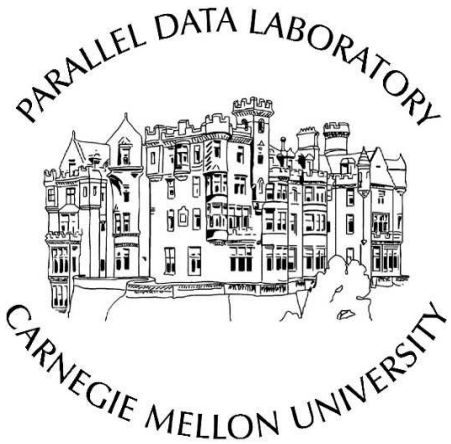


Storage Device Performance Prediction with CART Models

Mengzhi Wang, Kinman Au, Anastassia Ailamaki,
Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger

CMU-PDL-04-103
March 2004

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Storage device performance prediction is a key element of self-managed storage systems and application planning tasks, such as data assignment. This work explores the application of a machine learning tool, CART models, to storage device modeling. Our approach predicts a device's performance as a function of input workloads, requiring no knowledge of the device internals. We propose two uses of CART models: one that predicts per-request response times (and then derives aggregate values) and one that predicts aggregate values directly from workload characteristics. After being trained on our experimental platforms, both provide accurate black-box models across a range of test traces from real environments. Experiments show that these models predict the average and 90th percentile response time with an relative error as low as 16%, when the training workloads are similar to the testing workloads, and interpolate well across different workloads.

Acknowledgements: We thank the members and companies of the PDL Consortium (including EMC, Hewlett-Packard, Hitachi, Hitachi Global Storage Technologies, IBM, Intel, LSI Logic, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. We thank IBM for partly funding this work through a CAS student fellowship and a faculty partnership award. This work is funded in part by NSF grants CCR-0205544, IIS-0133686, BES-0329549, IIS-0083148, IIS-0113089, IIS-0209107, and IIS-0205224. We would also like to thank Eno Thereska, Mike Mesnier, and John Strunk for their participation and discussion in the early stage of this project.

Keywords: Performance prediction, storage device modeling

1 Introduction

The costs and complexity of system administration in storage systems [17, 35, 11] and database systems [12, 1, 15, 21] make automation of administration tasks a critical research challenge. One important aspect of administering self-managed storage systems, particularly large storage infrastructures, is deciding which data sets to store on which devices. To find an optimal or near optimal solution requires the ability to predict how well each device will serve each workload, so that loads can be balanced and particularly good matches can be exploited.

Researchers have long utilized performance models for such prediction to compare alternative storage device designs. Given sufficient effort and expertise, accurate simulations (e.g., [5, 28]) or analytic models (e.g., [22, 30, 31]) can be generated to explore design questions for a particular device. Unfortunately, in practice, such time and expertise is not available for deployed infrastructures, which are often comprised of numerous and distinct device types, and their administrators have neither the time nor the expertise needed to configure device models.

This paper attacks this obstacle by providing a black-box model generation algorithm. By "black box," we mean that the model (and model generation system) has no information about the internal components or algorithms of the storage device. Given access to a device for some "training period," the model generation system learns a device's behavior as a function of input workloads. The resulting device model approximates this function using existing machine learning tools. Our approach employs the Classification And Regression Trees (CART) tool because of its efficiency and accuracy. CART models, in a nutshell, approximate functions on a multi-dimensional Cartesian space using piece-wise constant functions.

Such learning-based black box modeling is difficult for two reasons. First, all the machine learning tools we have examined use vectors of scalars as input. Existing workload characterization models, however, involve parameters of empirical distributions. Compressing these distributions into a set of scalars is not straightforward. Second, the quality of the generated models depends highly on the quality of the training workloads. The training workloads should be diverse enough to provide high coverage of the input space.

This work develops two ways of encoding workloads as vectors: a vector per request or a vector per workload. The two encoding schemes lead to two types of device models, operating at the per-request and per-workload granularities, respectively. The request-level device models predict each request's response time based on its per-request vector, or "request description." The workload-level device models, on the other hand, predict aggregate performance directly from per-workload vectors, or "workload descriptions." Our experiments on a variety of real world workloads have shown that these descriptions are reasonably good at capturing workload performance from both single disks and disk arrays. The two CART-based models have a median relative error of 17% and 38%, respectively, for average response time prediction, and 18% and 43% respectively for the 90th percentile, when the training and testing traces come from the same workload. The CART-based models also interpolate well across workloads.

The remainder of this paper is organized as follows. Section 2 discusses previous work in the area of storage device modeling and workload characterization. Section 3 describes CART and its properties. Section 4 describes two CART-based device models. Section 5 evaluates the models using several real-world workload traces. Section 6 concludes the paper.

2 Related Work

Performance modeling has a long and successful history. Almost always, however, thorough knowledge of the system being modeled is assumed. Disk simulators, such as Pantheon [33] and DiskSim [5], use software to simulate storage device behavior and produce accurate per-request response times. Developing such simulators is challenging, especially when disk parameters are not publicly available. Predicting performance

using simulators is also resource intensive. Analytical models [7, 22, 24, 30, 31] are more computationally efficient because these models describe device behavior with a set of formulae. Finding the formula set requires deep understanding of the interaction between storage devices and workloads. In addition, both disk simulators and analytical models are tightly coupled with the modeled device. Therefore, new device technologies may invalidate existing models and require a new round of model building.

Our approach uses CART, which treats storage devices as black boxes. As a result, the model construction algorithm is fully automated and should be general enough to handle any type of storage device. The degenerate forms of "black-box models" are performance specifications, such as the maximum throughput of the devices, published by device manufacturers. The actual performance, however, will be nowhere near these numbers under some workloads. Anderson's "table-based" approach [3] includes workload characteristics in the model input. The table-based models remember device behavior for a wide range of workload and device pairs and interploates among tables entries in predicting. Anderson's models are used in an automated storage provision tool, Ergastulum [2], which formulates automatic storage infrastructure provisioning as an optimization problem and uses device models to guide the search algorithm in locating the solution. Our approach improves on the table-based models by employing machine learning tools to capture device behavior. Because of the good scalability of the tools to high dimensional datasets, we are able to use more sophisticated workload characteristics as the model input. As a result, the models are more efficient in both computation and storage.

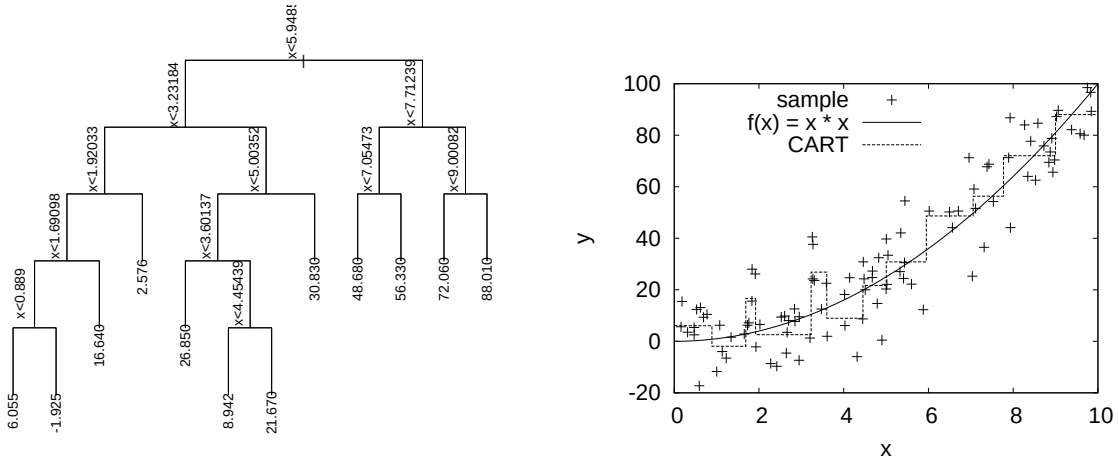
Workload characterization is an important part of device modeling because it provides a suitable representation of workloads. Despite abundant published work in modeling web traffic [23, 25, 8], I/O traffic modeling receives less attention. Direct application of web traffic analysis methods to I/O workloads is not adequate because of the different locality models. Network traffic has a categorical address space, and there is no notion of sequential scans. In contrast, the performance variability can be several orders of magnitude between random and sequential accesses for I/O workloads. Ganger [10] pointed out the complexity of I/O workloads, and even the detection of sequential scans is a hard problem [19]. Gomez et al. [14] identified self-similarity in I/O traffic and adopted structural models to generate I/O workloads. Kurmas et al. [20] employed an iterative approach to detect important workload characteristics. Rome [34] provided a general framework of workload specifications. All the approaches, in one way or another, use empirical distributions derived from given workloads as the parameter values. Our previous work [32] takes advantage of the self-similarity of I/O workloads and proposes a tool, the "entropy plot," to characterize the spatio-temporal characteristics of I/O workloads with three scalars. Since our CART-based models require workloads to be presented in the form of vectors of scalars, the entropy plot is an attractive choice.

