# Inducing Models of Black-Box Storage Arrays

Terence Kelly, Ira Cohen, Moises Goldszmidt, Kimberly Keeton
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2004-108
June 21, 2004*

statistical model induction, storage arrays, I/O response time prediction, performance model induction

This paper applies statistical model-induction techniques to the problem of forecasting response times in storage systems. Our work differs from prior research in several ways: we regard storage systems as *black boxes*; we automatically *induce* models rather than constructing them from detailed expert knowledge; we use lightweight *passive* observations, rather than extensive controlled experiments, to collect input data; we forecast *individual* response times rather than aggregates or averages; and we focus on large and complex *enterprise* storage arrays that comprise many RAID groups. We evaluate our methods using a lengthy storage trace collected in a real-world environment, and measure the predictive value of information available when requests are issued. This paper makes several contributions. First, we quantify the potential of a class of statistical methods for the challenging problem of automatic performance model induction. Second, we quantify improvements in accuracy that result when the range of information available to our models increases. Finally, we describe a general, low-cost modeling methodology that can be applied to a wide range of storage arrays.

* Internal Accession Date Only
Approved for External Publication

# Inducing Models of Black-Box Storage Arrays

Terence Kelly Ira Cohen Moises Goldszmidt Kimberly Keeton
Hewlett-Packard Laboratories, Palo Alto, CA, USA

June 11, 2004
$Revision: 1.38 $

## Abstract

This paper applies statistical model-induction techniques to the problem of forecasting response times in storage systems. Our work differs from prior research in several ways: we regard storage systems as *black boxes*; we automatically *induce* models rather than constructing them from detailed expert knowledge; we use lightweight *passive* observations, rather than extensive controlled experiments, to collect input data; we forecast *individual* response times rather than aggregates or averages; and we focus on large and complex *enterprise* storage arrays that comprise many RAID groups. We evaluate our methods using a lengthy storage trace collected in a real-world environment, and measure the predictive value of information available when requests are issued.

This paper makes several contributions. First, we quantify the potential of a class of statistical methods for the challenging problem of automatic performance model induction. Second, we quantify improvements in accuracy that result when the range of information available to our models increases. Finally, we describe a general, low-cost modeling methodology that can be applied to a wide range of storage arrays.

## 1 Introduction

Enterprise storage systems are increasingly hard to manage. Today's high-end storage arrays are complex and highly configurable, and therefore inherently difficult to reason about. Furthermore the trend toward storage consolidation in large data centers means that a single "black box" storage array can serve a variety of very different workloads. The mapping from fundamental device capabilities, configuration, and workload to performance often defies manual analysis by human experts, and researchers have therefore begun to automate tasks such as capacity planning and configuration [1, 3]. This approach centralizes performance modeling, but the construction of performance models remains challenging. State-of-the-art approaches rely heavily on expert analysis, but it is difficult for human analysts to keep pace with increasingly elaborate and often proprietary enterprise storage architectures. Enterprise storage trends call for modeling strategies that are more automated, less reliant on human expertise, and applicable to opaque devices.

This paper explores statistical approaches to automated performance model induction for enterprise storage systems. We classify individual requests to a large storage device as "fast" or "slow," using models induced offline to generate forecasts quickly enough for online application. Model induction relies on purely passive external observation of black-box storage arrays. We quantify improvements in predictive accuracy as we increase the range of data available to our models.

Our method begins with a raw trace of requests submitted to a storage array. We apply transformations to obtain a vector of *features* representing request characteristics and history-dependent system state at the instant each request is issued. In an offline operation, model-induction algorithms determine a mapping from feature vectors to response time forecasts. To generate forecasts online, we transform observable characteristics of individual requests into feature vectors and apply the induced models. We consider two extremes of a spectrum of models: one yields forecasts in the form of probability distributions over *values* of response time, the other yields distributions over *categories* of response time. In order to compare the two models on a common accuracy scale, we apply them to a binary classification

problem, i.e., we use them to predict whether individual read requests will be "fast" or "slow."

An understanding of disk head position and of seek and rotational latency can be crucial for determining response time, but the state of moving parts is unknown outside a black-box storage array. This places fundamental and severe limits on the potential accuracy of any modeling methodology, including ours. In the absence of such detailed knowledge, the question is to what extent other contextual information can increase prediction accuracy and increase our confidence in the predictions. The uncertainty inherent in our predictions is captured by probability distributions, and our goal is to use available information to *narrow* these distributions.

Our overall results can be summarized as follows: When the number of unfulfilled requests pending at the time a given request is issued is above 20 (around 5.4% of our data), response time is almost always slow and our algorithms achieve 95% classification accuracy. When the number of pending requests is less than 20, there is significant variability in the accuracy of prediction for different RAID groups within the storage array. Our algorithms yield an accuracy ranging from 70% to 85%. In all cases accuracy is much higher (by up to 24 percentage points) than one would obtain from naïve predictions based on prior probabilities alone, demonstrating that the information in our feature vector substantially improves predictive accuracy.

# 2 Applications

Our research on performance model induction is motivated by applications to scheduling, performance modeling, and anomaly detection. Before we can implement and evaluate these applications we must first understand the limits of our modeling approach. This paper considers modeling independently of applications. Our main concern is with the models themselves, the procedures used to induce them, and their predictive accuracy as a function of available information. To place our work in context, we briefly review in this section the complementary but orthogonal issue of applications for our models and their predictions.

Consider the problem of serving compound Web pages. Each screenful of material is assembled from a variety of components, e.g., static images and dynamically-generated HTML. Some of the corresponding HTTP requests primarily require I/O at the server end, while others mainly require CPU. The requests may arrive at the server simultaneously via HTTP's pipelining feature [15], and *all of them may be served in parallel*. In such situations, where transactions decompose naturally into CPU and I/O components that may be served concurrently, we can use I/O response time estimates for improved CPU scheduling and thereby obtain lower mean transaction times. We have explored the potential performance improvements of I/O-aware CPU scheduling using both analysis and stochastic simulation; not surprisingly, we find that its benefits depend sensitively on workload characteristics. More realistic results require very detailed workload traces that we have not yet been able to obtain.

Another application of performance models is performance monitoring and anomaly detection. Induced models may help us to discover deviations from expected system behavior. If the predictive accuracy of an induced model changes quickly and dramatically, standard statistical tests can determine whether the change is due to random fluctuations in workload or to a more fundamental change in behavior, e.g., an internal device failure. In a wide range of settings, including but not limited to enterprise storage arrays, performance models induced automatically from passive observations may provide an inexpensive way of alerting operators that "something about the system is very different today."

