Modeling the Relative Fitness of Storage

Michael P. Mesnier,[*] Matthew Wachs, Raja R. Sambasivan,
Alice X. Zheng, Gregory R. Ganger

Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

Relative fitness is a new black-box approach to modeling the performance of storage devices. In contrast with an absolute model that predicts the performance of a workload on a given storage device, a relative fitness model predicts performance *differences* between a pair of devices. There are two primary advantages to this approach. First, because a relative fitness model is constructed for a device pair, the application-device feedback of a closed workload can be captured (e.g., how the I/O arrival rate changes as the workload moves from device A to device B). Second, a relative fitness model allows performance and resource utilization to be used in place of workload characteristics. This is beneficial when workload characteristics are difficult to obtain or concisely express (e.g., rather than describe the spatio-temporal characteristics of a workload, one could use the observed cache behavior of device A to help predict the performance of B).

This paper describes the steps necessary to build a relative fitness model, with an approach that is general enough to be used with any black-box modeling technique. We compare relative fitness models and absolute models across a variety of workloads and storage devices. On average, relative fitness models predict bandwidth and throughput within 10–20% and can reduce prediction error by as much as a factor of two when compared to absolute models.

Categories and Subject Descriptors: I.6.5 [Model Development]: Modeling methodologies, D.4.8 [Performance]: Modeling and prediction, D.4.2 [Storage Management].

General Terms: Measurement, Performance.

Keywords: black-box, storage, modeling, CART.

1. INTRODUCTION

> *Relative fitness: the fitness of a genotype compared with another in the same gene system* [8].

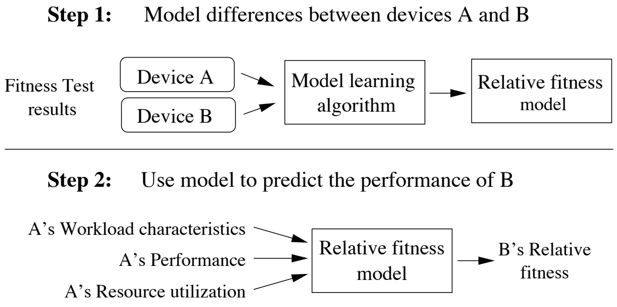


Figure 1: Relative fitness models predict changes in performance between two devices; an I/O "fitness test" is used to build a model of the performance differences. For new workloads, one inputs into the model the workload characteristics, performance, and resource utilization of a workload on device A to predict the relative fitness of device B.

Storage administration continues to be overly complex and costly. One challenging aspect of administering storage, particularly in large infrastructures, is deciding which application data sets to store on which devices. Among other things, this decision involves balancing loads, matching workload characteristics to device strengths, and ensuring that performance goals are satisfied. Storage administration currently relies on experts who use rules-of-thumb to make educated, but *ad hoc*, decisions. With a mechanism for predicting the performance of any given workload, one could begin to automate this process [1, 3, 5, 11].

Previous research on such prediction and automation focuses on per-device models that take as input workload characteristics (e.g., request arrival rate and read/write ratio) and output a performance prediction (e.g., throughput). Many envision these device models being constructed automatically in a black-box manner. That is, given predeployment measurements on a device, one can train a model to predict the performance of the device as a function of a workload's characteristics [2, 14, 27]. Such black-box models are *absolute* in the sense that they are trained to predict performance from assumedly static workload characteristics.

Though it sounds simple, the above approach has proven quite difficult to realize in practice, for several reasons. First, workload characterization has been an open problem for decades [9, 16, 27]; describing a complex workload in terms of concise characteristics, without losing necessary information, remains a challenge. Second, and more fundamentally,

[*] Intel and Carnegie Mellon University

SIGMETRICS'07, June 12–16, 2007, San Diego, California, USA.

an absolute model does not capture the connection between a workload and the storage device on which it executes. Generally speaking, application performance may depend on storage performance. If storage performance increases or decreases, the I/O rate of an application could also change. If such feedback is not accounted for, a model's applicability will be limited to environments where workloads are *open* (not affected by storage performance) or to devices that are similar enough that a workload's feedback will not change significantly when moving between them.

This paper proposes a new black-box approach, called *relative fitness* modeling, which removes the above constraints. In contrast with an absolute model, a relative fitness model predicts how a workload will change if moved from one device to another by comparing the devices' performance over a large number of workloads. This will naturally capture the effects of workload-device feedback. Further, since relative fitness models are constructed for pairs of devices, rather than one per device, they can take as input performance and resource utilization in addition to basic workload characteristics. In other words, the performance and resource utilization of one device can be used in predicting the performance of another. Often, such observations yield at least as much information as workload characteristics, but are much easier to obtain and describe. For example, although one may not know how to concisely describe access locality, a workload that experiences a high cache hit rate on one device may experience a high hit rate on another.

Relative fitness models can be used to solve storage administration problems in a similar way to absolute models. One would train models before deploying a new storage device. One would measure the characteristics, including the performance and resource utilization, of a workload on the device to which it was originally assigned. Then, one would use the model to predict the consequences of moving the workload to a different device.

This paper describes the mechanics of relative fitness models and evaluates their effectiveness. In contrast with per-device absolute models, we construct two relative fitness models for each pair of devices—one for translating measurements on the first to predictions on the second and one for going in the other direction. Our models capture the differences between devices and learn to predict performance scaling factors. For example, a relative fitness model from device j to device i might predict that device i is 30% faster than device j for sequential workloads.

We find that, for a given workload, the performance of one device is often the best predictor of the performance of another. For the workloads evaluated in the paper (synthetic I/O, Postmark, and TPC-C), prediction errors are reduced by as much as a factor of two when compared to an absolute model that only uses workload characteristics. For example, the prediction error for TPC-C is reduced from 50% to 20%. Overall, the error of our bandwidth and throughput models is usually within 10–20%.

The remainder of this paper is organized as follows. Section 2 describes black-box modeling in greater detail and motivates the case for relative fitness modeling. Section 3 describes the mechanics of relative fitness modeling. Section 4 constructs both absolute and relative fitness models and compares their prediction accuracy across a variety of workloads and storage devices.

2. BACKGROUND

Storage systems can be complex to manage. Management consists, among many other tasks, of device (aka volume or LUN) sizing, selecting a RAID level for each device, and mapping application workloads to the devices. Sadly, the state-of-the-art in storage management requires much of this to be done manually. Often, storage administrators use rules-of-thumb to determine the appropriate sizes, RAID levels, and workload-device mappings. At best, this results in an overworked administrator. However, it can also lead to suboptimal performance and wasted resources.

Automating management [1, 3, 11] is one way to offer the administrator some relief and help reduce the total cost of ownership in the data center. In particular, one could automate the mapping of workloads to storage devices. However, doing so efficiently requires accurate predictions as to how a workload will perform on a given storage device. Of course, not all devices will have the same performance characteristics. The challenge is predicting which workloads run best on which devices. A *model* of a storage device can be used to make these predictions.

2.1 Conventional modeling

Conventionally, a storage device model accepts as input the characteristics of an application workload and outputs an expected (predicted) performance, typically one of bandwidth, throughput, or latency [20]. For example, a disk drive model may predict an average throughput of 166 IO/sec for workloads with random I/O access patterns.

A variety of models have been researched, including analytical models [17, 22, 24], statistical or probabilistic models [14], models based on machine learning [27], and table-based models [2]. More generally, models are either *white-box* or *black-box*. Whereas white-box models use knowledge of the internals of a storage device (e.g., drives, controllers, caches), black-box models do not. Given the challenges in modeling modern-day (complex) storage devices [25], black-box approaches are attractive [2, 14, 27, 19].