3 Background: CART Models

This section gives a brief introduction of the CART models and justifies our choice of the tool. A detailed discussion of CART is available in [4].

3.1 CART Models

CART modeling is a machine learning tool that can approximate real functions in multi-dimensional Cartesian space. (It can also be thought of as a type of non-linear regression.) Given a function $Y = f(X) + \varepsilon$, where $X \in \Re^d$, $Y \in \Re$, and ε is zero-mean noise, a CART model approximates Y using a piece-wise constant function, $\hat{Y} = \hat{f}(X)$. We refer to the components of X as features in the following text. The term, ε, captures the intrinsic randomness of the data and the variability contributed by the unobservable variables. The variance of the noise could be dependent on X. For example, the variance of response time often depends on the arrival rate.



(a) Fitted tree (b) Data points and regression line

Figure 1: CART model for a simple one-dimensional data set. The data set contains 100 data points generated using $f(x) = x^2 + \varepsilon$, where ε follows a Guassian distribution with mean 0 and standard deviation 10.

The piece-wise constant function $\hat{f}(X)$ can be visualized as a binary tree. Figure 1(a) shows a CART model constructed on the sample one-dimensional data set in (b). The sample data set is generated using

$$y_i = x_i^2 + \varepsilon_i, \qquad i = 1, 2, \ldots, 100,$$

where x_i is uniformly distributed within (0,10), and ε_i follows a Guassian distribution of $N(0, 10)$. The leaf nodes correspond to disjoint hyper-rectangles in the feature vector space. The hyper-rectangles are degenerated into intervals for one-dimensional data sets. Each leaf is associated with a value, $\hat{f}(X)$, which is the prediction for all Xs within the corresponding hyper-rectangle. The internal nodes contain split points, and a path from the root to a leaf defines the hyper-rectangle of the leaf node. The tree, therefore, represents a piece-wise constant function on the feature vector space. Figure 1(b) shows the regression line of the sample CART model.

3.2 CART Model Properties

CART models are computationally efficient in both construction and prediction. The construction algorithm starts with a tree with a single root node corresponding to the entire input vector space and grows the tree by greedily selecting the split point that yields the maximum reduction in mean squared error. A more detailed discussion of the split point selection is presented in Appendix A. Each prediction involves a tree traversal and, therefore, is fast.

CART offers good interpretability and allows us to evaluate the importance of various workload characteristics in predicting workload performance. A CART model is a binary tree, making it easy to plot on paper as in Figure 1(a). More importantly, one can evaluate a feature's importance by its contribution in error reduction. Intuitively, a more important feature should contribute more to the error reduction; thus, leaving it out of the feature vector would significantly raise the prediction error. In a CART model, we use the sum of the error reduction related to all the appearances of a feature as its importance.

3.3 Comparison With Other Regression Tools

Other regression tools can achieve the same functionality as CART. We choose to use CART because of its accuracy, efficiency, robustness, and ease of use. Table 1 compares CART with four other popular tools to

Feature	CART	Linear regression	Neural networks	Support vector machines	k-nearest neighbors
Prediction error (median relative error)	low (36%)	high (505%)	fair (59%)	fair (66%)	low (28%)
Interpretability	Good	Good	Poor	Poor	Poor
Robustness to outliers	Good	Fair	Poor	Poor	Good
Ability to handle irrelevant input	Good	Good	Poor	Poor	Poor
Model construction time	fast (seconds)	fast (seconds)	slow (hours)	slow (hours)	N/A (no training)
Prediction time	fast (milliseconds)	fast (milliseconds)	fast (milliseconds)	fast (milliseconds)	slow (minutes)
Storage requirement	low(8 KB)	low(60 B)	low(5 KB)	low(2 MB)	high(11 MB)
Ease of use	Good	Good	Fair	Fair	Fair

Table 1: Comparison of regression tools in predicting per-request response time. (The same data set is used in Figure 5.) The comparison on row 2, 3, 4 and the last one is taken from [16]. We rank the features in the order of their importance. Interpretability is the model's ability to infer the importance of input variables. Robustness is the ability to function well under noisy data set. Irrelevant input refers to features that have little predictive powers.

build the request-level device model as described in Section 4.2. The models were constructed on the first day of *cello99a* and tests run on the second of the same trace. The informaion on the traces we used may be found in Section 5.

- The `linear regression` model [29] uses a linear function of X to approximate $f(X)$. Due to non-linear storage device behavior, linear models have poor accuracy.
- The `Neural Network` model [26] consists of a set of highly interconnected processing elements working in unison to approximate the target function. We use a single hidden layer of 20 nodes (best among 20 and 40) and a learning rate of 0.05. Half of the training set is used in building the model and the other half for validation. Such a model takes a long time to converge.
- The `Support Vector Machine` [6] maps the input data into a high dimensional space and performs a linear regression there. Our model uses the radial basis function

  $$K(x_i, x) = exp(\gamma||x - x_i||^2),$$

  as the kernel function, and γ is set to be 2 (best among 1, 3, 4, 6). We use an efficient implementation, SVM^{light} [18], in our experiment. Selecting the parameter values requires expertise and multiple rounds of trials.
- The `k-Nearest Neighbors` model [9] is memory-based because the model remembers all the training data points and prediction is done through averaging the output of the k nearest neighbors of the data point being predicted. We use the Euclidean distance function and a k value of 5 (best among 5, 10, 15, and 20). The model is accurate, but is inefficient in storage and computation.

The last three tools require that all the features and output be normalized to the unit length. For features of large value range, we take logarithms before normalization. Overall, CART is the best at predicting per-request response times, with the only downside being slightly lower accuracy compared to the much more space- and time-consuming `k-Nearest Neighbors` approach.

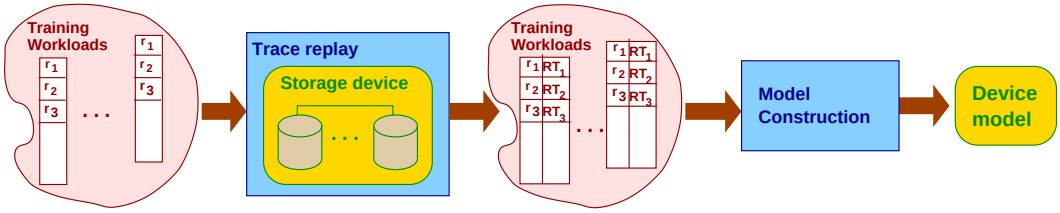


Figure 2: Model construction through training. RT_i is the response time of request r_i.

