Romano: Autonomous Storage Management using Performance Prediction in Multi-Tenant Datacenters

Nohhyun Park
University of Minnesota
parkx408@umn.edu

Irfan Ahmad
CloudPhysics, Inc.
irfan@cloudphysics.com

David J. Lilja
University of Minnesota
lilja@umn.edu

ABSTRACT

Workload consolidation is a key technique in reducing costs in virtualized datacenters. When considering storage consolidation, a key problem is the unpredictable performance behavior of consolidated workloads on a given storage system. In practice, this often forces system administrators to grossly overprovision storage to meet application demands. In this paper, we show that existing modeling techniques are inaccurate and ineffective in the face of heterogenous devices. We introduce *Romano*, a storage performance management system designed to optimize truly heterogeneous virtualized datacenters. At its core, Romano constructs and adapts approximate workload-specific performance models of storage devices automatically, along with prediction intervals. It then applies these models to allow highly efficient IO load balancing.

End-to-end experiments demonstrate that Romano reduces prediction error by 80% on average compared with existing techniques. The result is improved load balancing with lowered variance by 82% and reduced average and maximum latency observed across the storage systems by 52% and 78%, respectively.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance—*modeling and prediction*

General Terms

Algorithms, Design, Experimentation, Management, Measurement, Performance

Keywords

Storage, Virtualization, VM, QoS, Device, Modeling

1. INTRODUCTION

The cloud computing paradigm shift is built on virtualization. Most IT organizations and cloud service providers have deployed virtualized data centers in an effort to consolidate workloads, streamline management, reduce costs and increase utilization. Still, overall costs for *storage management* have remained stubbornly high. Over its lifetime, managing storage is often four times more expensive than its initial procurement [15]. 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 [20].

A key enabler for virtualization adoption is ability for a virtual machine (VM) to be efficiently migrated across compute hosts at run time [4, 26, 16]. Consequently, VM live migration has been utilized to perform active performance management in commercial products for some time [1]. Similarly, 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 [14].

SOCC'12, October 14-17, 2012, San Jose, CA USA

However, it is well-known that storage data associated with a VM is much harder to migrate compared to CPU and Memory state. Virtual disk migration time is typically a function of virtual disk size, size of the working set, source and destination storage state and capability, etc [14]. Migration can take anywhere from tens of minutes to a few hours depending on those parameters.

Given the high expense of data migration, intelligent and automatic storage performance management has been a relevant but difficult research problem [7, 8]. Virtualized environments can be extremely complex, with diverse and 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. Our goal of supporting heterogeneity is critical since data center operators want to keep the flexibility of buying the cheapest hardware available at any given time.

The key to automated management of storage lies in the ability to predict the performance of a workload on a given storage system. Without it, an experienced user must determine manually if migrating a virtual disk is beneficial, a complicated task. Previous work [7, 8] has relied on a heuristic model derived from empirical data.

When considering automated load balancing, the simplest approach would be to observe utilization of all data stores and offload a virtual disk from the most heavily loaded data store to the least heavily loaded data store.

There are several problems with this approach but the main problem is the unpredictability of the benefit of the moves. In fact, there exists no guarantee that the moves will even be beneficial. Storage systems do not react to different workloads in a homogeneous manner. A workload may stress one storage system significantly more than the other depending on the storage system configurations and optimizations.

The second problem is the workload interference. When multiple workloads are placed into a single storage systems, their effect on the storage system is not simply additive. While some workloads can be placed onto a single storage system and work with minimum interference, some experience huge performance hits.

Romano prediction model takes into account this heterogeneous response of storage systems and the interference between workloads improving the prediction accuracy by over 80%. Once these problems are recognized, it is obvious that the load balancing is no longer a bin packing problem as suggested by previous works [7, 8, 21]. Therefore, we introduce a new load balancing algorithm based on simulated annealing [10] that optimizes the overall performance in a stochastic manner. We show that Romano is capable of reducing the performance variance by 82% and reduce the maximum latency observed by 78%.

Romano makes following contributions.

Accuracy Romano residuals are reduced 80% on average compared to previous techniques. Furthermore, the residual is unbiased and does not propagate with recursive predictions.

Robustness Romano is able to specify the prediction interval which satisfies a specified confidence level. We show that the resulting prediction interval captures real performance with surprising accuracy.

Flexibility Different storage systems have different response characteristics to different workloads. Romano captures these differences by quantifying the effect of workload characteristics as well as the interactions of those characteristics on a given storage system.

Optimization Romano load-balancing algorithm avoids having system state stuck in a local optimum, instead identifying a global pseudo-optimal state through recursive prediction. At the same time, it also ensures that the most beneficial migrations are performed first to maximize the benefit.

We have prototyped Romano on VMware's ESXi and vCenter Server products. Minimal changes were made to the ESXi hypervisor to collect additional stats for vCenter where the core of Romano is implemented.

The next section describes the most relevant prior work upon which our work improves. Section 3 introduces linear regression and our modeling technique. Romano system design is added in Section 4 followed by evaluation results in Section 5. Section 6 lists future works and we conclude in Section 7.

2. RELATED WORK

Our experiments have revealed that storage workload and storage device models are highly inter-dependent. This confirms observations from other works [25, 17].

Early efforts to automatically manage storage resources in a virtualized environment such as *VectorDot* [21] ignored the workload and simply concentrated on utilization of each storage system. A workload from the over-loaded device is simply migrated to a less loaded device. This approach would fail to work in a heterogeneous storage environment since the resulting performance is unknown. Furthermore, the goal is not to simply ensure that no storage system suffers from over-utilization but to minimize the performance degradation caused by resource sharing across all workloads.

Other work has focused on highly accurate latency prediction of workloads across storage devices but this has come at the cost of practicality by requiring extensive off-line experimentation and modeling. For instance, Wang et al. [25] developed CART models off-line to develop workload-storage models. In contrast to their work, Romano aims to be learn storage behavior quickly using online calibration without resorting to intrusive off-line methods.

Earlier work in this area include Basil [7] and Pesto [8] which propose creating separate workload and device models and using heuristics to perform load balancing. The underlying assumption is the independence of the two classes of models. A commercial product [1] based on Pesto is now available. Whereas Basil and Pesto have very good average behavior, they suffer from severe corner case model inaccuracies. For example, it was shown in their work that the prediction error can be more than 10 times larger than the latency it predicts for sequential workloads. These can lead to bad data movement recommendations. To address these shortcomings, Pesto applies aggressive cost-benefit analysis to prune out potentially bad moves. The inadvertent result of this pruning to deal with model inaccuracies is that many highly beneficial moves are also left out.

Romano proposes to deal with these problems by creating a model capable of determining its own confidence of prediction using *prediction interval* [3]. Furthermore, Romano is capable of quantifying the effect of the workload characteristics and their interactions for a given data store resulting in much more accurate and robust modeling.

3. BACKGROUND

This section introduces two standard statistics concepts used in Romano models (experts can skip).

3.1 Analysis of Variance(ANOVA)

ANOVA is a widely used technique to separate the total variation (SST) in a set of measurements into a component due to random fluctuations (SSE) and a component due to actual difference among the alternatives (SSA) [11].

Intuitively, we can assume that the size of the actual component is 0 and calculate the probability of this assumption being true. If the assumption is highly unlikely to be true than we can reject this assumption. This assumption is typically called the *null hypothesis* and the procedure is called *hypothesis testing*. Given n experiments for each of k alternatives, total variation is sum of squares of difference between each of $n * k$ experiments and overall mean, $\overline{y}$. Similarly, the variation due to the effects of the alternatives is the sum of squares of the differences between the mean of the measurements for each alternatives and overall mean. Lastly, the variation due to errors is the sum of the squares of the differences between the individual measurements and the mean of all measurements for a particular alternative, $\overline{y_j}$.

To test if SSA is statistically significant, we use *F-test*. We can say that SSA is significantly higher than SSE at α level significance if the calculated *F-statistic* is larger than the *critical F value*, $F_{[1-\alpha;(k-1),k(n-1)]}$. Note that $k-1$ is the degree of freedom for SSA and $k(n-1)$ is the degree of freedom for SSE. We can calculate *F-statistic* of the experiment by:

$$\begin{aligned} \textit{F-statistics} &= \frac{SSA(k(n-1))}{SSE(k-1)} \\ &= S_a^2/S_e^2 \end{aligned} \tag{1}$$

where S_a^2 is the variance of SSA and S_e^2 is the variance of SSE. Therefore, F-statistic is nothing but a ratio of the signal variance and the noise variance. *Critical F values* defines the minimum required ratio for the signal to have a statistical significance. *Critical F values* for a given confidence are tabulated in numerous literature [11].

A more compact representation of the same information is the *p-value* which is defined to be

$$p = 1 - P(F \leq f) \tag{2}$$

where $P(F \leq f)$ can be found from *F-density function* which can be found in numerours statistics text books [11]. *P-value* represents the probability that SSA is not statistically significant which is typically rejected if the *p-value* is less than 0.05.

3.2 Linear Regression

A general model that predicts P given W is:

$$P = f_P(w_1, w_2, w_3, ...) + \epsilon \tag{3}$$

where w_i's are set of controlled variables and P is metric of interest. Estimating function f_P without any assumption is

W	Workload	M_W	Workload Model
S	Data Store	M_S	Data Store Model
P	Performance(Latency)	M_P	PerformanceModel(Prediction)
X	Random %	R	Read %
Z	IO size	O	Outstanding IO

Table 1: Variables defined in Romano.

