BASIL: Automated IO Load Balancing Across Storage Devices

Ajay Gulati
VMware, Inc.
agulati@vmware.com

Chethan Kumar
VMware, Inc.
ckumar@vmware.com

Irfan Ahmad
VMware, Inc.
irfan@vmware.com

Karan Kumar
Carnegie Mellon University
karank@andrew.cmu.edu

Abstract

Live migration of virtual hard disks between storage arrays has long been possible. However, there is a dearth of online tools to perform automated virtual disk placement and IO load balancing across multiple storage arrays. This problem is quite challenging because the performance of IO workloads depends heavily on their own characteristics and that of the underlying storage device. Moreover, many device-specific details are hidden behind the interface exposed by storage arrays.

In this paper, we introduce BASIL, a novel software system that automatically manages virtual disk placement and performs load balancing across devices without assuming any support from the storage arrays. BASIL uses IO latency as a primary metric for modeling. Our technique involves separate online modeling of workloads and storage devices. BASIL uses these models to recommend migrations between devices to balance load and improve overall performance.

We present the design and implementation of BASIL in the context of VMware ESX, a hypervisor-based virtualization system, and demonstrate that the modeling works well for a wide range of workloads and devices. We evaluate the placements recommended by BASIL, and show that they lead to improvements of at least 25% in both latency and throughput for 80 percent of the hundreds of microbenchmark configurations we ran. When tested with enterprise applications, BASIL performed favorably versus human experts, improving latency by 18-27%.

1 Introduction

Live migration of virtual machines has been used extensively in order to manage CPU and memory resources, and to improve overall utilization across multiple physical hosts. Tools such as VMware's Distributed Resource Scheduler (DRS) perform automated placement of virtual machines (VMs) on a cluster of hosts in an efficient and effective manner [6]. However, automatic placement and load balancing of IO workloads across a set of storage devices has remained an open problem. Diverse IO behavior from various workloads and hot-spotting can cause significant imbalance across devices over time.

An automated tool would also enable the aggregation of multiple storage devices (LUNs), also known as data stores, into a single, flexible pool of storage that we call a POD (*i.e.* Pool of Data stores). Administrators can dynamically populate PODs with data stores of similar reliability characteristics and then just associate virtual disks with a POD. The load balancer would take care of initial placement as well as future migrations based on actual workload measurements. The flexibility of separating the physical from the logical greatly simplifies storage management by allowing data stores to be efficiently and dynamically added or removed from PODs to deal with maintenance, out of space conditions and performance issues.

In spite of significant research towards storage configuration, workload characterization, array modeling and automatic data placement [8, 10, 12, 15, 21], most storage administrators in IT organizations today rely on rules of thumb and ad hoc techniques, both for configuring a storage array and laying out data on different LUNs. For example, placement of workloads is often based on balancing space consumption or the number of workloads on each data store, which can lead to hot-spotting of IOs on fewer devices. Over-provisioning is also used in some cases to mitigate real or perceived performance issues and to isolate top-tier workloads.

The need for a storage management utility is even greater in virtualized environments because of high degrees of storage consolidation and sprawl of virtual disks over tens to hundreds of data stores. Figure 1 shows a typical setup in a virtualized datacenter, where a set of hosts has access to multiple shared data stores. The storage array is carved up into groups of disks with some RAID level configuration. Each such disk group is further di-

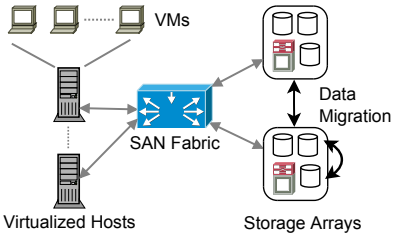

Figure 1: Live virtual disk migration between devices.

vided into LUNs which are exported to hosts as storage devices (referred to interchangeably as data stores). Initial placement of virtual disks and data migration across different data stores should be guided by workload characterization, device modeling and analysis to improve IO performance as well as utilization of storage devices. This is more difficult than CPU or memory allocation because storage is a stateful resource: IO performance depends strongly on workload and device characteristics.

In this paper, we present the design and implementation of BASIL, a light-weight online storage management system. BASIL is novel in two key ways: (1) identifying IO latency as the primary metric for modeling, and (2) using simple models both for workloads and devices that can be obtained efficiently online. BASIL uses IO latency as the main metric because of its near linear relationship with application-level characteristics (shown later in Section 3). Throughput and bandwidth, on the other hand, behave non-linearly with respect to various workload characteristics.

For modeling, we partition the measurements into two sets. First are the properties that are inherent to a workload and mostly independent of the underlying device such as seek-distance profile, IO size, read-write ratio and number of outstanding IOs. Second are device dependent measurements such as IOPS and IO latency. We use the first set to model workloads and a subset of the latter to model devices. Based on measurements and the corresponding models, the analyzer assigns the IO load in proportion to the performance of each storage device.

We have prototyped BASIL in a real environment with a set of virtualized servers, each running multiple VMs placed across many data stores. Our extensive evaluation based on hundreds of workloads and tens of device configurations shows that our models are simple yet effective. Results indicate that BASIL achieves improvements in throughput of at least 25% and latency reduction of at least 33% in over 80 percent of all of our test configurations. In fact, approximately half the tests cases saw at least 50% better throughput and latency. BASIL achieves optimal initial placement of virtual disks in 68% of our experiments. For load balancing of enterprise applications, BASIL outperforms human experts by improving latency by 18-27% and throughput by up to 10%.

The next section presents some background on the relevant prior work and a comparison with BASIL. Section 3 discusses details of our workload characterization and modeling techniques. Device modeling techniques and storage specific issues are discussed in Section 4. Load balancing and initial placement algorithms are described in Section 5. Section 6 presents the results of our extensive evaluation on real testbeds. Finally, we conclude with some directions for future work in Section 7.

2 Background and Prior Art

Storage management has been an active area of research in the past decade but the state of the art still consists of rules of thumb, guess work and extensive manual tuning. Prior work has focused on a variety of related problems such as disk drive and array modeling, storage array configuration, workload characterization and data migration.

Existing modeling approaches can be classified as either *white-box* or *black-box*, based on the need for detailed information about internals of a storage device. Black-box models are generally preferred because they are oblivious to the internal details of arrays and can be widely deployed in practice. Another classification is based on *absolute* vs. *relative* modeling of devices. Absolute models try to predict the actual bandwidth, IOPS and/or latency for a given workload when placed on a storage device. In contrast, a relative model may just provide the relative change in performance of a workload from device A to B. The latter is more useful if a workload's performance on one of the devices is already known. Our approach (BASIL) is a black-box technique that relies on the relative performance modeling of storage devices.

Automated management tools such as Hippodrome [10] and Minerva [8] have been proposed in prior work to ease the tasks of a storage administrator. Hippodrome automates storage system configuration by iterating over three stages: analyze workloads, design the new system and implement the new design. Similarly, Minerva [8] uses a declarative specification of application requirements and device capabilities to solve a constraint-based optimization problem for storage-system design. The goal is to come up with the best array configuration for a workload. The workload characteristics used by both Minerva and Hippodrome are somewhat more detailed and different than ours. These tools are trying to solve a different and a more difficult problem of optimizing overall storage system configuration. We instead focus on load balancing of IO workloads among existing storage devices across multiple arrays.

Mesnier *et al.* [15] proposed a black-box approach based on evaluating relative fitness of storage devices to predict the performance of a workload as it is moved

from its current storage device to another. Their approach requires extensive training data to create relative fitness models among every *pair* of devices. Practically speaking, this is hard to do in an enterprise environment where storage devices may get added over time and may not be available for such analysis. They also do very extensive offline modeling for bandwidth, IOPS and latency and we derive a much simpler device model consisting of a single parameter in a completely online manner. As such, our models may be somewhat less detailed or less accurate, but experimentation shows that they work well enough in practice to guide our load balancer. Their model can potentially be integrated with our load balancer as an input into our own device modeling.