4 Predicting Performance with CART

This section presents two ways of constructing device models based on CART models.

4.1 Overview

Our goal is to build a model for a given storage device which predicts device performance as a function of I/O workload. The device model receives a workload as input and predicts its aggregate performance. We define a workload as a sequence of disk requests, with each request, r_i, uniquely described by four attributes: arrival time ($ArrivalTime_i$), logical block number (LBN_i), request size in number of disk blocks ($Size_i$), and read/write type (RW_i). The storage device could be a single disk, a disk array, or some other like-interfaced component. The aggregate performance can be either the average or the 90-th percentile response time.

Our approach uses CART to approximate the function. We assume that the model construction algorithm can feed any workload into the device to observe its behavior for a certain period of time, also known as "training." The algorithm then builds the device model based on the observed response times, as illustrated in Figure 2. Model construction does not require any information about the internals of the modeled device. Therefore, it is general enough to model any device.

Regression tools are a natural choice to model device behavior. Such tools are designed to model functions on multi-dimensional space given a set of samples with known output. The difficulty is to transform workloads into data points in a multi-dimensional feature space. We explore two ways to achieve the transformation as illustrated in Figure 3. A request-level model represents a request r_i as a vector R_i, also known as the "request description," and uses CART models to predict per-request response times. The aggregate performance is then calculated by aggregating the response times. A workload-level model, on the other hand, represents the entire workload as a single vector W, or the "workload description," and predicts the aggregate performance directly from W. In both approaches, the quality of the input vectors is critical to the model accuracy. The next two sections present the request and workload descriptions in detail.

4.2 Request-Level Device Models

This section describes the CART-based request-level device model. This model uses a CART model to predict the response times of individual requests based on request descriptions. The model, therefore, is able to generate the entire response time distribution and output any aggregate performance measures.

We adopt the following two constraints in designing the request description.

1. R_i does not include any actual response times. One could relax this constraint by allowing the inclusion of the response time information for all the requests that have already been served when the current request arrives. This relaxation, however, is feasible only for online response time predictions; it would not be appropriate for application planning tasks because the planner does not run workloads on devices.

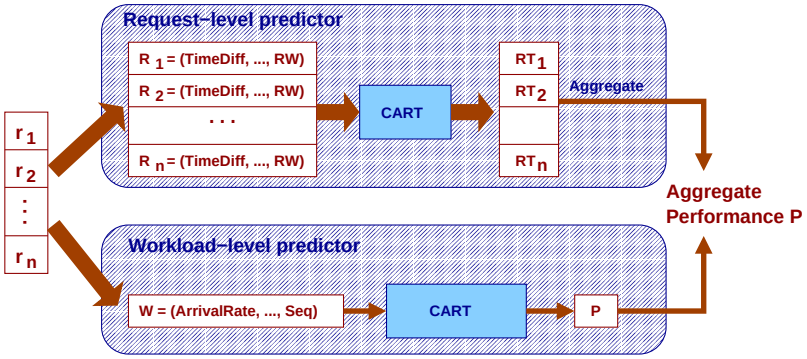


Figure 3: Two types of CART-based device models.

2. R_i can be calculated from r_j, $j \leq i$. This constraint simplifies the request description. In most cases, the response time of a current request depends only on previous requests and the request itself.

Our request description R_i for request r_i contains the following variables:

$$\begin{aligned} R_i = \quad \{ \quad & TimeDiff_i(1), \quad \ldots, \quad TimeDiff_i(k), \\ & LBN_i, \quad LBNDiff_i(1), \quad \ldots, \quad LBNDiff_i(l), \\ & Size_i, \quad RW_i, \\ & Seq(i) \quad \}, \end{aligned}$$

where $TimeDiff_i(k) = ArrivalTime_i - ArrivalTime_{i-2^{k-1}}$ and $LBNDiff_i(l) = LBN_i - LBN_{i-l}$. The first three groups of features capture three components of the response time, and $Seq(i)$ indicates whether the request is a sequential access. The first $(k+1)$ features measure the temporal burstiness of the workload when r_i arrives, and support prediction of the queuing time. We allow the $TimeDiff$ features to exponentially grow the distance from the current request to history request to accommodate large bursts. The next $(l+1)$ features measure the spatial locality, supporting prediction of the seek time of the request. $Size_i$ and RW_i support prediction of the data transfer time.

The two parameters, k and l, determine how far we look back for request bursts and locality. Small values do not adequately capture these characteristics, leading to inferior device models. Large values, on the other hand, leads to a higher dimensionality, meaning the need for a larger training set and a longer training time. The optimal values for these parameters are highly device specific, and Section 5.1 shows how we select the parameter values in our experiments.

4.3 Workload-Level Device Models

The workload-level model represents the entire workload as a single workload description and predicts aggregate device performance directly. The workload description W contains the following features.

$$\begin{aligned} W = \quad \{ \quad & \text{Average arrival rate}, \\ & \text{Read ratio}, \\ & \text{Average request size}, \\ & \text{Percentage of sequential requests}, \\ & \text{Temporal and spatial burstiness}, \\ & \text{Correlations between pairs of attributes} \quad \}. \end{aligned}$$

The workload description uses the entropy plot [32] to quantify temporal and spatial burstiness and correlations between attributes. Entropy value are plotted on one or two attributes against the entropy calculation granularity. The increment of the entropy values characterizes how the burstiness and correlations change from one granularity to the next. Because of the self-similarity of I/O workloads [13], the increment is usually constant, allowing us to use the entropy plot slope to characterize the burstiness and correlations. Appendix B describes the entropy plot in detail.

The workload-level device model offers fast predictions. The model compresses a workload into a workload description and feeds the description into a CART model to produce the desired performance measure. Feature extraction is also fast. To predict both the average and 90th percentile response time, the model must have two separate trees, one for each performance metric.

Workload modeling introduces a parameter called "window size." The window size is the unit of performance prediction and, thus, the workload length for workload description generation. For example, we can divide a long trace into one-minute fragments and use the workload-level model to predict the average response time over one-minute intervals. Fragmenting workloads has several advantages. First, performance problems are usually transient. A "problem" appears when a large burst of requests arrive and disappears quickly after all the requests in the burst are served. Using the workload in its entirety, on the other hand, fails to indentify such transient problems. Second, fragmenting the training trace produces more samples for training and reduces the required training time. Windows that are too small, however, contain too few requests for the entropy plot to be effective. We use one-minute windows in all of our experiments.

4.4 Comparison of Two Types of Models

There is a clear tradeoff between the request-level and workload-level device models. The former is fast in training and slow in prediction, and the latter is the opposite.

The model training time is dominated by trace replay, which, when taking place on actual devices, requires exactly the same amount of time as the trace length. Building a CART model needs only seconds of computation, but trace replay can require hundreds of hours to acquire enough data points for model construction. When operating at the request level, the device model gets one data point per request as opposed to one data point per one-minute workload fragment as in the workload-level device model. In order to get the same number of data points, the workload-level device model needs a training time 100 times longer than the request-level model when the arrival rate is 100 requests per minute.

The number of tree traversals determines the prediction time, since each predicted value requires a tree traversal. Therefore, the total number of tree traversals is the number of requests in the workload for the request-level device model and the number of workload fragments for the workload-level model. With an average arrival rate of 100 requests per minute, the request-level model is 100 times slower in prediction.

An item for future research is the exploration of the possibility of combining the two models to deliver ones that are efficient in both training and prediction.

5 Experimental Results

This section evaluates the CART-based device models presented in the previous section using a range of workload traces.