# 3 Related Work

This section reviews the large literature on storage system performance modeling, summarizes methodologies and key results, and relates our investigation to prior work. Performance modeling is often motivated by other research problems, e.g., disk scheduling, and many key results are strongly linked to the motivating problem or application. Our survey of the literature is therefore organized both by motive and by method.

## 3.1 Disk Scheduling

Researchers have long recognized that estimates of rotational position and rotational latency can be exploited for improved disk scheduling. Seltzer *et al.* described algorithms that employ such estimates [30]. Jacobson & Wilkes evaluated a taxon-

omy of algorithms that exploit rotational latency estimates, reporting that they match or outperform algorithms that consider only seek time [20]. More recently Lumb *et al.* introduced freeblock scheduling, which uses rotational latency estimates to fill foreground requests' rotational latency periods with useful background data transfers, thereby increasing disk bandwidth utilization without compromising performance for foreground requests [22]. A prototype freeblock scheduler operating outside disk firmware required extremely accurate predictions of disk service time components [21]. Lumb *et al.* report that sufficient information about black-box disks could be obtained from DIXtrac [28] to support acceptably accurate service time predictions for freeblock's needs. The modeling problem was made easier because a modified device driver limited the number of requests pending in the disk to two [21].

DIXtrac was an important enabling technology for this and several other scheduling studies [4,29]. It automatically obtains detailed data-layout, cache management, and timing data from black-box disks via "controlled experiment," i.e., by submitting carefully selected requests to the disk and measuring response times. Given this information, accurate performance forecasts can sometimes be obtained via simple calculations.

The modeling approaches motivated by individual disk scheduling are not applicable to large storage devices. Throttling the dispatch of requests to a storage array as in the freeblock-in-driver work, for instance, would preclude re-ordering optimizations within the array and thereby degrade performance. Fundamentally different approaches are required for enterprise storage arrays.

## 3.2 Simulation & Emulation

Mature, refined, and well-calibrated simulation models of individual disks have been available for many years. Ruemmler & Wilkes summarize the art of disk drive modeling, describing how increasing fidelity yields improved aggregate accuracy [27]. DiskSim represents the current state of the art in disk simulation; it models device drivers, buses, controllers, adapters, and disk drives [8]. Worthington *et al.* laid the foundations for DiskSim (as well as DIXtrac) nearly a decade ago [36]. The Pantheon storage system simulator can model *collections* of disks, e.g., RAID arrays [35]; RAIDframe supports evaluation of error-free and recovery behavior [19].

*Emulators* go one step further than simulators. In addition to modeling performance, they interoperate with real systems, transparently replacing storage devices and attempting to mimic their behavior. This allows researchers to explore conveniently how the performance of devices that do not yet exist, e.g., MEMS devices, might interact with existing systems [16].

## 3.3 Analytic Models

Management tools that search a large space of storage device configurations for an optimum have shown promise for automating storage management [1, 3]. The inner loop of such tools is a solver that maps device design, configuration, and workload to average performance characteristics. Simulations driven by synthetic or trace inputs are too slow for this inner loop, and researchers have therefore employed faster analytic models for such applications.

Ruemmler & Wilkes consider a continuum of models starting with the simplest possible analytic performance "model," a numeric constant [27]. They add detail and fidelity to obtain better analytic models, e.g., in which response time is proportional to I/O size plus measured seek time, and finally arrive at simulators that consider data layout and the physical state of the disk's moving parts.

Analytic models are often easier to reason about than simulators, and a further advantage is that they sometimes support convenient *composition*. Shriver *et al.*, for instance, develop algebraic descriptions of disk drive components and compose them to model mean disk service times [31]. The analytic composition approach has also been applied to modeling disk *arrays*. Uysal *et al.* validate a composite analytic model of mean throughput against measurements of a mid-range disk array and report agreement to within 15% [33]. More recently Varki *et al.* developed a model that predicts mean response time and queue length in addition to mean throughput [34].

One drawback of analytic models is that human experts are required to define them. *Table-based* models escape this reliance by systematically testing (via controlled experiment) a large number of points in the space of inputs, and interpolating between them to form predictions [2]. Such models require less expertise than conventional analytic models, but more calibration, because they must systematically sample a multi-dimensional space of workloads.

### 3.4 Evaluation Metrics

In cases where models generate individual response time forecasts that can be compared against actual response times, it is conventional to summarize accuracy using the "demerit" measure introduced by Ruemmler & Wilkes [27]. The demerit measure is the root mean square of horizontal distances between the *cumulative distributions* of actual and predicted response times; this can be normalized by dividing by the mean of the actual response times. It is zero for identical distributions, is dimensionless if normalized, and has convenient units (milliseconds) otherwise. Despite its attractive properties it does not measure the accuracy of *individual* forecasts. If an ordered list of actual response times is compared against a forecast consisting of a random permutation of itself, for instance, pairwise predictive accuracy may be very poor but the demerit figure will be zero.

Lumb *et al.* [21] provide the only published report of individual forecast accuracy of which we are aware. They report remarkably accurate *service* time[1] forecasts for an individual disk, particularly for small random reads; up to 99.3% of forecasts were within 50 $\mu$s for a particular disk considered. They achieved such high accuracy in part because they restricted the flow of requests to the disk, preventing re-ordering within the disk from foiling their models of the moving parts. To the best of our knowledge, accuracy results for individual *response* time predictions have not been published.

### 3.5 Summary

The methods reviewed in this section are not well suited to forecasting response times of individual requests to enterprise storage arrays. Analytic models predict *average* performance (e.g., mean throughput, mean response time) from *parametric* workload descriptions (e.g., mean request rate, read:write ratio). Simulation models can generate per-I/O predictions, but they require calibration. A good calibration tool is available for disks, but nothing analogous to DIXtrac exists for enterprise storage arrays. It is not clear that controlled experiments could, in a reasonable amount of time, extract from an enterprise storage array sufficiently detailed information to calibrate simulation or table-based models. The problem at hand requires different methods.

[1] Recall that response time = queueing time + service time.

The approach we pursue in this paper attempts to achieve some of the desirable properties of existing methods via different techniques. Like table-based models we rely on little knowledge of system internals and substitute extensive measurements for expert knowledge. Unlike the table-based approach we rely on purely *passive* observations; we trust the workload to decide what regions of the workload space are important, so to speak. Like simulators and emulators, we form predictions of individual response times quickly enough for online use, but we attempt to model systems whose complexity has thus far defied simulation.

## 4 Modeling Black-Box Arrays

This section formally defines the problem of modeling black-box storage devices using passive observations. It furthermore describes the special properties of large enterprise storage arrays and the challenges they pose to our modeling framework. Finally, it offers straightforward evidence suggesting that our approach is workable: passive external observations of unmodified enterprise storage arrays contain a substantial amount of information about response times.

