Fourier-Assisted Machine Learning of Hard Disk Drive Access Time Models

Adam Crume
University of California
Santa Cruz, CA
adamcrume@cs.ucsc.edu

Carlos Maltzahn
University of California
Santa Cruz, CA
carlosm@cs.ucsc.edu

Lee Ward
Sandia National Laboratories
Livermore, CA
lee@sandia.gov

Thomas Kroeger
Sandia National Laboratories
Livermore, CA
tmkroeg@sandia.gov

Matthew Curry
Sandia National Laboratories
Livermore, CA
mlcurry@sandia.gov

Ron Oldfield
Sandia National Laboratories
Livermore, CA
raoldfi@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. Others have created behavioral models of hard disk drive performance, but 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 show how hard disk drive access times can be predicted to within 0.83 ms using a neural net after these frequencies are found using Fourier analysis.

Categories and Subject Descriptors

I.6.5 [**Model Development**]: Modeling methodologies; D.4.8 [**Performance**]: Modeling and prediction; D.4.2 [**Storage Management**]:

General Terms

Performance

Keywords

black-box, storage, modeling, hard disk, machine learning, neural network, Fourier transform

1. INTRODUCTION

Hard disk drive performance models are often used as part of a much larger system simulation [38], for parallel file system simulation [39], for storage configuration [2], and for data placement [26]. The most accurate models are white-box models such as DiskSim [8], which is used by many people [9, 55, 39]. These models require extensive parameterization.

Manufacturers do not release details such as sector layout required by white-box models, which makes parameterizing the models a long and difficult process. In fact, one of us (Oldfield) spent several months configuring DiskSim to model an existing device. Tools such as DIG can extract some of this information [20]. 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 or other optimizations, as has happened in the past with Zoned Bit Recording, serpentine layouts, etc. Since manufacturers do not release this information, researchers must reverse-engineer a device before modifying DIG and DiskSim to support the new layout.

A more desirable approach is to use machine learning 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 [29, 12, 51, 55], but none can accurately model individual requests.

Access time has stubbornly resisted efforts to model it, which is what we focus on in this paper. We use workloads that read random single sectors, which minimizes caching and readahead effects. In this case, request latency consists of two components:

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, then be serviced out of order.

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.

After decomposing the 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 tested decision trees and neural nets for access time prediction. One of the complications in the problem 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 [49] as well as the two-spirals problem [14, 46]. We overcome this limitation by finding the frequencies of these periodic patterns with Fourier analysis and then feeding them into the neural net explicitly by augmenting the feature vector.

2. RELATED WORK

2.1 Predicting request latencies

DiskSim [8] 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 [34]. 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 experts.

Kelly et al. describe a black-box probabilistic model [29] similar to table-based models such as the one by Garcia et al. [17]. 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 the issue by essentially not requiring the entire table to be filled in, but the problem is not fully solved.

Mesnier et al. create a model for relative performance of storage devices [40]. Unfortunately, their approach still requires an accurate base model.

A popular approach is to use regression trees to predict response time. Dai et al. predict performance with a combination of regression trees and support vector regression [12]. 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 windows in their case [51].

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

2.2 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 [22, 21, 45, 15, 54]. Extrapolating beyond the training range leads to poor performance [31]. 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 [32, 13]. An alternative is to map it to $\sin(2\pi x/p)$ and $\cos(2\pi x/p)$ [18]. (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 known precisely ahead of time, and that the input be exactly periodic.

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 [53]. Periodic activation functions also introduce many local minima [48].

A powerful tool for time series data is recurrent or delay networks [43, 27]. 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.

3. FOURIER ANALYSIS

We assume that the access time function may have components that are locally periodic. This is a fairly lax assumption. First of all, it is well-known that 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 3.2 the cost is minor if these functions do not have locally periodic components.

