Automatic Generation of Behavioral Hard Disk Drive Access Time Models

Adam Crume*, Carlos Maltzahn*, Lee Ward†, Thomas Kroeger†, Matthew Curry†
*University of California, Santa Cruz
{adamcrume,carlosm}@cs.ucsc.edu
†Sandia National Laboratories
{lee,tmkroeg,mlcurry}@sandia.gov

***Abstract*—Predicting access times is a crucial part of predicting hard disk drive performance. Existing approaches use white-box modeling and require intimate knowledge of the internal layout of the drive, which can take months to extract. Automatically learning this behavior is a much more desirable approach, requiring less expert knowledge, fewer assumptions, and less time. While previous research has created black-box models of hard disk drive performance, none have shown low per-request errors. A barrier to machine learning of access times has been the existence of periodic behavior with high, unknown frequencies. We identify these high frequencies with Fourier analysis and include them explicitly as input to the model. In this paper we focus on the simulation of access times for random read workloads within a single zone. We are able to automatically generate and tune request-level access time models with mean absolute error less than $0.15\,\mathrm{ms}$. To our knowledge this is the first time such a fidelity has been achieved with modern disk drives using machine learning. We are confident that our approach forms the core for automatic generation of access time models that include other workloads and span across entire disk drives, but more work remains.**

I. Introduction

Hard disk drive performance models are often used as part of a much larger system simulation [1], for parallel file system simulation [2], for storage configuration [3], and for data placement [4]. The most accurate models are white-box models such as DiskSim [5], which is widely used [6], [7], [2]. These models require extensive parameterization.

Other researchers have noted the difficulty in parameterizing DiskSim [8], [9]. Many researchers use outdated disk models that come with DiskSim such as the Quantum Atlas 10K [10], [11], [12], [13], which was released in 1999 [12]; the Seagate Cheetah 9LP ST39102LW [11], [14], [15], [16], which appears to have been released in 1998 [17]; or the Seagate Barracuda 4LP ST32171 [7], which appears to have been released between 1996 and 1998 [18]. All of these papers were published in 2010 or later, meaning that they all used models that were at least 11 years old at the time of their publication. The capacities of these devices were between 2.1 GB and 9.2 GB, with sustained data transfer rates from 13 MB/s to 41 MB/s [19], [18], [17]. On the other hand, top of the line models released in 2010, such as the WD Caviar Green 3TB [20] or the Seagate 3TB FreeAgent GoFlex Desk [21], had capacities of around 3000 GB and sustained data transfer rates of around 123 MB/s [22] (sizes measured using 1 GB = 10^9 bytes). This amounts to a size increase of 326x and a speed increase of 3x. We believe that researchers used the outdated models that came with DiskSim because of the difficulty of obtaining or creating models for newer drives.

Parameterizing white-box models is a long and difficult process, because manufacturers do not release details such as sector layout required by white-box models. In fact, a fellow researcher (Ron Oldfield) was involved with configuring DiskSim to model an existing device, a process that took several months [23]. Tools such as DIG can extract some of this information [24]. Unfortunately, they require assumptions about the internal structure of the hard disk drive. This structure is likely to change in the future due to the introduction of shingled hard disk drives [25], dual-heads per surface [12], or other optimizations, as has happened in the past with Zoned Bit Recording and serpentine layouts. Since manufacturers do not release this information, researchers must reverse-engineer a device before modifying DIG and DiskSim to support the new layout.

Machine learning presents a more desirable approach to generate models that can reproduce the behavior with as few assumptions as possible. Some progress has been made in *behavioral modeling* of hard disk drive performance [26], [10], [27], [7], but none of these can accurately model individual requests.

Access time, which is what we focus on in this paper, has stubbornly resisted efforts to model it. We focus on workloads that read random single sectors, so as to minimize caching and readahead effects. In this scenario, two components contribute to request latency:

1) Queue time — the time the request spends in the device's queue, waiting to be processed. Requests may queue to some maximum depth in the device, and then be serviced out of order to minimize access time.
2) Access time — the time it takes to start reading sector B, given that sector A was just read. This includes seek time, rotational latency, and settle time. Medium and large seeks are relatively easy to model, because the time can be closely approximated by a simple, smooth function of the logical block numbers (LBNs). Settle time can be modeled as a constant and is easily subsumed into the seek model. Small seeks and rotational latency are difficult to model because these are very high-frequency functions in LBN-space.

978-1-4799-5671-5/14/$31.00

After decomposing per-request latency into these two parts, we discard the queue time and focus on the access time, which we approximate as the time the request was completed minus the time the previously completed request was completed.

We experimented with both decision trees and neural nets for access time prediction. One of the complications in these predictions is the existence of unknown, high frequency components caused by the rotational aspect of the drive. Unfortunately, a limitation of traditional neural nets and decision trees is their inability to recognize periodic patterns in the data. This can be seen with the checkerboard problem [28] as well as the two-spirals problem [29], [30]. We overcome this limitation by finding the frequencies of these periodic patterns with Fourier analysis and then feed these into the neural net or decision tree explicitly by augmenting the feature vector.

Neural nets and, to a lesser degree, decision trees, have many *hyperparameters* that must be chosen before training even begins. Hyperparameters (or metaparameters) are values that affect how the learning algorithm behaves and may bias it toward certain solutions. Examples of hyperparameters include layer sizes, learning rate, minimum leaf weight, etc. The optimal values may be problem-specific, and there is no fixed algorithm for finding the optimal values. We use a genetic algorithm to autotune the values of these hyperparameters.

We previously published preliminary results in a workshop paper using a manually tuned neural net [31]. This paper extends those results by automatically tuning the hyperparameters of the neural net and further winnowing the periods to include. The contributions of the work reported here are: 1) a method for finding useful frequencies, and, new in this paper: 2) a method for automatically tuning models that incorporate these frequencies.

This work is a first step toward automated generation of access time models and is not intended to be a finished solution. Further work is needed to expand the scope of this approach.

II. Related work

A. Predicting request latencies

DiskSim [5] is a well-regarded disk model based on discrete event simulation. It has been validated to produce request-level accuracy. However, it is computationally expensive and difficult to configure for modern disks.

Many analytic models exist, including work by Lebrecht, Dingle, and Knottenbelt [32]. Analytic models are relatively easy to understand and extremely fast due to their compact formulae. Unfortunately, they require very detailed expert knowledge to create. Often, they are limited to certain classes of workloads and are not useful alone in generalized contexts. Our approach shares the last two limitations (for now) but does not require domain expertise.

Kelly *et al*. describe a black-box probabilistic model [26] similar to table-based models such as Garcia *et al*. [33]. In this approach, requests are categorized based on features including size, LRU stack distance, number of pending reads, number of pending writes, and some RAID-specific information. Table-based models are limited by the table size. The work by Kelly *et al*. ameliorates this issue by essentially not requiring the entire table to be filled in, but the problem of table size is not fully solved.

Mesnier *et al*. create a model for relative performance of storage devices [34]. Unfortunately, their approach still requires an accurate base model.

Using regression trees to predict response time is a popular approach. Dai *et al*. predict performance with a combination of regression trees and support vector regression [10]. However, their models are workload-specific, and their prediction errors are based on one-second averages rather than per-request latencies. Wang *et al*. also calculate errors based on windows, using one-minute averages in their case [27].

None of the machine-learning-based approaches have shown low per-request errors.

