Performance Modeling of Storage Devices using Machine Learning

Mengzhi Wang

CMU-CS-05-185

January 23, 2006

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
Anastassia Ailamaki, Chair
Anthony Brockwell
Christos Faloutsos
Gregory R. Ganger
John Wilkes, Hewlett Packard Laboratories

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

Copyright Notation

This research was sponsored by the National Science Foundation under grant nos. BES-0329549, CCR-0205544, CCR-0326453, IIS-0133686, and CCR-0085938. It was also sponsored through generous grants from the EMC Corporation and the Intel Corporation, and a fellowship from the IBM Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Dedicated to Michelle and Hongliang

Abstract

Performance models of storage devices make it possible to evaluate storage resource configurations efficiently, allowing systems to search automatically a large number of candidates before locating an optimal or near-optimal one. This thesis explores the feasibility of using machine learning techniques to build such performance models. The models are constructed through "training", during which the model construction algorithm observes storage devices under a set of training traces and builds the models based on the observations. The main advantage of the approach is the automation of the model construction algorithm, in addition to the high efficiency in both computation and storage.

In our design, the models represent an I/O workload as vectors, and model its performance on storage devices as functions over the vectors using a regression tool. We have identified that vector representation of workloads, the regression tool, and training traces are three important factors in model quality. This thesis provides a thorough evaluation of existing techniques in addressing these issues. In addition, we have proposed the entropy plot to characterize the spatio-temporal behavior of I/O workloads and the PQRS model to generate traces of given characteristics to augment existing work in workload characterization.

Our experiments on real-world traces have shown that the learning-based models are fast and accurate when the training and testing traces are similar. Offline training using synthetic traces, however, is less effective because the synthetic trace generators fail to capture the strong correlations between requests. Our error analyses have shown both the vector representation and synthetic trace generators have space for further improvement.

Acknowledgements

During my years in Carnegie Mellon University, I have received help from a lot of people. This thesis would never exist without them.

First, I would like to express my deep appreciation to my advisor, Dr. Anastassia Ailamaki, for her insight and guidance, for her financial and moral support in my work, for her strong confidence in me, and for everything else she has done for me throughout my graduate studies. She has taught me not only how to do research, but also to keep faith in myself. This precious gift has helped me to get through my Ph.D. study, and will keep on benefiting me for my entire life. I can never enumerate all the things I have learned from her, just like that I can never express my thanks enough.

I would also like to thank Dr. Christos Faloutsos for always welcoming me with an open door and sharing his wisdom with me. I have benefited a lot from the fruitful discussions during these years. A special thank you needs to be given to Dr. Gregory R. Ganger for the support and inspirations I have received from him. Thanks to Dr. Anthony Brockwell for sharing his expertise and research experience with me. I would also like to express my gratitude to Dr. John Wilkes for the highly constructive feedback during my work and the necessary traces and software he provided to finish my thesis.

I would also like to thank IBM Toronto for their financial support and hospitality in hosting my CAS projects. It has been a great pleasure to know and work with Berni Schiefer, Kelly Lyons, Calisto Zuzarte, and Sunil Kamath.

I feel extremely blessed to have had the support of many people throughout my graduate studies. I would like to thank all the members of the database group and PDL, including but not limited to Stavros Harizopoulos, Kinman Au, Stratos Papadomanolakis, Minglong Shao, Jiri Schindler, Spiros Papadimitriou, Eno Thereska, Jay Wiley, John Strunk, Mike Mesnier, Brandon Salmon. Karen Lindenfelser, Joan Digney, Charlotte Yano, and Sharon Burks are among the others that have provided me with their fast responses to my miscellaneous requests.

Last but not the least, I would like to thank my family for the their love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automated system administration is becoming more and more important as system management cost becomes the dominant factor. One aspect of autonomic systems requires the ability of dynamic resource reallocation to deliver optimal performance and to cope with workload and hardware changes. Performance models of system components provide an efficient evaluation of resource allocation plans. This thesis explores the feasibility of using machine learning tools in modeling storage devices.

This chapter gives a brief overview of the entire thesis by starting with the motivation and background of the storage device modeling problem in Section 1.1. Section 1.2 provides the formal problem definition and the thesis statement. Section 1.3 summarizes my contributions, and Section 1.4 presents the organization of the thesis.

1.1 Autonomic Systems

The scale and complexity of large computer systems have grown too rapidly for human administration to keep up [50, 71]. For example, storage management cost is typically twice the cost of storage systems [89, 90], and a similar breakdown is observed in database systems, too [27]. Both industry and academia are taking initiatives to build autonomic systems that can understand and monitor themselves, adapt to varying circumstances, and allocate resources to deliver optimal performance. A large number of active research projects aim at automating system management for various applications, including general-purpose resource allocation [17, 72], file systems [57], e-commerce sites [56], storage system management [10, 41, 101, 121] and database systems [28, 52, 81].

In traditional hand-optimization, system administrators typically pick several candidate configurations based on their experience and select the best one by evaluating the candidates in actual deployment. Unfortunately, this approach is less feasible nowadays for two reasons. First, it is extremely challenging for administrators to achieve a comprehensive understanding of today's systems simply because of their scale and complexity. At the same time, the number of possible configurations scales exponentially with system sizes, and so does the difficulty in locating good candidate configurations. Second, evaluating candidates in actual deployment is expensive and sometimes unfeasible. For example, during online performance tuning, the resources can be in use, thus unavailable for such evaluation. Therefore, it is valuable to be able to evaluate resource allocation plans efficiently without the need of physical deployment.

Performance models of system components offer such an ability. These models can take workloads as input and predict their performance on system components accurately and quickly. Consequently, systems can use the models to evaluate a large number of candidate configurations and automatically locate high-quality ones in a short period of time. In fact, some work has formulated resource planning into an optimization problem, and used performance models to find good solutions [8]. The following two sample applications further illustrate the importance of storage device models in storage resource management.

Application 1: Automatic data layout. The data layout problem is to find a good layout of a set of data objects on a set of storage devices for a given workload. The data objects can be files or logical volumes for file servers or tables and indexes for database servers. The workload consists of the disk accesses of the data objects. A good layout could be defined on some desired performance metrics, such as the maximum throughput or the average response time over a certain period of time.

A good layout should not only balance the load across all the devices, but take into consideration the interleaving and compatibility of the accesses on different objects. For example, if two objects are never accessed simultaneously, we can safely put them on the same device even though the total traffic volume could be high. When formulated as an optimization problem, the automatic data layout evaluation algorithm can take advantage of performance models to efficiently evaluate the performance of candidate data layouts.

Application 2: Storage provisioning. Storage provisioning finds a set of storage devices for a given workload to meet both financial and performance goals. Unlike in the previous problem setting, one is free to choose storage devices. Finding a good storage configuration involves two steps: a) search among a large number of storage devices and b) evaluate the performance of each set of devices. Similarly, efficient and accurate device models are useful in comparing candidates.

The above two sample applications have shown that performance models for storage devices are valuable in autonomic storage resource allocation. This thesis focuses on exploring the feasibility of using machine learning techniques to build such performance models. Unlike traditional analytical models and disk simulators, our approach bases our device models on machine learning tools, and the models are constructed by observing storage devices under a set of training traces. The main advantage is automated model construction, in addition to high efficiency, accuracy, and a compact model representation. The model construction algorithm treats modeled devices as "black boxes", requiring no knowledge of the device internals and no human intervention, making it accessible to a larger audience. The approach also allows manufacturers to ship device models together with their products without the worry of revealing their technologies.

1.2 Problem Definition and Challenges

Performance models for storage devices are useful in evaluating storage resource allocation plans. Our thesis considers only block-storage devices, such as single SCSI or IDE disks and disk arrays consisting of multiple disks. A performance model for a given storage device takes an I/O workload as input and predicts the aggregate performance of the workload on the modeled storage device. We define an I/O workload as a sequence of requests, r_is, and the aggregate performance metric can be any percentile or the average response time over a certain period of time. We choose to predict the aggregate performance P_j in the j-th time interval, also known as the j-th fragment. The model does not predict the maximum throughput. Instead, we assume an open system, and the model predicts the aggregate response time given a detailed I/O trace, which consists of a sequence of disk requests. The storage device can be either a single disk or a disk array. The model should be efficient, accurate, compact, and easy to construct.

Traditional models for storage devices fall into two categories: disk simulators and analytic models. Disk simulators simulate the physical mechanism of storage devices using software and can therefore estimate per-request response times accurately given calibrated device parameters. Running simulations is slow and expensive, not suitable for resource planning tasks. Analytic models apply a set of formulas to input workload characteristics to predict device performance. Similar to disk simulators, building analytic models requires a deep understanding of the storage device internals. Incorporating new features into existing models usually involves human effort to understand the new features. The learning-based modeling approach, in contrast, removes humans from the development cycle, making model construction fully automated.

The learning-based models are difficult to construct because of the complexity of storage devices and I/O workloads. Storage devices show strong non-linear behavior because of the physical machinery in disks. Performance optimization, such as prefetching and caching, further complicates the situation. In addition, I/O workloads usually exhibit bursty arrivals and complex access patterns, and device performance depends on these characteristics. Performance modeling of storage devices requires a deep understanding of the interaction between workload characteristics and storage device behavior.

1.3 Automated Device Modeling

This thesis explores the feasibility of the learning-based approach in modeling storage devices. Our approach employs a special kind of machine learning tool, “regression tools”, to model target performance as functions of input I/O workloads. The models learn the functions by observing storage devices under a set of training traces for a certain period of time, known as “training”. This thesis investigates the design space of the learning-based modeling approach and evaluates the effectiveness of some existing techniques in implementing the approach.

Existing regression tools model real-value functions on a multi-dimensional Cartesian space. Therefore, an I/O workload should be internally represented as vectors in such a space before one can use regression tools to model device behavior. Figure 1.1 illustrates the automated model construction algorithm. First, a set of training traces are replayed on the target device to obtain response time information. Second, the important characteristics of the traces are represented as vectors. The vectors and response time information provide a set of samples for a regression tool to build the device model. A sample consists of the vector presentation of a sequence of disk requests and the aggregate performance of the requests. Third, a regression tool is used to build the device model from the samples. The model consists of the feature extraction algorithm and a regression model, which maps the vector representation of workloads to their performance on the modeled device. Once properly designed, the training can be fully automated.

During prediction, the feature extraction algorithm first converts an input I/O workload into vectors, and the regression model predicts workload performance from the vectors. With an appropriate regression tool, performance prediction can be accurate and efficient.

Implementing this approach involves three issues.

1. Design the vector representation of I/O workloads. The vector representation should capture sufficient workload characteristics for the regression tool to distinguish workloads of different performance. Moreover, the representation should be compact to

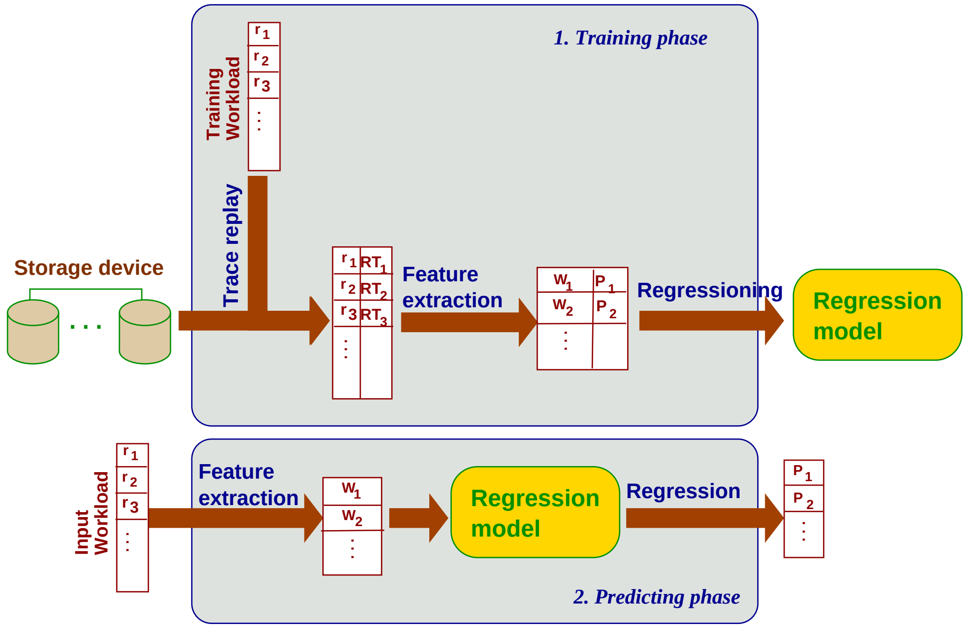


Figure 1.1: The learning-based modeling approach for storage devices. The device model takes an I/O workload as input and predicts the aggregate performance P_js of the workload on the modeled device. During training, the response times, RT_is, of training traces are obtained by replaying the traces on the device. The traces are then broken into fragments, a pair (W_j, P_j) is produced for each fragment, with W_j describing the important workload characteristics and P_j the aggregate performance of the fragment. A regression tool then models P_j as a function of W_j. During prediction, a workload is first converted into W_j, and the regression model predicts P_j from W_j.

have a fast training because the training length is linear to the number of samples needed to cover the vector space, and the number of samples is usually exponential to the dimensionality of the vectors. This phenomenon is known as the "curse of dimensionality" [16].

2. Select an appropriate regression tool to deliver accurate, efficient, and compact regression models.

3. Provide a set of training traces. The traces should offer a wide range of access patterns for device models to capture device behavior under different types of workloads because the models rely on training to learn device behavior.

In this thesis, we have explored the effectiveness of existing techniques in solving the above three issues and have designed two new tools, the entropy plot and PQRS model, to augment existing work in workload characterization. In addition, we use a sample application to illustrate the effectiveness of the device models in helping resource allocation.

Thesis statement. The learning-based approach can provide accurate, efficient, and compact models of storage devices for evaluating resource allocation plans if online model adaptation is allowed.

The major contributions of the thesis include:

- Evaluating the effectiveness of existing work in implementing the learning-based modeling approach for storage devices. Experiments on real-world traces have shown that when the testing traces are similar to the training traces used, these models can make decent predictions (a median relative error of 10% to 40% for most traces and devices.) Otherwise, the prediction error could be more than doubled due to the poor quality of synthetic training traces compared to the former case. We further show that when training data is collected and added to the models continuously, the model accuracy can be improved greatly.

- Proposing two new tools, the entropy plot and PQRS model. The former can efficiently characterize the arrival and locality patterns of I/O workloads with three scalars, becoming an important part of the vector representation of I/O workloads. The latter can generate traces with given characteristics and provide traces for training.

- Developing a methodology to analyze model error using empirical studies, including error breakdowns by error sources and performance comparisons between synthetic and real-world traces. The analysis has shown that the quality of the training traces is a major error source.

- Evaluating the usefulness of the learning-based models in a sample resource allocation application, physical database design. The device models have provided a correct ranking of the candidate designs even with inaccurate predictions.

1.4 Organization of Thesis

This thesis describes the learning-based modeling approach for storage devices. Chapter 2 summarizes previous work in related areas, including autonomic computing, storage device modeling, and workload characterization. Chapter 3 gives an overview of our approach and identifies three important issues in implementing the algorithm. The next three chapters, starting from Chapter 4 to 6, give a detailed description of our contributions in addressing the three issues. Chapter 7 presents an error analysis methodology and shows that synthetic trace generators need further improvement in order to provide realistic traces for constructing accurate device models. Chapter 8 illustrates the effectiveness of our device models using a real application. Finally, Chapter 9 summarizes the conclusions, and Chapter 10 discusses future work.

Chapter 2

Related Work

This chapter lists representative work in the areas of automated system administration, storage device modeling, and workload characterization.

2.1 Automated System Administration

System administration has become more and more expensive due to the increasing scale and complexity of large computer systems. Research teams from both industry and academia have started to build autonomic systems that can carry out a large range of administration tasks by themselves to reduce administration cost [23, 32, 50, 53, 71, 101, 118]. Autonomic systems have four aspects: self-managing, self-configuring, self-protecting, and self-healing [50]. Commercial products, such as AutoRAID [121], SQL Sever [52], DB2 [100], and Oracle [81], include self-management components to reduce administrators' burden.

Performance modeling is an important element in implementing the self-configuring feature of autonomic systems because it can allow a fast and efficient evaluation of system performance for making resource allocation decisions. Some systems, such as AutoRAID [121], use heuristic rules to make resource management decisions. Full-fledged performance models, however, can handle more complex interaction between workloads and the storage devices than heuristic rules. One technique for automated resource management is control-theoretic feedback loops [39, 48, 56, 57]. Some systems employ static or simple dynamic rules to guide resource allocation [102, 114, 118]. Others formulate the problem as an optimization problem and take advantage of combinatorial search techniques and performance models to find a good allocation plan [8, 17, 72]. Performance models allow the systems to compare different allocation plans efficiently and accurately without the need for actual deployment of workloads. For example, analytic queuing network models are used in resource allocation for large data centers [17, 72]. Anderson et al. [8, 10] combined a randomized greedy search algorithm and table-based models for storage devices to solve the storage provisioning problem. Borowsky et al. [19] studied capacity planning under intermixing phased workloads. Recent work by Thereska et al. [105] provided a framework to monitor system activities for answering "what-if" questions using operational laws.

Commercial database systems have also incorporated various features to achieve easy management [52]. Capacity planning is known as physical database design in database terminology. Several projects have studied the automated physical database design problem [3, 67, 88]. AutoPart [83] optimizes sequential accesses to scientific databases by automatically partitioning databases. Oracle [81] provides Automated Storage Management(ASM), which automatically replicates and stripes data across all the devices, to reduce administrators' tasks. LEO [100], the DB2's LEarning Optimizer, uses statistics collected in query execution to correct inaccurate histogram information of stored data. Another important task for database administrators is to create subsidiary data structures, such as indexes and materialized views, for speeding up data manipulation. For example, DB2 has DB2 Advisor [111] to help administrators choosing the index set for a given query set. Agrawal et al. [4] have studied automated selection of materialized views and indexes for Microsoft SQL servers.

Performance models of system components can provide fast and accurate predictions of system performance under different configurations, making them an useful element in automated resource allocation. This thesis focuses on building performance models for storage devices because they are important in making storage resource management decisions.

2.2 Storage Device Modeling

Storage devices are difficult to model because of their non-linear, state-dependent behavior. Early research work in file systems either assumed a constant service time or selected times from a uniform distribution. Ruemmler and Wilkes [94] laid the basis for the disk modeling work by simulating the axial and rotational head positions, allowing the seek, rotation, and transfer times to be computed. Later work started to handle complex storage devices, such as cache-enabled disk arrays [24, 30, 62, 77, 119]. These models usually fall into two categories, simulators and analytic models, both requiring expert knowledge of storage devices. A new approach, black-box modeling, on the other hand, treats storage devices as black boxes, thus requiring no information about storage device internals.

Simulators. Simulators produce per-request response times by simulating the physical mechanism of storage devices in software. Pantheon [119], DiskSim [24], and RAIDframe [51] represent the current state of the art in disk storage array simulation. These simulators can simulate complex storage systems given accurate specifications for caches, buses, controllers, adapters, and disk drivers.

While such simulators are helpful in understanding storage device performance, especially when devices are not immediately available, developing a simulator is challenging. Model developers should have access to the internal implementation details, such as the interconnection between devices, the caching policy, and layout. Unfortunately, this information is not always available for public access. Tools have been designed to extract disk parameters by issuing controlled disk accesses to disks [95, 103]. Recent work from Denehy et al. [38] has applied a similar technique to disk arrays to extract the number of disks, RAID level, and stripe unit size.

Moreover, replaying traces on a disk simulator is expensive in both time and resources, making it inappropriate for online planning applications.

Analytic Models. Analytic models use a set of equations to describe disk behavior. Therefore, prediction is usually fast. Early analytic models aim at single disks. Simple models simulate the head position and use functions to approximate the seek and rotation time. These models are useful in performance optimization, e.g. in disk scheduling [54, 97, 123], data layout [93], and developing disk simulators [24, 119].

With the great success of disk arrays [29], analytic models started to handle disk arrays. Disk arrays introduce more complexity to the modeling, including replication, striping, prefetching, and caching. Kim and Tantawi [61] were among the first to present an analytic method for approximating the disk service time of requests striped across n disks. Merchant et al. [76] modeled individual disks in a disk array using M/G/1 queuing model and design the best striping strategies for the replicated data on the disk array. Chen et al. [30] incorporated both redundancy and queuing in their performance models of RAID 5 disk arrays in normal mode. Lee and Katz [66] used a closed queuing model to build the utilization, response time, and throughput models for disk arrays. Merchant and Yu [75, 77] then analyzed RAID 5 and RAID 1 disk arrays in both normal and recovery modes. A throughput model for RAID 0/1 and RAID 5 arrays [74] was used in the automatic storage provisioning tool, Minerva [7]. Thomasian and Menon [106, 107], and Kuratti and Sanders [62] further studied normal, degraded, and rebuild modes in their RAID 5 performance model. Bachmat and Schindler [12] analyzed reconstruction in RAID 1 disk arrays. Menon [73] modeled explicit read-ahead and write-back by the array cache of a RAID 5 disk array.

Work by Shriver et al. [99] took a modular approach to support algebraic descriptions of disk driver components and composed them to model the mean disk service times. Uysal et al. [110] used a similar approach to predict the mean throughput of disk arrays under asynchronous I/O workloads. Vartki et al. [112] analyzed the effects of caching policies and parallelism of disks along with array controller optimizations on the performance of a disk array using queuing theories.

Performance prediction through analytic models is fast in computation. Building analytic models, however, is even more difficult than developing simulators because model developers not only face the same challenges, but also should have the ability to distill I/O workloads into a handful of parameters and summarize the interaction between modeled devices and workloads into equations.

Black-box models. In contrast to simulators and analytic models, black-box models assume little knowledge on the internal implementation details of storage devices, resulting in a fully automatable model construction algorithm.

The simplest form of such models is device specifications provided by manufacturers. The specifications usually contain the maximum throughput and the average seek time of the devices. In reality, the achieved throughput can be lower than the maximum depending on workloads.

The table-based approach by Anderson et al. [9] built throughput models for disk arrays that take workload characteristics as model input. The models view the behavior of a storage device as a mapping from workloads to their performance. In the training period, a set of synthetic or real traces are replayed on the modeled device, and a table records the mapping for the training traces. Prediction is achieved by interpolating among the table entries of similar characteristics. More data can be added to the table during online operation to improve model accuracy. Our device models are an enhancement to the table-based models. Our goal is to explore the design space and evaluate the effectiveness of existing techniques, such as workload models and regression tools, in building such learning-based device models.

Later work by Kelly et al. [60] focused on online prediction for per-request response time using machine learning tools. Their model takes workload information and system status (i.e., the number of pending requests) as input and predicts whether the requests can be served quickly or not. Our models, on the other hand, do not use any system status as input.

2.3 Workload Characterization

Workload characterization has been an active area since the birth of computer systems and has involved various workloads on different components of the systems, such as memory references [15, 31, 58], accesses in file systems [13, 45, 78, 82, 87], I/O requests [40, 43], and network traffic [36, 68]. I/O workloads, in particular, show complex arrival and access patterns. These characteristics play an important role in determining the workload performance [40, 64]. Rome, a QoS specification language for storage systems [120], has provided a formal language for specifying workload characteristics.

Arrival process. Most recent work in workload characterization aims at the arrival process of network traffic after the groundbreaking discovery of self-similarity in Ethernet traffic [68]. Various computer-generated traffic have been shown to exhibit strong burstiness and self-similarity [36, 42, 68], including I/O workloads [43]. In these workloads, the burstiness observed in small time scales does not disappear after aggregation in large time scales, which invalidates the traditional assumption of independent arrivals used in queuing theory [33]. Instead, long range dependence between requests exist. Studies [1, 2, 46, 69, 80] have shown that self-similarity has a detrimental effect on network performance, leading to an increased packet loss rate, delay, and the need of a large buffer size and source traffic control. The Hurst exponent [68] has been proposed to quantify the degree of self-similarity in the arrival process. In addition, trace generation with fractal Brownian motion for a given Hurst exponent requires intensive computation. It is, however, not necessarily a good metric because self-similarity does not always lead to burstiness.

Several statistical self-similar processes, including fractal Brownian motion [68, 80, 85], fractional ARIMA [109], and Multifractal Wavelet Models [91], have been employed to generate realistic arrival processes for network traffic. These self-similar processes have solid mathematical bases and can be easy to analyze, but are computationally intensive to generate traces.

One conjecture on the cause of the self-similarity is the heavy-tailed distributions of users' active and idle time. This conjecture leads to structural ON/OFF models [14, 122], which aggregate multiple ON/OFF sources to produce web traffic. Gomez et al. [44] mixed the ON/OFF models and M/G/1 processes to model permanent and transience processes in I/O workloads. To generate a trace of similar characteristics to a given one, the ON/OFF models require a large number of parameters to be obtained from the original trace, such as the number of sources, the average ON and OFF period lengths for each source. In addition, the number of parameters depends on the number of sources in the workload, unsuitable for our storage device models because the models need compact workload representations as input for performance prediction.

Access patterns. Access patterns for I/O workloads have both the spatial and temporal locality as well as correlations involving request types and sizes.

I/O traffic is more difficult to model than network traffic because of the richness in access patterns. Network traffic assumes a categorical address space, so spatial locality usually concerns whether two requests have the same destination. Both the Independent Reference Model [22] and the LRU stack distance [6, 14] are used for generating the location information in web traffic. In contrast, the address space of I/O workloads is ordered, and the response time difference can be several orders of magnitude between sequential and random accesses. In practice, the locality of an I/O trace is usually represented by the empirical distributions of the access frequencies of the disk blocks and/or the sequential run length [44, 65, 109]. Seldom do these models consider the correlations between subsequent requests in LBN, request size, and operation type. Ganger et al. [40] experimented with real I/O workloads to show that no existing models were able to capture the complexity of I/O workloads at that time.

Our learning-based modeling approach uses regression tools to model storage device behavior as functions over I/O workloads and learn the functions by observing training traces on the devices. Because regression tools take only vectors as input, workloads will be represented as vectors of scalars internally. In addition, the model construction requires traces of rich access patterns. Existing work focuses mostly on extracting workload characteristics and generating traces similar to given traces. This thesis, on the other hand, emphasizes generating traces with diverse access patterns. In addition, it proposes the entropy plot and PQRS model to augment existing work in the workload characterization area. The former is an effective tool to quantify the spatio-temporal behavior of an I/O workload with three scalars. The latter can generate I/O traces with given characteristics.

Chapter 3

Toward Learning-Based Device Modeling

Our modeling approach distinguish itself from traditional modeling approaches by treating storage devices largely as black boxes. This chapter gives a brief overview of the design and the evaluation methodology of our device models.

Section 3.1 gives a high-level description of the approach and introduces the three main issues in implementing the approach. Section 3.2 presents the two methods for evaluating the effectiveness of various techniques. Section 3.3 concludes the chapter.

3.1 Challenges and Contributions

Our learning-based modeling approach learns the behavior of storage devices by observing them under a set of training traces. A performance model for a given storage device takes an I/O workload as input and produces as output its performance on the device. An I/O workload is specified as a sequence of disk requests, with each request, r_i, defined by four attributes

$$r_i = (\mathrm{Time}_i, \mathrm{LBN}_i, \mathrm{Size}_i, \mathrm{Type}_i),$$

specifying the arrival time, the device logical block number (LBN) of the first requested disk block, the request size in the number of disk blocks, and the request type (read or write).* The performance is an aggregate performance metric, such as the mean or percentile response time over one-minute intervals. A storage device can be a single disk or a disk array.

*We assume a device LBN is mapped to a single or multiple physical disk blocks on a storage device for simplicity.

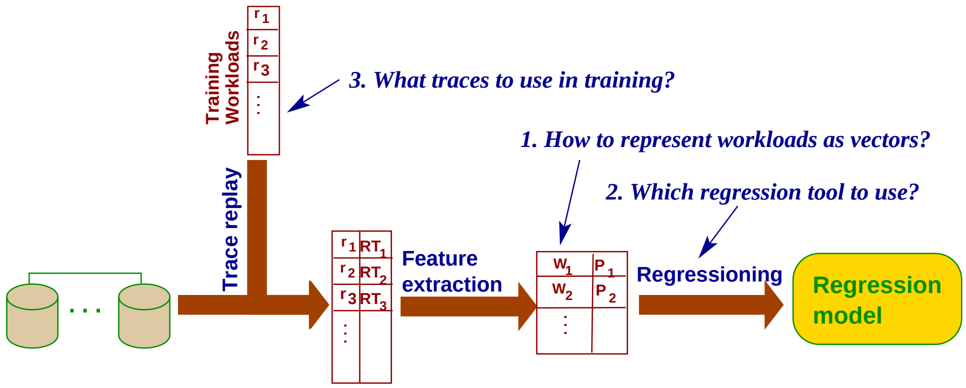


Figure 3.1: Model construction algorithm in the learning-based approach and three important issues in implementing the algorithm.

The black-box nature of the approach offers three advantages over traditional models, such as disk simulators and analytic models. First, by treating storage devices as black boxes, the model construction algorithm requires no prior knowledge of the device implementation details. Modeling a new device requires little human intervention. Second, by taking advantage of existing machine learning tools, it is possible for the device models to achieve high efficiency and accuracy because existing machine learning tools usually have strong mathematical roots and have experienced great successes in practice. Third, black-box models are attractive when privacy is a concern. Manufacturers can ship these models with their storage devices without revealing the technologies used in constructing these devices. All these features make black-box models attractive in practice.

This thesis explores the feasibility of using machine learning techniques to construct performance models for storage devices. During "training", the algorithm observes how the modeled device behaves under a set of training workloads and builds the models based upon these observations as shown in Figure 3.1. The system models device behavior as a mapping from workloads to their performance on the device. I/O workloads are internally represented as vectors and a regression tools models the target performance as a function over the vectors. Therefore, the device model consists of both the regression model built from training and the feature extraction algorithm. We identify three important issues in implementing this approach and evaluate the effectiveness of existing techniques in addressing them.

1. **Internal workload representation.** Regression tools model real-value functions over the multi-dimensional Cartesian space that input vectors live. Therefore, we need a feature extraction algorithms to convert workloads into vectors in such a space

before applying regression tools. It is difficult to achieve such vector representations due to the complex arrival and access patterns of I/O workloads.

2. **Selecting an appropriate regression tool.** There are a wide range of regression tools that offer different characteristics. We need to select an appropriate one to meet the efficiency and accuracy requirements.

3. **High-quality training traces.** The quality of training traces plays an important role in the resulting device models. The algorithm builds regression models by observing devices under a set of training traces. The model can only predict with high accuracy those workloads that are similar to the training traces. The training traces, therefore, should contain workloads of diverse characteristics for the device model to predict a real-world workload with high confidence. At the same time, the traces should be as short as possible to limit the training time.

The three issues are orthogonal, so we address them separately in three chapters. In each chapter, we explore the design space of the corresponding problem by always using the best solutions for the other two. The remainder of this section discusses the challenges and our main conclusions in detail. Table 3.1 lists the important symbols used in this thesis.

3.1.1 Vector Representation of Workloads

In our learning-based approach, an I/O workload is internally represented as vectors in a multi-dimensional Cartesian space, and the target performance metric is modeled as functions over the vectors. This step, therefore, is to determine the vector representation of I/O workloads, that is, the axes of the multi-dimensional space. It is impossible to model per-request response time RT_i as a function over r_i because the response time depends not only on the current request but also on those that arrived before the current one. It is, therefore, important to design a vector representation that can capture the correlations between the requests.

Converting traces to vectors can be viewed as "feature extraction", and the vectors are "feature vectors" that capture the important features of input data sets. The features are the axes of the multi-dimensional space and the important workload characteristics in determining device performance. I/O workloads exhibit strong correlations between requests, and show bursty arrivals and complex access patterns. Powerful tools are required to capture these important characteristics efficiently with vectors.

Vectors can work at different granularities on I/O workloads, as shown in Figure 3.2. A *request description* characterizes a single request, and the corresponding request-level

Symbol	Meaning
r_i	a disk request in a workload, which is specified by four attributes, arrival time $Time_i$, location LBN_i, size $Size_i$, and $Type_i$.
RT_i	the response time of request r_i on the modeled device
R_i	request description, describing the per-request workload characteristics. The request-level device model models RT_i as a function on r_i.
P_j	the aggregate performance measure for a workload fragment of a pre-defined length (in time).
W_j	workload-level description for a workload fragment. The workload-level device model models P_j as a function on W_j.
$\text{Timediff}_i(t)$	the difference in arrival time between r_{i-2^t} and r_i for $t = 0, 1, \ldots, t_{\max}$
$\text{Count}_i(w)$	the number of requests arrived in $[\text{Time}_i - 10^{w-1}, \text{Time}_i]$ for $w = 1, 2, \ldots, w_{max}$
$\text{LBNdiff}_i(l)$	the distance in LBN between r_{i-l} and r_i for $l = 1, 2, \ldots, l_{max}$
t_{max}	the maximum value for calculating Timediff.
l_{max}	the maximum value for calculating LBNdiff.
LRUstack_i	LRU stack distance of the current request.
Seq_i	flag indicating whether the current request is a sequential access
R_i^{timediff}	request description which uses the difference in arrival time from the current request to precedent requests to model the queuing time
R_i^{count}	request description which uses the number of history requests in a set of windows to model the queuing time

Table 3.1: Symbol table.

models predict per-request response times from request descriptions. Similarly, a *workload-level description* works on a sequence of requests, and workload-level models model the aggregate performance of sequences directly from their workload-level description.

Chapter 4 presents the design and evaluation of these vector representations. When training traces and testing traces offer similar characteristics, both models can make predictions with a median relative error ranging from 10% to 40% on real-world traces. This indicates the effectiveness of the vectors in capturing most important workload characteristics. The comparison between request- and workload-level models shows a clear trade-off between training and predicting time: the former are fast in training and slow in predicting, and the latter are just the opposite. Since workload-level models are more stable in prediction accuracy and fast in prediction, we chose the workload-level models for further investigation in our thesis.

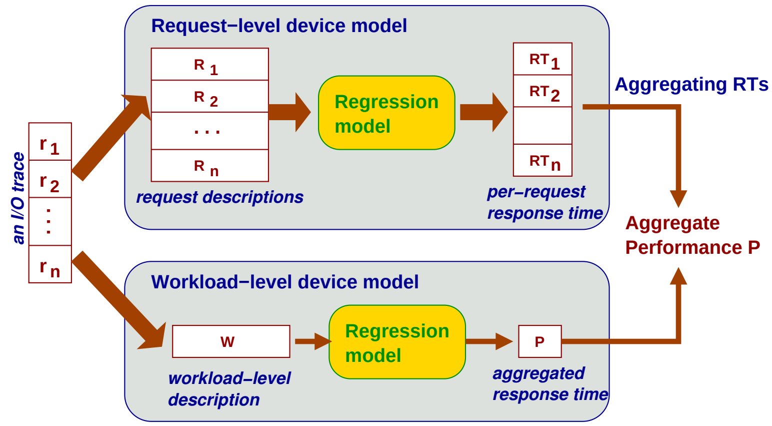


Figure 3.2: Modeling storage devices at the request and workload levels.

3.1.2 Selecting A Regression Tool

The learning-based modeling approach uses a regression tool to model device performance as a function over request or workload-level descriptions. To be useful in making resource allocation decisions, the chosen regression tool should offer accurate and fast predictions, low storage requirement, and an easy construction algorithm for device models. For example, the prediction accuracy should be high enough for resource planning. One study reported that an analytic model with an average relative error around 20% was accurate enough for their resource planning applications [7]. The prediction time should be in the order of milliseconds for evaluating the performance of an I/O trace of reasonable length. In addition, the ideal scenario is for the regression models to reside comfortably on storage devices or to be included in resource management software. So models under 1 MB should be tolerable. Easy model visualization and error analysis can be of additional value in understanding the models. There are a wide range of regression tools available, including linear regression, Classification And Regression Trees (CART), Artificial Neural Networks (ANN), Support Vector Machines (SVM), and k-Nearest Neighbors (k-NN). These tools offer different properties. We need to select one that fits our needs the best. Chapter 5 investigates these tools and justifies our choice of CART after empirical studies.

We have chosen to use CART models because they are accurate, compact, and easy to use. CART models use piece-wise linear functions to approximate a real-value function. Both the model construction and prediction are fast. In addition, model construction needs little human intervention, and the resulting models are compact, taking only several kilobytes to store.

3.1.3 Training Trace Generation

Training traces play an important role in the quality of the constructed models. On the one hand, the traces should be diverse enough to exercise storage devices during training so that the models can predict a workload with high confidence. On the other hand, training traces should be short to limit the training time.

Using real-world traces in training imposes three difficulties. First, real-world traces may not cover all the interesting workloads well. In addition, advanced sampling algorithms could require iteratively generating new traces with specific characteristics to augment training traces, and may not work with real-world traces because it can be difficult to alter the characteristics of existing traces into the desired ones. Second, these traces could incur large storage overhead. Finally, because of data privacy issues, these traces may not be publicly available. As a result, we chose to use synthetic traces in training.

Synthetic trace generation in our system has a subtle difference from previous work on the same area. Existing work has focused on generating traces of similar characteristics to a given trace, and these algorithms can choose appropriate parameters at their own will. Bootstrapping used in [104] can increase the confidence by testing on samples that are repeatedly drawn from the original sample set, but it can not produce samples that are missing in the original set. Our focus is to generate traces of diverse behavior with or without any real-world traces as samples.

Training trace generation involves two steps: a) produce a set of workload-level descriptions by sampling the workload-level description space, that is, the vector space which workload-level descriptions live in, and b) synthesize traces with the characteristics specified by the sample workload-level descriptions. Various sampling techniques exist, so the challenge lies in the second step. The workload-level descriptions are composed of scalars, and one needs to make additional assumptions to generate traces. Chapter 6 explores the design space of using synthetic traces in constructing the learning-based models. Our analysis has shown that the quality of our trace generators needs further improvement.

3.2 Evaluation Methodology

We use trace-based analysis and a resource planning application to evaluate our learning-based models. This section briefly describes the workloads and devices used in the evaluation.

3.2.1 Trace-Based Analysis

The trace-based analysis evaluates our device models using real-world traces. We compare the predicted performance against the measurements from simulations to quantify the effectiveness of the device models.

Traces. Table 3.2 lists the real-world traces we have used throughout this work.

The `cello99` traces are typical computer system research I/O workloads, collected at HP Labs in 1999 [44]. The system is a successor to the system described in [94]. We use the activities on the three most active devices (physical device number 0x1f068000, 0x1f034000, and 0x1f003000) among the 23 devices and label them as *cello99a*, *cello99b*, and *cello99c* respectively. The accesses to each trace fits in a 9 GB disk perfectly, so no trace editing is necessary. We report experimental results on a four-week snapshot (Feb. 1 to 28, 1999.) In `cello99b`, the two busiest hours (3 - 5 am) from each day are omitted because the simulated devices do not have sufficient resources to replay these traces.

The SAP traces were collected on an Oracle database server running SAP ISUCCS 2.5B in a power utility company [116]. The server has 3,000 users, and the disk accesses reflect the retrieval of customer invoices for updating and reviewing. The original trace is 15 minutes long. We use the activities on the four most active devices from the traces in our evaluation. The four traces have similar characteristics as the vector representations of the traces are clustered in the vector space, so we concatenate them to form an one-hour trace. The requests are mostly read-only in the traces.

The Financial traces and WebSearch traces are the two sets of traces published by Storage Performance Council (SPC) [34]. The Financial traces are from OLTP applications running at two large financial institutions. They are rescaled to double the interarrival time (i.e. 2$\times$ slowdown) for the traces to run on Atlas 10K disks. The WebSearch traces are from a popular search engine, and are read-only.

Unfortunately, we were unable to use other traces, such as the TPC-H and OpenMail traces [11, 63] because DiskSim does not provide models of storage devices of sufficient capacity. For example, the maximum LBN in OpenMail trace is 63 billion, and the trace will require a storage device of 32 TB to run.

Devices. We use our device modeling approach to model the following three devices. Table 3.3 lists the important configuration parameters of the devices. We obtain the response time for all the traces on these three devices using the state-of-the-art disk simulator, DiskSim 3.0 [24].

Name	Length	IO operations per second (IOPS)	reads	sequential accesses	mean request size
cello99a	1 month	29.42 (39.45)	20%	4%	7.12 KB (10.4)
cello99b	1 month	25.01 (52.25)	35%	13%	8.75 KB (4.26)
cello99c	1 month	33.05 (67.92)	25%	32%	8.25 KB (7.85)
SAP	1 hour	300.00 (81.98)	100%	27%	15.44 KB (15.86)
Financial1	1 day	61.49 (30.98)	23%	4%	3.52 KB (6.56)
Financial2	22 hours	45.45 (21.94)	82%	2%	2.31 KB (6.26)
WebSearch1	2 hours	166.45 (36.50)	100%	3%	15.04 KB (9.87)
WebSearch2	8.5 hours	147.73 (41.33)	100%	2.3%	15.00 KB (9.44)

Table 3.2: Summary statistics of I/O traces. The numbers in parentheses are the standard deviations.

- **Single disk:** a 9GB Atlas 10K SCSI disk model. The DiskSim 3.0 source code includes a calibrated model for the Atlas 10K disk.
- **RAID 1/0 disk array:** a RAID 1/0 disk array consisting of 16 Atlas 10K SCSI disks equipped with a 4GB cache.
- **RAID 5 disk array:** the same disk array as the RAID 1/0 except for the RAID level.

DiskSim provides a calibrated model of Atlas 10K, and Appendix A lists the DiskSim parameter files used in the simulating the two disk arrays. Table 3.4 summarizes the performance of the traces on these devices. We seeded the simulator differently for five runs of the traces on the modeled devices, and the response times stay the same across all the runs. The response time of a request is the time it spends in the devices. The `cello` traces have the response time information from the original `cello` system. Figure B.1 on page 139 compares the original response time distributions with the simulated ones.

Aggregate performance metrics. The device models are designed to predict the aggregate performance of a workload on storage devices. The aggregate performance can be the average or any percentile response time over a certain period of time. Generally, high percentile response times are more difficult to predict than low percentiles, and the medians are more stable than the averages. The variance of the aggregated performance decreases with the granularity of aggregation. A coarse-grained one may miss transient performance outrages. A fine granularity imposes more challenge in the vector representations because the queuing and caching effects from previous fragments become more significant on a fragment's performance. We selected to break the traces into one-minute-long fragments and

Parameter	Value/Units
size of the array cache	4 GB
cache replacement policy	LRU
cache line size	32 MB
flush policy	on demand: the dirty blocks are written back only when the allocation/replacement policy needs to reclaim them
read/write prefetching	disabled
write policy	Write-back: completion is reported immediately and dirty blocks are held in the cache for some time before being written back to disks
stripe unit size	32 KB
number of disks in disk array	16
disk type	Atlas 10K
disk capacity	9 GB
revolutions per minute	10,000
maximum seek time	10.83 ms

Table 3.3: Important characteristics of the three devices in our experiments.

built models to predict the aggregate performance over the fragments because we think one minute hits the right balance. Some requests of the traces suffer from large response times because the disks are too slow to handle the traces. As a result, the queuing time becomes a significant component of the response times, making accurate predictions more difficult to achieve.

Figure 3.3 shows how the variance in the aggregate performance changes with regard to the percentiles on the `cello99` fragments on Atlas 10K. As expected, the median value for the median response time increases with the percentile, and so does the standard deviation. Because of the increasing standard deviation, predicting high-percentile response times could be more difficult.

Figure 3.4 compares the cumulative response time distribution of the three `cello99` traces on the three devices. We observe that even though the three traces are from the same workload, they show different behavior on the devices. The response time distribution is highly skewed for the two busy traces, `cello99a` and `cello99b`. A large number of the requests have small response times, but a small portion of them have extremely large response time. `cello99c` is relatively smooth compared to the other two, and the maximum is usually less than 40 milliseconds. The high degree of skew leads to the high variance in the response time. For example, the standard deviation of the response times can be more than triple of the average for `cello99a`. The difficulty in performance prediction, however,

Traces	Atlas 10K	RAID 1/0	RAID 5
cello99a	96.19 (253.86)	11.75 (40.90)	30.53 (95.50)
cello99b	106.20 (309.48)	34.68 (89.50)	106.55 (251.53)
cello99c	3.74 (3.26)	3.05 (2.57)	177.67 (502.54)
SAP	-	12.67 (31.60)	19.07 (42.31)
Financial1	14.07 (17.33)	3.54 (5.71)	9.29 (23.25)
Financial2	5.07 (4.97)	2.62 (2.97)	3.27 (3.85)
WebSearch1	-	9.56 (4.47)	10.00 (5.51)
WebSearch2	-	9.27 (4.26)	9.66 (5.17)

Table 3.4: Average response time (in milliseconds) of the traces on modeled devices of one simulation run. The values in parentheses are the standard deviations of the response times.