Devices. We model two devices: a single disk and a disk array. The single disk is a 9GB Atlas 10K disk with an average rotational latency of 3 milliseconds. The disk array is a RAID 5 disk array consisting of 8 Atlas 10K disks with a 32 KB stripe size. We replay all the traces on the two devices except the *SAP* trace, which is beyond the capacity of the Atlas 10K disk.

Trace name	Trace description				Average response time	
	Length	# of requests	Average Size	% reads	single disk	RAID 5
cello92	4 weeks	7.8 Million	12.9 KB	35.4%	83.78 ms	59.28 ms
cello99a	4 weeks	43.7 Million	7.1 KB	20.9%	115.71 ms	32.58 ms
cello99b	4 weeks	13.9 Million	118.0 KB	41.6%	113.61 ms	22.03 ms
cello99c	4 weeks	24.0 Million	8.5 KB	26.4%	5.04 ms	5.04 ms
SAP	15 minutes	1.1 Million	15.1 KB	99.9%	-	7.40 ms

Table 2: Trace summary. We model an Atlas 10K 9GB and a RAID 5 disk array consisting of 8 Atlas 10K disks. The response time is collected by replaying the traces on DiskSim3.0 [5].

Traces. We use three sets of real-world traces in this study. Table 2 lists the summary statistics of the edited traces. The first two, *cello92* and *cello99* capture typical computer system research I/O workloads, collected at HP Labs in 1992 and 1999 respectively [27, 14]. We preprocess *cello92* to concatenate the LBNs of the three most active devices from the trace to fill the modeled device. For *cello99*, we pick the three most active devices, among the 23 devices, and label them *cello99a*, *cello99b*, and *cello99c*. The *cello99* traces fit in a 9GB disk perfectly, so no trace editing is necessary. As these traces are long (two months for *cello92* and one year for *cello99*), we report data for a four-week snapshot (5/1/92 to 5/28/92 and 2/1/99 to 2/28/99).

The *SAP* trace was collected from an Oracle database server running SAP ISUCCS 2.5B in a power utility company. The server has more than 3,000 users and disk accesses reflect the retrieval of customer invoices for updating and reviewing. Sequential reads dominate the *SAP* trace.

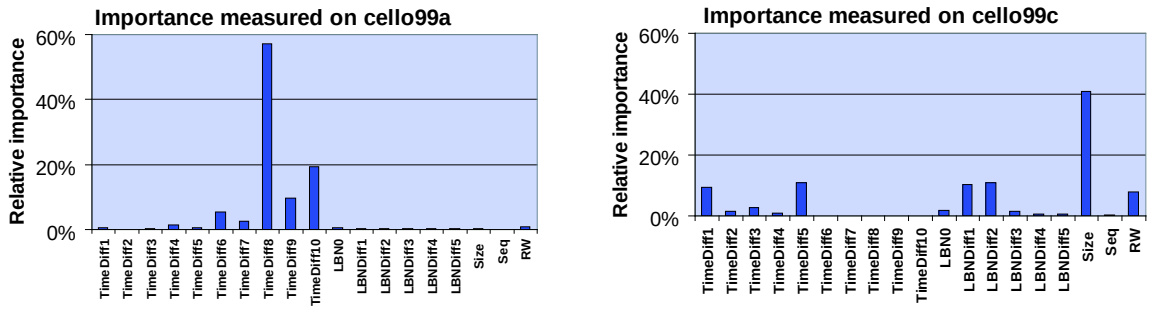
Evaluation methodology. The evaluation uses the device models to predict the average and 90th percentile response time for one-minute workload fragments. We report the prediction errors using two metrics: absolute error defined as the difference between the predicted and the actual value, $|\hat{Y} - Y|$, and relative error defined as $\frac{|\hat{Y}-Y|}{Y}$.

We use the first two weeks of *cello99a* in training because of the trace's relatively rich access patterns. The training trace is 19,583 minutes long. Because of the large number of requests, we use a uniform sampling rate of 0.01 to reduce the number of requests to 218,942 in training the request-level model.

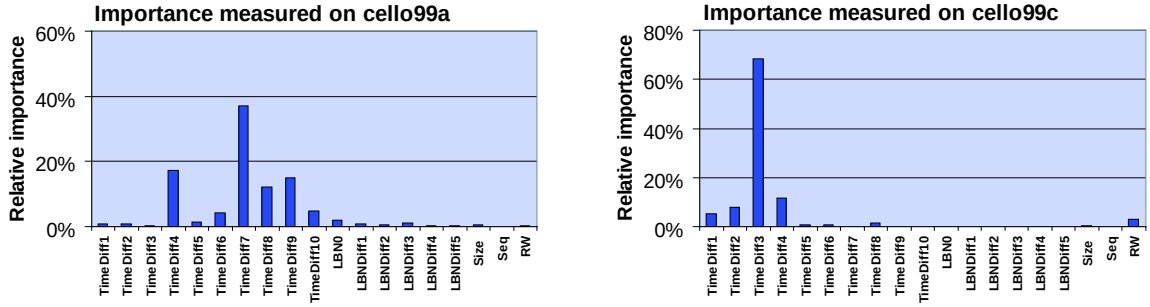
Predictors in comparison. We evaluate our two CART-based device models, denoted as `CART-request` and `CART-workload` in the remaining text, against three predictors.

- `constant` makes predictions using the average or quantile response time of the training trace.
- `periodic` divides a week into $24 \times 7 \times 60$ one-minute intervals and remembers the aggregate performance of the training workload for each interval. Prediction uses the corresponding value of the interval with the same offset within the week.
- `linear` does linear regression on the workload descriptions.

Note that the `constant` and `periodic` predictors model workloads rather than devices, because they do not take workload characteristics as input. Both predictors rely on the similarity between the training and testing workloads to produce accurate predictions. The difference between `linear` and `CART-workload`, on the other hand, shows the importance of using non-linear models, such as the CART models, in device modeling.



(a) relative importance measured on the 9GB Atlas 10KB disk using *cello99a* and *cello99c*



(b) relative importance measured on the RAID5 disk array using *cello99a* and *cello99c*

Figure 4: Relative importance of the request description features.

5.1 Calibrating Request-Level Models

This section describes how we select parameter values for k and l for the request-level device models.

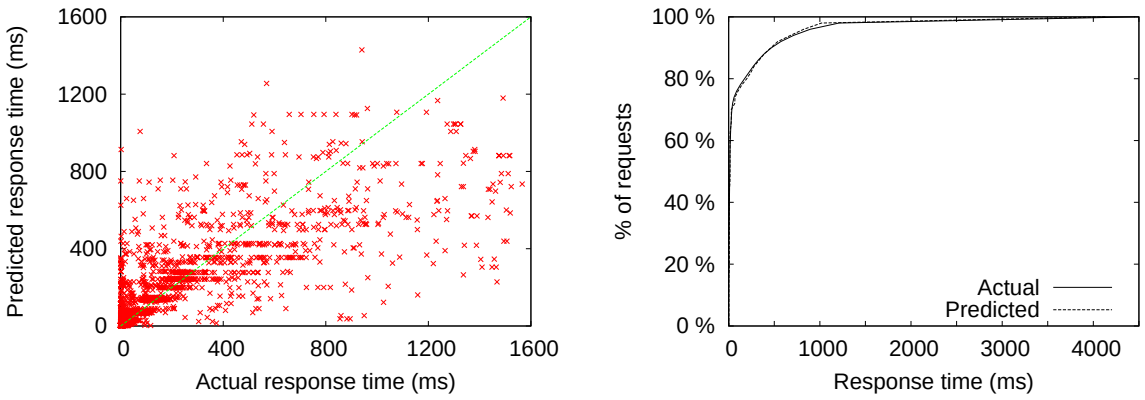
Figure 4 shows the relative importance of the request description features in determining per-request response time by setting k to 10 and l to 5. The feature's relative importance is measured by its contribution in error reduction. The graphs show the importance of request description features measured on both devices, trained on two traces (*cello99a* and *cello99c*). We use only the first day of the traces and reduce the data set size by 90% with uniform sampling.

