 cores, 8 GB RAM) **28 VMs**: 6 filebench-oltp, 6 filebench-mail, 6 filebench-webserver, 1 DVDStore, 2 Swingbench, 7 Windows-Iometer

Table 3: Experimental setup

Datastore	Slope (ms)	Intercept (ms)	Peak IOPS	Congestion Threshold
Datastore1	0.55	5.18	1818	26 ms
Datastore2	0.54	5.93	1851	30 ms
Datastore3	0.22	5.90	4545	30 ms
Datastore4	0.48	5.10	2083	25 ms
Datastore5	0.27	4.28	3703	22 ms
Datastore6	0.29	4.70	3448	24 ms
Datastore7	1.02	11.12	1000	60 ms

Table 4: Datastore models and congestion thresholds computed by Pesto.

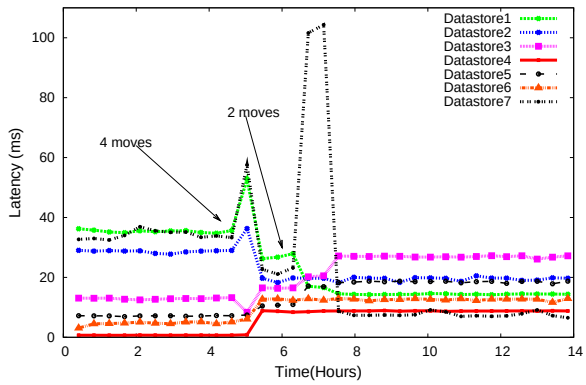
6.1 Device Model and Congestion Avoidance

When we enabled Pesto, it was able to construct a device model from scratch within minutes for all datastores. This is significant because BASIL requires workloads to be running and would have taken several hours at a minimum before producing performance models. In addition, Pesto automatically computed the congestion threshold based on the performance model and the desired operating point set by the administrator. Table 4 shows the slope, intercept, peak IOPS and congestion threshold latency values as computed by Pesto. The automatic congestion thresholds were computed based on the administrator-set operating point of 80% of peak throughput. The system can also be configured to respect a global maximum for the congestion threshold latency value.

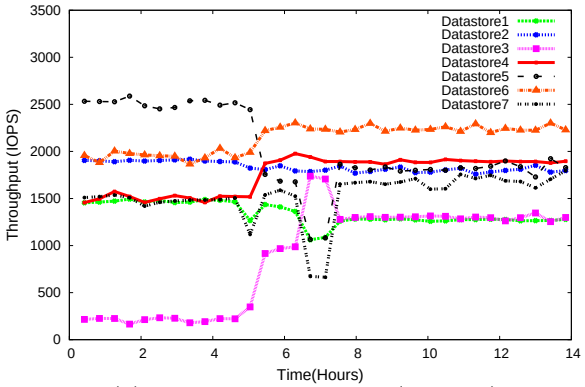
Recall that PARDA [11] uses a fixed, manually set congestion threshold (default of 30 ms), whereas Table 4 clearly shows that the appropriate threshold varies over a range of 22 to 60 ms for datastores based on their disk types. With Pesto, an administrator doesn't need to worry about picking the best congestion threshold value, given the option to instead choose an operating point in terms of a percentage of peak throughput.

6.2 IO Load Balancing

We initially assigned virtual disks to datastores randomly, but in a greedy manner so as to keep a space-balanced arrangement. This focused our evaluation on Pesto's performance management algorithms, rather than the system's out-of-space avoidance heuristics. We ran the VM work-



(a) Datastore latencies (in ms)



(b) Datastore throughput (in IOPS)

Figure 10: IO Load Balancing: Per-datastore latency and throughput observed during multi-hour experiment.

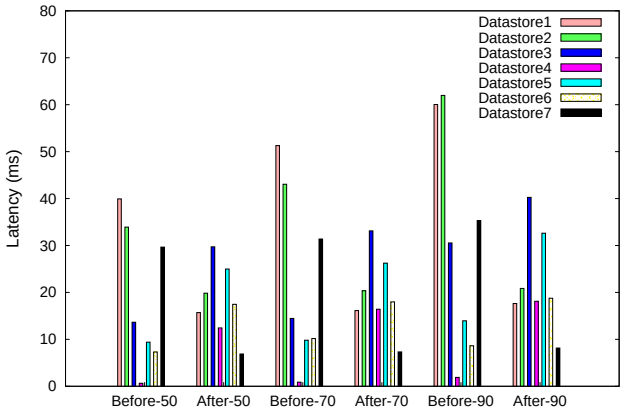


Figure 11: Latency Balance: 50^{th}, 70^{th} and 90^{th} percentile latencies before and after load balancing.

