# PREDICTIVE MODELING OF I/O CHARACTERISTICS IN HIGH PERFORMANCE COMPUTING SYSTEMS

Thomas C. H. Lux

Dept. of Computer Science
Virginia Polytechnic Institute
& State University
Blacksburg, VA 24061
tchlux@vt.edu

Layne T. Watson

Dept. of Computer Science
Dept. of Mathematics
Dept. of Aerospace & Ocean Eng.
Virginia Polytechnic Institute
& State University

Tyler H. Chang
Jon Bernard
Bo Li

Dept. of Computer Science
Virginia Polytechnic Institute
& State University

Li Xu

Dept. of Statistics
Virginia Polytechnic Institute
& State University

Godmar Back
Ali R. Butt
Kirk W. Cameron

Dept. of Computer Science
Virginia Polytechnic Institute
& State University

Yili Hong

Dept. of Statistics
Virginia Polytechnic Institute
& State University

## ABSTRACT

Each of high performance computing, cloud computing, and computer security have their own interests in modeling and predicting the performance of computers with respect to how they are configured. An effective model might infer internal mechanics, minimize power consumption, or maximize computational throughput of a given system. This paper analyzes a four-dimensional dataset measuring the input/output (I/O) characteristics of a cluster of identical computers using the benchmark IOzone. The I/O performance characteristics are modeled with respect to system configuration using multivariate interpolation and approximation techniques. The analysis reveals that accurate models of I/O characteristics for a computer system may be created from a small fraction of possible configurations, and that some modeling techniques will continue to perform well as the number of system parameters being modeled increases. These results have strong implications for future predictive analyses based on more comprehensive sets of system parameters.

**Keywords:** Regression, approximation, interpolation, performance modeling

## 1 INTRODUCTION AND RELATED WORK

Performance tuning is often an experimentally complex and time intensive chore necessary for configuring HPC systems. The procedures for this tuning vary largely from system to system and are often subjectively guided by the system engineer(s). Once a desired level of performance is achieved, an HPC system may

only be incrementally reconfigured as required by updates or specific jobs. In the case that a system has changing workloads or nonstationary performance objectives that range from maximizing computational throughput to minimizing power consumption and system variability, it becomes clear that a more effective and automated tool is needed for configuring systems. This scenario presents a challenging and important application of multivariate approximation and interpolation techniques.

Predicting the performance of an HPC system is a challenging problem that is primarily attempted in one of two ways: (1) build a statistical model of the performance by running experiments on the system at select settings, or (2) run artificial experiments using a simplified simulation of the target system to estimate architecture and application bottlenecks. In this paper the proposed multivariate modeling techniques rest in the first category, and they represent a notable increase in the ability to model complex interactions between system parameters.

Many previous works attempting to model system performance have used simulated environments to estimate the performance of a system (Grobelny et al. 2007, Wang et al. 2009, Wang et al. 2013). Some of these works refer to statistical models as being oversimplified and not capable of capturing the true complexity of the underlying system. This claim is partially correct, noting that a large portion of predictive statistical models rely on simplifying the model to one or two parameters (Snavely et al. 2002, Bailey and Snavely 2005, Barker et al. 2009, Ye et al. 2010). These limited statistical models have provided satisfactory performance in very narrow application settings. Many of the aforementioned statistical modeling techniques claim to generalize, while simultaneously requiring additional code annotations, hardware abstractions, or additional application level understandings in order to generate models. The approach presented here requires no modifications of the application, no architectural abstractions, nor any structural descriptions of the input data being modeled. The techniques used are purely mathematical and only need performance data as input.

Among the statistical models presented in prior works, Bailey and Snavely (2005) specifically mention that it is difficult for the simplified models to capture variability introduced by I/O. System variability in general has become a problem of increasing interest to the HPC and systems communities, however most of the work has focused on operating system (OS) induced variability (Beckman et al. 2008, De et al. 2007). The work that has focused on managing I/O variability does not use any sophisticated modeling techniques (Lofstead et al. 2010). Hence, this paper presents a case study applying advanced mathematical and statistical modeling techniques to the domain of HPC I/O characteristics. The models are used to predict the mean throughput of a system and the variance in throughput of a system. The discussion section outlines how the techniques presented can be applied to any performance metric and any system.

In general, this paper compares five multivariate approximation techniques that operate on inputs in $\mathbb{R}^d$ ($d$-tuples of real numbers) and produce predicted responses in $\mathbb{R}$. In order to provide coverage of the varied mathematical strategies that can be employed to solve the continuous modeling problem, three of the techniques are regression based and the remaining two are interpolants. The sections below outline the mathematical formulation of each technique and provide computational complexity bounds with respect to the size (number of points and dimension) of input data. Throughout, $d$ will refer to the dimension of the input data, $n$ is the number of points in the input data, $x^{(i)} \in \mathbb{R}^d$ is the $i$-th input data point, $x_j^{(i)}$ is the $j$-th component of $x^{(i)}$, and $f(x^{(i)})$ is the response value of the $i$-th input data point.