Analytical models have been proposed in the past for both single disk drives and storage arrays [14, 17, 19, 20]. Other models include table-based [9] and machine learning [22] techniques. These models try to accurately predict the performance of a storage device given a particular workload. Most analytical models require detailed knowledge of the storage device such as sectors per track, cache sizes, read-ahead policies, RAID type, RPM for disks etc. Such information is very hard to obtain automatically in real systems, and most of it is abstracted out in the interfaces presented by storage arrays to the hosts. Others need an extensive offline analysis to generate device models. One key requirement that BASIL addresses is using only the information that can be easily collected online in a live system using existing performance monitoring tools. While one can clearly make better predictions given more detailed information and exclusive, offline access to storage devices, we don't consider this practical for real deployments.

3 Workload Characterization

Any attempt at designing intelligent IO-aware placement policies must start with storage workload characterization as an essential first step. For each workload in our system, we currently track the average IO latency along the following parameters: seek distance, IO sizes, read-write ratio and average number of outstanding IOs. We use the VMware ESX hypervisor, in which these parameters can be easily obtained for each VM and each virtual disk in an online, light-weight and transparent manner [7]. A similar tool is available for Xen [18]. Data is collected for both reads and writes to identify any potential anomalies in the application or device behavior towards different request types.

We have observed that, to the first approximation, four of our measured parameters (*i.e.*, randomness, IO size, read-write ratio and average outstanding IOs) are inherent to a workload and are mostly independent of the underlying device. In actual fact, some of the characteristics that we classify as inherent to a workload can indeed be partially dependent on the response times delivered by the storage device; *e.g.*, IO sizes for a database logger might decrease as IO latencies decrease. In previous work [15], Mesnier *et al.* modeled the change in workload as it is moved from one device to another. According to their data, most characteristics showed a small change except write seek distance. Our model makes this assumption for simplicity and errors associated with this assumption appear to be quite small.

Our workload model tries to predict a notion of load that a workload might induce on storage devices using these characteristics. In order to develop a model, we ran a large set of experiments varying the values of each of these parameters using Iometer [3] inside a Microsoft Windows 2003 VM accessing a 4-disk RAID-0 LUN on an EMC CLARiiON array. The set of values chosen for our 750 configurations are a cross-product of:

Outstanding IOs {4, 8, 16, 32, 64}
IO size (in KB) {8, 16, 32, 128, 256, 512}
Read% {0, 25, 50, 75, 100}
Random% {0, 25, 50, 75, 100}

For each of these configurations we obtain the values of average IO latency and IOPS, both for reads and writes. For the purpose of workload modeling, we next discuss some representative sample observations of average IO latency for each one of these parameters while keeping the others fixed. Figure 2(a) shows the relationship between IO latency and outstanding IOs (OIOs) for various workload configurations. We note that latency varies linearly with the number of outstanding IOs for all the configurations. This is expected because as the total number of OIOs increases, the overall queuing delay should increase linearly with it. For very small number of OIOs, we may see non-linear behavior because of the improvement in device throughput but over a reasonable range (8-64) of OIOs, we consistently observe very linear behavior. Similarly, IO latency tends to vary linearly with the variation in IO sizes as shown in Figure 2(b). This is because the transmission delay increases linearly with IO size.

Figure 2(c) shows the variation of IO latency as we increase the percentage of reads in the workload. Interestingly, the latency again varies linearly with read percentage except for some non-linearity around corner cases such as completely sequential workloads. We use the read-write ratio as a parameter in our modeling because we noticed that, for most cases, the read latencies were very different compared to write (almost an order of magnitude higher) making it important to characterize a workload using this parameter. We believe that the difference in latencies is mainly due to the fact that writes return once they are written to the cache at the array and the latency of destaging is hidden from the application. Of course, in cases where the cache is almost full, the

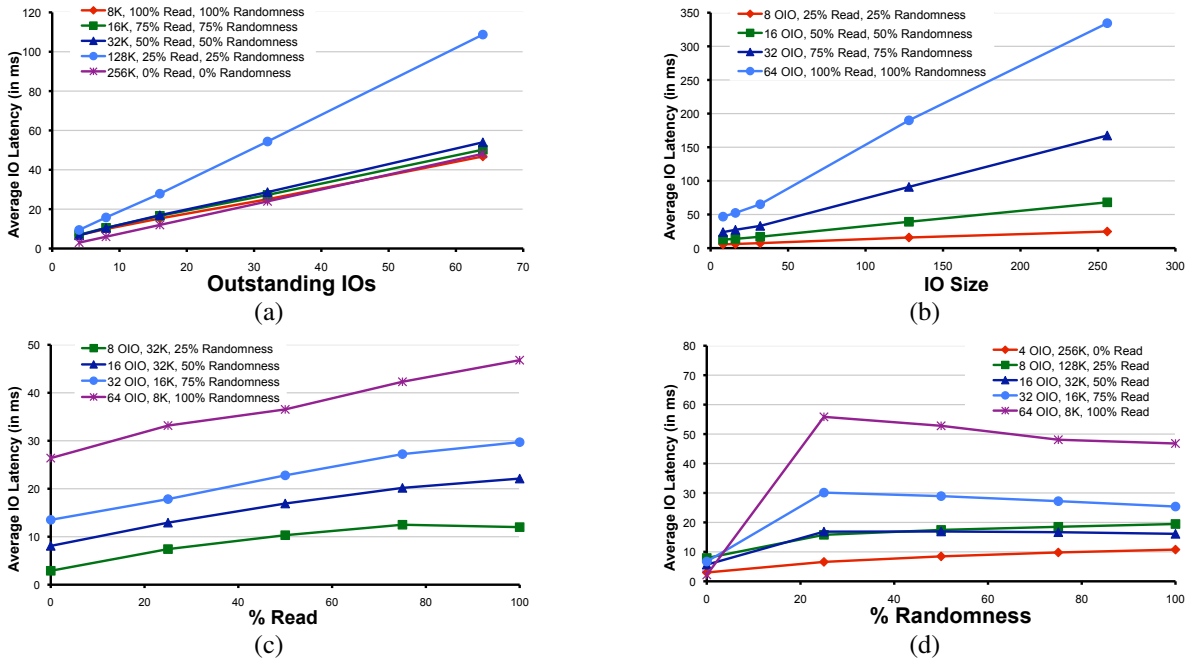

Figure 2: Variation of IO latency with respect to each of the four workload characteristics: outstanding IOs, IO size, % Reads and % Randomness. Experiments run on a 4-disk RAID-0 LUN on an EMC CLARiiON CX3-40 array.

writes may see latencies closer to the reads. We believe this to be fairly uncommon especially given the burstiness of most enterprise applications [12]. Finally, the variation of latency with random% is shown in Figure 2(d). Notice the linear relationship with a very small slope, except for a big drop in latency for the completely sequential workload. These results show that except for extreme cases such as 100% sequential or 100% write workloads, the behavior of latency with respect to these parameters is quite close to linear[1]. Another key observation is that the cases where we typically observe non-linearity are easy to identify using their online characterization.

Based on these observations, we modeled the IO latency (L) of a workload using the following equation:

$$L = \frac{(K_1 + OIO)(K_2 + IOsize)(K_3 + \frac{read\%}{100})(K_4 + \frac{random\%}{100})}{K_5} \quad (1)$$

We compute all of the constants in the above equation using the data points available to us. We explain the computation of K_1 here, other constants K_2, K_3 and K_4 are computed in a similar manner. To compute K_1, we take two latency measurements with different OIO values but the same value for the other three workload parameters. Then by dividing the two equations we get:

$$\frac{L_1}{L_2} = \frac{K_1 + OIO_1}{K_1 + OIO_2} \quad (2)$$

[1]The small negative slope in some cases in Figure 2(d) with large OIOs is due to known prefetching issues in our target array's firmware version. This effect went away when prefetching is turned off.

$$K_1 = \frac{OIO_1 - OIO_2 * L_1/L_2}{L_1/L_2 - 1} \quad (3)$$

We compute the value of K_1 for all pairs where the three parameters except OIO are identical and take the median of the set of values obtained as K_1. The values of K_1 fall within a range with some outliers and picking a median ensures that we are not biased by a few extreme values. We repeat the same procedure to obtain other constants in the numerator of Equation 1.