Perhaps the simplest of all black-box models is a numeric average [20]. For example, the fuel efficiency of a car (average miles per gallon) and a soccer player's performance (average goals per game) are both black-box models. Of course, such models can be easily extended with workload characteristics (e.g., highway or city, home game or away), and an average can be maintained for each type of workload.

Table 1 shows a black-box (table-based) model of a storage device, and Figure 2 shows the same information modeled with a regression tree. Both models are indexed using only one workload characteristic (the request size) and must be *trained* with sample workloads in order to learn the performance averages for each workload type. Some form of interpolation is required when an exact match is not found in the model. For example, to predict the performance of a 3 KB request size, using Table 1, one might average the 2 KB and 4 KB performance and predict 37 MB/sec.

Of course, storage researchers have explored a number of workload characteristics in addition to request size, including the read/write ratio, multi-programming level (queue depth), I/O inter-arrival delay, and spatial locality. More complex characteristics such as the burstiness [28] or spatio-temporal correlation [26] have also been investigated.

More formally, a model of a storage device i (white-box or black-box) can be expressed as a function F_i. During

Request size	Bandwidth
1 KB	15 MB/sec
2 KB	27 MB/sec
4 KB	47 MB/sec
8 KB	66 MB/sec

Table 1: A table-based model that records the performance of a disk drive for sequentially-read data.

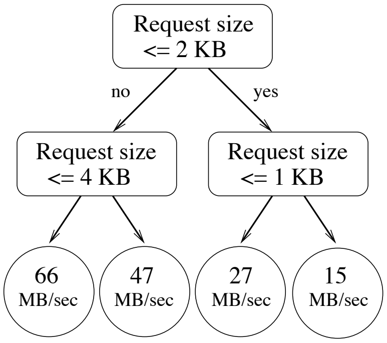


Figure 2: A regression tree that learns the performance of a disk drive for sequentially-read data.

training, the inputs are the workload characteristics $\mathbf{WC}_i$ of an application running on device i and the output is a performance metric P_i (bandwidth, throughput, or latency):

$$\mathrm{P}_i = F_i(\mathbf{WC}_i). \qquad (1)$$

However, in practice (during model testing), one does not possess $\mathbf{WC}_i$, as this would require running the workload on device i in order to obtain them. Because running the workload to obtain $\mathbf{WC}_i$ obviates the need for predicting the performance of device i, one instead uses the characteristics $\mathbf{WC}_j$ obtained from some other storage device j or, equivalently, from an I/O trace of device j.

We refer to Equation 1 as an *absolute model*, to signify that the inputs into the model are absolute, and not relative to some other device. Although the model expects $\mathbf{WC}_i$ as input, $\mathbf{WC}_j$ is used instead (a potential "type mismatch"). That is, the model assumes that the characteristics of a workload will not change across storage devices. More precisely, the absolute model assumes that $\mathbf{WC}_i = \mathbf{WC}_j$. However, as we will show in the evaluation, this is often not a safe assumption.

2.2 The challenges with absolute models

The primary challenges in building an absolute model all relate to workload characterization. First, one must discover the performance-affecting characteristics of a workload. This can be challenging given the heterogeneity of storage devices [15]. For example, a storage array with a large cache may be less sensitive to the spatial access pattern than an array with little cache, so models of the devices would likely focus on different workload characteristics when predicting performance. Therefore, to make predictions across a wide range of devices, the superset of workload characteristics must be maintained for each workload. This can be challenging, especially when obtaining the characteristics involves significant computation or memory.

A second challenge relates to the trade-off between expressiveness and conciseness. Because most models expect numbers as input, it can be challenging to express complex access patterns with just a few numbers. In effect, workload characterization compresses the I/O stream to just a few distinguishing features. The challenge is to compress the stream without losing too much information.

Finally, while the assumption $\mathbf{WC}_i = \mathbf{WC}_j$ is safe for *open* workloads, where the workload characteristics are independent of the I/O service time, it is not safe for *closed* workloads. The most obvious change for a closed workload is the I/O arrival rate: if a storage device completes the I/O faster, then an application is likely to issue I/O faster. Other characteristics can change too, such as the average request size, I/O randomness, read/write ratio, and queue depth. Such effects occur when file systems, page caches and other middleware sit between an application and the storage device. Although the application may issue the same I/O, the characteristics of the I/O as seen by the storage device could change due to write re-ordering, aggregation and coalescing, caching, pre-fetching and other interactions between an operating system and storage device. For example, a slower device can result in a workload with larger inter-arrival times and larger write requests (due to request coalescing) when compared to the same workload running on a faster device.

To illustrate how this could be problematic with an absolute model, consider the case where the average request size of a workload differs between two devices. Suppose the average request sizes for a given workload on devices j and i are 1 KB and 2 KB, respectively. If the workload characteristics measured on device j are input into the model for device i, the predicted performance (referring back to Table 1 or Figure 2) is 15 MB/sec instead of 27 MB/sec, which is the actual performance for a 2 KB request on device i. This hypothetical example illustrates the risk of indexing into a model with inaccurate workload characteristics. Given that most applications operate, at least in part, in a closed fashion [10], accounting for such changes in workload characteristics is essential when predicting their performance.

Collectively, these challenges motivate the work presented in this paper. Rather than present new workload characteristics that are expressive yet concise, and "absolute" across storage devices, we choose to user fewer in our models. In their place, we use performance and resource utilization. That is, we use the performance and utilization of device j to predict the performance of device i. Of course, doing so requires one to abandon the notion of an absolute model, as performance and resource utilization are device-specific.

3. RELATIVE FITNESS MODELS

The goal of relative fitness modeling is to use the performance and resource utilization of one device, in addition to workload characteristics, to predict the performance of another. The insight behind such an approach is best obtained through analogy: when predicting your performance in a college course (a useful prediction during enrollment), it is helpful to know the grade received by a peer and the number of hours he worked each week to achieve that grade (his resource utilization). Naturally, our own performance for certain tasks is a complex function of the characteristics of the task and our ability. However, we have learned to make predictions relative to the experiences of others with similar abilities, because it is easier.

Applying the analogy, two complex storage devices may behave similarly enough to be reasonable predictors for each other. For example, they may have similar RAID levels, caching algorithms, or hardware platforms. As such, their performance may be related. Even dissimilar devices may be related in some ways (e.g., for a given workload type, one usually performs well and the other poorly). The objective of relative fitness modeling is to learn such relationships.

Relative fitness models learn to predict scaling factors, or performance ratios, rather than performance itself. For example, device i may be 30% faster than device j for random reads (regardless of request size or other characteristics), 40% faster for sequential writes, and the same for all other workloads. Then, only three relative fitness values are needed to describe the performance of device i relative to device j: 1.3, 1.4, and 1.0. To predict the performance of device i, one simply multiplies the predicted scaling factor for device i by the performance of device j.

Starting with an absolute model, there are three additional steps to creating a relative fitness model. First, one must explicitly model the changes in workload characteristics between devices i and j. Second, in addition to workload characteristics, one trains a model of device i using the performance and resource utilization of device j. Third, rather than predict performance values, one trains a model to predict the performance ratio of device i to device j.