### 4.1 Formal Problem

Let $\vec{x}$ represent a vector of features describing an I/O request. The feature vector may include several kinds of information: characteristics of the request itself, e.g., the amount of data requested; information about the state of the storage device, e.g., the number of unfulfilled requests within it; and features that relate the current request to earlier ones, e.g., measures of reference locality.

Let $P^*(t_r|\vec{x})$ denote the conditional distribution of an individual request's response time $t_r$ given the information contained in $\vec{x}$. The use of a distribution to characterize response time $t_r$ is necessary in practice because $\vec{x}$ components available via observation of black-box devices do not completely describe all aspects of system state relevant to $t_r$; Section 4.2 discusses limitations of available information in greater detail. A probability distribution can capture the uncertainty inherent in this situation, and also uncertainty arising from sampling and measurement errors. Each value of $\vec{x}$ defines a probability distribution over response time, which can be computed online for a given $\vec{x}$ and $P^*$ or some approximation thereof. Fi-

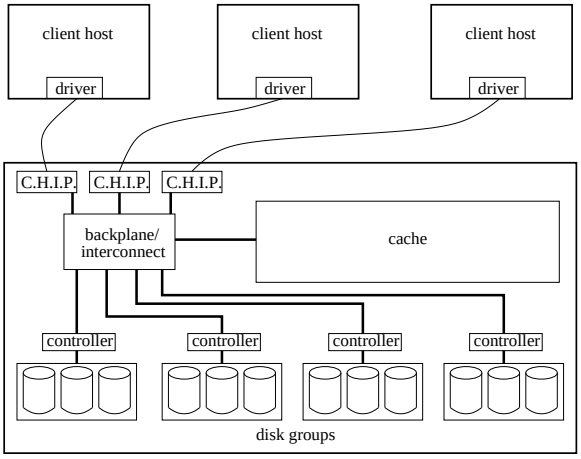

Figure 1: Simplified view of the HP XP 512.

nally, note that it is straightforward to collapse any distribution over continuous $t_r$ values into a binary forecast ("fast" vs. "slow") by simply considering probability mass above and below a threshold on $t_r$; we do this in our evaluations.

Our task in this investigation is twofold. First, we investigate algorithms and statistical models to induce an approximation $P$ of $P^*$ from a trace of $(\vec{x}, t_r)$ pairs. Second, we provide a characterization of the value of incorporating different elements in feature vector $\vec{x}$. The forms of the distributions we consider and the algorithms we use to induce estimates of them are described in Section 6. The remainder of this section explains the special challenges of applying our formal modeling framework to enterprise storage arrays, and suggests that sufficient information is nonetheless contained in $\vec{x}$ to justify the attempt.

## 4.2 Enterprise Storage Arrays

The complexity of individual hard disks and RAID groups is sufficient to pose formidable challenges to performance modeling. We focus on a class of devices that incorporates these as building blocks, and is therefore still more complex: enterprise storage arrays. Figure 1 depicts in highly simplified form the internal architecture of the array on which our trace data were collected, the Hewlett-Packard XP 512 [17]. Although they differ in many internal architectural details, the XP 512 and comparable offerings from EMC [14] and IBM [18] share several characteristics: Each contains a collection of independent disk groups behind a large non-volatile/battery-backed cache, and high-speed point-to-point interconnections and/or crossbar backplanes connect client host interface processors (CHIPs) with the cache and with disk-group controllers.

Enterprise storage arrays typically serve many physically distinct client hosts, to which they are connected via high-speed short-range networks (e.g., Fibre Channel). The mapping of user-visible entities (e.g., files in a filesystem) onto physical media involves several layers of indirection, all of which are highly configurable. Under typical conditions four factors dominate performance:

- *Queueing* in individual disks, in disk groups, in the array controller, and in client host device drivers.
- *Caching* in both the array cache and individual disks.
- *Seeking* to align disk read/write heads.
- *Rotation* of platters.

Throughout this paper we regard enterprise storage arrays as "black boxes" in the sense that we do not instrument or otherwise directly measure any aspect of their internal state. Instead, we consider only quantities observable in client hosts served by the array. This is a severe restriction, because the state of moving parts within the array is unavailable yet clearly crucial to performance; seek and rotation can dominate response times under light load and low cache hit rates. Some performance-related information, however, is available outside the storage array, including the following:

- *Pending requests* that have arrived in the client host device driver but have not yet been returned to the caller.
- *Locality* across accesses.
- *Configuration* of logical units onto disk groups within the array.
- *Sequentiality* of accesses.
- *Arrival times* of requests.

We present results based on the first three of these; in principle our methods could be applied to the last two as well. The remainder of this section shows that observations of the first two quantities contain a great deal of information about the response times of individual requests.

## 4.3 Predictive Value of Observables

Our early investigations revealed that the number of pending requests in a storage array is, by itself, a sur-

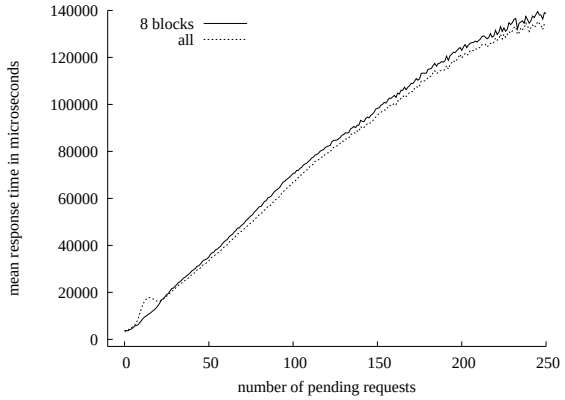


Figure 2: Mean response time vs. # pending requests.

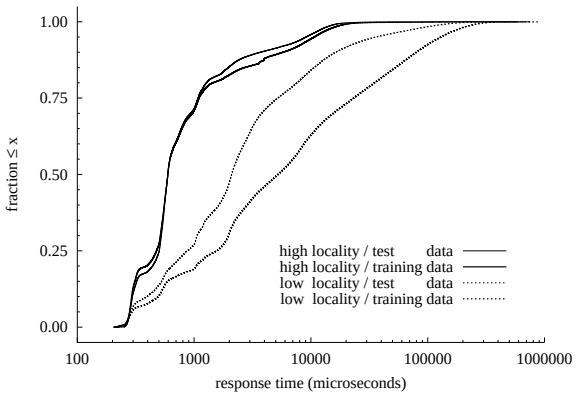


Figure 3: Distribution of response times for high and low locality read requests in training and test data.

prisingly good predictor of *average* response time. Figure 2 shows average response time of over 50 million read requests as a function of the number of requests pending at the time the measured request was issued.[2] The figure shows, for example, that among all reads issued when 100 requests were pending, mean response time was roughly 70 ms. Separate series are shown for all request sizes, and for those that access exactly 8 KB (the most common request size). The horizontal axis is truncated at 250 pending requests; fewer than 0.5% of reads in our data were issued when over 250 requests were pending. The 8-block series shows that mean response time depends roughly linearly on number of pending requests; the relationship is surprisingly simple.