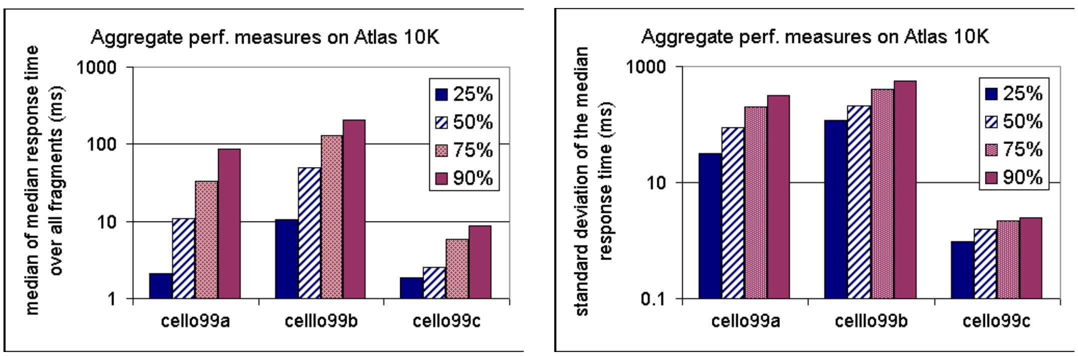


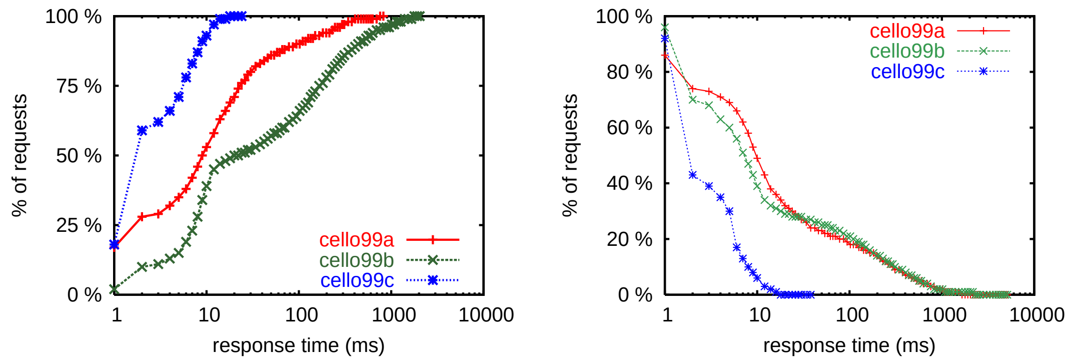
Figure 3.3: Median (left) and standard deviation (right) of the four response time percentiles over all the one-minute fragments on Atlas 10K. The graphs compare four response time percentiles: 25%, 50%, 75%, and 90%.

is not due to the large variance of the response times, but the complexity in determining and quantifying important workload characteristics so that the device models can capture device behavior as a function over these characteristics.

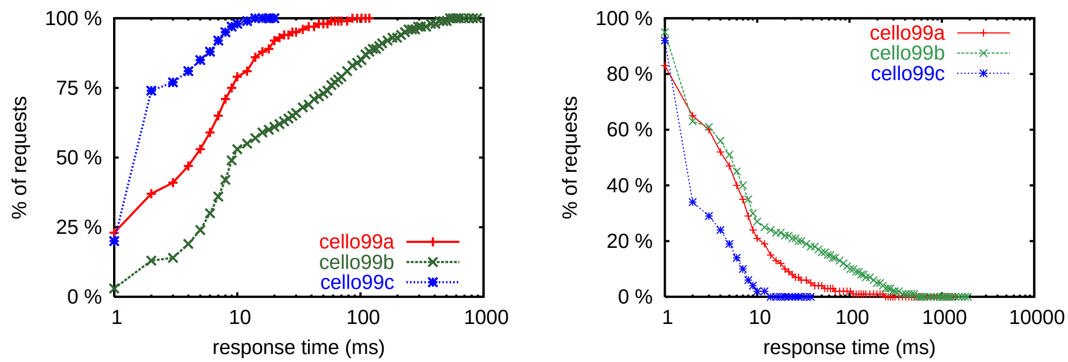
Error Metrics. We use the relative and absolute error to quantify model accuracy in both the per-request response time and aggregate performance prediction.

$$\begin{cases} \text{Absolute error} = |\text{Prediction} - \text{Actual value}|. \\ \text{Relative error} = \frac{\text{Absolute error}}{\text{Actual value}}. \end{cases} \tag{3.1}$$

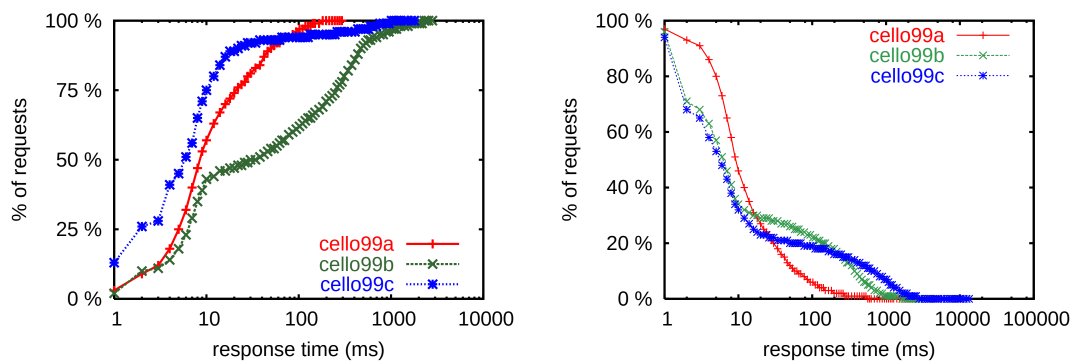
The actual performance is obtained from simulation on DiskSim 3.0. When we construct a model to predict the logarithm of the target performance, we convert the predictions back



(a) Atlas 10K



(b) RAID 1/0



(c) Raid 5

Figure 3.4: Response time distribution of the three `cello99` *traces on the three devices. (cumulative distribution functions on the left and negative cumulative ones on the right). The probability distribution functions are available in Figure B.2 on page 140.*

to the original scale before calculating the error. We report the median of the relative and absolute errors throughout this thesis because medians are more stable to extreme values than averages. Note that this value is different from the average relative error reported by previous studies [99, 74], which use the ratio between the average absolute error and the average simulated values as the average relative error. To make a fair comparison, we also report the latter when appropriate.

Another popular error metric is demerit [94], defined as the root mean square of the horizontal difference between the simulated and predicted cumulative response time distributions for evaluating storage device models. Request-level device models predict per-request response times, and workload-level models directly predict aggregate performance. Therefore, we only report the demerit numbers for request-level models.

3.2.2 Case Study

The ultimate use of our device models is for evaluating candidate configurations in resource allocation. In the case study, we use a sample application to demonstrate the effectiveness of our device models for such applications.

We use the device models to compare candidate physical database designs for a TPC-H workload. For large databases, the number of possible designs can be overwhelming, and it is difficult for administrators to find a good physical database design even with thorough knowledge of the workloads and the performance characteristics of the storage devices in use. Accurate performance models for storage devices are useful in evaluating a large number of candidate designs efficiently, making it possible to automate the design process.

Chapter 8 presents the design of the case study. The experiments have shown that even with inaccurate predictions from our models, the relative ranking matches the actual ranking of the candidate designs, suggesting the usefulness of the models in resource planning.

3.3 Summary

Our learning-based modeling approach aims to build efficient and accurate performance models for storage devices by observing the devices under a set of synthetic traces. I/O workloads are internally represented as vectors in a multi-dimensional Cartesian space, and a regression tool models the target performance as functions over the vectors. The models can be both efficient and accurate if we can capture important characteristics of I/O workloads with the vectors, and if the training traces are of high quality. This thesis investigates the effectiveness of existing techniques in addressing these issues. The next three chapters present the results on internal workload representation, regression tool selection, and training trace generation.

Chapter 4

Internal Workload Representation

The learning-based device models use vectors to represent an I/O workload and employ a regression tool to approximate device behavior as a function over the vectors. The vectors should efficiently capture the important workload characteristics so that the regression tool can distinguish workloads of different performance. Identifying and quantifying important workload characteristics can be useful in building other types of storage device models, too. For example, analytic models can include these characteristics in their input. The vector representation, however, is difficult to achieve because of the complex correlation structures in I/O workloads.

This chapter presents the design and evaluation of two types of vector representations. As we have mentioned in Chapter 3, there are total of three issues we address in this thesis, and we use the best solution for the other two when evaluating the design choices for one of them. That is, we always use CART models and real-world traces to construct device models in this chapter to study the effectiveness of different vector representations.

This chapter starts with a brief overview of the challenges in designing the vector representation and presents two types of vectors: *request descriptions* in Section 4.2 and *workload-level descriptions* in Section 4.3. The former describes the characteristics of single requests, and is used in per-request response time prediction. The latter describes the characteristics of sequences of disk requests and is used for predicting the aggregate performance measures. Both descriptions convert the complex correlations of I/O workloads into vectors of scalars. Section 4.4 compares the two vector presentations.

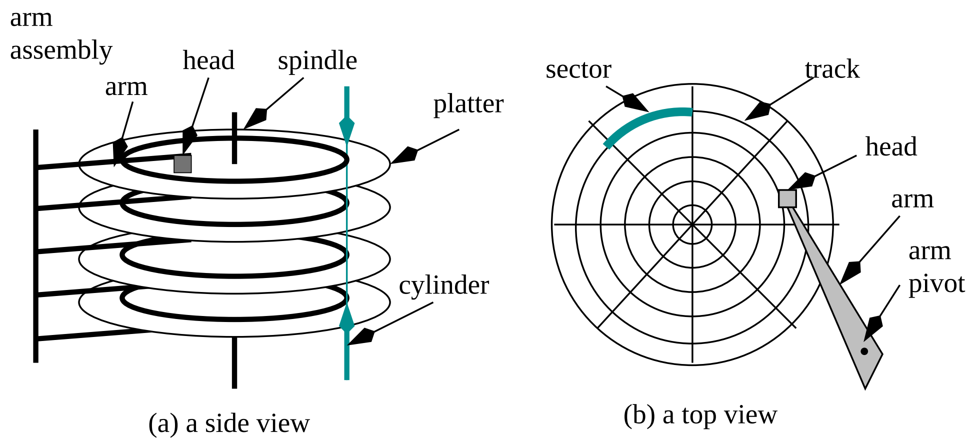


Figure 4.1: The mechanical components of a disk driver. The graphics are derived from [94].

4.1 Challenges

An I/O workload is internally represented as vectors in a multi-dimensional Cartesian space, and regression tools rely on the vectors to predict workload performance. The vectors should, therefore, capture important workload characteristics for storage device models to distinguish workloads of different performance. In addition, the vectors should be of a low dimensionality to keep the model compact and the training process fast.

The complexity in storage devices and I/O workloads both contribute to the workloads' performance on the devices. The vector representations should capture all of them.

4.1.1 Complexity in Storage Devices

Device performance is a result of the interaction between I/O workloads and storage devices. Different workload characteristics could be important on different storage devices. For example, the important set of workload characteristics is different on single disks and disk arrays, and on devices with or without caches.

Single disks without cache. The simplest form of storage devices are single disks. A disk consists of two major components: a mechanism and a controller [94]. In the mechanism, the recording media consists of a set of round platters and a set of heads for accessing the data recorded on the platters, as shown in Figure 4.1. The stack of the platters rotates in lockstep on a central spindle. Data on the platters are stored on concentric circles, known as "tracks". Each track is divided into sectors, that is, disk blocks, which are the smallest unit for data transfer between the mechanism and the controller. The heads float on top of the platters and perform the read and write operation on the media. The heads are attached to the arm assembly, which can position the heads to a desired track on the platters. The rotational speed determines the speed of the data transfer from the media to the disk controller. The disk controller performs the mapping from incoming logical addresses to the physical location of the data on the platters and manages the retrieval and storage of data.

The above physical organization of disks leads to strong non-linear performance. The service time of a request consists of four parts: the queuing time spent in the buffer of the controller, the seek time it takes for the heads to moved to the desired track, the rotational latency, and the data transfer time for the actual read or write. As a result, the correlations between a request and previous requests are important in determining the response time of the current request depending on how closely these previous requests are to the current request in arrival time and location.

Various techniques exist for performance optimization on single disks. For example, reordering requests [97] and changing disk block layout [5, 26] can reduce total seek time. Freeblock scheduling [70] uses the rotational latency delays of foreground requests to serve background requests. All these optimizations further complicate the workload characterization problem because workload characteristics should include new ones that affect workload performance on these devices.

Disk arrays. Single disks can be combined to form a disk array, allowing opportunities to deliver high performance and reliability [84, 29]. The main two parameters of disk arrays are the size of contiguous data stored on single disks in striping, known as the striping unit, and the redundancy scheme. The RAID level indicates the combination of the redundancy and the parity schema. For example, RAID 1/0 arrays use striping and mirroring, and RAID 5 arrays use striping and rotated parity.

Striping can greatly improve the throughput of I/O systems. With striping, the data is distributed across several disks in a round-robin fashion so that multiple independent requests can be served by separate disks in parallel. In this case, the seek time and rotational latency depend on previous requests that access the same disk, and these requests are not necessarily the ones that arrive just before the current request. In addition, requests that access more blocks than the striping unit can be served by multiple disks in coordination to take full advantage of the bandwidth. Therefore, the operation time will be a function of the striping unit and the request size besides other factors.

Redundancy can be implemented using either mirroring or parity. In a disk array with mirroring, the disks are divided into two or more sets, and each set maintains a copy of the data. A read request will typically be served by the copy that offers a shorter operation time. A write request will need to update all the copies. The parity scheme maintains an error correction code to deal with disk failures. The parity data can be on dedicated disks or uniformly distributed across all the disks, known as rotated parity. The request size is an important factor in determining the write time in this case. Large writes touch blocks on all disks and can compute the parity easily. Small writes require four separate I/Os, two reads to read the old data blocks and parity and two writes to write the new data and parity.

Optimization techniques. Buffering and caching are two common optimizations in I/O systems [29]. Write buffering, or asynchronous writes, allows returning control to applications before actual writes on media are completed. This feature is especially helpful in reducing the response time for small write requests on RAID 5 arrays.

Read caching can improve the response time and system throughput by storing recently-accessed disk blocks so that future reads of the same data can be served quickly. The common cache replacement policy is the least-recent-used algorithm (LRU), and the LRU stack distance, which measures the number of distinctive disk blocks between two subsequent accesses to same disk blocks, becomes a useful metric in quantifying the temporal locality. In addition, caching makes prefetching possible. The prefetched data can wait in the cache for future accesses.

Different write policies can be enforced on caches to determines whether a write needs to wait for the operation to complete. For a write-back cache, a write request is deemed to be completed by applications as soon as the data is cached. The data will be written to the media later by the devices. The write-through cache, on the other hand, needs to wait until the data is written on the magnetic media. In these devices, the request type makes a difference in device performance. Modern disk arrays usually use the write-back policy.

The different implementation of storage devices leads to the difference in device performance. The vector representation of I/O workloads should capture sufficient important workload characteristics that might affect performance.

4.1.2 Complexity in I/O Workloads

I/O workloads show complex correlation structures. The correlations exist between current and previous requests on all the four attributes. The performance of an I/O workload could depend on all these correlations.

Bursty arrivals. The arrival process exhibits strong burstiness and dependence between requests [43, 115]. The long range dependence leads to a longer tail in the queue length then the traditional Poisson or Markovian traffic [80]. It is, therefore, important for the vector representations of I/O workloads to capture the temporal burstiness and the long-range dependence. Previous work [68] has employed the Hurst exponent to describe the degree of self-similarity in addition to the average arrival rate, but self-similar processes, such as fractal Brownian motion, may not be bursty.

Access patterns. Access patterns refer to the correlations between requests in location, request size, and type. A typical I/O workload usually contains both sequential and random accesses. Sequential runs can be interleaved, incomplete, noisy, or of different strides, making them difficult to detect [59]. It is important to detect the sequential accesses because storage devices handle sequential accesses more efficiently than random accesses. Random accesses could show various degrees of temporal and spatial locality. The former refers to the tendency that accesses to the same objects happen closely in time. The latter means that nearby objects are more likely to be accessed closely in time than objects that are far away in space. Storage devices depend on strong locality in I/O workloads to perform well.

Access patterns are difficult to characterize because the characterization process needs tools to quantify correlations on high-dimensional data sets. For example, quantifying the correlation between reads and writes requires the ability to quantify correlations between multi-dimensional data sets because both reads and writes are request streams of three attributes: arrival time, location, and size. In network traffic, the addresses of target objects are usually categorical, so there is no notion of "distance" between two objects. The direct application of network traffic models is inadequate because of the difference in the locality model. Some work has used the percentage of sequential accesses and the average run size to characterize the sequentiality of an I/O workload and the access frequency distribution of disk blocks for random accesses [44, 40].

Figure 4.2 gives visual evidence of the complex correlations in arrival time and location using a real I/O workload, one hour worth of activities on `cello99a`. The top-left plot shows the projection of the trace on the time-LBN plane. The requests are clustered, indicating strong correlations between the arrival time and location of the requests. On the bottom are the number of requests that arrived in each time interval, the interval being a second and a millisecond respectively. The burstiness in the arrival process is significant at both time scales. The two plots on the right show the access frequencies of disk blocks aggregated by 1,024 and 16 disk blocks respectively. Strong skew exists in both plots.

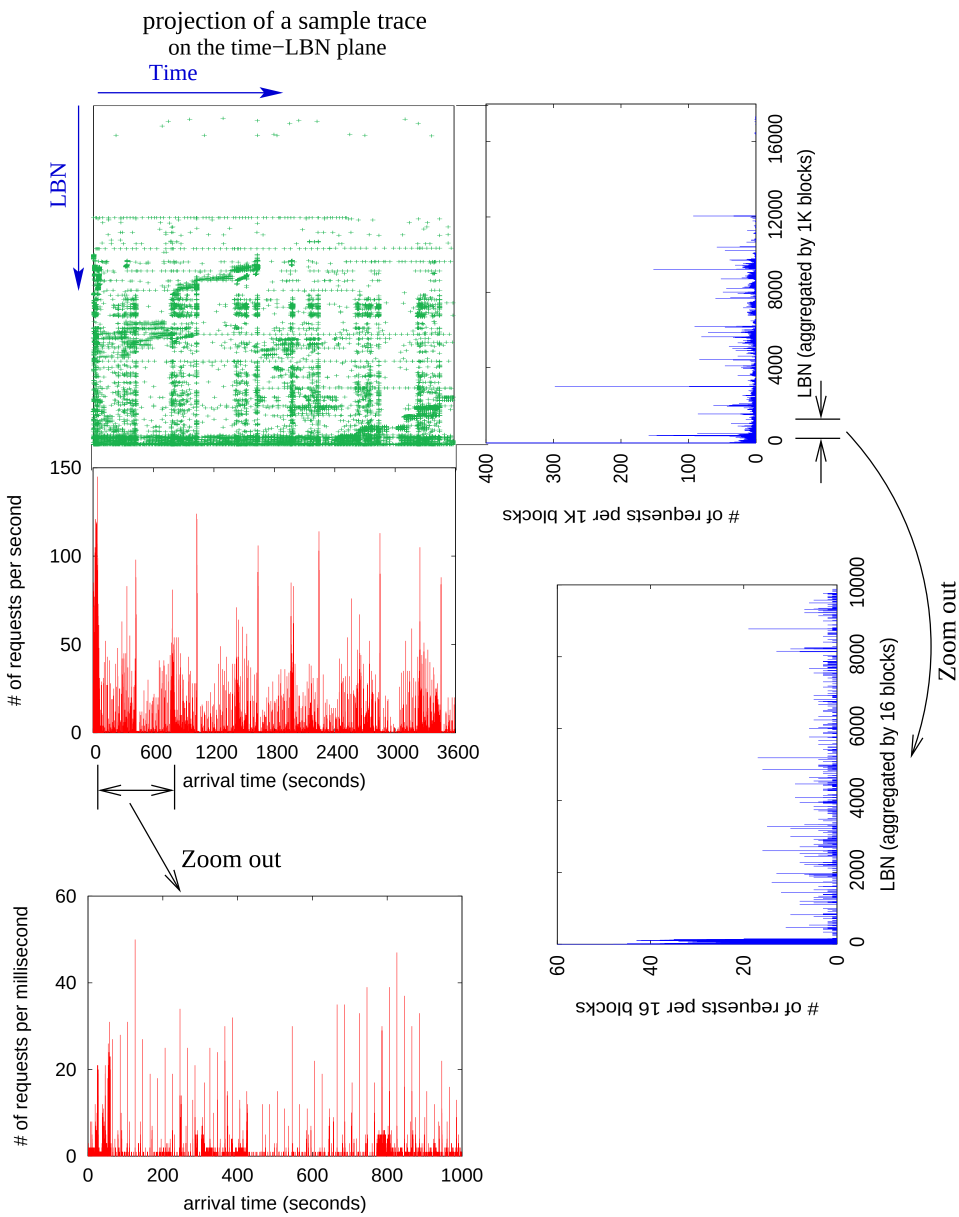


Figure 4.2: Visual presentation of a sample disk trace (the hour starting from 12:00am on Feb. 1, 1999 of `cello99a`*). The clustered requests in the top-left plot indicate strong temporal-spatial locality in the sample trace. The two plots on the right show the access frequencies on LBN at different aggregation levels, and strong skew exists. The two plots on the bottom left show the number of requests arrived in each time interval at two scales, and strong burstiness is observed.*

The vector representation should capture all these characteristics in order to build accurate performance models for storage devices.

4.1.3 Identifying Important Workload Characteristics

Identifying important workload characteristics is an essential part of workload characterization. Early work in queuing theory assumed Poisson arrivals and an exponential service time distribution to calculate the throughput of closed systems [33]. Recent work has explored more realistic arrivals and service time distributions [66, 76, 79, 80]. Some analytic models of storage devices take workload characteristics as input and study device behavior as functions over these characteristics [99, 110]. They, however, do not address how to extract workload characteristics and measure their effects on device performance.

Kurmas et al. [65] designed the Distiller to automatically identify the set of important workload characteristics for a given workload, and the tool is used in Rubicon for workload characterization and trace generation [113]. The idea is to study the importance of workload characteristics by comparing the performance of traces before and after one attribute is replaced with synthetic values. The tool can efficiently search among a given characteristic set to find a subset that characterizes the given trace the best.

Hong et al. [49] studied the important range of the dependence between requests in I/O workloads by time-slicing the traces and shuffling the slices. If the shuffled traces offer similar performance as the original one, the dependence beyond the slice length is not important in determining device behavior. Their work identified that the difference was negligible when the slices were 1 minute long or longer.

Our vector representations attempt to take advantage of the findings in these studies. However, due to the difference in the problem setting, we are unable to use all of these parameters. For example, the Distiller included the distribution of the request location to characterize real-world traces under study. Our work can not use such characteristics because vector representations take only scalar parameters. Instead, we design a new tool, entropy plot, to quantify the skew in the access frequencies of disk blocks as well as the bursty arrivals and locality of I/O workloads.

We have explored two types of vector representations. A request description, R_i, characterizes a individual request, r_i, and is used to predict its response time. A workload-level description W_j captures the characteristics of a sequence of requests, and a regression model can predict the aggregate performance P_j directly from W_j. Figure 4.3 illustrates the two types of learning-based device models based on these two representations.

Our current design uses a fixed workload description across all the workloads and devices. By taking advantage of the above work, a more advanced system could automatically

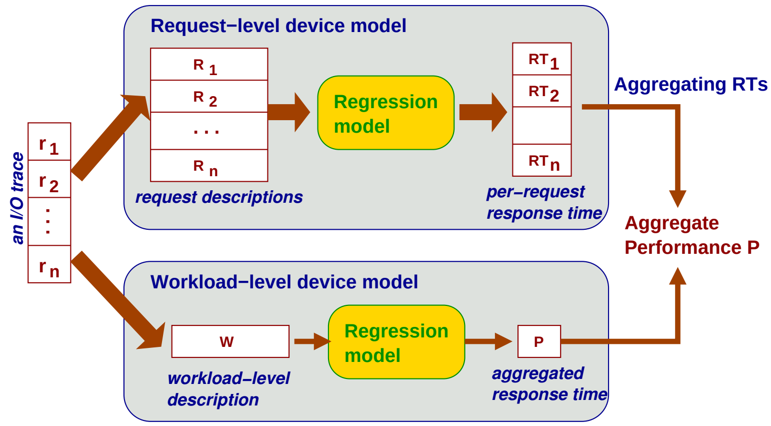


Figure 4.3: Request and workload-level device models.

identify important workload characteristics to be included in workload-level descriptions. We leave this to future work. The remainder of this chapter presents the design and evaluation of the two vector representations.

4.2 Request Description

A request description, R_i, captures the characteristics that determine the response time of request r_i. A request-level model, then, models the response time RT_i as a function on R_i. Even though r_i is a vector of four dimensions, using r_i as R_i, unfortunately, does not render accurate device models because RT_i relies not only on the current request, but also those requests that arrive before the current one. For example, if a request arrives at the system at an idle period, it can be served immediately by the device. If the same request arrives right in the middle of a burst of requests, the request will have to wait in the device queue for a long time before getting served, and the actual response time depends highly on the queue length. The request description, therefore, should capture the important dependencies between previous requests and the current one.

4.2.1 Composition of Disk Service Time

It is helpful to understand how storage devices handle requests before one designs request descriptions. For bursty traffic, the queuing time could be a significant component of the response time. The queuing time refers to the waiting time before a request gets served.

The actual operation time of a read request depends on whether a request can be served by cache. The answer is yes if the device is equipped with a cache and the requested data is available in there. A write request, in contrast, returns control to the caller as soon as the content is buffered if the device is configured to use the write-back policy and has space in its cache. Otherwise, the request has to wait for the completion of the actual write on the media.

If the request can not be fulfilled by the cache, the disk will need to access the media. In this case, the rest of the service time typically consists of three parts: the seek time, rotational latency, and data transfer time. A sequential access can be an exception here. For single disks, a sequential access starts from the end of its previous request, therefore, experiencing no seeking and rotational latency. On disk arrays, two requests can be turned into sequential accesses if the stride between them allow them to be mapped to the same disk to form a sequential run.

The composition of response times are slightly different in disk arrays. The queuing time still exists, and multiple levels of cache exist. The operation time strongly depends on the request size and type as we have discussed in Section 4.1.1.

4.2.2 Request Description

The request descriptions consist of four sets of parameters, each characterizing one particular response time component.

- **Modeling the queuing time.** The volume of requests arrived just before the current request plays an important role in the queuing time of a request. If a large number of requests do so, the chance for the current request to have a long queuing time increases.

  We can capture the queue length in two ways. The first option is to calculate the difference in arrival time between previous requests and the current one. The more closely these requests in arrivals, the more likely the current request suffers from a long queuing time. We define

  $$\text{TimeDiff}_i(t) = \text{Time}_i - \text{Time}_{i-2^t}, for\ t = 0, 1, \ldots, t_{max}. \tag{4.1}$$

  The subscript describes the sequence number of the requests. Therefore, $\text{TimeDiff}_i(t)$ is the difference in arrival time between the i-th request and the one that arrives 2^t requests before the current one. The value t_{max} determines the maximum number of requests used in calculating the request description.

  An alternative is to count the number of requests arrived before the current request in a set of fixed-sized windows. Similarly, the more requests in the windows, the more likely the current request has to wait for a long time before being served.

$$\text{Count}_i(w) = N(\text{Time}_i - 10^w, \text{Time}_i), for\ w = 0, 1, \ldots, w_{max}. \tag{4.2}$$

with 10^w specifying the window size in milliseconds and N being the number of requests arrived within the given time interval.

In our experiments, we use a value of 10 for t_{max} (corresponding to a maximum of 1,024 requests) and 5 for w_{max} (maximum window size of 10^5 milliseconds). We evaluated the effectiveness of using either parameter set in constructing the request-level device models in our experiments and concluded that the count-based approach is more effectively than the timediff-based one even though it uses fewer parameters. The fixed-sized windows allow the request descriptions to account for more requests for busy workloads, in which the queuing time dominates the response time.

- **Served by cache or not.** If a request asks for disk blocks that have been accessed recently, it is more likely to find the disk blocks in the cache. We use the LRU stack distance of the current request, $LRUstack_i$, to characterize this feature. The LRU stack distance measures the number of distinct requests between the last access to the same disk blocks and the current request.

- **Seek time and rotational latency.** The sum of the two depends on the original head position and the location of the current request. We use the seek distance LBNDiff(l) in addition to the address of the current request to quantify this part of the disk service time.

$$\text{LBNdiff}_i(l) = \text{LBN}_i - \text{LBN}_{i-l}, l = 1, 2, \ldots, l_{max}, \tag{4.3}$$

  To account for the request reordering, we look back more than one request for the seek distance. l_{max} is the maximum number of requests we use in the calculation. We set l_{max} to 3 in our experiment. The experiments show that the seek distance is insignificant in predicting per-request response times, especially in busy workloads. Increasing the value of l_{max} will not bring enough benefit.

- **Data transfer time.** The transfer time is usually proportional to the request size. We use Type_i and Size_i to predict the transfer time.

- **Sequential accesses.** A flag, Seq_i, indicates whether the current request is a sequential access. To account for both the sequential and strided accesses, we classify two contiguous requests as sequential accesses if the two requests have the same seek distance. For example, if three contiguous requests all have a seek distance of 16 blocks, we classify all three requests as sequential.

In modeling per-request response times on disk arrays, the queuing time, sequentiality, request size, and operation type are all important characteristics. These parameters should, therefore, be applicable to modeling the performance of disk arrays, too. This suggests that black-box models may still need some general knowledge on how these storage devices handle devices, but do not require specific knowledge on individual device parameters.

In summary, we provide two types of vectors for capturing the per-request characteristics depending on which approach used in modeling the queuing time. A timediff-based request description R^{timediff} uses the difference in arrival time for fixed numbers of requests to model the queuing time. A count-based request description R^{count} maintains the number of requests for windows of fixed sizes. The two request descriptions contain the following parameters:

$$\begin{aligned}
R_i^{\text{timediff}} &= \{\text{Timediff}_i(1), \ldots, \text{Timediff}_i(10), \\
&\quad \text{LBN}_i, \text{LBNdiff}_i(1), \ldots, \text{LBNdiff}_i(3), \\
&\quad \text{Type}_i, \text{Size}_i, \text{LRUstack}_i, \text{Seq}_i\}; \\
R_i^{\text{count}} &= \{\text{Count}_i(1), \ldots, \text{Count}_i(6), \\
&\quad \text{LBN}_i, \text{LBNdiff}_i(1), \ldots \text{LBNdiff}_i(3), \\
&\quad \text{Type}_i, \text{Size}_i, \text{LRUstack}_i, \text{Seq}_i\};
\end{aligned} \tag{4.4}$$

4.2.3 Evaluation

We investigate the effectiveness of the proposed request descriptions by an empirical study on real-world traces. We use the logarithm transformation on the response time before constructing the regression models to improve model accuracy. CART is used in the regression. (Please refer to Section 5.3.2 and 5.2.4 for a detailed discussion on the logarithm transformation and CART models.) We use the activities of Feb. 1, 1999 from the three traces in training and Feb. 2, 1999 in testing. The training trace is a mixture of the request descriptions drawn from three traces in a round-robin fashion. For the testing traces, we first calculate the request descriptions for all the requests and use a uniform sampling rate of 0.1% to reduce the data volume.

Comparing the two types of request descriptions. The graphs on the left column in Figure 4.4 to 4.6 compare the effectiveness of two types of request descriptions on the three `cello99` traces. We observe that count-based request descriptions consistently outperform timediff-based ones. The advantage is drastic especially for busy traffic such as `cello99a`

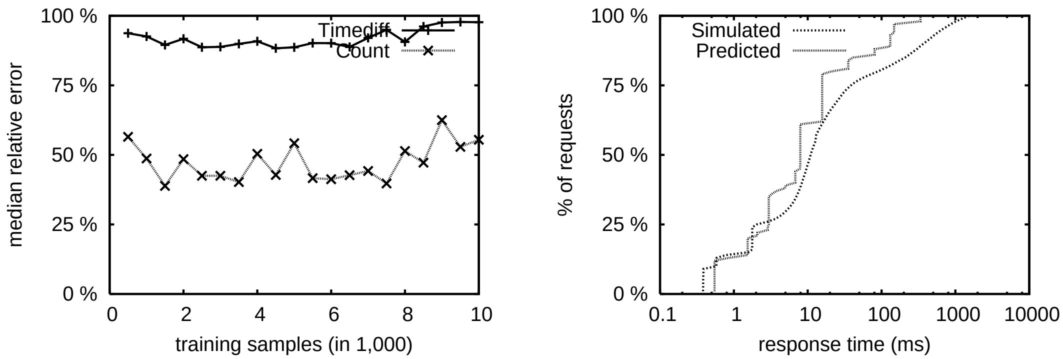
Trace	Atlas 10K	RAID 1/0	RAID 5
cello99a	327 ms (217%)	67 ms (302%)	136 ms (268%)
cello99b	293 ms (158%)	189 ms (159%)	485 ms (137%)
cello99c	45 ms (1122%)	10 ms (361%)	523 ms (196%)

Table 4.1: Demerits (absolute values and percentages over average response times) of the counted-based request-level models.

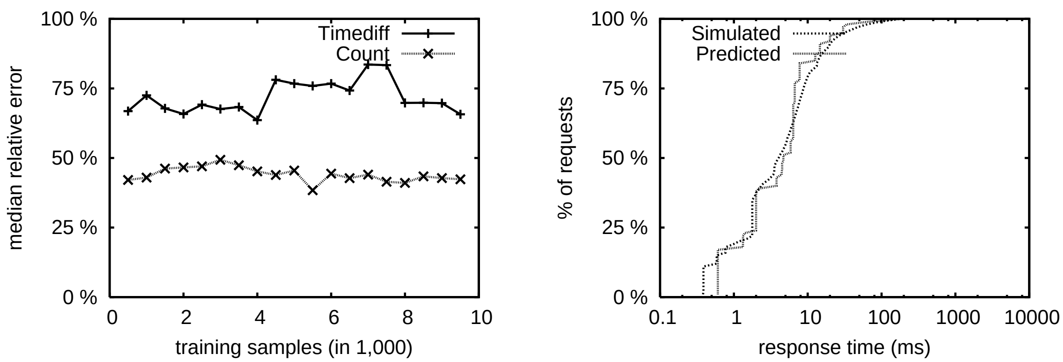
and `cello99b`. The major reason is that the timediff-based ones can investigate no more than 1,024 requests, which is much less than 5,853 requests, the median value of Count(5) for the mixed training trace. The performance difference between the two types of request descriptions tends to decrease from the single disk to the disk arrays because the queuing time is a smaller portion of the response time on disk arrays than single disks for the same workload. In summary, the count-based request descriptions are effective in characterizing the queuing time, leading to the superior performance for busy traffic. For the remainder of this thesis, we always use count-based request descriptions to construct request-level models.

The graphs on the right column compare the response time distributions predicted by R^{count} against the simulated ones. The predicted distributions matches the simulated ones better than one may expect from a median relative error of 50% in the per-request response time prediction. Aggregation in generating the distributions helps to reduce the prediction error by canceling out the over- and under-prediction errors. Table 4.1 lists the demerits on these workload and device combinations. The demerit figures, both the absolute and relative values, are high partially because of the logarithm transformation. The demerit metric is the root mean square of the horizontal distance between two cumulative distributions, and therefore penalizes mispredictions on requests with large response times. The logarithm transformation makes regression models favor requests with small response times, leading to a good match at the head part of the distributions.

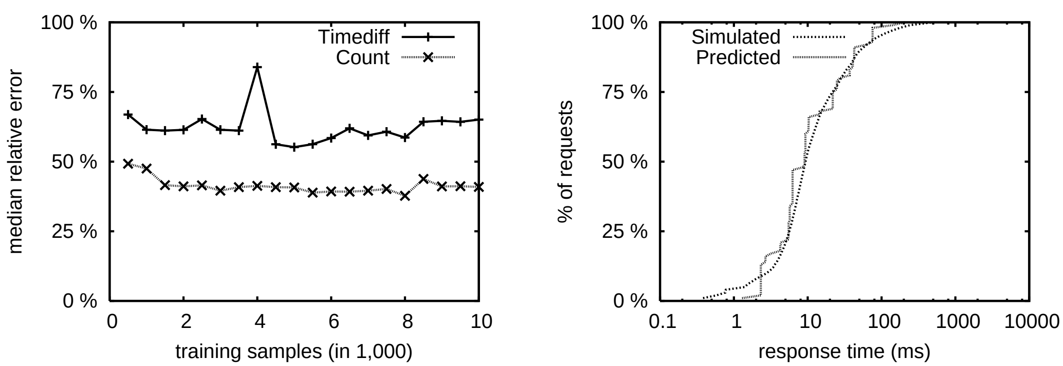
Figure 4.7 compares the median and standard deviation of the predicted and simulated response times. The predicted medians match the simulated ones much better than the standard deviations do. The reduced standard deviations for predictions are due to two reasons. First, CART models use the average value of the training requests that have similar characteristics to the predicted request as the prediction and treat the variance of the training requests as the intrinsic randomness of device behavior. Such variance includes the difference in device initial states and unobservable parameters. The averaging effectively removes the variance in the predicted values. Second, other errors come from the imperfect request descriptions. Some important characteristics are not captured by the descriptions.



(a) On Atlas 10K

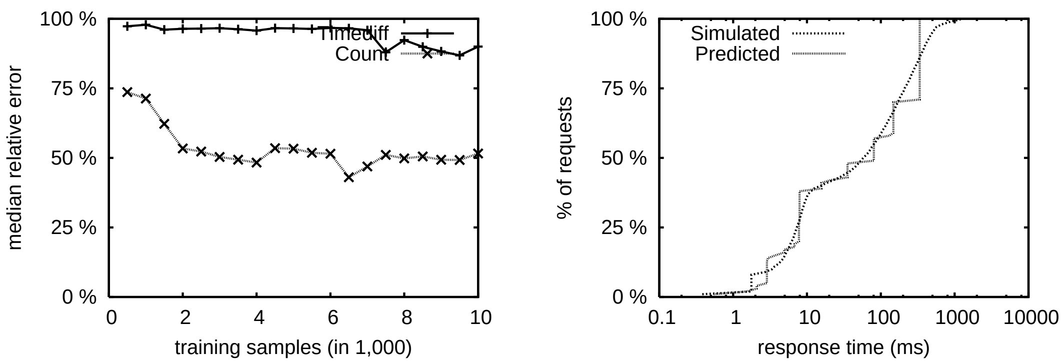


(b) On RAID 1/0

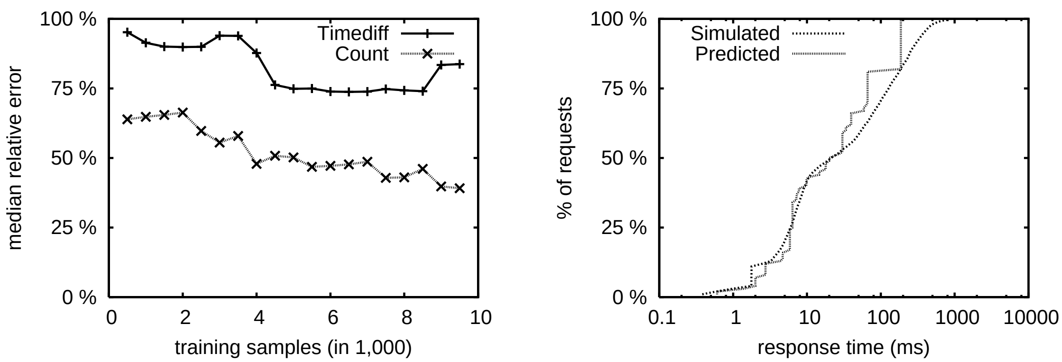


(c) On RAID 5

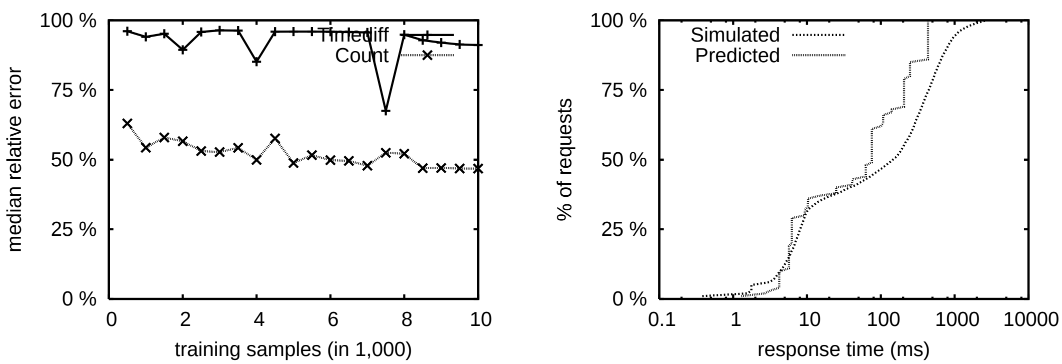
Figure 4.4: Comparison of two types of request descriptions in predicting the per-request response time of `cello99a`.



(a) On Atlas 10K

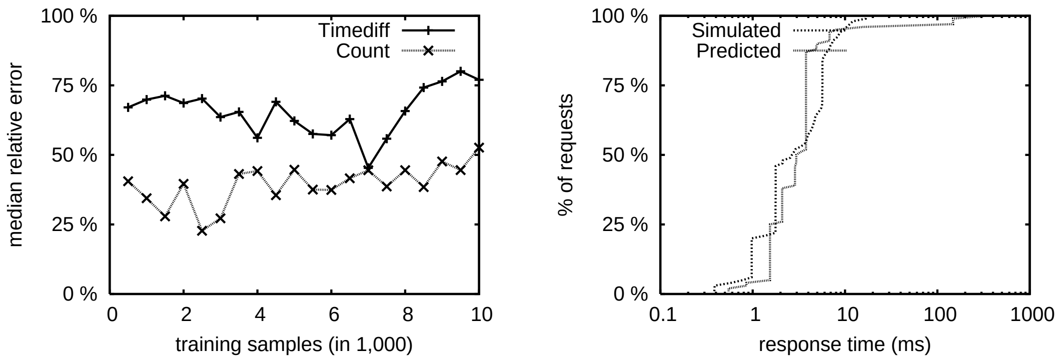


(b) On RAID 1/0

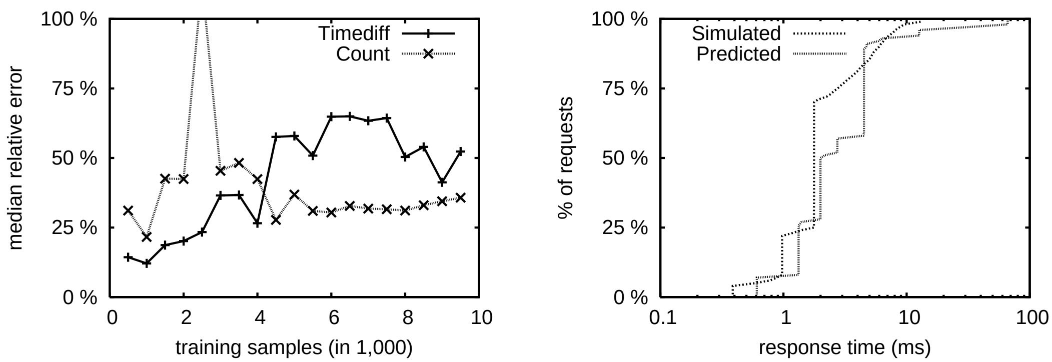


(c) On RAID 5

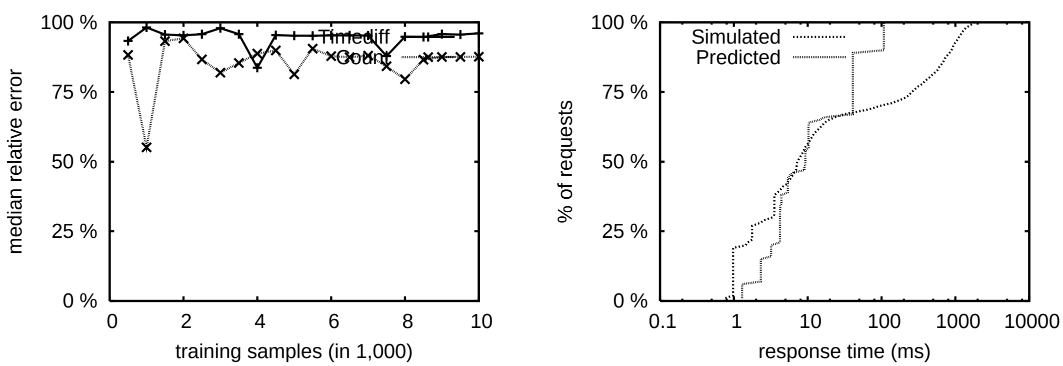
Figure 4.5: Comparison of two types of request descriptions in predicting the per-request response time of `cello99b`.