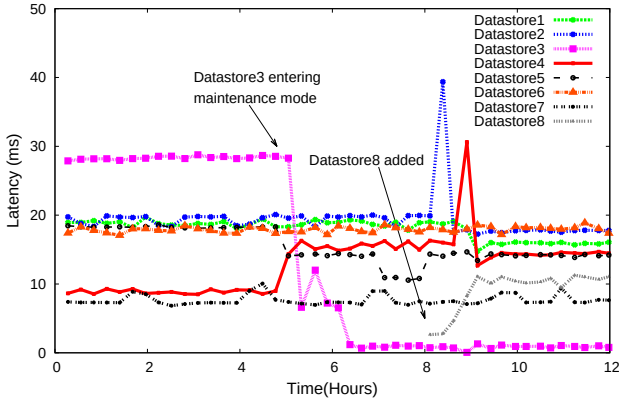


Figure 12: Datastore Changes: Latency observed across datastore removal and addition.

loads for eight hours. Based on the eight-hour statistics, Pesto suggested 4 recommendations to move virtual disks using storage vMotion [20, 34], which took approximately 40 minutes to complete. The workloads were not stopped during this process. After running the workloads in the new configuration for a while, Pesto recommended two additional virtual disk movements; after that the system converged and no further recommendations were generated.

Figure 10 plots per-datastore latency and throughput, averaged over 5-minute intervals during our 18-hour run. The initial four hours of each run are not shown, since nothing of interest was observed during this warmup period.The plotted data covers the time periods before, during and after the storage migration recommendations were applied.

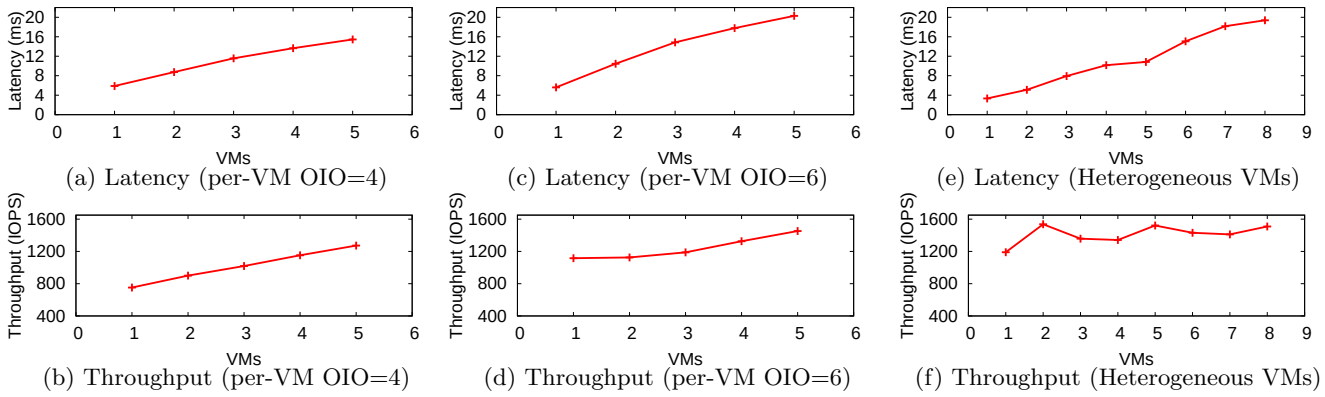
Figure 11 shows the 50^{th}, 70^{th} and 90^{th} percentile latency values used by the load balancing algorithm to perform cost-benefit analysis and recommend migrations. The Pesto-recommended moves improved overall average throughput by 11% from 10900 to 12122 IOPS. Similarly, the overall throughput-weighted average latency across all datastores decreased by 16.9%, from 17.6 ms to 14.6 ms. Users often care more about improvements in performance during peak loads than they do about the average improvement. At the 70^{th} percentile, the latency improvement was 19.8% whereas an 18.8% lower latency was observed at the 90^{th} percentile.

Over time, the latency across datastores became more balanced. This had the nice side effect of improving the performance of virtual machines that were the worst off in the previous, space-balanced configuration. All these improvements happened without manual intervention; Pesto generated and applied recommendations automatically.