To obtain the value of K_5, we compute a linear fit between actual latency values and the value of the numerator based on K_i values. Linear fitting returns the value of K_5 that minimizes the least square error between the actual measured values of latency and our estimated values.

Using IO latencies for training our workload model creates some dependence on the underlying device and storage array architectures. While this isn't ideal, we argue that as a practical matter, if the associated errors are small enough, and if the high error cases can usually be identified and dealt with separately, the simplicity of our modeling approach makes it an attractive technique.

Once we determined all the constants of the model in Equation 1, we compared the computed and actual latency values. Figure 3(a) (LUN1) shows the relative error between the actual and computed latency values for all workload configurations. Note that the computed values do a fairly good job of tracking the actual values in most cases. We individually studied the data points with high errors and the majority of those were sequential IO

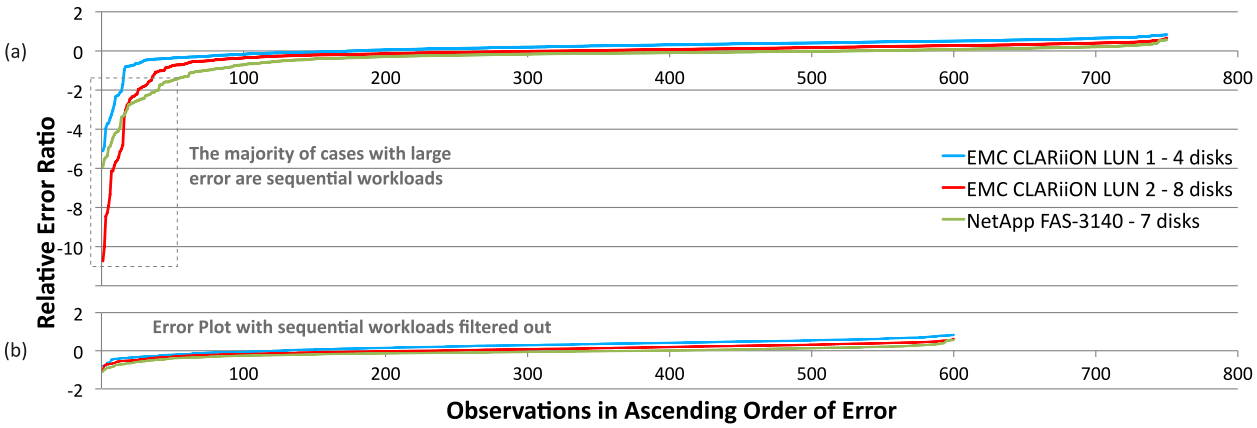

Figure 3: Relative error in latency computation based on our formula and actual latency values observed.

or write-only patterns. Figure 3(b) plots the same data but with the 100% sequential workloads filtered out.

In order to validate our modeling technique, we ran the same 750 workload configurations on a different LUN on the same EMC storage array, this time with 8 disks. We used the same values of K_1, K_2, K_3 and K_4 as computed before on the 4-disk LUN. Since the disk types and RAID configuration was identical, K_5 should vary in proportion with the number of disks, so we doubled the value, as the number of disks is doubled in this case. Figure 3 (LUN 2) again shows the error between actual and computed latency values for various workload configurations. Note that the computed values based on the previous constants are fairly good at tracking the actual values. We again noticed that most of the high error cases were due to the poor prediction for corner cases, such as 100% sequential, 100% writes, etc.

To understand variation across different storage architectures, we ran a similar set of 750 tests on a NetApp FAS-3140 storage array. The experiments were run on a 256 GB virtual disk created on a 500 GB LUN backed by a 7-disk RAID-6 (double parity) group. Figures 4(a), (b), (c) and (d) show the relationship between average IO latency with OIOs, IO size, Read% and Random% respectively. Again for OIOs, IO size and Random%, we observed a linear behavior with positive slope. However, for the Read% case on the NetApp array, the slope was close to zero or slightly negative. We also found that the read latencies were very close to or slightly smaller than write latencies in most cases. We believe this is due to a small NVRAM cache in the array (512 MB). The writes are getting flushed to the disks in a synchronous manner and array is giving slight preference to reads over writes. We again modeled the system using Equation 1, calculated the K_i constants and computed the relative error in the measured and computed latencies using the NetApp measurements. Figure 3 (NetApp) shows the relative error for all 750 cases. We looked into the mapping of cases with high error with the actual configurations and noticed that almost all of those configurations are completely sequential workloads. This shows that our linear model over-predicts the latency for 100% sequential workloads because the linearity assumption doesn't hold in such extreme cases. Figures 2(d) and 4(d) also show a big drop in latency as we go from 25% random to 0% random. We looked at the relationship between IO latency and workload parameters for such extreme cases. Figure 5 shows that for sequential cases the relationship between IO latency and read% is not quite linear.

In practice, we think such cases are less common and poor prediction for such cases is not as critical. Earlier work in the area of workload characterization [12,13] confirms our experience. Most enterprise and web workloads that have been studied including Microsoft Exchange, a maps server, and TPC-C and TPC-E like workloads exhibit very little sequential accesses. The only notable workloads that have greater than 75% sequentiality are decision support systems.

Since K_5 is a device dependent parameter, we use the numerator of Equation 1 to represent the load metric ($\mathscr{L}$) for a workload. Based on our experience and empirical data, K_1, K_2, K_3 and K_4 lie in a narrow range even when measured across devices. This gives us a choice when applying our modeling on a real system: we can use a fixed set of values for the constants or recalibrate the model by computing the constants on a per-device basis in an offline manner when a device is first provisioned and added to the storage POD.

4 Storage Device Modeling

So far we have discussed the modeling of workloads based on the parameters that are inherent to a workload. In this section we present our device modeling technique using the measurements dependent on the performance of the device. Most of the device-level characteristics such

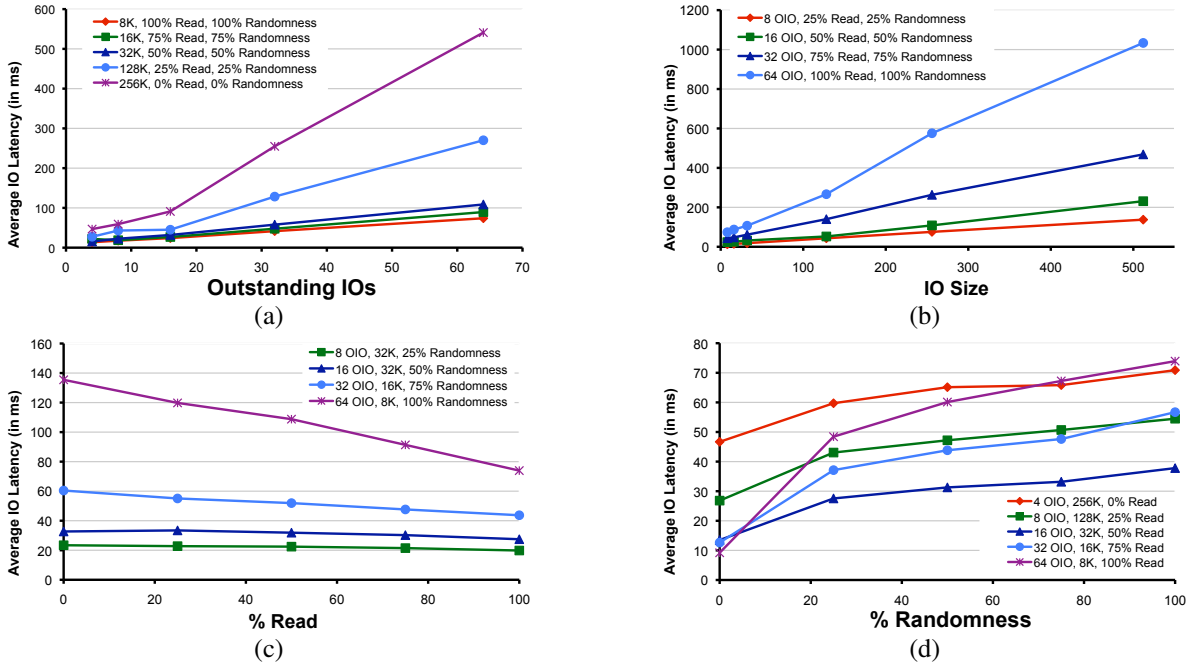

Figure 4: Variation of IO latency with respect to each of the four workload characteristics: outstanding IOs, IO size, % Reads and % Randomness. Experiments run on a 7-disk RAID-6 LUN on a NetApp FAS-3140 array.