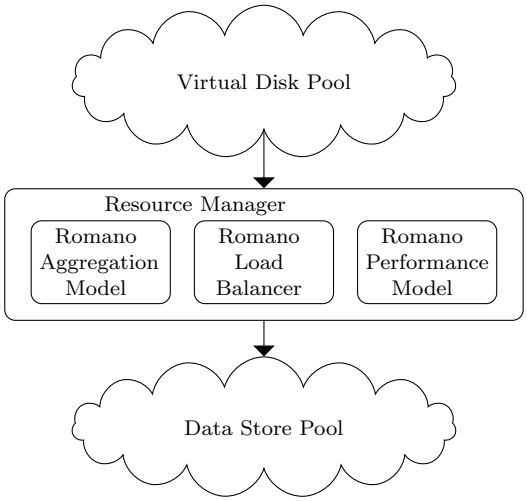


Figure 1: High level view of Romano. Romano maps virtual disks onto pool of data stores to satisfy the load balancing policy.

computationally expensive if not impossible. Therefore, we restrict the form of f_P to a generic linear model of the form:

$$P = \beta_0 + \beta_1 w_1 + \beta_2 w_2 + ... + \epsilon = W\beta + \epsilon$$
$$W = \begin{bmatrix} 1 & w_1 & w_2 & w_3 & ... \end{bmatrix},\ \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ ... \end{bmatrix} \tag{4}$$

One thing to note here is that the linear model only dictates how the β values interact with the controlled variables. The controlled variables themselves can be in the form of any scale. This means that contrary to what many believe, W does not need to have linear relationships with P. The goal is to estimate β such that ϵ is acceptable. Typically, we assume that the effects of any factors not being modeled result in white noise and that the ϵ should be normally distributed.

4. ROMANO DESIGN

Romano is a generic framework that allows automated load balancing of the virtual disks in a heterogeneous storage environment. Specifically, Romano predicts the storage performance using statistical techniques allowing the systems to relocate the virtual disks without the human intervention.

Figure 1 shows an overview of Romano framework. Romano sits within the Resource Manager which maps physical storage resources to the virtual disks. Romano framework consists of *Romano Performance Model*, *Romano Aggregation Model* and a *Romano Load Balancer*. Romano Performance Model predicts the performance of a workload on a given data store. Romano Aggregation Model predicts the characteristics of multiple independent workloads when they are aggregated on a single data store. Romano Load Balancer balances the workloads across heterogeneous data stores such that the mean and maximum latency observed by the virtual disks are minimized.

These components will be described in detail next. Before describing Romano design we define a few variables in Table 1 to help describe our model.

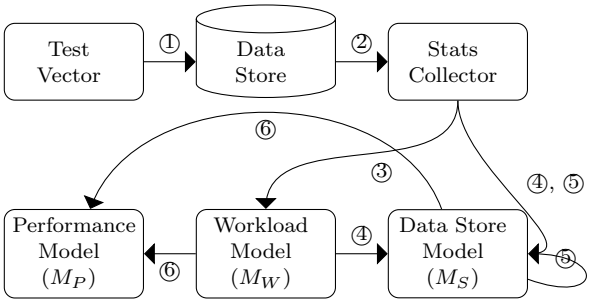


Figure 2: Romano Performance Model framework.

4.1 Romano Performance Model

Figure 2 outlines how the *Romano Performance Model* is generated and maintained. The edges in Figure 2 represents the following procedure.

1. The test vector is run on data stores.
2. Average latencies, P, as well as the characteristics of the workload (X, R, Z and O) per data store are collected at 20 seconds interval.
3. Workload model is generated by determining which of the workload characteristics and their interactions actually affect P.
4. From the workload model and the latencies, the data store model is generated via linear regression.
5. The model is updated periodically based on the new data collected while data stores are active.
6. The performance model is derived from the workload and data store model.

In this section we will describe and justify each step of the process involved in generating the performance model (M_P).

The Goal.

A system's performance is mainly determined by the system itself and its workload. Therefore, there should exist a function, f_P, that can map a given data store (S) and workload (W) onto performance (P).

$$P = f_P(S, W) \tag{5}$$

We define *IO workloads* to be the set of all possible workloads, $W = \{w_i\}$ and *Performance*, P, to be the average IO latency. However, the dimensions of S and W are too large to be parameterized. Furthermore, even if they were not, deriving an accurate non-linear function f_P is difficult. Therefore, we assume the form of f_P to be some linear combination shown in Equation 4 and fit S and W into M_S and M_W such that P can be approximated.

$$P = M_W M_S + \epsilon = M_P + \epsilon \tag{6}$$

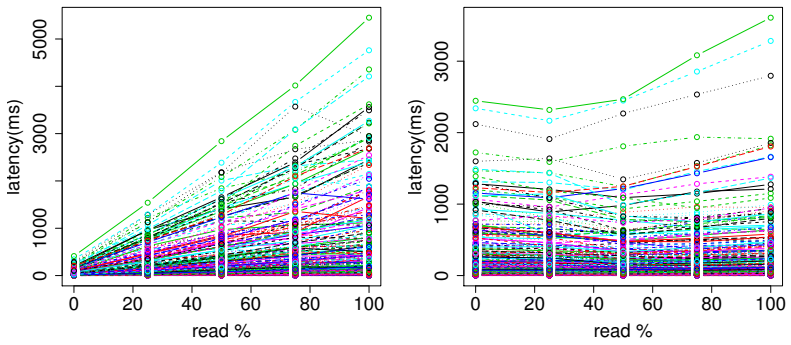
The goal of *Romano Performance Model* is to derive M_W and M_S such that resulting M_P is as close to the actual P as possible.

The Problem.

Since we can measure P, we need to determine M_W to solve the Equation 6 for M_S. Once M_W is defined, we can use linear regression to derive M_S. Without the knowledge about the true distribution of ϵ, we assume it to be Gaussian and solve for M_S by:

$$({M_W}^T M_W)^{-1} {M_W}^T P = M_S \tag{7}$$

Equation 7 is proven to result in smallest ϵ if the ϵ is Gaussian distributed [9]. While the true distribution of ϵ may not be Gaussian, we found the resulting ϵ small enough to justify the assumption. Therefore, the problem is finding a good M_W such that ϵ can be minimized. There is a practical constraint we must satisfy when solving this problem. The characteristics of workloads that can be used to derive M_W must be collected from the system at run time without too much overhead. VMware ESXi 5.0 already collects read ratio (R), average request size (Z) and average number of outstanding IOs (O) at 20 seconds granularity [5]. We implement a mechanism to collect one more characteristic to measure amount of seeks (X) since it is well known that randomness of IO has large effect on storage performance. The problem is now reduced to finding a good M_W from X, R, Z and O such that ϵ is minimized.



(a) E1 (b) N2

Figure 3: Effect of *read%* variation on the *latency* of E1 and N2 data stores. Each line represent latency (P) vs. read% (R) while other parameters(O, Z and X) are fixed. The effect of R on E1 is much more predictable(therefore significant) than N2.

Test Vector.

To investigate how M_W can be derived from X, R, Z and O, we first define the space of all possible workloads in terms of those characteristics.

$$random\%(X) = \{r | r \in \mathbb{I}, 0 \leq r \leq 100\} \tag{8}$$

$$read\%(R) = \{o | o \in \mathbb{I}, 0 \leq o \leq 100\} \tag{9}$$

$$IOsize(Z) = \{s | s \in 2^n B, n \in \mathbb{N}, 9 \leq n \leq 19\} \tag{10}$$

$$OutstandingIO(O) = \{i | i \in \mathbb{N}, 1 \leq i \leq 128\} \tag{11}$$

Note that we limited the range of O and Z which is not enforced theoretically. However, they are typically enforced by the system resource limitations such as buffer size. While different limits can be used, we believe these to be widely applicable in today's systems. Another thing to note is that we have only defined the R and X within $\mathbb{I}$. While this may result in loss of resolution, R and X are always discrete in practice since there can only be finite number of requests and we assume that the close enough values always result in similar latency. The size of the entire space is $|O| * |Z| * |R| * |X| = 14,363,008$. If we tested each point in space for 2 minutes, it would take 54 years to explore all possible combinations. Therefore we sample this workload characteristics space such that;

$$X = \{0\%, 25\%, 50\%, 75\%, 100\%\} \tag{12}$$

$$R = \{0\%, 25\%, 50\%, 75\%, 100\%\} \tag{13}$$

$$Z = \{1K, 2K, 4K, 8K, 16K, 32K, 64K\} \tag{14}$$

$$O = \{1, 2, 4, 8, 16, 32, 64\} \tag{15}$$

which would require 1600 tests and can be done in a couple of days. This sampled space is our *test vector*. The idea of vectorizing a multidimensional continuous space into a single discrete vector is analogous to the way application characteristics are represented in *HBench* [18].

The reduction in workload space due to sampling 1600 datapoints results in no significant loss of accuracy in the model. However, the space is still too large for the modeling to be carried out at low cost on on-line systems. The sampling technique that would provide a reasonable model with minimum testing is an interesting topic we leave for the future research.

We generate our test vector using Iometer [19].

Workload Model.

How to describe an IO workload in a generic manner is an open question [25, 13, 22, 23]. Unlike Basil [7] and Pesto [8], Romano generates a separate M_W for each data store. While it is tempting to derive M_W in a data store independent manner, we found the resulting performance model (M_P) to be neither robust nor accurate. This is because each data store interacts with the characteristics of the workload in a different manner. Figure 3 shows the relationship between P and R for two data stores E1 and N2 from Table 2. The plot shows the latency change observed on two different data stores while varying R and fixing X, Z and O. It is clear that R has distinct effect of E1 but not on N2. This means that M_W for E1 should

Name	No. of disks	RAID	Interface	Make
E1	3	0	SATA	EMC
E2	6	5	FC	EMC
N1	3	5	SATA	NetApp
N2	7	6	FC	NetApp

Table 2: Summary of test data stores used in Romano testing.