6.3 Datastore Maintenance Mode

We tested both initial placement and the datastore maintenance mode functionality in a single experiment: putting Datastore3 into maintenance mode would force "initial placement" of its 3 virtual disks onto the remaining datastores. This experiment was conducted on the same testbed as in Section 6.2, using the final, stable Pesto virtual disk placement. Pesto recommended moving the 3 virtual disks to Datastore4, Datastore5 and Datastore6, respectively.

Figure 12 shows the latency observed by all datastores. The time on the x-axis is renumbered from 0. The maintenance mode operation was invoked at $t = 5$ hrs. Note that the move destinations are the datastores with lower latency. Datastore4 had higher latency after the move but overall balance is maintained. Datastore5 received a virtual disk issuing 25% writes, leading to a reduction in overall latency, since writes typically return early from the array cache with very low latencies. Datastore6 received a virtual



(a) Latency (per-VM OIO=4) (c) Latency (per-VM OIO=6) (e) Latency (Heterogeneous VMs)

(b) Throughput (per-VM OIO=4) (d) Throughput (per-VM OIO=6) (f) Throughput (Heterogeneous VMs)

Figure 14: Capacity Planning: Predicting number of VMs to reach a certain latency or throughput value.

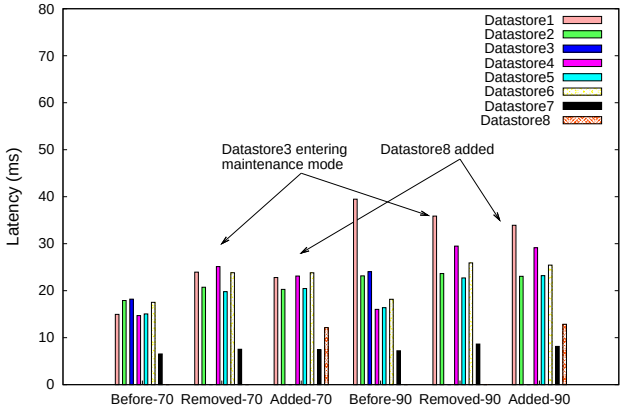


Figure 13: 70^{th} and 90^{th} percentile latency values before, after putting datastore in maintenance and adding a replacement datastore

disk with very light load, so its latency didn't change much. Figure 13 shows the 70^{th} and 90^{th} percentile latencies across datastores before and after the removal of Datastore3. We observed that the variance was similar in both cases showing that Pesto is able to place workloads in a balanced manner.

6.4 Adding a New Device

To test how Pesto handles the addition of a new datastore, we continued the previous experiment, adding a new datastore at time $t = 8$ hrs in Figure 12. Within the first two minutes, Pesto computed a device model with $m = 1.09$ and $C = 5.43$. It then recommended moving three virtual disks from Datastore1, Datastore2 and Datastore4, respectively. Figure 12 shows the datastore latencies before and after the moves.

Note that the overall latency across all datastores is lower and throughput is higher after the moves. This shows that Pesto can quickly make use of additional resources that are added to the system, mitigating higher latencies and improving overall throughput. Figure 13 shows the 70^{th} and 90^{th} percentile latencies before and after the addition of Datastore8. We again observed that the variance was similar in both cases showing that Pesto is able to place virtual disks on the new datastore while maintaining IO load balance.

6.5 Capacity Planning

We ran two experiments to test our capacity-planning approach using homogeneous and heterogeneous workloads. Our first experiment used identical workloads running in a set of VMs, each configured as an OLTP workload using Filebench [1] on a 20 GB data disk and a 1 GB log disk. The data disk is separate from the log disk and the VM's OS system disk. The experimental virtual disks were all placed on a single storage device, consisting of 6 FC disks in a RAID-5 configuration on an EMC CLARiioN array having a performance model with $m = 0.49$ and $C = 4.98$.

The Filebench workload profiles for the OLTP VMs were configured for either 4 or 6 outstanding IOs. We tested two policies: one where VMs can be placed until the latency reaches 15 ms, and another where throughput reaches 66% of the peak value. Based on the performance model, both of these conditions happen at 20 OIOs. This corresponds to the number of VMs that could be accommodated as 5 VMs with 4 OIOs, or 3 VMs with 6 OIOs. Figures 14(a)-(d) plot latency and throughput as the number of VMs accessing the LUN is varied from 1 to 5, each running an OLTP workload with either 4 or 6 OIOs. The estimated capacity based on the performance model closely matches the actual capacity.