3.1 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. To find periodic behavior in f, the Fourier transform is an obvious place to start. It can only find periods that are directly correlated to the value of f, but 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 swamped 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 takes a very long time. Given our test device's mean access time of 15.5 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 a very sparse sampling.

The sparsity means that we cannot use the Fast Fourier Transform, so we fall back to the brute force method of calculation. Let the vectors $x = (a, b)$ 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), M as the number of frequencies to search, 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. 4a), and the Fourier transform of a function with stripes has strong components in the direction orthogonal to the stripes [19], which means that strong components of the access time function should lie on $u = -v$. We see this empirically in fig. 1. By searching only this diagonal instead of the entire space, the computation time becomes feasible.

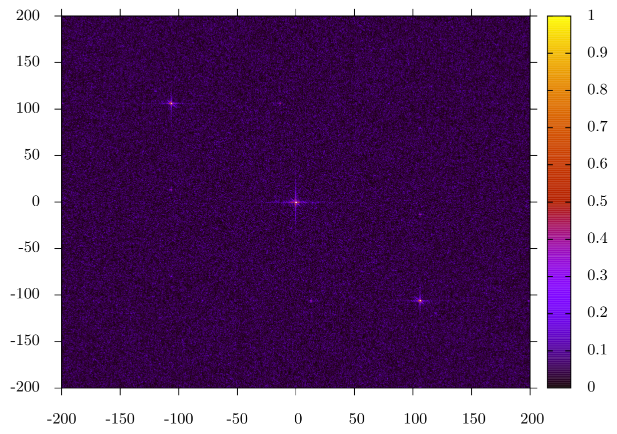


Figure 1: Full 2D Fourier spectrum for the first $K = 237{,}631$ sectors out to $\frac{200}{K} \times \frac{200}{K} = 8.42 \cdot 10^{-4} \times 8.42 \cdot 10^{-4}$, which corresponds to periods of at least $1/(8.42 \cdot 10^{-4}) = 1188.155$ sectors. Plot is clipped to magnitude 1 to show detail, but 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 $v = -u$.

Another way to approach the Fourier analysis, which we did not test, is to use a search-based method. Assuming the number of strong frequencies is small, this should be much faster, roughly $O(kN \log M)$ for k strong frequencies. Unfortunately, these methods require performing convolutions, and it is not clear if performing a convolution on sparse data is possible or meaningful.

Off-diagonal frequencies occur, but they are mostly related in a straightforward fashion to 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.

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. Assuming the magnitudes are normally distributed, this means that a frequency has roughly a 1 in 500 million chance of being spuriously flagged.

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.

3.2 Input augmentation

Once all interesting periods have been found, the input vectors for the neural net 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 is unchanged, and therefore the speed and accuracy of the model is unchanged.

4. SHARED WEIGHTS

A useful and device-independent assumption is that all sectors map from LBNs to geometrical space in the same way. This is encoded into the neural net by applying weight sharing across two subnets (see fig. 2). This means that the weights in each neuron in subnet 1 are identical to the weights in the corresponding neuron in subnet 2. This is essentially a very simple convolutional neural net with only two instances of the convolving subnet and no overlap of inputs. A similar approach was used by Kindermann et al. for solving functional equations with neural nets [30]. Another interpretation is that the subnets perform feature generation, knowing that the two sectors are instances in the same input space.

More formally, the net assumes f can be expressed as

$$f(a, b) = h(g(a), g(b))$$

where g (the subnet) maps from LBNs to geometry, and h (the main net) maps from geometry to access time. Reuse of g, which is equivalent to weight sharing in the neural network, formalizes the assumption that all sectors map to geometry in the same way. 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.