3.1 Deriving a relative fitness model

Relative fitness begins with an absolute model (Eq. 1). Recall that a workload is running on device j, we want to predict the performance of device i, and $\mathbf{WC}_j$ can be measured on device j. The first objective of relative fitness is to capture the changes in workload characteristics from device j to i, that is, to predict $\mathbf{WC}_i$ given $\mathbf{WC}_j$. Such change is dependent on the devices, so we define a function $G_{j\to i}$ that predicts the workload characteristics of device i given j:

$$\mathbf{WC}_i = G_{j\to i}(\mathbf{WC}_j).$$

We can now apply G in the context of an absolute model F_i:

$$\mathrm{P}_i = F_i(G_{j\to i}(\mathbf{WC}_j)).$$

However, rather than learn two functions, the composition of F and G can be expressed as a single composite function $RM_{j\to i}$ which we call a *relative model*:

$$\mathrm{P}_i = RM_{j\to i}(\mathbf{WC}_j). \qquad (2)$$

With each model now involving an origin j and target i, we can use the performance of device j ($\mathbf{Perf}_j$) and its resource utilization ($\mathbf{Util}_j$) to help predict the performance of device i. $\mathbf{Perf}_j$ is a vector of performance metrics such as bandwidth, throughput and latency. $\mathbf{Util}_j$ is a vector of values such as the device's cache utilization, the hit/miss ratio, its network bandwidth, and its CPU utilization:

$$\mathrm{P}_i = RM_{j\to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j). \qquad (3)$$

In other words, one can now describe a workload's behavior relative to some other device. Note that in conventional modeling, only one absolute model is trained for each device, thereby precluding the use of performance and resource utilization information, which are of course device-specific.

Next, rather than predict raw performance values, one can predict performance ratios. The expectation is that ratios are better predictors for new workloads, as they can naturally interpolate between known training samples. We call such a model a *relative fitness model*:

$$\frac{\mathrm{P}_i}{\mathrm{P}_j} = RF_{j\to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j). \qquad (4)$$

To use the relative fitness model, one solves for P_i:

$$\mathrm{P}_i = RF_{j\to i}(\mathbf{WC}_j, \mathbf{Perf}_j, \mathbf{Util}_j) \times \mathrm{P}_j.$$

Models based on Equations 1 through 4 are evaluated in Section 4. The relative model that only uses workload characteristics (Eq. 2) is included to separate the benefits of modeling changes in the workload characteristics from that of using performance to make predictions (Eqs. 3 and 4).

Discussion

In summary, whereas conventional black-box modeling constructs one absolute model per device and assumes that the workload characteristics are unchanging across devices, the relative approaches construct two models for each pair of devices (*A to B* and *B to A*) and implicitly model the changing workloads. In addition, the relative approaches use performance and resource utilization when making predictions, thereby relaxing the dependency on expressive workload characteristics. Of course, the cost of the relative approach is the additional model construction ($O(n^2)$ versus $O(n)$, where n is the number of storage devices). However, in our evaluation, model construction takes at most a few seconds. Moreover, models can be built and maintained by each storage device. That is, each device can construct $O(n)$ models that predict its fitness relative to all other devices. As such, the computational resources for maintaining the models can be made to scale with the number of devices. Also, in large-scale environments, certain collections of devices will be identical and can share models.

3.2 Training a relative fitness model

To train a model of a storage device, one needs *training data*, or samples that show how the performance of a storage device varies with workload characteristics. Samples can be obtained from applications, benchmarks, synthetic workload generators, or replayed I/O traces. The challenge, of course, is achieving adequate coverage. One must obtain samples that are representative of the workloads one expects to make predictions for, but determining this *a priori* can be difficult. Our approach is to use an I/O *fitness test*. A fitness test comprises multiple workload samples from a synthetic workload generator. The hope is that by exploring a wide range of workloads, one can train models that are general enough to make predictions for new workloads. More specifically, each workload sample created by the fitness test is described with three vectors: workload characteristics ($\mathbf{WC}_i$), performance ($\mathbf{Perf}_i$), and resource utilization ($\mathbf{Util}_i$). In the case of an absolute model, the goal is to learn relationships between $\mathbf{WC}_i$ and P_i (for some P_i in $\mathbf{Perf}_i$). Table 2(a) shows the training data for an absolute model. All but the last column are *predictor* variables. The last column is the *predicted* variable.

Relative models and relative fitness models, however, require workload samples from two devices. The goal is to learn relationships between the predictor variables ($\mathbf{WC}_j$, $\mathbf{Perf}_j$, and $\mathbf{Util}_j$) and the predicted variable P_i. Tables 2(b)

Sample	Predictor variables	Predicted var.
1	$\mathbf{WC}_{i,1}$	$\mathrm{P}_{i,1}$
2	$\mathbf{WC}_{i,2}$	$\mathrm{P}_{i,2}$
n	$\mathbf{WC}_{i,n}$	$\mathrm{P}_{i,n}$

(a) Absolute model

Sample	Predictor variables	Predicted var.
1	$\mathbf{WC}_{j,1}$ $\mathbf{Perf}_{j,1}$ $\mathbf{Util}_{j,1}$	$\mathrm{P}_{i,1}$
2	$\mathbf{WC}_{j,2}$ $\mathbf{Perf}_{j,2}$ $\mathbf{Util}_{j,2}$	$\mathrm{P}_{i,2}$
n	$\mathbf{WC}_{j,n}$ $\mathbf{Perf}_{j,n}$ $\mathbf{Util}_{j,n}$	$\mathrm{P}_{i,n}$

(b) Relative model

Sample	Predictor variables	Predicted var.
1	$\mathbf{WC}_{j,1}$ $\mathbf{Perf}_{j,1}$ $\mathbf{Util}_{j,1}$	$\mathrm{P}_{i,1}/\mathrm{P}_{j,1}$
2	$\mathbf{WC}_{j,2}$ $\mathbf{Perf}_{j,2}$ $\mathbf{Util}_{j,2}$	$\mathrm{P}_{i,2}/\mathrm{P}_{j,2}$
n	$\mathbf{WC}_{j,n}$ $\mathbf{Perf}_{j,n}$ $\mathbf{Util}_{j,n}$	$\mathrm{P}_{i,n}/\mathrm{P}_{j,n}$

(c) Relative fitness model

Table 2: Training data formats for the various models. The last column in each table is the variable that we train a model to predict. All other columns are predictor variables.

Request size	Queue depth	$RF_{j \to i}$
1 KB	1	.51
2 KB	2	.52
4 KB	1	.75
8 KB	2	.76

Table 3: An example of training samples that might be used to train a CART model.

and 2(c) show the format of the training data for the relative models and relative fitness models, respectively.

Given training data, one can construct a black-box model using a number of learning algorithms. The problem falls under the general scope of *supervised learning*, where one has access to a set of predictor variables, as well as the desired response (the predicted variable) [12]. It is as though an oracle (or supervisor), in this case the fitness test, gives us the true output value for each sample, and the algorithm needs only learn the mapping between input and output.

The domain of supervised learning problems can be further sub-divided into classification and regression. Classification deals with discrete-valued response variables. Regression deals with real-valued response variables (e.g., the performance or relative fitness values we train to predict). There are many regression models in the literature [12]. We focus on classification and regression tree (CART) models [6] for their simplicity, flexibility, and interpretability.

3.3 Reviewing CART

CART models predict a desired value based on predictor variables. In our case, the predictors are workload characteristics, performance, and resource utilization; and the predicted variable is a relative fitness value.

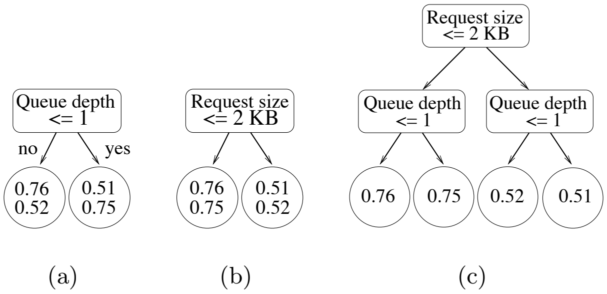


Figure 3: An example of the steps taken by CART in building a regression tree from the samples in Table 3. CART determines which split is "best" by inspecting the similarity of the samples in the leaf nodes. In this example, option b (request size) is a better first split than option a (queue depth). For the next split, CART then selects the queue depth.