Another notoriously opaque aspect of enterprise storage systems is cache management. Reads and writes may share a common cache, perhaps with dynamically-adjusted lower bounds on the fraction used by each. Interleaved sequential request streams may be identified, disentangled, and given special treatment. Readahead/prefetching may occur. Writes typically complete quickly due to write-back cache management enabled by battery-backed or otherwise non-volatile array caches; written data is de-staged to magnetic media later according to complex rules. Whereas the traditional measure of access locality, LRU stack distance, relates directly to the performance of an LRU cache [23], this measure does not correspond closely to the behavior of today's large storage arrays.

An inexact correspondence, however, can nonetheless be informative. Figure 3 separately plots the distributions of response times among low- and high-locality read requests issued during two consecutive four-day periods (the training and test data described in Section 5). High-locality requests are those for which the LRU stack distance of every accessed block is 1 GB or less; low locality means that LRU stack distance is greater than 32 GB. We see that response times are markedly lower for high-locality requests in both training and test data. The two high-locality distributions are nearly identical; the low-locality curves differ from one another, but not as much as either differs from the high-locality CDFs. Locality is an inexact but powerful predictor of response time.

We see from these simple examples that passive observations outside a black-box enterprise storage array have considerable predictive potential. Performance-critical aspects of system state, e.g., the state of the array cache and of queues within the array, are concealed from view. Observable quantities can, however, help us to refine response-time forecasts. The probabilistic modeling framework sketched in Section 4.1 is well-suited to such situations.

# 5 Traces & Features

This section explains the raw traces upon which our empirical evaluations are based, the feature vector we derive for each request, and the efficiency with which features can be computed.

[2] Data are taken from the 8-day "warm" subset of the trace described in Section 5.

| | full trace | 8-day "warm" trace | |
|---|---|---|---|
| | | train | test |
| Dates | 9/27–10/27 | 10/20–23 | 10/24–27 |
| Reads | 270,828,600 | 20,028,031 | 31,521,165 |
| Writes | 144,559,326 | 15,097,108 | 20,414,054 |
| R blks | 6,809,092,520 | 302,360,743 | 873,790,593 |
| W blks | 1,804,978,882 | 180,477,338 | 254,339,108 |

Table 1: Summary of Fall 2002 harp trace.

## 5.1 Raw Traces

Our empirical work uses a month-long trace of requests to an HP XP 512 storage array collected by the Storage Systems group at Hewlett-Packard Laboratories between 27 September and 27 October 2002. Our data is more recent than the three well-known "cello9x" traces previously made available by HPL [32] that have been used over the years by many storage researchers. We refer to it as the "harp" trace, to distinguish it from its predecessors.

The raw harp trace was collected by HP MeasureWare software (midaemon) running on a client host connected to the storage device. The host in question (named harp) ran the Storage Systems group file and compute server, and much of the trace ultimately derives from day-to-day activity by roughly two dozen computer scientists. These users ran a variety of applications typical of CS research computing, e.g., software development, trace analysis, and simulators; the request stream that reached the storage device is not representative of enterprise workloads, e.g., ERP or CRM packages. The measurement software running on harp records when individual I/O requests arrive in the device driver, depart for the storage array, and return from the array. The trace also records the request type (read or write), the number of 1-KB blocks requested, logical unit (LU) accessed, and the block offset within the LU. From the LU and configuration information we can identify the physical RAID group within the XP 512 that will ultimately serve each request (see Figure 1). The XP 512 had a 16 GB cache.

Table 1 summarizes the full raw trace and the subsets used in our evaluation. One of the features we derive from the raw trace, a locality measure described in Section 5.2, depends upon the history of prior requests. We therefore compute a full feature vector for the entire trace, but use only the last eight days for our evaluation; we refer to this eight-day suffix as the "warm" subset of our trace. Because we induce our performance models from data, we must "train" and "test" on different data in order to assess predictive accuracy. We divide the eight-day warm trace into training and test sets of equal duration, as shown in Table 1.

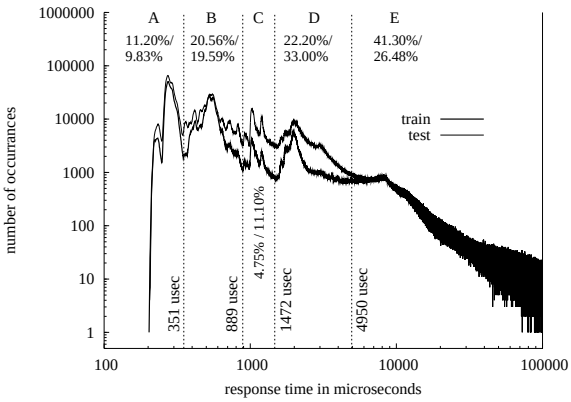


Figure 4: Frequency distributions of read response times in training and test data subsets.

In this paper we model response times for *read requests only*, not writes. Reads are far more common, accounting for roughly two thirds of all requests in the harp trace. Furthermore modeling reads is a greater challenge because the distribution of request times is far wider for reads than for writes (the XP 512's write-back cache ensures that writes are almost always very fast).

Figure 4 shows histograms of response times for read requests in the training and test sets; the figure omits the noisy tails of both distributions, which extend to roughly one second account for a small fraction of all requests. We have drawn boundaries between the five most prominent modes, chosen as the low points between adjacent modes in the training set, and labeled the modes "A" through "E". Mode A corresponds to hits in the XP's array cache, and B corresponds to hits in individual disk caches. Varying combinations of seek, rotation, and queueing explain C, D, and E. The percentages of reads in training and test sets, respectively, that fall into each mode are shown in the figure; for example, 4.75% of reads in the training set fall into mode C, and 19.59% of reads in the test set fall into mode B.

Harp was not the only host connected to the XP 512 during trace collection. Several other hosts shared the storage array with harp, and I/O activity by these other hosts is not reflected in the harp trace. One host performed backups of data on the device (i.e., pulled data from the XP) and we have a par-

tial schedule of backup activity, but do not know the number of requests or volume of data involved. We furthermore know that other hosts wrote data to the device. This untraced activity is cause for concern because it could have contended with the traced workload for the array cache.