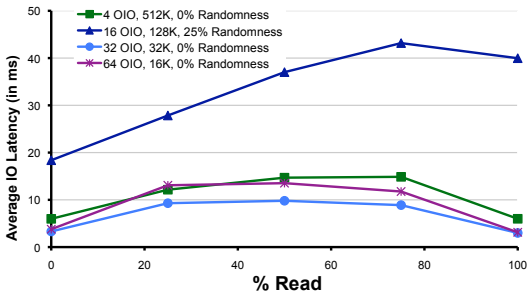

Figure 5: Varying Read% for the Anomalous Workloads

as number of disk spindles backing a LUN, disk-level features such as RPM, average seek delay, etc. are hidden from the hosts. Storage arrays only expose a LUN as a logical device. This makes it very hard to make load balancing decisions because we don't know if a workload is being moved from a LUN with 20 disks to a LUN with 5 disks, or from a LUN with faster Fibre Channel (FC) disk drives to a LUN with slower SATA drives.

For device modeling, instead of trying to obtain a white-box model of the LUNs, we use IO latency as the main performance metric. We collect information pairs consisting of number of outstanding IOs and average IO latency observed. In any time interval, hosts know the average number of outstanding IOs that are sent to a LUN and they also measure the average IO latency observed by the IOs. This information can be easily gathered using existing tools such as esxtop or xentop, without any extra overhead. For clustered environments, where multiple hosts access the same LUN, we aggregate this information across hosts to get a complete view.

We have observed that IO latency increases linearly with the increase in number of outstanding IOs (*i.e.*, load) on the array. This is also shown in earlier studies [11]. Given this knowledge, we use the set of data points of the form $\langle OIO, Latency \rangle$ over a period of time and compute a linear fit which minimizes the least squares error for the data points. The slope of the resulting line would indicate the overall performance capability of the LUN. We believe that this should cover cases where LUNs have different number of disks and where disks have diverse characteristics, *e.g.*, enterprise-class FC vs SATA disks.

We conducted a simple experiment using LUNs with different number of disks and measured the slope of the linear fit line. An illustrative workload of 8KB random IOs is run on each of the LUNs using a Windows 2003 VM running Iometer [3]. Figure 6 shows the variation of IO latency with OIOs for LUNs with 4 to 16 disks. Note that the slopes vary inversely with the number of disks.

To understand the behavior in presence of different disk types, we ran an experiment on a NetApp FAS-3140 storage array using two LUNs, each with seven disks and dual parity RAID. LUN1 consisted of enterprise class FC disks (134 GB each) and LUN2 consisted of slower SATA disks (414 GB each). We created virtual disks of size 256 GB on each of the LUNs and ran a workload

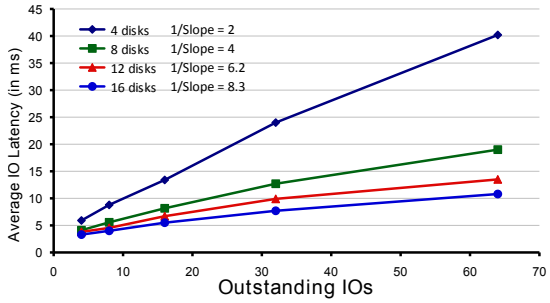

Figure 6: Device Modeling: different number of disks

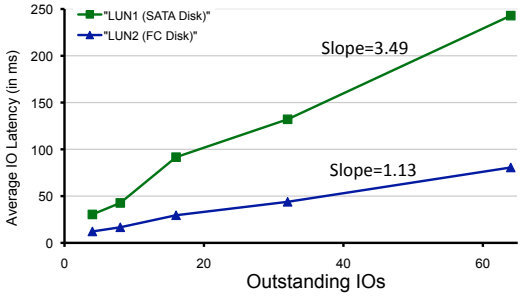

Figure 7: Device Modeling: different disk types

with 80% reads, 70% randomness and 16KB IOs, with different values of OIOs. The workloads were generated using Iometer [3] inside a Windows 2003 VM. Figure 7 shows the average latency observed for these two LUNs with respect to OIOs. Note that the slope for LUN1 with faster disks is 1.13, which is lower compared to the slope of 3.5 for LUN2 with slower disks.

This data shows that the performance of a LUN can be estimated by looking at the slope of relationship between average latency and outstanding IOs over a long time interval. Based on these results, we define a performance parameter $\mathscr{P}$ to be the inverse of the slope obtained by computing a linear fit on the $\langle OIO, Latency \rangle$ data pairs collected for that LUN.

4.1 Storage-specific Challenges

Storage devices are stateful, and IO latencies observed are dependent on the actual workload going to the LUN. For example, writes and sequential IOs may have very different latencies compared to reads and random IOs, respectively. This can create problems for device modeling if the IO behavior is different for various OIO values. We observed this behavior while experimenting with the DVD Store [1] database test suite, which represents a complete online e-commerce application running on SQL databases. The setup consisted of one database LUN and one log LUN, of sizes 250 GB and 10 GB respectively. Figure 8 shows the distribution of OIO and latency pairs for a 30 minute run of DVD Store. Note that the slope

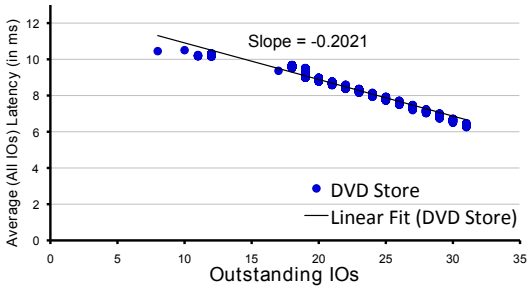

Figure 8: Negative slope in case of running DVD Store workload on a LUN. This happens due to a large number of writes happening during periods of high OIOs.

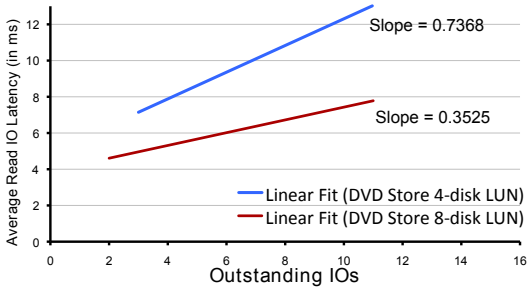

Figure 9: This plot shows the slopes for two data stores, both running DVD Store. Writes are filtered out in the model. The slopes are positive here and the slope value is lower for the 8 disk LUN.

turned out to be slightly negative, which is not desirable for modeling. Upon investigation, we found that the data points with larger OIO values were bursty writes that have smaller latencies because of write caching at the array.

Similar anomalies can happen for other cases: (1) Sequential IOs: the slope can be negative if IOs are highly sequential during the periods of large OIOs and random for smaller OIO values. (2) Large IO sizes: the slope can be negative if the IO sizes are large during the period of low OIOs and small during high OIO periods. All these workload-specific details and extreme cases can adversely impact the workload model.

In order to mitigate this issue, we made two modifications to our model: first, we consider only read OIOs and average read latencies. This ensures that cached writes are not going to affect the overall device model. Second, we ignore data points where an extreme behavior is detected in terms of average IO size and sequentiality. In our current prototype, we ignore data points when IO size is greater than 32 KB or sequentiality is more than 90%. In the future, we plan to study normalizing latency by IO size instead of ignoring such data points. In practice, this isn't a big problem because (a) with virtualization, single LUNs typically host VMs with numerous different workload types, (b) we expect to collect data for each LUN

over a period of days in order to make migration decisions, which allows IO from various VMs to be included in our results and (c) even if a single VM workload is sequential, the overall IO pattern arriving at the array may look random due to high consolidation ratios typical in virtualized systems.