Trees are built top-down, recursively, beginning with a root node. At each step in the recursion, the CART model-building algorithm determines which predictor variable in the training data best *splits* the current node into leaf nodes. The "best" split is that which minimizes the difference (e.g., root mean squared error, or RMS) among the samples in the leaf nodes, where the error for each sample is the difference between it and the average of all samples in the leaf node. In other words, a good split produces leaf nodes with samples that contain similar values. For example, consider the four training samples in Table 3. There are two predictor variables (request size and queue depth) and one predicted variable (the relative fitness value for device i when compared to device j). Splitting on the queue depth results in the tree shown in Figure 3(a). However, splitting on the request size, as shown in Figure 3(b), yields lower RMS error in the leaf nodes (i.e., the values in each leaf are more homogeneous). Therefore, the request size is the best first split. The CART algorithm then continues recursively on each subset. The same predictor variable may appear at different levels in the tree, as it may be best to split multiple times on that variable. In this case, the queue depth is the next best split, resulting in the tree shown in Figure 3(c).

Intuitively, the CART algorithm determines which of the predictor variables have the most "information" with respect to the predicted variable. The most relevant questions (splits) are asked first, and subsequent splits are used to refine the search. Trees are grown until no further splits are possible, thereby creating a *maximal tree*. A maximal tree is then *pruned* to eliminate over-fitting, and multiple pruning depths are explored. The optimal pruning depth for each branch is determined through *cross-validation* [18], in which a small amount of training data is reserved and used to test the accuracy of the pruned trees.

Finally, to make a prediction, one *drops* a new sample down a pruned tree (the CART model), following a path from the root node to a leaf node. As each node is visited, either the left or right branch is taken, depending on the splitting criterion and the values of the predictor variables for the sample. Leaf nodes contain the predicted variable that will be returned (the average of all samples in the leaf).

4. EVALUATION

Four model types are trained and tested: absolute models (Eq. 1), relative models (Eq. 2), relative models with performance (Eq. 3), and relative fitness models (Eq. 4). These models help support each of our research hypotheses:

Hypothesis 1 Workload characteristics can change across storage devices ($\mathbf{WC}_i \neq \mathbf{WC}_j$) and reduce the accuracy of an absolute model.

Hypothesis 2 A relative model (Eq. 2) can reduce inaccuracies that result from changing characteristics.

Hypothesis 3 Performance and resource utilization can improve prediction accuracy (Eq. 3).

Hypothesis 4 Performance ratios (Eq. 4) can provide better accuracy than raw performance values (Eq. 3).

This evaluation is organized as follows. Section 4.1 describes the experimental setup, 4.2 summarizes the characteristics of the I/O fitness test used to train the models, 4.3 describes how to interpret the models and use them to make predictions, and 4.4 evaluates the accuracy of each model.

4.1 Setup

Experiments are run on an IBM x345 server (dual 2.66 GHz Xeon, 1.5 GB RAM, GbE, Linux 2.6.12) attached to three Internet SCSI (iSCSI) [21] storage arrays. The arrays have different hardware platforms, software stacks, and are configured with different RAID levels.[1] More specifically,

VendorA is a 14-disk RAID-50 array with 1 GB of cache. (400GB 7200 RPM Hitachi Deskstar SATA)

VendorB is a 6-disk RAID-0 array with 512 MB of cache. (250GB 7200 RPM Seagate Barracuda SATA)

VendorC is an 8-disk RAID-10 array with 512 MB of cache. (250GB 7200 RPM Seagate Barracuda SATA)

The server attaches to each array using an iSCSI device driver [13] that contains counters (below the file system and page cache) for characterizing workloads and measuring their performance. Three workloads are used: a synthetic workload generator [13], TPC-C [23], and Postmark [4]. The workload generator serves as the *fitness test* used to both train and test the models. The remaining workloads are only used to test the models. All workloads run in an 8 GB Linux ext2 file system created on each array.

Obtaining training data

The fitness test compares the performance of the storage arrays across a wide range of workloads (samples created by the workload generator). For each sample, a file (approximately 8 GB in size) is created in an 8 GB partition of each array. I/O is then performed against this file with values chosen for the following parameters: read/write ratio, read request size, write request size, number of outstanding requests, read randomness, and write randomness. More precisely, the read and write request sizes take on a value from 1 KB to 256 KB (powers of two), the outstanding requests a value from 1 to 64 (powers of two), and the write percent and randomness one of 0%, 25%, 50%, 75% and 100%. For randomness, a value of $x\%$ means that $x\%$ of the I/Os are uniformly random, the rest sequential. Therefore, there are over 70,000 possible workloads ($9 \times 9 \times 7 \times 5 \times 5 \times 5$).

A total of 3,000 samples are generated against each array. Values for each of the above parameters are randomly selected, and the random number generator is seeded in order to create the same set of samples across the arrays. I/O is performed until all caches have been warmed, after which workload characteristics and performance are obtained.

The measured workload characteristics of each sample (**WC**) are similar to the parameters of the workload generator. They include the write percent, the write and read request sizes, the write and read randomness (average seek distance, in blocks, per I/O), and the queue depth (average number of outstanding I/Os). The performance of each sample (**Perf**) is the average bandwidth (MB/sec), throughput (IO/sec), and latency (msec). Resource utilization (**Util**) is not used in this evaluation, as this requires modifying storage device software to which we did not have access.

Training the models

One absolute model is trained for each array and performance metric, resulting in 9 absolute models. That is, one model is trained to predict the bandwidth of Vendor A, one is trained for throughput, and one is trained for latency; similarly for B and C. These models establish a baseline for measuring the improvements of the relative models and relative fitness models.

Two relative models that only use workload characteristics (Eq. 2) are trained for each pair of arrays (Vendor A to Vendor B, A to C, B to A, B to C, C to A, and C to B) for each performance metric, resulting in 18 relative models. That is, one model is trained to predict the bandwidth of Vendor A given the characteristics of a workload on Vendor B, one for throughput, and so on. Similarly, 18 relative models (Eq. 3) and 18 relative fitness models (Eq. 4) are trained with performance in addition to workload characteristics.

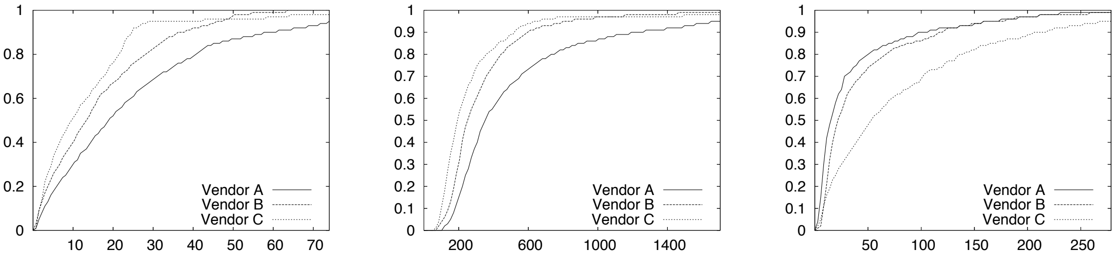
All models are trained using 4-fold cross validation [18]: the fitness data is divided into 4 folds (each containing 25% of the data, or 750 samples). Three of the folds are used to train a model (2,250 samples) and one is used for testing. Four such models are created, each one using a different fold for testing, and the prediction error for each fold is averaged to produce an overall error for the model. The error metric is discussed in Section 4.4 (Modeling accuracy).

4.2 Fitness test results

For all 3,000 samples, Figure 4 plots the cumulative distribution function (CDF) for bandwidth, throughput and latency. Vendor A is the fastest array with an average bandwidth of 25 MB/sec, an average throughput of 624 IO/sec and an average latency of 37 msec. Vendor B is the second fastest (17 MB/sec, 349 IO/sec, and 45 msec). Vendor C is third (14 MB/sec, 341 IO/sec, and 84 msec).