The remainder of the paper is broken up into five major parts. Section 2 provides an overview of the multivariate modeling techniques, Section 3 outlines the methodology for comparing and evaluating the performance of the models, Section 4 presents the IOzone predictions, Section 5 discusses the obvious and subtle implications of the models' performance, and Section 6 concludes and offers directions for future work.

## 2 MULTIVARIATE MODELS

### 2.1 Regression

Multivariate approximations are capable of accurately modeling a complex dependence of a response (in $\mathbb{R}$) on multiple variables (represented as a points in $\mathbb{R}^d$). The approximations to some (unknown) underlying function $f : \mathbb{R}^d \to \mathbb{R}$ are chosen to minimize some error measure related to data samples $f(x^{(i)})$. For example, least squares regression uses the sum of squared differences between modeled response values and true response values as an error measure.

#### 2.1.1 Multivariate Adaptive Regression Splines

This approximation was introduced in Friedman (1991) and subsequently improved to its current version in Friedman and the Computational Statistics Laboratory of Stanford University (1993), called fast multivariate adaptive regression splines (Fast MARS). In Fast MARS, a least squares fit model is iteratively built by beginning with a single constant valued function and adding two new basis functions at each iteration of the form

$$B_{2s-1}(x) = B_l(x)[c(x_i - v)]_+,$$
$$B_{2s}(x) = B_k(x)[c(x_i - v)]_-,$$

where $s$ is the iteration number, $B_l(x)$ and $B_k(x)$ are basis functions from the previous iteration, $c, v \in \mathbb{R}$,

$$w_+ = \begin{cases} w, & w \geq 0 \\ 0, & w < 0 \end{cases},$$

and $w_- = (-w)_+$. After iteratively constructing a model, MARS then iteratively removes basis functions that do not contribute to goodness of fit. In effect, MARS creates a locally component-wise tensor product approximation of the data. The overall computational complexity of Fast MARS is $\mathcal{O}(ndm^3)$ where $m$ is the maximum number of underlying basis functions. This paper uses an implementation of Fast MARS (Rudy and Cherti 2017) with $m = 200$.

#### 2.1.2 Multilayer Perceptron Regressor

The neural network is a well studied and widely used method for both regression and classification tasks (Hornik et al. 1989). When using the rectified linear unit (ReLU) activation function (Dahl et al. 2013) and training with the BFGS minimization technique (Møller 1993), the model built by a multilayer perceptron uses layers $l : \mathbb{R}^i \to \mathbb{R}^j$ defined by

$$l(u) = \left(u^t W_l\right)_+,$$

where $W_l$ is the $i$ by $j$ weight matrix for layer $l$. In this form, the multilayer perceptron (MLP) produces a piecewise linear model of the input data. The computational complexity of training a multilayer perceptron is $\mathcal{O}(ndm)$, where $m$ is determined by the sizes of the layers of the network and the stopping criterion of the BFGS minimization used for finding weights. This paper uses the scikit-learn MLP regressor (Pedregosa et al. 2011), a single hidden layer with 100 nodes, ReLU activation, and BFGS for training.

### 2.1.3 Support Vector Regressor

Support vector machines are a common method used in machine learning classification tasks that can be adapted for the purpose of regression (Basak et al. 2007). How to build a support vector regressor (SVR) is beyond the scope of this summary, but the resulting functional fit $p : \mathbb{R}^d \to \mathbb{R}$ has the form

$$p(x) = \sum_{i=1}^{n} a_i K(x, x^{(i)}) + b,$$

where $K$ is the selected kernel function, $a \in \mathbb{R}^n$, $b \in \mathbb{R}$ are coefficients to be solved for simultaneously. The computational complexity of the SVR is $\mathcal{O}(n^2 dm)$, with $m$ being determined by the minimization convergence criterion. This paper uses the scikit-learn SVR (Pedregosa et al. 2011) with a polynomial kernel function.

## 2.2 Interpolation

In some cases it is desirable to have a model that can recreate the input data exactly. This is especially the case when the confidence in the response values for known data is high. Both interpolation models analyzed in this paper are based on linear functions.