With these provisions in place, we used DVD Store again to perform device modeling and looked at the slope values for two different LUNs with 4 and 8 disks. Figure 9 shows the slope values for the two LUNs. Note that the slopes are positive for both LUNs and the slope is lower for the LUN with more disks.

Cache size available to a LUN can also impact the overall IO performance. The first order impact should be captured by the IO latency seen by a workload. In some experiments, we observed that the slope was smaller for LUNs on an array with a larger cache, even if other characteristics were similar. Next, we complete the algorithm by showing how the workload and device models are used for dynamic load balancing and initial placement of virtual disks on LUNs.

5 Load Balance Engine

Load balancing requires a metric to balance over multiple resources. We use the numerator of Equation 1 (denoted as $\mathscr{L}_i$), as the main metric for load balancing for each workload W_i. Furthermore, we also need to consider LUN performance while doing load balancing. We use parameter $\mathscr{P}_j$ to represent the performance of device D_j. Intuitively we want to make the load proportional to the performance of each device. So the problem reduces to equalizing the ratio of the sum of workload metrics and the LUN performance metric for each LUN. Mathematically, we want to equate the following across devices:

$$\frac{\sum_{\forall\ W_i\ on\ D_j} \mathscr{L}_i}{\mathscr{P}_j} \tag{4}$$

The algorithm first computes the sum of workload metrics. Let N be the normalized load on a device:

$$N_j = \frac{\sum \mathscr{L}_i}{\mathscr{P}_j} \tag{5}$$

Let $Avg(\{N\})$ and $\sigma(\{N\})$ be the average and standard deviation of the normalized load across devices. Let the imbalance fraction f be defined as $f(\{N\}) = \sigma(\{N\})/Avg(\{N\})$. In a loop, until we get the imbalance fraction $f(\{N\})$ under a threshold, we pick the devices with minimum and maximum normalized load to do pairwise migrations such that the imbalance is lowered with each move. Each iteration of the loop tries to find the virtual disks that need to be moved from the device with maximum normalized load to the one with the minimum normalized load. Perfect balancing between these two devices is a variant of subset-sum problem which is known to be NP-complete. We are using one of the approximations [16] proposed for this problem with a quite good competitive ratio of 3/4 with respect to optimal. We have tested other heuristics as well, but the gain from trying to reach the best balance is outweighed by the cost of migrations in some cases.

Algorithm 1: Load Balancing Step

```
foreach device D_j do
    foreach workload W_i currently placed D_j do
        S+ = L_i
    N_j ⟵ S/P_j
while f({N}) > imbalanceThreshold do
    d_x ⟵ Device with maximum normalized load
    d_y ⟵ Device with minimum normalized load
    N_x, N_y ⟵ PairWiseRecommendMigration(d_x, d_y)
```

Algorithm 1 presents the pseudo-code for the load balancing algorithm. The imbalance threshold can be used to control the tolerated degree of imbalance in the system and therefore the aggressiveness of the algorithm. Optimizations in terms of data movement and cost of migrations are explained next.

Workload/Virtual Disk Selection: To refine the recommendations, we propose biasing the choice of migration candidates in one of many ways: (1) pick virtual disks with the highest value of $\mathscr{L}_i/(disk\ size)$ first, so that the change in load per GB of data movement is higher leading to smaller data movement, (2) pick virtual disks with smallest current IOPS/$\mathscr{L}_i$ first, so that the immediate impact of data movement is minimal, (3) filter for constraints such as affinity between virtual disks and data stores, (4) avoid ping-ponging of the same virtual disk between data stores, (5) prevent migration movements that violate per-VM data reliability or data protection policies (*e.g.*, RAID-level), etc. Hard constraints (*e.g.*, access to the destination data store at the current host running the VM) can also be handled as part of virtual disk selection in this step. Overall, this step incorporates any cost-benefit analysis that is needed to choose which VMs to migrate in order to do load balancing. After computing these recommendations, they can either be presented to the user as suggestions or can be carried out automatically during periods of low activity. Administrators can even configure the times when the migrations should be carried out, *e.g.*, migrate on Saturday nights after 2am.

Initial Placement: A good decision for the initial placement of a workload is as important as future migrations. Initial placement gives us a good way to reduce potential imbalance issues in future. In BASIL, we use the over-

all normalized load N as an indicator of current load on a LUN. After resolving user-specified hard constraints (*e.g.*, reliability), we choose the LUN with the minimum value of the normalized load for a new virtual disk. This ensures that with each initial placement, we are attempting to naturally reduce the overall load imbalance among LUNs.

Discussion: In previous work [12], we looked at the impact of consolidation on various kinds of workloads. We observed that when random workloads and the underlying devices are consolidated, they tend to perform at least as good or better in terms of handling bursts and the overall impact of interference is very small. However, when random and sequential workloads were placed together, we saw degradation in throughput of sequential workloads. As noted in Section 3, studies [12, 13] of several enterprise applications such as Microsoft Exchange and databases have observed that random access IO patterns are the predominant type.

Nevertheless, to handle specific workloads such as log virtual disks, decision support systems, and multi-media servers, we plan to incorporate two optimizations. First, identifying such cases and isolating them on a separate set of spindles to reduce interference. Second, allocating fewer disks to the sequential workloads because their performance is less dependent on the number of disks as compared to random ones. This can be done by setting soft affinity for these workloads to specific LUNs, and anti-affinity for them against random ones. Thus we can bias our greedy load balancing heuristic to consider such affinity rules while making placement decisions.

Whereas we consider these optimizations as part of our future work, we believe that the proposed techniques are useful for a wide variety of cases, even in their current form, since in some cases, administrators may isolate such workloads on separate LUNs manually and set hard affinity rules. We can also assist storage administrators by identifying such workloads based on our online data collection. In some cases users may have reliability or other policy constraints such as RAID-level or mirroring, attached to VM disks. In those cases a set of devices would be unsuitable for some VMs, and we would treat that as a hard constraint in our load balancing mechanism while recommending placements and migrations. Essentially the migrations would occur among devices with similar static characteristics. The administrator can choose the set of static characteristics that are used for combining devices into a single storage POD (our load balancing domain). Some of these may be reliabilitity, backup frequency, support for de-duplication, thin provisioning, security isolation and so on.

Type	OIO range	IO size	%Read	%Random
Workstation	[4-12]	8	80	80
Exchange	[4-16]	4	67	100
OLTP	[12-16]	8	70	100
Webserver	[1-4]	4	95	75

Table 1: Iometer workload configuration definitions.

6 Experimental Evaluation

In this section we discuss experimental results based on an extensive evaluation of BASIL in a real testbed. The metrics that we use for evaluating BASIL are overall throughput gain and overall latency reduction. Here overall throughput is aggregated across all data stores and overall latency is the average latency weighted by IOPS across all data stores. These metrics are used instead of just individual data store values, because a change at one data store may lead to an inverse change on another, and our goal is to improve the overall performance and utilization of the system, and not just individual data stores.

6.1 Testing Framework

Since the performance of a storage device depends greatly on the type of workloads to which it is subjected, and their interference, it would be hard to reason about a load balancing scheme with just a few representative test cases. One can always argue that the testing is too limited. Furthermore, once we make a change in the modeling techniques or load balancing algorithm, we will need to validate and compare the performance with the previous versions. To enable repeatable, extensive and quick evaluation of BASIL, we implemented a testing framework emulating a real data center environment, although at a smaller scale. Our framework consists of a set of hosts, each running multiple VMs. All the hosts have access to all the data stores in the load balancing domain. This connectivity requirement is critical to ensure that we don't have to worry about physical constraints during our testing. In practice, connectivity can be treated as another migration constraint. Our testing framework has three modules: admin, modeler and analyzer that we describe in detail next.