First, we observe that the relative importance is workload dependent. As we expected, for busy traffic such as that which occurred in the *cello99a* trace, the queuing time dominates the response time, and thereby, the $TimeDiff$ features are more important. On the other hand, *cello99c* has small response times, and features that characterize the data transfer time, such as *Size* and *RW*, have good predictive power in modeling the single disk.

Second, we observe that the most imporant feature shifts from $TimeDiff8$ to $TimeDiff7$ where comparing the single disk to the disk array for *cello99a* because the queuing time becomes less significant for the disk array. The distinction between the two traces, however, persists.

We set k to 10 for $TimeDiff$ and l to 3 for $LBNDiff$ in the subsequent experiments so that we can model device behavior under both types of workloads.

We show the model accuracy in predicting per-request response times in Figure 5. The model is built for the Atlas 10K disk. The training trace is the first day of *cello99a*, and the testing trace is the second day of the same trace. Figure 5(a) is a scatter plot, showing the predicted response times against the actual ones for the first 5,000 requests. Most of the points stay close to the diagonal line, suggesting accurate prediction of the request-level device model. Figure 5(b) further compares the response time distributions. The long tail of the distribution is well captured by the request-level model, indicating that the request description is



(a) Scatter plot (b) Response time distribution

Figure 5: Prediction accuracy of the request-level model. The actual and predicted average response times are 137.96 ms and 133.01 ms respectively. The corresponding demerit, defined in [28] as the root mean square of horizontal distance between the actual and predicted curves in (b), is 46.06 milliseconds (33.4% of the actual average response time).

effective in capturing request-level characteristics needed to predict response times.

In summary, the request description effectively captures important per-request characteristics, leading to accurate request-level device models.

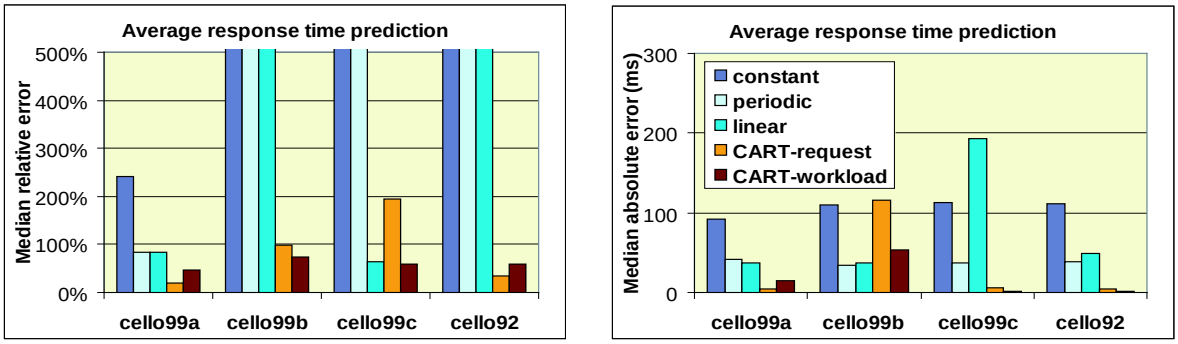
5.2 Modeling A Single Disk

Figure 6 compares the accuracy of all the predictors in modeling an Atlas 10K 9GB disk on real-world traces. As mentioned earlier, all the predictors are trained using the first two weeks of *cello99a*. Overall, the two CART-based device models provide good prediction accuracy in predicting both the average and 90th percentile response times, compared to other predictors. Several more detailed observations can be made.

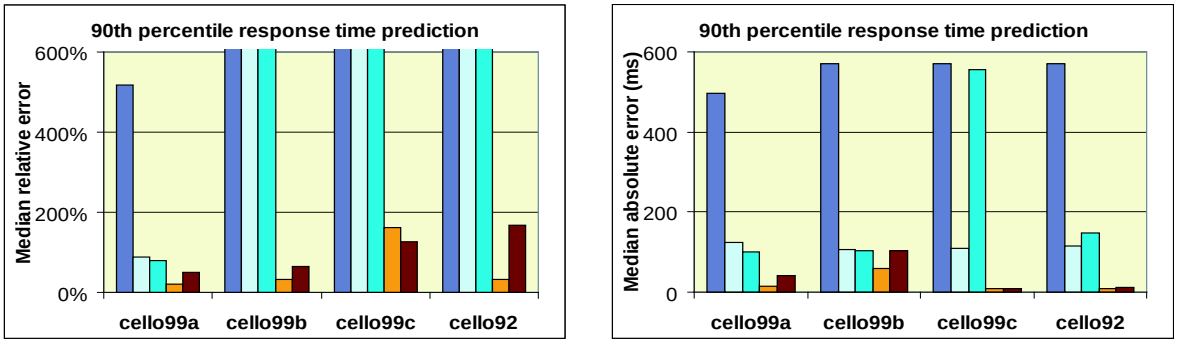
First, all of the models perform the best when the training and testing traces are from the same workload, e.g. *cello99a*, because the models have seen how the device behaves under such workloads. The `periodic` predictor also cuts the median prediction error of the `constant` predictor by more than a half because of the strong periodicity of the workload. `CART-request` and `CART-workload` further reduce the error to 4.84 milliseconds (19%) and 14.83 milliseconds (47%) respectively for the average response time prediction, and 20.46 milliseconds (15%) and 49.50 milliseconds (45%) respectively for the 90th percentile. The performance difference between `linear` and `CART-workload` roughly measures the benefit of using a non-linear model, such as CART, because both accept the same input. We observe a significant improvement from the former to the latter, suggesting non-linear device behavior.

Second, both CART-based device models interpolate better across workloads than the other models. `constant` and `periodic` rely blindly on similarities between the training and testing workloads to make good predictions. Consequently, it is not surprising to see huge prediction errors when the training and testing workloads differ. The CART-based predictors, on the other hand, are able to distinguish between workloads of different characteristics and are more robust to the difference between the training and testing workloads.

Third, model accuracy is highly dependent on the training workload quality for the CART-based models. The prediction error increases for workloads other than *cello99a*, because of the access pattern differences among these traces. The CART-based models learn device behavior through training; therefore, they cannot predict performance for workloads that have totally different characteristics from the training workloads.



(a) Prediction error for average response time



(b) Prediction error for 90th percentile response time

Figure 6: Comparison of predictors for a single 9GB Atlas 10K disk.

For example, `CART-request` constantly over-predicts for *cello99c* because the model was never trained wth the small sequential accesses that are particular to *cello99c*. Section 5.4 gives an informal error analysis and identifies inadequate training being the most significant error source.

Fourth, high quantile response times are more difficult to predict. We observe larger prediction errors from all the predictors for 90th percentile response time predictions than for average response time predictions. The accuracy advantage of the two CART-based models is higher for 90th percentile predictions.