For several reasons we believe that the harp trace is useful for our purposes despite untraced activity. The other hosts are thought to have used the XP 512 relatively lightly during trace collection. Furthermore the other hosts did not access the same internal RAID groups on the XP as harp, so we need not worry about contention within RAID groups. Most importantly, our methods are robust to noise; as noted in Section 4.1, a probabilistic modeling framework seamlessly incorporates uncertainty due to sampling and measurement errors. The main implication of untraced activity in the harp trace is that our accuracy results are *pessimistic*; we expect that accuracy would be higher if our trace included all requests handled by the array.

### 5.2 Feature Computation

From the raw harp trace and auxiliary information about the storage array's configuration, we derive a feature vector for use in forecasting response times. We capture the potential for queueing and congestion within the storage device by augmenting each I/O record with the numbers of requests of various types pending in various places at the time that the I/O was issued. We distinguish 1) between pending reads and pending writes,[3] 2) between pending requests destined for the same RAID group as the current request and those aimed at other RAID groups, and 3) between pending requests queued in the client host device driver and in the array. This yields eight features measuring the number of requests in each category. We compute eight analogous features representing the numbers of *blocks requested* across all pending I/Os, for a total of sixteen queue-related features.

We capture access locality by computing an aggregated and rounded LRU stack distance for each request: For a fixed set of cache sizes $\{1\text{ GB}, 2\text{ GB}, 4\text{ GB}, \ldots, 32\text{ GB}\} \cup \{12\text{ GB}, 20\text{ GB}, \infty\}$, we record the smallest size $S$ such that every block accessed by a request has LRU stack distance $\leq S$. Ideally we would simply compute the maximum LRU stack distance across all blocks accessed by a request. However, due to the large number of distinct blocks recorded in the harp trace, this proved to be infeasible on our computational platforms.

In summary, our full feature vector describing a request consists of the number of blocks requested, eight components related to the numbers of requests pending in various places, eight more components describing the total number of blocks requested across pending requests, and our locality measure. In Section 6 we use the locality measure in different ways than the other features, so we denote the former $L$ and the latter $\vec{x}$. Certain auxiliary information, e.g., the RAID group to which the request is directed, is used to partition data but does not inform the model-induction process. We shall not regard this extra information as part of the feature vector.

### 5.3 Online Feature Computation

We induce models that may be used to forecast response times online, i.e., given a feature vector the model can compute a prediction at very low computational cost. If computing the feature vector itself from measured quantities is prohibitively expensive, however, our technique cannot be applied online. Tabulating requests and blocks requested clearly involves constant-time operations. Our locality measure may be computed by simply simulating a collection of LRU caches, updating each of which requires constant time per block accessed. A more elegant approach might involve efficient approximate LRU stack distance algorithms [11]. In practice, our feature computation code requires under 400 $\mu$s per request to process the full harp trace. This code must perform several chores not required for online forecasting, e.g., it maintains a heap of pending requests; we believe that a streamlined implementation specifically designed for online forecasting could achieve much better performance.

## 6 The Probabilistic Model

Recall from Section 4.1 that our task is to induce a model $P(t_r|\vec{x})$ from observed $(t_r, \vec{x})$ pairs. Our starting point, and our baseline for comparison in our evaluations, is the predictive power of the *unconditional*

[3] We attempt to model response times for reads only; however our models consider the numbers of pending reads and writes. Similarly we compute our locality measure using both reads and writes, but we apply it only to modeling read response times.

(or *prior*) distribution $P(t_r)$. In practical terms, a prediction based on the prior is simply the distribution of response times observed in training data. This section describes the form of our models, the algorithms that induce them, and refinements added as we broaden the range of information available to them.

## 6.1 Adding Locality Information

As described in Section 5.2, we compute a locality feature that approximates the size of the smallest LRU cache in which every block accessed in a request would hit. We expect a significant difference in response time between a request that requires a transfer from magnetic media versus one that is satisfied by the storage array's cache or an individual disk cache. We furthermore expect that high-locality requests will tend to hit more often than low-locality requests (Figure 3).

Let $L$ denote our imperfect locality feature. In order to account for the uncertainty surrounding this feature we introduce a "hidden state" $j$, which models the likelihood that a cache hit occurs somewhere in the array given $L$ (e.g., array cache hit, disk cache hit). The variable $j$ is "hidden" because we have no direct observation of its state. Intuitively, its states correspond to modes in the $t_r$ distribution (Figure 4); a particular state of $j$ suggests a distribution of response times (Figure 3). Probabilistically, we have:

$$P(t_r|L) = \sum_j P(t_r|L, j) \times P(j|L) \quad (1)$$

Since we are summing over all the possible states of $j$, the equality is valid. We further simplify this equation by noting that once we know the state of $j$ (i.e., whether we have a cache hit, and in what cache) our knowledge of response time is not affected by the evidence provided by the $L$ variable. In other words, $t_r$ is probabilistically independent of $L$, given $j$.

$$P(t_r|L) = \sum_j P(t_r|j) \times P(j|L) \quad (2)$$

This equation has a natural and intuitive interpretation. The probability of response time depends on whether the request was a cache hit (i.e., given by $j$) weighted by the probability that a hit occurred given the evidence in $L$. In other words the response time is computed as an expectation over the possibility that the request hit in a cache. This expectation is necessary because we have no direct access to the several layers of caches in the device.

Two questions remain: How are we to statistically "fit" the distributions in the right hand side of Equation 2 since we lack explicit samples of the behavior of $j$? How many states should the variable $j$ have? We begin by fixing the number of states in $j$. To address the first question we use a standard algorithm from statistics called *expectation maximization* or EM [5, 24], one of the most powerful and commonly used tools in the statistician's toolkit.

EM is an iterative algorithm reminiscent of gradient descent; it searches over a given likelihood function to obtain a maximum. Informally, the algorithm proceeds as follows: We initialize the algorithm to a model represented by a (possibly arbitrary) probability distribution $P_0$. The algorithm then uses $P_0$ to compute the expected states for $j$ in each sample of the training data. Then, it uses this updated sample set to compute an updated *model* $P'$, and alternately updates data and model until it reaches a fixed point. This fixed point is guaranteed to be a point of locally maximal likelihood (i.e., the final model presents a maximum likelihood to have generated the observed set of samples). The final state depends on the starting point, as well as on the shape of the likelihood function. We based our starting point $P_0$ on a straightforward linear regression over the training data, informed by the variable $L$ (an informed starting point will accelerate convergence). To address the issue of local optima, we perturb the solutions obtained and restart EM from neighboring points.

In order to obtain the number of states for $j$ we perform a search, starting with two states and increasing the number of states looking for the result that yields maximum likelihood. In order to regularize the maximum likelihood score and avoid overfitting we plot these results and select the number of states where the gain in likelihood (the first derivative) starts to decrease significantly.[4] We find that five states of $j$ yields good results; this value is used in all of the experiments reported in Section 7.