Although Vendor A is the fastest, on average, it is not necessarily the fastest for all sample workloads in the fitness test. To illustrate, Figure 5 contains probability distribution functions (PDF) of the relative fitness values (for throughput) for all samples. Recall that the relative fitness value is a performance ratio (scaling factor) between two devices for a given sample. As can be seen in the PDF, not all samples

[1] RAID level 0 is striping, 1 is mirroring, 5 is striping with parity, 10 is striping over mirrored pairs (4 in this case), and 50 is striping over RAID-5 parity arrays (2 in this case).



(a) Bandwidth (MB/sec)

(b) Throughput (IO/sec)

(c) Latency (msec)

Figure 4: Performance CDFs for bandwidth, throughput and latency. For the bandwidth and throughput CDFs, "further to the right" is faster. For latency, "further to the left" is faster.

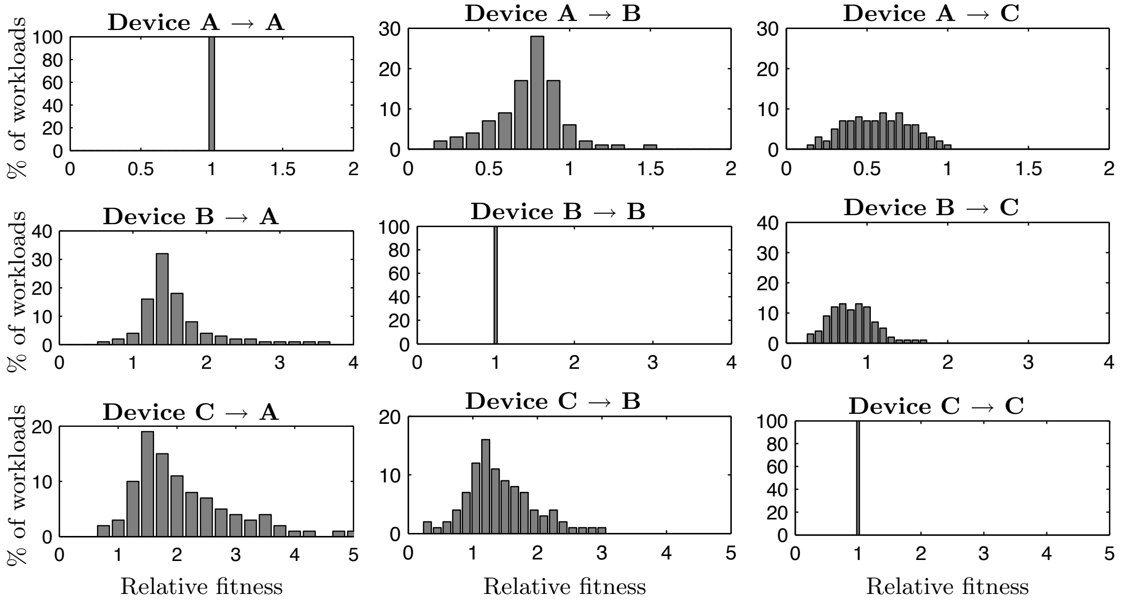


Figure 5: Probability distribution functions of the relative fitness values (throughput) for all device pairs.

share the same relative fitness value. For example, Figure 5 (see "Device $A \rightarrow B$") shows that for some of the samples Vendor B's throughput is approximately that of Vendor A. However, there are samples where Vendor B does better than A (relative fitness values greater than 1) and cases where it does worse (values less than 1). In short, the relative fitness of a device can vary with the workload characteristics.

As another example, Figure 6 illustrates how the sequential write bandwidth for each array varies for different request sizes and queue depths. From the 3,000 samples, we show only those with 100% writes and 0% randomness. There are 120 such samples, sorted by the performance of Vendor A. The graph illustrates the similar performance of the arrays. In particular, the prominent discontinuity in the graph is shared by all arrays (a drop in performance when there are only one or two outstanding requests). Also note how Vendor B is faster than Vendor C to the left of the discontinuity, but slower to the right. Such piecewise functions are ideally suited for algorithms such as CART.

Table 4 contains averages for the workload characteristics of each sample. Note the variance across devices ($\mathbf{WC}_i \neq \mathbf{WC}_j$), most notably the average write randomness which varies by as much as 38%. In particular, Vendor A experiences the most randomness (an average seek distance of 321 MB per write), Vendor B the second most (250 MB) and Vendor C third (233 MB). Although masked by the averages in Table 4, the request sizes and queue depths also vary across storage devices for some of the sample workloads. The extent to which such variance affects the performance predictions is explored further on in this evaluation.

In summary, the fitness test supports many of our hypotheses. In particular, we see that the relative fitness of one device to another will depend on the characteristics of the workload, that devices with different hardware and software configurations can still exhibit similar performance behavior, and that the characteristics of a workload can change as it moves across devices.

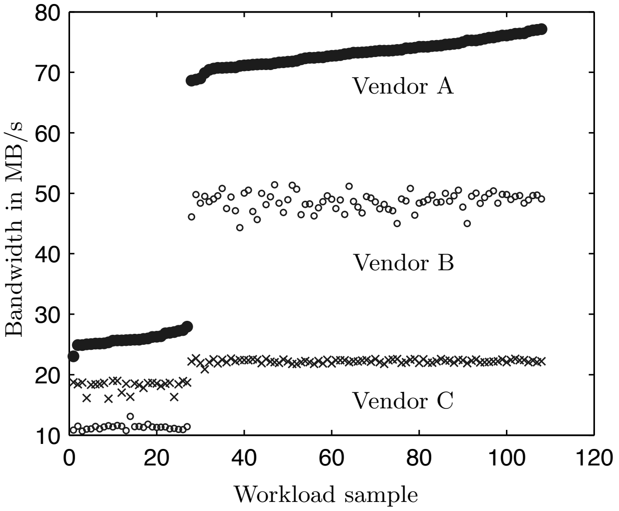


Figure 6: Device similarity. The performance of each array changes similarly, indicating that the performance of one array is a good predictor of another.

	Vendor			Maximum
WC	A	B	C	Difference
Write percent	40	39	38	5.2%
Write size (KB)	61	61	61	0%
Read size (KB)	40	41	41	2.5%
Write seek (MB)	321	250	233	**38%**
Read seek (MB)	710	711	711	0%
Queue depth	23	22	21	9.5%

Table 4: Fitness test workload characteristics.

4.3 Interpreting the models

Figure 7 illustrates four of the bandwidth models, one of each modeling type. The models are trained to predict the bandwidth of Vendor C using the workload characteristics of Vendor A (Eq. 1), the bandwidth of Vendor C using the workload characteristics of Vendor A (Eq. 2), and the bandwidth of Vendor C using the workload characteristics and performance of Vendor A (Eqs. 3 and 4). For readability, each tree is pruned to a depth of 4, resulting in at most 8 leaves (prediction rules).

The CART models are human readable and very easy to interpret. In effect, they represent a summary of the fitness test results. Recall that CART builds trees top-down, so nodes near the top of the tree have more information with respect to the predicted variable when compared to those near the bottom.

A couple of observations can be made from the structure of the trees. First, the absolute model F_i (Figure 7a) and relative model $RM_{j,i}$ (7b) are very similar; both find the average read seek distance to have the strongest relationship with the bandwidth of Vendor C and use the request size and write percent to further refine the search. The slight differences in the trees are due to differences in the fitness test workload characteristics between Vendors A and C (supporting hypothesis 1). Recall that even though the fitness test samples used to create the models are identical, the measured workload characteristics for each sample can vary across devices. These differences can result in different splits in the trees during model training. The second observation is that the relative model trained with performance (7c) and the relative fitness model (7d) both place performance at the top of the tree (supporting hypothesis 2). Namely, CART discovers that the bandwidth of Vendor A is the best predictor of the bandwidth of Vendor C.