(a) On Atlas 10K

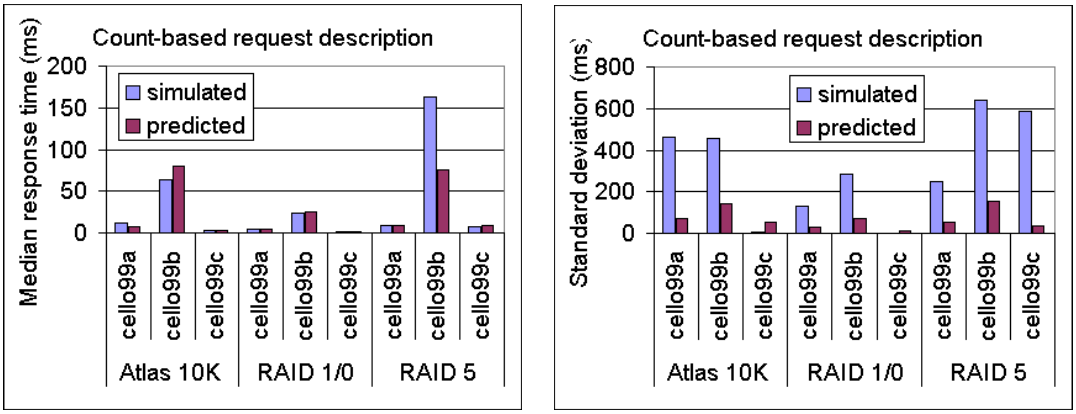


(b) On RAID 1/0



(c) On RAID 5

Figure 4.6: Comparison of two types of request descriptions in predicting the per-request response time of `cello99c`*.*



(a) Median response time (b) Standard deviation

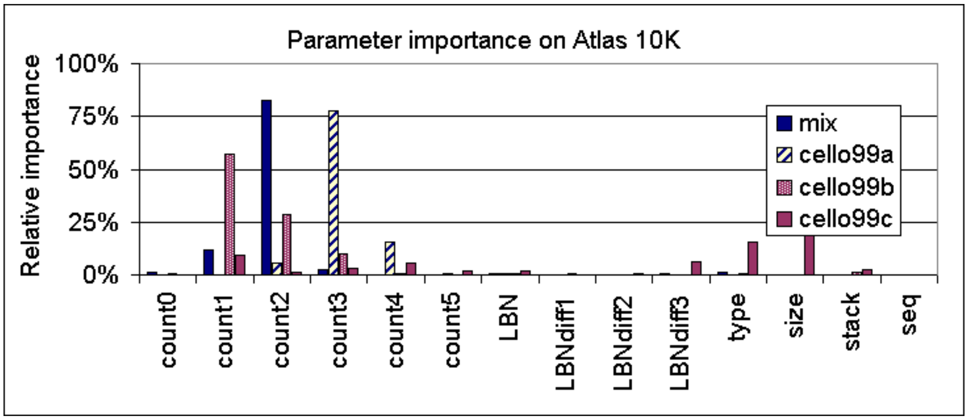
Figure 4.7: Comparison of summary response time statistics on count-based request-level models.

For example, a request following a large bursts of sequential accesses has a smaller response time than one that follows a burst of random requests even though the two might have the same count-based request descriptions. In this case, the regression models can not distinguish between the two, and the prediction depends on which type of requests the models have seen in training.

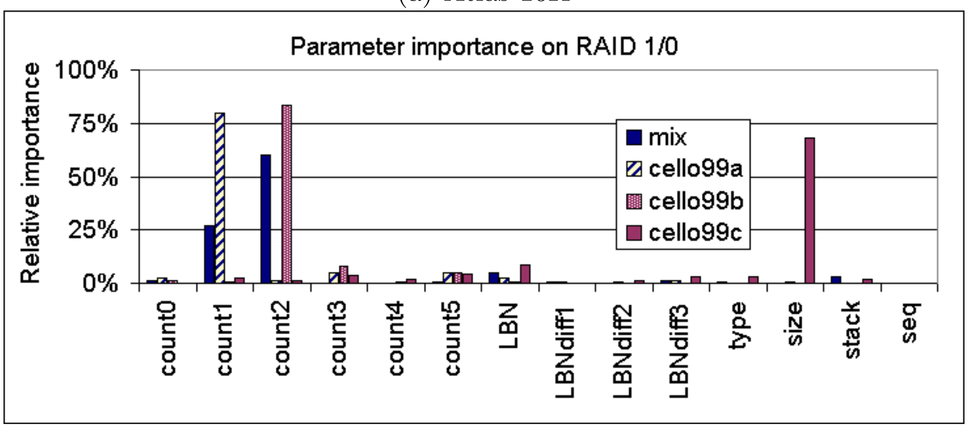
Importance of the parameters. Because of the non-linear dependence between request descriptions and response times, traditional metrics, such as the Pearson coefficient, can not correctly quantify the importance of the parameters in the request descriptions. As a result, we use the CART-specific approach described in Section 5.5 on page 87 in our analysis. In this approach, the importance of a parameter is measured by its contribution to error reduction. Please refer to Section 5.5 for more details of the calculation.

We need caution to interpret the results because the simulated measurements depend on the traces used in constructing the model. For example, if a training trace does not have any sequential accesses, then it is unlikely that sequentiality will have any importance in the constructed model. In addition, one parameter may overshadow another if the two are highly correlated, so the actual importance can be underestimated. Ideally, all the parameters should be independent in request descriptions because most regression analysis assumes independent parameters. The parameters in both types of request descriptions are strongly correlated, and overshadowing may exist.

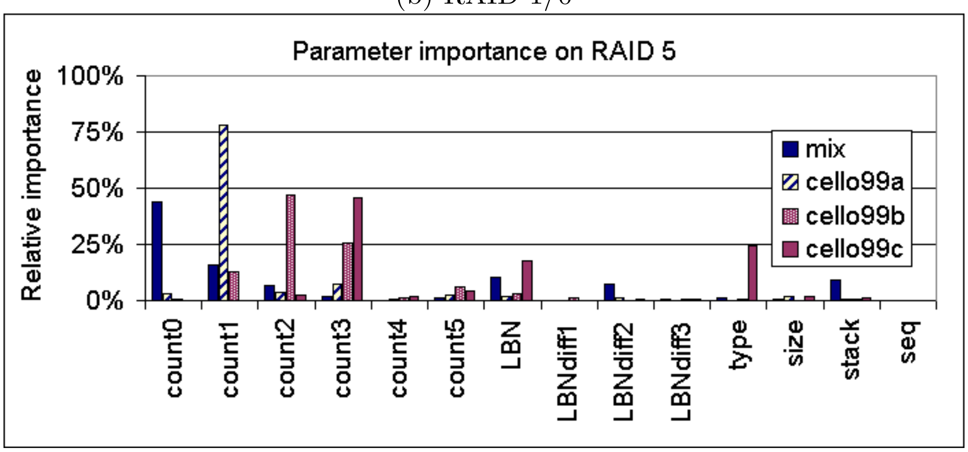
Figure 4.8 compares the importance of the parameters in the count-based request descriptions for all the workload/device pairs. That is, for each workload and device pair, we construct a request-level model and measure the parameter importance. The measurements



(a) Atlas 10K



(b) RAID 1/0



(c) RAID 5

Figure 4.8: Importance of request description parameters calculated on 12 workload/device combinations.

are obtained from the constructed models, so no prediction is involved. The detail of the algorithm is discussed in Section 5.5.

In general, the measurements are consistent with the simulated behavior of these traces on the modeled devices. For busy workloads like `cello99a` and `cello99b`, queuing time dominates response time, so the request counts are highly important. Using a large w_{max} would have little benefit because the significance of the counts diminishes after $w = 4$. For idle traces like `cello99c`, the request size shows significant contributions as data transfer time is more prominent in response time. None of the seek distances (LBNDiff) is important, indicating we can probably remove them from the request descriptions for the particular workload and device combinations we have studied.

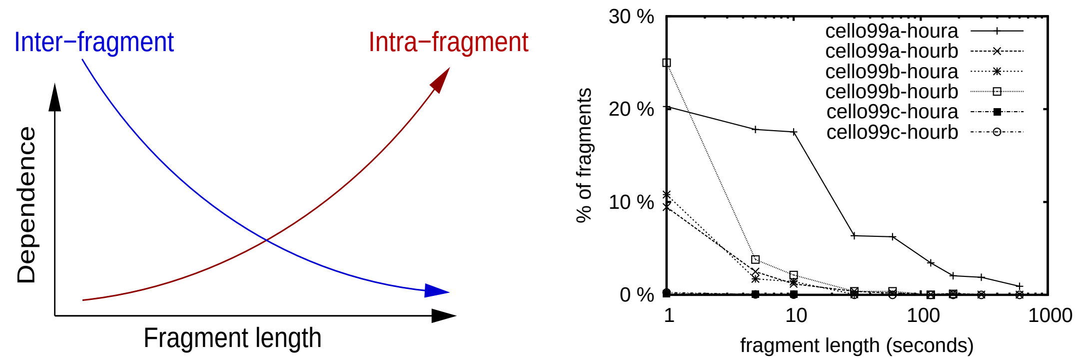
The above experiments have shown that count-based request descriptions are effective in capturing some important per-request characteristics, in particularly, the queuing effects. This is consistent with the observation in [94], in which modeling the queuing time can improve the model accuracy dramatically over a model of constant service time. We will investigate the design of workload-level descriptions in next section.

4.3 Workload-Level Description

In contrast to request descriptions, a workload-level description describes a sequence of disk requests, also known as a "workload fragment", and a workload-level model predicts the aggregate performance of the fragment based on its workload-level description.

The fragment length should reflect the correct balance between the dependence within and across the fragments. Inter-fragment dependence decreases when the fragment length grows, as Figure 4.9(a) illustrates. Inter-fragment dependence exists for several reasons. First, it is possible that when a new fragment starts, a storage device is still busy serving the requests from the previous fragment(s). Second, even if the device is idle, the performance of a fragment depends on other system status, such as the cache content and head position. At the finest granularity, each fragment contains only one request, and the performance of the request is highly dependent on the requests that arrive before the current one.

Intra-fragment dependence is the dependence between requests within a fragment. When the fragment length grows, the influence of the initial system status decreases, and the intra-fragment dependence becomes more important. The vector representation of a fragment, in this case, the workload-level description, should put more emphasis on the internal structure of the fragment. However, using a large fragment length may sometimes fail to identify transient performance outrage. The right choice of fragment length will balance the needs and maximize accurate predictions from our storage device models.



(a) inter- and intra-fragment dependence (b) percentage of fragments starting with a busy system

Figure 4.9: Fragment length chosen as a balance between the dependence in and across the fragments.

To select the fragment length in our experiments, we plot the percentage of fragments that start with a non-empty device against the fragment length in Figure 4.9(b). We selected 6 hours of different traffic volume from the `cello99` traces, and used fragment lengths ranging from 1 second to 200 seconds. The device under study is Atlas 10K. The percentages drop sharply for all the sample data when the fragment length increases from 1 second to 30 seconds, and then approach zero gradually, indicating that a fragment length larger than 30 seconds should have negligible influence from the queuing effects of previous fragments. The influence of cache content depends on the volume and footprint size, and could be slow to decay when the cache is large. Therefore, we have chosen to use 1 minute to increase the chance of detecting transient performance fluctuation.

This section describes the challenges in designing workload-level description and presents the entropy plot as an efficient tool for characterizing I/O workloads.

4.3.1 Previous Work in Workload Characterization

A workload-level description is a set of scalar parameters that captures important workload characteristics of a workload fragment. Even with a long history of workload characterization work, it is still a difficult task to determine what characteristics are important and how to quantify them efficiently because of the complexity in both storage devices and the correlation structures of I/O workloads.

For arrival processes, the average rate is the only parameter needed to describe a Poisson arrival. Modern systems, however, produce traffic that is far more complex than Poisson arrivals. Various types of computer-generated traffic have been shown to exhibit the self-similarity [36, 42, 43, 68]. Some work has used a statistical metric, the Hurst exponent [18], to quantify the degree of self-similarity. Other work [99] has divided the requests into bursts of requests based on a pre-set threshold on the interarrival time and used the average size of the bursts and the average arrival rate within the bursts as their model input. The threshold is selected to be the mean response time, which is, in turn, calculated from the performance of input workloads on the modeled device. The threshold selected based on one trace is not necessary optimal for others.

Limited work has addressed the locality patterns for I/O workloads. The LRU stack distance is a common metric of temporal locality in both web and I/O traffic to model the effects of caching [60, 110]. Spatial locality is much more difficult to quantify. Even detecting sequential accesses can be challenging [59]. The run count and run size are among popular parameters used to capture the sequentiality of I/O workloads [74, 99, 110]. Some work [40, 44] uses the seek distance distribution or the access frequencies of disk blocks incorporated with sequential accesses.

The correlation between reads and writes imposes more challenges if we model reads and writes as two three-dimensional data sets, and effectively quantifying the correlations between high-dimensional data sets remains a great research challenge. Simple models just record the percentage of reads [99]. Ganger [40] built a Markov model to model the change in request type from a request to its subsequent one. Other models maintain separate statistics for reads and writes [60].

Previous workload characterization work emphasizes on identifying important characteristics for generating I/O workloads and understanding device behavior. Our device modeling work needs a small set of important workload characteristics to make performance predictions. Because of the limitation of existing regression tools, the models uses only scalar metrics in characterizing workloads, which makes the problem even more difficult than it already is. In the following text, we introduce the entropy plot, an efficient tool for quantify the spatio-temporal behavior of I/O traffic, to supplement existing work in workload characterization.

4.3.2 Entropy Plot

The entropy plot takes advantage of entropy and mutual information to capture the spatio-temporal behavior of I/O workloads with three scalars. This section gives a brief introduction of the tool. Please refer to [115, 117] for more details on the tool.

Entropy and mutual information. Entropy is a well-known concept in information theory, and it measures the uniformity of a discrete probability function [98]. Recall that *entropy* on a random variable E, (e.g. LBN of a disk request), is defined as

$$H(E) = -\sum_{i=1}^{n} p_i \log_2 p_i, \tag{4.5}$$

where p_i is the probability that event E_i will happen (e.g. the i-th block will be accessed) and n is the total number of possible outcomes (e.g. the total number of disk blocks). H is close to 0 for a highly-skewed distribution and reaches its maximum value of $\log_2 n$ for a uniform distribution.

The *joint entropy* on two random variables is defined similarly: for a given probability function $P = \{p_{i,j}\}$ on two random variables $\{E\}$ and $\{F\}$, (e.g. arrival time and LBN of a request), where $p_{i,j}$ gives the probability that both event E_i and event F_j will happen, (e.g. a disk request on the j-th block will arrive at time i), the joint entropy on E and F is defined as

$$H(E, F) = -\sum_{i,j} p_{i,j} \log_2 p_{i,j}. \tag{4.6}$$

The *mutual information* between the two variables is

$$I(E; F) = H(E) + H(F) - H(E, F), \tag{4.7}$$

which quantifies the degree of dependence between E and F. It becomes zero if E and F are independent. Otherwise, the value is positive.

Entropy plot. Since entropy can quantify the irregularity of a discrete probability distribution, and mutual information the degree of dependence between two random variables, they should be useful in quantifying the bursty arrivals and the locality patterns of I/O workloads. The question is, then, at which granularity (i.e., time scale) we should apply them. The answer depends on the workloads. Intuitively, large time scales are preferred for busy traffic because of the long queuing time. Our answer is to calculate the entropy value at all granularities and use the rate of change as the characteristic value of the trace. Because of the strong self-similarity in I/O workloads, the rate stays the same at all time scales. Plotting the entropy value against the time scale gives the entropy plot.

In calculating mutual information, we need to decide the aspect ratio between arrival time and LBN, that is, the ratio between the interval lengths on the arrival time and LBN at a certain granularity. Our current design divides both dimensions into an equal number of units in calculation. Future work could explore the best aspect ratio.

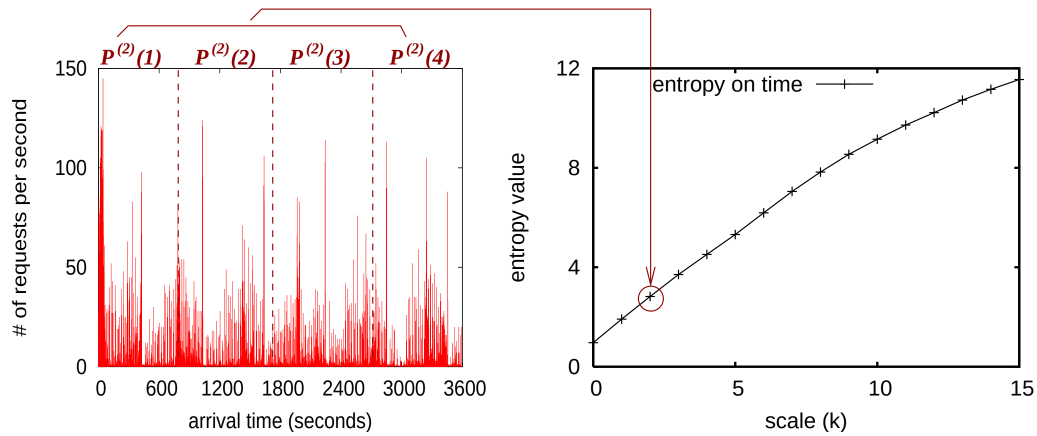


Figure 4.10: Calculating entropy plot on arrival time for the sample disk trace. At scale 2, the entire length of the arrival time is divided into 2^2 intervals of equal length. The entropy value at scale 2 is, then, calculated from the observed probabilities $P^{(2)}(j)$. The entire length of the trace is 1 hour. An interval at scale 2 is 15 minutes (1 hour divided by 2^2.)

Entropy plot in quantifying the temporal burstiness. For a given trace, calculating the entropy plot on arrival time involves two steps. First, counting the number of requests arrived in each time tick gives the marginal distribution of the trace on arrival time. We start from the finest granularity provided by the trace. If the arrival time of a time is accurate to microseconds, a time tick for calculating the distribution can be a microsecond. Second, aggregating the distribution at different scales enables one to calculate the entropy values. Assume that the trace is 2^n time ticks long. At scale k, we divide the entire length of the trace into 2^k equi-length internals and estimate the probability that a request arrives in each interval. The entropy value calculated on the 2^k intervals is the entropy value on arrival time at scale k. Plotting the entropy values against the scale gives the entropy plot on arrival time.

Figure 4.10 shows the entropy plot on arrival time for the sample trace in Figure 4.2. Because of the self-similarity in the trace, the points sit closely to a line of slope 0.702. The slope of the line characterizes how the entropy value or how the temporal burstiness changes from one scale to the next. For smooth traffic, the requests are evenly distribution between the time ticks, therefore, the entropy value increments by 1 at each scale. Otherwise, the increment becomes less as the burstiness grows. This indicate that the entropy plot slope can be used to quantify the burstiness of the arrival process. The slope has a maximum value of 1, corresponding to smooth traffic, and a smaller value for bursty traffic. The entropy plot slope in Figure 4.10 is 0.767, indicating the strong burstiness of the trace.

Entropy plot to measure the skew in the access frequencies of disk blocks. We can apply the entropy plot on LBN for a given trace using the same method. The slope characterizes the skew in the popularity of the disk blocks.

Joint entropy plot to quantify locality. To quantify the correlation between arrival time and LBN, we apply the entropy plot algorithm on the two-dimensional projection of the trace on arrival time and LBN to obtain the joint entropy plot. Similarly, at scale k, we divide the entire rectangle covered by the trace into $2^k \times 2^k$ rectangles and estimate the probability that a request falls into the rectangles. Plotting the joint entropy value against the scale gives the joint entropy plot.

We can obtain a fourth line, the mutual information, by subtracting the joint entropy plot from the sum of the entropy plots on arrival time and LBN. The mutual information measures the dependence between the arrival time and LBN in the given trace.

Figure 4.11 shows the four entropy plot lines for the sample trace in Figure 4.2. The entropy plot slope on time and LBN are 0.767 and 0.711 respectively, indicating strong temporal and spatial burstiness. The mutual information is positive, suggesting strong spatio-temporal correlation in the trace. In addition, all the four lines show strong linearity, which allows us to use the slopes of these lines to quantify the spatio-temporal behavior of the sample trace, that is, three parameters to capture the spatio-temporal behavior of the entire trace, including the temporal burstiness, the skew in disk block access frequencies, and the spatio-temporal correlation.

Joint entropy plot for other correlation structures. The limited value range of request type and size imposes difficulties on extending the entropy plot to handle dependence involving these two attributes. Actually, the entropy plot on request type has only one point in it, with the value controlled by the read/write ratio.

A quick solution is to use a fixed scale on the request size and type but change the scale on the other attributes. For example, to calculate the joint entropy plot on arrival time and request type, the scale change happens only on arrival time. Figure 4.12(a) illustrates the idea. All the requests on the two-dimensional space reside on two lines, reads on one and writes on the other. To calculate the joint entropy value at scale 2, we divide the length of time into 4 intervals and the joint entropy is then calculated on 8 rectangles. When the time scale grows, the number of intervals changes on the time dimension only. This allows us to measure the correlation between arrival time and request type.

Figure 4.12 shows the joint entropy plot involving request type and size on the sample trace. Both the joint entropy and mutual information grow linearly with the scale, making

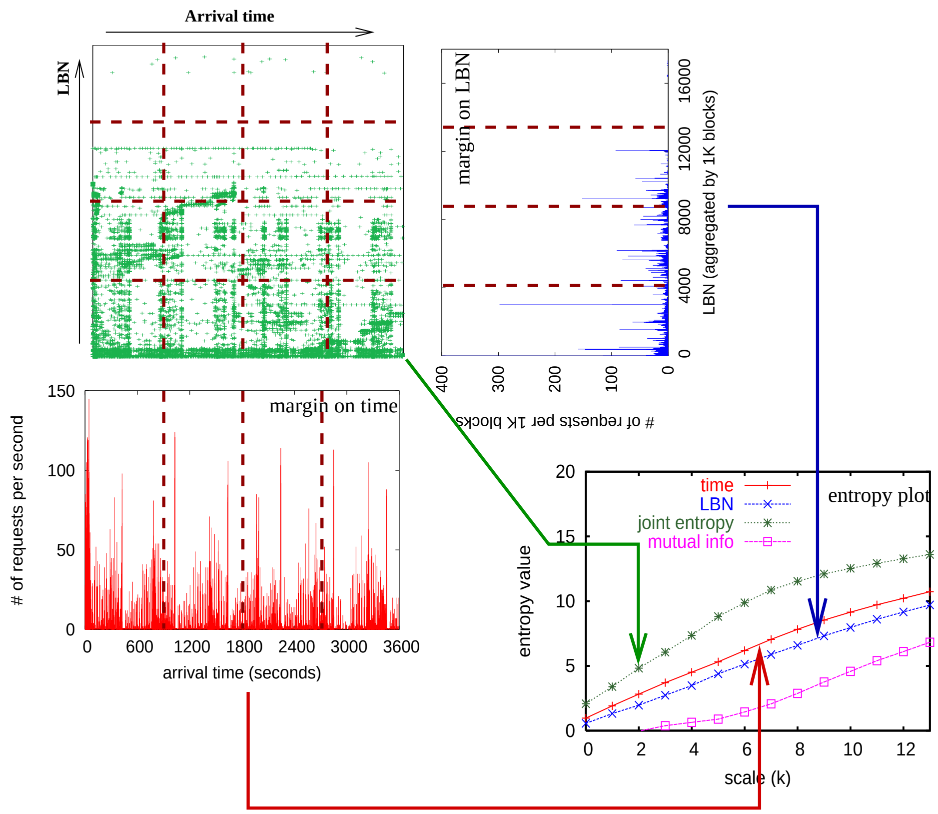
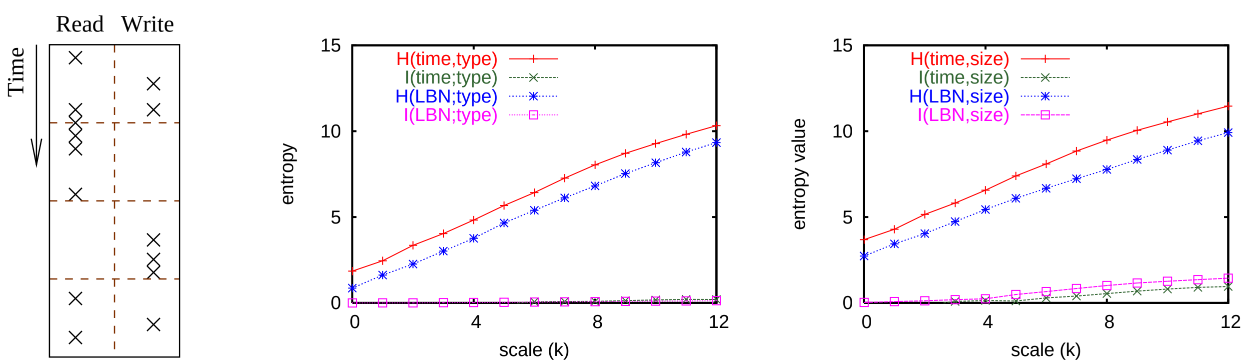


Figure 4.11: Entropy plot calculation on the sample trace in Figure 4.2.



(a) Joint entropy at scale 2 (b) joint entropy plot on type (c) joint entropy plot on size

Figure 4.12: Joint entropy plots for size- and type-related dependence on the sample trace in Figure 4.2.

it possible to use the slopes to summarize the dependence. The dependence, however, is relatively small compared to the spatio-temporal correlation for this particular trace.

4.3.3 Workload-Level Description

A workload-level description characterizes a workload fragment. The fragment length is selected to minimize inter-fragment lengths and maximize the possibility of capturing transient performance outrage. We would like to add use many parameters as possible to capture all the characteristics that might be important in determining device performance, but we would also like to limit the description size for a reasonable training time. Our design requires sampling over the workload-level description space to provide training traces of a good coverage over real-world traces. Therefore, the training time could be exponential to the number of dimensions.

Table 4.2 lists the 12 parameters in our workload-level description, which includes nearly all the scalar metrics for quantifying workload characteristics from existing work. Our evaluation has shown that these parameters are effective in capturing most of the important workload characteristics. Future work could explore the possibility of using automatic workload characterizers [65].

The parameters in our workload-level description can be categorized into three groups. The first set of parameters focuses on the arrival patterns of I/O workloads. ReqCount quantifies the traffic volume in a workload fragment. BurstSize and BurstRate are the average size of request bursts and the average interarrival time within bursts. The two parameters need a pre-set threshold (10 milliseconds in our design) to break a request stream into bursts of requests. A burst of request starts with a request with an interarrival time larger than the threshold, and all the other requests belong to some bursts. Future work could investigate the effects of the threshold value on model accuracy. EntropyTime, the entropy plot slope on arrival time, describes the temporal burstiness.

The second group of parameters characterize the locality patterns, including two entropy plot slopes (EntropyLBN and MutualInfo,) percentage of sequential accesses (Sequentiality,) the average run length (RunLength,) and the average re-reference intervals (LRUStack.)

The rest of the parameters describes characteristics involving request size and type, such as the percentage of reads (ReadRatio) and the average request size (ReqSize.)

Five of these parameters are the average values derived from distributions. The average values are not necessarily the best way to capture the distributions. We have investigated another two ways of converting the distributions into scalars in our experiments as shown in Figure 4.17 on page 59, and all three approaches yield similar performance. Further analysis has shown that it is more important to capture the dependence between the requests than

Parameters		Description	Note
Arrival process	ReqCount	The number of requests in a fragment	
	BurstSize, BurstRate	The average size of request bursts in terms of the number of requests and the average request rate within bursts	[99]
	EntropyTime	Temporal burstiness measured by the entropy plot slope	Proposed
Locality	EntropyLBN	The degree of skew on the popularity of disk blocks measured by the entropy plot slope	Proposed
	Seek	The average seek distance	
	LRUStack	The average distance between two subsequent requests to the same disk blocks in the number of distinctive requests	[6, 110]
	RunLength	The average run length in the number of requests	[99, 110]
	Sequentiality	Percentage of sequential accesses	[99]
	MutualInfo	The entropy plot slope of the mutual information between arrival time and LBN.	Proposed
Others	ReadRatio	Percentage of read requests	[44, 110]
	ReqSize	Average request size in terms of numbers of disk blocks	

Table 4.2: Parameters in workload-level descriptions. A workload-level description describes the characteristics of a workload fragment. We use 1 minute as the fragment length in our work.

these distributions.

4.3.4 Evaluation

We evaluate the effectiveness of this workload-level description using traced-based analysis in this section.

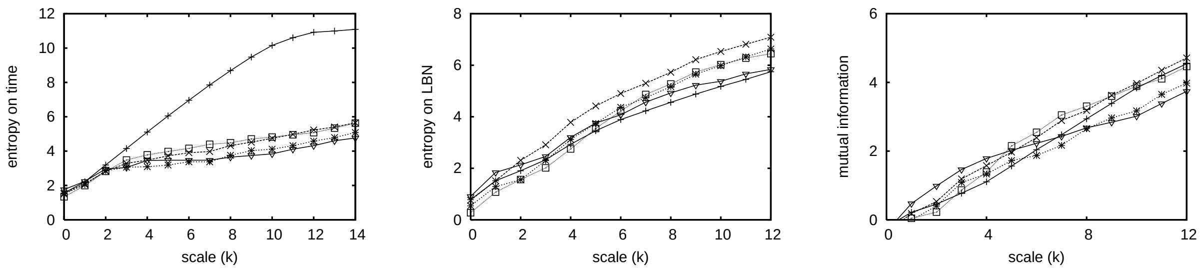
Validating linear entropy plot. The entropy plot assumes linear dependence between the entropy value and the scale at which the value is calculated. Otherwise, it is inappropriate to fit a line and use the slope to characterize the plot. Fortunately, strong self-similarity in I/O workloads usually leads to linear entropy plots.

Figure 4.13 and Figure 4.14 show the entropy plot for five one-minute fragments from `cello99a`. Strong linearity is observed in most plots and strong dependence exists between time and LBN for these fragments. In particular, one fragment shows a much larger slope in the entropy plot on time because its traffic is less bursty than the other ones. In contrast, the dependence involving request type and size is weaker, and the plots are not as linear as the ones shown in Figure 4.14. Our workload-level description includes the three types of entropy plot slopes in Figure 4.13.

Figure 4.15 shows the distributions of the three entropy plot slopes for all the traces. We observe that the `cello99` traces, especially `cello99b`, show stronger burstiness than the others, leading to their large response times. This shows from another perspective that the entropy plot is able to distinguish workloads of different characteristics.

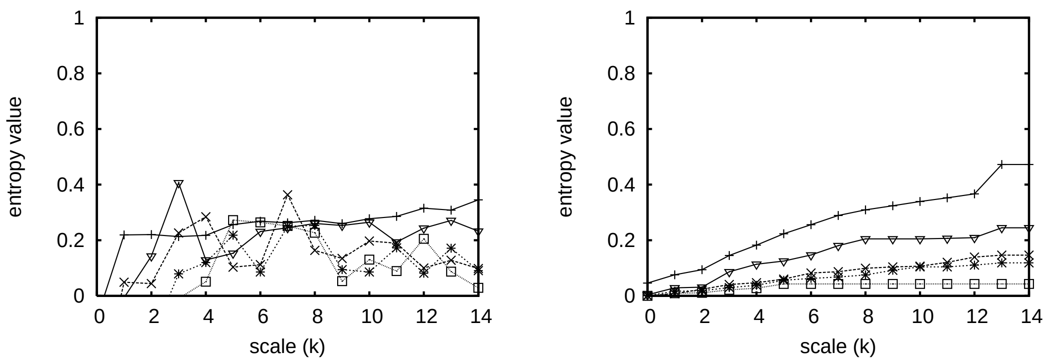
Whether to use averages in workload-level descriptions. We have selected to use the mean values to represent the distributions in our workload-level description. For complex distributions, the average value is not descriptive enough to represent the distributions. For example, the median values are more stable than the averages for long-tailed distributions, and therefore might be a better metric for characterizing the distributions. Figure 4.16 shows that the distributions derived from real-world traces are neither Gaussian or long-tailed. Figure B.3 in Appendix B on page 140 shows the probability distribution functions of these distributions.

To study the effectiveness of the averages in our workload-level description, we have built three storage device models using three ways to characterize the distributions and compared their accuracy. The first model, `average`, is based on the workload-level description listed in Table 4.2. The second model, `average+stdev`, uses the standard deviations of the distributions in addition to the ones use in the first model. The last model, `median`, replaces all the averages in model `average` with the median values of the distributions. Figure 4.17

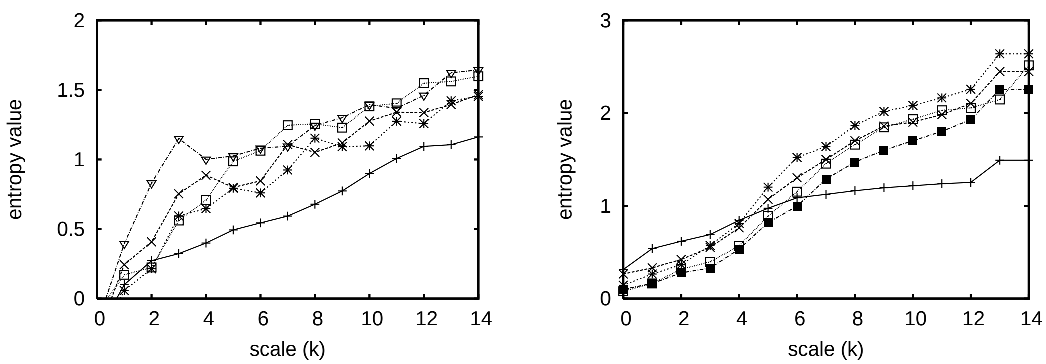


(a) entropy plot on time (b) entropy plot on LBN (c) mutual information

Figure 4.13: Entropy plot on five one-minute fragments from `cello99a` *(0:00 - 0:05 am Feb. 1, 1999).*

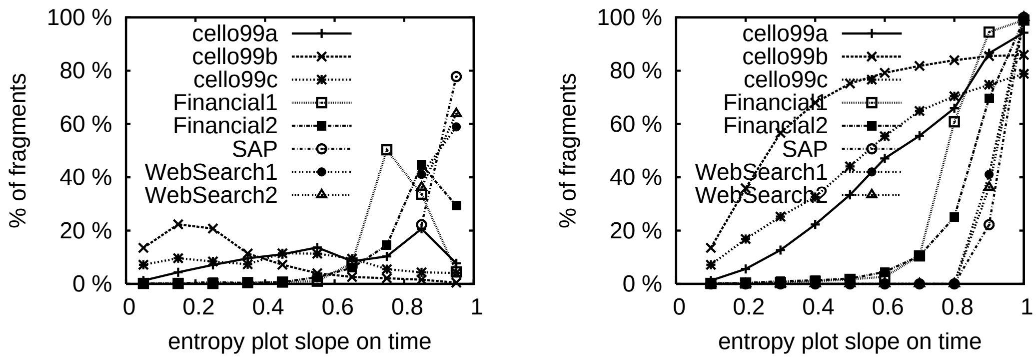


(a) time vs. type (b) LBN vs. type

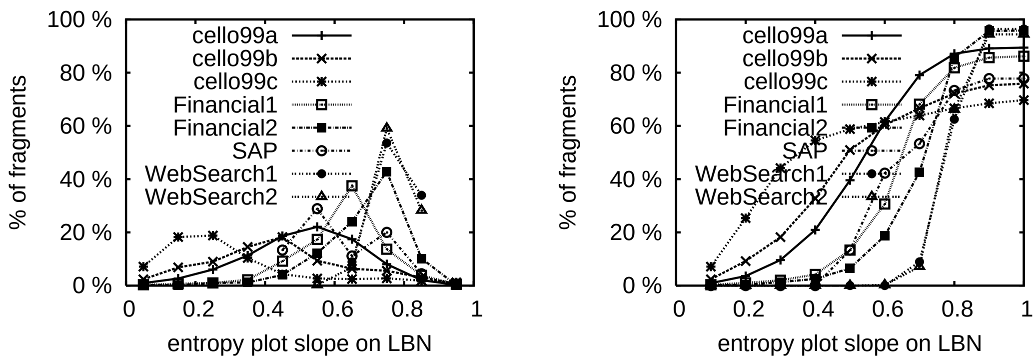


(c) time vs. size (d) LBN vs. size

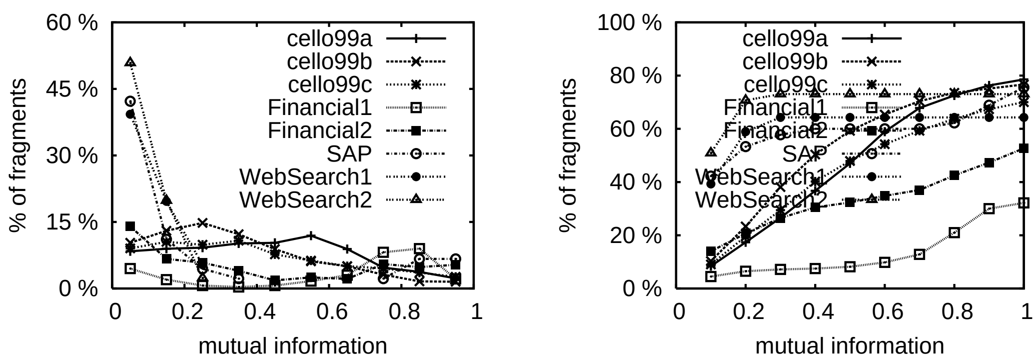
Figure 4.14: Mutual information plot involving request type and size for five one-minute fragments from `cello99a`*.*



(a) Entropy plot on time

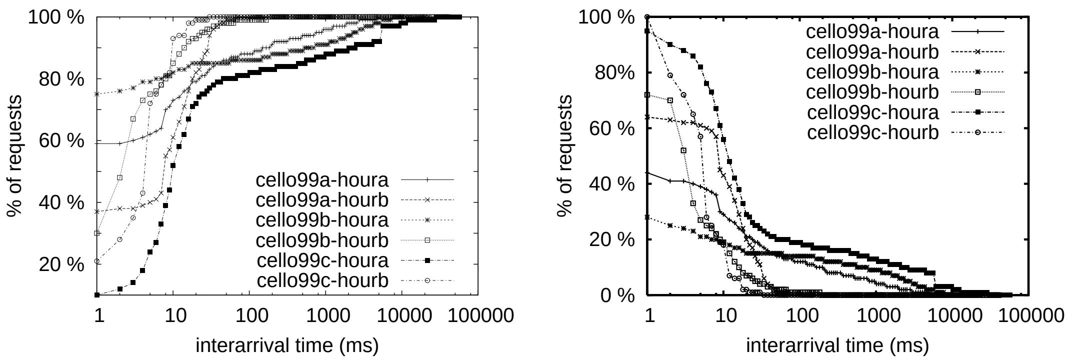


(b) Entropy plot on LBN

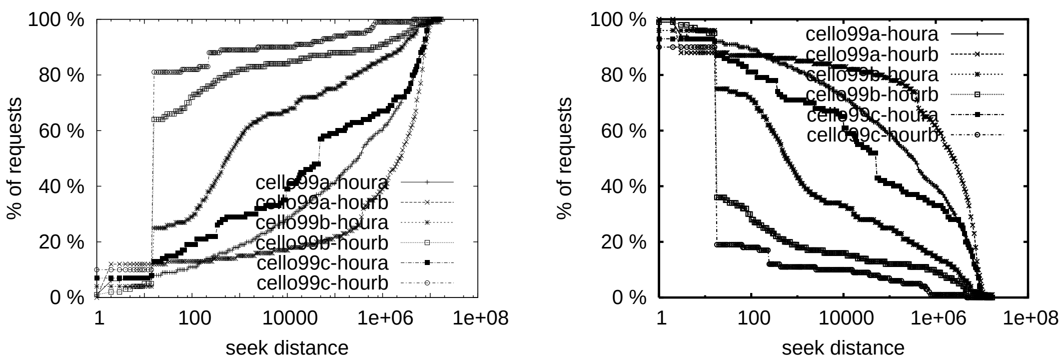


(c) Mutual information

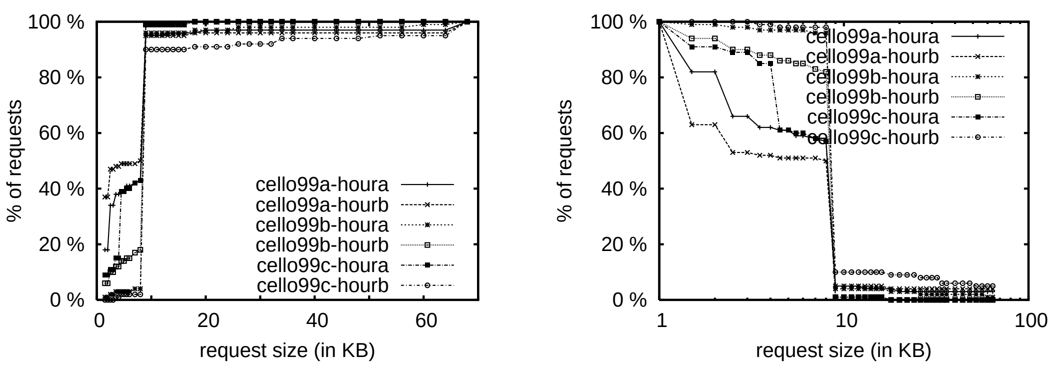
Figure 4.15: Distributions of entropy plot slopes for real-world traces (probability distribution functions on the left and cumulative ones on the right). `cello99b` *tends to have smaller entropy slopes on time, suggesting strong burstiness in its arrival.*



(a) interarrival time distribution



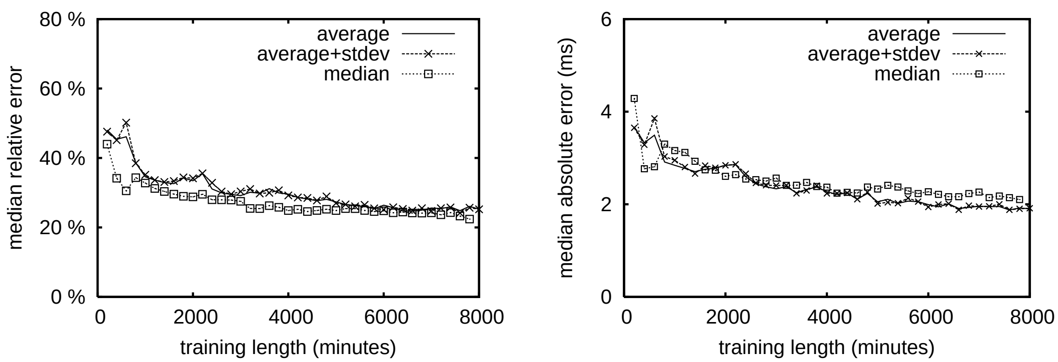
(b) seek distance distribution



(c) request size distribution

Figure 4.16: Distributions of the interarrival time, seek distance, and request size from 6 hours of different volumes from the three `cello99` *traces (cumulative distribution functions on the left and negative cumulative ones on the right). The distributions are far from Gaussian or long-tailed Pareto distributions. In addition, the distributions are of different characteristics on different traces.*

compares the prediction accuracy of the three models using `cello99a`. The training trace is 100,000 minutes of activities from the first week of `cello99a` (Feb. 1 - 7, 1999), and the testing trace is 3 days of activities from the second week of the same trace (Feb. 8 - 10, 1999). The three models perform similarly, suggesting that the averages provide the same amount of information about the distributions as the medians do, and the standard deviations provide little extra information. We argue that the majority of the remaining prediction error comes from the uncaptured correlations across the requests as suggested in Chapter 7. Therefore, we should rather focus on better characterization of the correlations for any further improvement of our models.



(a) Median relative error (b) Median absolute error

Figure 4.17: Comparison of three ways of characterizing the distributions used in the workload-level description.

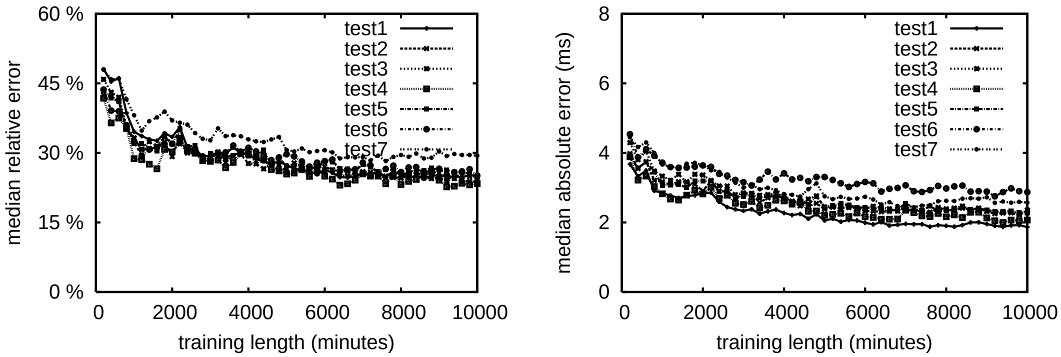
Effectiveness of workload-level description. In this set of experiments, we first explore the number of samples needed for workload-level models to achieve their best accuracy. Please note that each sample is a one-minute-long fragment, so the training length is the number of samples. We use the `cello99` traces in this experiments because the other traces are too short to provide enough data for conducting this experiment. The training data is the mixture of the workload-level descriptions drawn from three days of `cello99` (Feb. 1-3, 1999,) in a round-robin fashion to allow a proper mix in the training traces. CART models are used to predict the logarithm of the median response time over one minute, and we calculate the error after recovering the predicted value to the original scale. The rest of these traces (Feb. 4 - 28, 1999) are used in testing. Each trace is divided into seven groups, labeled as `test1` to `test7`.