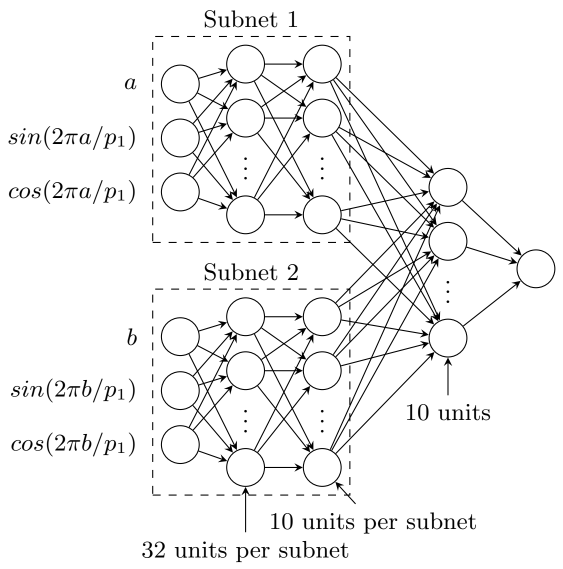


Figure 2: Network architecture when subnets are used. Note that the weights in subnet 1 are identical to the weights in subnet 2.

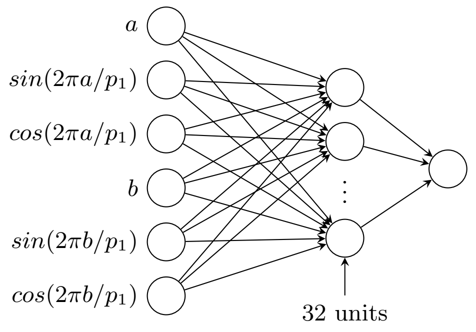


Figure 3: Network architecture when subnets are not used

Each subnet has 32 hidden and 10 output units, and the main net has 10 hidden units. All neurons use a sigmoidal activation function except the final output, which is linear. This is common practice for neural nets performing regression [15]. Note that when 6 periods are used, the number of parameters is nearly equal with or without subnets, so the configurations should have the same expressive power.

We know of no analogous operation for decision trees, so they were unmodified in this respect.

5. 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).

Decision trees were tested with Weka. In all cases, the minimum leaf weight (minNum) was set to 0. Bagging, a popular ensemble method used to improve the accuracy of decision trees, was also tested. Bagging generates many decision trees trained on random subsets of the data, then averages their predictions [6]. When using bagging, the number of decision trees was set to 100.

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 between 0 and 5000 sectors.

6. RESULTS

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, which is a frequency of $1/2166.63 = 4.62 \cdot 10^{-4}$. From the Fourier analysis, the actual dominant period is $1/(4.52 \cdot 10^{-4}) = 2212.99$ sectors, which is close to our prediction of 2166.63.

Access time errors are listed in table 1. The first entry is the error for a model that always predicts the mean value of the dataset.

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. The lowest error achieved with decision trees was 1.1906 ms.

When using neural nets, adding the sines and cosines reduced the error significantly. Weight sharing reduced the error further by a small but noticeable amount. The lowest error achieved with neural nets was 0.830 ms.

Most of the time, neural net tests with random periods fared worse than tests with no period information. However, one test with random periods performed much better, although not as well as tests with the correct period information. This appears to have been caused by one of the random periods being nearly equal to twice the most important period.

When comparing the actual and predicted access time functions (fig. 4), we see that the shape is matched very well. Even the notches in the stripes are in the predicted locations.