As a hypothetical example of how to use the trees to make a prediction, suppose we have a workload running on the Vendor A array and want to predict its performance on Vendor C. Also suppose that the workload, as measured by Vendor A, has an average read seek distance of 2048 blocks (1 MB), a read and write size of 64 KB, a write percentage $< .5\%$, a bandwidth of 83 MB/sec, and a throughput of 1328 IO/sec. Based on these values, the absolute model for Vendor C will predict 75.0 MB/sec (see the highlighted path in Figure 7a). The relative model from Vendor A to C (Figure 7b) also predicts about 75 MB/sec. The relative model trained with performance (Figure 7c), however, predicts 65.0 MB/sec. Finally, the relative fitness model (Figure 7d) predicts a relative fitness value of 0.63; the predicted bandwidth of Vendor C is therefore 63% of Vendor A, or 51 MB/sec. This hypothetical example illustrates that the models can make different predictions, even with the same input.

4.4 Modeling accuracy

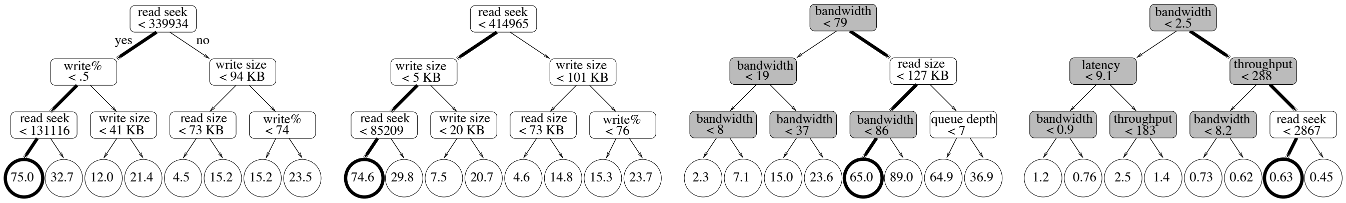
The models are tested using three workloads: the fitness test, TPC-C, and Postmark. The error for each model is the average relative error of its predictions, expressed as a percentage. For example, if the actual performance of Vendor C, using the previous example, is 45 MB/sec and the prediction is 51 MB/sec, then the relative error is $\frac{|45-51|}{45} \times 100$, or 13.3%. Previous modeling approaches have reported prediction errors in the range of 15% to 30% [2, 27]. The goal of this evaluation is to first train an absolute model that predicts in the 15% to 30% range when the workload is characterized on the same device for which the prediction is being made (i.e., $\mathbf{WC}_i = \mathbf{WC}_j$). That is, we want a fair representative of an absolute model. Then, we will show that by characterizing the workload on another storage device ($\mathbf{WC}_i \neq \mathbf{WC}_j$), prediction error increases due to changing workload characteristics. Next, we will show that adding performance information to the training data improves performance. Finally, we will show that predicting performance ratios can be more accurate than raw performance values.

Fitness test predictions

The fitness test prediction error is that from cross-validation. With 4-fold cross-validation, four different trees are built, each holding out and testing against a different fold that contains 25% of the training samples. That is, each tree is built with 75% of the fitness samples and tested against 25% of the samples. The error is the average prediction error across all folds (i.e., an average over 3,000 predictions).

As a baseline, Table 5 contains the average relative error of the bandwidth, throughput, and latency predictions for the absolute model. The table is organized pairwise. Workloads characteristics ($\mathbf{WC}_j$) are obtained from one array and predictions (P_i) are made for another. For example, the average relative error of the bandwidth predictions when characterizing on Vendor A and predicting for Vendor C is 22% (the top right cell in Table 5).

The first observation is that the most accurate predictions



(a) Absolute (b) Relative (c) Relative with perf. (d) Relative fitness

Figure 7: CART models trained to predict the bandwidth of Vendor C for a workload running on Vendor A. The absolute and relative model only use workload characteristics as measured by Vendor A. The relative model (with perf.) and the relative fitness model also use performance information (shaded). Note, the leaf nodes in all but the relative fitness model represent bandwidth predictions. For the relative fitness model, the leaf nodes represent bandwidth scaling factors relative to Vendor A.

$\mathbf{WC}_j$	P_i (Bandwidth)		
	Vendor A	Vendor B	Vendor C
Vendor A	**23%**	25%	22%
Vendor B	29%	**19%**	21%
Vendor C	30%	25%	**17%**

$\mathbf{WC}_j$	P_i (Throughput)		
	Vendor A	Vendor B	Vendor C
Vendor A	**20%**	23%	22%
Vendor B	28%	**15%**	21%
Vendor C	26%	21%	**14%**

$\mathbf{WC}_j$	P_i (Latency)		
	Vendor A	Vendor B	Vendor C
Vendor A	**20%**	39%	59%
Vendor B	31%	**21%**	52%
Vendor C	26%	30%	**21%**

Table 5: Fitness test prediction error for the absolute model (Eq. 1). The workloads characteristics ($\mathbf{WC}_j$) are obtained from array j and the predictions ($\mathbf{P}_i$) are made for device i.

$\mathbf{WC}_j$	P_i (Bandwidth)		
	Vendor A	Vendor B	Vendor C
Vendor A	<1%	15%,14%	18%,16%
Vendor B	18%,18%	<1%	17%,16%
Vendor C	22%,21%	19%,16%	<1%

$\mathbf{WC}_j$	P_i (Throughput)		
	Vendor A	Vendor B	Vendor C
Vendor A	<1%	16%,15%	19%,19%
Vendor B	22%,20%	<1%	17%,18%
Vendor C	25%,24%	19%,18%	<1%

$\mathbf{WC}_j$	P_i (Latency)		
	Vendor A	Vendor B	Vendor C
Vendor A	<1%	28%,27%	39%,38%
Vendor B	20%,18%	<1%	42%,44%
Vendor C	21%,21%	29%,27%	<1%

Table 6: Fitness test prediction errors for the relative model (Eq. 3, shown first) and the relative fitness model (Eq. 4), both trained with performance. Workload characteristics and performance are measured on one array, predictions made for another.

occur when the workload is characterized on the same device for which the prediction is being made ($\mathbf{WC}_j = \mathbf{WC}_i$), as indicated by the diagonals in bold. The prediction errors are all in the 15–30% range and are consistent with previous black-box modeling approaches.

Of course, if one runs a workload on device i to obtain the workload characteristics $\mathbf{WC}_i$, there is no need to make a prediction. These predictions are only included to illustrate how changing workload characteristics ($\mathbf{WC}_j \neq \mathbf{WC}_i$) can affect the prediction accuracy. As examples, the bandwidth prediction error for Vendor A increases from 23% to 29% and 30% when the workload is characterized on Vendors B and C, respectively; throughput predictions for Vendor B increase from 15% to 23% and 21%; and latency predictions for Vendor C increase from 21% to 59% and 52%. Overall, Table 5 confirms hypothesis 1: changing workload characteristics can affect the accuracy of an absolute model.

Table 6, in comparison, shows the fitness test prediction errors for the relative model (Eq. 3) and relative fitness model (Eq. 4), both of which use performance information to make a prediction. Note that because the models use the performance of device j to predict that of device i, the error along the diagonal ($i = j$) represents the slight modeling error when using the performance of device i to predict the performance of device i. Again, in practice one would not need to make such predictions. The values off of the diagonal, however, can be directly compared to those in Table 5. Overall, the relative fitness models reduce the maximum bandwidth prediction error from 30% to 21%, throughput from 28% to 24%, and latency from 59% to 44%. Moreover, in most cases, the relative fitness model is slightly more accurate than the relative model. These results confirm that models trained with performance can be more accurate (hypotheses 2 and 3) and that predicting performance ratios can further improve prediction accuracy (hypothesis 4).