E1		E2	
factor	p-value	factor	p-value
O	4.83e-06	O	< 2e-16
R:O	3.09e-08	X:O	7.92e-15
Z:O	3.45e-03	Z:O	< 2e-16
X:R:O	< 2e-16	X:R:O	< 2e-16
X:Z:O	4.51e-03	X:Z:O	3.98e-05
X:R:Z:O	5.99e-11		
N1		**N2**	
factor	p-value	factor	p-value
O	6.85e-03	O	3.06e-16
X:O	< 2e-16	X:O	< 2e-16
Z:O	< 2e-16	Z:O	< 2e-16
X:R:O	2.33e-05	X:R:O	4.68e-08
X:Z:O	< 2e-16	X:R:Z:O	< 2e-16
X:R:Z:O	< 2e-16		

Table 3: Result of factorial ANOVA with latency as response and characteristics of workload as the treatment. Only the factors with p-values less than 0.05 are shown.

contain R but not N2. An important thing to note is that the inclusion of R in M_W for N2 will result in over-fitting since it is likely to only add to the noise. Therefore, removing R from the workload model actually results in a more robust model that is less sensitive to noise.

这里说明了没有考虑到特征之间的影响因素

Another difficulty arises from the fact that the interactions between the characteristics are significant and cannot be ignored. While it is possible to use a simple tuple of X, R, Z and O, we found the resulting ϵ to be very large. Interactions between the characteristics can be factored in by simply adding additional terms for the products of two or more characteristics. For example, the interaction between X and R can be factored in to the model by adding the additional term $X : R = X * R$. Therefore, given 4 characteristics, there are $2^4 - 1 = 15$ possible terms including the interactions. To check which of the 15 terms actually have a significant effect on the performance, we conduct ANOVA at 95% confidence controlling the 15 terms (using the test vector) and observing the latency on four test data store shown in Table 2.

The *p-values* derived from the ANOVA test is shown in Table 3. These values have a significant effect on performance (P) with 95% confidence. For example, the M_W for E2 is a row vector of the form;

$$M_W = \begin{bmatrix} O & X*O & Z*O & X*R*O & X*Z*O \end{bmatrix} \tag{16}$$

The resulting M_W accounts for any interactions as well as their significance on a given data store. While the cost of the experiment is not cheap, this only has to be done once per data store type in the system.

Data Store Model.

Using the M_W, we find the M_S using Equation 7. Table 4 shows the resulting M_S for the four test data stores. The *coefficients* columns of each data store form a column vector which is M_S of that data store. The accuracy of M_S depends heavily on the number of data points. While initially it can be derived from our test vector, it can be updated from the data collected in production environment to improve its accuracy.

E1		E2	
factor	coefficients	factor	coefficients
O	1.064	O	8.370e-01
R:O	2.828e-02	X:O	6.163e-03
Z:O	1.505e-05	Z:O	1.172e-05
X:R:O	2.277e-03	X:R:O	4.079e-04
X:Z:O	2.545e-07	X:Z:O	9.970e-08
X:R:Z:O	9.385e-09		
N1		**N2**	
factor	coefficients	factor	coefficients
O	1.428	O	1.310
X:O	1.653e-01	X:O	1.403e-01
Z:O	1.293e-04	Z:O	9.078e-05
X:R:O	-3.24e-04	X:R:O	1.579e-06
X:Z:O	2.052e-06	X:R:Z:O	1.514e-08
X:R:Z:O	2.613e-08		

Table 4: Data store model (M_S) for four test data stores. (M_S) is a column vector whose entries are shown in the *coefficients* column of the each data store.

Performance Model.

Finally, we are now ready to calculate M_P, which is the prediction of P, by taking the cross product of M_S and M_W. One interesting thing to note from Table 3 is that all the factors contain O. This is because the average latency (P) of all data stores are linear to the number of outstanding IOs (O). Note that it also means that P is not linear to X, R and Z.

We have tested 12 different data stores and verified that the linear relatioship between the outstanding IOs and the latencies hold. Therefore, we can rewrite Equation 7 as;

$$\begin{aligned} M_P &= M_W M_S = M'_W M_S O \\ M'_W &= \frac{M_W}{O} \end{aligned} \tag{17}$$

Since the *outstanding IO* (O) is the number of IO requests outstanding in the queue and P is time the requests spend in the system, by Little's law [12], $M'_W M_S$ is the average inter-arrival time of requests. It is also the average service time of requests at equilibrium and its inverse is the throughput (T) in IO/s. Pesto [8] describes the linear relationship between O and P in much more depth and we refer interested readers to their work.

The Figure 4 shows latency (P) change as O is varied while fixing X, R and Z. Except for some exceptions caused by the measurement errors, all plots show linear relationship between O and P. $M'_W M_S$ is called *LQ-slope* in Pesto, and we continue to use the terminology for the sake of consistency. Therefore, we can rewrite Equation 17 to our final form of M_P.

$$P = M_P + \epsilon = LQslope * O + \epsilon = M'_W M_S O + \epsilon \tag{18}$$

There are two benefits of using Equation 18 compared with Equation 6. First benefit is that O no longer needs to be modeled. This effectively reduces the size of M_S and M_W simplifying the modeling precess as well as prediction process. The second benefit comes when we derive the *Romano Aggregation Model* which we will describe in next section.

Prediction Interval.

The most important feature of Romano is that it provides the confidence of its own prediction. Romano keeps track of the variance (σ^2) of ϵ from which the prediction interval is calculated to be;

$$M_P \pm \frac{1.96\sigma}{\sqrt{n}} \tag{19}$$

at 95% confidence. Therefore, the 95% of actual P should fall within the confidence interval. n is the number of data points and $\pm 1.96\sigma$ is used to capture the 95% of the Gaussian distribution which we assume is the distribution of ϵ. In short, the prediction interval is proportional to the variance of residuals and inversely proportional to the square root of the number of observations. If the prediction interval is too large, the prediction is useless. To tighten the interval we can do two things. First is to reduce the confidence level. However this approach may lead to too many bad predictions that could result in bad

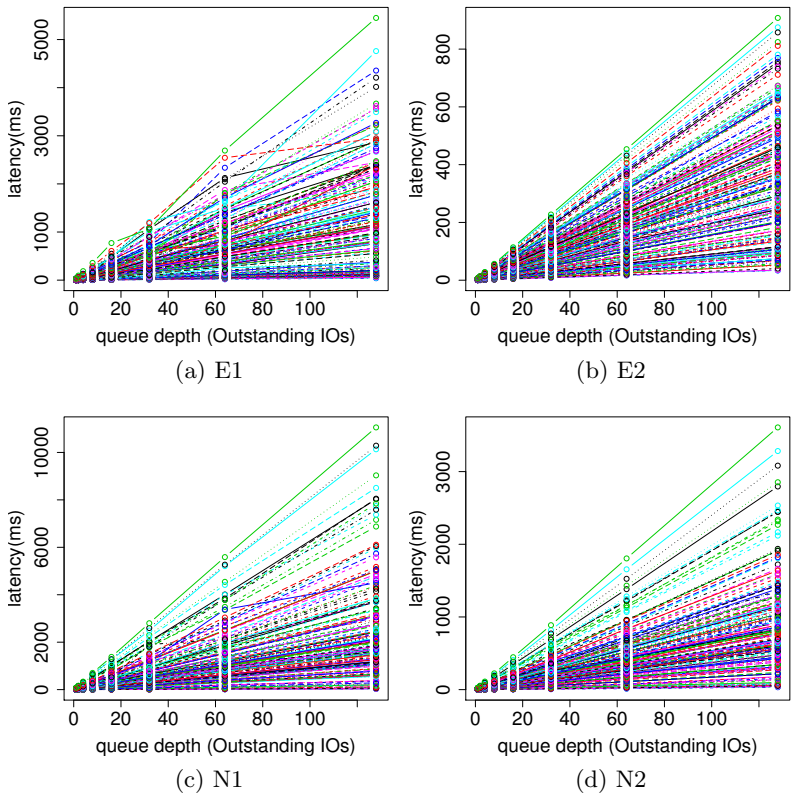


Figure 4: Effect of outstanding IO (O) variation (from 1 to 128) on the latency of test data stores. Each line represent latency (P) vs. outstanding IO (O) while other parameters are fixed. Except for the few noise, the relationship is clearly linear for all data stores.

decision making. Another approach is to increase the number of observations. In Romano, the performance statistics and the workload characteristics are collected in real time to update the M_S in effort to tighten the prediction interval.

Stats Collector.

In order to update M_S, the *stats collector* collects workload characteristics and performance in real time. P is measured at vSCSI layer as the completion time subtracted from time of queueing per request. R and Z are simply read from IO request attributes. O is measured from the queue depth of the *vSCSI* [2] device which is created per virtual machine (VM) and virtualizes the SCSI device layer. We had to implement the mechanism to collect X at SCSI device layer since vSCSI is unaware of large seeks that result from seeking between multiple virtual disks in a data store. X is measured by simply looking at the percentage of seeks larger than 10 times the average request size. The reason for the multiplier is that some workloads exhibit significant variance in their request size. Since the seeks are measured in terms of address difference between consecutive requests, large requests could falsely register as seeks even when they are sequential. All stats are averaged and reported to *Resource Manager* (in Figure 1) at 20 seconds granularity.

4.2 Romano Aggregation Model

Predicting the performance of a data store given a workload is not always enough. If the data store is already running different workloads, we need to figure out what the characteristics of aggregated workload will be.

In Basil, this is done by simply adding their workload parameter ω [7]. This made sense under their assumption that all data stores react to different workloads in a similar manner. Since ω represents the amount of work a data store has to process, aggregating workloads only involve addition of ω's.