Figure 4.18 to 4.20 show how the model error (in median relative and absolute error) changes with the number of samples used in training. Because of the training traces contain fragments similar to the predicted ones, the model error is small with only 200 minutes of training and converges to its best case after 3,000 minutes (samples) of training. The final median relative error is around 25% and the absolute less than 2 milliseconds except for `cello99b`. If we calculate the average absolute error over the average response time, the values are 18%, 13%, 21% respectively for the three `cello99` traces on Atlas 10K, 15%, 12%, 25% on RAID 1/0, and 21%, 18%, 4% on RAID 5. These numbers are comparable to the ones reported in some analytic models [74, 99, 110]. Figure 4.15(a) shows that, in `cello99b`, more fragments show high burstiness in arrival time (entropy plot slope less than 0.3) than the other traces, making it difficult to predict the performance of `cello99b`. The additional observation is that all the groups of `cello99` perform consistently on the model. The good prediction accuracy indicates the workload-level description is effective in capturing most important workload characteristics.

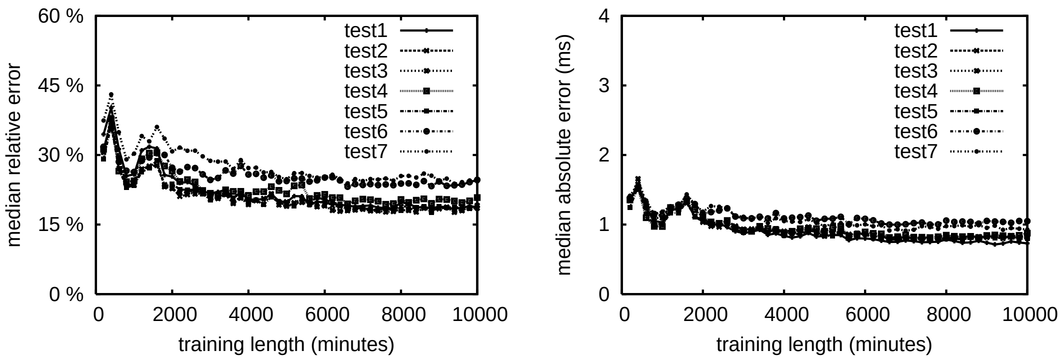
Next, we explore the capability of the models in predicting other percentile response times. We use 3,000 minutes from the mixed training traces as more training does not provide significant improvement. Figure 4.21 compares the model accuracy in predicting the 25th, 50th, 75th, and 90th percentile response times on one-minute intervals. That is, we build four CARTs for each device, one for predicting each percentile response time. Generally, because of the large value and variance of the high-percentile response times, the absolute error grows with the percentiles, but the median relative error stays at the same level across all the workloads and devices.

We conduct a little error analysis here. Chapter 7 provides a more systematic analysis to study the error sources in our approach. Figure 4.22 shows the distribution of the relative and absolute error in modeling the median response time on the Atlas 10K. The main observation is that the model can predict more than half of the fragments with high accuracy, but does poorly on a few of them by mostly underpredicting. One suspicion is that the large error happens on a certain type of fragments. The simplest form might be that the error depends on one parameter value.

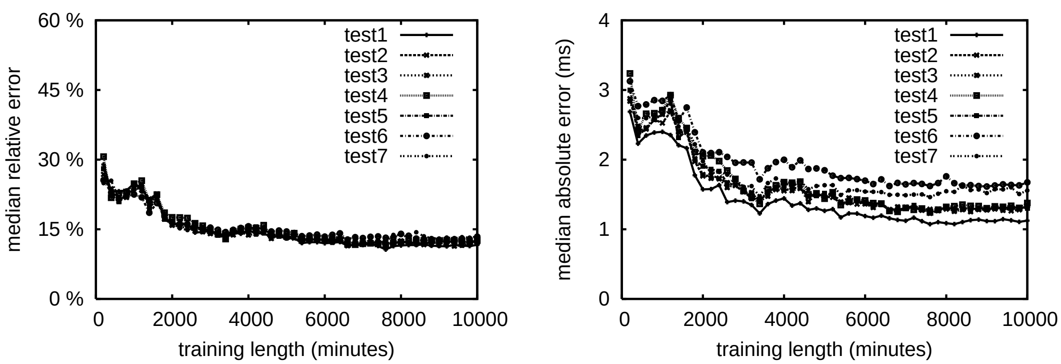
To test whether the simple conjecture is true, we bucketize the value of single workload-level description parameters and calculate the aggregate prediction errors by buckets. A clear trend of the prediction error may suggest that the conjecture is true. Figure 4.23 shows the results of such aggregation on the three `cello99` traces by bucketizing the request count. First, the fragment count distribution is different from trace to trace, confirming the difference in the three traces. Second, the model error shows little dependence on the request count. We have repeated the same experiment on other workload-level description parameters and made the same observation: there is little dependence between any workload-level description parameters and the model error. The conclusion is that the error either depends



(a) Modeling Atlas 10K

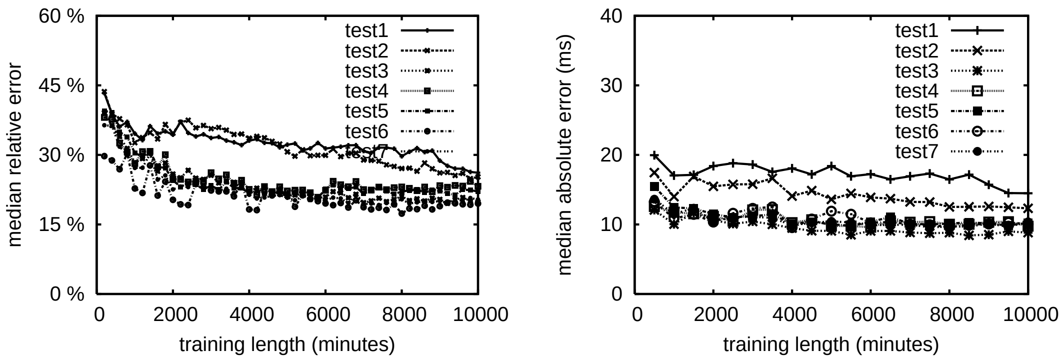


(b) Modeling RAID 1/0

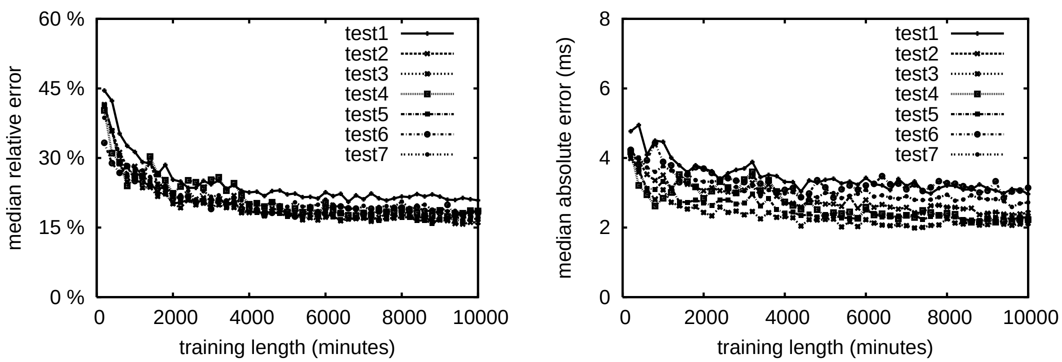


(c) Modeling RAID 5

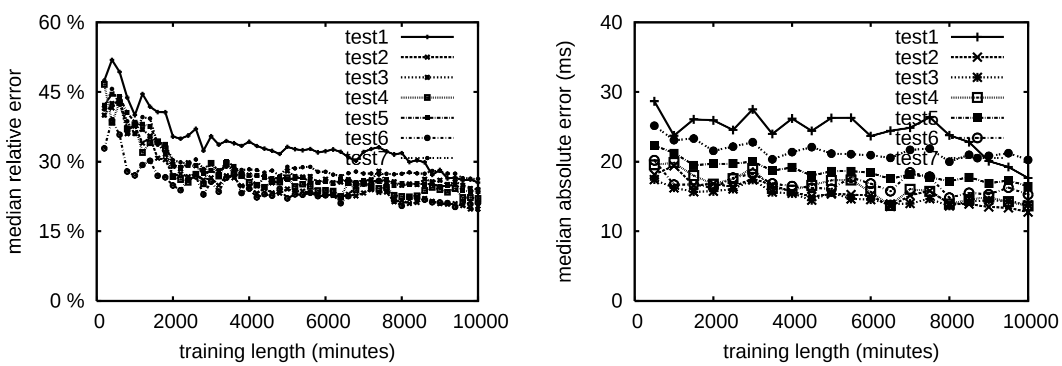
Figure 4.18: Workload-level model accuracy on `cello99a` *in predicting the median response time over one-minute intervals.*



(a) Modeling Atlas 10K

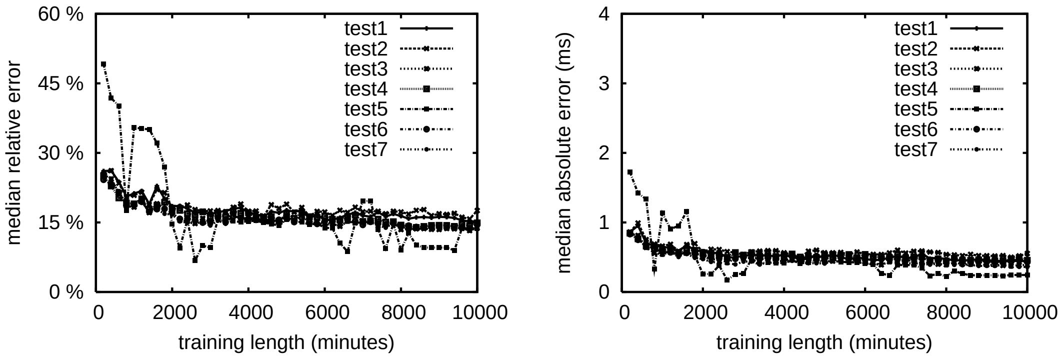


(b) Modeling RAID 1/0

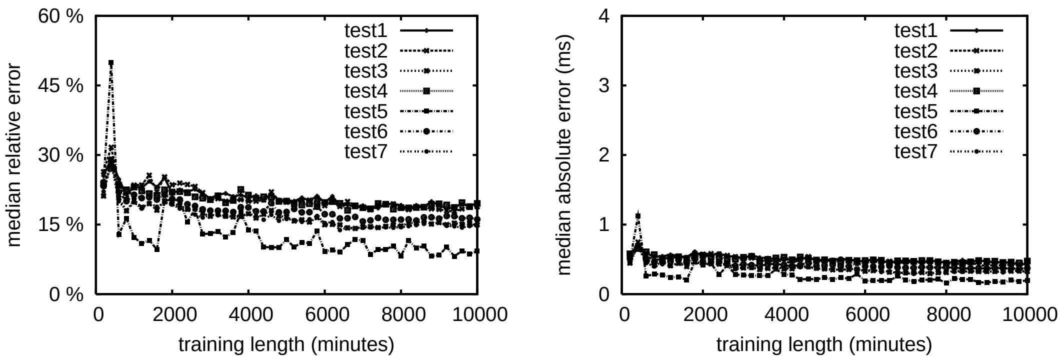


(c) Modeling RAID 5

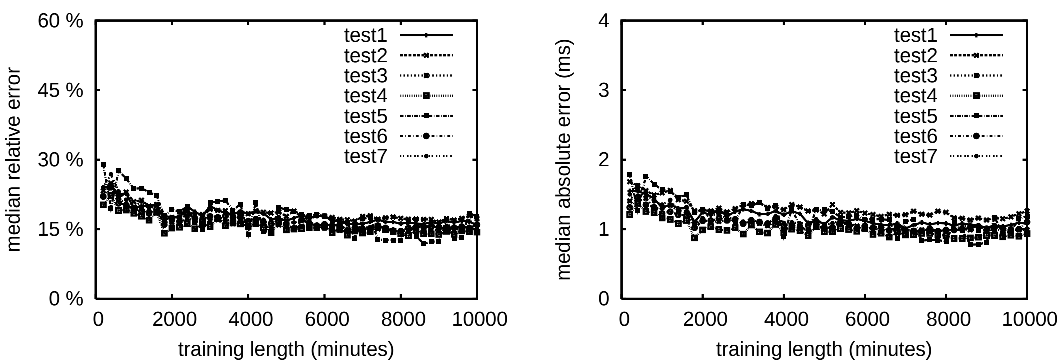
Figure 4.19: Workload-level model accuracy on `cello99b` *in predicting the median response time over one-minute intervals.*



(a) Modeling Atlas 10K



(b) Modeling RAID 1/0



(c) Modeling RAID 5

Figure 4.20: Workload-level model accuracy on `cello99c` *in predicting the median response time over one-minute intervals.*

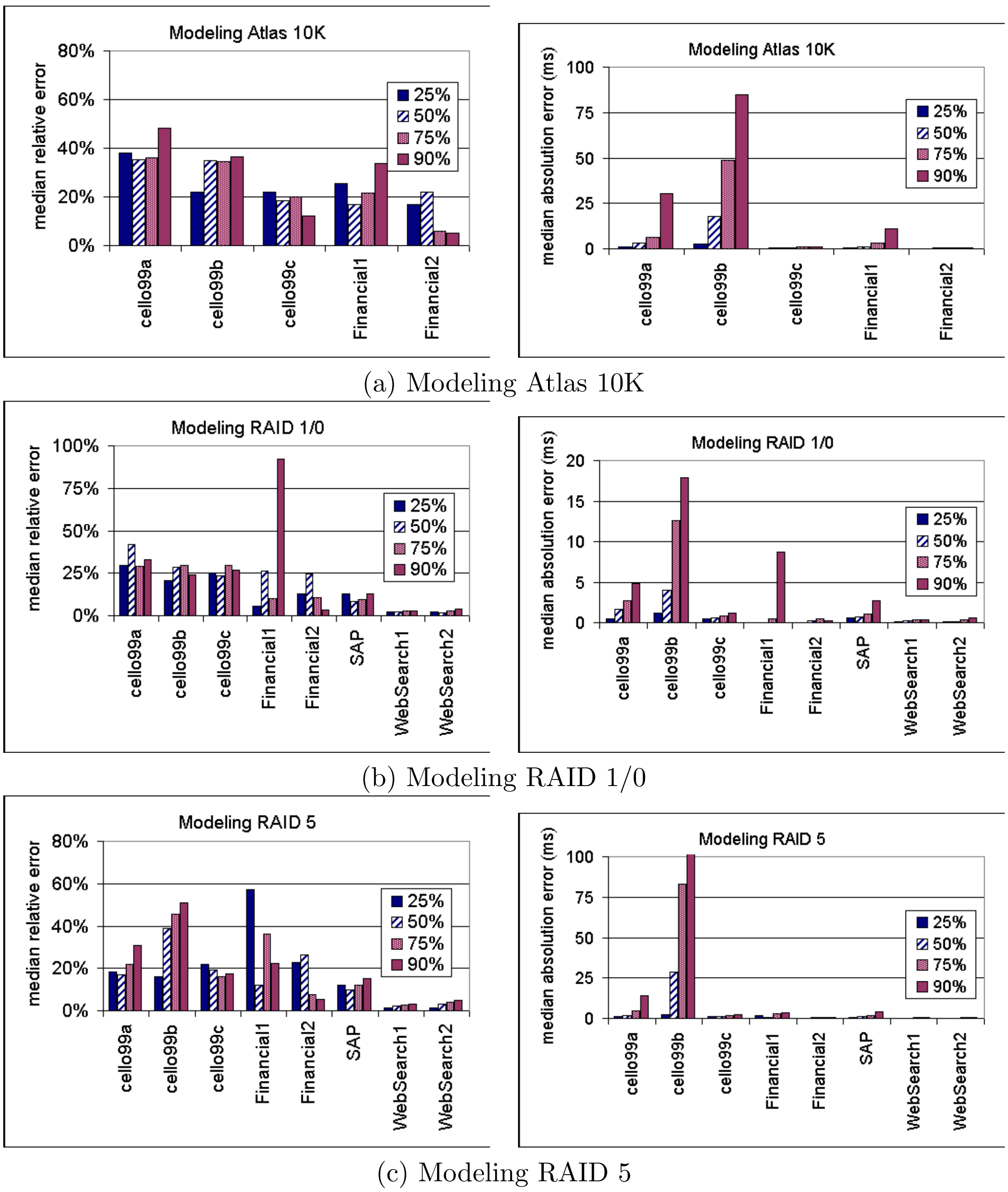


Figure 4.21: Workload-level model accuracy in predicting the 25th, 50th, 75th, and 90th percentile response times over one-minute intervals.

on more than one parameters, or is distributed in the workload-level description space randomly. This could be due to unobserved variables in the modeling approach and/or missing characteristics in workload-level descriptions. Distinguishing these two types of error is challenging.

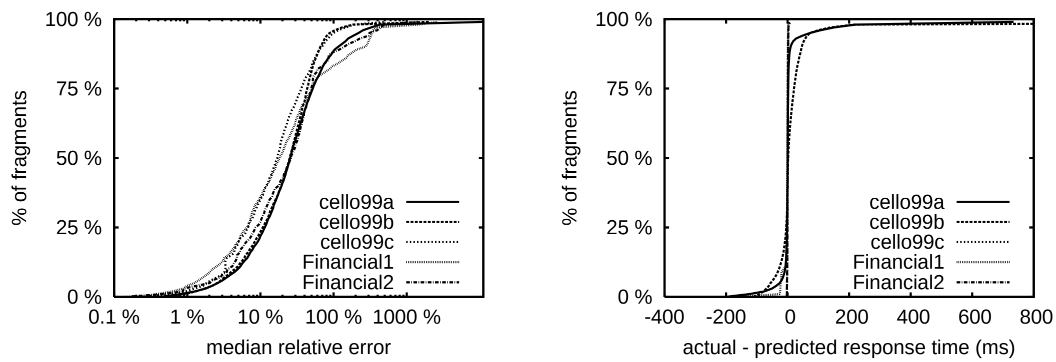
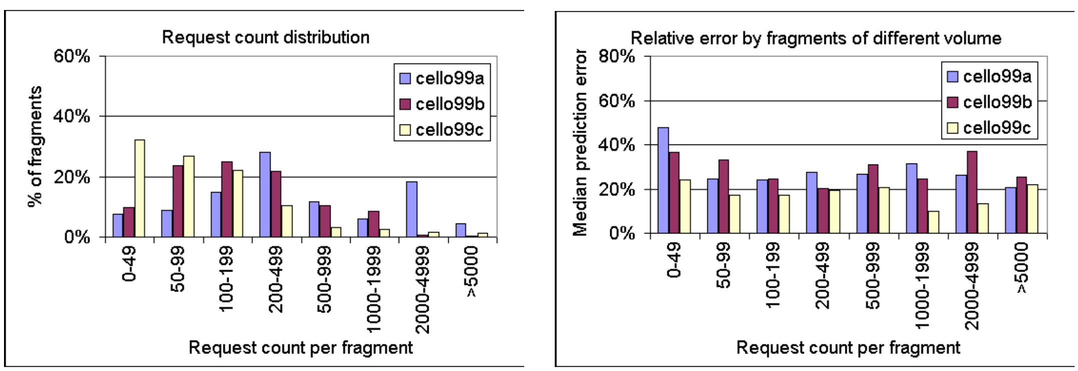


Figure 4.22: Relative and absolute error distribution in modeling Atlas 10K.



(a) fragment counts (b) median relative error

Figure 4.23: Model error grouped by the request counts in the fragments.

Importance of workload-level description parameters. We can calculate the importance of the workload-level description parameters using the CART-specific method described in Section 5.5. We explore how the importance changes along three dimensions: the devices being modeled, the target performance metric, and the training traces, as shown by the three plots in Figure 4.24. In (a), we use the mixture of the training part of the `cello99` and `Financial` in building storage devices for the three storage devices to predict the median response time over one-minute intervals. In (b), we build four storage devices models for Atlas 10K using the above mixed trace to predict four percentile response times

Feature		request-level model	workload-level model
Training	Trace replay	4,000 requests(5 minutes)	3,000 minutes (2.4 millions of requests)
	Feature extraction	1 minute	45 minutes
	Total	6 minutes	3,045 minutes
Prediction	Feature extraction	60 minutes	12 minutes
	Regression	34 minutes	0.57 second
	Total	94 minutes	12 minutes

Table 4.3: Comparison of request and workload-level device models in computational and storage overhead. All the numbers are based on predicting three one-day-long traces from `cello99`.

over one-minute intervals. In (c), we build five storage devices based on five individual traces and predict the median response time over one-minute intervals. Generally, the interarrival time, burst size, and burst rate are the most important because of the dominating queuing time in the training traces.

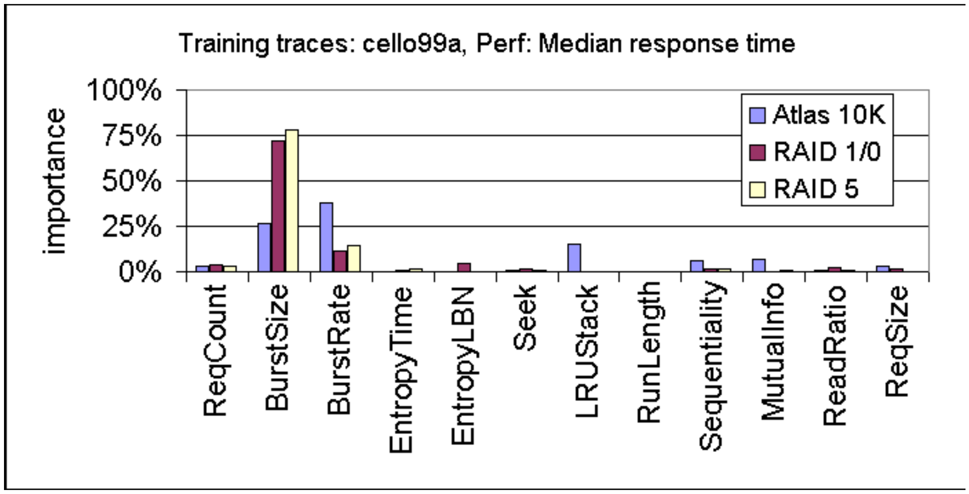
In summary, the experiments on real-world traces have shown that the workload-level models offer decent predictions in predicting aggregate performance when the training and testing traces are similar, indicating the effectiveness of the workload-level descriptions. For traffic of low load, data transfer time becomes an important component in response time, and the average request size makes significant contribution in prediction. For busy traffic, the temporal burstiness becomes important as queuing time dominates response time.

4.4 Comparison of Two Representations

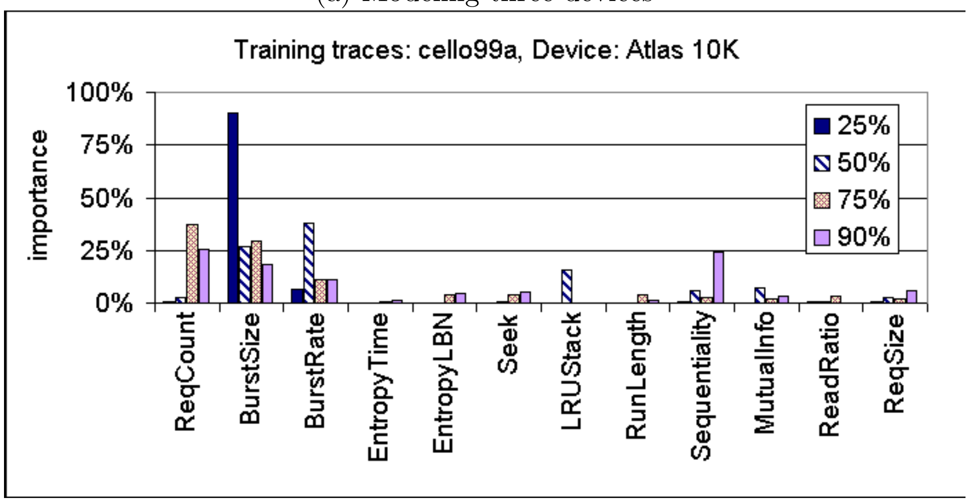
We have introduced two types of learning-based models depending on the vectors they work on. The request-level models predict per-request response times from request descriptions. The workload-level models predict aggregate performance of workload fragments from workload-level descriptions.

Computational overhead. Table 4.3 shows the overhead of the two types of models in predicting three traces from `cello99`, one day from each. We observe a clear tradeoff between training and prediction time for the two types of models.

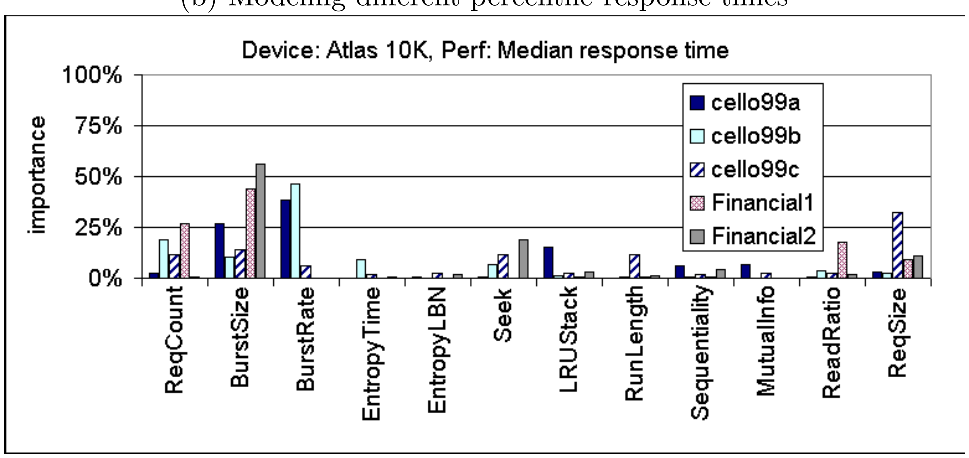
Prediction is more efficient in workload-level models. The actual saving in computation depends on the volume of the testing trace. The prediction phase includes two steps: generating the descriptions and running the descriptions against a regression model. In



(a) Modeling three devices



(b) Modeling different percentile response times



(c) Trained on different workloads

Figure 4.24: Parameter importance in workload-level models on different devices, target performance metrics, and training traces.

request-level models, each request becomes a vector, so the prediction time is linear in the number of requests. For workload-level models, each one-minute fragment produces a vector, so prediction is much faster. For example, workload-level models are more than 7 times faster when measured on `cello99`.

For the same reason, request-level models require less training time than workload-level models. A one-minute-long trace produces only one sample for the latter, but can contain hundreds of requests, that is, hundreds of samples for the former. In our experiments, we use 4,000 requests (around 5 minutes) to construct request-level models and 3,000 minutes of traces to construct workload-level models.

When based on efficient regression tools such as CARTs, the regression models usually only take less than 10 KB storage space, which can easily fit in devices themselves. Note that request-level device models are able to generate the entire response time distribution. A workload-level model can predict only one target aggregate performance metric. One needs to build a separate workload-level model for each metric, but the storage overhead remains small even if building several regression models.

Model accuracy. Figures 4.25 to 4.27 compare the accuracy of the request- and workload-level models in predicting the four percentile response times on the three devices using all the real-world traces. Table 3.4 on page 24 lists the average response times of these traces on the three devices. The main observation is that there is no clear winner between the two because the ranking depends on the workloads used in evaluation. The workload-level models seem to be more consistent.

First, both models can predict most traces with a median relative error less than 30%. In some cases, the request-level models show a huge relative error probably because of their small response time and because of the inadequacy of request descriptions. For example, the models perform poorly because sequentiality is not entirely captured by request descriptions. The Count parameters take into consideration only the number of requests arrived just before a request, but not their response times. Consequently, the models can not distinguish requests that follow a sequential run from ones that follow a burst of random accesses, while in practice, the latter should experience a longer waiting time. Overall, both vector representations can capture some important workload characteristics, and the performance of the workload-level models is more consistent.

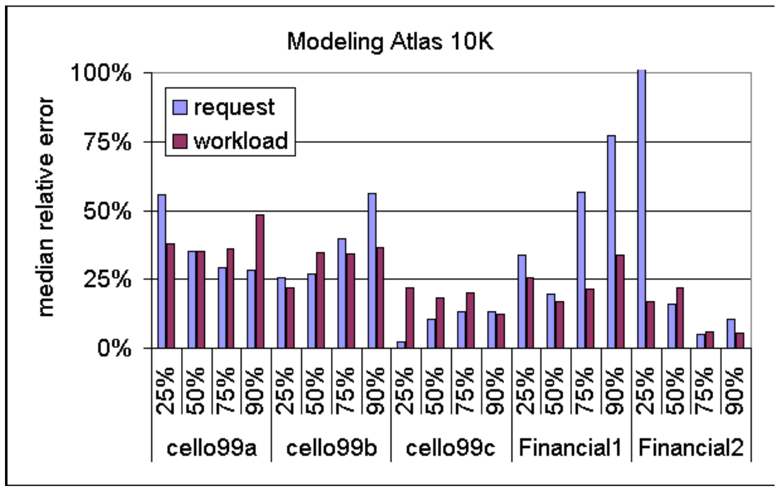
Second, a similar relative error is observed across percentiles. Large percentiles are usually more difficult to predict than small ones due to the large variance in their values. As a result, the absolute error tends grow with the percentile. Since the actual values increase at the same time, the relative error usually stays the same. The observation suggests that the characteristics captured by the descriptions are equally important for all the devices and percentiles.

In some cases, the two types of models can rank differently in relative and absolute errors. For example, in prediciting the 90th percentile response times for `cello99b` on RAID 5, the request-level models are better than the workload-level models in median relative error, but worse in median absolute error. The difference is due to our error calculation method. Because we calculate the median relative error by first computing the relative errors for all the data points and taking the median, a different absolute error distribution of the same median absolute error can lead to a different median relative error. Other studies have reported the average relative error as the ratio between the average absolute error and the average response time, in which case, the average absolute and relative errors should always produce the same ranking [74, 99]. Our error measurement is more consistent with the statistical meaning of the median relative error, and the different ranking is the natural result of the algorithm.

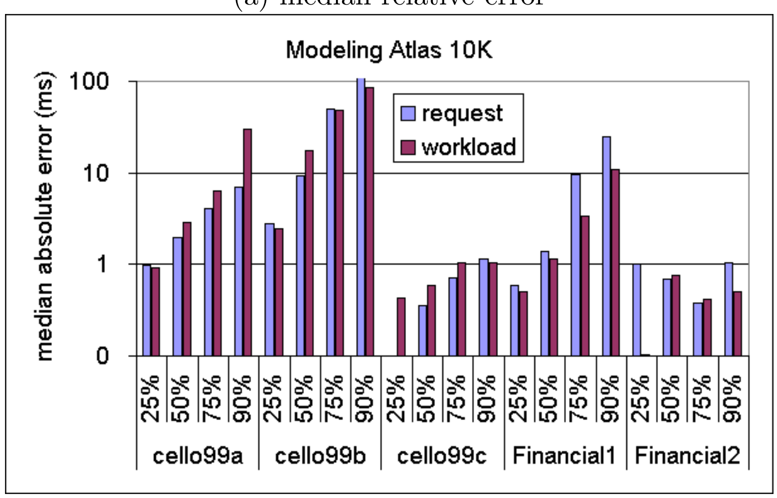
We select workload-level models over request-level models for further investigation in this work because of their stable performance and low prediction overhead. Long training is tolerable because one only needs to build models once, and it could be done by manufacturers.

4.5 Summary

This chapter presents the issues and solutions involved in obtaining the vector representations of I/O workloads. The vector representation can operate at two levels. A request description characterizes a single request and captures the important characteristics in determining its response time. A workload-level description, on the other hand, summarizes the characteristics of a sequence of disk requests. As a result, the former requires more intensive computation during the prediction phase. The experiments have shown that the vectors can offer models of comparable accuracies for most workloads. The workload-level models are more efficient in computation and provide more stable predictions with median relative errors of 10% to 50%.



(a) median relative error



(b) median absolute error

Figure 4.25: Comparison of request- and workload-level device models in modeling Atlas 10K.

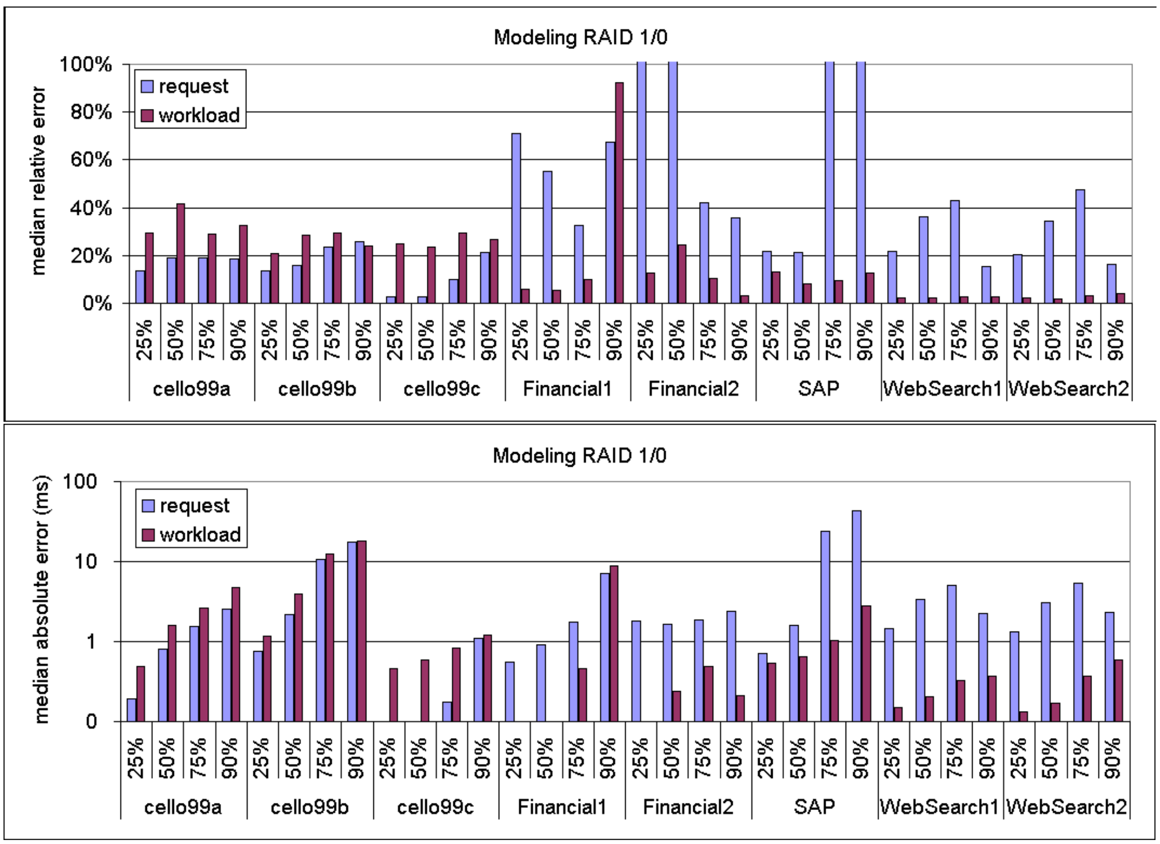


Figure 4.26: Comparison of request- and workload-level device models in modeling RAID 1/0.

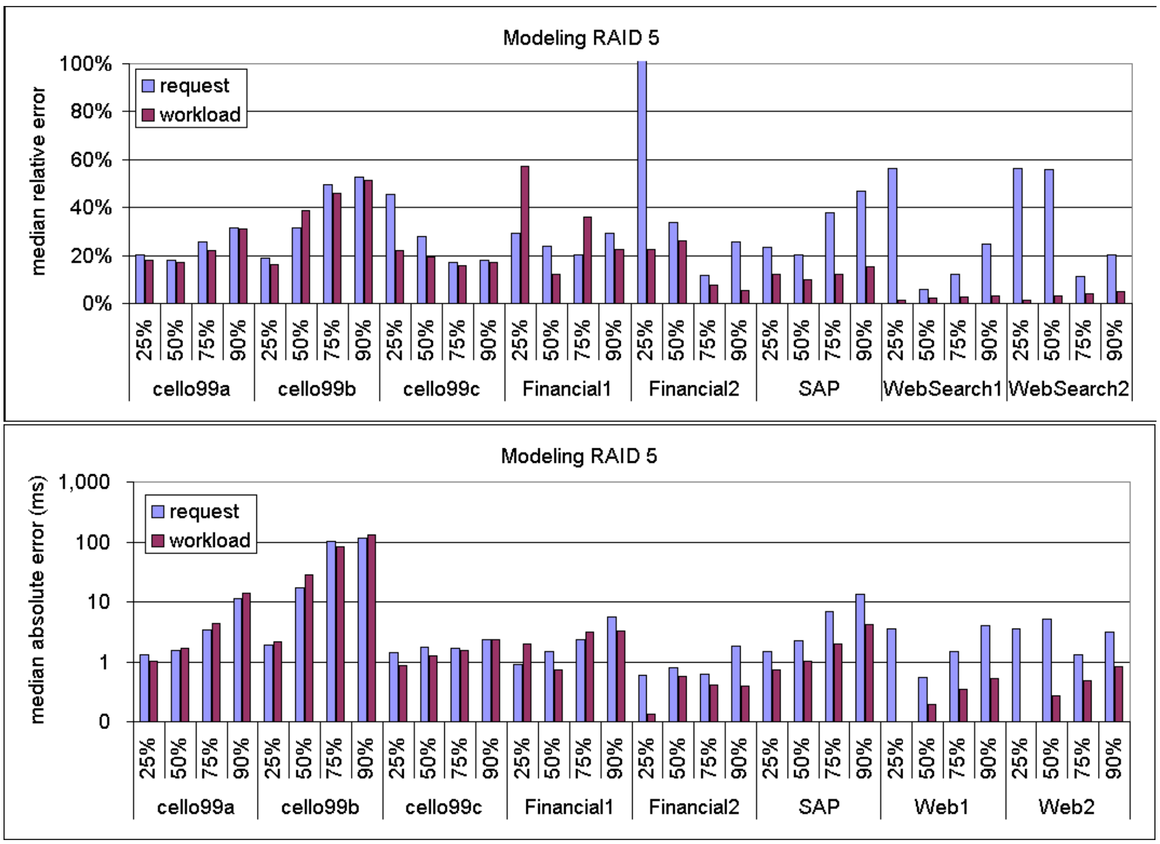


Figure 4.27: Comparison of request- and workload-level device models in modeling RAID 5.

Chapter 5

Selecting A Regression Tool

A regression tool models a real-value function over vectors in a multi-dimensional Cartesian space. In our device models, it models the target performance as a function over I/O workloads, with workloads represented as vectors. This chapter evaluates the effectiveness of popular regression tools in modeling such storage device behavior.

The chapter starts with the desired features of the regression tool in the learning-based models in Section 5.1. Section 5.2 briefly describes five popular regression tools, and Section 5.3 discusses various parameter tuning algorithms. Finally, we justify our choice of CART by empirical studies in Section 5.4 and provide several analysis techniques for understanding the models in Section 5.5.

5.1 Device Modeling Using Regression Tools

The learning-based models uses a regression tool to model device behavior as a function over the vector representations of an I/O workload. That is, the target performance metric, Y, is modeled as a function of vector X.

$$Y = f(X) + \epsilon \tag{5.1}$$

where $X \in \Re^d$, $Y \in \Re$, and ϵ is zero-mean noise. Here, $\Re^d$ is a d-dimensional Cartesian space. The term, ϵ, captures the intrinsic randomness of the data and the variability contributed by unobservable variables. For example, the random error could come from different initial status of storage devices and randomness of the physical mechanism and measurement.

The previous chapter presents two vector designs. In request-level models, X is a request description, and Y is the response time of the corresponding request. In workload-level models, X is a description of a workload fragment, and Y is an aggregate performance of the fragment.

In both types of models, the regression tool learns the mapping from X to Y through training. The model construction algorithm replays training traces on a given device and collects their response time information. The regression tool can use the set of Xs and Ys derived from the training traces to construct a regression model. The model will be consulted to predict the Y for a given X in prediction. Therefore, the regression tool should store the mapping efficiently and make fast and accurate predictions. We first give a brief introduction to five popular regression tools before evaluating their effectiveness in constructing storage device models.

5.2 Regression Tools

Regression tools are designed to model real-value functions on a multi-dimensional Cartesian space. Regression tools differ in the way they store the information and in the assumptions they make on the structure of the data.

We give a brief description of popular regression tools in this section using a synthetic one-dimensional data set. The data set is generated using

$$y_i = x_i^2 + \epsilon_i, \qquad i = 1, 2, \ldots, 100, \tag{5.2}$$

where x_i is uniformly distributed within (0,100), and ϵ_i follows a Gaussian distribution of $N(0, 100)$. That is, the Guassian distribution has a mean value of 0, and standard deviation of 100. We use 100 data points in training.

5.2.1 Linear Regression

A linear regression model [96] uses a linear function of X to approximate the value of Y. The model uses $\hat{f}$ to approximate f, where

$$\hat{f}(X_i) = \hat{\beta}_0 + \sum_{j=1}^{d} \hat{\beta}_j \times X_i^{(j)}; \quad \beta_j \in \Re forallj = 0, 1, \ldots, d.$$

The values of β_is are selected to minimize the error, usually measured by the mean squared error, on the training data. This can be achieved using simple matrix operations over the training data.

$$\hat{\beta} = (X'X)^{-1}X'Y,$$

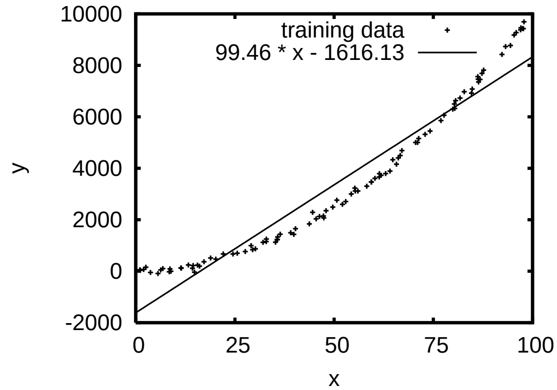


Figure 5.1: Linear regression model on the sample data set.

where

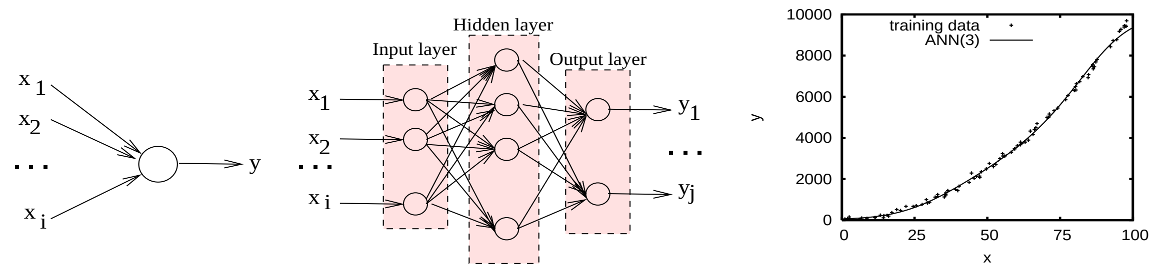
$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_k \end{bmatrix} \qquad X = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \\ 1 & X_n \end{bmatrix} \qquad Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

Linear regression models are easy to construct and efficient in prediction. However, they can incur large errors when the target function is not linear, as shown in Figure 5.1. Therefore, linear regression models are inappropriate for storage device modeling because of the non-linear behavior of storage devices. Some regression models using polynomials and splines can model non-linear functions, but require users to select the degree of the polynomials and/or the division points. The parameter selection could be difficult for high-dimensional data sets. We do not consider these regression tools in our study.

5.2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) [92] are inspired by the way biological nervous systems, such as brains, process information. An ANN usually consists of a set of highly interconnected processing elements, known as "neurons", working in unison to approximate the target function.

A neuron is a device of many inputs and one output, as shown in Figure 5.2(a). It determines the output by making a transformation of the weighted sum of the inputs according to a pre-defined function. Common transfer functions include linear, threshold, and sigmoid functions. A set of neurons can be connected to approximate complex non-linear functions.



(a) A neuron (b) A 3-layer feedforward network (c) Regression line

Figure 5.2: Artificial neural network example. (c) shows the regression line for the network constructed on the sample data set.

A typical network usually consists of three layers: the input, hidden, and output layers, and the topology is constrained to be feedforward, as shown in Figure 5.2(b). The number of neurons in the output and input layers are equal to the output and input variables respectively. The number of neurons in the hidden layer determines the power of the network. Complex functions need more hidden nodes. A network of two hidden layers is powerful enough to model any function [47]. Users need to specify the network structure and the transfer function. The training phase involves the process of finding the set of weights that minimizes the model error during the training phase. The popular back-propagation algorithm iteratively adjusts the weights by an amount proportional to the difference between the desired output and the actual output. An additional parameter, the learning rate, controls the amount of the adjustment in each step. An ANN requires the input and output values to be within the range of [0,1], so data sets may need rescaling in the pre-processing step.

Figure 5.2(c) shows the regression line produced by a network with 3 hidden nodes for the sample data set. The regression line matches the actual function closely.

5.2.3 Support Vector Machine

A Support Vector Machine (SVM) [25] transforms the input data into a feature space $\mathcal{F}$, by $\Phi(X) = \mathcal{X} \to \mathcal{F}$, and builds a linear model in the feature space. The feature space, $\mathcal{F}$, is usually of a higher dimensionality than the original vector space. A non-linear mapping allows SVMs to solve non-linear problems.