Admin module: This module initiates the workloads in each VM, starts collecting periodic IO stats from all hosts and feeds the stats to the next module for generation of workload and device models. The IO stats are collected per virtual disk. The granularity of sampling is configurable and set to 2-10 seconds for experiments in this paper. Finally, this module is also responsible for applying migrations that are recommended by the analyzer. In order to speed up the testing, we emulate the migrations by shifting the workload from one data store to another, instead of actually doing data migration. This is possible because we create an identical copy of each virtual disk

		Before Running BASIL			After Running BASIL		
Iometer Workload	**BASIL Online Workload Model [OIO, IOsize, Read%, Random%]**	**Latency (ms)**	**Throughput (IOPS)**	**Location**	**Latency (ms)**	**Throughput (IOPS)**	**Location**
oltp	[7, 8, 70, 100]	28	618	3diskLUN	22	1048	3diskLUN
oltp	[16, 8, 69, 100]	35	516	3diskLUN	12	1643	9diskLUN
workstation	[6, 8, 81, 79]	60	129	3diskLUN	24	338	9diskLUN
exchange	[6, 4, 67, 100]	9	940	6diskLUN	9	964	6diskLUN
exchange	[6, 4, 67, 100]	11	777	6diskLUN	8	991	6diskLUN
workstation	[4, 8, 80, 79]	13	538	6diskLUN	21	487	9diskLUN
webserver	[1, 4, 95, 74]	4	327	9diskLUN	29	79	9diskLUN
webserver	[1, 4, 95, 75]	4	327	9diskLUN	45	81	9diskLUN
Weighted Average Latency or Total Throughput		16.7	4172		14.9 (-11%)	5631 (+35%)	

Table 2: BASIL online workload model and recommended migrations for a sample initial configuration. Overall average latency and IO throughput improved after migrations.

			Before BASIL		After BASIL	
Data Stores	**# Disks**	$\mathscr{P}$ **=*I/Slope***	**Latency (ms)**	**IOPS**	**Latency (ms)**	**IOPS**
3diskLUN	3	0.7	34	1263	22	1048
6diskLUN	6	1.4	10	2255	8	1955
9diskLUN	9	2.0	4	654	16	2628

Table 3: BASIL online device model and disk migrations for a sample initial configuration. Latency, IOPS and overall load on three data stores before and after recommended migrations.

on all data stores, so a VM can just start accessing the virtual disk on the destination data store instead of the source one. This helped to reduce our experimental cycle from weeks to days.

Modeler: This module gets the raw stats from the admin module and creates both workload and device models. The workload models are generated by using per virtual disk stats. The module computes the cumulative distribution of all four parameters: OIOs, IO size, Read% and Random%. To compute the workload load metric $\mathscr{L}_i$, we use the 90th percentile values of these parameters. We didn't choose average values because storage workloads tend to be bursty and the averages can be much lower and more variable compared to the 90th percentile values. We want the migration decision to be effective in most cases instead of just average case scenarios. Since migrations can take hours to finish, we want the decision to be more conservative rather than aggressive.

For the device models, we aggregate IO stats from different hosts that may be accessing the same device (*e.g.*, using a cluster file system). This is very common in virtualized environments. The OIO values are aggregated as a sum, and the latency value is computed as a weighted average using IOPS as the weight in that interval. The $\langle OIO, Latency \rangle$ pairs are collected over a long period of time to get higher accuracy. Based on these values, the modeler computes a slope $\mathscr{P}_i$ for each device. A device with no data, is assigned a slope of zero which also mimics the introduction of a new device in the POD.

Analyzer: This module takes all the workload and device models as input and generates migration recommendations. It can also be invoked to perform initial placement of a new virtual disk based on the current configuration. The output of the analyzer is fed into the admin module to carry out the recommendations. This can be done iteratively till the load imbalance is corrected and the system stabilizes with no more recommendations generated.

The experiments presented in the next sections are run on two different servers, one configured with 2 dual-core 3 GHz CPUs, 8 GB RAM and the other with 4 dual-core 3 GHz CPUs and 32 GB RAM. Both hosts have access to three data stores with 3, 6 and 9 disks over a FC SAN network. These data stores are 150 GB in size and are created on an EMC CLARiiON storage array. We ran 8 VMs for our experiments each with one 15 GB OS disk and one 10 GB experimental disk. The workloads in the VMs are generated using Iometer [3]. The Iometer workload types are selected from Table 1, which shows Iometer configurations that closely represent some of the real enterprise workloads [5].

6.2 Simple Load Balancing Scenario

In this section, we present detailed analysis for one of the input cases which looks balanced in terms of number of VMs per data store. Later, we'll also show data for a large number of other scenarios. As shown in Table 2, we started with an initial configuration using 8 VMs, each running a workload chosen from Table 1 against one of the three data stores. First we ran the workloads in VMs without BASIL; Table 2 shows the corresponding throughput (IOPS) and latency values seen by the workloads. Then we ran BASIL, which created workload and device models online. The computed workload model is shown in the second column of Table 2 and device model is shown as $\mathscr{P}$ (third column) in Table 3. It is worth noting that the computed performance metrics for

Iometer Workload	BASIL Online Workload Model [OIO, IOsize, Read%, Random%]	Before Running BASIL Latency (ms)	Throughput (IOPS)	Location	After Running BASIL Latency (ms)	Throughput (IOPS)	Location
exchange	[8, 4, 67, 100]	37	234	6diskLUN	62	156	6diskLUN
exchange	[8, 4, 67, 100]	39	227	6diskLUN	12	710	3diskLUN
webserver	[2, 4, 95, 75]	54	43	6diskLUN	15	158	9diskLUN
webserver	[2, 4, 95, 75]	60	39	6diskLUN	18	133	9diskLUN
workstation	[7, 8, 80, 80]	41	191	6diskLUN	11	657	9diskLUN
workstation	[8, 8, 80, 80]	51	150	6diskLUN	11	686	9diskLUN
oltp	[8, 8, 70, 100]	64	402	6diskLUN	28	661	6diskLUN
oltp	[8, 8, 70, 100]	59	410	6diskLUN	28	658	6diskLUN
Weighted Average Latency or Total Throughput		51.6	1696		19.5 (-62%)	3819 (+125%)	

Table 4: New device provisioning: 3DiskLUN and 9DiskLUN are newly added into the system that had 8 workloads running on the 6DiskLUN. Average latency, IO throughput and placement for all 8 workloads before and after migration.

Data Stores	# Disks	$\mathscr{P} = 1/Slope$	Before BASIL Latency (ms)	Before BASIL IOPS	After BASIL Latency (ms)	After BASIL IOPS
3diskLUN	3	0.6	0	0	12	710
6diskLUN	6	1.4	51	1696	31	1475
9diskLUN	9	1.7	0	0	11	1634

Table 5: New device provisioning: latency, IOPS and overall load on three data stores.

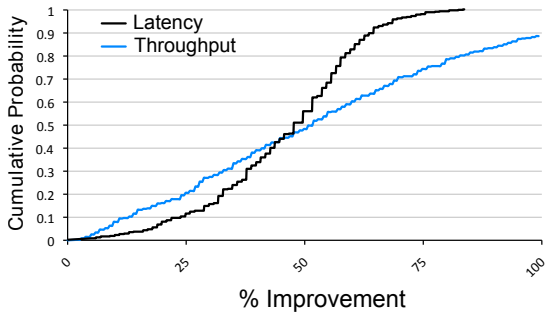

Figure 10: CDF of throughput and latency improvements with load balancing, starting from random configurations.

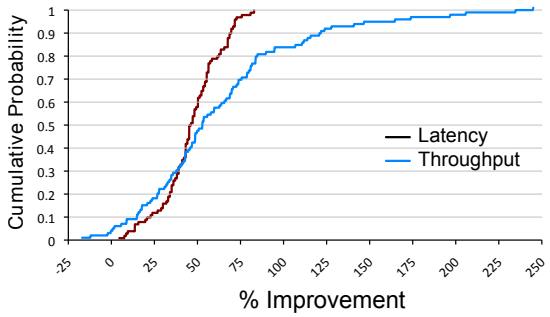

Figure 11: CDF of latency and throughput improvements from BASIL initial placement versus random.

devices are proportional to their number of disks. Based on the modeling, BASIL suggested three migrations over two rounds. After performing the set of migrations we again ran BASIL and no further recommendations were suggested. Tables 2 and 3 show the performance of workloads and data stores in the final configuration. Note that 5 out of 8 workloads observed an improvement in IOPS and reduction in latency. The aggregated IOPS across all data stores (shown in Table 2) improved by 35% and overall weighted latency decreased by 11%. This shows that for this sample setup BASIL is able to recommend migrations based on actual workload characteristics and device modeling, thereby improving the overall utilization and performance.