Romano's workload model is a tuple of different combinations of characteristics. It makes more sense to aggregate each characteristic separately. In our closed loop assumption, aggregation of workloads does not affect the number of IOs each workload keeps in the system. Therefore we can safely assume that the *Outstanding IO*s (O) will simply be added.

However, *Size* (Z) and *Read Ratio* (R) can not be simply added together. It needs to be averaged based on ratio of number

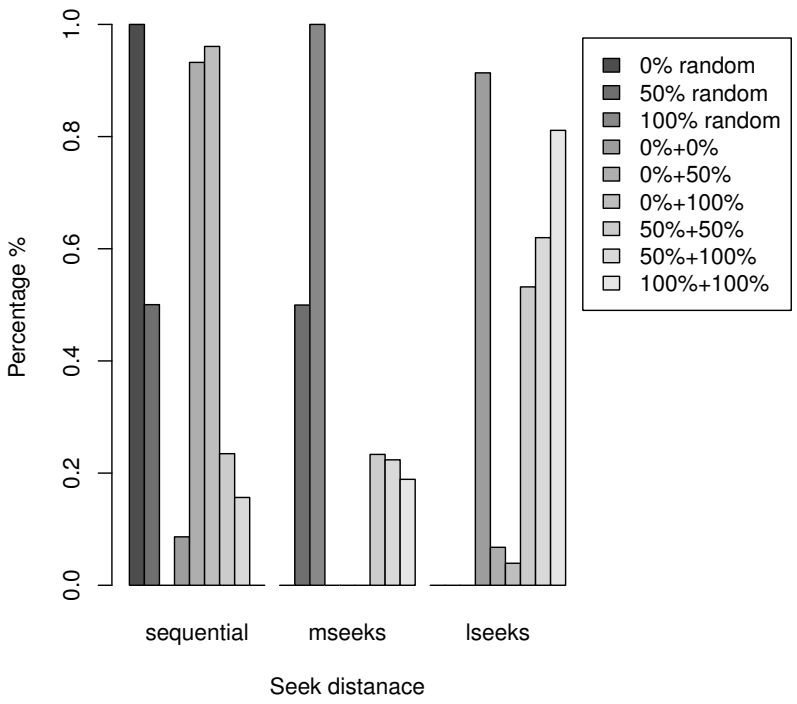


Figure 5: Seek profile of different workloads. Each color represents different mix of X for workload with $Z = 4KB$, $R = 100\%$ read, $O = 1$. *mseeks* represents seeks done within a virtual disk and *lseeks* represents seeks done across the virtual disks.

of requests seen by the two workloads that are being aggregated. Pesto weights each value by their O, assuming that higher O means higher rate of requests. However, O can be highly independent of arrival rate. For example, imagine a process that generates 5 IO requests and issues additional IO whenever a request completes. The arrival rate will only depend on the service rate of the requests and not on outstanding IOs which would be fixed at 5. For Romano this weight to be used for averaging is already available as the inverse of *LQ-slope*, which represents the throughput (T) from our performance model in Equation 18.

The *Random* (X) is tricky because the randomness of access cannot be defined per IO request and cannot be simply averaged. Figure 5 shows the changes in the seek profile as you mix 0%, 50% and 100% random workloads on E1 data store. Our measurements show that the *LQ-slope* for these workloads are 0.476, 17.6 and 35.8 respectively. Therefore, the throughput (T), of these workloads are 2100.52, 56.77 and 27.93 respectively.

Note that *mseeks* represents seeks within a virtual disk while *lseeks* represents seeks across the virtual disks. If we define the *randomness* of the workloads as percentage of the seeks, the effect of aggregation is completely non-linear and hard to predict. However, we make the following observations.

1. Mixing sequential workload and non-sequential workloads results in very low *mseeks*.

2. The number of lseeks is determined by the lower throughput of two workloads.

3. The number of lseeks reduces the sequential and mseeks by the ratio of their average T.

More formally,

$$\overline{seq} = \frac{seq_1 T_1 * seq_2 T_2}{T_1 + T_2} \tag{20}$$

$$seq = \overline{seq} - (\overline{seq} * \%lseek) \tag{21}$$

$$\%lseek = \frac{\min(T_1, T_2)}{T_1 + T_2} * 2 * \overline{seq} \tag{22}$$

which can be solved for $X = 1 - seq$ to be:

$$R_{aggr} = f_X(X_1, X_2, T_1, T_2) = 1 - \frac{T_1 T_2 (1 - X_1)(1 - X_2)(1 + 2\min(T_1, T_2))}{T_1 + T_2}. \tag{23}$$

Equation 20 is a simple throughput weighted average of the $sequential\%$ representing average $mseeks$. However, this does not

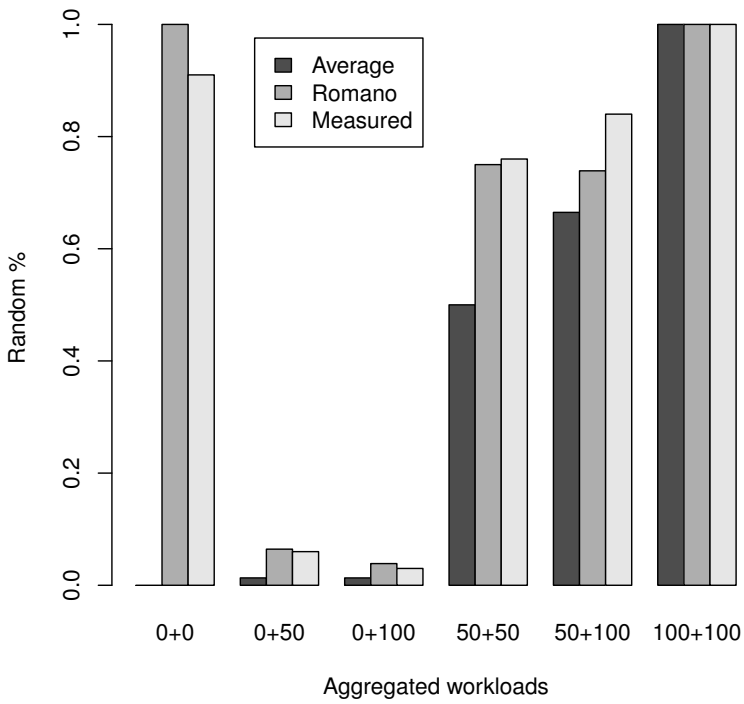


Figure 6: Comparison of f_R and T weighted average for the randomness of aggregated workloads. Each color represents different methods. Other characteristics are fixed at $S = 4KB$, $R = 100\%$ read, $O = 1$.

account for additional seeks due to serving multiple virtual disks. Therefore Equation 21 subtracts the amount of $lseeks$ from the $sequential\%$. Note that the $\%lseek$ in the equation represents the percentage of $lseek$s. Intuitively, the storage system is typically serving the high throughput workload and occasionally seeks to low throughput workload which causes the $lseek$. Therefore, Equation 22 calculates the amount of $lseeks$ which is proportional to the ratio between the low throughput and the total throughput. This value is multiplied by $2 * \overline{seq}$ since almost every $lseek$ between the sequential workload is matched by a $lseek$ in the opposite direction to serve both virtual disks.

Figure 6 shows the result of f_X against simple T weighted average. The maximum and average error of doing T weighted average is 100% and 48% respectively. For f_X these values are 28% and 5% respectively. The maximum error in both cases are when aggregating two completely sequential workloads whose resulting X_{aggr} has inherently high variance.

Putting it together, *Romano Aggregation Model* is;

$$(X_{new}, R_{new}, Z_{new}, O_{new}) = (f_X(X_i, T_i), \frac{\sum_i R_i T_i}{\sum_i T_i}, \frac{\sum_i Z_i T_i}{\sum_i T_i}, \sum_i O_i) \tag{24}$$

where aggregating more than 2 workloads require f_X to be recursively applied to the workloads. In practice, this is naturally done since only a single virtual disk is migrated at a time.

4.3 Romano Load Balancer

The *Romano Load Balancer* is different from traditional load balancers [7, 8, 21] in two ways. Firstly, it optimizes the global state rather than a single storage migration. Secondly, it is proactive. The balancing mechanism does not wait until a overutilization is detected. It continuously adjust the overall state whenever possible to ensure that systems are less likely to be over-utilized. This is made possible by Romano's accurate and robust modeling techniques.

Merit Metric.

Our load balancing goal is to minimize the overall latency while reducing the maximum latency. To represent both aspect of our goal with a single metric, we define Romano load balancing metric of Merit (M) to be the form;

$$M = \left(\sum_i P_i^{\alpha} \right)^{\frac{1}{\alpha}} \tag{25}$$

Algorithm 1: Romano Load Balancing Algorithm using Simulated Annealing

```
updateMerit (W_cand, S_cand, Φ) begin
    W_src ← getW(getS(W_cand, Φ), Φ)
    W_dest ← getW(S_cand, Φ)
    W'_src ←aggregate(W_src − W_cand)
    W'_dest ←aggregate(W_dest + W_cand)
    P'_src ←predict(S_src, W'_src)
    P'_dest ←predict(S_dest, W'_dest)
    return merit(P'_all)
end
minimizeMerit (Φ_i, M_i) begin
    Φ ← Φ_i; M ← M_i; Φ_f ← Φ_i; M_f ← M_i; K ← 0; while K < K_max do
        T ←temperature(k/k_max)
        W_cand ←randomChoose(W_all)
        S_src ← getS(W_cand, Φ)
        S_cand ←randomChoose(S_all − S_src)
        M' ←updateMerit(W_cand, S_cand, Φ)
        if G(M, M', T) > random() then
            Φ ←(W_cand → S_cand)
            M ← M'
        if M < M_f then
            Φ_f ← (W_cand → S_cand)
            M_f ← M'
        K ← K + 1
    return Φ_f, M_f
end
loadBalance (Φ_i) begin
    M_i ←merit(Φ_i); Mlist ← ∅
    (Φ_f, M_f) ←minimizeMerit(Φ_i, M_i)
    foreach W ∈ Φ_f do
        if getS(W, Φ_f) ≠ getS(W, Φ_i) then
            MoveCandidate ← W
    while MoveCandidate do
        foreach W ∈ MoveCandidate do
            M' ← updateMerit(W, getS(W, Φ_f), Φ_i)
            append(Mlist, (M', W, getS(W, Φ_f)))
        sortDescendingByM(Mlist)
        (M, W, S) ← Mlist[0]
        Move(W, S)
        M_i ← updateMerit(W, S, Φ_i)
        Φ_i ← (W → S)
        MoveCandidate ← MoveCandidate − W
end
```

where α controls the weight on the maximum latency. For example, if α is 1, M is simply the sum of all latencies. However, as α becomes larger, more weight is given to high latencies. Our experiment shows that 5 is a reasonable value for α. However, further exploration of α space is left for the future work.