B. Learning periodic functions

Neural nets can be trained on periodic functions in many ways. Some setups predict the value of the function directly from the input. These invariably use a fixed interval with a very small number of oscillations [35], [36], [37], [38], [39]. With this approach, extrapolating beyond the training range leads to poor performance [40]. This approach is infeasible for our problem because the number of oscillations (roughly, the number of tracks) is on the order of a million.

If the function is known to have period p, another approach is to map the input x into the range $(0, p)$ using $x \bmod p$. Common examples include time-of-day or day-of-year inputs for functions that are daily or yearly periodic [41], [42]. An alternative is to map it to $\sin(2\pi x/p)$ and $\cos(2\pi x/p)$ [43]. (This pair of functions has nice properties; they are both continuous and bounded, and weighted sums are equal to other sinusoids with varying phase.) Either way, this approach requires the period to be precisely known ahead of time, and that the input be exactly periodic. Note that such a period cannot be calculated from manufacturer's specifications.

Neurons in a neural net calculate their output by taking a weighted sum of the inputs and applying an *activation function* or *transfer function*, typically $\tanh(x)$ or $\frac{1}{1+e^{-x}}$. Rather than using one of these as the activation function, one may use $\sin(x)$. Unfortunately, $\sin(x)$ does not approach a limit for large x, while $\tanh(x)$ and $\frac{1}{1+e^{-x}}$ do. Lack of a such a limit can lead to instability [44]. Periodic activation functions also introduce many local minima [45].

A powerful tool for time series data is recurrent or delay networks [46], [47]. These feed the network back into itself, so that the network predicts y values from other y values, rather than from x values. This is usually applied to problems with one dimension of recursion, but ours has two, one for the previously accessed sector and one for the current sector to access. With dense data, only one level of recursion is necessary, because the other y values are already known. If the data is sparse, missing values must be recursively computed. Our sampling is (by necessity) so sparse that the recursion would be extremely deep, making computation time impractical.

C. Tuning hyperparameters

Hyperparameters are often tuned using some combination of manual search and grid search [48], [49], [50], [51], although genetic algorithms are sometimes used [52], [53]. Grid search is known to be inefficient if not all hyperparameters have equal importance [51], and manual search is not reproducible.

Autotuning using genetic algorithms has been used successfully for IO workload parameters [54], PID controllers [55], water resource planning and management [56], and neural networks [57], [58].

III. White-box versus black-box models

Modern hard disk drives are complex. A non-exhaustive list of performance-relevant factors includes caching, readahead, writeback, zoned bit recording, cylinder skew, serpentine layouts, sector sparing, bad sector remapping, settle time, and head switch time. Furthermore, new changes will appear in the future that introduce more complexity, such as shingled drives [25] or possibly dual-heads per surface [12]. All of this complexity means that accurate models must be complex and include many parameters.

White-box models are created based on human understanding of the internal construction and behavior of a system. While they can be very accurate for well-understood systems, the amount of expert knowledge required for hard disk drives is a limitation. A white-box model must be modified manually whenever the system changes structurally, which involves writing code for a model such as DiskSim.

Another issue for white-box models is that of parameterization. White-box models of hard disk drives have many important parameters that are not disclosed by the device manufacturers. Many parameters for DiskSim can be discovered by a tool called DIG [24], although this is imperfect, incomplete, and takes a long time. Parameters not discovered by DIG must be estimated by other means, perhaps even trial-and-error. Finding parameters can take a very long time, six months in one case [23]. As mentioned in section I, many recent papers involving DiskSim use models that are over a decade old, presumably because parameterization is so difficult.

Black-box models, on the other hand, do not require human understanding of a system's internals. Therefore, they are useful for systems that are very complex or whose internals are hidden. Another advantage is that a process for creating black-box models can be used for multiple devices, as long as they have the same interface. Specifically, this means that a process for creating black-box models of block storage devices could also be applied to SSDs, RAID arrays, newer generations of hard disk drives, and so on.

Black-box models, because of the flexibility and the automated construction, are clearly desirable for performance modeling of storage devices.

IV. Periodicity

Periodic functions are difficult to learn directly with a generic machine learning algorithm. Examples include the checkerboard problem [28] and the two-spirals problem [59], which are often used to measure the effectiveness of a machine learning algorithm.

The rotational nature of hard disk drives introduces a periodic component into the access time function. This component has large amplitude, meaning it is important to model. It also has high frequency, meaning it is difficult to model directly as a smooth function. This combination implies that we must address this periodic component to achieve good results.

We therefore assume that the access time function may have components that are locally periodic and explicitly incorporate this assumption into our algorithm. This is a fairly reasonable assumption. First of all, as stated above, these functions do have locally periodic components for existing hard drives, due to track lengths and track skews that are constant within a serpentine. Periodicity is likely to occur in other devices as well, because of the benefits of regular repetition in designs. Secondly, as described in section IV-B, the runtime speed and accuracy of the model is unaffected if these functions do not have locally periodic components.

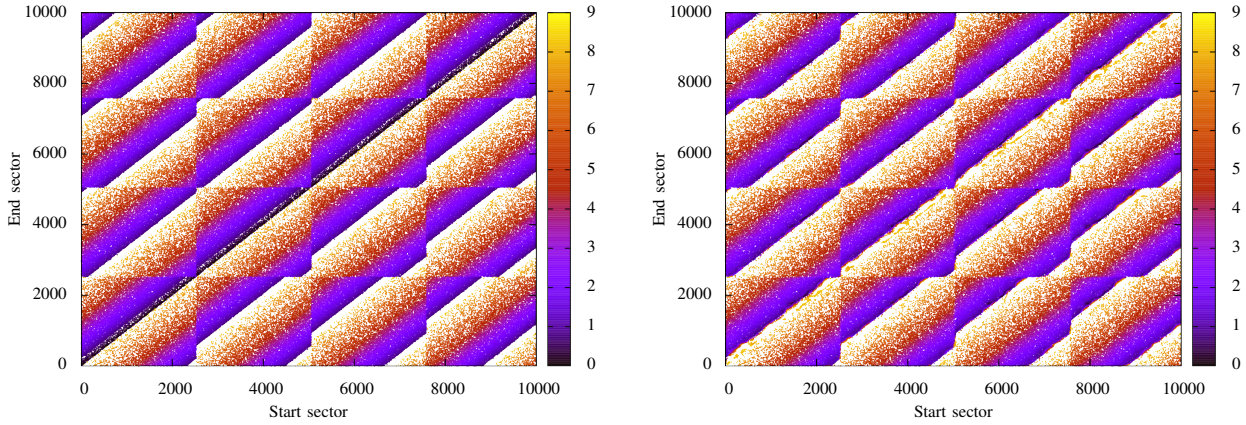
A search through the set of periods with the genetic algorithm used for tuning hyperparameters, while possible, would be extremely inefficient. First, the space is huge. For a device with k sectors (on the order of billions for modern devices), there are roughly $k/2$ frequencies to search through. Second, evaluating a set of frequencies, which involves training the neural net or building the decision tree, is relatively slow. Instead, we compute the Fourier transform of the access time function and use the magnitude of the Fourier transform as a proxy for the importance of a frequency. We set a relatively low bar on the magnitude to generate a set of candidate frequencies which are then filtered by the genetic algorithm (see section VI).