6.3 New Device Provisioning

Next we studied the behavior of BASIL during the well known operation of adding more storage devices to a storage POD. This is typically in response to a space crunch or a performance bottleneck. In this experiment, we started with all VMs on the single 6DiskLUN data store and we added the other two LUNs into the system. In the first round, BASIL observed the two new data stores, but didn't have any device model for them due to lack of IOs. In a full implementation, we have the option of performing some offline modeling at the time of provisioning, but currently we use the heuristic of placing only one workload on a new data store with no model.

Table 4 shows the eight workloads, their computed models, initial placement and the observed IOPS and latency values. BASIL recommended five migrations over two rounds. In the first round BASIL migrated one workload to each of 3DiskLUN and 9DiskLUN. In the next round, BASIL had slope information for all three data stores and it migrated three more workloads from 6DiskLUN to 9DiskLUN. The final placement along with performance results are again shown in Table 4. Seven out of eight workloads observed gains in throughput and decreased latencies. The loss in one workload is offset by gains in others on the same data store. We believe

that this loss happened due to unfair IO scheduling of LUN resources at the storage array. Such effects have been observed before [11]. Overall data store models and performance before and after running BASIL are shown in Table 5. Note that the load is evenly distributed across data stores in proportion to their performance. In the end, we observed a 125% gain in aggregated IOPS and 62% decrease in weighted average latency (Table 4). This shows that BASIL can handle provisioning of new storage devices well by quickly performing online modeling and recommending appropriate migrations to get higher utilization and better performance from the system.

6.4 Summary for 500 Configurations

Having looked at BASIL for individual test cases, we ran it for a large set of randomly generated initial configurations. In this section, we present a summary of results of over 500 different configurations. Each test case involved a random selection of 8 workloads from the set shown in Table 1, and a random initial placement of them on three data stores. Then in a loop we collected all the statistics in terms of IOPS and latency, performed online modeling, ran the load balancer and performed workload migrations. This was repeated until no further migrations were recommended. We observed that all configurations showed an increase in overall IOPS and decrease in overall latency. There were fluctuations in the performance of individual workloads, but that is expected given that load balancing puts extra load on some data stores and reduces load on others. Figure 10 shows the cumulative distribution of gain in IOPS and reduction in latency for 500 different runs. We observed an overall throughput increase of greater than 25% and latency reduction of 33% in over 80% of all the configurations that we ran. In fact, approximately half the tests cases saw at least 50% higher throughput and 50% better latency. This is very promising as it shows that BASIL can work well for a wide range of workload combinations and their placements.

6.5 Initial Placement

One of the main use cases of BASIL is to recommend initial placement for new virtual disks. Good initial placement can greatly reduce the number of future migrations and provide better performance from the start. We evaluated our initial placement mechanism using two sets of tests. In the first set we started with one virtual disk, placed randomly. Then in each iteration we added one more disk into the system. To place the new disk, we used the current performance statistics and recommendations generated by BASIL. No migrations were computed by BASIL; it ran only to suggest initial placement.

Workload	BASIL Online Workload Model [OIO, IOsize, Read%, Random%]
dvdstore-1	[5, 8, 100, 100]
dvdstore-2	[3, 62, 100, 100]
dvdstore-3	[6, 8, 86, 100]
swing-1	[13, 16, 67, 100]
swing-2	[31, 121, 65, 100]
fb-mail-1	[4, 5, 16, 99]
fb-mail-2	[5, 6, 52, 99]
fb-mail-3	[7, 6, 47, 99]
fb-mail-4	[5, 5, 60, 99]
fb-oltp-1	[1, 2, 100, 100]
fb-oltp-2	[6, 8, 86, 100]
fb-web-1	[8, 18, 99, 98]
fb-web-2	[5, 5, 60, 99]

Table 6: Enterprise workloads. For the database VMs, only the table space and index disks were modeled.

Data Stores	# Disks	RAID	LUN Size	$\mathscr{P}$ =*1/Slope*
EMC	6 FC	5	450 GB	1.1
NetApp-SP	7 FC	5	400 GB	0.83
NetApp-DP	7 SATA	6	250 GB	0.48

Table 7: Enterprise workload LUNs and their models.

We compared the performance of placement done by BASIL with a random placement of virtual disks as long as space constraints were satisfied. In both cases, the VMs were running the exact same workloads. We ran 100 such cases, and Figure 11 shows the cumulative distribution of percentage gain in overall throughput and reduction in overall latency of BASIL as compared to random selection. This shows that the placement recommended by BASIL provided 45% reduction in latency and 53% increase in IOPS for at least half of the cases, as compared to the random placement.

The second set of tests compare BASIL with an oracle that can predict the best placement for the next virtual disk. To test this, we started with an initial configuration of 7 virtual disks that were randomly chosen and placed. We ran this configuration and fed the data to BASIL to find a data store for the eighth disk. We tried the eighth disk on all the data stores manually and compared the performance of BASIL's recommendation with the best possible placement. To compute the rank of BASIL compared to the oracle, we ran 194 such cases and BASIL chose the best data store in 68% of them. This indicates that BASIL finds good initial placements with high accuracy for a wide variety of workload configurations.

6.6 Enterprise Workloads

In addition to the extensive micro-benchmark evaluation, we also ran enterprise applications and filebench workload models to evaluate BASIL in more realistic scenarios. The CPU was not bottlenecked in any of the experiments. For the database workloads, we isolated the data and log virtual disks. Virtual disks containing data

Workload	T	Space-Balanced			After Two BASIL Rounds			Human Expert #1			Human Expert #2		
	Units	R	T	Location	R	T	Location	R	T	Location	R	T	Location
dvd-1	opm	72	2753	EMC	78	2654	EMC	59	2986	EMC	68	2826	NetApp-SP
dvd-2	opm	82	1535	NetApp-SP	89	1487	*EMC*	58	1706	EMC	96	1446	EMC
dvd-3	opm	154	1692	NetApp-DP	68	2237	*NetApp-SP*	128	1821	NetApp-DP	78	2140	EMC
swing-1	tpm	n/r	8150	NetApp-SP	n/r	8250	NetApp-SP	n/r	7500	NetApp-DP	n/r	7480	NetApp-SP
swing-2	tpm	n/r	8650	EMC	n/r	8870	EMC	n/r	8950	EMC	n/r	8500	NetApp-DP
fb-mail-1	ops/s	38	60	NetApp-SP	36	63	NetApp-SP	35	61	NetApp-SP	15	63	EMC
fb-mail-2	ops/s	35	84	NetApp-SP	37	88	NetApp-SP	34	85	NetApp-SP	16	88	EMC
fb-mail-3	ops/s	81	67	NetApp-DP	27	69	NetApp-DP	30	73	NetApp-SP	28	74	NetApp-SP
fb-mail-4	ops/s	9.2	77	EMC	14	75	EMC	11	76	EMC	16	75	EMC
fb-oltp-1	ops/s	32	25	NetApp-SP	35	25	NetApp-SP	70	24	NetApp-DP	44	25	NetApp-DP
fb-oltp-2	ops/s	84	22	NetApp-DP	40	22	NetApp-DP	79	22	NetApp-DP	30	23	NetApp-SP
fb-web-1	ops/s	58	454	NetApp-DP	26	462	*NetApp-SP*	56	460	NetApp-DP	22	597	EMC
fb-web-2	ops/s	11	550	EMC	11	550	EMC	21	500	NetApp-SP	14	534	EMC

Table 8: Enterprise Workloads. Human expert generated placements versus BASIL. Applying BASIL recommendations resulted in improved application as well as more balanced latencies. *R* denotes application-reported transaction response time (ms) and *T* is the throughput in specified units.

	Space-Balanced		After Two BASIL Rounds		Human Expert #1		Human Expert #2	
	Latency (ms)	IOPS	Latency (ms)	IOPS	Latency (ms)	IOPS	Latency (ms)	IOPS
EMC	9.6	836	12	988	9.9	872	14	781
NetApp-SP	29	551	19	790	27	728	26	588
NetApp-DP	45	412	23	101	40	317	17	340
Weighted Average Latency or Total Throughput	23.6	1799	15.5	1874	21.2	1917	18.9	1709

Table 9: Enterprise Workloads. Aggregate statistics on three LUNs for BASIL and human expert placements.

were placed on the LUNs under test and log disks were placed on a separate LUN. We used five workload types as explained below.