### 2.2.1 Delaunay

The Delaunay method of interpolation is a well studied geometric technique for producing an interpolant (Lee and Schachter 1980). The Delaunay triangulation of a set of data points into simplices is such that the sphere defined by the vertices of each simplex contains no data points in the sphere's interior. For a $d$-simplex S with vertices $v^{(0)}$, $v^{(1)}$, …, $v^{(d)}$, $x \in S$, and data values $f(v^{(i)})$, $i = 0, \ldots, d$, $x$ is a unique convex combination of the vertices,

$$x = \sum_{i=0}^{d} w_i v^{(i)}, \quad \sum_{i=0}^{d} w_i = 1, \quad w_i \geq 0, \quad i = 0, \ldots, d,$$

and the Delaunay interpolant to $f$ at $x$ is given by

$$p(x) = \sum_{i=0}^{d} w_i f(v^{(i)}).$$

The computational complexity of the Delaunay triangulation (for the implementation used here) is $\mathcal{O}(n^{\lceil d/2 \rceil})$, which is not scalable to $d > 10$ (Sartipizadeh and Vincent 2016). The scipy interface (Jones et al. 2017) to the QuickHull implementation (Barber et al. 1996) of the Delaunay triangulation is used here.

### 2.2.2 Linear Shepard

The linear Shepard method (LSHEP) is a blending function using local linear interpolants, a special case of the general Shepard algorithm (Thacker et al. 2010). The interpolant has the form

$$p(x) = \frac{\sum_{k=1}^{n} W_k(x) P_k(x)}{\sum_{k=1}^{n} W_k(x)},$$

| **System Parameter** | **Values** |
|---|---|
| File Size | 64, 256, 1024 |
| Record Size | 32, 128, 512 |
| Thread Count | 1, 2, 4, 8, 16, 32, 64, 128, 256 |
| Frequency | $\{12, 14, 15, 16, 18, 19, 20, 21, 23, 24, 25, 27, 28, 29, 30, 30.01\} \times 10^5$ |
| **Response Values** | |
| Throughput Mean | $[2.6 \times 10^5, 5.9 \times 10^8]$ |
| Throughput Variance | $[5.9 \times 10^{10}, 4.7 \times 10^{16}]$ |

Table 1: A description of the system parameters being considered in the IOzone tests. Record size must not be greater than file size and hence there are only six valid combinations of the two. In total there are $6 \times 9 \times 16 = 864$ unique system configurations.

where $W_k(x)$ is a locally supported weighting function and $P_k(x)$ is a local linear approximation to the data satisfying $P_k\left(x^{(k)}\right) = f\left(x^{(x)}\right)$. The computational complexity of LSHEP is $\mathcal{O}(n^2d^3)$. This paper uses the Fortran#95 implementation of LSHEP in SHEPPACK (Thacker et al. 2010).

## 3 METHODOLOGY

### 3.1 Data

In order to evaluate the viability of multivariate models for predicting system performance, this paper presents a case study of a four-dimensional dataset produced by executing the IOzone benchmark from Norcott (2017) on a homogeneous cluster of computers. All experiments were performed on parallel shared-memory nodes common to HPC systems. Each system had a lone guest Linux operating system (Ubuntu 14.04 LTS//XEN 4.0) on a dedicated 2TB HDD on a 2 socket, 4 core (2 hyperthreads per core) Intel Xeon E5-2623 v3 (Haswell) platform with 32 GB DDR4. The system performance data was collected by executing IOzone 40 times for each of a select set of system configurations. A single IOzone execution reports the max I/O throughput seen for the selected test in kilobytes per second. The 40 executions for each system configuration are converted into the mean and variance, both values in $\mathbb{R}$ capable of being modeled individually by the multivariate approximation techniques presented in Section 2. The summary of data used in the experiments for this paper can be seen in Table 1. Distributions of raw throughput values being modeled can be seen in Figure 1.

### 3.2 Dimensional Analysis

This work utilizes an extension to standard $k$-fold cross validation that allows for a more thorough investigation of the expected model performance in a variety of real-world situations. Alongside randomized splits, two extra components are considered: the amount of training data provided, and the dimension of the input data. It is important to consider that algorithms that perform well with less training input also require less experimentation. Although, the amount of training data required may change as a function of the dimension of the input and this needs to be studied as well. The framework used here will be referred to as a multidimensional analysis (MDA) of the IOzone data.

#### 3.2.1 Multidimensional Analysis

This procedure combines random selection of training and testing splits with changes in the input dimension and the ratio of training size to testing size. Given an input data matrix with $n$ rows (points) and $d$ columns (components), MDA proceeds as follows:

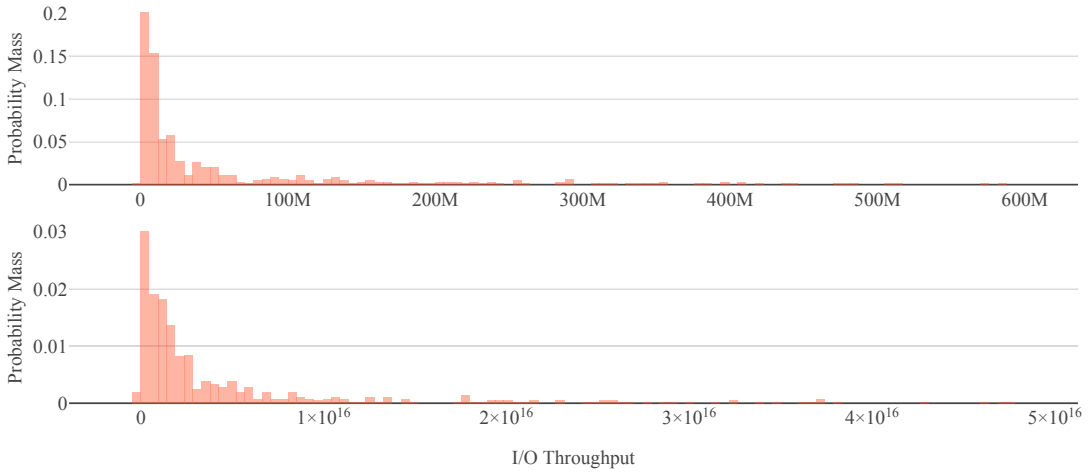


Figure 1: Histograms of 100-bin reductions of the PMF of I/O throughput mean (top) and I/O throughput variance (bottom). In the mean plot, the first 1% bin (truncated in plot) has a probability mass of .45. In the variance plot, the second 1% bin has a probability mass of .58. It can be seen that the distributions of throughputs are primarily of lower magnitude with occasional extreme outliers.

1. For all $k = 1, \ldots, d$ and for all nonempty subsets $F \subset \{1,2,\ldots,d\}$, reduce the input data to points $(z, f_F(z))$ with $z \in \mathbb{R}^k$ and $f_F(z) = E\left[\left\{f\left(x^{(i)}\right) \mid \left(x_F^{(i)} = z\right)\right\}\right]$, where $E[\cdot]$ denotes the mean and $x_F^{(i)}$ is the subvector of $x^{(i)}$ indexed by $F$.
2. For all $r$ in $\{5, 10, \ldots, 95\}$, generate $N$ random splits $(train, test)$ of the reduced data with $r$ percentage for training and $100 - r$ percentage for testing.
3. When generating each of $N$ random $(train, test)$ splits, ensure that all points from $test$ are in the convex hull of points in $train$ (to prevent extrapolation); also ensure that the points in $train$ are well spaced.

In order to ensure that the testing points are in the convex hull of the training points, the convex hull vertices of each set of (reduced dimension) points are forcibly placed in the training set. In order to ensure that training points are well spaced, a statistical method for picking points from Amos et al. (2014) is used:

1. Generate a sequence of all pairs of points $\left(z^{(i_1)}, z^{(j_1)}\right), \left(z^{(i_2)}, z^{(j_2)}\right), \ldots$ sorted by ascending pairwise Euclidean distance between points, so that $\left|\left|z^{(i_k)} - z^{(j_k)}\right|\right|_2 \leq \left|\left|z^{(i_{k+1})} - z^{(j_{k+1})}\right|\right|_2$.
2. Sequentially remove points from candidacy until only $|train|$ remain by randomly selecting one point from the pair $\left(z^{(i_m)}, z^{(j_m)}\right)$ for $m = 1, \ldots$ if both $z^{(i_m)}$ and $z^{(j_m)}$ are still candidates for removal.

Given the large number of constraints, level of reduction, and use of randomness in the MDA procedure, occasionally $N$ unique training/testing splits may not be created or may not exist. In these cases, if there are fewer than $N$ possible splits, then deterministically generated splits are used. Otherwise after $3N$ attempts, only the unique splits are kept for analysis. The MDA procedure has been implemented in Python#3 while most regression and interpolation methods are Fortran wrapped with Python. All randomness has been seeded for repeatability.

For any index subset $F$ (of size $k$) and selected value of $r$, MDA will generate up to $N$ multivariate models $f_F(z)$ and predictions $\hat{f}_F\left(z^{(i)}\right)$ for a point $z^{(i)} \in \mathbb{R}^k$. There may be fewer than $N$ predictions made for any given point. Extreme points of the convex hull for the selected index subset will always be used for training, never for testing. Points that do not have any close neighbors will often be used for training in order to ensure well-spacing. Finally, as mentioned before, some index subsets do not readily generate $N$ unique training and testing splits. The summary results presented in this work use the median of the ($N$ or fewer) values $\hat{f}_F(z)$ at each point as the model estimate for error analysis.