A. Finding strong frequencies

We refer to the previously accessed sector as the *start sector*, which is the current position of the head, and the first sector of the current request as the *end sector*, which will be the new position of the head. Let $f(a, b)$ be the access time function, where a and b are the LBNs of the start and end sectors, and f returns the access time in milliseconds. The Fourier transform is an obvious place to start to find periodic behavior in f. While it can only find periods that are directly correlated to the value of f, it is much faster than training a neural net for every period to see which ones are useful. Furthermore, if many periods are useful, their individual impact on the neural net's accuracy may be obscured by noise. Using the Fourier transform allows us to pinpoint useful periods relatively quickly and with high precision.

Capturing the access time for every sector pair would take a very long time. Given our test device's mean access time of $15.5\,\text{ms}$ and $976{,}773{,}168^2$ pairs of sectors, the time to capture all of them would be roughly 469 million years. Obviously, this is infeasible, so we are limited to an extremely sparse sampling.

The sampling sparsity means that we cannot use the Fast Fourier Transform, so we fall back to the brute force method of calculation, also known as the discrete Fourier transform at nonequispaced nodes (NDFT). Let the vectors $x = (a, b)$



(a) Actual times

(b) Predicted times from neural net with subnets

Fig. 1: Access time over the first 4 tracks of a hard drive, in milliseconds

and $\xi = (u, v)$, where u and v are coordinates in frequency space. Further define N as the number of data points (in our case, the number of requests in the trace), and $\hat{f}$ as the Fourier transform of f. The discrete Fourier transform is defined:

$$\hat{f}(\xi) = \frac{1}{N} \sum_k f(x_k) e^{-2\pi i x_k \cdot \xi}$$

which takes $O(N)$ time to calculate for a single frequency vector ξ, so calculating $\hat{f}$ for M frequencies takes $O(MN)$ time. Scanning all frequencies in the 2D space leads to infeasible computation time for large datasets or datasets over larger regions of the drive.

Note that the access time depends mostly on the difference between the sectors. This is intuitively true, as the time to move from sector 0 to sector 9 should be roughly the same as the time to move from sector 1 to sector 10. This causes stripes in the access time function along $a = b$ (see fig. 1a), and the Fourier transform of a function with stripes has strong components in the direction orthogonal to the stripes [60], which means that strong components of the access time function should lie on $u = -v$. We see this empirically in fig. 2. By searching only this diagonal instead of the entire space, the computation time becomes feasible.

Note that this is mathematically identical to finding the Fourier transform of the 1D function $d(c)$ where $c = b - a$.

$$\begin{aligned}\hat{f}(u, v) &= \frac{1}{N} \sum_k f(x_k) e^{-2\pi i (au + bv)} \\ &= \frac{1}{N} \sum_k f(x_k) e^{-2\pi i (b-a)v} \text{ (where } u = -v) \\ &= \hat{d}(v)\end{aligned}$$

Off-diagonal frequencies occur, but they are mostly combinations of pairs of frequencies on the diagonal. An example would be a dataset that includes track sizes of 10 and 11. Strong coefficients are likely at periods (10, 10), (10, 11), (11, 10), and (11, 11). Since the diagonal provides (10, 10) and (11, 11) (or just 10 and 11 in 1D), the locations of the others could be inferred, although that is not actually necessary.

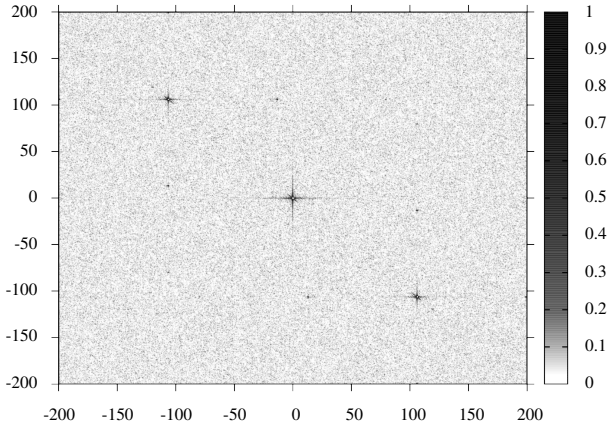


Fig. 2: Full 2D Fourier spectrum for the first $K = 237{,}631$ sectors out to $\frac{200}{K} \times \frac{200}{K} = 8.42 \cdot 10^{-4} \times 8.42 \cdot 10^{-4}$, which corresponds to periods of at least $1/(8.42 \cdot 10^{-4}) = 1188.155$ sectors. Plot is clipped to magnitude 1 to show detail, but central spike goes up to 8.6, and other diagonal spikes go up to 3.9. This figure shows that strong frequencies do lie on the diagonal $v = -u$.

To determine a threshold for strong frequencies, we sample 1000 frequencies at random and calculate $|\hat{f}|$. The threshold is then set to the mean plus six standard deviations. Since the square magnitudes are exponentially distributed [61], this means the magnitudes are Rayleigh distributed, so this threshold means that a frequency has roughly a 1 in 700,000 chance of being spuriously flagged. A few false positives are not a problem, since they will be filtered out during the tuning stage (see section VI).

We scan across $v = -u$ with a step size of $0.1/blockCount$, looking for local maxima above the threshold. The peaks were often not centered on integer multiples of $1/blockCount$, which is why the step size is not

$1/blockCount$. After finding a maximum, we perform a local search to fine-tune its position.

Due to structures such as serpentines, f may have higher-order periods. In other words, f may have period p_1 for a subrange, then p_2, then p_1, then p_2, etc., with the switch between p_1 and p_2 being periodic. Currently, we have no method for detecting these explicitly, although we have seen them show up as interesting periods in their own right. These actually cause further problems by reducing the utility of the lower-level periods; the multiple regions using period p may be out of phase, causing $|\hat{f}|$ to drop. Essentially, this is a form of mixed interference which results in a weaker signal than if the signals interfered purely constructively.

Alternative algorithms exist for finding periodic components. The Nonequispaced Fast Fourier Transform (NFFT) reduces computational complexity from $O(MN)$ to $O(M \log M + N)$ [62], but for the full two-dimensional case, M is still roughly 10^{18}. A better approach is to use an algorithm that assumes the Fourier transform is sparse. Pawar and Ramchandran describe an algorithm that takes $O(k \log k)$ time to find a k-sparse one-dimensional Fourier transform in the noiseless case, but they do not describe asymptotic complexity in the noisy case [63]. Ghazi *et al.* describe an algorithm that takes $O(k \log^2 N)$ time to find a k-sparse two-dimensional Fourier transform, but they require $k = \Theta(\sqrt{M})$ for the noisy case [64]. We leave use of an advanced Fourier algorithm to future work.

B. Input augmentation

Once all interesting periods have been found, the input vectors for the machine learning algorithm are augmented. Given an input (a, b) and periods $p_1, \ldots, p_k$, the augmented input vector is $\big(a, \cos(2\pi a/p_1), \sin(2\pi a/p_1), \ldots, \cos(2\pi a/p_k), \sin(2\pi a/p_k), b, \cos(2\pi b/p_1), \sin(2\pi b/p_1), \ldots, \cos(2\pi b/p_k), \sin(2\pi b/p_k)\big)$. We use sinusoids of a and b separately rather than sinusoids of $b - a$ because the period changes for each input separately. For example, if a is in a region where p_1 dominates, and b is in a region where p_2 dominates, then $\sin(2\pi a/p_1)$ and $\sin(2\pi b/p_2)$ are useful, but $\sin(2\pi(b-a)/p_3)$ is not likely to be useful for any period p_3.