We still must specify the actual distributions used. For $P(j|L)$ we used multinomials as both $j$ and $L$ are discrete variables. We use the standard maximum likelihood method for fitting the parameters which in this case reduces to the appropriate frequency counts

[4] Regularization is necessary because at the limit, we will obtain maximum likelihood where the number of states $j$ is equal to the number of points in the data. The regularization technique of plotting the likelihood function versus the number of states in $j$, and then selecting the point at the knee of the curve, is one of many regularization methods.

on the states of $j$ divided by the states of $L$ [5, 9]. We model $P(t_r|j)$ with a Gaussian distribution. Note that this is not as naïve as it may seem. Given the fact that we have an additional degree of freedom with the number of states of $j$, we are actually using a mixture of Gaussians. Given enough states for $j$, a mixture of Gaussians can model any distribution, just as given enough linear segments we can model any function. Of course we run the danger of overfitting the data, which in our case we avoid by regularizing on the number of states. It is not surprising that $j = 5$, i.e., a mixture of five Gaussians, works well, given the five prominent modes of Figure 4.

In summary, we model $P(t_r|L)$ using a mixture of Gaussian distributions, informed by the variable $L$. We obtain a model that maximizes the likelihood of the observed data, given the model.

## 6.2 Adding $\vec{x}$: Mixtures of Regressions

We can further refine our model by providing it with the full feature vector $\vec{x}$ described in Section 5, which includes the number of blocks read and the number and total size of requests pending in various places in the storage array. Formally, we augment Equation 2 as follows:

$$P(t_r|L,\vec{x}) = \sum_j P(t_r|j,\vec{x}) \times P(j|L,\vec{x}) \tag{3}$$

Again, assuming that $\vec{x}$ contains no information pertinent to the relation between $j$ and $L$ (i.e., assuming statistical independence between $j$ and $\vec{x}$ given $L$) we rewrite Equation 3 as

$$P(t_r|L,\vec{x}) = \sum_j P(t_r|j,\vec{x}) \times P(j|L) \tag{4}$$

The main addition to the previous model is that in our mixture, we actually take into account the information in $\vec{x}$. We fit a standard least squares regression for each member of $P(t_r|j,\vec{x})$ [26].[5]. The value of increasing the complexity of the model is given by the increase in accuracy obtained.

## 6.3 Classification Model

This model directly addresses the fact that we are interested in distinguishing between two modes of operation, *fast* and *slow*.[6] This is a standard pattern classification problem [5, 13] and we can fit probabilistic models that are specialized for these problems. Now $t_r$ is a discrete variable that takes two values $t_r^+$ and $t_r^-$, which denote fast and slow respectively. Then we are really interested in finding whether

$$\log \frac{P(t_r^+|\vec{x},L)}{P(t_r^-|\vec{x},L)} + d \geq 0 \tag{5}$$

where if $d = 0$ we are basically saying that if $P(t_r^+|\vec{x},L) \geq P(t_r^-|\vec{x},L)$ then the response time will be fast.[7] The issue is now, how to fit the conditional distributions in Equation 5. From the many possibilities we borrowed a convention from the recent pattern recognition literature, which reportedly yields good results in domains as diverse as natural language processing and systems research. Using Bayes rule we can rewrite the conditional in Equation 5 as:

$$P(t_r^+|\vec{x},L) = P(\vec{x},L|t_r^+) \times \frac{P(t_r^+)}{P(\vec{x},L)} \tag{6}$$

Now assuming that all of the members $x_i$ of $\vec{x}$ are independent of each other given the state of $t_r$ and making the appropriate substitutions in Equation 5, we obtain

$$\log \frac{P(t_r^+|\vec{x},L)}{P(t_r^-|\vec{x},L)} = \sum_i \log \frac{P(x_i|t_r^+)}{P(x_i|t_r^-)} + \log \frac{P(L|t_r^+)}{P(L|t_r^-)} + \log \frac{P(t_r^+)}{P(t_r^-)} \tag{7}$$

This equation has a very intuitive interpretation. It evaluates the decision as a linear combination (in log space) of the contribution of each feature in $\vec{x}$ and $L$. This is known as the "naïve-Bayes classifier." The assumption of independence is clearly unrealistic, but this classifier has a number of advantages that account for its popularity [12, 13]. First it is very robust to noise in the data. As a consequence it works well in high dimensional spaces. Note that statistically we are breaking a high dimensional space (fitting a conditional distribution on $\vec{x}$ and $t_r$) into a number of small dimensional spaces, where we fit univariate $P(x_i|t_r^+)$ distributions. The assumption of independence may hurt us in finding the exact probability,

[5] Strictly speaking, by performing a least square regression we are indeed fitting a Gaussian distribution, one in which the mean is linearly dependent on the elements of $\vec{x}$.

[6] The generalization where we quantify (discretize) the range of possible response times $t_r$ in a set of finite regions of interest follows directly from the exposition in this subsection.

[7] The threshold $d$, denoting how much more probable one response time has to be in order to be selected can be adjusted according to criteria such as minimizing false-positives, etc.

yet, the important task here is finding the right separating surface between the two regions of $t_r^+$ and $t_r^-$ (i.e., fast and slow). We tested a more sophisticated model that removes this assumption of independence by fitting a full covariance matrix; accuracy did not improve for our data.

### 6.4 Online Forecasting

Our model induction procedures are not very computationally expensive: the process of automated model induction, refinement, *and testing* together took a few hours for 20 million training data points on an inexpensive laptop computer. Currently we regard model induction as an offline procedure, primarily because EM is an iterative algorithm. Online EM updating approaches exist [25], so it may be possible to induce a model offline and update it online.

Forecasting individual response times is feasible online. Given an induced model and a feature vector $\vec{x}$, computing a response time forecast $P(t_r|\vec{x})$ involves only constant-time operations and a constant number of calls to an error function implementation (`erf()` on most Unix systems.) We find that `erf()` requires roughly 0.235 $\mu$s on an inexpensive Intel/Linux laptop, and we estimate that $P(t_r|\vec{x})$ can be computed in well under one microsecond. For comparison, the fastest response time in our trace was exactly 200 $\mu$s.