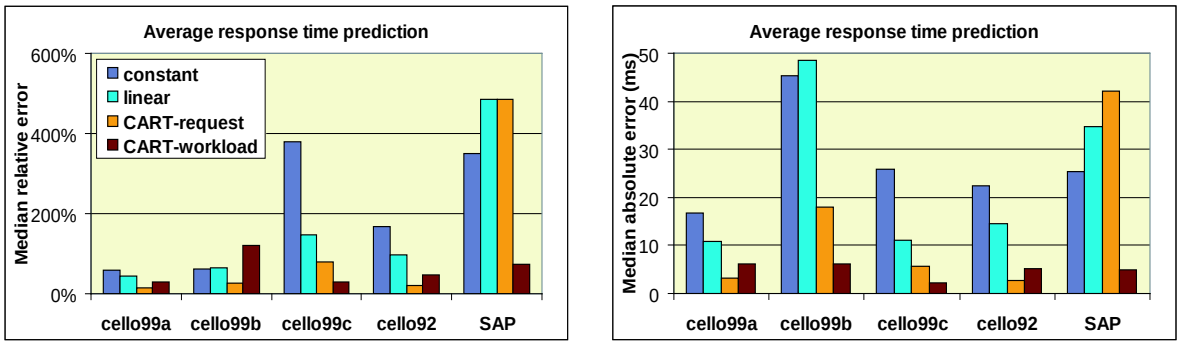
In summary, the two CART-based models give accurate predictions when the training and testing workloads share the same characteristics and interpolate well otherwise. The good accuracy suggests the effectiveness of the request and workload descriptions in capturing important workload characteristics.

5.3 Modeling A Disk Array

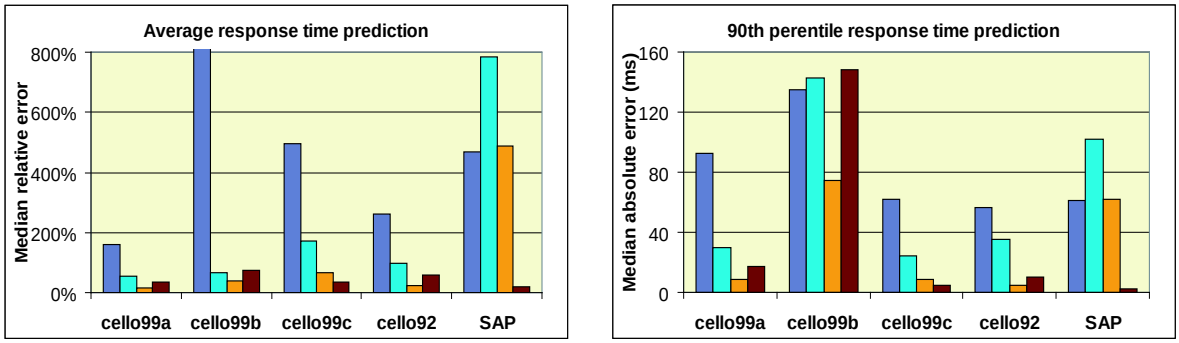
Figure 7 compares the accuracy of the four predictors in modeling the disk array. The `periodic` predictor is not presented because the *SAP* trace does not provide enough information on arrival time for us to know the offset within a week. The overall results are similar to those for the single disk. The two CART-based models are the most accurate predictors. The absolute errors become smaller due to the decreased response time from the single disk to the disk array. The relative accuracy among the predictors, however, stays the same. Overall, the CART-based device modeling approach works well for the disk array.

5.4 Error Analysis

This section presents an informal error analysis to identify the most significant error source for the CART-based device models.



(a) prediction error for average response time

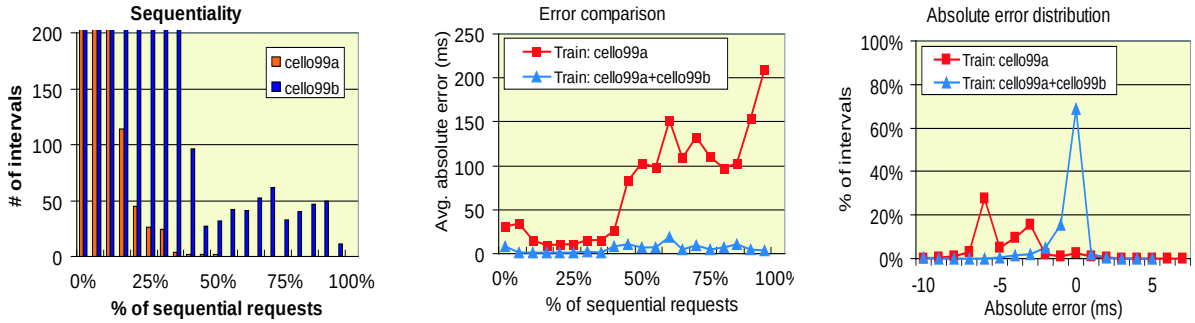


(b) prediction error for 90th percentile response time

Figure 7: Comparison of predictors for a RAID 5 disk array of 8 Atlas 10K disks.

A model's error consists of two parts. The first part comes from intrinsic randomness of the input data, such as measurement error, and this error cannot be captured by any model. The rest of the error comes from the modeling approach itself. The CART-based models incur error at three places. First, the transformation from workloads to vectors introduces information loss. Second, the CART-based models use piece-wise constant functions, which could be different from the true functions. Third, a low-quality training trace yields inaccurate models because CART relies on the information from the training data to make predictions. An inadequate training set has only a limited range of workloads and leads to large prediction errors for workloads outside of this range. We find that the last error source, inadequate training data, causes the most trouble in our experiments.

We conduct a small experiment to verify our hypothesis. Figure 8(a) compares the difference in sequentiality between *cello99a* and *cello99c*. The spectrum of sequentiality (from 0% to 100% of requests in the workload being sequential) is divided into 20 buckets, and the graphs shows the number of one-minute workload fragments in each bucket for both traces. We observe a significant number of high sequentiality fragments in *cello99b*, but no fragment goes beyond 50% sequentiality in *cello99a*. This difference leads to large prediction errors for high sequentiality fragments when we build the workload-level model on *cello99a* and use it to predict the performance of *cello99b*, as shown in (b). The errors are reduced significantly when we include the first half of *cello99b* in training. The dramatic error reduction suggests that prediction errors from the other sources are negligible when compared with the ones introduced by inadequate training. Figure 8(c) further shows the absolute error histogram with 1 millisecond buckets. The spike shift to 0 milliseconds when we train the model on the combined training trace, indicating that it is reasonable to assume a zero-mean noise term. We conclude from this evidence that contributing efforts in black-box device modeling should be directed toward generating a good training set that covers a broad range of workload types.



(a) Histograms on sequentiality (b) Average absolute error (c) Absolute error distribution

Figure 8: Effects of different training workloads.

6 Conclusions

Storage device performance modeling is an important element in self-managed storage systems and other application planning tasks. Our target model takes a workload as input and predicts its aggregate performance on the modeled device efficiently and accurately. This paper presents our initial results in exploring machine learning tools to build device models. A black box predictive tool, CART, makes device models independent of the storage devices being modeled, and thus, general enough to handle any type of devices. The model construction, also known as training, consists of two phases: replaying traces on the devices and building a CART model based on the observed response times. Modeling a new device involves only training on the target device.

CART-based models take input in the form of vectors, so workloads must be transformed into vectors in order to use CART as the basis for device models. This paper presents two ways to accomplish such a transformation, yielding two types of device models. The request-level device models represent each request as a vector and predict its response time. As a result, the models are able to predict the entire response time distribution. The experiments show that the predicted response time has a demerit figure of 33% for a modern UNIX file server trace, leading to a median relative error as low as 16% for aggregate performance predictions. The workload-level device models, on the other hand, transform a workload fragment into a vector and predict its aggregate performance directly. The vector takes advantage of the efficient entropy plot metric to capture the temporal and spatial burstiness as well as the correlations within I/O workloads. The median relative error can be as low as 29% for the workload-level device models.

The error analysis suggests that the quality of the training workloads plays a critical role in the model accuracy. The models are unable to predict workloads that are different from the training workloads. To accurately predict arbitrary workloads, it is important for the training workloads to be as diverse as possible to cover a wide range of workloads. Our future work will explore the effectiveness of existing synthetic workload generators in producing high-quality training workloads.