If no interesting periods are found, then the input vectors are unchanged. This means that for devices with no periodicity, the cost of the periodicity assumption is just the time to search for periods. The neural net or decision tree is unchanged, and therefore the speed and accuracy of the model is unchanged.

V. Neural network structure

The neural net maps pairs of locations in logical space to access time. This mapping contains a lot of structure, and we find it useful to decompose the net into two components: a neural network that maps from logical space to physical space, and a neural network that maps from pairs of physical locations to access time. In other words, instead of directly computing a mapping $f : L^2 \to T$, we decompose it into $g : L \to P$ and $h : P^2 \to T$ so that $f(a, b) = h(g(a), g(b))$. (Technically, f is not restricted as long as the intermediate space is at least as large as the input space, since one could always use $g(x) = x$ and $h = f$, but this predisposes the neural net to learn more useful decompositions.) Since the values of P are hidden, *i.e.*, we do not know the physical locations of sectors, we cannot train g and h separately. Instead, we have to chain them together as f and train them simultaneously. Since g is used twice in f, this means f will have two subnets (corresponding to g) that are identical, since they are simply two occurrences of the same object. Concretely, this means the structure and weights in these subnets will be identical. This is achieved by applying weight sharing across two subnets (see fig. 3). A similar approach was used by Kindermann *et al.* for solving functional equations with neural nets [65].

Another interpretation is that the subnets perform feature generation, knowing that the two logical locations are instances in the same input space. In this interpretation, g extracts useful features from our raw data, and h maps from those higher-level features to our output.

For comparison, we also evaluated a traditional neural net without subnets (see fig. 4). This network is fully connected between layers and does not use weight sharing. This equates to the case of directly computing the mapping $f : L^2 \to T$.

All neurons use a sigmoidal activation function except the final output, which is linear. This is common practice for neural nets performing regression [38].

We know of no simple way to decompose the mapping computed by decision trees, so they were unmodified in this respect.

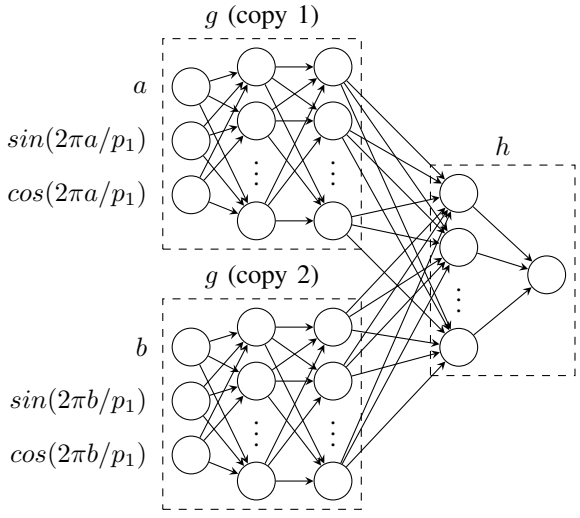


Fig. 3: Network architecture when subnets are used. Note that the weights in the two g subnets are identical.

VI. Hyperparameter tuning

Hyperparameters are often tuned using some combination of manual search and grid search [48], [49], [50], [51], although genetic algorithms are also used [58], [66], [52], [53]. Grid search is known to be inefficient if not all hyperparameters have equal importance [51], and manual search is not reproducible. We use a genetic algorithm to tune the machine learning algorithms' hyperparameters.

A set of hyperparameters is evaluated by running the machine learning algorithm with those parameters and evaluating

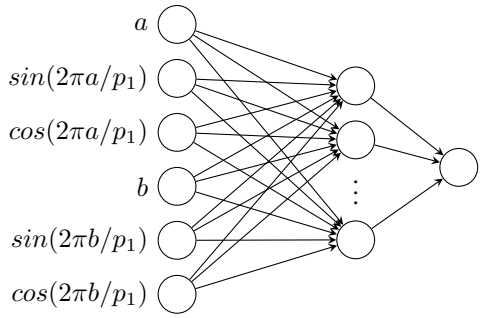


Fig. 4: Network architecture when subnets are not used

its performance. Since training a neural network or building an ensemble of decision trees is relatively expensive, we use a relatively small population size to reduce the amount of computation needed.

We chose to minimize the L_1 norm (equivalent to minimizing Mean Absolute Error, or MAE) instead of the L_2 norm (equivalent to minimizing Root Mean Square Error, or RMSE) mainly because the L_2 norm is known to be sensitive to outliers [67], [68]. For an empirical comparison of the two, see section VIII-E.

A. Neural network tuning

Neural network hyperparameters include layer sizes, learning rate, momentum, and magnitudes of the initial weights.

The representation used in our genetic algorithm includes booleans (for periods to be included or not), positive integers (for layer sizes), and positive reals (for the learning rate, momentum, and initial weight magnitudes). Using a representation that matches our problem should provide better performance than using a simple bit string [69]. When mutating, booleans are flipped, and integers are incremented or decremented, and reals are multiplied by a log-normally distributed value.

Initial hidden layer sizes were sampled from a log-normal distribution with $\mu = \ln 10$ and $\sigma = \ln 10$ (then cast to an integer). The top 25 periods with the largest Fourier magnitudes were used as candidate periods. Each candidate period was included with a probability of 10%. The initial learning rates were sampled from a log-normal distribution with $\mu = \ln(4 \times 10^{-3})$ and $\sigma = \ln 100$, the momenta were sampled uniformly from 0 to 1, and the initial weight standard deviations were sampled from a log-normal distribution with $\mu = 0$ and $\sigma = \ln 10$, all mutated as described below.

Individuals were evaluated by training a neural net with stochastic backpropagation and minimizing the L_1 error. Initial weights were sampled from a normal distribution with mean 0 and standard deviation that is a hyperparameter that depended on the layer. A random 10% of the dataset was set aside as a test set, and the neural net was trained on the remainder for 10 epochs. A penalty of 1.8×10^{-5} per connection was added to the error to discourage unnecessary bloat. This value for the penalty was chosen to be roughly 10% of the raw error.

After each generation, the best 25% was kept for mating, and the rest were discarded. Random pairs were selected for mating, and attributes were randomly swapped to generate pairs of children. Each child was mutated such that the expected number of attributes changed was 5/8. On mutation, a period's inclusion would be toggled, a layer size would be incremented or decremented, and real values would be multiplied by a value sampled from a log-normal distribution with $\mu = 0$, $\sigma^2 = 10^{-2}$ (so that an "average" change would be $\pm 10\%$).

B. Decision tree tuning

For decision trees, we tune the maximum depth, minimum number of instances per leaf, minimum variance proportion, whether or not to use pruning, and number of folds when pruning.

Bagging is a popular ensemble method used to improve the accuracy of decision trees. Bagging generates many decision trees trained on random subsets of the data, then averages their predictions [70]. We test both with and without bagging. When bagging is enabled, the ensemble size is also tuned.

As with neural nets, we use a genetic algorithm to tune the hyperparameters, and we use the same representation. A set of hyperparameters is evaluated by building a decision tree (or ensemble of decision trees) with those parameters and evaluating its performance. Illegal combinations, such as pruning enabled and less than two pruning folds, results in infinite error.

Decision trees were tested with WEKA [71]. When using bagging in the manual case, the number of decision trees was set to 100. When using bagging in the autotuning case, a penalty of 7×10^{-4} times the ensemble size is added to prevent the ensemble size from growing unnecessarily large. This results in a roughly 10% increase on top of the raw error.