The results in Tables 5 and 6 are obtained from trees pruned to a depth of 12. However, large trees can potentially over-fit the training data, resulting in trees that are specific to the behavior of the fitness test. Therefore, shorter trees should be used whenever possible, as they are more general. Note, a tree pruned to a depth of one corresponds to a single-node tree. In the case of absolute and relative models, it corresponds to the average performance of a de-

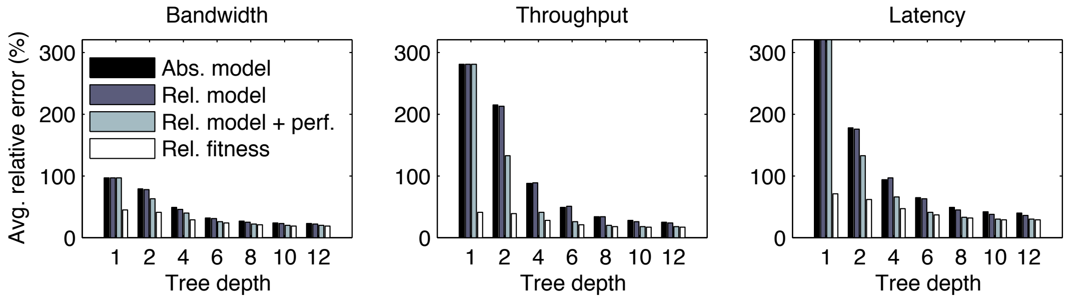
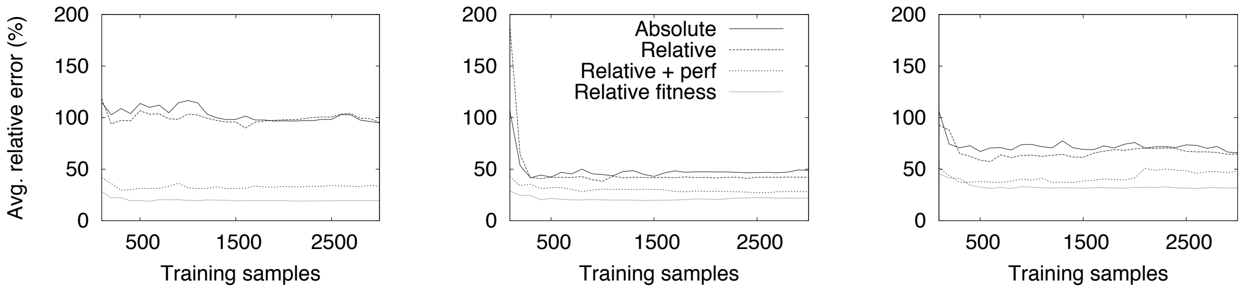


Figure 8: Accuracy varies with tree depth. At all tree depths, the relative fitness models are the most accurate, suggesting that relative fitness is a more efficient/general approach to modeling performance.



(a) Bandwidth

(b) Throughput

(c) Latency

Figure 9: Accuracy versus number of training samples. The top line represents the absolute model, the next line is a relative model that captures changes in the workload characteristics. The third line is a relative model trained with performance. The fourth line (least error) is relative fitness. As the number of training samples increases, the relative fitness models are the most stable and also the most accurate.

vice for the fitness test. For relative fitness, it is the average scaling factor for a specific device pair. Figure 8 shows the prediction error for various tree depths. At every depth, relative fitness is more accurate than an absolute model. For example, at a depth of 6 (at most 32 leaf nodes), the relative fitness model achieves an average bandwidth prediction error of 21%, compared to 49% for the absolute model. This result suggests that relative fitness models are more easily pruned, hence more efficient (general) learners.

The last result from the fitness test compares the prediction error as the number of training samples is increased. For each sample size, 4-fold cross validation is used to evaluate the accuracy of the models. In this test, the relative model that only uses workload characteristics (Eq. 2) is used to illustrate 1) how the changing workload characteristics can affect prediction accuracy of an absolute model and 2) the improvement in accuracy when performance information is used in the prediction (Eq. 3). Figure 9 shows how the prediction error varies with the number of training samples (all trees are pruned to a depth of 4). The topmost line (most error) represents the error of the absolute model, the next line represents the improvement from modeling workload changes (hypothesis 2), the third line from the top shows the large reduction in error by adding performance information (hypothesis 3), and the last line (least error) is that of relative fitness (hypothesis 4).

In summary, the fitness test results supports all four hypotheses. Namely, workload characteristics can change across devices and the change is enough to impact the accuracy of an absolute model; a relative model can reduce the inaccuracy due to changing workloads; performance can be used to predict performance; and relative fitness values can be better predictors than raw performance values.

TPC-C & Postmark predictions

TPC Benchmark C [23] is an OLTP benchmark, including order entry, payment, order status, and monitoring inventory. We run TPC-C on the Shore storage manager [7], configured with 8 KB pages, 10 warehouses, and 10 clients; the footprint is 5 GB. We characterize the workload and performance of TPC-C on each of the arrays and use this information to predict the throughput on the other two arrays. Similarly, Postmark performs transactions against a pool of files, including create, delete, read, or write. We configure Postmark with 100k files, 100k transactions, and 100 subdirectories. File size ranges from 1K to 16K. Workload characteristics and performance are measured after the files are created and transactions have begun. TPC-C and Postmark are run until the device caches have been warmed and throughput reaches a steady state. We then characterize the workload and measure the performance. The models are used to predict the steady-state throughput of each array.

	TPC-C			Postmark		
WC	A	B	C	A	B	C
Write percent	24	13	24	71	55	62
Write size (KB)	8	9	8	11	8	8
Read size (KB)	8	8	8	7	7	7
Write seek (GB)	.2	.1	.2	.4	.4	.3
Read seek (GB)	.3	.3	.3	5.3	5.2	5.4
Queue depth	2	1	1	44	17	22

	TPC-C			Postmark		
Perf	A	B	C	A	B	C
Bandwidth (MB/s)	1.7	2.4	2.5	3.8	1.4	1.4
Throughput (IO/s)	271	127	173	398	186	186
Latency (ms)	2	11	7	7	50	54

Table 7: Workload characteristics and performance averages for TPC-C and Postmark (by Vendor).

Table 7 contains the workload characteristics and performance averages for each array. The variation in workload characteristics is much more pronounced than with the fitness test. In particular, the write percent of TPC-C varies by up to 85%, and write size by 13%. Similarly, the write percent of Postmark varies by as much as 29%, write size by 38%, write randomness by 31% (Vendor A has a 429 MB average seek per write, Vendor B 385 MB, and Vendor C 328 MB), and queue depth by over a factor of 2. These averages are obtained from at least three runs, again highlighting that $\mathbf{WC}_i = \mathbf{WC}_j$ is not a safe assumption.

Three models are evaluated: an absolute model, a relative model, and a relative fitness model. The first of the four trees created from the fitness test during cross-validation is chosen (arbitrarily) to make these predictions. The relative and relative fitness models used to predict TPC-C are only trained with performance, as predictions actually improve by omitting workload characteristics entirely. That is, for TPC-C, the bandwidth, throughput and latency of device j are the best predictors of the throughput of device i.

Figure 10 graphs the average prediction error for each model across a variety of tree depths. Each bar represents an average over the 6 predictions (i.e., characterize on A and predict for B and C, characterize on B and predict for A and C, and so on). At nearly all depths the relative and relative fitness models predict more accurately than the absolute model. For TPC-C, the lowest error for the absolute model is 50% at a tree depth of 4, compared to 20% for a relative model (depth 6), and 26% for relative fitness (depth 6). For Postmark, the lowest error for the absolute model is 27% (depth 10), compared to 30% for a relative model (depth 10) and 20% for a relative fitness model (depth 8). Although the absolute model is more accurate than the relative model for this particular tree depth, on average, the relative model is still more accurate.