Consider the linear regression model $f(\Phi(X))$ on the feature space.

$$f(\Phi(X)) = \langle w, \Phi(X) \rangle + b \text{ with } w \in \mathcal{F}, b \in \Re \tag{5.3}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathcal{F}$. In ϵ-SV regression, the goal is to find w and b

so that $f(\Phi(x_i))$ has at most ϵ deviation from Y_i for the training data. That is, the error for all the data points should be smaller than ϵ. Such a solution, however, is not always feasible. The Vapnik's ϵ-insensitive loss function can be added to the model to relax the constraint. The loss function is defined as

$$|Y - f(X)|_\epsilon := \max\{0, |Y - f(X)| - \epsilon\}. \tag{5.4}$$

Figure 5.3(a) depicts the loss function graphically.

The target of the regression is then to minimize

$$\frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} |Y_i - f(X_i)|_\epsilon. \tag{5.5}$$

Written as a constrained optimization problem, this is equivalent to the following optimization problem after the application of the Lagrange multipliers.

$$\begin{array}{ll} \text{maximize} & -\epsilon \sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)Y_i \\ & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle\Phi(X_i), \Phi(X_j)\rangle \\ \text{subject to} & \begin{cases} \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases} \end{array} \tag{5.6}$$

The regression estimate takes the form

$$f(X) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle\Phi(X_i), \Phi(x)\rangle + b. \tag{5.7}$$

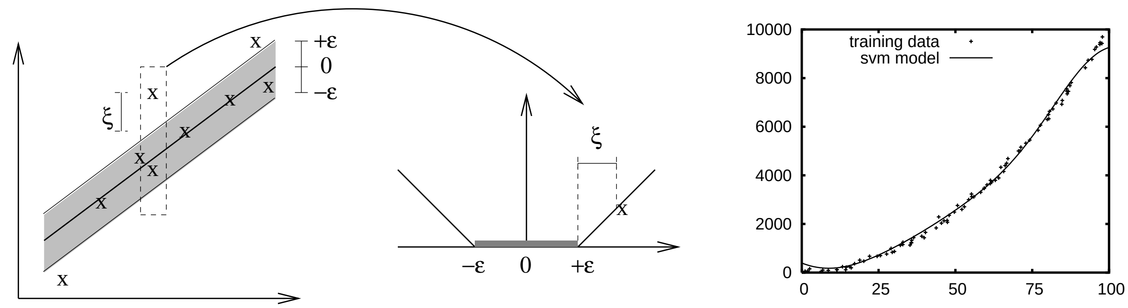
The importance of the above equation is that the SV algorithm depends only on the dot products between $\Phi(X_i)$s. This allows us to solve the equations without computing $\Phi(x)$ as long as we can compute their dot products. This is achieved using kernel functions

$$k(X_i, X_j) := \langle\Phi(X_i), \Phi(X_j)\rangle. \tag{5.8}$$

Therefore, the final solution can be written as

$$w = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\Phi(X_i) \text{ and } f(X) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)k(X_i, X) + b. \tag{5.9}$$

We can calculate the predictions from the dot products between the predicting vector and the training data.



(a) Soft margin loss (b) Regression line of SVM

Figure 5.3: A support vector machine example. (a) shows the soft margin loss corresponding to a linear support vector machine. (b) shows the regression line for the support vector machine built on the sample data set.

The kernel functions should have certain mathematical properties. One commonly used kernel function is the radial basis function because of its localized and finite responses across the entire range of the real x-axis.

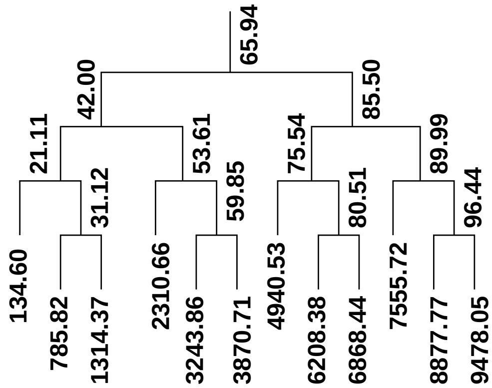
$$K(x_i, x) = exp(\gamma||x - x_i||^2). \tag{5.10}$$

Figure 5.3(b) shows the regression line for the SVM constructed on the sample data set using the radial basis function. We use an efficient implementation, SVM^{light} [55], in our experiment. We normalize the input data set so that the values on all the dimensions have a zero mean and unit standard deviation. SVMs are able to model non-linear functions.

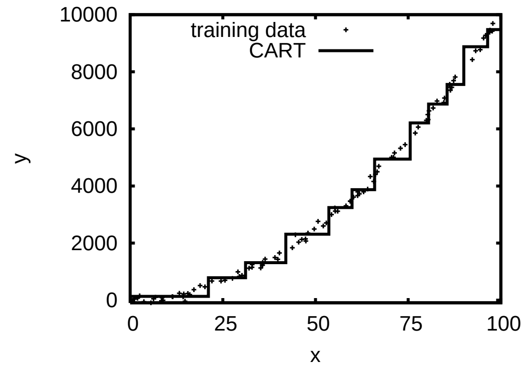
5.2.4 Classification And Regression Trees

Classification And Regression Trees [21] approximate the target function using piece-wise constant functions that can be visualized as binary trees. Figure 5.4(a) shows the CART model constructed on the sample data set. The binary tree divides the input vector space into disjoint regions. The root node represents the entire vector space, and each internal node divides its corresponding region along one dimension into two regions, one for each of the two subtrees. A leaf node holds a value that is the prediction for all the data points that fall into the corresponding leaf node region. The value is estimated using the mean of all the training data points within the region. The disjoint regions degenerate into intervals for one-dimensional data sets. Figure 5.4(b) shows the regression line of the CART model in (a).

The model construction algorithm is a recursive process. It starts with a root node and grows the tree downward by finding the best splitting point in each step. CART



(a) Fitted tree



(b) Data points and regression line

Figure 5.4: CART model for the sample data set.

models could use different splitting criteria to suit applications' needs. For regression, the common criteria is the reduction in mean squared error measured on the training data. Using information gain leads to decision trees, popular for classification.

The splitting can continue until each region has only one training data point in it. In this case, the error on the training data is zero. However, the training error is a poor estimation of testing error, and the testing error increases if the tree grows too deep. Such a situation is called "overfitting". The splitting should stop earlier to avoid overfitting. Please refer to Section 5.3.1 for more information on methods to deal with overfitting.

CART models can model non-linear functions, and are fast in construction and prediction. Further improvements on CART models include Multivariate Adaptive Regression Splines (MARS), which fit a linear model within each leaf to improve the model accuracy.

5.2.5 K-Nearest Neighbors

The k-Nearest Neighbors (kNN) [37] algorithm is different from all the previous models in its internal model representation. The kNN algorithm is "memory-based" because the model simply remembers all the training data during training. All the computation is done during prediction. For a given query point, the algorithm selects the k points from the training data that are the closest to the query point, and the average output of the k data points is the prediction. A more sophisticated approach can use a weighted sum of the k or all the data points, with the weights calculated based on the distances from the query point.

$$\hat{f}(x) = \sum_{i=1}^{k} W(x_i, x) y_i \tag{5.11}$$

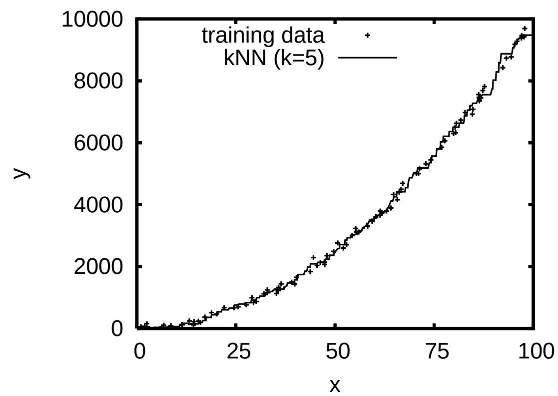


Figure 5.5: A kNN model on the sample data set. The kNN model uses the average of the 5 nearest neighbor as the predictions. The distance function is the Euclidean distance function.

In this case, k can be arbitrarily large because a distant data point will have a small weight and little effect on the prediction.

The distance function and the k are two important parameters that can strongly influence the quality of the predictions. The distance function defines how we quantify the distance between two data points in the d dimensional space. Popular functions include Euclidean, Euclidean squared, City-block, and Chebychev. All the distance functions require that the values along each dimension are normalized so that the unit length has the same influence on the target value. This can be difficult for multi-dimensional data sets. For example, both the request and workload-level descriptions are such data sets, and it is not clear how we should scale these parameters to make them comparable.

Choosing the right value of k is also important in obtaining an accurate model. k should be set to a value large enough to minimize the prediction error and small enough so that the k nearest neighbors are close enough to the predicting data point. The cross-validation algorithm can be used to estimate the value of k. If we select to use the weighted sum, k can be arbitrarily large, but the weight function needs careful adjustment.

Figure 5.5 shows the regression line for the sample data set using a k value of 5. The kNN model is fast in training but computationally intensive in prediction. In addition, the storage requirement is high because the model needs to remember all the training data.

5.2.6 Comparison

We have described five popular regression tools using a sample data set. Table 5.1 compares the multi-regression tools with regard to the features that we are interested in.

Feature	Linear	ANN	SVM	CART	kNN
predictive power	linear	non-linear	non-linear	non-linear	non-linear
computation in prediction	fast	fast	fast	fast	slow
model construction	fast	slow	fair	fast	fast
model compactness	good	good	good	good	poor
parameter tuning	no	difficult	difficult	no	fair

Table 5.1: Comparison of regression tools on desired features for device modeling.

Linear regression models are inappropriate because of the non-linearity of storage device performance. The k-NN algorithm is expensive in prediction and storage. ANN and SVM are powerful enough to model non-linear functions, but parameter tuning and scaling the input vector space could be challenging. CART models offer good accuracy, compact model representation, and efficient predictions, with little parameter tuning. Section 5.4 presents an evaluation of these regression tools in the context of the learning-based device models.

5.3 Model Tuning

Almost all the regression models need some degree of tuning in order to achieve the best prediction accuracy. We discuss how to select the best parameter fit and pre-process data.

5.3.1 Underfitting And Overfitting

Nearly all the regression models have a complexity parameter that has to be determined. For example, the parameter can be the size of a CART, the value of k in kNN, or the number of hidden nodes in a three-layer ANN. Take CART models as an example. A CART can keep growing until each leaf node contains only one training data point. In this case, the prediction error on the training data becomes zero. The training error, however, is a poor estimation of testing error. Figure 5.6 shows the typical curves of the training and testing error as a function of model size or complexity. As a model becomes more and more complex, the training error keeps on decreasing. The testing error, on the other hand, increases when the model becomes overly complex. We need to find the optimal model complexity to avoid both "underfitting" (the shaded area on the left) and "overfitting" (the shaded area on the right).

Cross-validation is a well-established technique for estimating prediction error [47]. The general idea is to divide training data into groups and estimate the error by building and validating models using different groups of the data. Cross-validation helps to determine the optimal tree size for the CART models and the k value for the kNN. In our experiments,

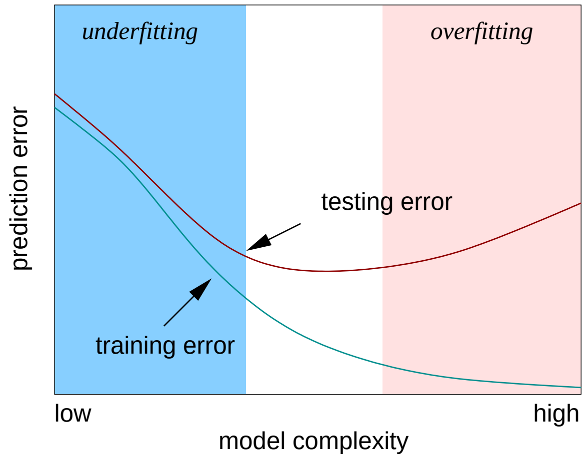


Figure 5.6: Training and testing error as a function over model complexity.

we use two-fold cross-validation to select the optimal CART size. In specific, we use two sets of data, one for training and one for testing, and plot the testing error curve to choose the optimal tree size. For request-level models, a typical model contains 80 to 160 leaves depending on the modeled device. The workload-level models can contain up to 500 leaves. The cross validation on CART can be fully-automated. Other regression tools still need human effort to determine certain parameter values. For example, one needs to decide the network topology of a neural network before she can use cross validation to decide when the weight adjusting should stop under the given topology.

5.3.2 Data Transformation

Some data sets may need transformation before being fed into regression tools. Such transformations can serve three purposes. First, regression tools such as SVM and ANN, work only on data sets of certain ranges. For example, we normalize data sets to have a zero mean and unit standard deviation on all dimensions before building a SVM. An ANN requires that the values on all the dimensions are within range [0,1], so rescaling is needed in pre-processing.

Second, some regression tools may require the unit length on all the dimensions to have the same semantics. For example, the Euclidean distance function in kNN measures the distance between two data points using the weighted sum of the squared distances on all the dimensions, so the dimensions should be scaled so that all the dimensions are comparable. The rescaling could be difficult when the semantic meanings of the dimensions are unclear. For example, it is difficult to determine how the request count should correlate the average seek distance in the workload-level descriptions.

Third, the transformation can help to change the structure of the data. For example, we can take the logarithm of both the input and output of the sample data set to convert the dependence between them into a linear one so that linear regression models can model the data with high accuracy.

$$y = x^2 + \epsilon \text{ becomes} \ln y = 2 \ln x + \epsilon'. \tag{5.12}$$

However, it is usually difficult to detect the structure of the data to select an appropriate transformation, especially for high-dimensional data sets.

A common pre-processing procedure is the logarithm transformation on output value. The transformation is useful for data sets with multiplicative error terms. The transformation converts the multiplicative error term into an additive one, making mathematical analysis easier. In addition, model accuracy will also be improved. Take CART models as example. Real-world traces show long tails in response time distributions: a small number of requests show large response times. Without the logarithm transformation, the splitting focuses on these requests because of their large prediction error. After the transformation, the prediction errors for requests of small response times are magnified, and these requests get more attention from the splitting. Therefore, the model is more accurate in predicting these requests. Because there are more requests of small response times, the overall model accuracy is greatly improved. Figure 5.7 in Section 5.4 compares the standard deviation of the residual error without and with the logarithm transformation in building workload-level models. After the transformation, the increasing trend is removed and model accuracy is greatly improved, too, as shown in Figure 5.8.

Other transformations, such as dimension reduction techniques, are also useful. First, some regression tools, such as linear, ANN, SVM, and kNN, are sensitive to irrelevant input features. The irrelevant parameters can confuse the model construction algorithm, leading to models of poor quality. Removing those irrelevant parameters will improve model accuracy. A common method is stepwise regressions [47], which are essentially a greedy algorithm. The search can happen in two directions. In forward selection, the algorithm starts with an empty set and adds parameters to the set one at time, ordered by the strength of their influence on the output. The best set size is then identified by investigating the prediction error curve. A backward algorithm starts with a full set and removes one parameter at a time. A hybrid one considers both the forward and backward moves in each step.

Second, the dependence between input dimensions can also complicate the model construction algorithm. Techniques such as Principle Component Analysis (PCA) and Independent Component Analysis (ICA), can reduce the dependence by projecting the data onto a different sets of vectors. In PCA, the new vector sets, known as the principle components (PCs,) are the eigenvectors of the training data, and the direction of the PCs see the maximum variances of data in decreasing order. Therefore, the projection of the data on the last several PCs can be removed from the vector with little information loss if the variances in these directions are small. In this case, the dependence between input dimensions is reduced, and so is the dimensionality of the data set.

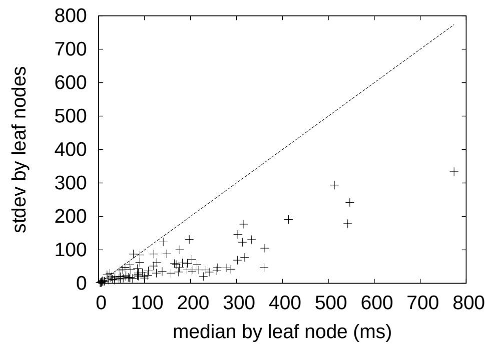
In our device models, we do not consider these advanced techniques because our analysis has shown that most of the error comes from other parts of the system.

5.4 Selecting A Tool for Device Modeling

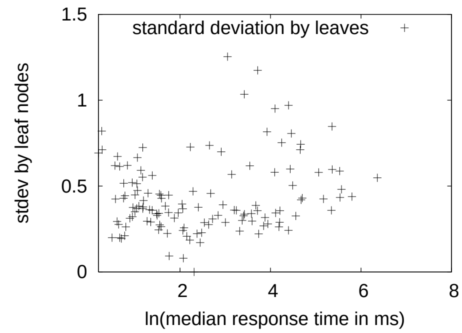
The selected regression tool should be compact, computationally efficient, and accurate. Ease of use can be an additional value. In this section, we compare the afore-mentioned regression tools in constructing the learning-based models for storage devices. We use the regression tools to build workload-level device models for Atlas 10K. Without special notice, the models are constructed on 3,000 minutes from the mixture of the `cello99` traces (Feb. 1-3, 1999.) and tested on one day worth of `cello99` (Feb. 4, 1999.)

Effectiveness of the logarithm transformation. The logarithm transformation removes the dependence between the error term ϵ and the output value Y if the error term is multiplicative on Y. If the random error grows with the target value, the model will perform poorly on data points of small target values. This is exactly the case in performance prediction for storage devices: busy periods usually show a larger variance in response time than idle periods because bursty arrivals make queuing effects worse. Taking the logarithm effectively removes the dependence, and improves model accuracy.

Figure 5.7 shows the effectiveness of the logarithm transformation in turning the multiplicative error term into an additive one. Two graphs plot the standard deviation of the residual error, $|\hat{Y} - Y|$, against Y for the models built on the original and transformed data. Since we do not know the actual function, we generate the graphs in the following way. We build two CART-based models using the same training trace. One model is constructed to predict the median response time, and the other predicts the logarithm of the median response time. A CART model naturally divides training data into groups of similar characteristics and performance, one for each leaf node. We plot the standard deviation of the residual error against the mean target value for all the leaf nodes in the two trees. Figure 5.7(a) is on the CART model without the transformation. Even though we only have limited data points with large target values, the increasing trend is obvious. In (b), the plot

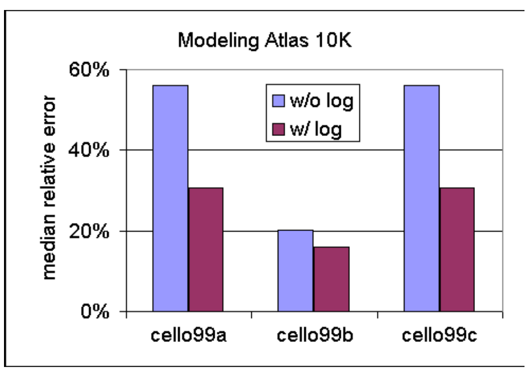


(a) on original data

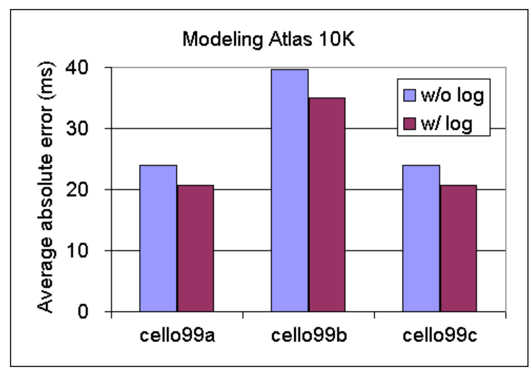


(b) on transformed data

Figure 5.7: Residual error without and with the logarithm transformation measured on the CART models. The standard deviation of the residual error, which shows an increasing trend to the mean of the target value, becomes randomly distributed with the logarithm transformation.

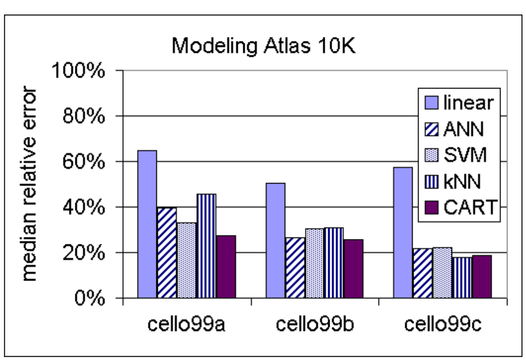


(a) without the transformation

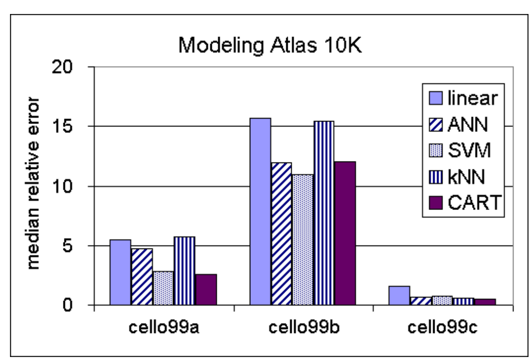


(b) with the transformation

Figure 5.8: Comparison of models constructed without and with the logarithm transformation on the output value.



(a) median relative error



(b) median absolute error

Figure 5.9: Comparison of regression tools.

Feature	Linear	ANN	SVM	kNN	CART
Accuracy	58%	35%	28%	31%	23%
Prediction time	210 ms	138 ms	560 ms	30 sec	300 ms
Storage requirement	60 Bytes	600 Bytes	22KB	180 KB	2 KB
Model construction	270 ms	45 min	90 ms	0	570 ms
Tuning parameters	0	> 2	> 2	> 2	0
Pre-processing	No	Rescaling	Normalization	Normalization	No
Interpretability	Good	Poor	Poor	Poor	Good

Table 5.2: Comparison of regression tools in interesting features. The accuracy is the averaged median relative error of the three `cello99` *traces (Feb 4 - 6, 1999). The logarithm transformation is not considered as pre-processing in the table because we deem it an optimization rather than a necessary step before applying the model.*

is for the CART model built on the data after the logarithm transformation. No significant dependence between the error term and the target value is observed in this case, indicating the effectiveness of the transformation in removing the dependence.

Figure 5.8 compares the accuracy of two workload-level device models. Note that we always recover the predicted values by the model with the logarithm transformation to their original scale. The logarithm transformation brings dramatic improvements for all the traces. The improvement is more prominent in the average errors than the relative errors because the logarithm transformation allows the model to focus on requests of small response times, which incur huge error without the transformation. Therefore, we always use the transformation in all the experiments.

Comparison across regression tools. Figure 5.9 compares the prediction accuracy of the models built using the five regression tools in predicting the median response time over one-minute intervals on Atlas 10K. The linear regression models incur a much larger prediction error than the other tools because of the non-linear device behavior. All the other tools behave similarly in terms of model accuracy although kNN seems to have the largest variance.

Table 5.2 summarizes the comparison. All the tools have similar prediction accuracy except linear regression. In terms of computational and storage requirements, all the models are efficient and compact except kNN, which needs to remember all training data and makes predictions by scanning the training data. CART and linear regression require no manual parameter tuning at all in model construction.* from the In ANN, the number of hidden nodes and the learning rate are the two most important parameters. In SVM, one needs to

*The size of CART models can be automatically chosen by cross validation.

determine the kernel function and the relaxation factor. In kNN, one needs to specify the value of k and the distance function.

We chose to use CART models as our device models because of their competitive accuracy and high efficiency. In addition to these desired features, CART models come with good interpretability: one can easily visualize and understand a CART model, which is useful when diagnosing faults in device models. We have designed several CART-specific techniques for analyzing model error in Chapter 7.

5.5 Understanding the Model

It is useful to determine the importance of workload characteristics on device performance. Such measurements provide a ranking of the parameters for the stepwise regression algorithms and help in understanding the interaction between storage devices and I/O workloads. Such understanding can be useful in designing new storage devices, or redesigning applications to exploit device characteristics to improve performance.

The importance measurements are not easy to obtain, especially when the parameters are correlated with each other, and the dependence between input parameters and output is not linear. Pearson's correlation coefficient measures the degree of linear dependence between two random variables. Linear regression models use Z-score to determine the importance of a parameter. These metrics are inappropriate in our device models because of the non-linear behavior of storage devices. Instead, we use empirical studies to measure the importance of a parameter. Caution is needed in interpreting the results because the results highly depend on the data used in the studies.

Two model-independent approaches exist for measuring the "relative" importance of parameters. These measurements are "relative" because they are intended for comparison only. The *additive* approach builds a model using single parameters and uses the model accuracy as their importance. In particular, we first construct a "total" model using all the parameters, and record its accuracy using its total error reduction on the training trace. We use the mean squared error in our calculation because CART models use this criteria to compare two models. To evaluate the importance of a parameter, we build a model using only that parameter and calculate the model accuracy in the same way. The ratio in accuracy between this model and the "total" model is the relative importance of the parameter. Figure 5.10(a) shows the measurements for the workload-level description parameters using the additive approach on the three devices. The models are constructed using the same training trace as in Section 5.4 to predict median response time. We can clearly see that BurstSize and BurstRate are significantly more important than the other parameters. The results are as expected since the queuing time dominates the response time. Surprisingly, ReqSize alone offers good accuracy, too, suggesting strong correlations between the arrival and request sizes in the training trace. Figure 5.11 plots the first three levels of the CART model built on Atlas 10K. It is not surprising to see these two parameters appear to be so high in the tree hierarchy multiple times. The additive approach can measure the degree of dependence between the evaluated parameter and the target value, but is unable to handle the cases when a parameter combination is needed to make good predictions.

The *subtractive* approach, on the other hand, builds a model using all the parameters except the one being evaluated. Intuitively, missing an important parameters leads to an inferior model, so we can use the extra model error as the importance of the parameter. The subtractive approach can not correctly measure the importance of strongly-correlated parameters because excluding any one of them will not degrade the model much. Unfortunately, the workload-level description parameters are strongly correlated, and the model accuracy stays almost the same when we omit any parameter as shown in Figure 5.10(b). One can still conclude that BurstSize and BurstRate are more important than the others.

The additive and subtractive approaches are applicable to any regression tool. Model-specific methods exist, too. For example, the Z-score is such a metric in linear regression. We introduce another CART-specific approach here. Each split in a CART tree reduces model error by a certain amount. One can summarize the contribution of a parameter in error reduction for a given tree and use the sum as the importance of the parameter. Because all the parameters are used in constructing the model, the "overshadowing" problem in the subtractive approach still exists. For example, if two parameters are correlated, the model may pick only one in constructing the tree. Applying the above approach will show that the unpicked one is not important even though both should be. Ideally, the dependence between parameters should be removed before the application of regression tools to maximize performance. Our workload-level descriptions exhibit strong dependence, and future work could explore the effectiveness of PCA and ICA in reducing the dependence between parameters. Figure 5.10(c) shows the importance measured by this approach. As BurstSize and BurstRate are both shown to be of similar importance by the additive approach, BurstRate seems to be overshadowed, so is ReqSize.

We have used the CART-specific approach in Chapter 4 to measure the importance of the parameters in request and workload-level descriptions because of its effectiveness and efficiency. Of course, one needs to be cautious in interpreting the results because of the "overshadowing" problem.

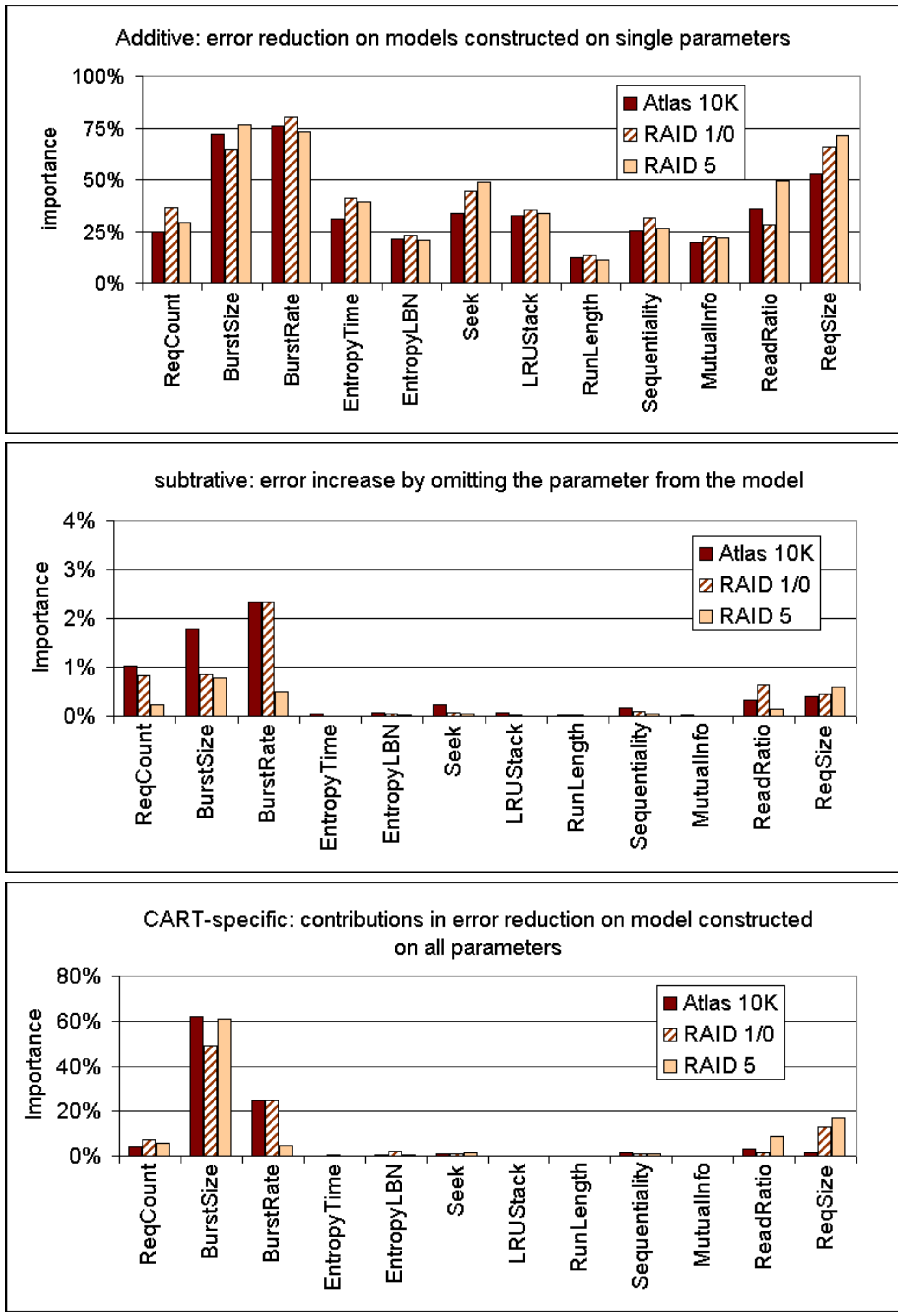


Figure 5.10: Comparison of three ways of measuring the parameter importance. The training traces are the mixture of `cello99a`*,* `cello99b`*, and* `cello99c`*. The model predicts the median response time over 1-minute intervals on the three devices.*

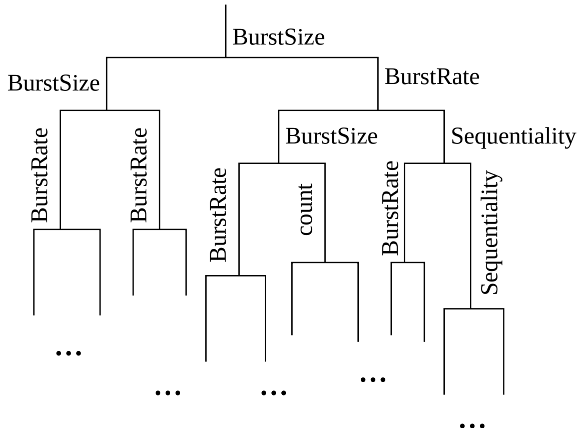


Figure 5.11: The first three levels of a CART model constructed for Atlas 10K using 3,000 minutes of a mixture of `cello99`*. BurstSize and BurstRate appear high and frequently here, suggesting their importance in determining workload performance.*

5.6 Summary

This chapter uses empirical studies to select an appropriate regression tool for our learning-based models. The regression tool takes request or workload-level descriptions as input to predict device performance. The desired features for the chosen tool include accurate and fast predictions, a compact model representation, and an automated model construction algorithm.

We briefly describe a set of popular regression tools and use real-world traces to compare them. The experiments have shown CART models offer competitive prediction accuracy and are compact, efficient, and easy to use. Therefore, we select CART models as the basis of the learning-based models for storage devices.

Chapter 6

Training Workload Generation

The learning-based modeling approach learns a storage device's behavior by observing it under a set of training traces. The training traces should be diverse enough to cover a large range of workloads for device models to predict the performance of an I/O workload accurately because the models can only predict workloads that are similar to the training traces. Moreover, the training traces should be as short as possible to have a short training time. This chapter discusses our design of the training traces.

We elected to use synthetic traces in training because of their low storage overhead and high controllability. The process consists of two steps: a) producing a small set of workload-level descriptions that offers a broad coverage over the workload-level description space using effective sampling techniques (presented in Section 6.3) and b) generating traces for the sample workload-level descriptions (presented in Section 6.4.) Section 6.5 evaluates the effectiveness of these techniques and concludes that synthetic trace generators need further improvement to deliver accurate device models.

6.1 Design Considerations

The quality of training traces determines the accuracy of our learning-based models. This section explores the design space of the training traces.

6.1.1 Design Alternatives

The learning-based models construct themselves by observing how storage devices handle training traces. One option is to use real-world traces in training. We selected to use synthetic traces instead for three reasons. First, collecting real-world traces could require human effort to build the infrastructure. Second, the collected traces may offer only a limited coverage, and there is limited work on how to alter the characteristics of existing traces. Finally, publishing real traces invokes data privacy issues.

The advantage of synthetic traces is that one can control the characteristics of the synthetic traces to ensure a maximum coverage over the workload-level description space. Moreover, little storage space is required since the traces can be generated on the fly. On the other hand, replicating realistic correlation structures in synthetic traces can be difficult, as is knowing whether and which such correlations matter.

6.1.2 Diverse Synthetic Traces by Sampling

To ensure the coverage, we generate traces from a set of workload-level descriptions drawn from the workload-level description space using sampling techniques. In this case, the training trace generation involves three steps.

1. **Step 1:** Determine the region in the workload-level description space that needs sampling.

2. **Step 2:** Sample over the region to produce a set of workload-level descriptions that offers a good coverage.

3. **Step 3:** Generate traces from the sample workload-level descriptions.

Sampling ensures the region is covered efficiently by a small number of workload-level descriptions. With realistic synthetic trace generators, the final traces will meet both the coverage and length requirements.

6.1.3 Difference from Traditional Synthetic Workload Generation

Synthetic workload generation has always been an active area because of the efficiency and convenience offered by synthetic workloads in system evaluation. It is, however, difficult for our system to employ previous work in workload generation because of the subtle difference in the problem setting.

First, we have a different goal from traditional trace generation. Existing work usually focused on generating traces with given characteristics, and the characteristics are usually specified in the form of a sample trace. So the target is to produce traces of similar characteristics, hence similar performance to the given trace. Consequently, their focus is on extracting important characteristics from workloads and faithfully reproducing them in synthetic traces. In contrast, we emphasize producing traces of a variety of arrival and access patterns. Consequently, we are also responsible for producing reasonable parameter combinations in addition to the "real" trace generation.

Second, our generators may receive an impossible parameter combination as input. The sampling step generates workload-level descriptions, with little knowledge of the dependence between the workload-level description parameters. As a result, some samples can contain contradictory parameter values. For example, it is impossible for a workload to have an average run length of 10 when its sequentiality is 0%. In traditional trace generators, the parameters are extracted from real workloads, and such contradiction will not happen.

Third, our generators are limited to use only scalar parameters in specifying workload characteristics because our regression tools only receive scalar vectors as input. We could potentially add the first, second, and even third moments of a distribution to the workload-level descriptions, but the generators still need to make assumptions to generate the entire distribution in generating synthetic traces. Traditional trace generators, on the other hand, can use both scalars and distributions to characterize a given workload, minimizing the number of assumptions they need to make. For example, the access frequencies of disk blocks are commonly used to characterize the locality pattern.

Fourth, traditional trace generators can choose to use parameters that are convenient in their models to specify workloads. Our generators, on the other hand, have a fixed parameter set, that is, the workload-level description, and no existing generators use the same parameters as we do.

In summary, trace generation in our system imposes different challenges than the traditional one because of the difference in the problem setting. We present the three steps in our trace generation next.

6.2 Bounding the Workload Description Space

Before sampling, we need to determine the region in the workload-level description space that needs samples. This region should effectively cover all the possible workloads that can run on the modeled device without saturation. If the region is too large, we could waste time in producing sample traces that would never be able to run on the device.

We currently take the bounding box of all traces as the sampling region. Even though we have observed saturation on some sample traces, it does not necessarily mean that the bounding box covers all the possible workloads that can run on the device. Figure 6.1 shows such a scenario. Even with a bounding box that misses workloads, we could still experience saturations from traces within the bounding box. For the purpose of our experiments, the bounding box of all the traces should be enough to cover all the workloads tested.

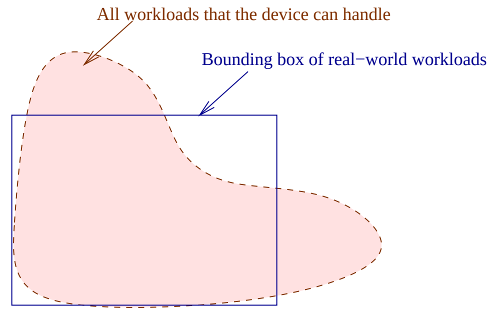


Figure 6.1: Bounding box in the workload-level description space.

6.3 Sampling Over the Workload Description Space

Sampling is responsible for generating a small set of workload-level descriptions to cover the sampling region. The simplest one is uniform sampling, which generates samples using a uniform distribution over the entire region. Uniform sampling is straightforward to implement, but uses the samples inefficiently. The sampling rate stays the same for all the regions of the input vector space, regardless of the variance of the target value, that is, the aggregate performance in performance modeling of storage devices in our problem setting. Consequently, one may need a large number of samples to ensure high fidelity for the models to capture device behavior over high-variance regions.

Ideally, the sampling rate should be biased, with the rate adapting to the variance of the target value. It is reasonable to devote more samples to regions of high variance. This allows a more efficient use of the samples: the same number of samples can provide more information about the target function in biased sampling than in uniform sampling. Therefore, training can be shorter.

Implementing biased sampling, however, requires the knowledge of the target function for estimating the variance. In a static approach, one can use expert knowledge to predetermine the sample distribution over the workload-level description space. For example, traces of high traffic volume can have a larger variance in workload performance and therefore, the models need more sample traces of high volume to capture device behavior under heavy loads. The sample rate could grow with the arrival rate. The static biased sampling is easy to implement, but requires domain-specific knowledge to design the sample distribution.

A more sophisticated approach is a dynamic one, which estimates the variance of the target value from observed behavior and adjusts the sampling rate accordingly without any prior assumptions on the target function. This approach is also known as adaptive sampling, and is widely used in statistical experiment designs [108]. Adaptive sampling requires an iterative model construction algorithm in which more training traces are continuously added to improve model accuracy.

We used uniform sampling in our experiments because error analysis has shown the major error source is the synthetic trace generators. Therefore, immediate efforts should be directed to improve the quality of trace generators.

None of the sampling techniques described above takes into consideration the dependence between the workload-level description parameters. They draw parameter values independently. It is difficult to enforce dependence in sampling because we lack a comprehensive understanding of the dependence. Ignoring the dependence leads to impossible workload-level descriptions. We incorporated the following two simple rules in sampling to avoid such contradictions, and leave the rest to the trace generators.

1. When the sequentiality is 0, the average run length will be set to 0. Otherwise, it will be set to a value larger than 2. A sequential run has at least two requests as defined by our feature extraction algorithm.

2. The joint entropy plot slope should be smaller than the sum of the entropy plot slopes on arrival and LBNs to have positive mutual information.

We enforce the rules by simply discarding any samples that do not meet the requirements. This does not affect the coverage of the samples because such workloads should never exist in practice. Such early pruning is beneficial in reducing the unnecessary work in generating traces for these samples.

6.4 Generating Traces From Samples

Once the sample workload-level descriptions are available, the next step is to generate traces with the specified characteristics. The synthetic traces should exhibit the given characteristics and, at the same time, provide similar performance on the modeled device to the real traces with similar characteristics. This is difficult to achieve for three reasons.

- A workload-level description contains correlated parameters. It is difficult to develop a unified trace generator to accept the workload-level description as a whole and produce traces with all matching characteristics.

- One needs to make assumptions to generate traces. Ideally, a workload-level description should capture all the important workload characteristics so that any additional assumptions will have little impact on the performance of synthetic workloads. Our workload-level description, however, is not yet ideal.

- Even if two traces have the same characteristics, they could perform differently because of the intrinsic variance in device behavior. The variance can come from the initial system status and unobservable variables not captured by our workload-level descriptions. The variance tends to grow with the load because the strong correlations between requests make the queueing effects more prominent.

We choose to maintain a set of generators to meet both the coverage and faithfulness requirements. The next section presents the design in detail.

6.4.1 Generator List

In our trace generation, a generator usually takes a subset of the workload-level descriptions as input, and is responsible for generating one attribute value independently from the other attributes. Table 6.1 lists all the generators in the system. For example, the B-model can generate request arrival times independently from their location, request type, and size. As a result, to generate a synthetic trace for a given workload-level description, we pick a generator for each attribute and use the combination to generate synthetic traces. An exception is the PQRS model, which can generate request arrival time and location simultaneously. As a result, (Burst, LBNDiff, Markov, Exponential) is a valid combination, and so is (PQRS, PQRS, Markov, Exponential). The synthetic traces will match the characteristics specified by all the parameters used in the generator combination, but not necessarily match the others. Since there is no unified model to take all the parameters as input, combining various generators is the best effort we can offer with this approach to meet specified workload-level descriptions. In addition, this approach naturally resolves the contradicting samples because it reduces the dependence between the input parameter set of each combination to a minimum.

The next sections describe the two new generators, B-model and PQRS.

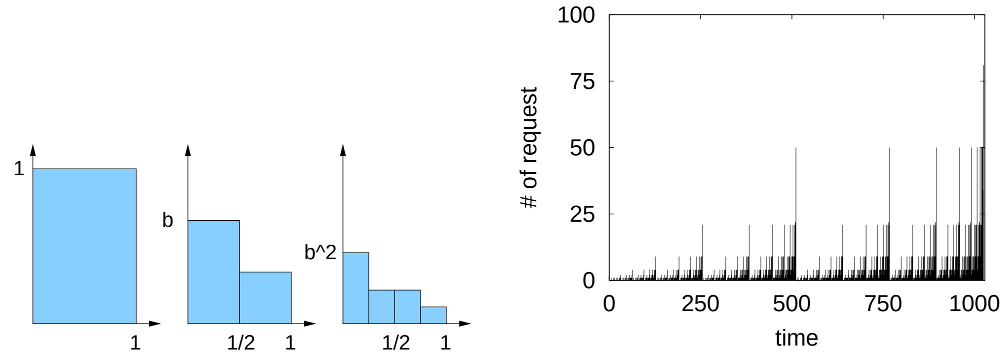
6.4.2 B-model and PQRS Model

The B-model and PQRS models are designed to take advantage of the entropy plot to generate synthetic traces. The B-model can generate either the arrival process or the access frequency distribution of disk blocks given an entropy plot slope. The PQRS model extends the B-model to simultaneously generate the arrival time and LBN of a disk trace with given entropy plot slopes. Please refer to [115, 117] for further details on the two models.

B-model. The B-model takes a parameter, bias b, as input and generates a one-dimensional probability distribution function. The bias controls the burstiness of the distribution func-

Attribute	Generator	Description	Parameters
Arrival time	Gamma	Use the gamma distribution to generate the interarrival time of each request independently	ReqCount (equivalent to arrival rate)
	Burst	Generate bursts of requests, with the burst size and interarrival time within bursts following the exponential distribution	BurstSize, BurstRate
	B-model	Generate arrivals with the B-model [117]	EntropyTime
	PQRS model	Use the PQRS model [115] to generate correlated arrival time and location	ReqCount, EntropyTime
LBN	Sequential	Generate sequential runs of exponentially-distributed size	RunLength, Sequentiality
	LBNDiff	Assume an exponential distribution of the seek distance and generate the location of next request by adding the seek distance to the current location	Seek
	Reuse	Assume an exponential distribution of the LRU distance for the current request	ReuseReq
	B-model	Use B-model to generate the access frequency of disk blocks and draw location from the distribution	EntropyLBN
	PQRS model	Use the PQRS model to generate correlated arrival time and location	EntropyLBN, MutualInfo
Type	Ratio	Generate the request type independently	ReadRatio
	Markov	Model the type transition of previous request to current request using Markov model	ReadRatio
Size	Exponential	Draw request size from exponential distribution	ReqSize

Table 6.1: Generators maintained in the system. The sequential generator is incorporated in all the LBN generators: when generating the LBN for a new request, the algorithm will check the current percentage of sequential accesses. If more sequential requests are needed, the request will become the starting request of a sequential run, whose run size is drawn from an exponential distribution with the given mean.