Load Balancing.

Romano uses *simulated annealing* [10] to optimize the overall placement and then makes moves that result in maximum reduction of M. The reason for the first step is that greedy approaches used in previous works [7, 8, 21] could result in a local optimum that is far from the global optimum. The solution cannot be greedy since any optimal sub-placement needs to be reevaluated once a new workload is added due to the aggregation function. Adding a new data store is worse since

every data store has its own performance model. While the simulated annealing process does not necessarily result in optimal placements, it is guaranteed to be better than finding a local optimal if the algorithm can remember the best state it has seen so far [6].

However, the resulting state may require too many workloads to be migrated. Therefore, once the target state is defined through simulated annealing, we use a greedy method to get there. This way, we can make moves that result in the largest reduction of M while always moving towards the optimal state. Of course, there may be a move that results in an even larger reduction but the move may prohibit the possibility of further reduction of M.

Algorithm 1 outlines the Romano load balancer. In this algorithm, Φ represents the current mapping of workloads to data stores. $getS(W, \Phi)$ represents the data store on which W is mapped in current State Φ while $getW(S, \Phi)$ represents all the workloads mapped onto data store S in state Φ. *updateMerit* function takes the current system state (all Workload mappings and their performance) and calculates new *Merit* value. Note that our goal is to minimize the Merit value. The *aggregate* function is *Romano Aggregation Model* and *predict* function is *Romano Performance Model*. Therefore, *updateMerit* function is given a workload candidate to be migrated and destination data store. New aggregated workloads are calculated and their performances are predicted. Given the new performances of source and destination data stores, global merit is recalculated.

The *minimizeMerit* function uses simulated annealing to minimize the Merit value. The *temperature* function and G functions are taken from the earlier works on simulated annealing [10]. The *temperature* function simply divides k/k_max by 1.2 every iteration. G function returns 1 if current merit is lower than previous merit so that the move is always accepted, but returns $e^{(M-M')/T}$ when current merit value is higher. Therefore, there exists a finite chance of accepting a move even when the merit increases. This probability decreases fast with the number of iterations and the difference between the new and old merit values. With every iteration, the function chooses a random workload to move and a random destination data store. New merit is calculated and the move is either rejected or accepted based on the value returned by G and a random number between 0 and 1. All our experiments required less than 200 iterations to complete. The convergence rate of simulated annealing is difficult to predict without any structural information of the state space. However, since we do not need the absolute minimum M, we can simply fix the maximum number of iterations to a reasonable number and choose the best state found.

The *loadBalance* function takes the current state (Φ) and calls on *minimizeMerit* function to determine the pseudo-optimal mapping of W on S. Once the mapping is determined, it uses a greedy approach to determine which moves to execute. *updateMerit* function is called on all possible moves to determine the next move. Note that *updateMerit* function executes in $O(1)$. The process is repeated until the pseudo-optimal mapping is reached.

In practice, it is unlikely that all the moves will be carried out due to the cost involved with the migration. Furthermore, the workload themselves are going to change over time. Therefore, Algorithm 1 will periodically rerun *minimizeMerit* function and continually transition the system towards the newly found pseudo-optimal state without ever actually getting there.

5. EVALUATION

As described in Section 4, there are three main components which contribute to better prediction accuracy of *Romano* over current techniques [7, 8]. The first component is a more robust (by removing characteristics that amplify the noise of the prediction) and accurate (by modeling of interactions) performance model. Furthermore, Romano generates a 95% prediction interval which statistically guarantees that 95% of all predictions will fall within the interval. The second component is the ability to handle heterogeneous response patterns of different data stores by quantifying the significance of workload characteristics on each types of data store using ANOVA. The last component is the aggregation of workload based on the predicted IOPS of the workloads. While this approach is not perfect we show that it works better than Pesto in practice.

All experiments are performed on the data stores shown in Table 2.

5.1 Prediction Accuracy

This section evaluates the accuracy and the robustness of the Romano model. Specifically, we use Basil [7] and Pesto [8] as the baseline models and assume that the *full factorial linear model* (FFLM) is the ideal case since it uses all possible linear combinations. To evaluate the accuracy we use *test vector* and run them against for 4 test data stores (Table 2). The resulting 6400 latencies are compared against the predictions of 4 techniques.

Table 5 shows aggregated ϵ values, the difference between the prediction (M_P) and the measurements (P), of different modeling techniques. The values were aggregated from latency prediction of all four data stores under test (from 6400 data points). It shows that the performance of Basil is unacceptable in a heterogeneous data store environment for most applications with high mean ϵ of $642ms$. This is expected since Basil generates a single model for any data stores. Pesto takes into the account the performance difference of each data store. However, it assumes that one data store is simply more powerful

	Basil	Pesto	FFLM	Romano
Min.	-27380	-1.33	-1669	-1667
1st Qu.	-441.8	12.03	-0.1037	-8.610
Median	-105.2	52.67	-0.7699	-0.1033
Mean	-565.6	292.5	1.860e-13	1.191
|Mean|	642.4	292.5	44.43	44.34
3rd Qu.	-22.31	243.8	7.872	9.043
Max.	6242	10590	1141	1146

Table 5: Comparison of residuals (ϵ) for different modeling techniques. The residuals were aggregated from all four data store models. 1st Qu. and 2nd Qu. represents 25% and 75% quantile values respectively. |Mean| represents mean of absolute values of residuals. Units are in ms.

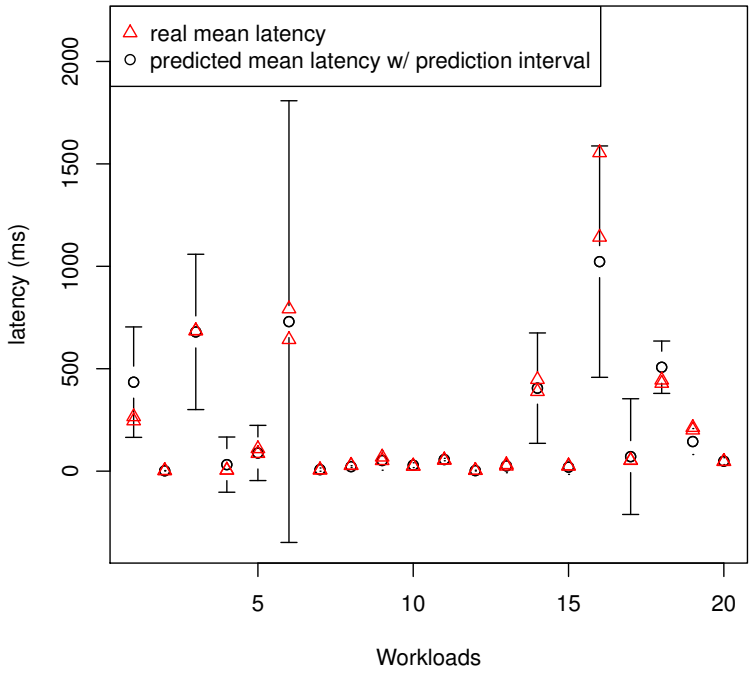


Figure 7: The average latency (over 20 second intervals) observed from 20 different workloads (running on separate virtual disks) on 4 different test data stores during 1 hour interval. The max and min latency is plotted. Also plotted are the Romano's 95% prediction intervals.

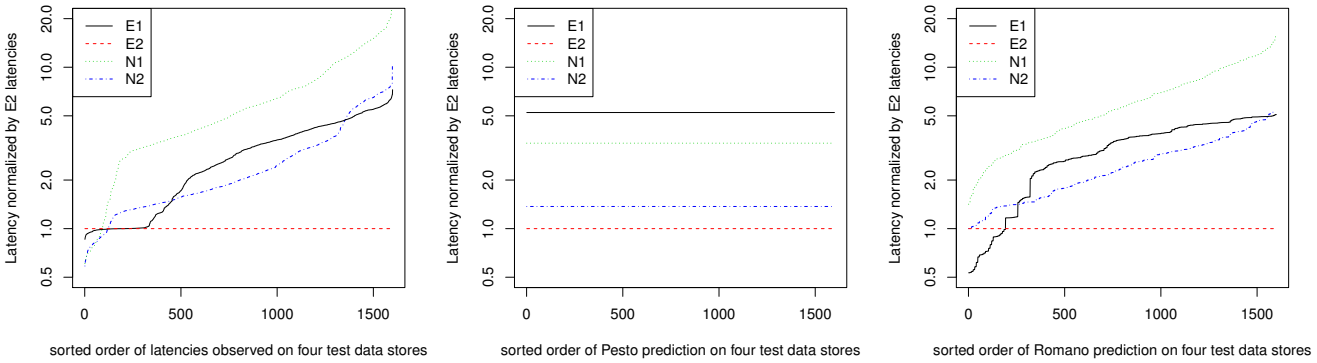
than the other regardless of the workload. While it provides a better accuracy than Basil by 60% on average, it still shows unreasonable maximum ϵ of 10s and average ϵ of 260ms.