## 4 RESULTS

A naïve multivariate prediction technique such as nearest neighbor could experience relative errors in the range $[0, \left(\max_x f(x) - \min_x f(x)\right) / \min_x f(x)]$ when modeling a nonnegative function $f(x)$ from data. The IOzone mean data response values span three orders of magnitude (as can be seen in Table 1) while variance data response values span six orders of magnitude. It is expected therefore, that all studied multivariate models perform better than a naïve approach, achieving relative errors strictly less than $10^3$ for throughput mean and $10^6$ for throughput variance. Ideally, models will yield relative errors significantly smaller than 1. The time required to compute thousands of models involved in processing the IOzone data through MDA was approximately five hours on a CentOS workstation with an Intel i7-3770 CPU at 3.40GHz. In four dimensions for example, each of the models could be constructed and evaluated over hundreds of points in less than a few seconds.

### 4.1 I/O Throughput Mean

Almost all multivariate models analyzed make predictions with a relative error less than 1 for most system configurations when predicting the mean I/O throughput of a system given varying amounts of training data. The overall best of the multivariate models, Delaunay, consistently makes predictions with relative error less than .05 (5% error). In Figure 3 it can also be seen that the Delaunay model consistently makes good predictions even with as low as 5% training data (43 of the 864 system configurations) regardless of the dimension of the data.

### 4.2 I/O Throughput Variance

The prediction results for variance resemble those for predicting mean. Delaunay remains the best overall predictor (aggregated across training percentages and dimensions) with median relative error of .47 and LSHEP closely competes with Delaunay having a median signed relative error of -.92. Outliers in prediction error are much larger for all models. Delaunay produces relative errors as large as 78 and other models achieve relative errors around $10^3$. The relative errors for many models scaled proportional to the increased orders of magnitude spanned by the variance response compared with mean response. As can be seen in Figure 4, all models are more sensitive to the amount of training data provided than their counterparts for predicting mean.

### 4.3 Increasing Dimension and Decreasing Training Data

As can be seen in Figure 2, all of the models suffer increasing error rates in higher dimension. This is expected, because the number of possible interactions to model grows exponentially. However, LSHEP and Delaunay maintain the slowest increase in relative error. The increase in error seen for Delaunay suggests that it is capable of making predictions with a range of relative errors that grows approximately linearly with increasing dimension input. This trend suggests that Delaunay would remain a viable technique for accurately modeling systems with 10's of parameters given only small amounts of training data. All models, with the exception of MARS, produce smaller errors given more training data. Increasing the amount of training data most notably reduces the number of prediction error outliers.

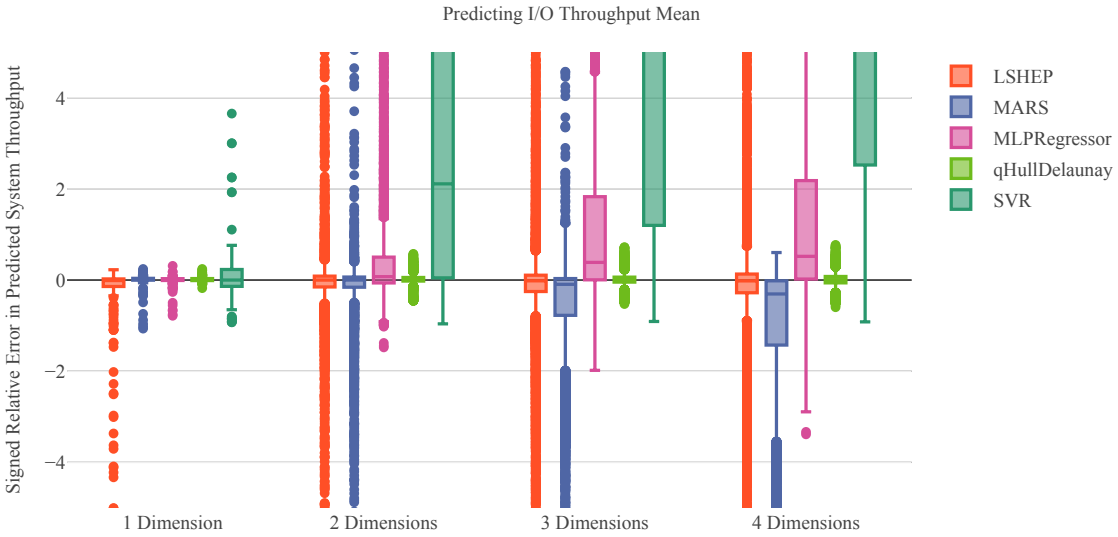


Figure 2: These box plots show the prediction error of mean with increasing dimension. The top box whisker for SVR is 40, 80, 90 for dimensions 2, 3, and 4, respectively. Notice that each model consistently experiences greater magnitude error with increasing dimension. Results for all training percentages are aggregated.