Continuing research can improve the model prediction accuracy. First, our experiments show the relevance of training traces. Generating rules to assist in training such models broadly enough will be important. Second, the workload characterization problem persists, affecting the workload-level models. We believe, however, that the context offered by the models can help produce insight into this long-standing problem. Third, the two types of device models show desirable properties in training and predicting, respectively. It should be valuable to have a model that combines the best of both approaches.

References

[1] N. Allen. Don't waste your storage dollars: What you need to know. Research note, Gartner Group, 2001.

[2] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: Quickly finding near-optimal storage system designs. Technical Report HPL-SSP-2001-05, HP Labs, 2001.

[3] Eric Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-4, HP Labs, 2001.

[4] L. Brieman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[5] John Bucy, Greg Ganger, and contributors. The DiskSim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, 2003.

[6] C.J.C. Burges. A tutorial on Support Vector Machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167.

[7] Shenze Chen and Don Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[8] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In *Proc. of the 1996 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, May 1996.

[9] B. Dasarathy. *Neareast neighbor pattern classification techniques*. IEEE Computer Society Press, 1991.

[10] Gregory R. Ganger. Generating representative synthetic workloads: An unsolved problem. In *Proceedings of the Computer Measurement Group (CMG) Conference*, pages 1263–1269, 1995.

[11] Gregory R. Ganger, John D. Strunk, and Andrew J. Klosterman. Self-* storage: Brick-based storage with automated administration. Technical Report CMU-CS-03-178, Carnegie Mellon University, 2003.

[12] Gartner Group. Total cost of storage ownership — A user-oriented approach. Research note, Gartner Group, 2000.

[13] María E. Gómez and Vicente Santonja. Analysis of self-similarity in I/O workload using structural modeling. In *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 234–243, 1999.

[14] María E. Gómez and Vicente Santonja. A new approach in the modeling and generation of synthetic disk workload. In *Proceedings of 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 199–206, 2000.

[15] J. Gray. A conversation with Jim Gray. *ACM Queue*, 1(4), 2003.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2001.

[17] International Business Machines Corp. Autonomic computing: IBM's perspective on the state of information technology. http://www.research.ibm.com/autonomic/manifesto/.

[18] T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT-Press, 1999.

[19] K. Keeton, G. Alvarez, E. Riedel, and M. Uysal. Characterizing I/O-intensive workload sequentiality on modern disk arrays. In *The Fourth Workshop On Computer Architecture Evaluation Using Commercial Workloads*, 2001.

[20] Zachary Kurmas, Kimberly Keeton, and Ralph Becker-Szendy. I/O workload characterization. In *4th workshop on computer architecture evaluation using commercial workloads*, 2001.

[21] E. Lamb. Hardware spending sputters. *Red Herring*, pages 32–33, 2001.

[22] Edward K. Lee and Randy H. Katz. An analytic performance model of disk arrays. In *Proceedings of the 1993 ACM SIGMETRICS*, pages 98–109, 1993.

[23] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In *ACM SIGCOMM*, pages 13–17, 1993.

[24] Arif Merchant and Guillermo A. Alvarez. Disk array models in Minerva. Technical Report HPL-2001-118, HP Laboratories, 2001.

[25] Ruldolf H. Riedi, Matthew S. Crouse, Vinay J. Ribeiro, and Richard G. Baraniuk. A multifractal wavelet model with application to network traffic. *IEEE Transactions on Information Theory*, 45(3):992–1018, April 1999.

[26] B.D. Ripley. *Patern recognitions and Neural Networks*. Cambridge University Press, 1996.

[27] Chris Ruemmler and John Wilkes. Unix disk access patterns. In *Winter USENIX Conference*, pages 405–420, 1993.

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

[29] G. Seber. *Multivariate observations*. Wiley, 1984.

[30] Elizabeth Shriver, Arif Merchant, and John Wilkes. An analytical behavior model for disk drives with readahead caches and request reordering. In *Proceedings of International Conference on Measurement and Modeling of Computer Systems*, pages 182–191, 1998.

[31] Mustafa Uysal, Guillermo A. Alvarez, and Arif Merchant. A modular, analytical throughput model for modern disk arrays. In *Proceedings of 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 183–192, 2001.

[32] Mengzhi Wang, Anastassia Ailamaki, and Christos Faloutsos. Capturing the spatio-temporal behavior of real traffic data. *Performance Evaluation*, 49(1/4):147–163, 2002.

[33] J. Wilkes. The Pantheon storage-system simulator. Technical Report HPL–SSP–95–14, Hewlett-Packard Laboratories, 1995.

[34] John Wilkes. Traveling to Rome, QoS specifications for automated storage system management. In *Proc. Intl. Workshop on Quality of Service (IWQoS'2001)*, pages 75–91. Springer-Verlag Lecture Notes in Computer Science 2092, 2001.

[35] John Wilkes. Data services – from data to containers. Keynote address at File and Storage Technologies Conference (FAST'03), March – April 2003.

Appendix A: Constructing CART Models

A CART model is a piecewise-constant function on a multi-dimensional space. This appendix gives a brief description of the model construction algorithm. Please refer to [4] for a complete discussion of CART models.

The CART model has a binary tree structure built by recursive binary splits. Suppose we have N observations, $\{X_i | i = 1, 2, \ldots, N\}$, with corresponding outputs $\{Y_i | i = 1, 2, \ldots, N\}$. Each observation consists of p input features, $X_i = (x_{i1}, \ldots x_{ip})$. The construction algorithm starts with a tree with only a root node and grows the tree downward by splitting one node a time. The chosen split offers the most benefit in reducing the mean squared error. The average Y_i for all the X_is in a leaf node is used as the predictive value for the leaf node. The algorithm continues until certain criteria are met.

We describe how the split is chosen in detail next. The algorithm evaluates all the possible distinct splits on all the leaf nodes of the tree (or the root node in the first step). A node corresponds to a hyer-rectangle region of the input vector space, and a split decides along which feature and at what value the region should be divided into two. For example, at $node(t)$, a split on feature j at value v defines two nodes, $node(t_1)$ and $node(t_2)$.

$$\begin{aligned} X_i \in node(t_1) &\leftarrow \{X_i | x_{ij} \leq v \bigwedge X_i \in node(t)\}, \\ X_i \in node(t_2) &\leftarrow \{X_i | x_{ij} > v \bigwedge X_i \in node(t)\}. \end{aligned}$$

If we denote the number of observations in $node(t)$ as $N(t)$ and the predictive value as $Y(t)$, the mean squared error at $node(t)$ before the split is

$$MSE(t) = \sum_{i:X_i \in node(t)} \frac{1}{N(t)} (Y_i - \bar{Y}(t))^2,$$

where $\bar{Y}(t)$ is the predictive value assigned to $node(t)$. After the split, the mean squared error becomes

$$\begin{aligned} MSE(t)' &= \sum_{i:X_i \in node(t_1)} \frac{1}{N(t_1)} (Y_i - \bar{Y}(t_1))^2 \\ &\quad + \sum_{i:X_i \in node(t_2)} \frac{1}{N(t_2)} (Y_i - \bar{Y}(t_2))^2. \end{aligned}$$