DVDStore [1] version 2.0 is an online e-commerce test application with a SQL database, and a client load generator. We used a 20 GB dataset size for this benchmark, 10 user threads and 150 ms think time between transactions.

Swingbench [4] (order entry workload) represents an online transaction processing application designed to stress an underlying Oracle database. It takes the number of users, think time between transactions, and a set of transactions as input to generate a workload. For this workload, we used 50 users, 100-200 ms think time between requests and all five transaction types (*i.e.*, new customer registration, browse products, order products, process orders and browse orders with variable percentages set to 10%, 28%, 28%, 6% and 28% respectively).

Filebench [2], a well-known application IO modeling tool, was used to generate three different types of workloads: OLTP, mail server and webserver.

We built 13 VMs running different configurations of the above workloads as shown in Table 6 and ran them on two quad-core servers with 3 GHz CPUs and 16 GB RAM. Both hosts had access to three LUNs with different characteristics, as shown in Table 7. To evaluate BASIL's performance, we requested domain experts within VMware to pick their own placements using full knowledge of workload characteristics and detailed knowledge of the underlying storage arrays. We requested two types of configurations: space-balanced and performance-balanced.

The space-balanced configuration was used as a baseline and we ran BASIL on top of that. BASIL recommended three moves over two rounds. Table 8 provides the results in terms of the application-reported transaction latency and throughput in both configurations. In this instance, the naive space-balanced configuration had placed similar load on the less capable data stores as on the faster ones causing VMs on the former to suffer from higher latencies. BASIL recommended moves from less capable LUNs to more capable ones, thus balancing out application-visible latencies. This is a key component of our algorithm. For example, before the moves, the three DVDStore VMs were seeing latencies of 72 ms, 82 ms and 154 ms whereas a more balanced result was seen afterward: 78 ms, 89 ms and 68 ms. Filebench OLTP workloads had a distribution of 32 ms and 84 ms before versus 35 ms and 40 ms afterward. Swingbench didn't report latency data but judging from the throughput, both VMs were well balanced before and BASIL didn't change that. The Filebench webserver and mail VMs also had much reduced variance in latencies. Even compared to the two expert placement results, BASIL fares better in terms of variance. This demonstrates the ability of BASIL to balance real enterprise workloads across data stores of very different capabilities using online models.

BASIL also performed well in the critical metrics of maintaining overall storage array efficiency while balanc-

ing load. Table 9 shows the achieved device IO latency and IO throughput for the LUNs. Notice that, in comparison to the space-balanced placement, the weighted average latency across three LUNs went down from 23.6 ms to 15.5 ms, a gain of 34%, while IOPS increased slightly by 4% from 1799 to 1874. BASIL fared well even against hand placement by domain experts. Against expert #2, BASIL achieved an impressive 18% better latency and 10% better throughput. Compared to expert #1, BASIL achieved a better weighted average latency by 27% albeit with 2% less throughput. Since latency is of primary importance to enterprise workloads, we believe this is a reasonable trade off.

7 Conclusions and Future Work

This paper presented BASIL, a storage management system that does initial placement and IO load balancing of workloads across a set of storage devices. BASIL is novel in two key ways: (1) identifying IO latency as the primary metric for modeling, and (2) using simple models both for workloads and devices that can be efficiently obtained online. The linear relationship of IO latency with various parameters such as outstanding IOs, IO size, read % etc. is used to create models. Based on these models, the load balancing engine recommends migrations in order to balance load on devices in proportion to their capabilities.

Our extensive evaluation in a real system with multiple LUNs and workloads shows that BASIL achieved improvements of at least 25% in throughput and 33% in overall latency in over 80% of the hundreds of micro-benchmark configurations that we tested. Furthermore, for real enterprise applications, BASIL lowered the variance of latencies across the workloads and improved the weighted average latency by 18-27% with similar or better achieved throughput when evaluated against configurations generated by human experts.

So far we've focused on the quality of the BASIL recommended moves. As future work, we plan to add migration cost considerations into the algorithm and more closely study convergence properties. Also on our roadmap is special handling of the less common sequential workloads, as well as applying standard techniques for ping-pong avoidance. We are also looking at using automatically-generated affinity and anti-affinity rules to minimize the interference among various workloads accessing a device.

Acknowledgments

We would like to thank our shepherd Kaladhar Voruganti for his support and valuable feedback. We are grateful to Carl Waldspurger, Minwen Ji, Ganesha Shanmuganathan, Anne Holler and Neeraj Goyal for valuable discussions and feedback. Thanks also to Keerti Garg, Roopali Sharma, Mateen Ahmad, Jinpyo Kim, Sunil Satnur and members of the performance and resource management teams at VMware for their support.

References

[1] DVD Store. http://www.delltechcenter.com/page/DVD+store.

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

[3] Iometer. http://www.iometer.org.

[4] Swingbench. http://www.dominicgiles.com/swingbench.html.

[5] Workload configurations for typical enterprise workloads. http://blogs.msdn.com/tvoellm/archive/2009/05/07/useful-io-profiles-for-simulating-various-workloads.aspx.

[6] Resource Management with VMware DRS, 2006. http://vmware.com/pdf/vmware_drs_wp.pdf.

[7] AHMAD, I. Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server. *IISWC* (Sept. 2007).

[8] ALVAREZ, G. A., AND ET AL. Minerva: an automated resource provisioning tool for large-scale storage systems. In *ACM Transactions on Computer Systems* (Nov. 2001).

[9] ANDERSON, E. Simple table-based modeling of storage devices. Tech. rep., SSP Technical Report, HP Labs, July 2001.

[10] ANDERSON, E., AND ET AL. Hippodrome: running circles around storage administration. In *Proc. of Conf. on File and Storage Technology (FAST'02)* (Jan. 2002).

[11] GULATI, A., AHMAD, I., AND WALDSPURGER, C. PARDA: Proportionate Allocation of Resources for Distributed Storage Access. In *USENIX FAST* (Feb. 2009).

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

[13] KAVALANEKAR, S., WORTHINGTON, B., ZHANG, Q., AND SHARDA, V. Characterization of storage workload traces from production windows servers. In *IEEE IISWC* (Sept. 2008).

[14] MERCHANT, A., AND YU, P. S. Analytic modeling of clustered raid with mapping based on nearly random permutation. *IEEE Trans. Comput. 45*, 3 (1996).

[15] MESNIER, M. P., WACHS, M., SAMBASIVAN, R. R., ZHENG, A. X., AND GANGER, G. R. Modeling the relative fitness of storage. *SIGMETRICS Perform. Eval. Rev. 35*, 1 (2007).

[16] PRZYDATEK, B. A Fast Approximation Algorithm for the Subset-Sum Problem, 1999.

[17] RUEMMLER, C., AND WILKES, J. An introduction to disk drive modeling. *IEEE Computer 27*, 3 (1994).

[18] SHEN, Y.-L., AND XU, L. An efficient disk I/O characteristics collection method based on virtual machine technology. *10th IEEE Intl. Conf. on High Perf. Computing and Comm.* (2008).

[19] SHRIVER, E., MERCHANT, A., AND WILKES, J. An analytic behavior model for disk drives with readahead caches and request reordering. *SIGMETRICS Perform. Eval. Rev. 26*, 1 (1998).

[20] UYSAL, M., ALVAREZ, G. A., AND MERCHANT, A. A modular, analytical throughput model for modern disk arrays. In *MASCOTS* (2001).

[21] VARKI, E., MERCHANT, A., XU, J., AND QIU, X. Issues and challenges in the performance analysis of real disk arrays. *IEEE Trans. Parallel Distrib. Syst. 15*, 6 (2004).

[22] WANG, M., AU, K., AILAMAKI, A., BROCKWELL, A., FALOUTSOS, C., AND GANGER, G. R. Storage Device Performance Prediction with CART Models. In *MASCOTS* (2004).