(a) construction algorithm (b) a sample B-model trace

Figure 6.2: The recursive construction algorithm of B-model. (b) shows a sample B-model trace generated with $b = 0.7$. The sample trace has linear entropy plot of slope 0.881.

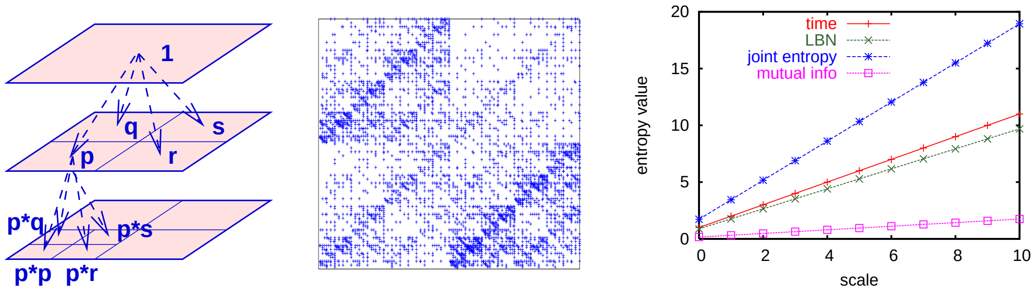
tion, and is usually estimated from the entropy plot slope from a real trace. Figure 6.2(a) illustrates the recursive generation process. The recursive split stops when the desired granularity is met. Once the distribution is available, the model can distribute data points along the dimension using the distribution. For example, we can apply the B-model to the arrival time to generate the arrival process of an I/O trace. Applying the model to LBN produces the access frequency distribution of the disk blocks.

B-model traces have linear entropy plots. The burstiness stays the same at all the granularities because the same parameter b is used in each recursive split. The slope of the entropy plot depends on the value of b.

$$E = -b \log_2 b - (1 - b) \log_2 (1 - b) \tag{6.1}$$

The B-model trace in Figure 6.2(b) has a linear entropy plot of slope 0.881, which corresponds to the bias value of 0.7 used in generating the trace.

PQRS model. The PQRS model is a natural extension of the B-model to handle two-dimensional data sets. The model takes four parameters, $p + q + r + s = 1$, as input, and generates traffic of given burstiness and locality. Figure 6.3(a) illustrates the recursive cascading algorithm to generate a PQRS trace. The synthetic traces have linear entropy plots because the same parameter values are used in all the steps. The slope of the entropy



(a) trace generation (b) a sample trace (c) entropy plot

Figure 6.3: The recursive construction algorithm of the PQRS model. (b) shows a sample PQRS model trace with $(p, q, r, s) = (0.2, 0.3, 0.1, 0.4)$. *The sample trace has linear entropy plots with slopes of* $1, 0.881, 1.722, 0.159$.

plot is controlled by the p, q, r, s values.

$$\begin{aligned} E_{time} &= -(p+q)\log_2(p+q) - (r+s)\log_2(r+s) \\ E_{LBN} &= -(p+r)\log_2(p+r) - (q+s)\log_2(q+s) \\ E_{joint} &= -p\log_2 p - q\log_2 q - r\log_2 r - s\log_2 s \end{aligned} \tag{6.2}$$

Figure 6.3(b) shows the trace generated from the PQRS model with parameter values of (0.2,0.3,0.4,0.1), and (c) shows the entropy plot for the synthetic PQRS trace. The PQRS trace has linear entropy plots, and the slopes match the values given in Equation 6.2. The trace looks different from the real-world trace shown in Figure 4.2 because of the different entropy plot slopes in the two traces.

The beauty of the B-model and PQRS model lies in their simplicity and flexibility. A small set of scalar parameters controls the characteristics of entire traces, and the models can simulate traffic of different burstiness and locality.

6.4.3 Mixing versus Bagging

Once we have the synthetic traces generated by different generator combinations, there are two ways of using these traces in training. The “mixing” approach constructs a single model based on all the traces. The other option, “bagging”, builds separate models, one for each generator combination, and uses the bag of the models to determine the final prediction [20]. In this approach, each model produces a prediction and its “confidence” in the prediction, and the prediction with the highest confidence becomes the final answer. Generally, the confidence should grow with the number of the samples used in generating the prediction, and decrease with the standard deviation of the samples. The following are two simple ways of calculating the confidence in CART.

- **count.** Use the number of samples in generating the prediction as the confidence measure.
- **stdev.** Use the reciprocal of the standard deviation of the samples generating the prediction as the confidence measure.

In CART models, both can be easily computed by looking at the training data that fall into the same leaf node as the predicted data. Bagging can work with online model adaptation easily. A new model based on new training data to improve can be added to participate in prediction. The mixing approach, however, requires remembering all the existing training data before one can add new training data to construct a better model.

6.5 Experimental Results

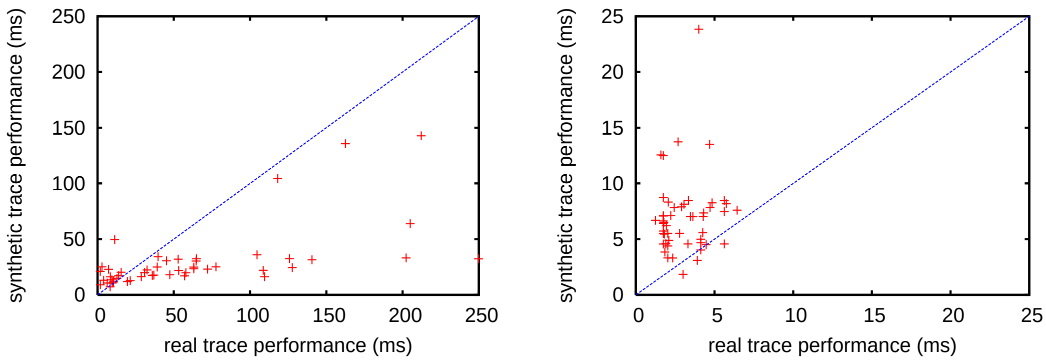
We study the effectiveness of sampling and our trace generators in generating useful training traces for our learning-based models in this section.

6.5.1 Effectiveness of Trace Generators

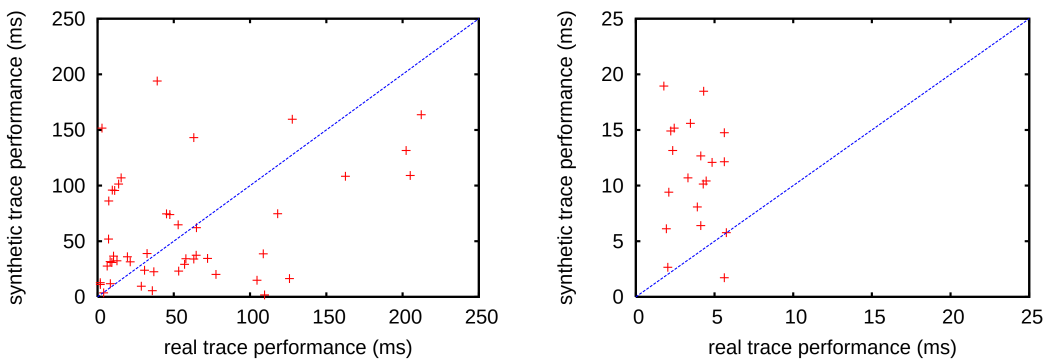
The trace generators should produce traces that are similar to real traces in both workload characteristics and performance. In this set of experiments, we get workload-level descriptions from real traces and use our generators to generate synthetic traces for these workload-level descriptions. Ideally, the synthetic traces should have the same workload-level descriptions and performance as the real ones.

We randomly select 150 one-minute fragments from the three `cello99` traces, 50 from each. For each fragment, we use our generators to generate 30 fragments and compare their performance against the real traces. Though these data points may not offer a good coverage of the workload-level description space, they offer diverse access patterns as we can see from the different distributions of the entropy plots in Figure 4.15,

Figure 6.4 compares the performance of the real and synthetic traces. We calculate the average of the median response time for each group of 30 fragments and plot the value against the actual median of the real traces. Ideally, the data points should lie close to the diagonal line. We observe that, in Figure 6.4(a), Burst generates traces with smaller response times than the real traces because the synthetic traces are less bursty (the graph on the left). On the other hand, when the traffic is smooth and light, LBNDiff produces traces of poorer locality than real ones, leading to larger response times (the graph on



(a) (Burst, LBNDiff, Markov, Exponential) combination



(b) (PQRS, PQRS, Markov, Exponential) combination

Figure 6.4: Comparison of synthetic and real traces in their median response times on `cello99b` *and* `cello99c` *(right.)* `cello99b` *(left) is of higher volume than* `cello99c`*. Each data point is the average of 30 synthetic trace fragments generated from the workload-level description of a real trace fragment.*

the right). The performance difference is due to the ignorance of the strong dependence between requests. For example, in both Burst and LBNDiff, the attribute values of each current request are generated independently from the other attributes and from the previous requests. In reality, strong correlations exist, so requests tend to arrive in batches and are likely to clustered along the LBN space. Ignoring the correlations leads to smoothy traffic of poor locality.

Figure 6.4(b) compares another generator combination. The PQRS model generates traces of stronger burstiness in arrivals because of huge spike in the arrival process leads to a long queue. The total number of disk blocks accessed by a PQRS trace is small, and the locality is better.

In summary, it is difficult to generate realistic traces from workload-level descriptions using our method. The provided generators are inaccurate in reproducing the character-

index	Arrival time	LBN	Type	Size
1	Burst	LBNDiff	Markov	Exponential
2	Burst	Reuse	Markov	Exponential
3	Burst	B-model	Markov	Exponential
4	Burst	Zipf	Markov	Exponential
5	Bmodel	LBNDiff	Markov	Exponential
6	Bmodel	Reuse	Markov	Exponential
7	PQRS	PQRS	Markov	Exponential
8	Gamma	LBNDiff	Markov	Exponential
9	Gamma	Reuse	Markov	Exponential

Table 6.2: Trace generator combinations used in the experiments.

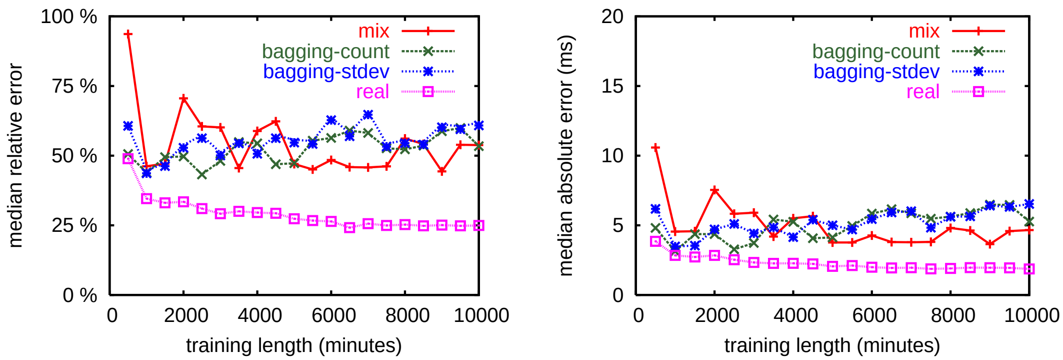
istics in real traces for synthetic traces to have the same performance as real ones. The unfaithfulness is due to the ignorance of the correlations between requests.

6.5.2 Comparison of Mixing and Bagging

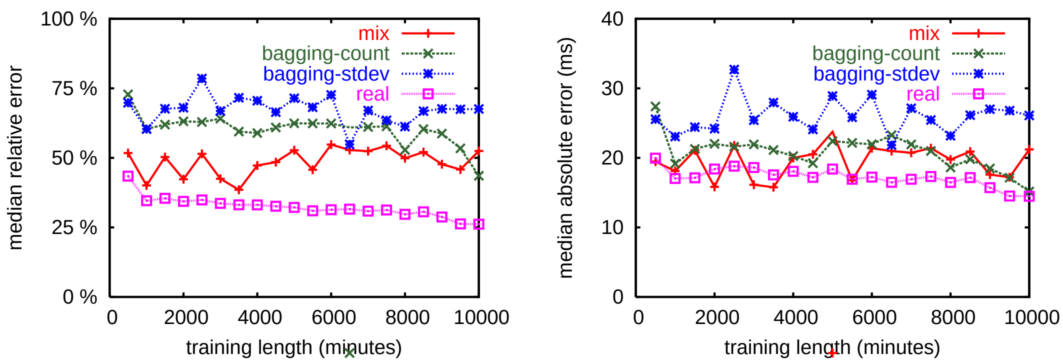
As presented in Section 6.4.3, we can use Mixing or Bagging to build device models from synthetic traces. We compare the two approaches in this set of experiments. Both `count` and `stdev` are used in bagging to select the predictions. We use the 9 generator combinations listed in Table 6.2 to generate synthetic traces. All the combinations use the same generators for request type and size because we discover that if we replace the values of these two attributes in the original traces with the new values generated from these two generators, the change in performance is negligible. We bound the workload-level description space by taking the bounding box of all the `cello99` traces, and produce 3,000 sample points using a uniform sampling. More samples do offer any improvement.

Figure 6.5 shows the comparison results. The models are built to predict the median response time on Atlas 10K. The main observation is that there is little distinction between Mixing and Bagging. In both cases, model error is more than doubled than if we build the model using real traces. The additional error is due to the poor quality of the training traces as we have shown previously. Even though the synthetic traces are generated from the given workload-level descriptions, unrealistic assumptions, such as the ignorance of the long range dependence between requests, lead to unrealistic synthetic traces. The next chapter systematically analyzes model error and confirms this observation.

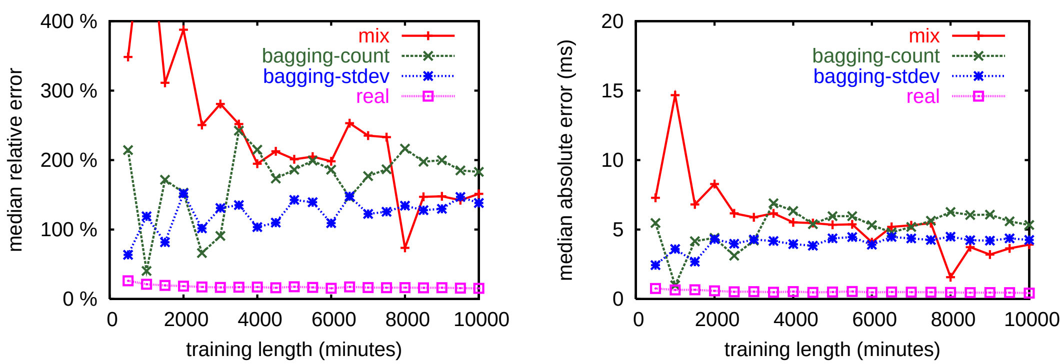
We have evaluated two algorithms, `count` and `stdev` as presented in Section 6.4.3, in selecting predictions in Bagging. Figure 6.6 studies the frequency and accuracy of individual models used in `stdev`. Figure 6.6(a) shows the percentage of predictions used as the final prediction for each generator combination, and Figure 6.6(b) shows the prediction error



(a) tested on `cello99a`

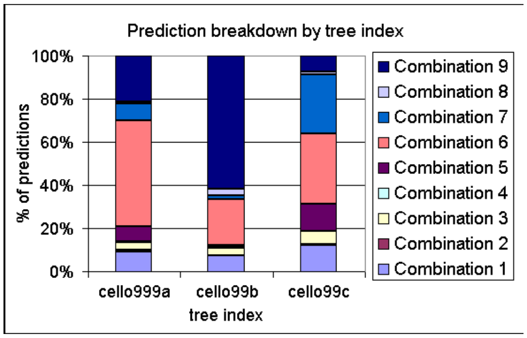


(b) tested on `cello99b`

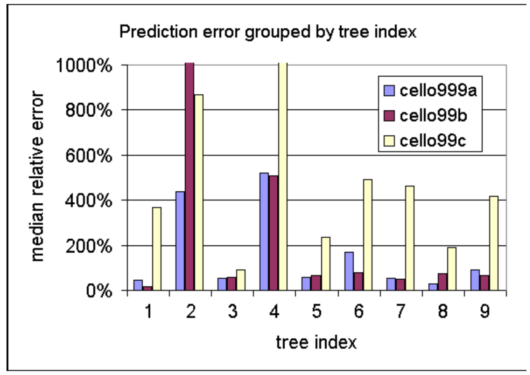


(c) tested on `cello99c`

Figure 6.5: Comparison of Mixing and Bagging in constructing device models for Atlas 10K.



(a) count breakdown



(b) error aggregated by combinations

Figure 6.6: Model error analysis by combinations from Table 6.2.

grouped by the combinations. We observe that combination 2 and 4 do not contribute much to final predictions, and when they do, their predictions are far from accurate. We should exclude the traces produced by those two models in our voting algorithm.

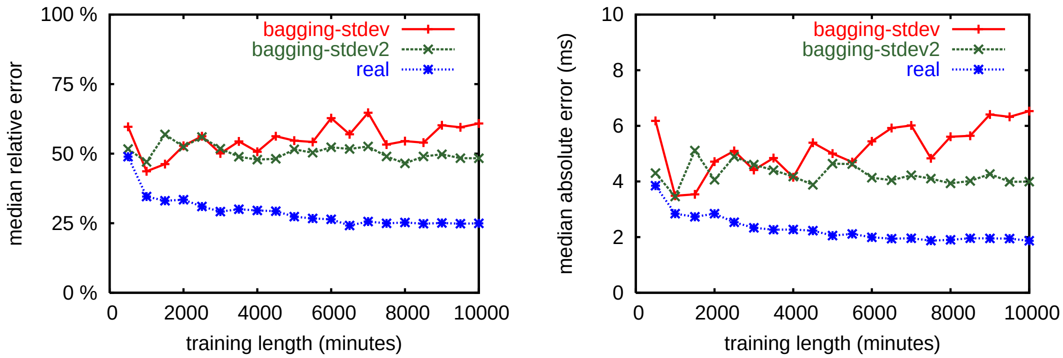
Figure 6.7 shows the model error of Bagging after combination 2 and 4 are removed from the training set. We observe a slight improvement in the model accuracy, but the overall model error is still large compared to the ones built on the real traces. This result is expected because the prediction accuracy on other combinations are far from ideal as shown in Figure 6.6(b). This experiment has indicated that high-quality trace generators are needed to deliver accurate models.

In summary, our trace generators are too naive to produce realistic traces for constructing the learning-based models of storage devices. Our experiments have compared Mixing and Bagging, but because of the poor quality of our trace generators, it is difficult to make concrete conclusions based on inaccurate models. Future work should investigate the difference between real and synthetic traces with higher quality trace generators.

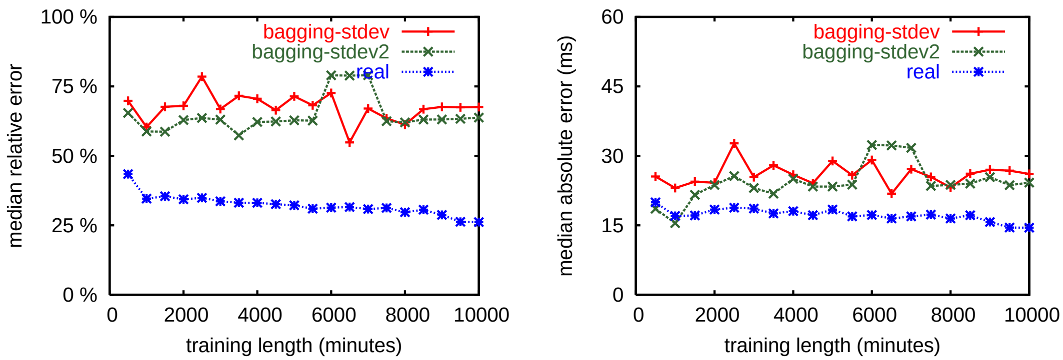
6.6 Summary

The quality of the learning-based models depends highly on the training traces used in constructing the models. The traces should cover as many access patterns as possible for the constructed models to predict I/O workloads, but the training traces should be as short as possible to speed up trace replay. This chapter has explored the design space of using synthetic traces in training.

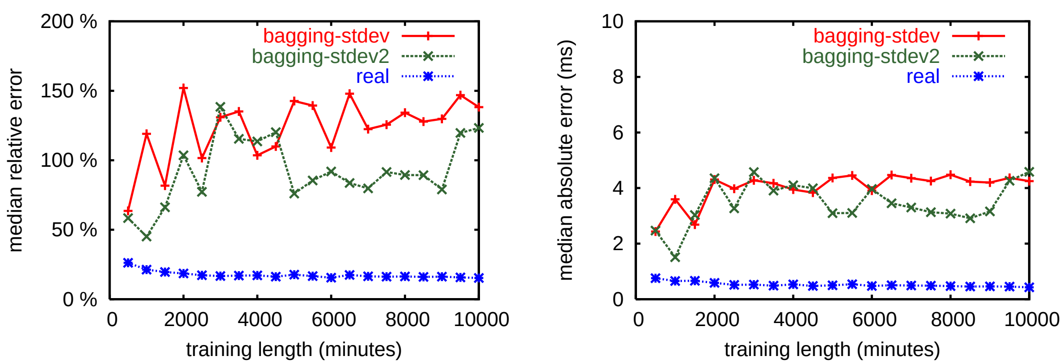
The trace generation process involves sampling over the workload-level description space and generating synthetic traces from the sample workload-level descriptions. The goal of



(a) tested on `cello99a`



(b) tested on `cello99b`



(c) tested on `cello99c`

Figure 6.7: Effects of removing combination 2 and 4 in the training set. `voting-stdev` *and* `voting-stdev2` *are the models constructed before and after removing combination 2 and 4 from the train set respectively.*

sampling is to generate a small number of sample workload-level descriptions to efficiently cover the interesting region in the workload-level description space, and trace generation produces synthetic traces from the sample workload-level descriptions. We have evaluated uniform sampling and several generators in generating training traces for our learning-based models and discovered that the synthetic trace generators are not effective enough to produce traces that are similar to real traces mostly. The unfaithfulness is due to the unrealistic assumptions of independence between requests. The next chapter presents a detailed analysis of model error to determine the major error source.

Chapter 7

Error Analysis

The previous chapters have described the model construction algorithm of our learning-based approach. The experiments have shown that the device models are not 100% accurate. It is important to understand where these errors come from to further improve the quality of the device models. This chapter provides a systematic analysis of the model error based on empirical studies.

Section 7.1 discusses the error sources and provides the breakdown of the model error. Section 7.2 provides a set of tools to study the coverage of synthetic traces and to compare the performance between synthetic and real traces. We also investigate the possibility of online model adaptation in Section 7.3.

7.1 Error Breakdown

Each step of the model construction algorithm introduces errors to the device models, in addition to the intrinsic variance in device behavior. This section studies the composition of the model error and introduces an approach to break down the model error based on empirical studies.

7.1.1 Error Sources

Figure 7.1 shows the errors introduced in model construction. We divide the model error into four parts, depending on the sources.

$$\text{Error} = \text{Error}_{\text{coverage}} + \text{Error}_{\text{generation}} + \text{Error}_{\text{vector}} + \text{Error}_{\text{random}}. \tag{7.1}$$

1. $\text{Error}_{\text{coverage}}$ refers to the error introduced by the insufficient coverage from sampling

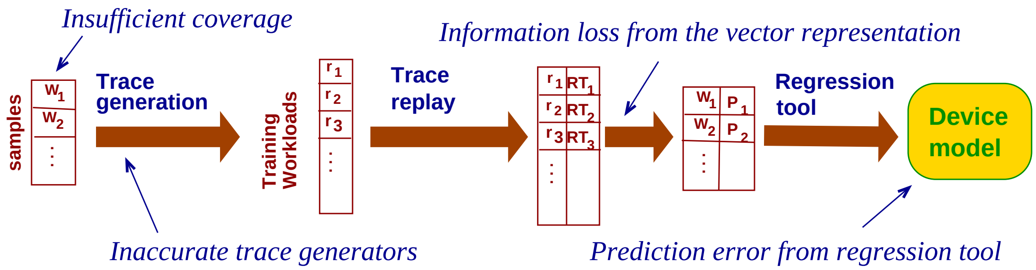


Figure 7.1: Error sources the learning-based modeling approach.

the input workload-level description space. The device models learn device behavior by observing storage devices under a set of sample traces. The device models can not predict traces that exhibit access patterns not seen in training. Sufficient coverage is mandatory to build an accurate device model that can predict an interesting workload.

2. $\text{Error}_{\text{generation}}$ refers to the error introduced by inaccurate trace generators. The device models base their predictions on the performance of training traces. The predictions will be off if the synthetic and real-world traces are of similar characteristics but of different performance.

3. $\text{Error}_{\text{vector}}$ is the error caused by the information loss from the internal workload-level representation. The vector representation is a compression of actual I/O workloads. Missing important workload characteristics makes it difficult for the models to distinguish workloads of different performance on the target device.

4. $\text{Error}_{\text{random}}$ describes all the other errors that are not covered by the above three terms. These errors include the intrinsic variance of the modeled devices and the prediction error of the regression tool.

Breaking down model error by error sources is helpful in directing future efforts to improve model quality. Analytical analysis is difficult to achieve because of the complex interaction between workloads and devices. Otherwise, we could easily build analytic disk models. Instead, we choose to use empirical studies to achieve the breakdown. One needs to be cautious in interpreting the results because they depend on the storage devices and the workloads used in the evaluation.

We use the following three types of models as a vehicle to understand the partial sums of the four error terms in Equation 7.1.

- The `real` models use traces that are similar to the ones being predicted in training. The sampling and synthetic trace generation steps are eliminated from the construction algorithm. Therefore, the model error of the `real` models, E^{real}, approximates the sum of $\text{Error}_{\text{random}}$ and $\text{Error}_{\text{vector}}$.

- The `replicate` models are built from synthetic traces generated from the workload-level descriptions of traces in evaluation. Such device models simulate the scenario of an ideal set of samples on the workload-level description space for the testing traces and experience minimal sampling error. As a result, the model error of `replicate`, $E^{\text{replicate}}$, can approximate the sum of $\text{Error}_{\text{generation}}$, $\text{Error}_{\text{vector}}$, and $\text{Error}_{\text{random}}$.

- The `uniform` models are constructed on synthetic traces generated from a uniform sampling over the workload-level description space. This is the model constructed with no information from the testing traces. The model error, $\text{Error}^{\text{uniform}}$, is the sum of all four error terms.

Once the above three types of models are available, one can use the difference between these models to estimate the error terms in Equation 7.1 as follows.

$$\begin{cases} \text{Error}^{\text{real}} = \text{Error}_{\text{random}} + \text{Error}_{\text{vector}} \\ \text{Error}^{\text{replicate}} = \text{Error}_{\text{random}} + \text{Error}_{\text{vector}} + \text{Error}_{\text{generation}} \\ \text{Error}^{\text{uniform}} = \text{Error}_{\text{random}} + \text{Error}_{\text{vector}} + \text{Error}_{\text{generation}} + \text{Error}_{\text{coverage}} \end{cases} \tag{7.2}$$

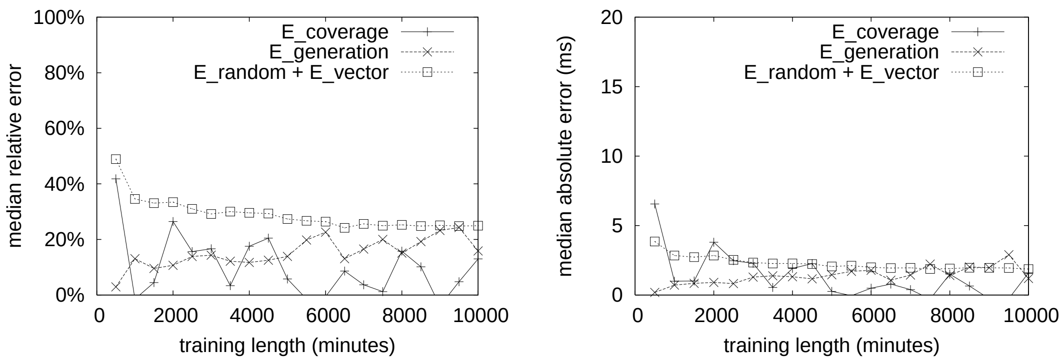
is equal to

$$\begin{cases} \text{Error}_{\text{random}} + \text{Error}_{\text{vector}} = \text{Error}^{\text{real}} \\ \text{Error}_{\text{generation}} = \text{Error}^{\text{replicate}} - \text{Error}^{\text{real}} \\ \text{Error}_{\text{coverage}} = \text{Error}^{\text{uniform}} - \text{Error}^{\text{replicate}} \end{cases} \tag{7.3}$$

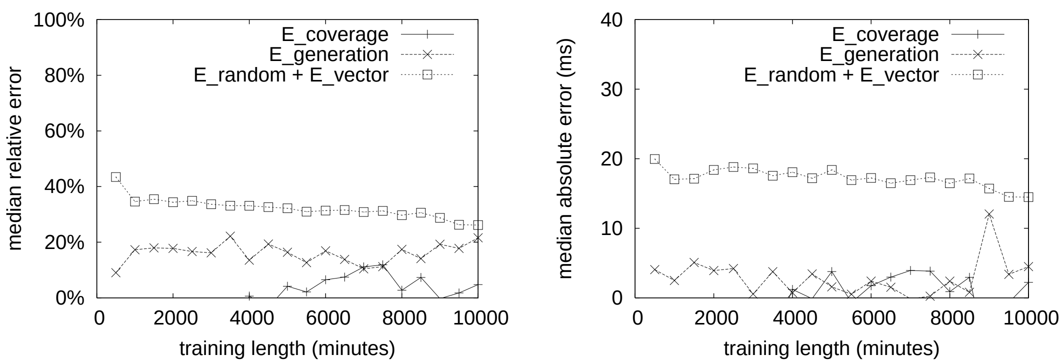
The above approach allows us to break model error down into three parts. It is, however, difficult to further distinguish $\text{Error}_{\text{random}}$ from $\text{Error}_{\text{vector}}$ because we lack full knowledge of the device behavior of the modeled devices.

7.1.2 Experimental Results

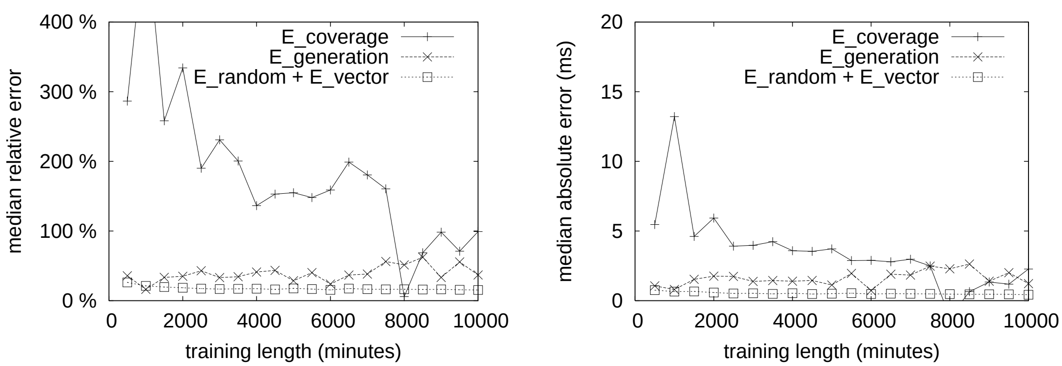
Figure 7.2 uses the above methodology to break down the errors in modeling the median response time on Atlas 10K. We use the mixture of the three `cello99` traces (February 1-3, 1999) in training and test the models on the activities from `cello99` (Feb 4-6, 1999). We built three device models, `real`, `replicate`, and `uniform`, to estimate E^{real}, $E^{\text{replicate}}$, and E^{uniform}. Then, we use Equation 7.3 to breakdown the prediction error. The `real` model is constructed using the training trace. The training trace in `replicate` is generated from the workload-level descriptions of the training trace using the 9 generator combinations in



(a) tested on `cello99a`



(b) tested on `cello99b`



(c) tested on `cello99c`

Figure 7.2: Error breakdown on Atlas 10K based on Equation 7.3.

Table 6.2. The `uniform` model is based on the mixture of the uniform-sampled synthetic traces, same as the ones used in Section 6.5.

The main observation is that the model error breakdown does depend on the workloads used in the evaluation. For `cello99a` and `cello99b`, the sum of $\text{Error}_{\text{random}} + \text{Error}_{\text{vector}}$ is higher than either of $\text{Error}_{\text{coverage}}$ or $\text{Error}_{\text{generation}}$ in median relative error, indicating the vector workload representation may miss some important workload characteristics. For `cello99c`, $\text{Error}_{\text{coverage}}$ is the most significant error source. One possibility is that the uniform samples do not provide enough coverage on `cello99c` as the `replicate` traces do. Or, the workload-level description does not cover all the important workload characteristics so that synthetic traces perform differently from real traces. We use a CART-specific approach in the next section to further study the model error. The results indicates that both contribute to the model error.

In conclusion, the coverage provided by the synthetic traces seems to be, in general, sufficient for the real-world traces we have evaluated. The majority of the model error comes from the unrealistic trace generators and imperfect workload descriptions. We introduce a CART-specific approach in the next section to validate these observations.

7.2 Coverage

We study the coverage of the synthetic traces and compare the performance difference between the synthetic and real-world traces in this section. The training traces should cover the portion of the workload-level description space that will actually be encountered in practice. Visual inspection of the coverage is impossible because of the high dimensionality of the workload-level description space. This section introduces a CART-specific method to achieve the goal.

7.2.1 Methodology

CART models naturally group data points into sets, one for each leaf. All the data points in the same leaf belong to a hypercube in the input vector space and have similar output values such that there is no effective split on the hypercube to further reduce model error. We take advantage of this knowledge to study the coverage of synthetic traces over real-world traces.

We build a device model based on the mixture of synthetic and real-world traces and then investigate the composition of the data points in each leaf node. If all the leaves containing data points from the real trace also contain data points from the synthetic one, we can conclude that the synthetic trace has a reasonably good coverage for predicting the real one.

Moreover, we can also compare the performance of the two traces grouped by each leaf node. Ideally, all the data points in a leaf node, from both traces, should perform similarly. If the performance difference is significant, then it means either the synthetic trace generators need improvement, or the workload-level description does not capture all the important characteristics that determine workload performance.

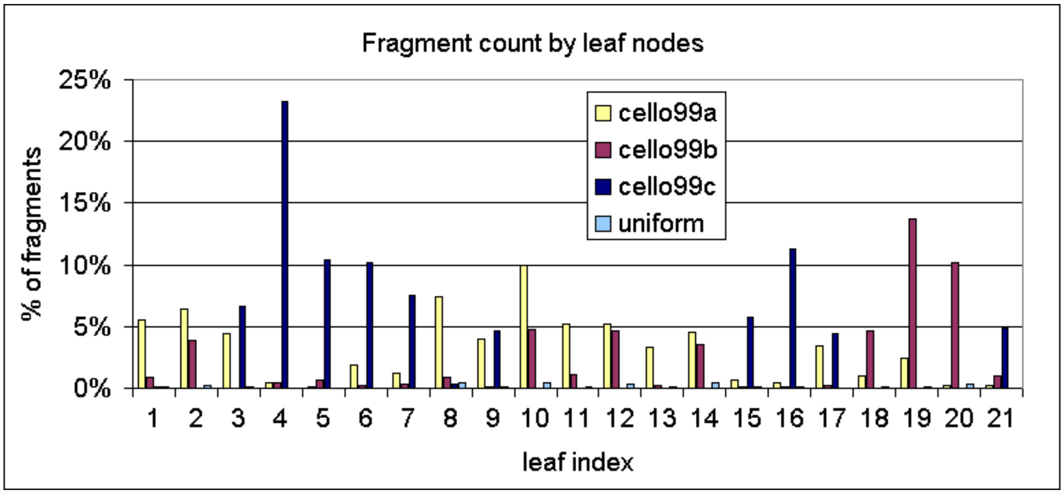
7.2.2 Experimental Results

We use the methodology described above to study the model built for Atlas 10K. We use 5,000 minutes of the `uniform` traces and 5,000 minutes from the mixture of the three `cello99` traces to build a workload-level model for Atlas 10K. Figure 7.3 shows the comparison of the number of data points and the performance measurements between the two traces grouped by the leaves in the CART model. The graphs show only those leaf nodes that have significant number of data points from the `cello99` traces.

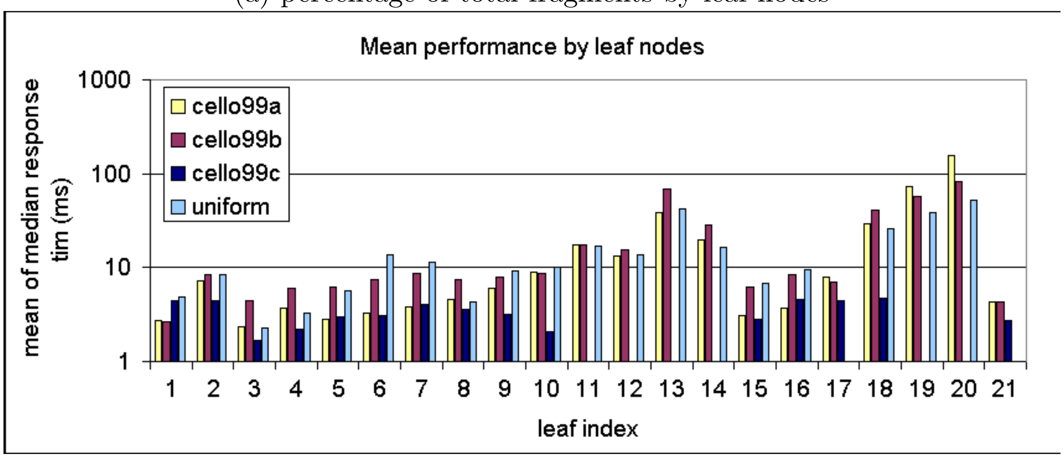
Figure 7.3(a) shows the percentage of the fragments on these leaf nodes. We observe that the real-world traces are clustered in the workload-level description space as a small number of leaves account for the majority of the fragments. In addition, the three traces cover different portions of the workload-level description space, indicating one may need to use more real traces to achieve enough coverage if deciding to use real-world traces in training. The `uniform` trace covers these leaves, too, but poorly because the number of data points per leaf ranges from 10 to 50, which may not be enough to remove the inherent variance in trace generation. A large portion of the uniform trace is devoted to regions that are never seen in the three `cello99` traces. Future work could investigate more real-world traces to direct the sampling algorithm to important regions in the workload-level description space.

If we compare the mean and standard deviation of the aggregate performance measurements of the fragments by the leaf nodes, we observe that, for some leaf nodes, such as leaf 1, 12, and 14, the performance of the synthetic and real-world traces are similar. For some other leaf nodes, such as leaf 3, 10, and 18, even the real-world traces show different performance from each other, indicating that some important characteristics are missing in workload-level descriptions.

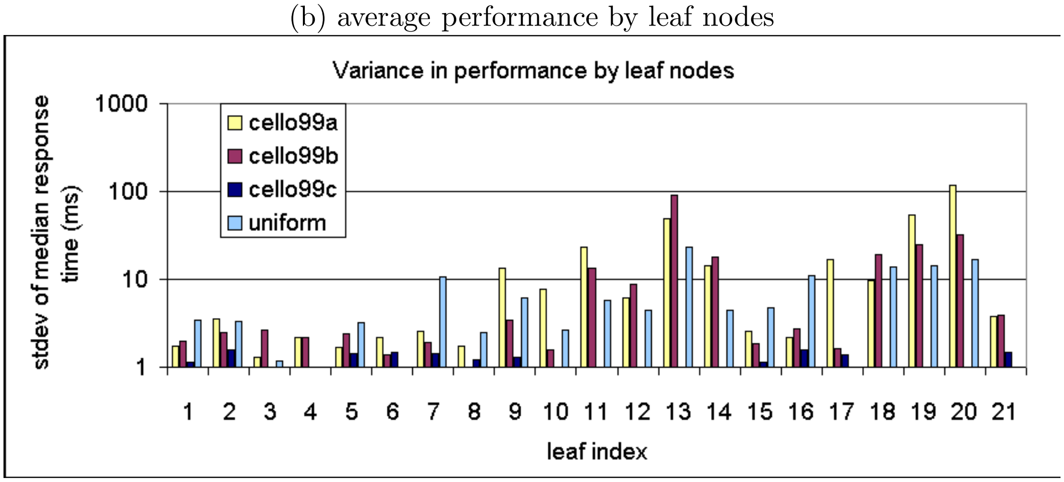
In general, the standard deviation of the performance scales with the mean value. For example, leaf 1 has fragments with small response times, and the standard deviation on leaf 1 is small, too. For leaf 13, 19, and 20, the standard deviation becomes large as the mean value is large. The trend suggests that the prediction is more difficult for heavy-load traffic.



(a) percentage of total fragments by leaf nodes

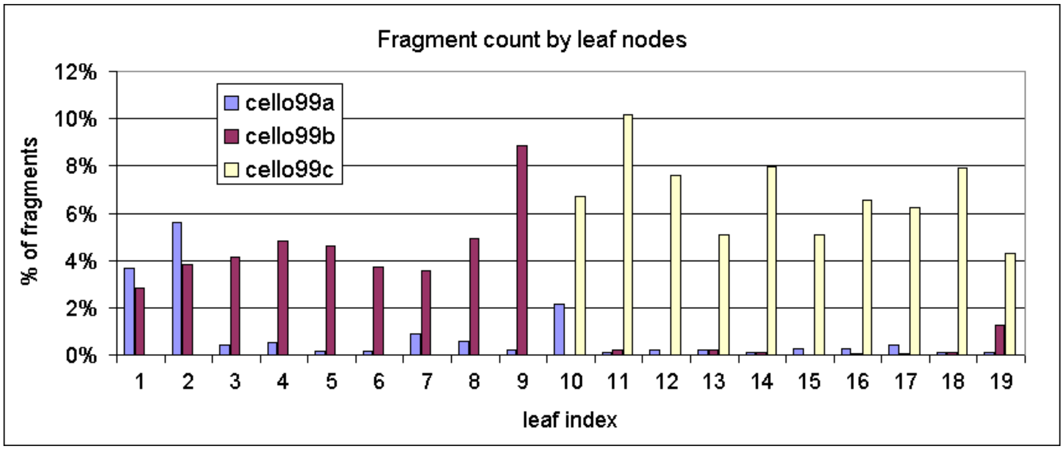


(b) average performance by leaf nodes

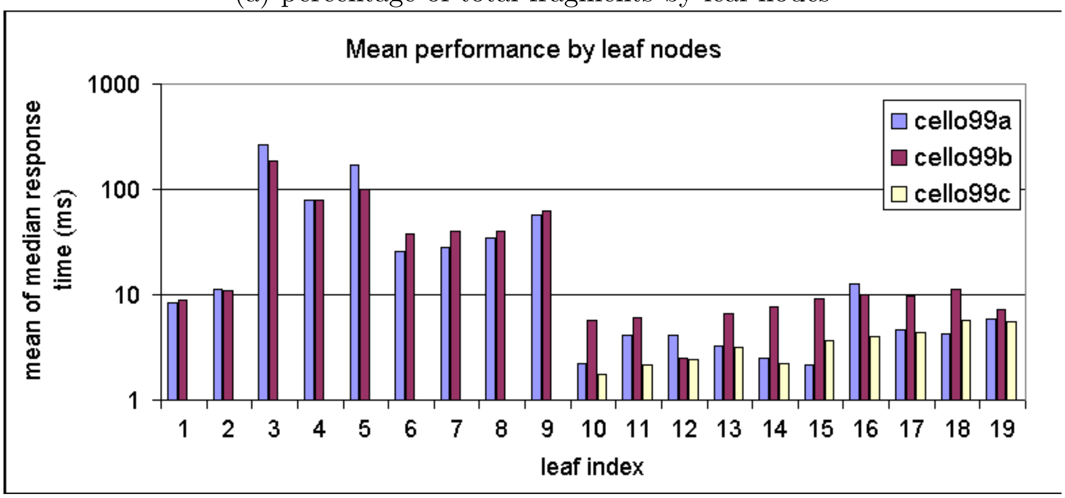


(c) standard deviation of performance by leaf nodes

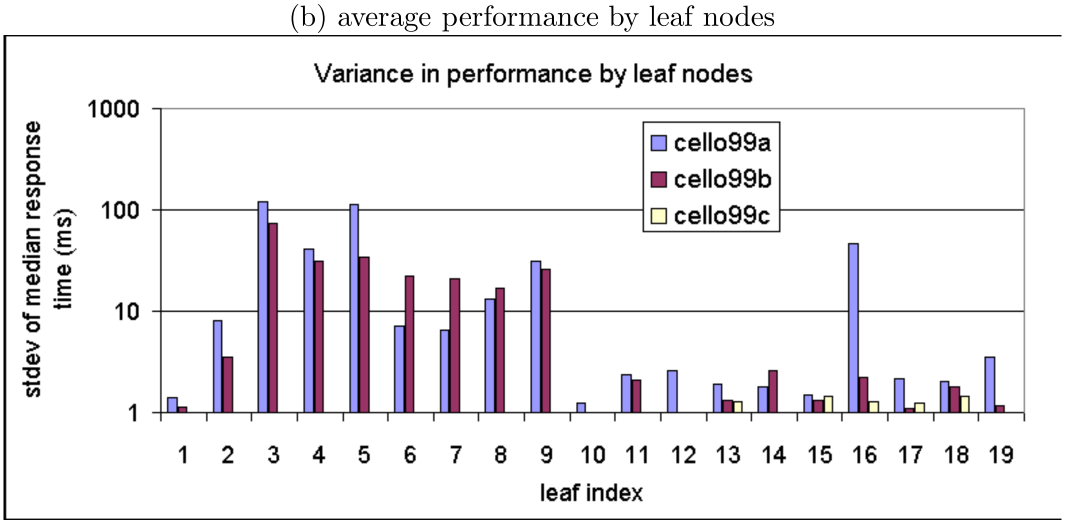
Figure 7.3: Fragment counts and performance by leaf nodes on the CART model constructed on mixture of synthetic and real-world traces.