VII. Experimental setup

The device we modeled is a Western Digital Caviar Black WD5002AALX. It has a capacity of 500GB (976,773,168 sectors), rotational speed of 7200 RPM, cache size of 32 MB, and NCQ queue length of 32. The drive was connected with SATA. The host runs 64-bit Ubuntu Linux, kernel version 2.6.35-28-server. It has an Intel Core i7 quad-core CPU (plus hyper-threading) running at 2.80 GHz and 12 GB of RAM.

Block-level traces were captured using blktrace. Request latency was defined to be the D2C or device-to-completion time. This is the time between the OS IO scheduler sending the request to the device driver and receiving a response from the device driver. Excluding device driver times is difficult and requires a hardware setup. In modern computers, the time spent in the device driver should be negligible compared to the time spent in the hard drive.

The dataset is of $N = 32{,}000$ random reads of the first 94 tracks, or 237,631 sectors, which is the first 0.024% of the drive. This corresponds to the first part of the first serpentine, so all tracks have the same size (2528 sectors). We chose this limitation to simplify the problem and plan to remove it in future work.

A population of 104 individuals was used, which provided good utilization of the 52 compute cores available for

the experiment, and the genetic algorithm was run for 400 generations. We trained and tested each configuration five times and reported the mean and standard deviation. For tests with random periods, each run used different random periods selected uniformly from the range 0 to 5000 sectors.

VIII. Results

A. Overview

The Fourier analysis for our dataset takes approximately 7 minutes on the 52-core cluster, the genetic algorithm takes approximately 160 minutes to run on the 52-core cluster, the final training of a bag of decision trees takes approximately 2 minutes on a single core, and the final training of a neural net takes approximately 15 minutes on a single core. Access time errors for various configurations are listed in table I. The first entry is the error for a model that always predicts the mean value of the training set, which is used as a baseline.

The lowest L_1 error achieved with decision trees was 0.526 ms. The lowest L_1 error achieved with neural nets was 0.157 ms.

B. Periodicity

For reference, we ran DIG on our test hard disk drive. From the DIG data, the track length at the beginning of the drive is 2528 sectors, and the skew is 1.1908 ms, which corresponds to 361.37 sectors (given the rotation time of 8.33 ms). We expected to see a dominant period of $2528-361.37 = 2166.63$ sectors. From the Fourier analysis, the actual dominant period is 2212.99 sectors, which is close to our prediction.

When using decision trees, adding the sines and cosines as additional features reduced the error noticeably. Bagging the decision trees reduced error further, but only when the periodic information was included.

When using neural nets, adding the sines and cosines reduced the error significantly. Weight sharing reduced the error further, cutting it by half in the autotuning case. For autotuned neural nets with weight sharing, including periodicity information reduces the L_1 error by 90%.

In roughly half of the autotuning trials, tests with random periods fare no better than tests with no period information. In the other trials, at least one random period is close to an important period, and the error is reduced. While random periods are occasionally useful for the current setup, the probability of a period being useful plummets to virtually zero when this approach is scaled up to an entire device.

C. Period count penalty

A penalty of 4×10^{-3} per included period is added to prevent inclusion of unnecessary periods and to improve convergence. When no penalty term is added, the included periods do not converge well (fig. 6a). Although some periods are strongly selected for (top of the plot), and others are strongly selected against (bottom of the plot), many flip back and forth within a run and are inconsistent across runs. By including the penalty term, the included periods converge quickly (clear middle area, fig. 6b).

The value of the penalty was chosen by starting with 10^{-3}, which would give a penalty term roughly equal to 10% of the L_1 norm. We then doubled the penalty until we saw that the periods converged well.

D. Neural net generation length

Multiple generation lengths were tested to see if the optimal parameters depend on the generation length. Generation length is the number of epochs spent training each neural network.

Different experiments converge to the same values for most hyperparameters regardless of generation length (fig. 7). This is useful because it means that the hyperparameters can be chosen using a short generation length, and then the final neural network can be trained for a longer period of time. This is the approach we take, with neural nets being trained for 10 epochs during hyperparameter tuning, then trained for 1000 epochs for the final evaluation.

Unsurprisingly, a higher learning rate is chosen when the generation length is very short. This makes intuitive sense because if the generation stops while a neural network's error is still dropping rapidly, a small increase in the learning rate will provide a large benefit. Even though the higher learning rate increases noise in the training error, the early cutoff means that the training is stopped before this becomes a problem.

Since the optimal learning rate and momentum are dependent on the generation length, we discard those learned values and used fixed values of 4×10^{-4} for the learning rate and 0.3 for the momentum when training a network for evaluation.

E. Comparing L_1 and L_2 norms

Minimizing the L_1 norm gives a tighter correspondence between predicted and actual values compared with the L_2 norm (fig. 8). We also found that minimizing the L_1 error reduces both the L_1 error and, surprisingly, the L_2 error (table II). In other words, minimizing the L_1 error during training

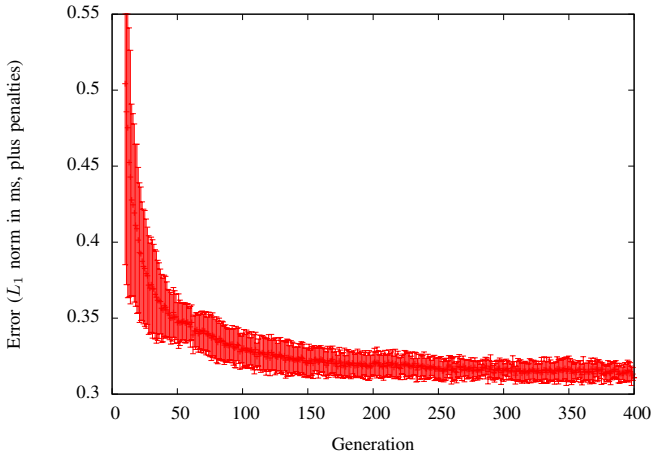


Fig. 5: Error versus generation of the genetic algorithm for neural nets with subnets. Note that this is higher than the final error because 1) this error includes penalties, and 2) this error is calculated based on neural nets trained for a smaller number of epochs.

resulted in lower L_2 error during testing than minimizing the L_2 error during training. We believe this is probably due to the sensitivity of the L_2 norm to non-gaussian errors causing the model to generalize poorly.

	L_1 test error	L_2 test error
Minimizing L_1 training error	0.139 ± 0.003	0.730 ± 0.019
Minimizing L_2 training error	0.273 ± 0.005	0.799 ± 0.019

TABLE II: Effect of minimizing L_1 versus L_2 as measured by L_1 and L_2, with 95% confidence interval over 50 runs

Figure 8 shows large errors at extreme values. The cause is the fact that the access time function has many large discontinuities (see fig. 1a) where it "wraps" from the maximum to minimum value. Near a discontinuity, a small error may push the predicted value to the wrong side of the discontinuity compared to the actual value, yielding a large error. Hence, extreme values have a higher chance of large error.

F. Other

The error of the genetic algorithm drops quickly, with most of the gain seen in the first 50 generations (fig. 5). We ran the genetic algorithm for 400 generations to ensure maximum benefit, but this is not necessary if one wants faster results.

For neural nets with subnets, the H2 layer, which corresponds to the output of the g subnets, evolves to a relatively small size (about 6 or 7 neurons) with little variance (fig. 7a). This suggests that the dimensionality of the space of physical sector locations is low, and that allowing for a larger intermediate space may actually be detrimental.