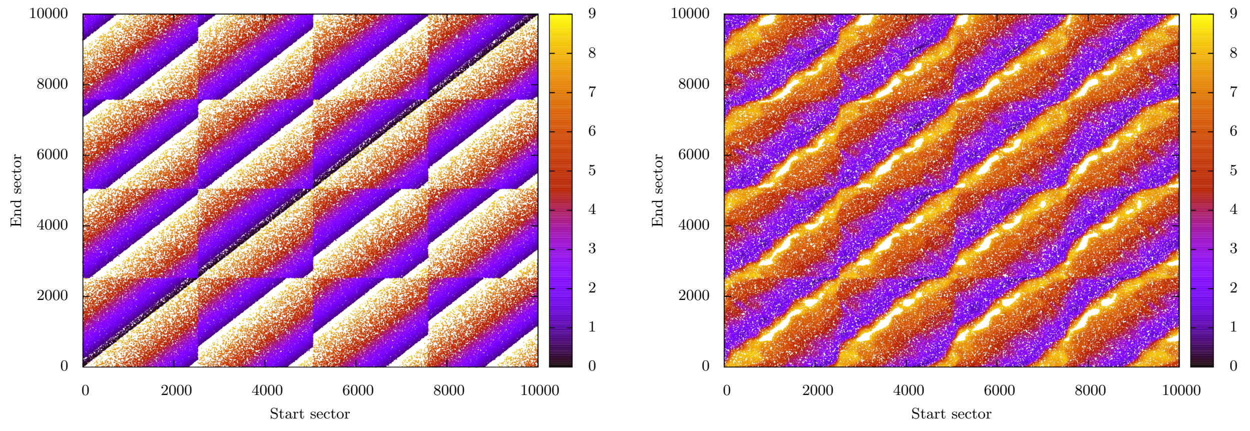
7. CONCLUSIONS

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) many important features are individually uncorrelated with the output, which decision trees do not learn from well.

Adding periodic features requires only a mild assumption

	Configuration	Error (ms)
	constant value	2.013 ± 0.000
Decision trees	no periods, without bagging	2.075 ± 0.014
	no periods, with bagging	2.067 ± 0.001
	6 random periods, without bagging	2.019 ± 0.013
	6 random periods, with bagging	2.015 ± 0.013
	6 periods, without bagging	1.649 ± 0.154
	6 periods, with bagging	1.123 ± 0.009
Neural nets	no periods, without subnets	2.014 ± 0.034
	no periods, with subnets	2.012 ± 0.019
	6 random periods, without subnets	1.924 ± 0.176
	6 random periods, with subnets	1.992 ± 0.059
	6 periods, without subnets	0.954 ± 0.052
	6 periods, with subnets	0.830 ± 0.031

Table 1: Average RMS errors for access time predictions



(a) Actual times (b) Predicted times

Figure 4: Access time over the first 4 tracks of a hard drive, in milliseconds

and improves multiple machine learning algorithms significantly. This approach is likely to be a crucial part of future behavioral modeling of storage device performance.

8. ONGOING WORK

So far, we have restricted the workload to a portion of the hard disk drive with a single track length. To make the approach useful, we need to extend it across the entire hard disk drive. Unfortunately, applying the method directly does not work, in part because the Fourier transform is heavily distorted (fig. 5).

To solve this, we split the space into distinct regions where the periods change. Each region has its Fourier transform computed separately and gets its own subnet (fig. 6). When training or evaulating the neural net, the appropriate subnets are swapped in based on which regions the requests lie in.

To find the boundaries between regions, we search recursively. When evaluating a region, that region is split into two halves, and the Fourier transform is computed for each half separately. If significant periodicity is found, and if