Next, we experimented with heterogeneous workloads consisting of three different workload profiles in Filebench: OLTP, webserver and varmail. Each workload runs in its own VM, and its corresponding virtual disks are placed on the same datastore used in the first experiment. We added the workloads in the following order: two varmail, two webservers and then four OLTP VMs. Given the workload profile, we expect 4-6 OIOs from varmail, 1-2 OIOs from webserver and 4 OIOs from OLTP workloads.

Recall that based on the performance model, 15 ms latency should correspond to $Q = 20$ OIOs. Given the OIO profiles for these VMs and the order in which VMs are added, we expect that the datastore should be able to support 2 varmail, 2 webserver and 2 OLTP workloads – a total of 22 OIOs – before reaching the latency target of 15 ms. Figure 14(e) shows that after adding these VMs the latency of the LUN is 15.07 ms. Similarly, we expect the throughput to be close to 75% of peak throughput once all the VMs are added: this is confirmed in Figures 14(e)-(f), which indicate throughput of roughly 1500 IOPS at 20 ms latency. Our throughput

predictions are less accurate, but the peak throughput is inline with the model predictions.

7. CONCLUSIONS

In this paper, we presented Pesto, an automated and online storage management system for virtualized datacenters. At its core, Pesto uses our new analytical result that the peak throughput of a storage device is approximately equal to the inverse of its LQ-slope (*i.e.*, the rate of change of latency against the number of outstanding IOs). Leveraging this result, Pesto constructs online device models and supports powerful operations across a group of datastores, managing them as a single pool of resources. Our experimental evaluation on a diverse set of storage devices demonstrated that Pesto's online device model is accurate enough to guide storage planning and management decisions. In addition, we showed the effectiveness of our modeling techniques for several well-known storage provisioning tasks, including load balancing, addition and removal of datastores, congestion management and capacity planning. Pesto-recommended placement improved overall throughput by more than 10% and reduced peak-load latency by up to 19%.

Apart from the practical applications shown in this paper, our black-box model can be used to study the performance of LUNs without requiring physical access. This is especially useful in cloud scenarios, where customers leasing storage from service providers do not have visibility into the provided service levels. Using our technique, customers can verify device performance quickly, using an injected workload, and estimate the peak throughput of the device. The LQ-slope for a standard workload can be used to compare the relative performance of LUNs [13]. We believe that the storage industry could benefit from using LQ-slope as an independently-verifiable metric to describe the performance of devices with particular configurations.

Acknowledgments

We would like to thank Anne Holler, Shankari Kalyanaraman, Limin Wang, Tahir Mobashir, Rajasekar Shanmugam and all other members of resource management team at VMware, who contributed to building the necessary infrastructure, statistics collection framework, test suite, UI workflows and helped create a product based on this research. We are very grateful to Jinpyo Kim, Tariq Magdon-Ismail, Emre Celebi, Chethan Kumar and Xiaoyun Zhu for extensive performance testing and numerous discussions that helped us uncover many issues and add various optimizations during the development process. We are also thankful to the anonymous reviewers for their valuable insights and feedback that helped us improve the paper and add necessary details.

8. REFERENCES

[1] Filebench. `http://solarisinternals.com/si/tools/filebench/index.php`.

[2] Iometer. `http://www.iometer.org`.

[3] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An Automated Resource Provisioning Tool for Large-Scale Storage Systems. In *ACM Transactions on Computer Systems*, Nov. 2001.

[4] E. Anderson. Simple table-based modeling of storage devices. Technical report, HPL-SSP-2001-4, HP Labs, July 2001.

[5] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Proceedings of the 1st USENIX conference on File and Storage Technologies*, FAST'02, Berkeley, CA, USA, 2002. USENIX Association.

[6] S. Chen and D. Towsley. The Design and Evaluation of RAID 5 and Parity Striping Disk Array Architectures. *Journal on Parallel and Distributed Computing*, 17(1-2):58–74, 1993.

[7] S. Chen and D. Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45:1116–1130, 1996.

[8] T. E. Denehy, J. Bent, F. I. Popovici, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Deconstructing storage arrays. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XI, pages 59–71, New York, NY, USA, 2004. ACM.

[9] G. Ganger. Automated disk drive characterization. `http://www.pdl.cmu.edu/Dixtrac/index.shtml`.