(a) percentage of total fragments by leaf nodes



(b) average performance by leaf nodes



(c) standard deviation of performance by leaf nodes

Figure 7.4: Fragment counts and performance by lead nodes on CART model constructed on mixture of synthetic and real-world traces.

To further study the effectiveness of our workload-level descriptions, we conduct the same experiment on the device model constructed on the real-world traces only. Excluding the synthetic traces leads to a more accurate model for us to study the performance difference between the fragments from different trace.

Figure 7.4 shows the coverage analysis using only the real-world traces in training. Please note that there is no one-to-one mapping between the leaves in Figure 7.3 and Figure 7.4, because the two CART models are totally different.

First, we observe that the fragments from `cello99b` and `cello99c` live in two totally different sets of leaves. The fragments from `cello99a` are distributed more sparsely across the leaves. This suggests that the workload-level descriptions are enable to distinguish workloads of different characteristics to some extent. If we compare the performance of the three traces on the leaf nodes, we observe that some leaf nodes, such as leaf 1 to 9, see similar performance from `cello99a` and `cello99b`. (There is no data point from `cello99c` on leaf 1 to 9.) Other nodes, from leaf 10 to 18, show different behavior across traces within leaves. The difference indicates that some important characteristics are still missing in workload-level descriptions. Further study on these fragments should be helpful in identifying the missing characteristics. Including parameters to describe these characteristics in workload-level descriptions should improve the quality of the learning-based models.

From the above experiments, we conclude that the synthetic trace generators and workload-level descriptions still have space for further improvement to deliver accurate performance models for storage devices. Therefore, future work should focus on identifying important workload characteristics and developing effective trace generators.

7.3 Online Model Adaptation

During online operation, a system can observe system performance and make necessary adjustments to performance models to improve model accuracy. We call such adjustment online model adaptation. As we have shown in Chapter 4, our device models incur only 10% to 40% of median relative error if we use real-world traces that are similar to the predicted ones in training. Consequently, online adaptation should be able to improve model accuracy. The following experiment is to show the possibility of online adaptation in the learning-based device models.

We first construct a model to predict the median response time over one-minute intervals of Atlas 10K using 5,000 minutes of the `uniform` trace. The trace is the same trace we used in producing Figure 6.6. That is, the trace is a mixture of uniform samples from combination 1, 3, 4, 5, 6, 7 listed in Table 6.2, each combination providing the same amount of fragments.

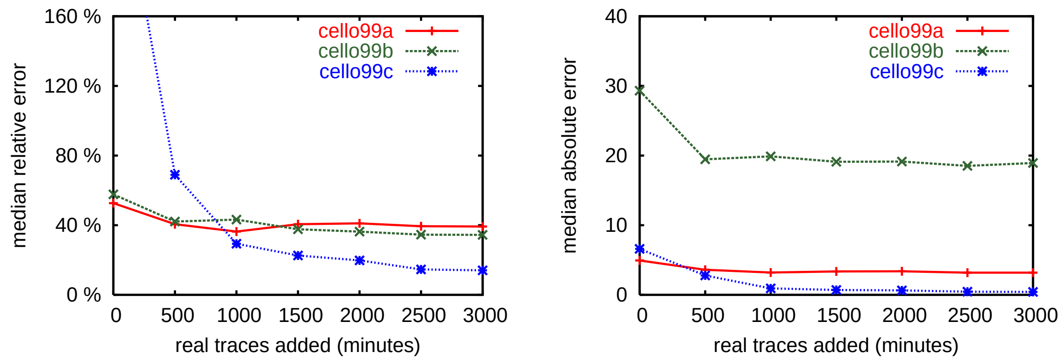


Figure 7.5: Model error reduction versus the amount of real-world traces added to the training set.

We build a bag of trees, one from each combination and select the final predictions using the `stdev` criteria. Later on, we augment the bag of trees with ones constructed from real-world traces, the same training trace used in Section 4.4, one tree per 500 minutes of traces. That is, the original model has 6 trees, and with every 500 minutes of real-world traces, the number of trees grows by 1.

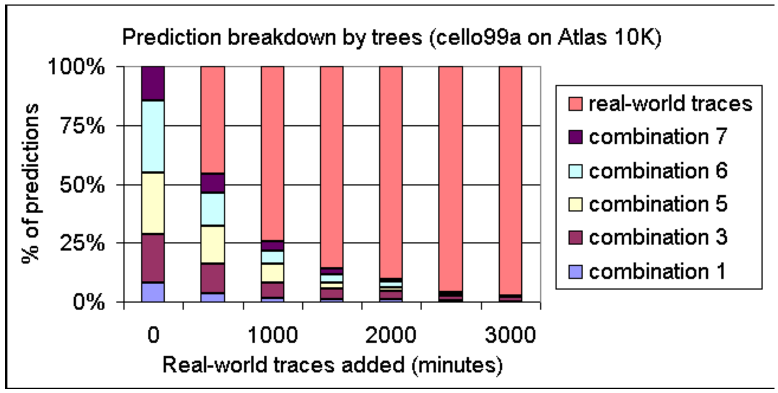
Figure 7.5 shows how the model error decreases as the amount of real-world traces in the training grows. The testing traces are the first set of testing traces used in Section 4.3.4, and have no overlap with the ones used in training. Adding one tree can effectively reduce the model error significantly, especially for `cello99c`, which was poorly predicted by the original model. The benefit of adding more traces to the training set is small because the first 500 minutes have already captured most of device behavior under the predicted traces.

Figure 7.6 shows the frequency that each tree is selected by the `stdev` criteria for final predictions. Once real-workload traces are added to the model, the model chooses to use the trees constructed on the real-world traces for most predictions, leading to the improvement in overall model accuracy.

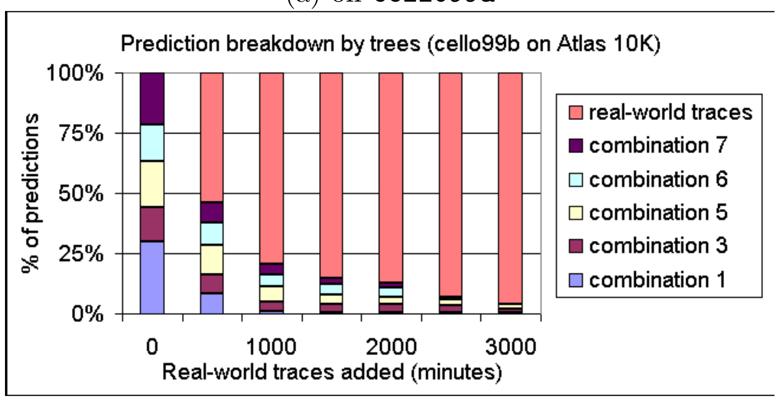
This experiments have shown that when online adaptation is available, our device models can use information collected online to correct themselves, delivering better prediction accuracy.

7.4 Summary

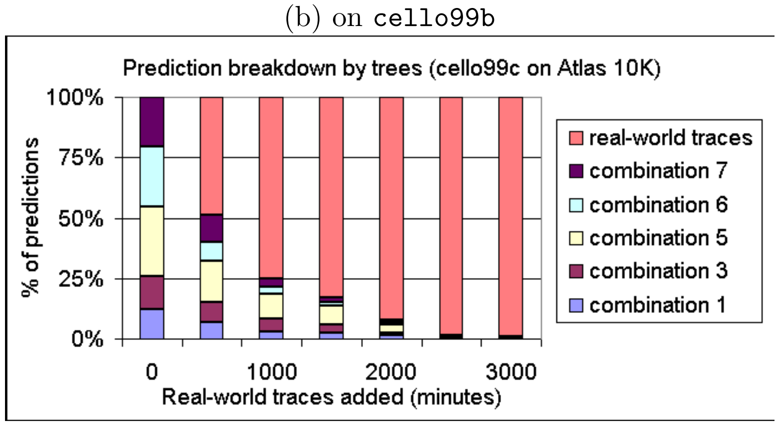
The learning-based device models introduce error to the models in each step of the construction algorithm. The error sources include insufficient coverage of training traces over the workload-level description space, unrealistic assumptions used in generating synthetic



(a) on `cello99a`



(b) on `cello99b`



(c) on `cello99c`

Figure 7.6: Breakdown of predictions by trees used in prediction.

traces from workload-level descriptions, information loss from the transformation from I/O workloads to workload-level descriptions, and intrinsic variance of storage device behavior.

This section has presented two ways to diagnose the models. The first approach uses different traces in training in order to break down model error by their error sources. The second approach takes advantage of the properties of CART models to study the coverage of synthetic traces on the workload-level description space and compare the performance between real-world and synthetic traces. The empirical studies have shown that sampling has probably provided enough coverage over the studied traces, but the quality of the workload-level description and trace generators need further improvement.

Online adaptation allows device models to use information collected during online operation to correct themselves. The experiments have shown that our models can benefit significantly after a small amount of real-world traces (500 minutes).

Chapter 8

Case Study

The target application of performance models is to evaluate system performance under different resource allocation plans for systems to make resource management decisions. All the previous chapters have used trace-based analysis to evaluate the effectiveness of our device models. This chapter presents a resource planning application to evaluate how effective our models are in helping making decisions for storage configuration.

The chapter starts with the design and the evaluation methodology of the application in Section 8.1. Section 8.2 shows the experimental results. Finally, Section 8.4 concludes the chapter.

8.1 Design

The goal of the case study is to evaluate the effectiveness of our learning-based models in facilitating resource allocation. This section presents the design of the sample application and the evaluation methodology.

8.1.1 Target Application

The chosen application is physical database design evaluation. Databases store information in database objects, including relations and indexes. Physical database design refers to the task of assigning storage space to these database objects, equivalent to data layout in file systems. To achieve optimal or near-optimal performance, one needs to understand both workloads and storage devices. Large-scale databases usually contain hundreds of database objects. The number of possible designs is overwhelming, making it impossible for administrators to locate a good physical design. It is, therefore, desirable to automate the process by using performance models to evaluate individual database designs.

In the case study project, we used our learning-based device models to predict the performance of different physical database designs for a TPC-H workload. A TPC-H workload consists of both sequential and random accesses, making it ideal to show the effectiveness of our device models.

TPC-H is a prevailing benchmark simulating decision support systems [35]. A TPC-H database consists of eight tables, and the database size can scale from 1 GB to 1 TB. A power-run includes 22 complex queries in random order. The access patterns include both sequential and random accesses. We run the TPC-H benchmark on DB2 Version 8.2 FixPak 1 (equivalent to Version 2.1 FixPak 8) on a RedHat Linux server. The server is equipped with a 2.6 GHz Intel Pentium IV CPU and 2 GB memory, and runs RedHat Linux 9 (Shrike) with kernel version 2.4.27. In addition to the three 17GB SCSI disks of model ST318406LW by SEAGATE for storing database objects, we store the temporary and log files on a separate Maxtor 6Y080P0 ATA 40GB IDE disk. All the disks have single partitions managed by the `ext2` file system. The database size is 10 GB, and the buffer pool size is 32 MB. The database uses a main-memory page size of 8 KB. Appendix C on page 141 lists the important server information and the parameter files we used to set up the database.

8.1.2 Methodology

Given a layout plan, that is, a physical database design, it takes three steps to estimate the overall system performance. Figure 8.1 illustrates the steps. The algorithm takes a workload and a layout plan as input. The input consists of the access streams on all data objects, and the layout plan determines which storage device each data object resides on. In a more sophisticated layout, data objects can be striped across several devices.

1. **Layout mapping:** construct the disk request streams on each storage device using the layout plan and the workload.

2. **Performance prediction:** predict the performance of each storage device using the performance models. These performance models take the request streams as input and output the aggregate performance measurements.

3. **Aggregation:** derive the overall system performance from the individual device performance. For example, we can calculate the overall average response time if we know the averages and request counts for all the fragments. Or, we can use the maximum of all the fragments.

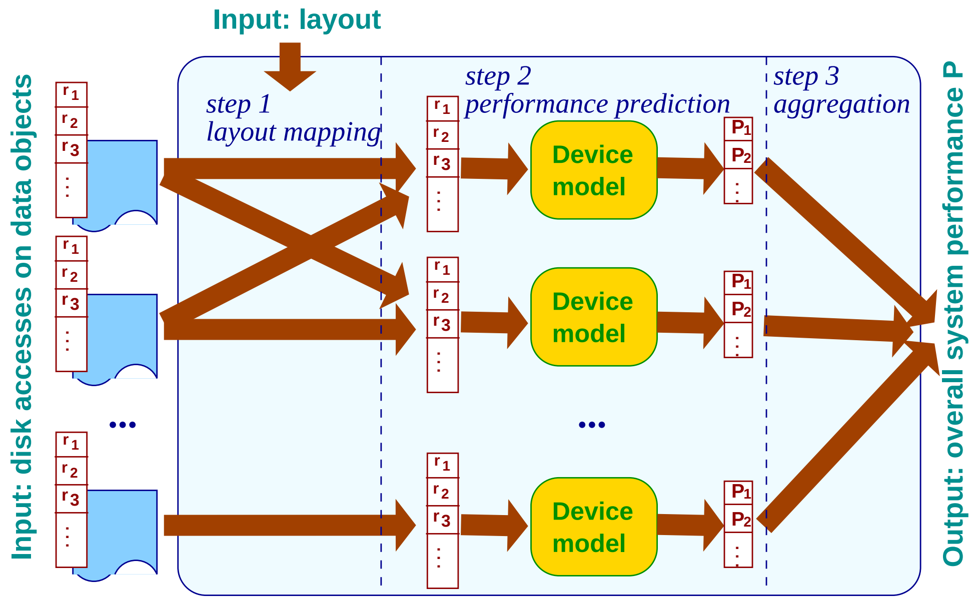


Figure 8.1: Data layout evaluation using performance models for storage devices. Given a workload consisting of all the accesses to the data objects, the first step is to reconstruct the disk request streams on each storage device under the given layout. Performance models then predict the individual device performance, and aggregating them offers the overall system performance.

This approach requires the workload to be presented as the access streams on individual objects. We need to collect traces of running systems and determine the target object of each request to do the layout mapping. The evaluation of our device models, therefore, involves three steps: collecting traces, constructing disk models, and evaluating the layouts.

We collect the traces of 5 power-runs on a 10GB TPC-H database. The machine is rebooted after every run in order to flush the memory cache. We patched the Linux kernel with the `dtb` package to intercept all the accesses to the three SCSI disks. `Dtb` is a low-level kernel patch for Linux, which allows one to collect I/O activities on SCSI disks [86]. We create the database using system-managed table spaces. That is, each table lives in a file managed by the file system. By scanning these files, we are able to know the LBN of each data block in these files, and therefore, determine the target table of each disk request in the I/O trace. Appendix C.2 list the SQL statement we used to create the table spaces and the tables. Because the file system makes their best effort in allocating contiguous disk blocks to files, we observe nearly all the disk blocks are contiguous in the LBN space.

We used 3,000 minutes of `uniform` trace and the collected TPC-H runs to construct device models for the SCSI disks. Because the three SCSI disks are identical, we only

Layout index	Tables on each disk			Measured performance		
	disk 1	disk 2	disk3	run time	request count	response time
1	all	(empty)	(empty)	8334(22)	1.64 million	3.34ms
2	rest	lineitem	partsupp, part	8198(19)	1.65 million	3.16ms
3	lineitem, rest	lineitem, part	lineitem	5880(69)	1.32 million	4.46 ms
4	all	all	all	5143(175)	1.18 million	6.96 ms

Table 8.1: Four candidate database designs. If a table is shown in more than one column, the data and index on the table are striped across these devices with stripe size 256 KB. The "rest" stands for the accesses other than the ones on the specified tables. The run time is the average of 8 runs and is presented in seconds. The values in parenthesis are the standard deviation of the 8 runs.

replay synthetic traces on one using Buttress [11] to construct a model to predict the average response time over one-minute intervals.

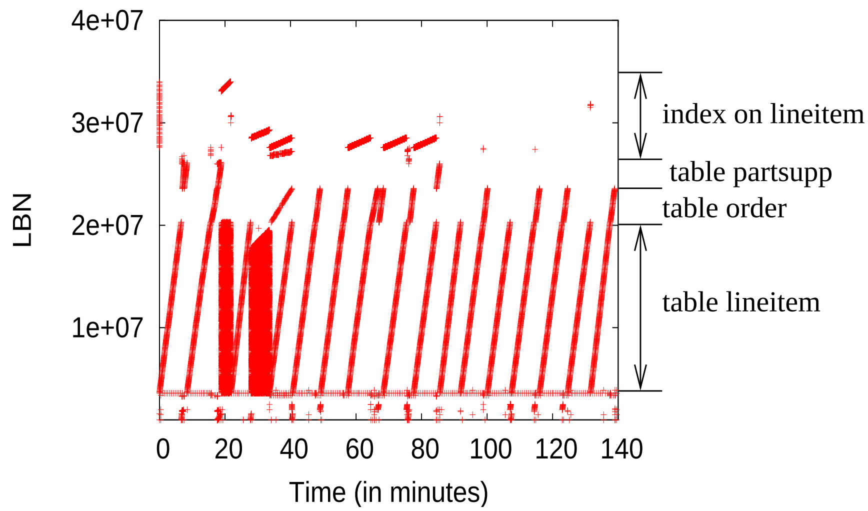
8.2 Experimental Results

We use our device models to compare the ranking of different physical database designs for TPC-H databases. This section presents the experimental results.

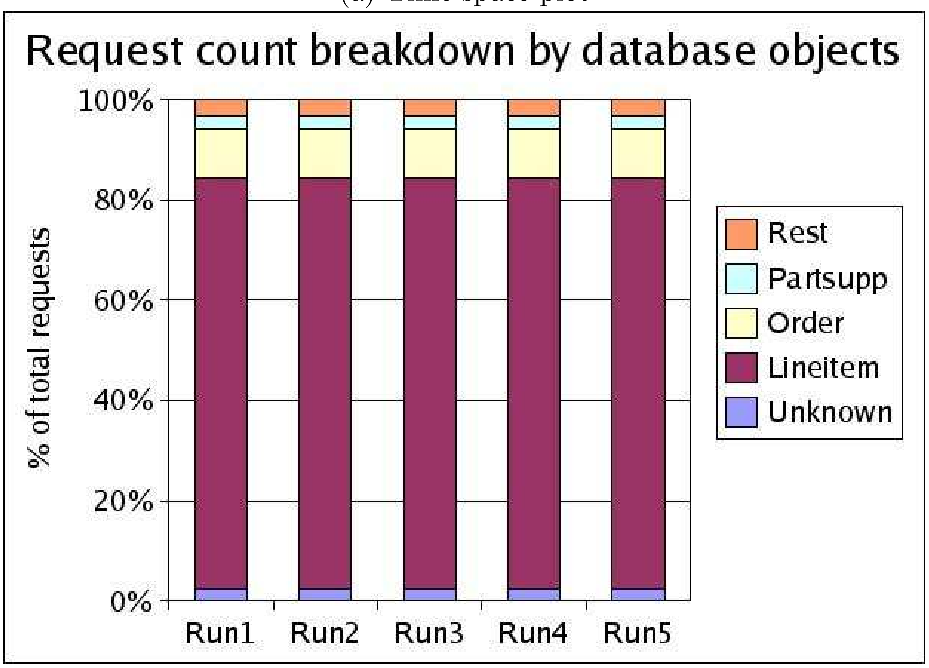
Physical database designs. Table 8.1 lists four physical designs. We implement the striping by assigning the tables to a table space consisting of three files on the three disks. The database system automatically stripes data across the three files with the extent size as the stripe unit, which is 512 KB in our configuration. For each layout, we ran 8 power-runs and reported the average run time as well as the standard deviation in parentheses. Table 8.1 also lists the measured performance of the four physical layouts. For each layout, the average and standard deviation of the run time are based on 8 power-runs. The request counts are calculated on traces collected on separate power-runs.

Because the sequential runs on different tables are not interleaved, moving entire tables to different disks, such as layout 2, offers only little advantage over layout 1. If we stripe table `lineitem` across the three disks as in layout 3, we observe a 30% reduction in the running time. Further striping all the tables (layout 4) helps even more.

The speedup offered by good layouts comes from two sources. First, the total number of I/O requests tends to decrease for good layouts because the average request size is larger, and the total bytes accessed by the application decreases, too. Table 8.2 lists the summary characteristics of the workload. Second, the sequentiality increases from layout 1 to 4, too,



(a) Time-space plot



(b) Breakdown by tables

Figure 8.2: Disk activities of 5 power-runs on a 10GB TPC-H database. (a) shows the activity of a power-run. (b) shows the request breakdown by tables. There is little variance across different runs. (For the tables shown in the figure, the standard deviation of the request counts is less than 1% of the average.)

Layout index	Real trace characteristics			Predicted trace characteristics			Predicted
	request count	request size	seq.	request count	request size	seq.	response time
1	1.64 million	93.7 KB (46.9)	33%	1.64 million	93.7 KB	33%	4.21 ms
2	1.65 million	93.0 KB (46.9)	33%	1.64 million	93.7 KB	33%	3.68 ms
3	1.32 million	100.7 KB (43.8)	43%	1.64 million	93.7 KB	42%	3.32 ms
4	1.18 million	102.6 KB (43.7)	53%	1.64 million	93.7 KB	47%	3.11 ms

Table 8.2: Comparison of workload characteristics between the real traces and the ones produced by layout mapping and the predicted response times. We report the average and standard deviation (in parentheses) request size in the table. All the predicted request counts and request sizes are the same as the ones for the actual layout 1 trace because layout mapping uses the actual layout 1 trace as input.

leading to more efficient I/O accesses. As a result, layout 4 is the best physical one among the four layouts.

Figure 8.3(a) shows the execution time breakdown for the four layouts. The breakdown is the average of 8 power-runs. During each run, we used the `iostat` command to collect the computation time and I/O blocking time every 30 seconds. The graph shows that the speedup from layout 1 to 4 is mainly due to the decreasing I/O blocking time. At the same time, the computation time decreases, too, that is, 10% for layout 3 and 22% for layout 4. A part of the decrement comes from the decreased I/O overhead, and the rest may be the result of the increased chance of instruction-level parallelism enabled by fast I/O operations. The standard deviation of the total running time tends to increase from layout 1 to 4 because a greater chance of the overlapping between I/O and computation leads to a higher variance in the running time.

TPC-H power-run characteristics. Figure 8.2 shows the characteristics of the TPC-H power-runs. During trace collecting, we put all the tables on a single SCSI disk. The run is dominated by the sequential runs on table `lineitem`, which comprise more than 80% of the total activity. The accesses on table `order` account for another 10% of the requests. Less than 3% of the requests are on the file system meta data and the temporary table space. The breakdown across different runs is stable, which is ideal for performance prediction.

Effectiveness of layout mapping. Figure 8.4 compares the actual and predicted request counts on the three SCSI disks for layout 2 to 4. Our layout mapping algorithm, as illustrated in Figure 8.1, takes a layout plan and a layout 1 trace to reconstruct the request streams received by the three disks under the planned layout. First, we notice that the total number of requests on layout 3 and 4 is significantly lower than the total number of requests on layout 1. Table 8.2 lists the statistics. As a result, the predicted volumes on layout 3

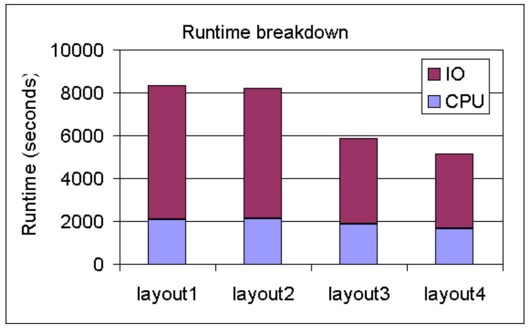


Figure 8.3: Measured runtime breakdown for the four physical database design obtained by the `iostat` *command on Linux. The CPU time is the product of the CPU utilization and the elapsed time. We report the rest of the elapsed time as I/O time.*

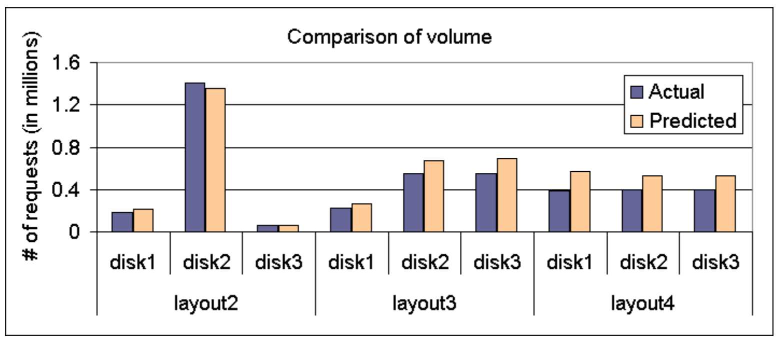


Figure 8.4: Comparison of the actual and predicted volume received by individual disks in the three physical data designs.

and 4 are higher than the actual volumes because the algorithm uses the trace collected on layout 1 to construct the streams. Second, the mapping tends to overestimate the volume on disk 1 percentage-wise because the mapping algorithm directs all the accesses to the file system meta data to disk 1.

Table 8.2 compares the characteristics between the actual traces collected on the four layouts and the traces constructed by the layout mapping algorithm. The predicted request count and size are the same as the actual workload on layout 1 because layout mapping uses this trace as input. The sequentiality of the predicted traces increases from the first two layouts to the last two layouts, following the same trend as observed in the actual traces though the actual sequentiality is underestimated by the algorithm. As a result, we should expect to see that the predicted response time is higher than the actual response time because of the increased volume and reduced sequentiality for layout 3 and 4.

Layout index	Predicting constructed traces	Predicting constructed traces after rescaling	Predicting collected traces	Measured response time
1	4.21 ms	4.21 ms	4.21 ms	3.21 ms
2	3.68 ms	5.56 ms	3.36 ms	3.16 ms
3	3.32 ms	8.88 ms	4.80 ms	4.46 ms
4	3.11 ms	12.15 ms	5.13 ms	6.96 ms

Table 8.3: Predicted performance for the four physical layouts. Before rescaling, the requests inherit their arrival times from the layout 1 trace, so the length of the constructed traces are the running time of a power-run under layout 1. After rescaling the arrivals, the lengths are equal to the average running time on the layouts.

Effectiveness of storage device models. We used 3,000 minutes of `uniform` trace and the collected TPC-H runs to construct device models for the SCSI disks. The traces were collected on a 10GB database deployed with layout 1. We calculate the overall average response time by weighting the average response times on individual disks using their volume. The last column in Table 8.2 reports the predicted performance. The observation is that the predicted response times rank the four layouts in the same order as the actual running time does. However, comparing against the measured response times as reported in Table 8.1, we observe that a significant difference exist between the predicted and measured response times. We need to understand where the difference comes from before we can make conclusions on the usefulness of our storage device models. The next section provides more analysis in understanding the difference.

8.3 Understanding Results

To verify the validity of the predictions, we have conducted an additional set of experiments, and this section presents the results and our conclusion. The predicted response times for the four layouts are ranked similarly as the actual running times. Column 1 of Table 8.3 reports the predicted response times on the constructed traces. These numbers are the same as the ones in the last column of Table 8.2. Column 4 of Table 8.3 reports the measured response times (the same as the ones in Table 8.1), and these response times are for the actual deployment of the layouts, that is, on the collected traces. The difference between column 1 and 4 comes from the following three sources.

1. Difference in time scales between the collected and constructed traces. The constructed request streams use the timestamps generated from layout 1, which span a longer period of time than the actual runs. As a result, the average arrival rate in the constructed traces is lower than in the actual traces. To show the contribution of

the difference in time scales, we introduce a third set of traces, the rescaled traces, in which the arrival times have been rescaled so that the rescaled traces have the same length as the collected traces. That is, the rescaled traces are the same as the constructed traces except for the arrival times. Table 8.3 reports the predicted response times on the rescaled traces in column 2. We observe that rescaling makes a significant difference in the predictions, and the predictions on the rescaled traces provide nearly the same order as the measured response times do. All the predictions for layout 1 stay the same because we use the layout 1 trace in layout mapping and the constructed and rescaled traces are the same for layout 1.

2. Difference in trace characteristics between the collected and constructed traces. Table 8.3 reports the predicted response times on the collected traces in column 3. The predictions on the rescaled traces are significantly higher than the ones on the collected traces because of reduced sequentiality and increased traffic volume in the rescaled traces as the summary statistics in Table 8.2 shows. As a result, the predictions for the collected traces are lower than the ones on the rescaled traces.

3. Prediction errors of the storage device model. Comparing the predicted and measured response times on the collected traces (column 3 and 4 in Table 8.3) shows that the predicted response times are ranked similarly to the measured response times, despite that the model has a median relative error of 18% and a 1 ms median absolute error. The correct ranking by the predictions suggests that the storage device models can provide accurate performance predictions when given good training traces. The result further confirms that the storage device model has made reasonable predictions for making resource planning decisions.

In summary, the above three factors contribute to the difference between the first and fourth columns in Table 8.3.

$$
\begin{aligned}
& \text{Diff(predicted response times, measured response times)} \\
= \; & \text{Diff(column 1, column 4)} = -0.87 \text{ ms} \\
= \; & \text{Difference in time scales} \\
& +\text{Difference in trace characteristics} \\
& +\text{Prediction error of storage device model} \qquad (8.1) \\
= \; & \text{Diff(column 1, column 2)} \\
& +\text{Diff(column 2, column 3)} \\
& +\text{Diff(column 3, column 4)} \\
= \; & -4.19 \text{ ms} + 3.33 \text{ ms} - 0.07 \text{ ms}
\end{aligned}
$$

where we define Diff(a, b) = Average(a - b). From the above analysis, we conclude that the performance predictions from our storage device model lead to the right decision in this case study. The result demonstrates the usefulness of our learning-based storage device models: the models can provide fast and accurate predictions without physical deployment of workloads, making them a useful tool in resource planning.

Another lesson we have learned from this experiment is that, in practice, the average response time do not always have a strong correlation with the running time because asynchronous I/O allows applications to hide the I/O latency and the number of I/Os can also change from one storage configuration to another. As a result, the actual response time is not a good metric for such resource planning, and it is more appropriate to predict the throughput or device utilization, the maximum volume of traffic that a storage device can sustain.

8.4 Summary

The ultimate goal of our device models is fast and efficient evaluation of storage resource allocation plans. This chapter employs a real application, physical database design, to illustrate the usage of the device models. We have shown that our device models can provide accurate rankings of physical database designs for a TPC-H workload, proving the usefulness of such performance models in making resource planning decisions.

In evaluating resource allocation plans, it is more important for the models to give an accurate ranking rather than accurate predictions because the ultimate goal is to use the predictions to compare those plans. This poses an interesting question: how to evaluate a device model if only the relative ranking is important. The question is difficult to answer. The closer our predictions are to the actual measurements, the better chance the models can rank resource allocation plans correctly. How good is good enough for a device model to be useful in practice depends on applications, and it requires another thesis to answer this question.

Chapter 9

Conclusions

Performance models can provide fast estimations of system performance for autonomic systems to make resource allocation decisions. This thesis has focused on the feasibility of using machine learning techniques to construct performance models for storage devices. A major advantage of the learning-based models is the fully-automated model construction algorithm. Storage devices are treated as black boxes, so no prior knowledge about the devices are required. The models learn device behavior by observing the devices under a set of training traces and model the behavior as functions over input workloads. Internally, an I/O workload is represented as vectors, and a regression tool models the target performance measurements as functions over the vectors.

The main contributions of the thesis include exploring the design space of the learning-based approach, evaluating existing techniques in addressing these issues, and an error analysis framework based on model analysis. We have discovered that when the training and testing traces are similar, the models can make decent predictions. Offline training using synthetic traces is less effective because of the less-than-ideal vector representations of workloads and unrealistic synthetic traces. Online model adaptive could remove most model error introduced by the offline training. In particular, our contributions and conclusions are:

- We have designed and evaluated two types of vector representations of I/O workloads. A request description characterizes the characteristics of a single request and is used to predict its response time. A workload-level description captures the important characteristics of a sequence of requests, and a workload-level model models the aggregate performance of the sequence as a function over the workload-level description. We have used several real-world traces to evaluate the effectiveness of these vector representations in capturing important workload characteristics. The experimental results have shown that both can yield models of a median relative error around 10% to 40% (less than 10 milliseconds) for most traces and devices when the training and testing traces are similar in workload characteristics. This result suggests the effectiveness of the descriptions in capturing important workload characteristics.

- We have compared popular regression tools to select an appropriate one as the basis of our models. The final answer is CART models after an empirical study on real-world traces. CART models offer competitive accuracy and efficiency in both computation and storage, in addition to good interpretability.

- We have explored the effectiveness of using synthetic traces in training to construct accurate device models. Sample workload-level descriptions are produced first to offer a good coverage over the interesting region of the workload-level description space, and the system generates synthetic traces for these sample workload-level descriptions. Our evaluation used the uniform sampling technique and a set of trace generators to generate traces. The experimental results have shown that the trace generators are unable to produce training traces of high quality, the reason being the ignoring of the strong correlations between requests. Model error is more than doubled if we compare the models constructed from the synthetic traces against the ones built using real-world traces.

- We have proposed a set of tools based on empirical studies to analyze model error systematically. We have identified four error sources in our modeling approach. By breaking down the error by the error sources and comparing workloads of similar characteristics, we conclude that both the trace generators and workload-level descriptions have space for future improvement.

Other contributions include:

- We have proposed two tools, the entropy plot and the PQRS model, to supplement existing work in workload characterization. The entropy plot takes advantage of self-similarity of I/O workloads and quantifies the spatio-temporal behavior of I/O workloads using three scalars. The PQRS model can take the entropy plot slope measurements as input and produce traces of specified characteristics. The two tools become important parts of workload-level descriptions and trace generators respectively in our modeling approach.

- We have evaluated the effectiveness of our device models in helping making resource allocation decisions using a real application. The evaluation has shown that even with inaccurate predictions, the device models can provide a correct ranking of different resource allocation plans.

- We have presented three ways to quantify the importance of the parameters in the vector representations of I/O workloads in determining device performance. The measurements can help us to understand the interaction between storage devices and I/O workloads.

Chapter 10

Future Work

This thesis has explored the feasibility of using machine learning techniques to build performance models for storage devices. The following issues could be interesting to explore but are not addressed in this thesis.

- **Improving the workload-level descriptions.** We have shown in the error analysis that our workload-level descriptions still miss important characteristics even though they have captured a small set of them to make decent predictions. It is worthy to study the difference between workloads with similar characteristics but different performance to identify the missing characteristics.

- **Fine tuning of the parameters in the workload-level descriptions.** One possibility is to explore different values for preset thresholds. For example, we currently use 10 milliseonds to break I/O traces into bursts of requests for calculating BurstSize and BurstRate, and one minute as the window size for aggregate performance. Another direction is to design other methods than the average to characterize a distribution in workload-level descriptions. For long-tailed distributions, the sample mean is a poor predictor of the actual mean. It could be interesting to investigate the benefits of using other ways to capture distributions.

- **Applying the parameter selection and dimension reduction algorithms** as described in Chapter 5. Strong correlations exist between the parameters of our descriptions, imposing difficulties in understanding the models and producing meaningful samples from the vector space for the training trace generation. Techniques such PCA and ICA could potentially reduce the dependence.

- **Improving the synthetic trace generators.** This step is tightly bounded to workload-level descriptions because the generators use them as input. Our error analysis has shown unrealistic synthetic traces used in training make significant contributions to the model error. A next step should investigate the difference between synthetic traces and real-world ones.

Other issues related to the device modeling problem include:

- **Automatic feature extraction.** Our current design uses a pre-defined set of parameters to characterize I/O workloads on all the devices. Since devices perform differently, the set of important parameters could vary. The model construction algorithm can incorporate existing work on identifying important workload characteristics [65] to automatically acquire a compact and effective vector representation.

- **Answering "how good is good enough"** for device models to be useful in resource management. This answer may depend on applications, storage devices, and the search algorithm used in locating the optimal solution. Understanding the dependence will help us to determine the target accuracy of our device models.

- **Finding effective ways to describe mixed workloads.** In evaluating candidate resource allocation plans, our current approach takes a detailed I/O trace as input and generates request streams on individual devices by mapping requests to target devices. Ideally, we would like to keep a compressed version of the trace and derive the characteristics of the request streams from the compressed data to reduce storage overhead.

Our current approach assumes that a device is physically available for training. In addition, new training is required if the device changes its configuration, such as the cache capacity and/or the RAID level. It is helpful for us to predict performance changes after a reconfiguration without new training. A possible solution is to build a set of performance models for a set of configurations, and use an interpolation algorithm to predict device performance of other configurations. In addition, it could be interesting to extend the learning-based approach to other system components, such as networks, memory systems, etc.

Appendix A

DiskSim Parameter Files

We list the important part of the DiskSim parameter file used in simulating the RAID 1/0 array as follows. Please refer to the DiskSim 3.0 manual [24] for the meaning of the parameters.

```
disksim_iodriver DRIVER0 {
  type = 1,
  Constant access time = 0.0,
  Scheduler = disksim_ioqueue {
   Scheduling policy = 3,
   Cylinder mapping strategy = 1,
   Write initiation delay = 0.0,
   Read initiation delay = 0.0,
   Sequential stream scheme = 0,
   Maximum concat size = 0,
   Overlapping request scheme = 0,
   Sequential stream diff maximum = 0,
   Scheduling timeout scheme = 0,
   Timeout time/weight = 30,
   Timeout scheduling = 3,
   Scheduling priority scheme = 0,
   Priority scheduling = 3
  }, # end of Scheduler
  Use queueing in subsystem = 0
} # end of DRV0 spec

disksim_bus BUS0 {
  type = 2,
  Arbitration type = 1,
  Arbitration time = 0.0,
  Read block transfer time = 0.0,
  Write block transfer time = 0.0,
  Print stats =  0
} # end of BUS0 spec
```

```
disksim_bus BUS1 {
  type = 1,
  Arbitration type = 1,
  Arbitration time = 0.0,
  Read block transfer time = 0.0,
  Write block transfer time = 0.0,
  Print stats =  0
} # end of BUS0 spec

disksim_ctlr CTLR0 {
  type = 1,
  Scale for delays = 0.0,
  Bulk sector transfer time = 0.0,
  Maximum queue length = 0,
  Print stats =  1,
Cache = disksim_cachemem {
     Cache size = 8196,
     SLRU segments = [ 1.0 ],
     Line size = 64,
     Bit granularity = 1,
     Lock granularity = 64,
     Shared read locks = 1,
     Max request size = 512,
     Replacement policy = 2,
     Allocation policy = 0,
     Write scheme = 3,
     Flush policy = 0,
     Flush period = 0.0,
     Flush idle delay = -1.0,
     Flush max line cluster = 8,
     Read prefetch type  = 0,
     Write prefetch type = 0,
     Line-by-line fetches = 0,
     Max gather = 7
}  # end of cachemem spec
}

disksim_ctlr CTLR1 {
  type = 1,
  Scale for delays = 0.0,
  Bulk sector transfer time = 0.0,
  Maximum queue length = 0,
  Print stats =  1
}

source atlas10k.diskspecs

# component instantiation
```

```
instantiate [ statfoo ] as Stats
instantiate [ outerctlr0, outerctlr1] as CTLR1
instantiate [ ctlr0, ctlr1 ] as CTLR0
instantiate [ outerbus, middlebus0, middlebus1 ] as BUS0
instantiate [ bus0, bus1  ] as BUS1
instantiate [ disk0 .. disk15 ] as QUANTUM_TORNADO_validate
instantiate [ driver0 ] as DRIVER0
# end of component instantiation

# system topology
topology disksim_iodriver driver0 [
disksim_bus outerbus [
        disksim_ctlr outerctlr0 [
                disksim_bus middlebus0 [
                        disksim_ctlr ctlr0 [
                                disksim_bus bus0 [
                                    disksim_disk disk0 [],
                                    disksim_disk disk1 [],
                                    disksim_disk disk2 [],
                                    disksim_disk disk3 [],
                                    disksim_disk disk4 [],
                                    disksim_disk disk5 [],
                                    disksim_disk disk6 [],
                                    disksim_disk disk7 []
                                  ] # bus0
                        ] # ctlr0
                ] # middlebus0
        ], # outerctlr0
        disksim_ctlr outerctlr1 [
             disksim_bus middlebus1 [
                  disksim_ctlr ctlr1 [
                        disksim_bus bus1 [
                            disksim_disk disk8 [],
                            disksim_disk disk9 [],
                            disksim_disk disk10 [],
                            disksim_disk disk11 [],
                            disksim_disk disk12 [],
                            disksim_disk disk13 [],
                            disksim_disk disk14 [],
                            disksim_disk disk15 []
                        ] # bus2
                  ] # ctlr1
             ] # middlebus1
        ] # outerctlr1
] # outerbus
]

disksim_logorg org0 {
   Addressing mode = Array,
```

```
   Distribution scheme = Striped,
   Redundancy scheme = Shadowed,
   devices = [ disk0 .. disk15 ],
   Stripe unit  =  64,
   Synch writes for safety =  0,
   Number of copies =  2,
   Copy choice on read =  6,
   RMW vs. reconstruct =  0.5,
   Parity stripe unit =  64,
   Parity rotation type =  1,
   Time stamp interval =  0.000000,
   Time stamp start time =  60000.000000,
   Time stamp stop time =  10000000000.000000,
   Time stamp file name =  stamps
} # end of logorg org0 spec
```

The parameter file used for simulating the RAID 5 array shares almost the same configuration with the previous except the logical organization of the disks, which is listed below.

```
disksim_logorg org0 {
   Addressing mode = Array,
   Distribution scheme = Striped,
   Redundancy scheme = Parity_rotated,
   devices = [ disk0 .. disk15 ],
   Stripe unit  =  64,
   Synch writes for safety =  0,
   Number of copies =  2,
   Copy choice on read =  6,
   RMW vs. reconstruct =  0.5,
   Parity stripe unit =  64,
   Parity rotation type =  1,
   Time stamp interval =  0.000000,
   Time stamp start time =  60000.000000,
   Time stamp stop time =  10000000000.000000,
   Time stamp file name =  stamps
} # end of logorg org0 spec
```

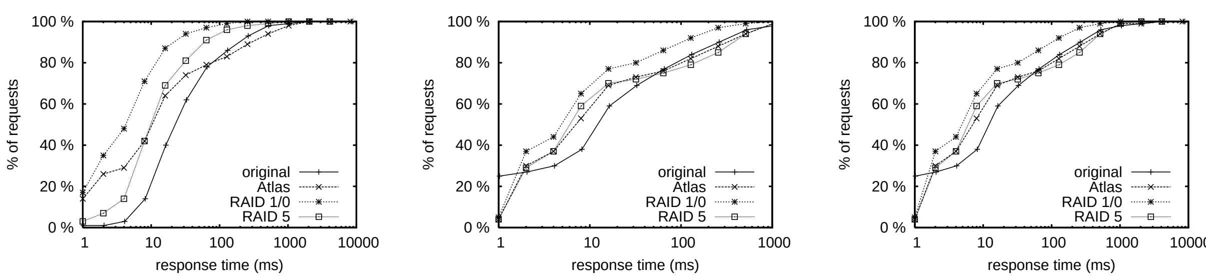
Appendix B

Various Distribution Functions

We show additional plots for various distributions in this section in addition to the cumulative and negative cumulative distribution functions shown in the formal text.

B.1 Comparing Against Original Response Times

The `cello99` traces come with the response time information collected in the original system. Figure B.1 compares the original response time distributions and the ones calculated from the simulated runs on the three devices we studied in this work.

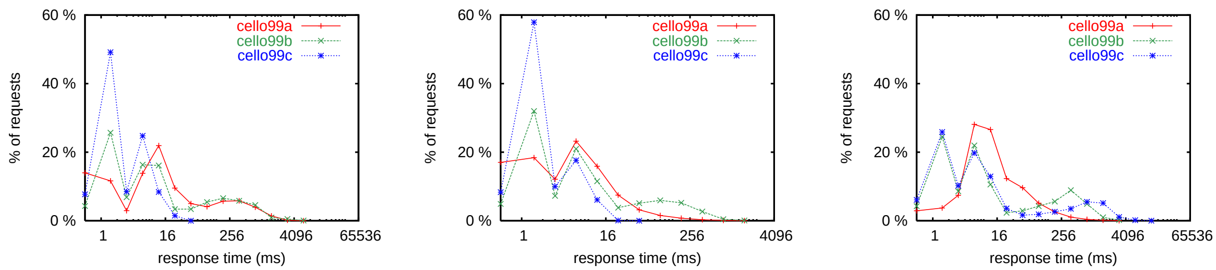


(a) cello99a (b) cello99b (c) cello99c

Figure B.1: Comparison of the original cello system performance against the simulated runs.