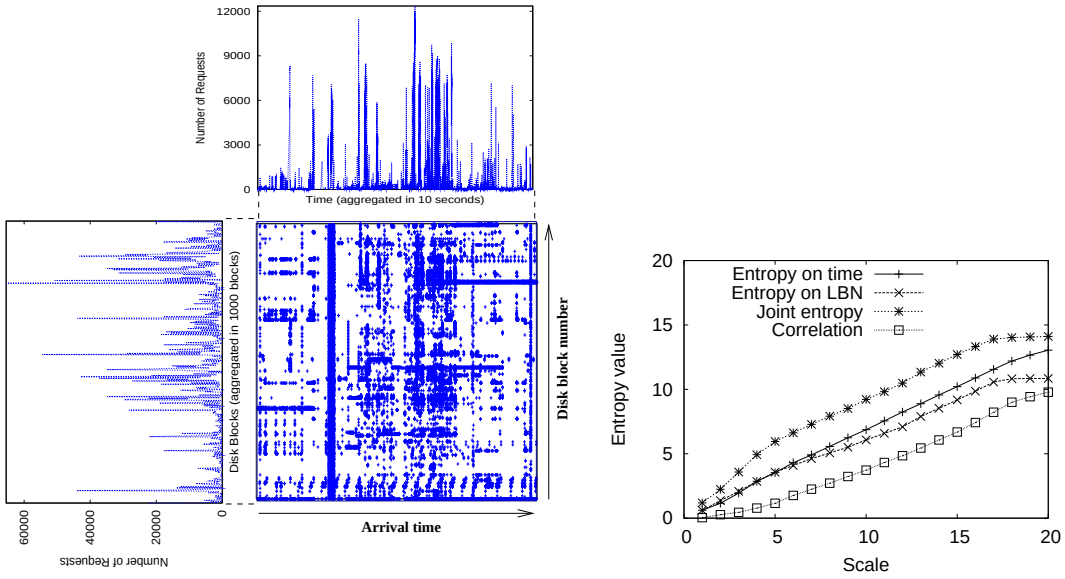
Hence, the reduction in mean squared error for the is

$$\Delta MSE = MSE(t) - MSE(t)'. \tag{A-1}$$

The algorithm calculates all the possible distinct splits and selects one that yields the largest value given by Equation A-1. The splitting continues until either the error reduction or the number of data points in the nodes becomes too small. At the end, we obtain the CART model.

Appendix B: Entropy Plot

The entropy plot quantifies I/O workload characteristics on individual attributes and their correlations by plotting entropy values against the granularity of the entropy calculation.



(a) Sample disk trace (b) Entropy plot

Figure 9: A sample disk trace and its entropy plot.

Entropy plot on one-dimensional datasets. The one-dimensional entropy plot captures characteristics of individual attributes, such as the temporal and spatial burstiness. These two types of burstiness measures the burstiness in the arrival process and the skew in access frequencies of disk blocks. We use the entropy plot for arrival time as an example to show how the entropy plot works.

Given a workload, we can derive its "margin" on the arrival time by counting the number of requests that arrive into the system at each time tick. The top graph of Figure 9(a) shows the sample trace's margin on arrival time.

Assume that the trace is 2^n time ticks long, and the margin is $C(i), i = 1,2,\ldots,2^n$. We calculate the entropy value at scale k by applying the entropy function on the aggregated margin at scale k. The aggregated margin is $C^{(k)}(j), j = 1,2,\ldots,2^k$, where

$$C^{(k)}(j) = \sum_{i=1}^{2^{n-k}} C(2^{(n-k)} * (j-1) + i).$$

Intuitively, the entire length of the margin is divided into 2^k equi-lengthed intervals at scale k. Thus, applying the entropy function on $C^{(k)}$ gives

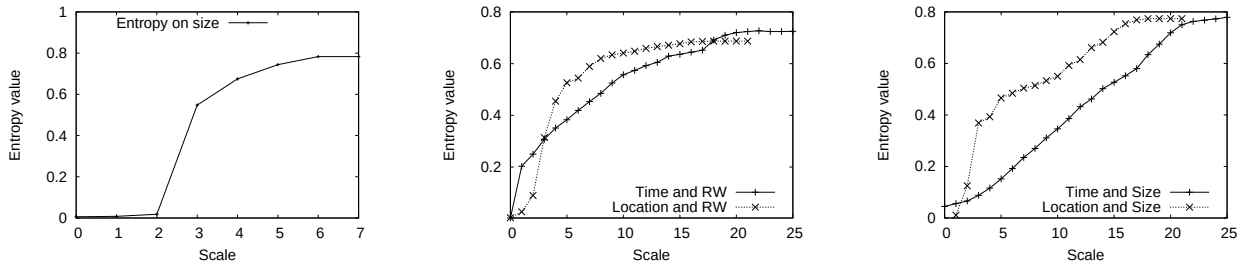
$$H^{(k)} = -\sum_{j=1}^{2^{n-k}} P^{(k)}(j) \log_2 P^{(k)}(j),$$

where

$$P^{(k)}(j) = \frac{C^{(k)}(j)}{\sum_{i=1}^{2^n} C(i)}.$$

Plotting $H^{(k)}$ against scale k gives the entropy plot.

We use the entropy value because it measures the irregularity of a discrete distribution function. The entropy value reaches its maximum of l for $P(i), i = 1,2,\ldots 2^l$, when all the $P(i)$s are equal. A highly skewed distribution has a smaller entropy value. Therefore, bursty traffic has a smaller entropy value at scale k than smooth traffic.



(a) Entropy plot on time (b) Entropy plot with operation type (c) Entropy plot with size

Figure 10: Entropy plots to quantify other workload characteristics.

A similar calculation on the trace's "margin" on LBN gives the entropy plot on LBN. Figure 9(b) shows the entropy plot on both arrival time and LBN of the sample trace.

We make two observations. First, the entropy plot shows strong linearity, suggesting the skew in arrival time and LBN stays constant at all granularities. The constant increment of the entropy value from one scale to the next suggests that the degree of skew stays the same at all the scales. That is, the sample trace has the same burstiness at all scales, which confirms the self-similarity of I/O workloads observed in previous studies [13]. Second, the linear entropy plot allows us to use the entropy plot slopes to characterize the burstiness. Smooth traffic has an entropy plot of slope close to 1. Real-world traces, however, have strong burstiness. In summary, the entropy plot defined on the trace margins allows us to use two scalars to characterize both the temporal and spatial burstiness of I/O workloads.

Entropy plot on two-dimensional datasets. We extend the entropy plot to handle two-dimensional datasets to measure the correlations between two attributes. As before, the entropy plot calculates the entropy value at different scales, only this time on two-dimensional data sets. Given a two-dimensional projection of a trace, $C(i,j), i,j = 1,2,\ldots,2^n$, we divide the projection into $2^k \times 2^k$ grids,which aggregates both dimensions with scale k. This gives a series $C^{(k)}$ of $2^k \times 2^k$ elements. Applying the entropy function to $C^{(k)}$ gives the joint entropy value at scale k on the two dimensions.

The joint entropy allows us to calculate the correlation between the two attributes. The correlation is the difference between the sum of the entropy value on the two attributes and the joint entropy plot. Figure 9(b) shows both the joint entropy and the correlation on arrival time and LBN for the sample disk trace. We observe that a strong correlation exists between arrival time and LBN, and also that the correlation stays constant at all scales. Thus, we are able to use a scalar value, the correlation slope, to quantify the correlation between arrival time and LBN.

Entropy plot involving request size and operation type. It is possible to extend the entropy plot to handle operation type and request size. The only difference is the limited value ranges of the two attributes, which limit the number of data points in the entropy plot. As a result, the workload description does not include entropy plot slopes on these two attributes.

Quantifying the correlations involving either of the two attributes faces the same problem. Our solution is to always use the finest granularity on the request size or operation type, but to change the scale on the other attribute. For example, to calculate the joint entropy plot on arrival time and request type, the aggregation happens only on the arrival time. Figure 10 shows the entropy plots that involve operation type and request size. These entropy plots are not as linear as previous ones. Therefore, it is not straightforward to compress each line into a scalar. Currently, our workload description uses the average increment between two adjacent scales.