When comparing the actual and predicted access time functions (fig. 1), we see that the shape is matched very well. Even the notches in the stripes are in the predicted locations.

IX. Conclusion

Access time prediction of individual requests is a difficult problem, largely because of the rotational nature of hard disk drives. We have shown how it can be done using Fourier analysis and machine learning.

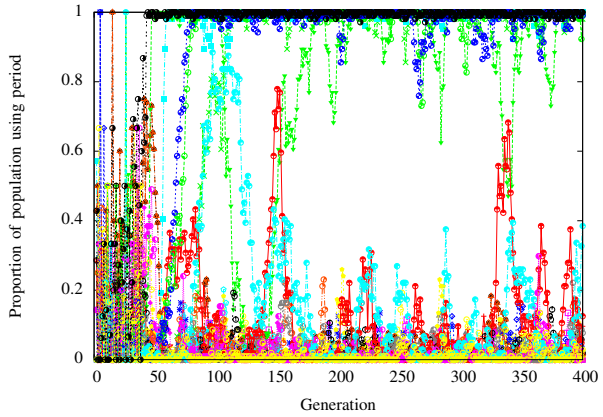
Neural nets achieved lower error than decision trees in all tests. We suspect this is caused by two things: 1) the structure imposed by weight sharing cannot be implemented with decision trees, and 2) interactions between inputs, which decision trees do not learn from well [72].

Splitting the neural net into subnets and using weight sharing across the subnets significantly reduces error. Weight sharing requires no assumptions about the nature of the device, only the knowledge that the input consists of two objects from the same set, namely, block requests. Partitioning the neural net into subnets that use weight sharing can be seen as combining feature construction and learning into one step.

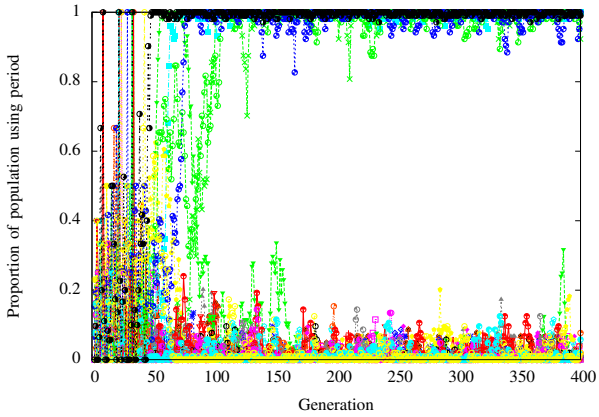
Adding periodic features requires only a mild assumption and improves multiple machine learning algorithms significantly. This approach is likely to be a crucial part of future behavioral modeling of storage device performance.

	Configuration	L_1 error (ms)	L_2 error (ms)
	constant value	1.651 ± 0.017	2.014 ± 0.019
Manually tuned decision trees	no periods, without bagging	1.650 ± 0.001	2.014 ± 0.008
	no periods, with bagging	1.655 ± 0.000	2.023 ± 0.004
	random periods, without bagging	1.608 ± 0.062	1.969 ± 0.056
	random periods, with bagging	1.545 ± 0.152	1.925 ± 0.180
	Fourier periods, without bagging	1.082 ± 0.085	1.544 ± 0.183
	Fourier periods, with bagging	0.748 ± 0.009	1.349 ± 0.338
Auto-tuned decision trees	no periods, without bagging	1.640 ± 0.009	2.006 ± 0.016
	no periods, with bagging	1.649 ± 0.001	2.016 ± 0.005
	random periods, without bagging	1.519 ± 0.169	2.022 ± 0.015
	random periods, with bagging	1.058 ± 0.336	1.507 ± 0.295
	Fourier periods, without bagging	0.865 ± 0.046	1.717 ± 0.129
	Fourier periods, with bagging	0.526 ± 0.019	1.032 ± 0.018
Manually tuned neural nets	no periods, without subnets	1.603 ± 0.016	2.058 ± 0.017
	no periods, with subnets	1.593 ± 0.023	2.043 ± 0.026
	random periods, without subnets	1.616 ± 0.024	2.072 ± 0.030
	random periods, with subnets	1.401 ± 0.426	1.894 ± 0.365
	Fourier periods, without subnets	0.308 ± 0.009	0.969 ± 0.034
	Fourier periods, with subnets	0.236 ± 0.010	0.808 ± 0.036
Auto-tuned neural nets	no periods, without subnets	1.613 ± 0.020	2.079 ± 0.026
	no periods, with subnets	1.608 ± 0.027	2.059 ± 0.030
	random periods, without subnets	1.270 ± 0.455	1.791 ± 0.390
	random periods, with subnets	1.394 ± 0.417	1.862 ± 0.363
	Fourier periods, without subnets	0.298 ± 0.012	0.961 ± 0.060
	Fourier periods, with subnets	0.157 ± 0.015	0.785 ± 0.065
	"unrestrained" neural net*	0.149 ± 0.021	0.781 ± 0.072

TABLE I: Average errors for access time predictions. Neural nets were trained for 1000 epochs. Solid bars show the mean error, and thin bars show the standard deviation of the error.
*The "unrestrained" neural net was evolved with all penalties set to 0, so it has lower error at the expense of larger size.

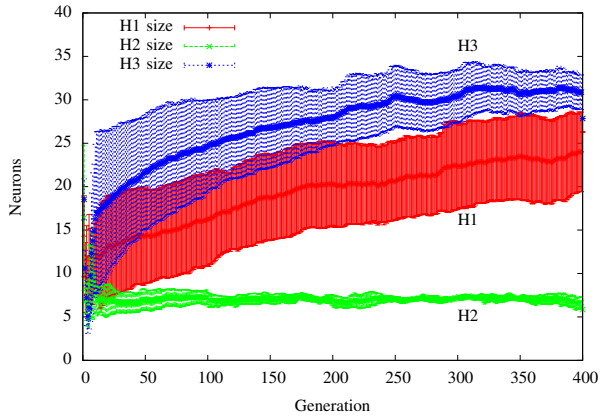


(a) no penalty per period

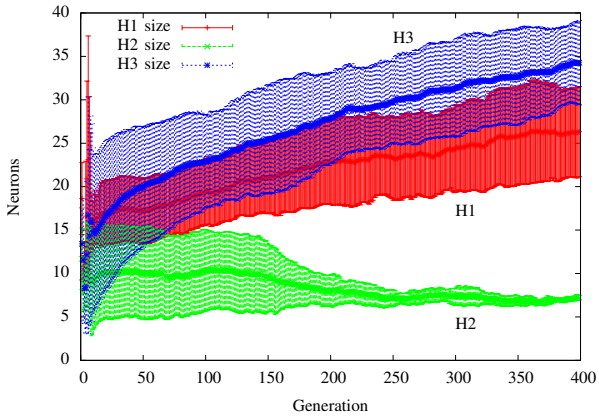


(b) $4 \cdot 10^{-3}$ penalty per period

Fig. 6: Usage of periods versus generation (neural nets with subnets). Note that with no penalty, periods converge poorly, *i.e.*, cross $y = 0.5$ even in later generations.