[10] C. C. Gotlieb and G. H. MacEwen. Performance of Movable-Head Disk Storage Devices. *Journal of the ACM*, 20(4):604–623, 1973.

[11] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional allocation of resources for distributed storage access. In *Proccedings of the 7th conference on File and Storage Technologies*, pages 85–98, Berkeley, CA, USA, 2009. USENIX Association.

[12] A. Gulati, C. Kumar, and I. Ahmad. Storage Workload Characterization and Consolidation in Virtualized Environments. In *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009.

[13] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar. BASIL: Automated IO load balancing across storage devices. In *Proceedings of the 8th USENIX conference on File and Storage Technologies*, FAST'10, Berkeley, CA, USA, 2010. USENIX Association.

[14] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach, Fourth edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[15] R. Jain and I. Chlamtac. The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28:1076–1085, October 1985.

[16] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton. Inducing models of black-box storage arrays. Technical Report HPL-2004-108, HP Labs, 2004.

[17] M. Kim and A. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Transactions on Computers*, 40(7):801–810, 1991.

[18] E. K. Lee and R. H. Katz. An analytic performance model of disk arrays. *SIGMETRICS Performance Evaluation Review*, 21(1):98–109, 1993.

[19] J. D. C. Little. A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research*, 9(3), 1961.

[20] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai. The Design and Evolution of Live Storage Migration in VMware ESX. In *Proc. USENIX Annual Technical Conference (ATC '11)*, June 2011.

[21] A. Merchant and P. S. Yu. An analytical model of reconstruction time in mirrored disks. *Performance Evaluation*, 20:115–129, May 1994.

[22] A. Merchant and P. S. Yu. Analytic Modeling of Clustered RAID with Mapping Based on Nearly Random Permutation. *IEEE Transactions on Computers*, 45(3), 1996.

[23] D. R. Merrill. Storage economics: Four principles for reducing total cost of ownership. May 2009. `http://www.hds.com/assets/pdf/four-principles-for-reducing-total-cost-of-ownership.pdf`.

[24] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Modeling the relative fitness of storage.

SIGMETRICS Performance Evaluation Review, 35(1), 2007.

[25] J. D. Padhye, A. L. Rahatekar., and L. W. Dowdy. A Simple LAN File Placement Strategy. In *International CMG Conference*, 1995.

[26] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27:17–28, 1994.

[27] E. Shriver, A. Merchant, and J. Wilkes. An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering. *SIGMETRICS Performance Evaluation Review*, 26(1):182–191, 1998.

[28] N. Simpson. Building a data center cost model. Jan 2010. `http://www.burtongroup.com/Research/DocumentList.aspx?cid=49`.

[29] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, and G. R. Ganger. Informed data distribution selection in a self-predicting storage system. In *International Conference on Autonomic Computing*, 2006.

[30] E. Thereska and G. R. Ganger. IRONModel: Robust performance models in the wild. *SIGMETRICS Performance Evaluation Review*, 36:253–264, June 2008.

[31] A. Thomasian and J. Menon. Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation. In *Proceedings of the Tenth International Conference on Data Engineering*, pages 111–119. IEEE Computer Society, 1994.

[32] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, 2001.

[33] E. Varki, A. Merchant, J. Xu, and X. Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems*, 15:559–574, 2004.

[34] VMware, Inc. VMware Storage VMotion: Non-Disruptive, Live Migration of Virtual Machine Storage, 2007. `http://vmware.com/files/pdf/storage_vmotion_datasheet.pdf`.

[35] VMware, Inc. *vSphere Resource Management Guide: ESX 4.1, ESXi 4.1, vCenter Server 4.1*. 2010.

[36] VMware, Inc. *VMware vSphere*. 2011. http://www.vmware.com/products/vsphere/overview.html.

[37] VMware, Inc. *VMware vStorage VMFS*. 2011. http://www.vmware.com/files/pdf/VMware-vStorage-VMFS-DS-EN.pdf.

[38] T. Voellm. Useful IO profiles for simulating various workloads. `http://blogs.msdn.com/b/tvoellm/archive/2009/05/07/useful-io-profiles-for-simulating-various-workloads.aspx`.

[39] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage Device Performance Prediction with CART Models. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 588–595, 2004.

[40] P. S. Yu and A. Merchant. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Transactions on Computers*, 44:419–433, March 1995.