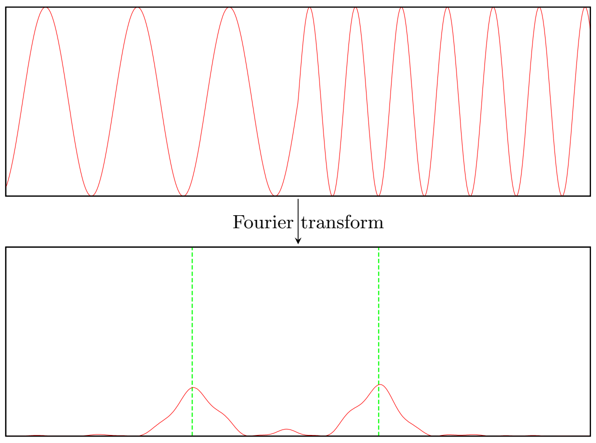


Figure 5: The Fourier transform of mixed data is messy

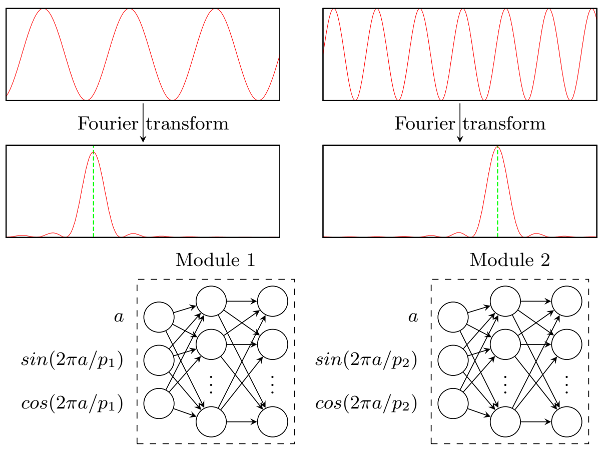


Figure 6: Splitting regions results in clean Fourier transforms. Each region gets its own subnet.

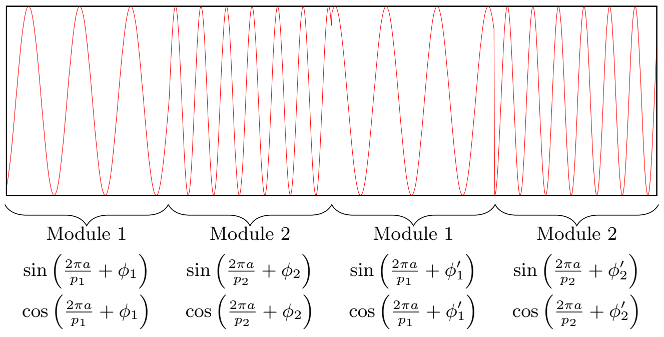

Module 1
Module 2
Module 1
Module 2
$\sin\left(\frac{2\pi a}{p_1}+\phi_1\right)$
$\cos\left(\frac{2\pi a}{p_1}+\phi_1\right)$
$\sin\left(\frac{2\pi a}{p_2}+\phi_2\right)$
$\cos\left(\frac{2\pi a}{p_2}+\phi_2\right)$
$\sin\left(\frac{2\pi a}{p_1}+\phi_1'\right)$
$\cos\left(\frac{2\pi a}{p_1}+\phi_1'\right)$
$\sin\left(\frac{2\pi a}{p_2}+\phi_2'\right)$
$\cos\left(\frac{2\pi a}{p_2}+\phi_2'\right)$

Figure 7: Similar regions reuse subnets, with a phase offset

the spectra of the subregions are sufficiently similar, then it is assumed that the entire region is one piece and has a uniform spectrum. Otherwise, the search procedure is repeated on the subregions recursively. Regions that straddle a search boundary may be split unnecessarily. To fix this, a postprocessing step combines adjacent regions with similar spectra.

Because of serpentines, there may be many regions, even though the total number of unique track lengths is low, meaning that many regions have identical spectra but are discontiguous. Treating each region independently is impractical because of the very large number of parameters involved. To remedy this, we reuse subnets across regions with identical spectra (fig. 7). The regions may not be in phase with each other, so we add a phase offset (ϕ) to allow for this.

9. ACKNOWLEDGEMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

10. ADDITIONAL AUTHORS

Additional authors: Patrick Widener (Sandia National Laboratories, email: `pwidene@sandia.gov`).

11. REFERENCES

[1] H. A. Abbass. Speeding up backpropagation using multiobjective evolutionary algorithms. *Neural Computation*, 15(11):2705–2726, 2003.

[2] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla, and Q. Wang. Quickly finding near-optimal storage designs. *ACM Transactions on Computer Systems (TOCS)*, 23(4):337–374, 2005.

[3] J. S. Armstrong and F. Collopy. Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, 8(1):69–80, 1992.

[4] 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.

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

[6] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

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

[8] 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.