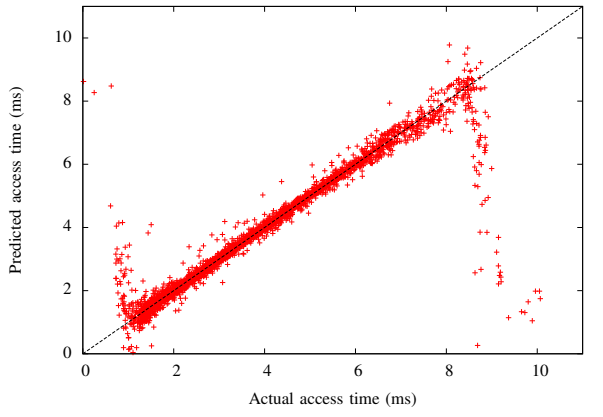


(a) 10 epochs per generation

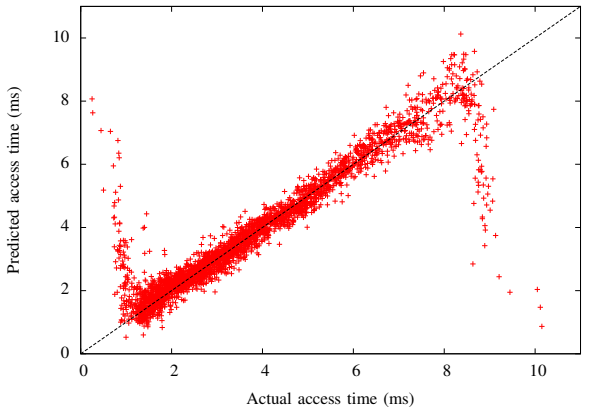


(b) 40 epochs per generation

Fig. 7: Size of hidden layers versus generation. The same sizes are chosen even when the generation length is varied.



(a) Predicted versus actual access times when trained with L_1 norm



(b) Predicted versus actual access times when trained with L_2 norm

Fig. 8: Comparison of L_1 versus L_2 norm for neural nets with subnets. (A narrow cluster along the diagonal $y = x$ is better.) High errors are seen at extreme values due to discontinuities in the access time function. For details, see section VIII-E.

REFERENCES

[1] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *MSST/SNAPI 2012*, Pacific Grove, CA, April 16 - 20 2012.

[2] Y. Liu, R. Figueiredo, D. Clavijo, Y. Xu, and M. Zhao, "Towards simulation of parallel file system scheduling algorithms with PFSsim," in *Proceedings of the 7th IEEE International Workshop on Storage Network Architectures and Parallel I/O (May 2011)*, 2011.

[3] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang, "Quickly finding near-optimal storage designs," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 4, pp. 337–374, 2005.

[4] H. Huang, W. Hung, and K. G. Shin, "FS2: dynamic data replication in free disk space for improving disk performance and energy consumption," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 263–276, 2005.

[5] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors, *The DiskSim Simulation Environment Version 4.0 Reference Manual*, Carnegie Mellon University, Pittsburgh, PA, May 2008. [Online]. Available: http://www.pdl.cs.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf

[6] Y. Chen, W. W. Hsu, and H. C. Young, "Logging RAID - an approach to fast, reliable, and low-cost disk arrays," in *Euro-Par 2000 Parallel Processing*, ser. Lecture Notes in Computer Science, A. Bode, T. Ludwig, W. Karl, and R. Wismüller, Eds. Springer Berlin Heidelberg, 2000, vol. 1900, pp. 1302–1311. [Online]. Available: http://dx.doi.org/10.1007/3-540-44520-X_182

[7] L. Zhang, G. Liu, X. Zhang, S. Jiang, and E. Chen, "Storage device performance prediction with selective bagging classification and regression tree," *Network and Parallel Computing*, pp. 121–133, 2010.

[8] A. Núñez, J. Fernández, R. Filgueira, F. García, and J. Carretero, "SIMCAN: A flexible, scalable and expandable simulation platform for modelling and simulating distributed architectures and applications," *Simulation Modelling Practice and Theory*, vol. 20, no. 1, pp. 12–32, 2012.

[9] D. Lingenfelter, A. Khurshudov, and D. Vlassarev, "Efficient disk drive performance model for realistic workloads," *Magnetics, IEEE Transactions on*, vol. 50, no. 5, pp. 1–9, May 2014.

[10] C. Dai, G. Liu, L. Zhang, and E. Chen, "Storage device performance prediction with hybrid regression models," in *PDCAT'12*, Beijing, China, December 2012.

[11] A. Thomasian, "Survey and analysis of disk scheduling methods," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 8–25, 2011.

[12] M. Li and J. Shu, "DACO: A high-performance disk architecture designed specially for large-scale erasure-coded storage systems," *Computers, IEEE Transactions on*, vol. 59, no. 10, pp. 1350–1362, 2010.

[13] T. Xie and Y. Sun, "Dynamic data reallocation in hybrid disk arrays," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 9, pp. 1330–1341, 2010.

[14] L. Liu, Z. H. Liu, L. Xu, and J. Zhang, "FBctlr - a novel approach for storage performance virtualization," in *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. IEEE, 2011, pp. 268–272.

[15] E. Varki, A. Hubbe, and A. Merchant, "Improve prefetch performance by splitting the cache replacement queue," in *Advanced Infocomm Technology*. Springer, 2013, pp. 98–108.

[16] L. Liu, L. Xu, Z. H. Liu, and J. Zhang, "QClock: An interposed scheduling algorithm for performance virtualization in shared storage systems," in *Networked Computing (INC), 2011 The 7th International Conference on*. IEEE, 2011, pp. 17–21.

[17] L. Newman and J. Nowitzke, *Cheetah 9LP Family - Product Manual*, C ed., Seagate, August 1998.

[18] D. Ashby, B. Norman, and K. Tan, *Barracuda 4LP Family - Product Manual*, D ed., Seagate, Feb 1998.

[19] "Quantum atlas 10k ii," http://www.seagate.com/staticfiles/maxtor/en_us/documentation/data_sheets/atlas_10k_ii_datasheet.pdf, 2000, aIIDS0600.

[20] J. Dìaz, "This is the largest SATA drive in the world," http://gizmodo.com/5667594/this-is-the-largest-sata-drive-in-the-world, October 2010.

[21] "Seagate breaks capacity ceiling with world's first 3 terabyte external desktop drive," http://media.seagate.com/2010/06/seagatetechnology/seagate-breaks-capacity-ceiling-with-worlds-first-3-terabyte-external-desktop-drive/, June 2010.

[22] "WD caviar green series disti spec sheet," http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-701229.pdf, January 2012, 2879-701229-A25.

[23] R. Oldfield, Personal communication, January 2013.

[24] J. Gim and Y. Won, "Extract and infer quickly: Obtaining sector geometry of modern hard disk drives," *Trans. Storage*, vol. 6, pp. 6:1–6:26, July 2010. [Online]. Available: http://doi.acm.org.oca.ucsc.edu/10.1145/1807060.1807063

[25] Y. Shiroishi, K. Fukuda, I. Tagawa, H. Iwasaki, S. Takenoiri, H. Tanaka, H. Mutoh, and N. Yoshikawa, "Future options for HDD storage," *Magnetics, IEEE Transactions on*, vol. 45, no. 10, pp. 3816–3822, 2009.

[26] T. Kelly, I. Cohen, M. Goldszmidt, and K. Keeton, "Inducing models of black-box storage arrays," HP Laboratories, Palo Alto, CA, Technical Report HPL-2004-108, June 2004.

[27] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *MASCOTS 2004*, 2004.

[28] D. F. Specht and P. Shapiro, "Generalization accuracy of probabilistic neural networks compared with backpropagation networks," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. i, 1991, pp. 887–892 vol.1.

[29] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-90-100, February 1990.