Romano shows a much stronger predictive power with average ϵ of 44ms and the maximum ϵ of 1.3s. This is an 82% reduction for the maximum residual and 83% reduction for the average residual compared to Pesto. In fact, Romano performs slightly better than FFLM although the difference is too small to be meaningful. However, Romano is capable of faster model updates and prediction since it only uses 4-5 terms versus 16 terms used by FFLM. Furthermore, 50% of all predictions (between the 1st and 3rd quantile) exhibit less than 10ms of error.

In short, Romano shows 80% improvement in overall prediction accuracy on heterogeneous workloads and data stores compared to the most recent technique [8].

5.2 Prediction Interval

Perhaps the most significant benefit of Romano model is its ability to provide a prediction interval at pre-defined confidence for its predictions. Figure 7 shows the average latencies of 20 different virtual disks running different workloads randomly selected from our test vector on four test data stores. The average was evaluated over two minutes. The predicted latency and its corresponding interval (shown as the error bar) was calculated using Romano at 95% confidence (allowing 5% of measurements to be outside of the prediction interval).



(a) Measured Relative Latency (b) Pesto Prediction - relative (c) Romano Prediction - relative

Figure 8: Sorted normalized latency plot for four test data stores. It shows the relative latencies of E1, N1 and N2 compared to E2. The 1600 latencies measured from each data store were normalized by E2 and than sorted. The y-axis represents the normalized latency and the x-axis represents the sorted index. The actual measurements as well as predictions from Pesto and Romano are plotted.

E1	E2	N1	N2
25.798101	4.922433	16.666000	6.753044

Table 6: Pesto LQ-slope of four test data stores. The units are ms/io. According to these slopes, E2 is the fastest data store and E1 is the slowest.

Figure 7 shows that the predictions can be off by up to $500ms$ (*workload 16*). However, all measurements fall within the prediction interval. In fact, out of 1600 workloads ran against 4 data stores in the previous experiment, Romano failed to capture the measured latency within its prediction interval 423 times which is 6.6% (close to predefined 5%) of 6400 predictions made.

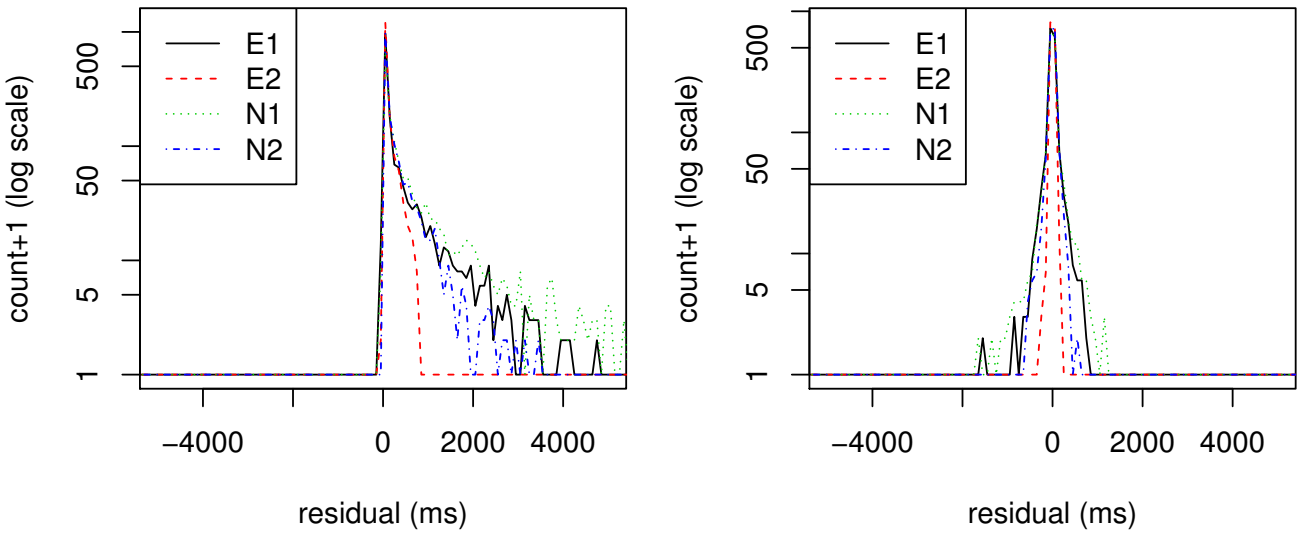
Another interesting thing is that there were roughly 5% of workloads whose confidence interval is so large, like the *workload 6* in Figure 7, that the prediction itself is meaningless. Pesto [8] claimed that these workloads had specific characteristics (such as purely sequential) and can be filtered out. However, we found them to be more of random occurrences and difficult to predict in advance. By default, the prediction is done every 8 hours to consider the possible moves in current VMware products [5]. However, we force prediction every 2 minutes and observed prediction interval change after 5 hours. In theory, the prediction interval should converge after about 2 hours (12 measurements), however we found the convergence to take a much longer time. We suspect the reason to be fluctuation in the workload. A detailed analysis and speeding up this convergence will be our future work.

In short, Romano is capable of providing prediction interval at a specified confidence level. The level of confidence can be lowered to trigger more aggressive load balancing with tighter prediction interval. On the other hand a higher confidence interval will result in moves that are only triggered when predicted benefit is large enough to over come the larger prediction interval.

5.3 Handling Heterogeneous Data Stores

In this section, we evaluate Romano's ability to handle heterogeneous data stores using Pesto [8] as the baseline. Basil is not compared here since it only generates a single model across the data stores and can not handle heterogeneity. Pesto injects a *4KB, random and read-only* workload into the data store to derive the *LQ-slope* for a given data store. Table 6 shows the Pesto LQ-slope of four test data stores. Since each data store has a fixed LQ-slope regardless of the workload, running same workloads on multiple data stores will always result in constant ratio equal to the LQ-slope ratio. Figure 8b plots this effect.

Figure 8a shows the latency normalized by E2 latencies from running our *test vector*. As expected, most of latencies observed in other data stores are larger than that of E2 (with the smallest LQ-slope hence the fastest) but surprisingly, there exist some corner cases where even the slowest E1 out performs E2. In fact, N1 seems to have the highest latency of all data stores on average even though its LQ-slope is lower than E1. Figure 8c shows that Romano is able to capture this relative difference fairly accurately. For those corner cases where Romano fails to capture the exact relative order of performance,



(a) Pesto (b) Romano

Figure 9: Distribution of residuals (ϵ) for Pesto and Romano. x-axis is the value of ϵ in ms. y-axis is number of occurrences+1. Addition of 1 was required to allow y-axis to be in log scale.

Workload	*IOSize*	*Read%*	*Random%*
Web File Server	4KB	95%	75%
Web File Server	8KB	95%	75%
Web File Server	64KB	95%	75%
Support DB	1MB	100%	100%
Media Streaming	64KB	98%	0%
SQL Server Log	64KB	0%	0%
Web Server Log	8KB	0%	100%
OLTP DB	8KB	70%	100%
Exchange Server	4KB	67%	100%
Workstation	8KB	80%	80%
VOD	512KB	100%	100%

Table 7: Workload characteristics suggested by Wang [24]. These workloads are used to ensure repeatable experiments.

the prediction intervals tend to be very large. Therefore, Romano will not make any predictions for those cases with high confidence indicating that actions should not be made based on the prediction values only.

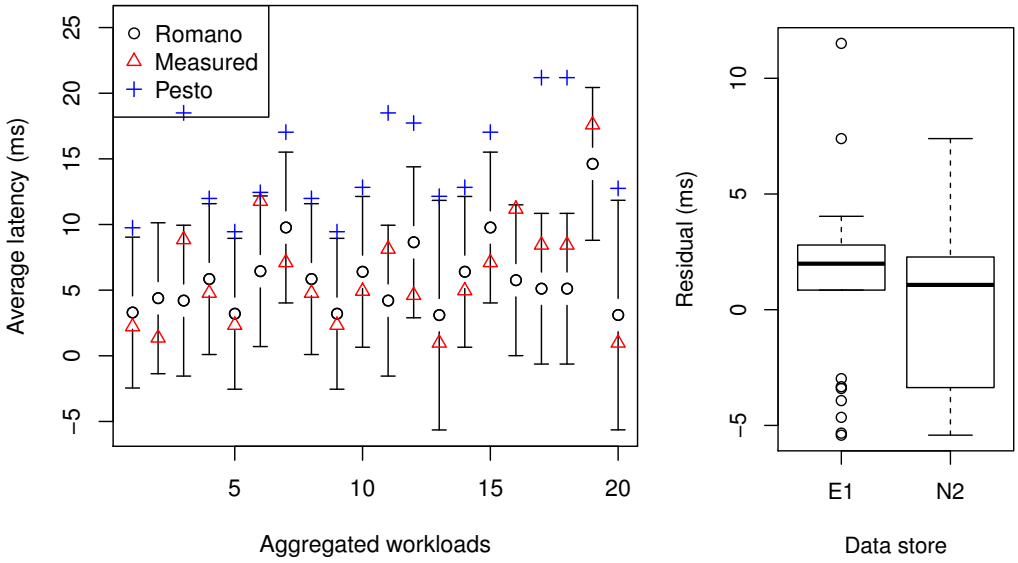
Figure 9a shows that the residual (ϵ) distribution of Pesto varies widely depending on the data store. The accuracy of Pesto on a single data store provides little information about how it might do on another data store. As the new data stores are introduced into the storage pool, the applicability of Pesto needs to be re-evaluated.