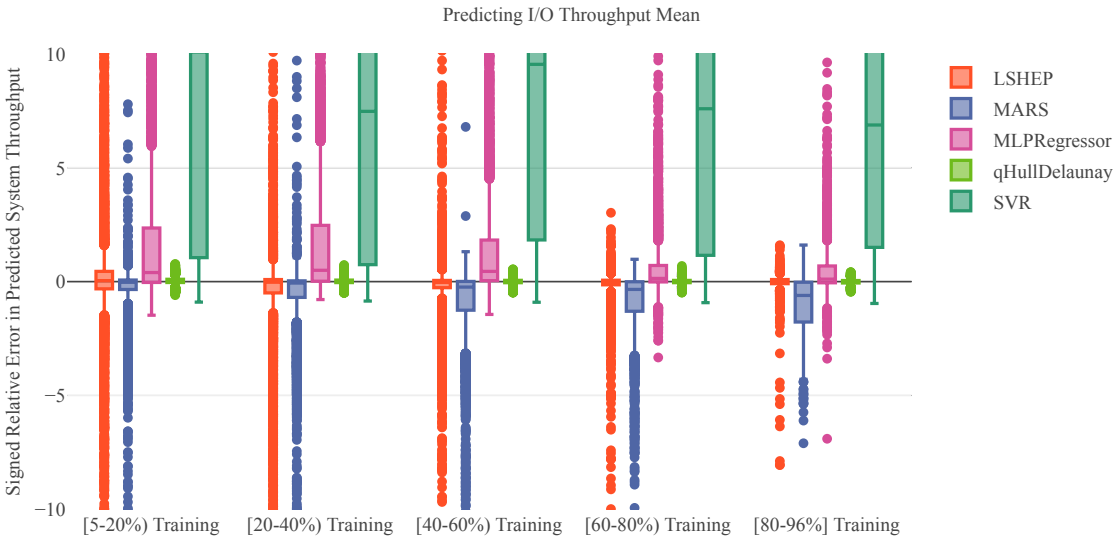


Figure 3: These box plots show the prediction error of mean with increasing amounts of training data provided to the models. Notice that MARS is the only model whose primary spread of performance increases with more training data. Recall that the response values being predicted span three orders of magnitude and hence relative errors should certainly remain within that range. For SVR the top box whisker goes from around 100 to 50 from left to right and is truncated in order to maintain focus on better models. Results for all dimensions are aggregated. Max training percentage is 96% due to rounding training set size.

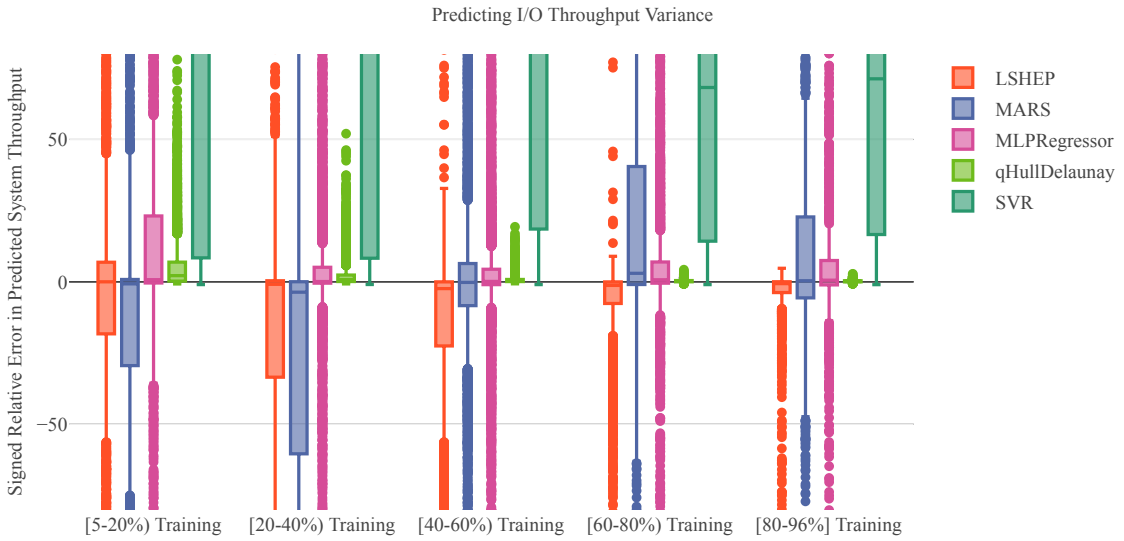


Figure 4: These box plots show the prediction error of variance with increasing amounts of training data provided to the models. The response values being predicted span six orders of magnitude. For SVR the top box whisker goes from around 6000 to 400 (decreasing by factors of 2) from left to right and is truncated in order to maintain focus on better models. Results for all dimensions are aggregated. Max training percentage is 96% due to rounding training set size.