[30] Y. Shang and B. W. Wah, "Global optimization for neural network training," *Computer*, vol. 29, no. 3, pp. 45–54, 1996.

[31] A. Crume, C. Maltzahn, L. Ward, T. Kroeger, M. Curry, R. Oldfield, and P. Widener, "Fourier-assisted machine learning of hard disk drive access time models," in *Proceedings of the 8th Parallel Data Storage Workshop*. ACM, 2013, pp. 45–51.

[32] A. S. Lebrecht, N. J. Dingle, and W. J. Knottenbelt, "A performance model of zoned disk drives with I/O request reordering," in *Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems*, ser. QEST '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 97–106. [Online]. Available: http://dx.doi.org/10.1109/QEST.2009.31

[33] J. Garcia, L. Prada, J. Fernandez, A. Nunez, and J. Carretero, "Using black-box modeling techniques for modern disk drives service time simulation," in *Simulation Symposium, 2008. ANSS 2008. 41st Annual*, april 2008, pp. 139 –145.

[34] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger, "Modeling the relative fitness of storage," in *SIGMETRICS 2007*, 2007.

[35] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989–993, 1994.

[36] A. Guez and Z. Ahmad, "Solution to the inverse kinematics problem in robotics by neural networks," in *Neural Networks, 1988., IEEE International Conference on*. IEEE, 1988, pp. 617–624.

[37] B. E. Rosen, "Ensemble learning using decorrelated neural networks," *Connection Science*, vol. 8, no. 3-4, pp. 373–384, 1996.

[38] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *Neural Networks, IEEE Transactions on*, vol. 16, no. 1, pp. 24–38, 2005.

[39] Z. Zainuddin and O. Pauline, "Function approximation using artificial neural networks," *WSEAS Transactions on Mathematics*, vol. 7, no. 6, pp. 333–338, 2008.

[40] K. Kosanovich, A. Gurumoorthy, E. Sinzinger, and M. Piovoso, "Improving the extrapolation capability of neural networks," in *Intelligent Control, 1996., Proceedings of the 1996 IEEE International Symposium on*. IEEE, 1996, pp. 390–395.

[41] J. Kwon, B. Coifman, and P. Bickel, "Day-to-day travel-time trends and travel-time prediction from loop-detector data," *Transportation*

Research Record: Journal of the Transportation Research Board, vol. 1717, no. 1, pp. 120–129, 2000.

[42] D. Elizondo, G. Hoogenboom, and R. McClendon, "Development of a neural network model to predict daily solar radiation," *Agricultural and Forest Meteorology*, vol. 71, no. 1, pp. 115–132, 1994.

[43] M. Gardner and S. Dorling, "Neural network modelling and prediction of hourly NO_x and NO_2 concentrations in urban air in london," *Atmospheric Environment*, vol. 33, no. 5, pp. 709 – 719, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1352231098002301

[44] K.-W. Wong, C.-S. Leung, and S.-J. Chang, "Use of periodic and monotonic activation functions in multilayer feedforward neural networks trained by extended Kalman filter algorithm," in *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 149, no. 4. IET, 2002, pp. 217–224.

[45] J. M. Sopena, E. Romero, and R. Alquezar, "Neural networks with periodic and monotonic activation functions: a comparative study in classification problems," in *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, vol. 1. IET, 1999, pp. 323–328.

[46] G. Noone and S. D. Howard, "Investigation of periodic time series using neural networks and adaptive error thresholds," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4. IEEE, 1995, pp. 1541–1545.

[47] S. Jagannathan and F. L. Lewis, "Multilayer discrete-time neural-net controller with guaranteed performance," *Neural Networks, IEEE Transactions on*, vol. 7, no. 1, pp. 107–130, 1996.

[48] G. Hinton, "A practical guide to training restricted Boltzmann machines," University of Toronto, Tech. Rep., August 2010, uTML TR 2010-003.

[49] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*. Springer-Verlag, 1998, pp. 9–50.

[50] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 473–480.

[51] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[52] Y. Jin, T. Okabe, and B. Sendhoff, "Neural network regularization and ensembling using multi-objective evolutionary algorithms," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 1. IEEE, 2004, pp. 1–8.

[53] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[54] B. Behzad, L. H. V. Thanh, J. Huchette, S. Byna, R. A. Prabhat, Q. Koziol, and M. Snir, "Taming parallel I/O complexity with auto-tuning," in *Proceedings of 2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2013)*, 2013.

[55] C. F. M. Toledo, J. M. G. Lima, and M. da Silva Arantes, "A multi-population genetic algorithm approach for PID controller auto-tuning," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–8.

[56] J. Nicklow, P. Reed, D. Savic, T. Dessalegne, L. Harrell, A. Chan-Hilton, M. Karamouz, B. Minsker, A. Ostfeld, A. Singh *et al.*, "State of the art for genetic algorithms and beyond in water resources planning and management," *Journal of Water Resources Planning and Management*, vol. 136, no. 4, pp. 412–432, 2009.

[57] C. Harpham, C. W. Dawson, and M. R. Brown, "A review of genetic algorithms applied to training radial basis function networks," *Neural Computing & Applications*, vol. 13, no. 3, pp. 193–201, 2004.

[58] F. H.-F. Leung, H.-K. Lam, S.-H. Ling, and P. K.-S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *Neural Networks, IEEE Transactions on*, vol. 14, no. 1, pp. 79–88, 2003.

[59] S. K. Chalup and L. Wiklendt, "Variations of the two-spiral task," *Connection Science*, vol. 19, no. 2, pp. 183–199, 2007.

[60] J. M. Gauch, "Ch4 - Fourier transform," http://www.csce.uark.edu/~jgauch/5683/notes/ch04a.pdf. [Online]. Available: http://www.csce.uark.edu/~jgauch/5683/notes/ch04a.pdf

[61] D. R. Brillinger, *Time Series: Data Analysis and Theory*. SIAM, 1981, vol. 36.

[62] S. Kunis and D. Potts, "Time and memory requirements of the nonequispaced FFT," *Sampling Theory in Signal & Image Processing*, vol. 7, no. 1, 2008.

[63] S. Pawar and K. Ramchandran, "Computing a k-sparse n-length discrete Fourier transform using at most 4k samples and O(k log k) complexity," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 464–468.

[64] B. Ghazi, H. Hassanieh, P. Indyk, D. Katabi, E. Price, and L. Shi, "Sample-optimal average-case sparse Fourier transform in two dimensions," *arXiv preprint arXiv:1303.1209*, 2013.

[65] L. Kindermann, A. Lewandowski, and P. Protzel, "A framework for solving functional equations with neural networks," in *Proceedings of Neural Information Processing (ICONIP2001)*, vol. 2. Fudan University Press, Shanghai, 2001, pp. 1075–1078.

[66] H. A. Abbass, "Speeding up backpropagation using multiobjective evolutionary algorithms," *Neural Computation*, vol. 15, no. 11, pp. 2705–2726, 2003.

[67] J. S. Armstrong and F. Collopy, "Error measures for generalizing about forecasting methods: Empirical comparisons," *International Journal of Forecasting*, vol. 8, no. 1, pp. 69–80, 1992.

[68] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, no. 1, p. 79, 2005.

[69] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[70] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[71] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009, http://www.cs.waikato.ac.nz/~ml/weka/. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[72] S. Esmeir and S. Markovitch, "Anytime learning of decision trees," *The Journal of Machine Learning Research*, vol. 8, pp. 891–933, 2007.