As these tests show, changing workload characteristics can have a significant impact on prediction accuracy (hypothesis 1), so significant that removing them entirely (in the case of TPC-C) actually reduces prediction error. In their place performance information can be used (hypothesis 3), resulting in significant reductions in error. As for hypothesis 2, TPC-C suggests that even a relative model has difficulty modeling the change in workload characteristics; fortunately (for the relative and relative fitness models), performance information can be used in place of workload characteristics. Further, the TPC-C result suggests that a relative fitness model is not strictly better than a relative model. Although both do well, the relative model did slightly better than the relative fitness model for the TPC-C prediction.

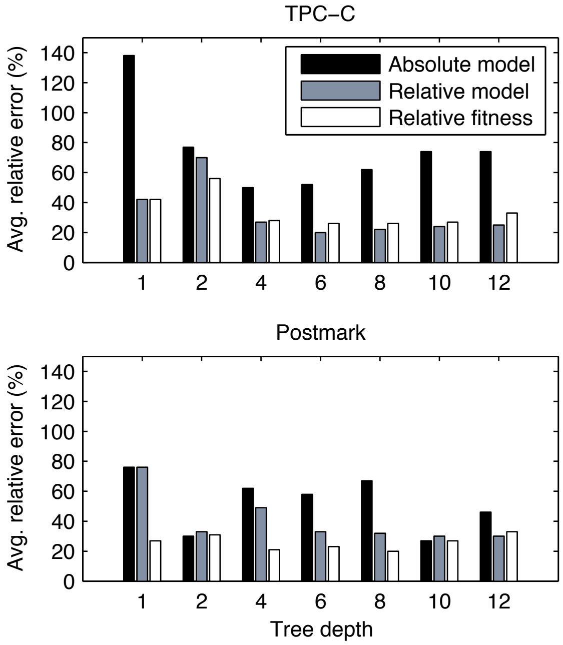


Figure 10: Average throughput prediction errors for TPC-C and Postmark across various tree depths.

5. CONCLUSION

Relative fitness is a promising new approach to modeling the performance of storage devices. In contrast with an absolute model, a relative fitness model predicts performance differences between device pairs. As such, application-device feedback can be modeled, and performance and resource utilization can be used when making predictions.

This paper describes the steps necessary to build a relative fitness model, with an approach that is general enough to be used with any black-box modeling technique. We compare relative fitness models and absolute models across a variety of workloads and storage devices. On average, the relative fitness models predict bandwidth and throughput within 10–20% and reduce the prediction error by as much as a factor of two when compared to an absolute model.

Acknowledgements

We thank Michael Stroucken (CMU) for his support and a flawlessly run machine room, and Eno Thereska (CMU) for providing the TPC-C infrastructure. We thank Christos Faloutsos (CMU), Arif Merchant (HP), and Mic Bowman (Intel) for their valuable feedback. We thank the members and companies of the PDL Consortium (APC, Cisco, EMC, Google, Hewlett-Packard, Hitachi, IBM, Intel, Network Appliance, Oracle, Panasas, Seagate, and Symantec) for their interest, insight, feedback, and support. We also thank EqualLogic, Intel, IBM, LeftHand Networks, Network Appliance, Open-E, Seagate, and Sun for hardware and software donations that enabled this work. This research is sponsored in part by the National Science Foundation, via grants #CNS-0326453, #CCF-0621499, and 0431008; and by the Army Research Office, under agreement DAAD19–02–1–0389. Matthew Wachs is supported in part by an NDSEG Fellowship from the Department of Defense.

6. REFERENCES

[1] G. A. Alvarez, J. Wilkes, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, and A. Veitch. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, **19**(4):483–518. ACM, November 2001.

[2] E. Anderson. *Simple table-based modeling of storage devices*. SSP Technical Report HPL–SSP–2001–4. HP Laboratories, July 2001.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 175–188. USENIX Association, 2002.

[4] N. Appliance. PostMark: A New File System Benchmark. http://www.netapp.com.

[5] E. Borowsky, R. Golding, A. Merchant, L. Schreier, E. Shriver, M. Spasojevic, and J. Wilkes. Using attribute-managed storage to achieve QoS. *International Workshop on Quality of Service* (Pittsburgh, PA, 21–23 March 1997). IFIP, 1997.

[6] L. Breiman, J. H.Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth.

[7] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring up persistent applications. *ACM SIGMOD International Conference on Management of Data* (Minneapolis, MN, 24–27 May 1994). Published as *SIGMOD Record*, **23**(2):383–394. ACM Press, 1994.

[8] D. J. Futuyma. *Evolutionary Biology. Third edition.* SUNY, Stony Brook. Sinauer. December 1998.

[9] G. R. Ganger. Generating representative synthetic workloads: an unsolved problem. *International Conference on Management and Performance Evaluation of Computer Systems* (Nashville, TN), pages 1263–1269, 1995.

[10] G. R. Ganger and Y. N. Patt. Using system-level models to evaluate I/O subsystem designs. *IEEE Transactions on Computers*, **47**(6):667–678, June 1998.

[11] G. R. Ganger, J. D. Strunk, and A. J. Klosterman. *Self-* Storage: brick-based storage with automated administration*. Technical Report CMU–CS–03–178. Carnegie Mellon University, August 2003.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer–Verlag. 2001.

[13] Intel. iSCSI. www.sourceforge.net/projects/intel-iscsi.

[14] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton. *Inducing models of black-box storage arrays*. Technical report HPL-2004-108. HP, June 2004.

[15] Z. Kurmas and K. Keeton. Using the distiller to direct the development of self-configuration software. *International Conference on Autonomic Computing* (New York, NY, 17–18 May 2004), pages 172–179. IEEE, 2004.

[16] Z. Kurmas, K. Keeton, and K. Mackenzie. Synthesizing representative I/O workloads using iterative distillation. *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Orlando, FL, 12–15 October 2003). IEEE/ACM, 2003.

[17] A. Merchant and P. S. Yu. Analytic modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Transactions on Computers*, **45**(3):367–373, March 1996.

[18] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[19] F. I. Popovici, A. C. A. Dusseau, and R. H. A. Dusseau. Robust, portable I/O scheduling with the disk mimic. *USENIX Annual Technical Conference* (San Antonio, TX, 09–14 June 2003), pages 297–310. IEEE, 2003.

[20] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, **27**(3):17–28, March 1994.

[21] J. Satran. iSCSI. http://www.ietf.org/rfc/rfc3720.txt.

[22] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Madison, WI, 22–26 June 1999). Published as *ACM SIGMETRICS Performance Evaluation Review*, **26**(1):182–191. ACM Press, 1990.

[23] Transaction Processing Performance Council. TPC Benchmark C. http://www.tpc.org/tpcc.

[24] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Cincinnati, OH, 15–18 August 2001), pages 183–192. IEEE, 2001.

[25] E. Varki, A. Merchant, J. Xu, and X. Qiu. Issues and challenges in the performance analysis of real disk arrays. *Transactions on Parallel and Distributed Systems*, **15**(6):559–574. IEEE, June 2004.

[26] M. Wang, A. Ailamaki, and C. Faloutsos. Capturing the spatio-temporal behavior of real traffic data. *IFIP WG 7.3 Symposium on Computer Performance* (Rome, Italy, 23–27 September 2002). Published as *Performance Evaluation*, **49**(1–4):147–163, 2002.

[27] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with CART models. *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Volendam, The Netherlands, 05–07 October 2004), pages 588–595. IEEE/ACM, 2004.

[28] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: fast algorithms for modeling bursty traffic. *International Conference on Data Engineering* (San Jose, CA, 26–01 March 2002), pages 507–516. IEEE, 2002.