[9] Y. Chen, W. W. Hsu, and H. C. Young. Logging RAID - an approach to fast, reliable, and low-cost disk arrays. In A. Bode, T. Ludwig, W. Karl, and R. Wismüller, editors, *Euro-Par 2000 Parallel*

Processing, volume 1900 of *Lecture Notes in Computer Science*, pages 1302–1311. Springer Berlin Heidelberg, 2000.

[10] A. Crume, C. Maltzahn, L. Ward, T. Kroeger, and M. Curry. Automatic generation of behavioral hard disk drive access time models.

[11] 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*, pages 45–51. ACM, 2013.

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

[13] D. Elizondo, G. Hoogenboom, and R. McClendon. Development of a neural network model to predict daily solar radiation. *Agricultural and Forest Meteorology*, 71(1):115–132, 1994.

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

[15] S. Ferrari and R. F. Stengel. Smooth function approximation using neural networks. *Neural Networks, IEEE Transactions on*, 16(1):24–38, 2005.

[16] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[17] 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*, pages 139 –145, april 2008.

[18] 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*, 33(5):709 – 719, 1999.

[19] J. M. Gauch. Ch4 - Fourier transform. `http://www.csce.uark.edu/~jgauch/5683/notes/ch04a.pdf`.

[20] J. Gim and Y. Won. Extract and infer quickly: Obtaining sector geometry of modern hard disk drives. *Trans. Storage*, 6:6:1–6:26, July 2010.

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

[22] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the Marquardt algorithm. *Neural Networks, IEEE Transactions on*, 5(6):989–993, 1994.

[23] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009. `http://www.cs.waikato.ac.nz/~ml/weka/`.

[24] C. Harpham, C. W. Dawson, and M. R. Brown. A review of genetic algorithms applied to training radial basis function networks. *Neural Computing & Applications*, 13(3):193–201, 2004.

[25] G. Hinton. A practical guide to training restricted Boltzmann machines. Technical report, University of Toronto, August 2010. UTML TR 2010-003.

[26] 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*, 39(5):263–276, 2005.

[27] S. Jagannathan and F. L. Lewis. Multilayer discrete-time neural-net controller with guaranteed performance. *Neural Networks, IEEE Transactions on*, 7(1):107–130, 1996.

[28] Y. Jin, T. Okabe, and B. Sendhoff. Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 1–8. IEEE, 2004.

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

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

[31] 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*, pages 390–395. IEEE, 1996.

[32] 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*, 1717(1):120–129, 2000.

[33] 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*, pages 473–480. ACM, 2007.

[34] 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*, QEST '09, pages 97–106, Washington, DC, USA, 2009. IEEE Computer Society.

[35] 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*, pages 9–50. Springer-Verlag, 1998.

[36] 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*, 14(1):79–88, 2003.

[37] M. Li and J. Shu. DACO: A high-performance disk architecture designed specially for large-scale erasure-coded storage systems. *Computers, IEEE Transactions on*, 59(10):1350–1362, 2010.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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*, 136(4):412–432, 2009.

[43] 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*, volume 4, pages 1541–1545. IEEE, 1995.

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

[45] B. E. Rosen. Ensemble learning using decorrelated neural networks. *Connection Science*, 8(3-4):373–384, 1996.

[46] Y. Shang and B. W. Wah. Global optimization for neural network training. *Computer*, 29(3):45–54, 1996.

[47] 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*, 45(10):3816–3822, 2009.

[48] 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)*, volume 1, pages 323–328. IET, 1999.

[49] 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*, volume i, pages 887–892 vol.1, 1991.

[50] 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*, pages 1–8. IEEE, 2012.

[51] 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.

[52] 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*, 30(1):79, 2005.

[53] 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-*, volume 149, pages 217–224. IET, 2002.

[54] Z. Zainuddin and O. Pauline. Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, 7(6):333–338, 2008.

[55] 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*, pages 121–133, 2010.