## 5 DISCUSSION

The present results demonstrate that a straightforward application of multivariate modeling techniques can be used to effectively predict HPC system performance. Some modeling effort on the part of a systems engineer combined with a significant amount of experimentation (days of CPU time for the IOzone data used here) can yield a model capable of accurately tuning an HPC system to the desired performance specification, although qualitatively correct predictions can be achieved with much less (10%, say) effort.

### 5.1 Modeling the System

The modeling techniques generated estimates of drastically different quality when predicting I/O throughput mean and variance. A few observations: SVR has the largest range of errors for all selections of dimension and amounts of training data; MARS and LSHEP produce similar magnitude errors while the former consistently underestimates and the latter consistently overestimates; Delaunay has considerably fewer outliers than all other methods. SVR likely produces the poorest quality predictions because the underlying parametric representation is global and oversimplified (a single polynomial), making it unable to capture the complex local behaviors of system I/O. It is still unclear, however, what causes the behaviors of LSHEP, MARS, and Delaunay. An exploration of this topic is left to future work.

While the Delaunay method appears to be the best predictor in the present IOzone case study, it is important to note that the Delaunay computational complexity increases with the dimension of the input more rapidly than other techniques. The implementation of Delaunay (QuickHull) used would experience unacceptably large training times beyond ten-dimensional input. This leaves much room for other techniques to perform best in higher dimension unless a more efficient implementation of Delaunay can be used.

Finally, the ability of the models to predict variance was significantly worse than for the I/O mean. The larger scale in variance responses alone do not account for the increase in relative errors witnessed. This

suggests that system variability has a greater underlying functional complexity than the system mean and that latent factors are reducing prediction performance.

### 5.2 Extending the Analysis

System I/O throughput mean and variance are simple and useful system characteristics to model. The process presented in this work is equally applicable to predicting other useful performance characteristics of HPC systems such as: computational throughput, power consumption, processor idle time, context switches, RAM usage, or any other ordinal performance metric. For each of these there is the potential to model system variability as well. This work has chosen variance as a measure of variability, but the techniques used in this paper could be applied to more precise measures of variability such as the percentiles of the performance distribution or the entire distribution itself. A thorough exploration of HPC systems applications of multivariate modeling constitutes future work.

## 6 CONCLUSION

Multivariate models of HPC system performance can effectively predict I/O throughput mean and variance. These multivariate techniques significantly expand the scope and portability of statistical models for predicting computer system performance over previous work. In the IOzone case study presented, the Delaunay method produces the best overall results making predictions for 821 system configurations with less than 5% error when trained on only 43 configurations. Analysis also suggests that the error in the Delaunay method will remain acceptable as the number of system parameters being modeled increases. These multivariate techniques can and should be applied to HPC systems with more than four tunable parameters in order to identify optimal system configurations that may not be discoverable via previous methods nor by manual performance tuning.

### 6.1 Future Work

The most severe limitation to the present work is the restriction to modeling strictly ordinal (not categorical) system parameters. Existing statistical approaches for including categorical variables are inadequate for nonlinear interactions in high dimensions. Future work could attempt to identify the viability of different techniques for making predictions including categorical system parameters.

There remain many other multivariate modeling techniques not included in this work that should be evaluated and applied to HPC performance prediction. For I/O alone, there are far more than the four tunable parameters studied in this work. Alongside experimentation with more models, there is room for a theoretical characterization of the combined model and data properties that allow for the greatest predictive power.

## REFERENCES

Amos, B. D., D. R. Easterling, L. T. Watson, W. I. Thacker, B. S. Castle, and M. W. Trosset. 2014. "Algorithm XXX: QNSTOP—quasi-Newton algorithm for stochastic optimization". *Technical Report 14-02, Dept. of Computer Science, VPI&SU, Blacksburg, VA*.

Bailey, D. H., and A. Snavely. 2005. "Performance modeling: Understanding the past and predicting the future". In *European Conference on Parallel Processing*, pp. 185–195. Springer.

Barber, C. B., D. P. Dobkin, and H. Huhdanpaa. 1996, December. "The Quickhull Algorithm for Convex Hulls". *ACM Trans. Math. Softw.* vol. 22 (4), pp. 469–483.

Barker, K. J., K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. 2009. "Using performance modeling to design large-scale systems". *Computer* vol. 42 (11).

Basak, D., S. Pal, and D. C. Patranabis. 2007. "Support vector regression". *Neural Information Processing-Letters and Reviews* vol. 11 (10), pp. 203–224.

Beckman, P., K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj. 2008. "Benchmarking the effects of operating system interference on extreme-scale parallel machines". *Cluster Computing* vol. 11 (1), pp. 3–16.

Dahl, G. E., T. N. Sainath, and G. E. Hinton. 2013. "Improving deep neural networks for LVCSR using rectified linear units and dropout". In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013*, pp. 8609–8613. IEEE.