### 6.5 Data Subsets

Exploratory analysis of the harp trace revealed distinct regions of the feature vector space that called for different model parameterizations. We therefore subdivide our data in several ways before applying our forecasting algorithms. A handful of particular values of our feature vector $\vec{x}$ account for a large fraction of reads in our trace, e.g., many requests read eight blocks at times when no requests are pending anywhere in the storage device. The distribution of response times for each of these "pathologically popular" $\vec{x}$ values spans orders of magnitude, because our $\vec{x}$ does not capture all relevant system state, e.g., moving parts, for these cases. (This is to be expected because we regard the storage array as a black box; our feature vector derived, from external observations, is incomplete.) We have taken the obvious approach of handling the "pathologically popular" $\vec{x}$ values separately. We classify an $\vec{x}$ value as pathological if it appears with 1% or more of reads in our training set. Exactly seven $\vec{x}$ values satisfy this criterion, and 38.8% of reads in our warm (training + test) data falls into this category.

We partition the remaining non-pathological data according to the number of requests pending anywhere (client host device driver or storage array) at the time the request is issued. If this number is greater than twenty the request is placed in the "long queue" category, otherwise it is designated a "short queue" request. The threshold of 20 lies just above a nonlinear irregularity in the plot of mean response time vs. number of pending requests (see Figure 2, "all" series, lower left). The short- and long-queue data account for 52.1% and 9.1% of the warm trace, respectively. Finally, we induce and evaluate models separately for requests destined for each of seven RAID groups and aggregate accuracy results. This is natural because, by design, RAID groups are intended to be largely performance-isolated from one another; furthermore it improves accuracy. We do not consider requests destined for RAID group 4, which account for less than 0.6% of the warm trace, because too few requests access it.

In summary, the distribution $P_G(t_r|\vec{x})$ for each of seven RAID groups $G$ is composed of nine distributions: two for the different regions defined by the number of pending requests, and seven defined by the "pathologically popular" instances of $\vec{x}$. Thus we have:

$$P_G(t_r|\vec{x},L) = \begin{cases} P_{Gi}(t_r|\vec{x},L) & i \in \{1,\ldots,7\}, \text{if pathological} \\ P_{Gh}(t_r|\vec{x},L) & \text{if} > 20 \text{ requests pending} \\ P_{Gl}(t_r|\vec{x},L) & \text{if} \leq 20 \text{ requests pending} \end{cases}$$

## 7 Evaluation & Results

We evaluate the accuracy of our methods by applying them to a binary classification problem: we define a boundary between "fast" and "slow" response times, use a forecasting method to select one of these categories for each read request in our test set, and report how often we predict correctly. This allows us to evaluate all of the algorithms described in Section 6 in the same framework. Our mixture-of-regressions models yield forecasts in the form of weighted sums of Gaussian distributions over response time; we collapse these to a binary forecast by simply computing the probability mass in both fast and slow categories and choosing the more probable. Note that generalizing this approach to more than two categories is straightforward.

The fast/slow threshold will depend on the application in which forecasts are used, and must be chosen tastefully; if it is set too high, for instance, a trivial forecasting method that always guesses "fast" will achieve high accuracy. We define response times no greater than 1472 $\mu$s to be fast. This is the lowest point in the response time distribution between modes representing "clearly electronic" and "clearly mechanical" speeds (modes B and D in Figure 4). 36.5% of reads in our training set and 40.5% of reads in our test set have response times at or below this threshold. As noted in the discussion surrounding Table 1 in Section 5.1, our warm data is divided into two four-day periods, a training set and a test set. We induce models based only on the training data and measure their accuracy using the test set.

Table 2 presents our results. The top row shows the breakdown of our test set into three categories (pathological, short queue, and long queue); absolute and relative accuracy data appear beneath.

Overall, forecasts based on the training set's prior probabilities alone are correct in 18.6 million out of 31.5 million test cases—less than 60% of the time. The number of correct predictions increases to 19.5 million when we take locality into account (see Section 6.1), roughly a 5% increase in the number of correct classifications. When we incorporate all available information—locality as well as other components of our feature vector—using a mixture-of-regressions model (Section 6.2) we forecast 21.2 million requests correctly. When we apply naïve-Bayes classification to all available information (Section 6.3), accuracy increases for both long and short queues: we classify roughly 21.4 million requests correctly, an increase of 14.7% compared with the prior.

Predictive accuracy varies within each of the three subsets of our data. Predictions based on the prior are slightly *worse* than a coin toss for the pathological data, but we correctly classify 9% more transactions when we take into account all available information. Sophisticated forecasting methods yield little benefit when many requests are pending; for the long-queue data, mixed-regression models yield only modest improvement over the prior, and naïve-Bayes classification *reduces* accuracy. Simply knowing that queues are long leads us to predict "slow," which is right over 95% of the time; roughly 5% of the data fall into the long-queue category.

The short-queue data represents an interesting intermediate point between the long-queue and pathological subsets, and it accounts for nearly two thirds of the test data. A diverse variety of feature vector values are available in the short-queue subset (unlike in the pathological subset), and prior prediction alone leaves room for improvement (unlike in the long-queue subset). In this case, naïve-Bayes classification increases the number of correct predictions by over 20%, from 12.2 million to 14.7 million.

Table 3 decomposes our accuracy results by RAID group for the short queue subset of our data (recall from Section 6.5 that RAID group 4 is omitted because very few requests access it). The table shows that accuracy is higher in some RAID groups than in others; this is due to differences in workload and perhaps also to the untraced activity described in Section 5.1. Furthermore note that the relative accuracy of our two modeling techniques varies by RAID group, and therefore we can obtain better overall accuracy by selecting the best modeling technique for each RAID group. In the best case accuracy approaches 85%.

It is difficult to compare the accuracy of our model with that of published alternatives for two reasons. First, to the best of our knowledge, no models that predict the response times of individual requests in enterprise storage systems have been published. Second, existing results on the accuracy of response time forecasts for smaller devices, e.g., individual disks, compare the *distributions* of predicted vs. actual response times using the "demerit" figure. Demerit scores do not measure the accuracy of individual forecasts (see Section 3.4).

Our models associate probabilities with forecasts (e.g., "the probability that this request will be slow is 31%"). We can therefore evaluate the *calibration* as well as the accuracy of our binary forecasts. A model is said to be well-calibrated if, on average, it neither overstates nor understates the probability that a request will be slow. If a model is well-calibrated, roughly $X$% of all requests deemed "slow with probability $X$%" are in fact slow. Space limitations prevent us from presenting a detailed evaluation of our models in terms of calibration and related statistical "scoring rules" for forecasters [7, 10]. Briefly, we find that the mixture-of-regressions model yields well-calibrated predictions; the naïve-Bayes classifier is less well calibrated, but by applying corrective procedures we can obtain a well-calibrated naïve-Bayes classifier.