On the other hand, Romano shows a much more even distribution across the data stores in Figure 9b. While there are some differences, prediction accuracy of Romano is robust against the heterogeneity of data stores. Furthermore, Romano's prediction interval further improves its robustness by generating larger interval for the data stores that are difficult to model. In short, Romano provides robust and adaptive models across the heterogeneous data stores with a single framework.

5.4 Workload Aggregation

To test the effect of workload aggregation, we need to able to generate a repeatable workload. We use the workload settings shown in Table 7 to simulate the real workloads. Previous works have shown that synthetic workloads do a good job of representing real workload's performance characteristics when the parameters are extracted from the real workloads [23, 17].

All possible combinations of two workloads were chosen from the 11 workloads shown in Table 7 resulting in 55 combinations.



(a) Sample of latency prediction

(b) Residuals box plot

Figure 10: Result of workload aggregation. Two Workloads from Table 7 were placed randomly across two data stores, E1 and N2. Figure 10a shows 20 randomly sampled results from 110 placements on both data stores (55 each). Figure 10b shows the residual of all 55 placements on E1 and N2.

In Romano workload characteristics space, aggregating 2 workloads and 3 workloads are the same since the aggregation process is commutative and associative. O was randomly assigned from 1 to 3. The test was repeated on two data stores for which the Romano model performed worst (see Figure 9b).

The workload aggregation method used in Pesto [8] is to average their workload metric ω weighted by OIO. To isolate the effect of workload aggregation, we average individual workload characteristics weighted by O and use *Romano Performance Model* to predict the latency.

The result is shown in Figure 10. Figure 10a shows the latency from 20 randomly selected experiments as well as the prediction interval of Romano. Only in 1 case, Romano failed to capture the latency within its prediction interval. In fact, only 3 out of 110 cases (2.7%) showed Romano failing to capture the measure latency. This is better performance than specified 5% confidence level. We believe that this is due to the unbiased nature of Romano residual shown in Figure 9b. As the workloads are aggregated, the residuals (ϵ) of *LQ-slope* prediction, used for workload aggregation, are more likely to cancel each other out.

The accuracy of Pesto's approach of O weighted average does not seem too bad at first but there are some cases where the prediction won't even register on the Figure 10a (workload 2 and 16). This is especially true if the workloads running together are highly skewed such that one workload has much higher throughput than the other. This verifies our assumption that the workload should be aggregated based on its throughput rather than O. Figure 10a also shows that while Pesto consistently overpredicts the aggregated latency, Romano residuals are more unbiased shown in Figure 10b.

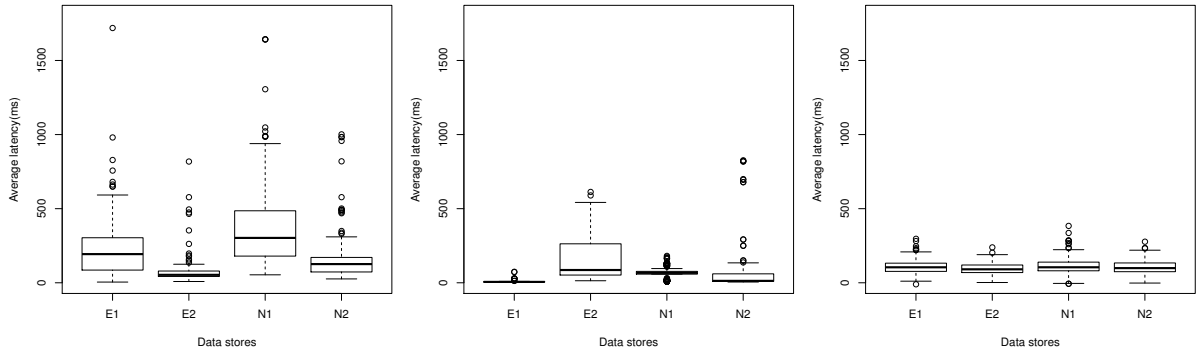
Another interesting result was that all 3 cases where Romano failed were observed on E1 data store. This is as expected from the amount of outliers shown in Figure 10b. We believe that amount of uncertainty introduced by workload aggregation is different on each data store. We leave the further investigation to future work.

5.5 Load Balancing Using Romano

In this section we randomly placed 8-14 workloads from Table 7 allowing repetition on 4 data stores. Romano load balancing algorithm from Algorithm 1 was ran to optimize the data placement by moving a single virtual disk at a time. The experiment was repeated 50 times.

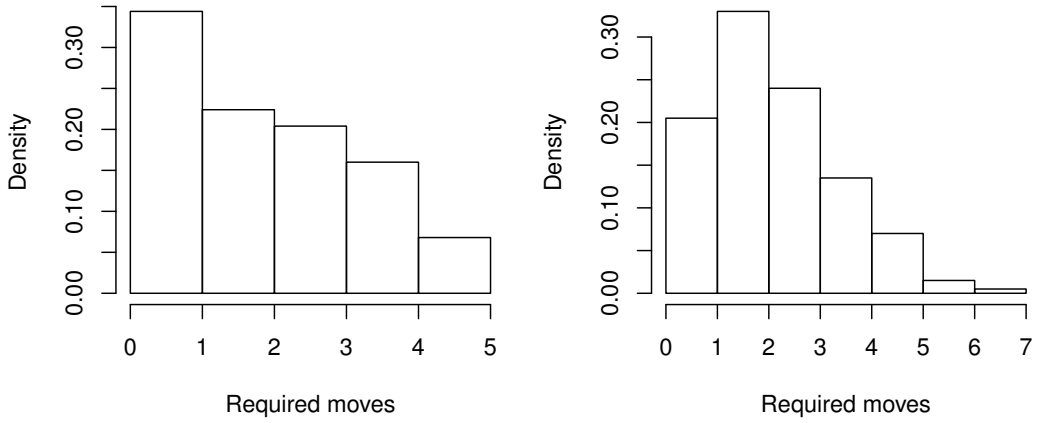
Figure 11 shows the aggregated result of load balancing using Basil and Romano. It should be noted that the load balancing algorithm deployed by Basil and Pesto are the same. Both Basil and Romano does a good job of reducing the average latency by 47% and 52% respectively. However, the variance is reduced by 82% by Romano while Pesto reduces it by 37%. Most importantly, the maximum latency observed was reduced by 78% with Romano while only 21% by Pesto.

There are two factors which result in improved performance by Romano. The first is the more accurate modeling. Both



(a) Initial data stores latencies. (b) Result of Basil load balancing (c) Result of Romano load balancing.

Figure 11: Boxplot of average data store latencies on 50 different test cases. Each test randomly placed 8-14 workloads from Table 7 (allowing repetition) on 4 data stores. Basil's greedy approach is compared with Romano's random optimization technique.



(a) Basil (b) Romano

Figure 12: The distribution of number of moves resulting from Basil and Romano.

Pesto and Basil assume that there exists only a constant performance difference between different data stores for any given workload. Therefore, it has tendency to move virtual disks to a data store that is more powerful regardless of the workload. Furthermore, their aggregation model is based on the outstanding IOs only. This allows workload that do not performance well together to be placed on a single data store. The result of these inaccuracies in the model forces the load balancing algorithm to place more workloads on the powerful data stores. In fact 51% of all the moves by Pesto were made to E2 data store compared to 28% for Romano. Figure 11b shows that resulting E1 and N1 latencies are smaller than that of Romano while E2 and N2 exhibit large latencies that we would like to avoid. In fact, Basil does not place any workload on E1 90% of the time even though E1 actually has lower latency for most workloads than N1 as shown in Figure 8a.

The second factor is the load balancing algorithm itself. Basil and Pesto uses a greedy approach [7] which terminates the algorithm once no beneficial moves are found. Romano uses simulated annealing to find the pseudo optimal placement first and than uses greedy approach to get to the optimal placement. This allows Romano to avoid moves that may result in maximum benefit but prohibits any further moves that may provide additional benefits.

One limitation of this approach is that it does require more moves to be made. Figure 12 shows the number of moves required for Basil and Romano. On average Basil requires 2.2 moves where as Romano required 2.6. Storage migration is expensive and should avoided if possible. However, we show that even if we limit the maximum number of moves to 2, Romano still out performs Basil as shown in Figure 13. This is due to the greedy approach in which Romano chooses moves once the pseudo-optimal placement is identified. Figure 14 shows an example of latency changes for a single test case. It is shown that most critical balancing is done within the first couple of moves.

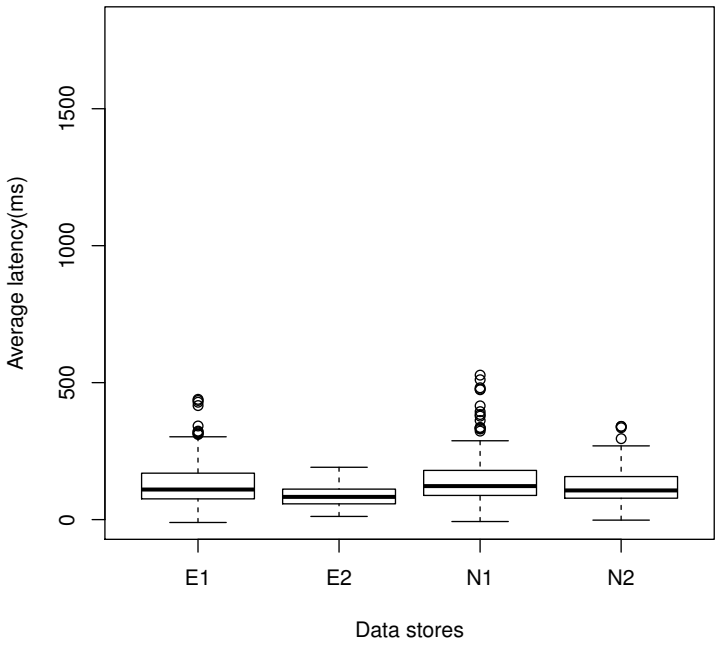


Figure 13: Data store latencies after maximum of 2 moves with Romano.