De, P., R. Kothari, and V. Mann. 2007. "Identifying sources of operating system jitter through fine-grained kernel instrumentation". In *IEEE International Conference on Cluster Computing*, pp. 331–340. IEEE.

Friedman, J. H. 1991. "Multivariate adaptive regression splines". *The Annals of Statistics*, pp. 1–67.

Friedman, J. H., and the Computational Statistics Laboratory of Stanford University. 1993. *Fast MARS.*

Grobelny, E., D. Bueno, I. Troxel, A. D. George, and J. S. Vetter. 2007. "FASE: A framework for scalable performance prediction of HPC systems and applications". *Simulation* vol. 83 (10), pp. 721–745.

Hornik, K., M. Stinchcombe, and H. White. 1989. "Multilayer feedforward networks are universal approximators". *Neural networks* vol. 2 (5), pp. 359–366.

Jones, E. and Oliphant, T. and Peterson, P. 2017. "SciPy: Open source scientific tools for Python". http://www.scipy.org/ [Online; accessed 2017-06-23].

Lee, D.-T., and B. J. Schachter. 1980. "Two algorithms for constructing a Delaunay triangulation". *International Journal of Computer & Information Sciences* vol. 9 (3), pp. 219–242.

Lofstead, J., F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf. 2010. "Managing variability in the IO performance of petascale storage systems". In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2010*, pp. 1–12. IEEE.

Møller, M. F. 1993. "A scaled conjugate gradient algorithm for fast supervised learning". *Neural networks* vol. 6 (4), pp. 525–533.

Norcott, W. D. 2017. "IOzone Filesystem Benchmark". http://www.iozone.org [Online; accessed 2017-11-12].

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* vol. 12, pp. 2825–2830.

Rudy, J. and Cherti, M. 2017. "Py-Earth: A Python Implementation of Multivariate Adaptive Regression Splines". https://github.com/scikit-learn-contrib/py-earth [Online; accessed 2017-07-09].

Sartipizadeh, H., and T. L. Vincent. 2016. "Computing the Approximate Convex Hull in High Dimensions". *CoRR* vol. abs/1603.04422.

Snavely, A., L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. 2002. "A framework for performance modeling and prediction". In *Supercomputing, ACM/IEEE Conference*, pp. 21–21. IEEE.

Thacker, W. I., J. Zhang, L. T. Watson, J. B. Birch, M. A. Iyer, and M. W. Berry. 2010. "Algorithm 905: SHEPPACK: Modified Shepard algorithm for interpolation of scattered multivariate data". *ACM Transactions on Mathematical Software (TOMS)* vol. 37 (3), pp. 34.

Wang, G., A. R. Butt, P. Pandey, and K. Gupta. 2009. "A simulation approach to evaluating design decisions in mapreduce setups". In *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'09)*, pp. 1–11. IEEE.

Wang, G., A. Khasymski, K. R. Krish, and A. R. Butt. 2013. "Towards improving mapreduce task scheduling using online simulation based predictions". In *International Conference on Parallel and Distributed Systems (ICPADS), 2013*, pp. 299–306. IEEE.

Ye, K., X. Jiang, S. Chen, D. Huang, and B. Wang. 2010. "Analyzing and modeling the performance in xen-based virtual cluster environment". In *12th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 273–280. IEEE.

## AUTHOR BIOGRAPHIES

**THOMAS C. H. LUX** is a Ph.D. student at Virginia Tech studying computer science under Dr. Layne Watson.

**LAYNE T. WATSON** (Ph.D., Michigan, 1974) has interests in numerical analysis, mathematical programming, bioinformatics, and data science. He has been involved with the organization of HPCS since 2000.

**TYLER H. CHANG** is a Ph.D. student at Virginia Tech studying computer science under Dr. Layne Watson.

**JON BERNARD** is a Ph.D. student at Virginia Tech studying computer science under Dr. Kirk Cameron.

**BO LI** is a senior Ph.D. student at Virginia Tech studying computer science under Dr. Kirk Cameron.

**LI XU** is a Ph.D. student at Virginia Tech studying statistics under Dr. Yili Hong.

**GODMAR BACK** (Ph.D., University of Utah, 2002) has broad interests in computer systems, with a focus on performance and reliability aspects of operating systems and virtual machines.

**ALI R. BUTT** (Ph.D., Purdue, 2006) has interests in cloud computing, distributed computing, and operating system induced variability.

**KIRK W. CAMERON** (Ph.D., Louisiana State, 2000) has interests in computer systems design, performance analysis, and power and energy efficiency.

**YILI HONG** (Ph.D., Iowa State, 2009) has interests in engineering statistics, statistical modeling, and data analysis.