| | Pathological | Short Queue | | Long Queue | | Total | |
|---|---|---|---|---|---|---|---|
| # in test set | 9,674,514 (30.69%) | 20,137,832 (63.89%) | | 1,708,819 (5.42%) | | 31,521,165 (100.00%) | |
| Accuracy: | | | | | | | |
| Prior only | 4,792,487 (49.69%) | 12,191,400 (61.28%) | | 1,634,494 (95.66%) | | 18,618,381 (59.07%) | |
| Locality only | 5,118,133 (53.07%) | 12,769,840 (64.19%) | | 1,635,043 (95.69%) | | 19,523,016 (61.94%) | |
| | | MR | NB | MR | NB | MR | NB |
| All | 5,230,095 | 14,314,920 | 14,692,070 | 1,635,511 | 1,432,940 | 21,180,526 | 21,355,105 |
| Features | (54.23%) | (71.95%) | (73.85%) | (95.72%) | (83.87%) | (67.19%) | (67.75%) |

Table 2: Number and (percentage) of I/Os classified correctly. "MR" is mixture-of-regressions (Section 6.2); "NB" is naïve-Bayes classifier (Section 6.3).

| | RAID group | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 5 | 6 | 7 | 8 |
| # reads | 1,710,023 | 1,894,178 | 4,116,476 | 2,031,880 | 2,012,381 | 4,096,683 | 4,033,536 |
| NB | 59.33% | 68.38% | 78.09% | 69.26% | 69.07% | 76.36% | 80.38% |
| MR | 70.08% | 70.40% | 84.65% | 74.59% | 74.59% | 64.85% | 65.08% |

Table 3: Number of reads and accuracy by RAID group for short queue data, naïve-Bayes classifier (NB) and mixed-regression model (MR).

# 8 Discussion

At first glance, forecasting the response times of individual read requests in enterprise storage arrays might appear to be a hopeless undertaking. Performance-critical moving parts lie at the bottom of these devices' architectural hierarchy, obscured behind opaque layers of controllers, queues, and caches whose behavior is highly complex and often proprietary or undocumented. In addition to these fundamental challenges, our investigation considers the further restriction that only passive observations are available. Finally, the measurements available to our induction algorithms were incomplete, because they pertain to only one of several hosts served by the device we studied.

Despite these formidable challenges, our results demonstrate that well-understood statistical methods can induce informative models of enterprise storage array performance. The models require relatively little knowledge of device internals and relatively little domain expertise. Model induction, refinement, and testing require modest computational resources, suggesting that statistical approaches can be cost effective when compared to expert- and knowledge-intensive analytic and simulation approaches. We believe that the methods we have used can generalize easily across a wide range of enterprise storage arrays, because we have incorporated none of XP 512's particular characteristics in our work. We furthermore believe that the predictive accuracy of our models will improve if controlled experiments are employed; these can be guided by the induced models themselves using standard techniques [6].

# Acknowledgments

We thank John Wilkes, Arif Merchant, and Mustafa Uysal, for useful discussions, descriptions of modern storage devices, and feedback. Hernan Lafitte provided assistance with systems administration. Dawn Banard helped to develop our modeling methodologies. Mahesh Kallahalla reviewed drafts and provided useful feedback.

# References

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

[2] E. Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-4, HP Labs Storage Systems Program, July 2001.

[3] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: Running circles around storage administration. In *FAST*, Jan. 2002.

[4] E. Bachmat and J. Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *SIGMETRICS*, June 2002.

[5] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford, 1995.

[6] G. Box and N. Draper. *Empirical Model-Building and Response Surfaces*. Wiley, 1987. ISBN 0-471-81033-9.

[7] G. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.

[8] J. S. Bucy and G. R. Ganger. The DiskSim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University CS Dept., Jan. 2003.

[9] M. DeGroot. *Probability and Statistics*. Addison-Wesley, 1986. ISBN 0-201-11366-X.

[10] M. H. DeGroot and S. E. Feinberg. The comparison and evaluation of forecasters. *The Statistician*, 32(1):12–22, Mar. 1983.

[11] C. Ding and Y. Zhong. Predicting whole-program locality through reuse distance analysis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2003. `http://www.cs.rochester.edu/~cding/Documents/Abstracts/pldi03.html`.

[12] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[13] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[14] EMC Corporation. Symmetrix high-end servers, Sept. 2003. `http://www.emc.com/products/systems/DMX_series.jsp`.

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol—HTTP/1.1, June 1999.

[16] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger. Timing-accurate storage emulation. In *FAST*, pages 75–88, Jan. 2002.

[17] Hewlett-Packard Corporation. hp StorageWorks disk array xp512, Sept. 2003. `http://www.hp.com/products1/storage/products/disk_arrays/highend/xp512/`.

[18] IBM Corporation. Enterprise Storage Server (ESS), Sept. 2003. `http://www.storage.ibm.com/disk/ess/index.html`.

[19] W. V. C. II, G. Gibson, M. Holland, and J. Zelenka. A structured approach to redundant disk array implementation. In *International Performance and Dependability Symposium*, Sept. 1996.

[20] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7, Hewlett-Packard Laboratories, Mar. 1991.

[21] C. R. Lumb, J. Schindler, and G. R. Ganger. Freeblock scheduling outside of disk firmware. In *FAST*, pages 275–288, Jan. 2002.

[22] C. R. Lumb, J. Schindler, G. R. Ganger, D. F. Nagle, and E. Riedel. Towards higher disk head utilization: Extracting free bandwidth from busy disk drives. In *OSDI*, pages 87–102, Oct. 2000.

[23] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[24] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997. ISBN 0-471-12358-7.

[25] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, 1998.

[26] J. Neter, M. H. Kutner, C. J. Nachtsheim, and W. Wasserman. *Applied Linear Statistical Models*. Irwin, fourth edition, 1996. ISBN 0-256-11736-5.

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

[28] J. Schindler and G. R. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, Carnegie Mellon University CS Dept., Dec. 1999.

[29] J. Schindler, J. L. Griffin, C. R. Lumb, and G. R. Ganger. Track-aligned extents: Matching access patterns to disk drive characteristics. In *FAST*, pages 259–274, Jan. 2002.

[30] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *USENIX Winter*, pages 313–323, Jan. 1990.

[31] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *SIGMETRICS*, June 1998.

[32] Storage Systems Lab, Hewlett-Packard Labs. Publicly-available storage traces ("cello9x traces"), Sept. 2003. `http://tesla.hpl.hp.com/public_software/`.

[33] M. Uysal, G. A. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays. In *MASCOTS*, pages 183–192, Aug. 2001.

[34] E. Varki, A. Merchant, J. Xu, and X. Qiu. An integrated performance model of disk arrays. In *MASCOTS*, Oct. 2003.

[35] J. Wilkes. The Pantheon storage-system simulator. Technical Report HPL-SSP-95-14, HP Labs, Dec. 1995.

[36] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-line extraction of SCSI disk drive parameters. In *SIGMETRICS*, pages 146–156, May 1995.