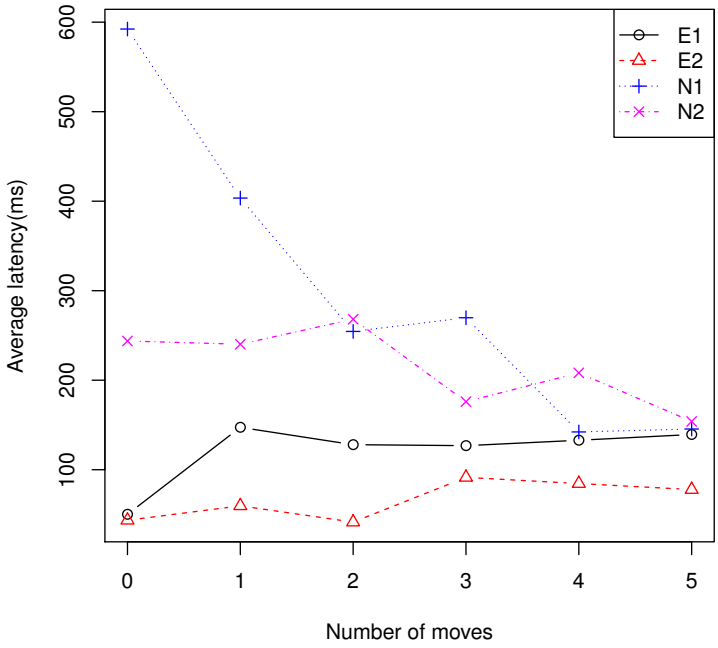


Figure 14: Typical Romano load balancing process with large number of moves.

6. FUTURE WORK

Romano model generation currently plays 1600 workloads against each data store to create an initial model. We are currently actively working to find the minimum number of workloads that needs to be ran in order to generate a decent model using stratified sampling over the workload space instead of uniform sampling.

The online update of the model shown in Figure 2 needs to be evaluated. While we have shown that the increased number of data points will allow Romano to adjust its prediction interval for a higher confidence, we have not evaluated Romano's capability to adjust to changes in storage system response characteristics. Given a sequence of performance values and

workload characteristics, calculating a new model is extremely cheap. In this paper, we have used all past data to update the model. In reality, we need keep only a subset of the data points. Furthermore, we need to guarantee the coverage of the workload space. If a particular type of workload is never seen in production, that workload may need to be injected into the data store to get fresh data points.

Another important future work is to determine better characterization parameters for the workload. A good characterization would simplify the workload aggregation while reducing the variability of accuracy across different data stores. For example, if a characteristics X had a significant effect on a particular data store and we did not include X in our characterization, then its effect is treated as noise in our model creating a significant noise. Another reason is that characteristics like randomness is inherently difficult to incorporate into a model since there are infinite number of ways to be *random*. By replacing it with one or more deterministic parameters, we believe we can simplify the aggregation function as well as improve the accuracy of the model.

Lastly, the convergence rate of load balancing scheme needs further investigation. Specifically, the effect of *temperature* function and G function on the performance of the algorithm needs further research. There are also other variations of the algorithm that do not randomly select the state to be evaluated which can also potentially increase the performance of the algorithm.

7. CONCLUSION

We have presented Romano, a load balancing framework for virtual disks on heterogeneous storage systems. The kernel of Romano is a performance predictor given a set of parameters that characterize the workloads and the storage devices. We have shown that Romano can outperform previous systems with same goals by up to 80% in prediction accuracy. This increased accuracy results in 78% reduction in maximum latency observed and 82% reduction in variance while load balancing.

At a deeper level, Romano contributes a recognition that there is inherent noise in storage system performance that is not easily dealt with. Romano is capable of capturing this noise within the prediction interval. As more data is gathered there is higher confidence that the prediction interval will contain the measured latency.

Another key contribution of Romano is quantification of effect of various workload characteristics and their interaction on heterogeneous storage devices. We have shown that the performance differences between data stores cannot be described with a single number. More specifically, we have shown that the storage performance must be described in terms of the workload.

The last contribution follows the second contribution. Since the performance of storage systems effectively change with the workload, the load balancing problem is no longer a bin-packing problem as previously believed. Therefore we present a probabilistic approach to finding a pseudo-optimal mapping of workloads to the data stores. It is important to mention that probabilistic approach is only used to find the final state of the system and the moves are actually made greedily to speed up the convergence and minimize the number of moves that has to be made.

We believe that Romano not only provides means for more efficient load balancing but also in many other areas such as QoS management, power management and performance tunning in tiered storage.

Whereas Romano raises the bar on the accuracy of practical modeling techniques, there remain ample opportunities for improvements in future work. Better interference modeling between workloads when they are placed on same data store is one such area where we are making good progress. Another work in progress is to come up with a better set of workload characteristics such that the workload description can be more complete.

8. REFERENCES

[1] Resource management with VMware DRS. `http://vmware.com/pdf/vmware_drs_wp.pdf`, 2006.

[2] Ahmad, I., Gulati, A., and Mashtizadeh, A. vic: Interrupt coalescing for virtual machine storage device io. In *Proceedings of the 2011 USENIX Annual Technical Conference* (2011), USENIX ATC'11, USENIX Association.

[3] Chatfield, C. Calculating interval forecasts. *Journal of Business & Economic Statistics* (1993), 121–135.

[4] Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., and Warfield, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation* (2005), NSDI'05, USENIX Association.

[5] Epping, D., and Denneman, F. *VMware vSphere 5 clustering: technical deepdive*. VMware, 2011.

[6] Granville, V., Krivánek, M., and Rasson, J. Simulated annealing: A proof of convergence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 16*, 6 (1994), 652–656.

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

[8] Gulati, A., Shanmuganathan, G., Ahmad, I., Waldspurger, C. A., and Uysal, M. Pesto: Online storage performance management in virtualized datacenters. In *Proceedings of the 2nd ACM Symposium on Clound Computing* (2011), SOCC'11, ACM.

[9] Hayashi, F. *Econometrics*. Princeton University Press, 2000.

[10] Kirkpatrick, S., Gelatt Jr, C., and Vecchi, M. Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680.

[11] Lilja, D. *Measuring computer performance: a practitioner's guide*. Cambridge Univ Press, 2005.

[12] Little, J. A proof for the queuing formula: L= λ w. *Operations research* (1961), 383–387.

[13] Liu, Z., Jiang, B., Zhao, Z., and Jiang, Y. Modeling and simulation of self-similar storage i/o. In *Proceedings of the 3rd International Conference on Advances in Grid and Pervasive Computing* (2008), GPC'08, IEEE.

[14] Mashtizadeh, A., Celebi, E., Garfinkel, T., and Cai, M. The design and evolution of live storage migration in vmware esx. In *Proceedings of the 2011 USENIX Annual Technical Conference* (2011), USENIX ATC'11, USENIX Association.

[15] Merrill, D. R. Storage economics: Four principles for reducing total cost of ownership. Tech. rep., Hitachi Data Systems, 2009. `http://www.hds.com/assets/pdf/four-principles-for-reducing-total-cost-of-ownership.pdf`.

[16] Nelson, M., Lim, B., and Hutchins, G. Fast transparent migration for virtual machines. In *Proceedings of the 2005 USENIX Annual Technical Conference* (2005), USENIX ATC'05, USENIX Association.

[17] Park, N., Xiao, W., Choi, K., and Lilja, D. A statistical evaluation of the impact of parameter selection on storage system benchmarks. In *7th IEEE International Workshop on Storage Network Architecture and Parallel I/O* (2011), SNAPI'11, IEEE.

[18] Seltzer, M., Krinsky, D., Smith, K., and Zhang, X. The case for application-specific benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on* (1999), IEEE, pp. 102–107.

[19] Sievert, J. Iometer: The I/O performance analysis tool for servers, 2004.

[20] Simpson, N. Building a data center cost model. Tech. rep., Burton Group, 2010. `http://www.burtongroup.com/Research/DocumentList.aspx?cid=49`.

[21] Singh, A., Korupolu, M., and Mohapatra, D. Server-storage virtualization: integration and load balancing in data centers. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (2008), SC'08, IEEE Press.

[22] Swaroop Kavalanekar, Bruce Worthingto, Q. Z., and Sharda, V. Characterization of storage workload traces from production windows servers. In *Proceedings of the 2008 IEEE International Symposium on Workload Characterization* (2008), IISWC'08, IEEE.

[23] Tarasov, V., Kumar, S., Ma, J., Hildebrand, D., Povzner, A., Kuenning, G., and Zadok, E. Extracting flexible, replayable models from large block traces.

[24] Wang, L. Workload configurations for typical enterprise workloads. Tech. rep., Microsoft, 2009. `http://blogs.msdn.com/b/tvoellm/archive/2009/05/07/useful-io-profiles-for-simulating-various-workloads.aspx`.

[25] Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C., and Ganger, G. Storage device performance prediction with cart models. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (2004), MASCOTS'04, IEEE.

[26] Wood, T., Shenoy, P., Venkataramani, A., and Yousif, M. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th conference on Symposium on Networked Systems Design & Implementation* (2007), USENIX Association.