B.2 Response Time Distribution

Figure B.2 shows the probability distribution functions of the response times for the three `cello99` traces on the three storage devices under investigation. The cumulative and negative cumulative distribution functions are in Figure 3.4 on page 25. In generating these plots, we bucketize the x-axis by the powers of two. That is, we count the number of requests within intervals $[0-1], (1-2], (2-4], (4-8], \ldots$, and plot the percentage of requests against the logarithm of the middle values of the intervals.

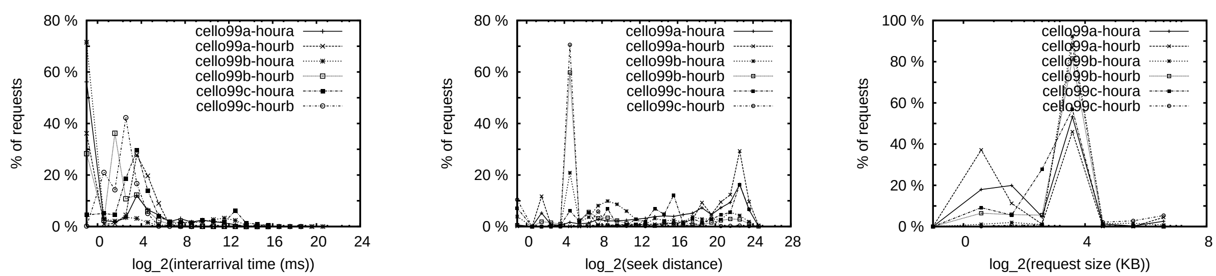


(a) Atlas 10K (b) RAID 1/0 (c) RAID 5

Figure B.2: Response time distribution of the three `cello99` *traces on the three devices.*

B.3 Workload-Level Description Parameters

Figure B.3 shows the probability distribution functions of the three parameters estimated on six hours of `cello99` traces. The cumulative and negative cumulative distribution functions are in Figure 4.16 on page 58. The same method as mention in the previous paragraph is used.



(a) interarrival time distribution (b) seek distance distribution (c) request size distribution

Figure B.3: Distributions of the interarrival time, seek distance, and request size from 6 hours of different volumes from the three `cello99` *traces.*

Appendix C

Configuring the TPC-H Database

C.1 Server Parameters

The database server rans on a RedHat 9 (Shrike) server with a kernel version of 2.4.27. The server is equipped with a 2.6 GHz Intel Pentium IV CPU and 2 GB memory, and has total of 1 IDE disk and 4 SCSI disks. The disks are partitioned and mounted as follows.

```
> more /proc/partitions
major minor  #blocks  name     rio rmerge rsect ruse wio wmerge wsect wuse running\
 use aveq

   8     0   17921835 sda 20 108 256 80 0 0 0 0 0 80 80
   8     1   17920476 sda1 19 105 248 70 0 0 0 0 0 70 70
   8    16   17921835 sdb 2864 11416 113634 6230 290806 75476 2932224 4441190 0 \
191020 4447420
   8    17   17920476 sdb1 2863 11413 113626 6230 290806 75476 2932224 4441190 0 \
191020 4447420
   8    32   17921835 sdc 36 88 386 200 295 0 2360 21960 0 3350 22160
   8    33   17920476 sdc1 35 85 378 190 295 0 2360 21960 0 3340 22150
   8    48    8891620 sdd 12 68 160 70 0 0 0 0 0 70 70
   8    49    8883913 sdd1 11 65 152 70 0 0 0 0 0 70 70
   3     0   80043264 hda 98154 256529 2834842 581260 11293467 9525260 166988898 \
39353497 -4 32757510 38488915
   3     1     104391 hda1 32 73 210 130 86 43 258 198940 0 99700 199070
   3     2    2040255 hda2 13 41 168 40 672 9235 79296 34730 0 960 34770
   3     3   20474842 hda3 41527 32035 588466 207290 2817366 517124 27105688 \
27872922 0 1111380 28677012
   3     4          1 hda4 0 0 0 0 0 0 0 0 0 0 0
   3     5   10241406 hda5 18818 176678 1563554 98970 12679 30841 348376 21543950 0 \
236860 21642920
   3     6   23591421 hda6 37409 47602 679506 270820 8456738 8968017 139407872 \
10286218 0 1585880 10660468
   3     7   23583388 hda7 350 85 2898 3930 5926 0 47408 22366410 0 195890 22370340
```

```
> more /etc/fstab
LABEL=/                 /                       ext2    defaults        1 1
LABEL=/boot             /boot                   ext2    defaults        1 2
none                    /dev/pts                devpts  gid=5,mode=620  0 0
none                    /proc                   proc    defaults        0 0
none                    /dev/shm                tmpfs   defaults        0 0
LABEL=/usr0             /usr0                   ext2    defaults        1 2
LABEL=/usr1             /usr1                   ext2    defaults        1 2
LABEL=/usr2             /usr2                   ext2    defaults        1 2
/dev/hda2               swap                    swap    defaults        0 0
/dev/fd0                /mnt/floppy             auto    noauto,owner,kudzu 0 0
# /dev/sda1               /usr3                   ext2    defaults        1 2
/dev/sdb1               /usr4                   ext2    defaults        1 2
/dev/sdc1               /usr5                   ext2    defaults        1 2
/dev/cdrom              /mnt/cdrom              udf,iso9660 noauto,owner,kudzu,ro 0 0
```

We store the database objects on `sda1`, `sdb1`, and `sdc1` (3 17GB SCSI disks of model ST318406LW by SEAGATE) and the log files on partition `/` , which is on a Maxtor 6Y080P0 ATA 40GB IDE disk.

C.2 SQL Commands Used in Database Creation

The following SQL commands are used to create the TPC-H database in the case study.

```
CREATE TEMPORARY TABLESPACE TPCDTEMP1
  PAGESIZE 8k
  MANAGED BY system
  USING ('/usr3/mzwang/tpch1_db2_tblsp/tpcdtemp1')
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDNATION
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdnation' 500)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDREGION
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdregion' 500)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDPART
  PAGESIZE 8k
```

```
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdpart' 5000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDSUPPLIER
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdsupplier' 1000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDPARTSUPP
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdpartsupp' 30000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDCUSTOMER
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdcustomer' 10000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDORDER
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdorder' 50000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDLINEITEM
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdlineitem' 200000)
  BUFFERPOOL bp8k;

CREATE REGULAR TABLESPACE TPCDINDEX
  PAGESIZE 8k
  MANAGED BY database
  USING
    (file '/usr3/mzwang/tpch1_db2_tblsp/tpcdindex' 20000)
  BUFFERPOOL bp8k;

commit work;
```

```
CREATE TABLE TPCD.NATION  ( N_NATIONKEY  INTEGER NOT NULL,
                            N_NAME       CHAR(25) NOT NULL,
                            N_REGIONKEY  INTEGER NOT NULL,
                            N_COMMENT    VARCHAR(152))
IN TPCDNATION;

CREATE TABLE TPCD.REGION  ( R_REGIONKEY  INTEGER NOT NULL,
                            R_NAME       CHAR(25) NOT NULL,
                            R_COMMENT    VARCHAR(152))
IN TPCDREGION;

CREATE TABLE TPCD.PART  ( P_PARTKEY     INTEGER NOT NULL,
                          P_NAME        VARCHAR(55) NOT NULL,
                          P_MFGR        CHAR(25) NOT NULL,
                          P_BRAND       CHAR(10) NOT NULL,
                          P_TYPE        VARCHAR(25) NOT NULL,
                          P_SIZE        INTEGER NOT NULL,
                          P_CONTAINER   CHAR(10) NOT NULL,
                          P_RETAILPRICE FLOAT NOT NULL,
                          P_COMMENT     VARCHAR(23) NOT NULL )
IN TPCDPART;

CREATE TABLE TPCD.SUPPLIER ( S_SUPPKEY     INTEGER NOT NULL,
                             S_NAME        CHAR(25) NOT NULL,
                             S_ADDRESS     VARCHAR(40) NOT NULL,
                             S_NATIONKEY   INTEGER NOT NULL,
                             S_PHONE       CHAR(15) NOT NULL,
                             S_ACCTBAL     FLOAT NOT NULL,
                             S_COMMENT     VARCHAR(101) NOT NULL)
IN TPCDSUPPLIER;

CREATE TABLE TPCD.PARTSUPP ( PS_PARTKEY     INTEGER NOT NULL,
                             PS_SUPPKEY     INTEGER NOT NULL,
                             PS_AVAILQTY    INTEGER NOT NULL,
                             PS_SUPPLYCOST  FLOAT   NOT NULL,
                             PS_COMMENT     VARCHAR(199) NOT NULL )
IN TPCDPARTSUPP;

CREATE TABLE TPCD.CUSTOMER ( C_CUSTKEY     INTEGER NOT NULL,
                             C_NAME        VARCHAR(25) NOT NULL,
                             C_ADDRESS     VARCHAR(40) NOT NULL,
                             C_NATIONKEY   INTEGER NOT NULL,
                             C_PHONE       CHAR(15) NOT NULL,
                             C_ACCTBAL     FLOAT    NOT NULL,
                             C_MKTSEGMENT  CHAR(10) NOT NULL,
                             C_COMMENT     VARCHAR(117) NOT NULL)
IN TPCDCUSTOMER;

CREATE TABLE TPCD.ORDERS ( O_ORDERKEY       INTEGER NOT NULL,
```

```
                           O_CUSTKEY        INTEGER NOT NULL,
                           O_ORDERSTATUS    CHAR(1) NOT NULL,
                           O_TOTALPRICE     FLOAT NOT NULL,
                           O_ORDERDATE      DATE NOT NULL,
                           O_ORDERPRIORITY  CHAR(15) NOT NULL,
                           O_CLERK          CHAR(15) NOT NULL,
                           O_SHIPPRIORITY   INTEGER NOT NULL,
                           O_COMMENT        VARCHAR(79) NOT NULL)
IN TPCDORDER;

CREATE TABLE TPCD.LINEITEM ( L_ORDERKEY    INTEGER NOT NULL,
                             L_PARTKEY     INTEGER NOT NULL,
                             L_SUPPKEY     INTEGER NOT NULL,
                             L_LINENUMBER  INTEGER NOT NULL,
                             L_QUANTITY    FLOAT NOT NULL,
                             L_EXTENDEDPRICE  FLOAT NOT NULL,
                             L_DISCOUNT    FLOAT NOT NULL,
                             L_TAX         FLOAT NOT NULL,
                             L_RETURNFLAG  CHAR(1) NOT NULL,
                             L_LINESTATUS  CHAR(1) NOT NULL,
                             L_SHIPDATE    DATE NOT NULL,
                             L_COMMITDATE  DATE NOT NULL,
                             L_RECEIPTDATE DATE NOT NULL,
                             L_SHIPINSTRUCT CHAR(25) NOT NULL,
                             L_SHIPMODE     CHAR(10) NOT NULL,
                             L_COMMENT      VARCHAR(44) NOT NULL)
IN TPCDLINEITEM;

COMMIT WORK;

--TERMINATE;

create unique index tpcd.c_ck on tpcd.customer (c_custkey asc) pctfree 0;
create index tpcd.c_nk on tpcd.customer (c_nationkey asc) pctfree 0;

create unique index tpcd.p_pk on tpcd.part (p_partkey asc) pctfree 0;

create unique index tpcd.s_sk on tpcd.supplier (s_suppkey asc) pctfree 0;
create index tpcd.s_nk on tpcd.supplier (s_nationkey asc) pctfree 0;

create index tpcd.ps_pk on tpcd.partsupp (ps_partkey asc) pctfree 0;
create index tpcd.ps_sk on tpcd.partsupp (ps_suppkey asc) pctfree 0;
create unique index tpcd.ps_pk_sk on tpcd.partsupp (ps_partkey asc,
       ps_suppkey asc) pctfree 0;
create unique index tpcd.ps_sk_pk on tpcd.partsupp (ps_suppkey asc,
       ps_partkey asc) pctfree 0;

create unique index tpcd.o_ok on tpcd.orders (o_orderkey asc) pctfree 3;
create index tpcd.o_ck on tpcd.orders (o_custkey asc) pctfree 3;
```

```
create index tpcd.o_od on tpcd.orders (o_orderdate asc) pctfree 3;

create index tpcd.l_ok on tpcd.lineitem (l_orderkey asc) pctfree 3;
create index tpcd.l_pk on tpcd.lineitem (l_partkey asc) pctfree 3;
create index tpcd.l_sk on tpcd.lineitem (l_suppkey asc) pctfree 3;
create index tpcd.l_sd on tpcd.lineitem (l_shipdate asc) pctfree 3;
create index tpcd.l_cd on tpcd.lineitem (l_commitdate asc) pctfree 3;
create index tpcd.l_rd on tpcd.lineitem (l_receiptdate asc) pctfree 3;
create index tpcd.l_pk_sk on tpcd.lineitem (l_partkey asc,
       l_suppkey asc) pctfree 3;
create index tpcd.l_sk_pk on tpcd.lineitem (l_suppkey asc,
       l_partkey asc) pctfree 3;

create unique index tpcd.n_nk on tpcd.nation (n_nationkey asc) pctfree 0;
create index tpcd.n_rk on tpcd.nation (n_regionkey asc) pctfree 0;

create unique index tpcd.r_rk on tpcd.region (r_regionkey asc) pctfree 0;
```

C.3 Database Configurations

We list the database manager and database configuration below. The configurations are reported by the "`get database manager configuration`" and "`get database configuration`" commands.

```
          Database Manager Configuration

     Node type = Enterprise Server Edition with local and remote clients

 Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 3.306410e-07
 Communications bandwidth (MB/sec)      (COMM_BANDWIDTH) = 1.000000e+02

 Max number of concurrently active databases     (NUMDB) = 8
 Data Links support                          (DATALINKS) = NO
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) = /opt/IBMJava2-131

 Diagnostic error capture level              (DIAGLEVEL) = 3
 Notify Level                              (NOTIFYLEVEL) = 3
 Diagnostic data directory path               (DIAGPATH) =
```

```
 Default database monitor switches
   Buffer pool                         (DFT_MON_BUFPOOL) = OFF
   Lock                                   (DFT_MON_LOCK) = OFF
   Sort                                   (DFT_MON_SORT) = OFF
   Statement                              (DFT_MON_STMT) = OFF
   Table                                 (DFT_MON_TABLE) = OFF
   Timestamp                         (DFT_MON_TIMESTAMP) = ON
   Unit of work                            (DFT_MON_UOW) = OFF
 Monitor health of instance and databases   (HEALTH_MON) = ON

 SYSADM group name                        (SYSADM_GROUP) =
 SYSCTRL group name                      (SYSCTRL_GROUP) =
 SYSMAINT group name                    (SYSMAINT_GROUP) =
 SYSMON group name                        (SYSMON_GROUP) =

 Client Userid-Password Plugin          (CLNT_PW_PLUGIN) =
 Client Kerberos Plugin                (CLNT_KRB_PLUGIN) =
Group Plugin                             (GROUP_PLUGIN) =
 GSS Plugin for Local Authorization    (LOCAL_GSSPLUGIN) =
 Server Plugin Mode                    (SRV_PLUGIN_MODE) = UNFENCED
 Server List of GSS Plugins      (SRVCON_GSSPLUGIN_LIST) =
 Server Userid-Password Plugin        (SRVCON_PW_PLUGIN) =
 Server Connection Authentication          (SRVCON_AUTH) = NOT_SPECIFIED
 Database manager authentication        (AUTHENTICATION) = SERVER
 Cataloging allowed without authority   (CATALOG_NOAUTH) = NO
 Trust all clients                      (TRUST_ALLCLNTS) = YES
 Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT
 Bypass federated authentication            (FED_NOAUTH) = NO

 Default database path                       (DFTDBPATH) = /home/mzwang

 Database monitor heap size (4KB)          (MON_HEAP_SZ) = 90
 Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 512
 Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0
 Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC
 Backup buffer default size (4KB)            (BACKBUFSZ) = 1024
 Restore buffer default size (4KB)           (RESTBUFSZ) = 1024

 Sort heap threshold (4KB)                  (SHEAPTHRES) = 8000

 Directory cache support                     (DIR_CACHE) = YES

 Application support layer heap size (4KB)   (ASLHEAPSZ) = 15
 Max requester I/O block size (bytes)         (RQRIOBLK) = 32767
 Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000

 Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 10
```

```
Priority of agents                           (AGENTPRI) = SYSTEM
Max number of existing agents               (MAXAGENTS) = 400
Agent pool size                        (NUM_POOLAGENTS) = 200(calculated)
Initial number of agents in pool       (NUM_INITAGENTS) = 0
Max number of coordinating agents     (MAX_COORDAGENTS) = (MAXAGENTS - NUM_INITAGENTS)
Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections      (MAX_CONNECTIONS) = MAX_COORDAGENTS

Keep fenced process                        (KEEPFENCED) = YES
Number of pooled fenced processes         (FENCED_POOL) = MAX_COORDAGENTS
Initial number of fenced processes     (NUM_INITFENCED) = 0

Index re-creation time and redo index build  (INDEXREC) = RESTART

Transaction manager database name         (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec)     (RESYNC_INTERVAL) = 180

SPM name                                     (SPM_NAME) =
SPM log size                          (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit                 (SPM_MAX_RESYNC) = 20
SPM log path                             (SPM_LOG_PATH) =

TCP/IP Service name                          (SVCENAME) = tpch1
Discovery mode                               (DISCOVER) = SEARCH
Discover server instance                (DISCOVER_INST) = ENABLE

Maximum query degree of parallelism   (MAX_QUERYDEGREE) = 1
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO

No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = 4096
Number of FCM request blocks              (FCM_NUM_RQB) = AUTOMATIC
Number of FCM connection entries      (FCM_NUM_CONNECT) = AUTOMATIC
Number of FCM message anchors         (FCM_NUM_ANCHORS) = AUTOMATIC

Node connection elapse time (sec)         (CONN_ELAPSE) = 10
Max number of node connection retries (MAX_CONNRETRIES) = 5
Max time difference between nodes (min) (MAX_TIME_DIFF) = 60

db2start/db2stop timeout (min)        (START_STOP_TIME) = 10

      Database Configuration for Database tpcd

Database configuration release level                    = 0x0a00
Database release level                                  = 0x0a00
Database territory                                      = US
Database code page                                      = 819
Database code set                                       = ISO8859-1
Database country/region code                            = 1
Database collating sequence                             = BINARY
```

```
Alternate collating sequence              (ALT_COLLATE) =

Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE

Discovery support for this database       (DISCOVER_DB) = ENABLE

Default query optimization class         (DFT_QUERYOPT) = 5
Degree of parallelism                      (DFT_DEGREE) = 1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
Default refresh age                   (DFT_REFRESH_AGE) = 0
Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM
Number of frequent values retained     (NUM_FREQVALUES) = 10
Number of quantiles retained            (NUM_QUANTILES) = 20

Backup pending                                          = NO

Database is consistent                                  = YES
Rollforward pending                                     = NO
Restore pending                                         = NO

Multi-page file allocation enabled                      = YES

Log retain for recovery status                          = NO
User exit for logging status                            = NO

Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60
Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60
Data Links Number of Copies             (DL_NUM_COPIES) = 1
Data Links Time after Drop (days)        (DL_TIME_DROP) = 1
Data Links Token in Uppercase                (DL_UPPER) = NO
Data Links Token Algorithm                   (DL_TOKEN) = MAC0

Database heap (4KB)                            (DBHEAP) = 1024
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = (MAXAPPLS*4)
Log buffer size (4KB)                        (LOGBUFSZ) = 128
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000
Buffer pool size (pages)                     (BUFFPAGE) = 64000
Extended storage segments size (4KB)    (ESTORE_SEG_SZ) = 16000
Number of extended storage segments   (NUM_ESTORE_SEGS) = 0
Max storage for lock list (4KB)              (LOCKLIST) = 100

Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000
Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128

Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB)                         (SORTHEAP) = 25600
SQL statement heap (4KB)                     (STMTHEAP) = 1024
```

```
 Default application heap (4KB)             (APPLHEAPSZ) = 512
 Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8)
 Statistics heap size (4KB)               (STAT_HEAP_SZ) = 4384

 Interval for checking deadlock (ms)         (DLCHKTIME) = 10000
 Percent. of lock lists per application       (MAXLOCKS) = 10
 Lock timeout (sec)                        (LOCKTIMEOUT) = -1

 Changed pages threshold                (CHNGPGS_THRESH) = 60
 Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1
 Number of I/O servers                   (NUM_IOSERVERS) = 5
 Index sort flag                             (INDEXSORT) = YES
 Sequential detect flag                      (SEQDETECT) = YES
 Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC

 Track modified pages                         (TRACKMOD) = OFF

 Default number of containers                            = 1
 Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32

 Max number of active applications            (MAXAPPLS) = AUTOMATIC
 Average number of active applications       (AVG_APPLS) = 1
 Max DB files open per application            (MAXFILOP) = 64

 Log file size (4KB)                         (LOGFILSIZ) = 1000
 Number of primary log files                (LOGPRIMARY) = 5
 Number of secondary log files               (LOGSECOND) = 20
 Changed path to log files                  (NEWLOGPATH) =
 Path to log files                                       = /home/mzwang/mzwang\
/NODE0000/SQL00001/SQLOGDIR/
 Overflow log path                     (OVERFLOWLOGPATH) =
 Mirror log path                         (MIRRORLOGPATH) =
 First active log file                                   =
 Block log on disk full                (BLK_LOG_DSK_FUL) = NO
 Percent of max active log space by transaction(MAX_LOG) = 0
 Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0

 Group commit count                          (MINCOMMIT) = 1
 Percent log file reclaimed before soft chckpt (SOFTMAX) = 100
 Log retain for recovery enabled             (LOGRETAIN) = OFF
 User exit for logging enabled                (USEREXIT) = OFF

 HADR database role                                      = STANDARD
 HADR local host name                  (HADR_LOCAL_HOST) =
 HADR local service name                (HADR_LOCAL_SVC) =
 HADR remote host name                (HADR_REMOTE_HOST) =
 HADR remote service name              (HADR_REMOTE_SVC) =
 HADR instance name of remote server  (HADR_REMOTE_INST) =
 HADR timeout value                       (HADR_TIMEOUT) = 120
```

```
HADR log write synchronization mode     (HADR_SYNCMODE) = NEARSYNC

First log archive method                 (LOGARCHMETH1) = OFF
Options for logarchmeth1                  (LOGARCHOPT1) =
Second log archive method                (LOGARCHMETH2) = OFF
Options for logarchmeth2                  (LOGARCHOPT2) =
Failover log archive path                (FAILARCHPATH) =
Number of log archive retries on error   (NUMARCHRETRY) = 5
Log archive retry Delay (secs)         (ARCHRETRYDELAY) = 20
Vendor options                              (VENDOROPT) =

Auto restart enabled                      (AUTORESTART) = ON
Index re-creation time and redo index build  (INDEXREC) = SYSTEM (RESTART)
Log pages during index build            (LOGINDEXBUILD) = OFF
Default number of loadrec sessions    (DFT_LOADREC_SES) = 1
Number of database backups to retain   (NUM_DB_BACKUPS) = 12
Recovery history retention (days)     (REC_HIS_RETENTN) = 366

TSM management class                    (TSM_MGMTCLASS) =
TSM node name                            (TSM_NODENAME) =
TSM owner                                   (TSM_OWNER) =
TSM password                             (TSM_PASSWORD) =

Automatic maintenance                      (AUTO_MAINT) = OFF
  Automatic database backup            (AUTO_DB_BACKUP) = OFF
  Automatic table maintenance          (AUTO_TBL_MAINT) = OFF
    Automatic runstats                  (AUTO_RUNSTATS) = OFF
    Automatic statistics profiling    (AUTO_STATS_PROF) = OFF
      Automatic profile updates         (AUTO_PROF_UPD) = OFF
    Automatic reorganization               (AUTO_REORG) = OFF
```

Bibliography

[1] A. Adas and A. Mukherjee. On resource management and QoS guarantees for long range dependent traffic. In *The IEEE Conference on Computer Communications (INFOCOM), Boston, MA*, pages 779–787. IEEE Computer Society, April 1995.

[2] R. G. Addie, M. Zukerman, and T. Neame. Fractal traffic: measurements, modeling, and performance evaluation. In *The IEEE Conference on Computer Communications (INFOCOM), Boston, MA*, pages 977–984. IEEE Computer Society, April 1995.

[3] Sanjay Agrawal, Surajit Chaudhuri, Abhinandan Das, and Vivek Narasayya. Automating layout of relational databases. In *The 19th International Conference on Data Engineering (ICDE), Bangalore, India*, pages 607–618. IEEE Computer Society, March 2003.

[4] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated selection of materialized views and indexes for SQL databases. In *Proceeding of International Conference on Very Large Data Bases (VLDB), Cairo, Egypt*, pages 496–505. Morgan Kaufmann Publishers Inc., San Francisco, CA, September 2000.

[5] Sedat Akyürek and Kenneth Salem. Adaptive block rearrangement. *ACM Transactions on Computer Systems (TOCS)*, 13(2):89–121, May 1995.

[6] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS), Miami Beach, FL*, pages 92–103. IEEE Computer Society, December 1996.

[7] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems (TOCS)*, 19(4):483–518, November 2001.

[8] 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 Laboratories, Palo Alto, CA, June 2002. Available at http://hpl.hp.com/research/ssp/papers/ergastulum-paper.pdf (last accessed on September 12, 2005).

[9] Eric Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-4, HP Laboratories, Palo Alto, CA, July 2001. Available at http://www.hpl.hp.com/research/ssp/papers/HPL-SSP-2001-04.pdf (last accessed on September 12, 2005).

[10] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: Running circles around storage administration. In *Proceedings of the 1st USENIX Conference on File And Storage Technologies (FAST'02), Monterey, CA*, pages 175–188, January 2002.

[11] Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, and Ram Swaminathan. Buttress: A toolkit for flexible and high fidelity I/O benchmarking. In *Proceedings of the 3rd USENIX Conference on File And Storage Technologies (FAST'04), San Fransisco, CA*, pages 45–58. USENIX, Berkeley, CA, April 2004.

[12] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Marina Del Rey, CA*, pages 55–65. ACM Press, June 2002.

[13] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles (SOSP), Pacific Grove, CA*, pages 198–212. ACM Press, October 1991.

[14] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Madison, WI*, pages 151–160. ACM Press, June 1998.

[15] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA'98), Barcelona, Spain*, pages 3–14. IEEE Computer Society, June 1998.

[16] Richard Ernest Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, 1961.

[17] Mohamed N. Bennani and Daniel A. Menascé. Resource allocation for autonomic data centers using analytic performance models. In *The 2nd IEEE International Conference on Autonomic Computing (ICAC-05), Seattle, WA*, pages 229–240. IEEE Computer Society, June 2005.

[18] Jan Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, New York, NY, 1994.

[19] E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes. Capacity planning with phased workloads. In *Proc. of the 1st Workshop on Software and Performance (WOSP), Santa Fe, NW*, pages 199–207. ACM Press, October 1998.

[20] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[21] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, January 1 1984.

[22] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *The IEEE Conference on Computer Communications (INFOCOM), New York, NY*, pages 126–134. IEEE Computer Society, March 1999.

[23] Aaron B. Brown, Joseph Hellerstein, Matt Hogstrom, Tony Lau, Sam Lightstone, Peter Shum, and Mary Peterson Yost. Benchmarking autonomic capabilities: Promises and pitfalls. In *The 1st International Conference on Autonomic Computing (ICAC-04), New York, NY*, pages 266–267, May 2004.

[24] John S. Bucy, Gregory R. Ganger, and Contributors. The DiskSim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, Pittsburgh, PA, January 2003. Code and reference manual available at http://www.pdl.cmu.edu/DiskSim/.

[25] Christopher J. C. Burges. A tutorial on Support Vector Machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, June 1998.

[26] Scott D. Carson. A system for adaptive disk rearrangement. *Software – practice and experience*, 20(3):225–242, March 1990.

[27] Surajit Chaudhuri, Benoît Dageville, and Guy M. Lohman. Self-managing technology in database management systems. In *Tutorial on International Conference on Very Large Data Bases (VLDB), Toronto, CA*, page 1243. Morgan Kaufmann Publishers Inc., San Francisco, CA, August 2004.

[28] Surajit Chaudhuri and Gerhard Weikum. Rethinking database system architecture: towards a self-tuning RISC-style database system. In *Proceeding of International Conference on Very Large Data Bases (VLDB), Cairo, Egypt*, pages 1–10. Morgan Kaufmann Publishers Inc., San Francisco, DA, September 2000.

[29] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (SCUR)*, 26(2):145–185, June 1994.

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

[31] Trishul M. Chilimbi. Efficient representations and abstractions for quantifying and exploiting data reference locality. In *Proceeding of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation (PLDI), Snowbird, UT*, pages 191–202. ACM Press, June 2001.

[32] Hewlett-Packard Company. Adaptive enterprise. Available at http://www.hp.com/products1/promos/adaptive_enterprise/us/adaptive_enterprise.html (last accessed on September 12, 2005).

[33] Robert B. Cooper. *Introduction to queueing theory (2nd edition).* North-Holland (Elservier), 1981.

[34] Storage Performance Council and University of Massachusetts. UMass trace repository. Traces available at http://signl.cs.umass.edu/repository/walk.php?cat=Storage (last accessed on September 12, 2005).

[35] Transaction Processing Performance Council. TPC benchmark H (TPC-H). Specifications available at http://www.tpc.org/tpch/default.asp (last accessed on September 12, 2005).

[36] Mark E. Crovella and Azer Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Philadelphia, PA*, pages 160–169. ACM Press, May 1996.

[37] Belur V. Dasarathy, editor. *Neareast neighbor pattern classification techniques (Nn Norms: Nn Pattern Classification Techniques).* IEEE Computer Society Press, December 1990.

[38] Timothy E. Denehy, John Bent, Florentina Popovici, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Deconstructing storage arrays. In *Proceeding of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA*, pages 59–71. ACM Press, October 2004.

[39] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital control of dynamic systems.* Prentice Hall, December 1998.

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

[41] 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, Pittburgh, PA, August 2003. Available at http://www.pdl.cmu.edu/PDL-FTP/SelfStar/selfstar.pdf (last accessed on September 12, 2005).

[42] Mark W. Garrett and Walter Willinger. Analysis, modeling, and generation of self-similar VBR video traffic. In *Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications, London, United Kingdom*, pages 269–280. ACM Press, August 1994.

[43] María E. Gómez and Vicente Santonja. Analysis of self-similarity in I/O workload using structural modeling. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOT'99), College Park, Maryland*, pages 234–243. IEEE Computer Society, October 1999.

[44] 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 (MASCOT'00), San Francisco, CA*, pages 199–206. IEEE Computer Society, August 2000.

[45] Steven D. Gribble, Gurmeet Singh Manku, Drew Roselli, Eric A. Brewer, Timothy J. Gibson, and Ethan L. Miller. Self-similarity in file systems. In *Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Madison, WI*, pages 141–150. ACM Press, June 1998.

[46] Matthias Grossglauser and Jean-Chrysostome Bolot. On the relevance of long-range dependence in network traffic. *IEEE/ACM Transactions on Networking (TON)*, 7(5):629–640, October 1999.

[47] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York, NY, 2001.

[48] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback control of computing systems*. Wiley-IEEE Press, Hoboken, NJ, August 2004.

[49] Bo Hong and Tara Madhyastha. The relevance of long-range dependence in disk traffic and implications for trace synthesis. In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005), Monterey, CA*, pages 316–326. IEEE Computer Society, April 2002.

[50] IBM. Autonomic computing: IBM's perspective on the state of information technology. Available at http://www.research.ibm.com/autonomic/ (last accessed on September 12, 2005).

[51] William V. Courtright II, Garth Gibson, Mark Holland, and Jim Zelenka. A structured approach to redundant disk array implementation. In *Proceedings of IEEE International Computer Performance and Dependability Symposium (IPDS), Urbana, IL*, page 11. IEEE Computer Society, September 1996.

[52] Microsoft Inc. AutoAdmin project at Microsoft Research. Available at http://research.microsoft.com/dmx/AutoAdmin/, (last accessed on September 12, 2005).

[53] Microsoft Inc. Dynamic systems initiative. Available at http://www.microsoft.com/windowsserversystem/dsi/default.mspx (last accessed September 12, 2005).

[54] David M. Jacobson and John Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7, HP Laboratories, Palo Alto, CA, May 1991. Available at http://www.hpl.hp.com/research/ssp/papers/HPL-CSP-91-7rev1.pdf, (last accessed on September 12, 2005).

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

[56] Abhinav Kamra, Vishal Misra, and Erich M. Nahum. Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In *International workshop on Quality of Service (IWQoS), Montreal, Canada*, pages 47–56. IEEE Computer Society, June 2004.

[57] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: performance isolation and differentiation for storage systems. In *International workshop on Quality of Service (IWQoS), Montreal, Canada*, pages 67–74. IEEE Computer Society, June 2004.

[58] K. Keeton, D. A. Patternson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a quad pentium pro SMP using OLTP workloads. In *Proceedings of the 25th International Symposium on Computer Architecture (ISCA'98), Barcelona, Spain*, pages 15–26. IEEE Computer Society, June 1998.

[59] Kimberly Keeton, Guillermo A. Alvarez, Erik Riedel, and Mustafa Uysal. Characterizing I/O-intensive workload sequentiality on modern disk arrays. In *The Fourth Workshop On Computer Architecture Evaluation Using Commercial Workloads (CAECW-2001), Monterrey, Mexico*, January 2001.

[60] Terence Kelly, Ira Cohen, Moises Goldszmidt, and Kimberly Keeton. Inducing models of black-box storage arrays. Technical Report HPL-2004-108, HP Laboratories, Palo Alto, CA, June 2004. Available at http://www.hpl.hp.com/techreports/2004/HPL-2004-108.html, (last accessed on September 12, 2005).

[61] Michelle Y. Kim and Asser N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Transactions on Computers*, 40(7):801–810, July 1991.

[62] A. Kuratti. Performance analysis of the RAID5 disk array. In *Proceedings of the IEEE International Computer Performance and Dependability Symposium (IPDS'95), Erlangen, Germany*, pages 236–245. IEEE Computer Society, April 1995.

[63] Zachary Kurmas and Kimberly Keeton. Using the Distiller to direct the development of self-configuration software. In *Proceedings of the International Conference on Automatic Computing (ICAC-04), New York, NY*, pages 172–179. IEEE Computer Society, May 2004.

[64] Zachary Kurmas, Kimberly Keeton, and Ralph Becker-Szendy. Iterative development of an I/O workload characterization. In *4th workshop on computer architecture evaluation using commercial workloads (CAECW), Monterrey, Mexico*, January 2001.

[65] Zachary Kurmas, Kimberly Keeton, and Kenneth Mackenzie. Synthesizing representative I/O workloads using iterative distillation. In *Proceedings of the 11th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Orlando, FL*, pages 6–15. IEEE Computer Society, October 2003.

[66] Edward K. Lee and Randy H. Katz. An analytic performance model of disk arrays. In *Proceedings of the 1993 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA*, pages 98–109, May 1993.

[67] Mong Li Lee, Masaru Kitsuregawa, Beng Chin Ooi, Kian-Lee Tan, and Anirban Mondal. Towards self-tuning data placement in parallel database systems. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas, Texas*, pages 225–236. ACM Press, May 2000.

[68] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In *Proceedings of the ACM SIGCOMM'93 Conference on Communications Architectures, Protocols and Applications, San Francisco, CA*, pages 183–193. ACM Press, September 1993.

[69] N. Likhanov and B. Tsybakov. Analysis of an ATM buffer with self-similar ("fractal") input traffic. In *The IEEE Conference on Computer Communications (INFOCOM), Boston, MA*, pages 985–992. IEEE Computer Society, April 1995.

[70] Christopher R. Lumb, Jiri Schindler, and Gregory R. Ganger. Freeblock scheduling outside of disk firmware. In *Proceedings of the 1st USENIX Conference on File And Storage Technologies (FAST'02), Monterey, CA*, pages 275–288. USENIX, Janurary 2002.

[71] Brian Melcher and Bradley Mitchell. Towards an autonomic framework: Self-configuring network services and developing autonomic applications. *Intel Technology Journal*, 8(4), November 2004. Available at http://developer.intel.com/technology/itj/2004/volume08issue04/art03_autonomic/p01_abstract.htm (last accessed on September 12, 2005).

[72] D.A. Menascé and M.N.Bennani. On the use of performance models to design self-managing computer systems. In *Proceedings of the Computer Measurement Group Conference (CMG), Dallas, TX*, pages 7–12. Computer Measurement Group, December 2003.

[73] J. Menon. Performance of RAID 5 disk arryas with read and write caching. *Distributed and Parallel Databases*, 2(3):115–129, 1994.

[74] Arif Merchant and Guillermo A. Alvarez. Disk array models in Minerva. Technical Report HPL-2001-118, HP Laboratories, Palo Alto, CA, May 2001. Available at http://www.hpl.hp.com/techreports/2001/HPL-2001-118.html, last accessed on September 12, 2005).

[75] Arif Merchant and Philip S.Yu. An analytical model for reconstruction time in mirrored disks. *Performance evaluation*, 20(1-3):115–129, 1994.

[76] Arif Merchant and Philip S.Yu. Analytic modeling and comparison of striping strategies for replicated disk arrays. *IEEE Transactions on Computers*, 44(3):419–433, 1995.

[77] Arif Merchant and Philip S. Yu. Analytical modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Transactions on Computers*, 45(3):367–373, 1996.

[78] Ethan L. Miller and Randy H. Katz. Input/output behavior of supercomputer applications. In *Proceedings of ACM/IEEE Supercomputing, Albuquerque, New Mexico*, pages 567–576. ACM Press, November 1991.

[79] Philippe Nain. Impact of bursty traffic on queues. *Statistical Inference for Stochastic Processes*, 5(3):307–320, 2002.

[80] Ilkka Norros. A storage model with self-similar input. *Queueing Systems*, 16:387–396, 1994.

[81] Oracle Corporation. *Oracle Database 10g*. Introduction available at http://www.oracle.com/technology/software/products/database/oracle10g/index.html.

[82] John K. Ousterhout, Hervé Da Costa, David Harrison, John A. Kunze, Michael D. Kupfer, and James G. Thompson. A trace-driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the 10th ACM Symposium on Operating System Principles (SOSP'85), Orcas Island, WA*, pages 15–24. ACM Press, December 1985.

[83] Stratos Papadomanolakis and Anastassia Ailamaki. AutoPart: Automating schema design for large scientific databases using data partitioning. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM), Santorini Island, Greece*, pages 383–392. IEEE Computer Society, June 2004.

[84] David A. Pattern, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disk (RAID). In *Proceeding of the 1988 SIGMOD International Conference on Management of Data, Chicago, IL*, pages 109–116. ACM Press, June 1988.

[85] Vern Paxson. Fast, approximate synthesis of fractional Gaussian noise for generating self-similar network traffic. *ACM SIGCOMM Computer communication review*, 27(5):5–18, October 1997.

[86] Brigham Young Univeristy Performance Evaluation Laboratory. Dtb trace format specification. Available at http://traces.byu.edu/new/Documentation/ (last accessed on October 10, 2005).

[87] K. K. Ramakrishnan, Prabuddha Biswas, and Ramakrishna Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of the 1992 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Newport, RI*, pages 78–90. ACM Press, June 1992.

[88] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy Lohman. Automating physical database design in a parallel database. In *Proceeding of the 2002 SIGMOD International Conference on Management of Data, Madison, WI*, pages 558–569. ACM Press, June 2002.

[89] Gartner Group/Dataquest report. Server storage and RAID worldwide, 1999.

[90] ITCentrix report. Storage on tap: Understanding the business value of storage service providers, 2001.

[91] Rudolf 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.

[92] B. D. Ripley. *Pattern recognitions and neural networks.* Cambridge University Press, 1996.

[93] Chris Ruemmler and John Wilkes. Disk shuffling. Technical Report HPL-91-156, HP Laboratories, Palo Alto, CA, October 1991. Available at http://www.hpl.hp.com/research/ssp/papers/HPL-91-156.pdf (last accessed on September 12, 2005),.

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

[95] Jiri Schindler and Gregory R. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, Carnegie Mellon University, December 1999. Available at http://www.pdl.cmu.edu/Dixtrac/ (last accessed on September 12, 2005),.

[96] George A. F. Seber. *Multivariate observations*. John Wiley & Sons Inc. New York, NY, 1984.

[97] Margo Seltzer, Peter Chen, and John. Ousterhout. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Technical Conference, Washington, DC*, pages 313–324. USENIX Association, January 1990.

[98] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1963.

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

[100] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's LEarning Optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Rome, Italy*, pages 19–28. Morgan Kaufmann Publishers Inc., September 2001.

[101] John D. Strunk and Gregory R. Ganger. A human organization analogy for Self-* systems. In *First Workshop on Algorithms and Architectures for Self-Managing Systems, in conjunction with Federated Computing Research Conference*, pages 1–6, 2003.

[102] David G. Sullivan and Margo Seltzer. A resource management framework for central servers. In *200 USENIX Annual Technical Conference, San Diego, CA*, pages 337–350. USENIX Association, 2000.

[103] Nisha Talagala, Remzi H. Arpaci-Dusseau, and David Patterson. Microbenchmark-based extraction of local and global disk characteristics. Technical Report CSD-99-1063, University of California, Berkeley, 1999. Available at http://www.cs.wisc.edu/ remzi/Postscript/disk.pdf (last accessed on September 12, 2005).

[104] Chandramohan A. Thekkath, John Wilkes, and Edward D. Lazowska. Techniques for file system simulation. *Software-Practice and Experience*, 24(11):981–999, November 1994.

[105] Eno Thereska, Dushyanth Narayanan, and Gregory R. Ganger. Towards self-predicting systems: What if you could ask "what-if"? August 2005.

[106] Alexander Thomasian and Jai Menon. Performance analysis of RAID 5 disk arrays with a vacationing server model for rebuild mode operation. In *Proceedings of the 10th International Conference on Data Engineering (ICDE'94), Houston, TX*, pages 111–119. IEEE Computer Society, February 1994.

[107] Alexander Thomasian and Jai Menon. RAID 5 performance with distributed sparing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):640–657, June 1997.

[108] Steven K. Thompson and George A. F. Seber. *Adaptive sampling*. John Wiley and Sons, New York, NY, June 1996.

[109] Nancy Tran and Daniel A. Reed. Automatic ARIMA time series modeling for adaptive I/O prefetching. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):362–377, April 2004.

[110] 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 (MASCOTS'01), Cincinnati, OH*, pages 183–192. IEEE Computer Society, August 2001.

[111] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohnman, and Alan Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *The 16th International Conference on Data Engineering (ICDE'00), San Diego, CA*. IEEE Computer Society, February 2000.

[112] Elizabeth Varki, Arif Merchant, Juanzhang Xu, and Xiaozhou Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):559–574, June 2004.

[113] Alistair Veitch and Kim Keeton. The Rubicon workload characterization tool. Technical Report HPL-SSP-2003-13, HP Laboratories SSP technical report, Palo Alto, CA, March 2003. Available at http://www.hpl.hp.com/research/ssp/papers/rubicon-tr.pdf (last accessed on September 12, 2005).

[114] Carl A. Waldspurger. Memory resource management in VMWare ESX server. In *Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA*, pages 181–194. USENIX Association, December 2002.

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

[116] Mengzhi Wang, Kinman Au, Anastassia Ailamaki, Anthony Brockwell, Christos Faloutsos, and Gregory R. Ganger. Storage device performance prediction with CART models. Technical Report CMU-PDL-04-103, Carnegie Mellon University, Pittsburgh, PA, 2004. Available at http://www.pdl.cmu.edu/Workload/index.html (Last accessed on September 12, 2005).

[117] Mengzhi Wang, Tara Madhyastha, Ngai Hang Chan, Spiros Papadimitrioun, and Christos Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *The 18th Internal Conference on Data Engineering (ICDE'02), San Jose, CA*, pages 507–516. IEEE Computer Society, March 2002.

[118] Jonathan Wildstrom, Peter Stone, Emmett Witchel, Raymond J. Mooney, and Mike Dahlin. Towards self-configuring hardware for distributed computer systems. In *The 2nd International Conference on Autonomic Computing (ICAC-05), Seattle, WA*, pages 241–249. IEEE Computer Society, June 2005.

[119] John Wilkes. The Pantheon storage-system simulator. Technical Report HPL-SSP-95-14, HP Laboratories, Palo Alto, CA, 1995. Available at http://www.hpl.hp.com/research/ssp/papers/PantheonOverview.pdf (last accessed on September 12, 2005).

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

[121] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID heirarchical storage system. *IEEE Transaction on computer systems*, 14(1):108–136, February 1996.

[122] Walter Willinger, Vern Paxson, and Murad S. Taqqu. Self-similarity and heavy tails: structural modeling of network traffic. In R. Alder, R. Feldman, and M. S. Taqqu, editors, *A practical guide to heavy tails: statistical techniques and applications*, pages 27–53. Birkhauser Boston Inc., 1998.

[123] Bruce L. Worthington, Gregory R. Ganger, and Yale N. Patt. Scheduling algorithms for modern disk drivers. In *Proceedings of the 1994 ACM SIGMETRIC International Conference on Measurement and Modeling of Computer Systems, Nashville, Tennessee*, pages 241–251. ACM Press, May 1994.