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.

SpringSim-HPC, 2018 April 15-18, Baltimore, MD, USA;

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 non stationary 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 the sections, 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 four 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 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, $c, v \in \mathbb{R}$, $w_+ = \{w, w \geq 0, 0, w < 0\}$, 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 MARS (Rudy et al. 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 multi layer perceptron (MLP) produces a piecewise linear model of the input data. The computational complexity of training a multi layer 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 with $b \in \mathbb{R}$, and ε is the error tolerance. 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 $v^{(0)}, v^{(1)}, \ldots, v^{(d)}$, $x \in S$, 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, Eric and Oliphant, Travis and Peterson, Pearu and others 01) to the QuickHull implementation (Barber et al. 1996) of the Delaunay triangulation is used here.

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 Sample 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.

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

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

where $W_k(x)$ is a locally supported weighting function and $P_k(x)$ is a local linear approximation to the data satisfying $P_k(x^{(k)} = f(x^{(x)})$. The computational complexity LSHEP is $\mathcal{O}(n^2 d^3)$. This paper uses the FORTRAN95 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. 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. The 40 executions for each system configuration are converted into the mean and sample 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 number of input dimensions, 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.

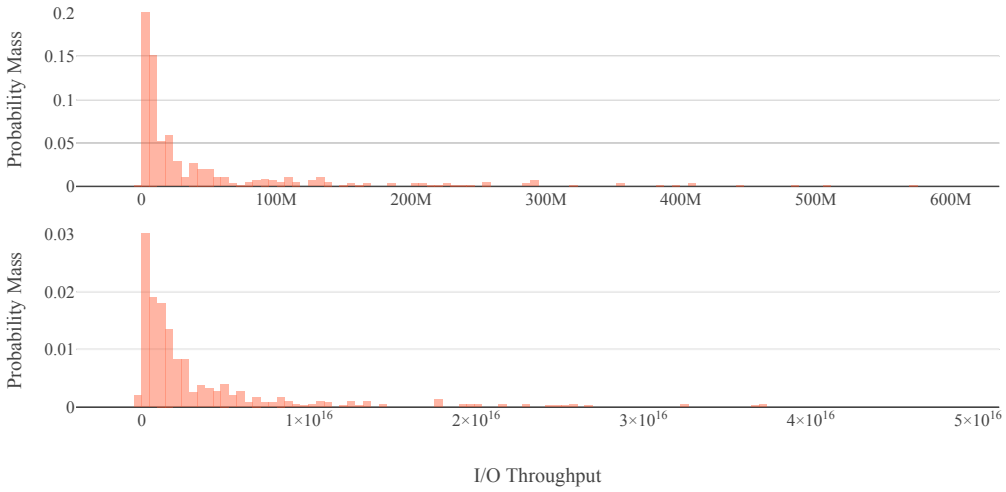

Figure 1: Histograms of 100-bin reductions of the PMF of I/O throughput mean (top) and I/O throughput sample variance (bottom). In the mean plot, the first 1% bin (truncated in plot) has a probability mass of .45. In the variance plot, the first 1% bin has a probability mass of .58. It can be seen that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

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 (features), the multidimensional analysis (MDA) is performed as follows:

1. for all $k = 1,\ldots,d$ and for all subsets of input features $F = \{j_1, j_2, \ldots j_k\}$; reduce the input data to unique points $z \in \mathbb{R}^k$ with $f(z) = E[\{f(x^{(i)}) \quad \forall i \quad s.t. \quad (x_F^{(i)} = z)\}]$.
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 generate each random $(train, test)$ split, ensure that all points from $test$ are on or inside the convex hull of points in $train$; also ensure that the points in $train$ are well spaced.

In order to ensure that the testing points are inside the convex hull of training points, this paper identifies the set of (reduced dimension) points that compose the convex hull and forcibly places those points in the training set. In order to ensure that training points are well spaced, this paper relies on the statistical method for picking points from Amos et al. (2014) that procedes as follows:

1. Generate a sequence of all pairs of points sorted by ascending pairwise L_2 distance between points, $(x^{(i_1)}, x^{(j_1)}), (x^{(i_2)}, x^{(j_2)}), \ldots$ such that $||x^{(i_k)} - x^{(j_k)}||_2 \leq ||x^{(i_{k+1})} - x^{(j_{k+1})}||_2$
2. Iteratively remove points from candidacy until only N remain by sequentially selecting one point from the pair $(x^{(i_k)}, x^{(j_k)})$ for $k = 1, \ldots$ if both $x^{(i_k)}$ and $x^{(j_k)}$ are still candidates.

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, those deterministically generated splits are used. Otherwise after $3 * N$ attempts,

only the unique splits are kept for analysis. The MDA procedure has been implemented in python3 while most regression and interpolation methods are FORTRAN wrapped with python. All randomness has been seeded for reproducability.

For any input feature subset F (of size k) and selected value of r, MDA will generate up to N predictions that each multivariate model has made for each point $z^{(i)} \in \mathbb{R}^k$. There may be fewer than N predictions made for any given point. Points in the convex hull for the selected subest of features 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 spacedness. Finally, as mentioned before, some subsets of features do not readily generate N unique training and testing splits.

MDA results in a distribution of possible estimates that a given multivariate model could make for a point in $\mathbb{R}^k$, $k \leq d$ given feature set F. The summary results presented in this work use the expected value of the distributions at each point as the model estimate for error analysis.

4 RESULTS

A naive multivariate prediction technique such as nearest neighbor could experience relative errors in the range $[0, 10^m]$ where m is $log_{10}(\max_f - \min_f)$ when modeling a function f from data. The IOzone mean data response values span 3 orders of magnitude (as can be seen in Table 1) while variance data response values span 6 orders of magnitude. It is expected therefore, that all studied multivariate models perform better than a naive approach, achieving relative errors strictly less than 10^3 for mean throughput and 10^6 when for throughput sample variance.

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 2 it can also be seen that the Delaunay model consistently makes good predictions even with as low as 5% training data (43 system configurations) regardless of the dimension of the data.

4.2 I/O Throughput Variance

The prediction results for variance resemble that of predicting I/O throughput mean, with each of the errors being scaled proportional to the change in scale of the response values. Delaunay remains the best overall predictor with LSHEP closely competing, but yielding much more outliers.

4.3 Increasing Dimension

As can be seen in figure 3, 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 particularly slow increase in error seen for Delaunay appears to suggest that it is capable of making predictions with a range of relative errors approximately spanning $\pm .2 * dim$.

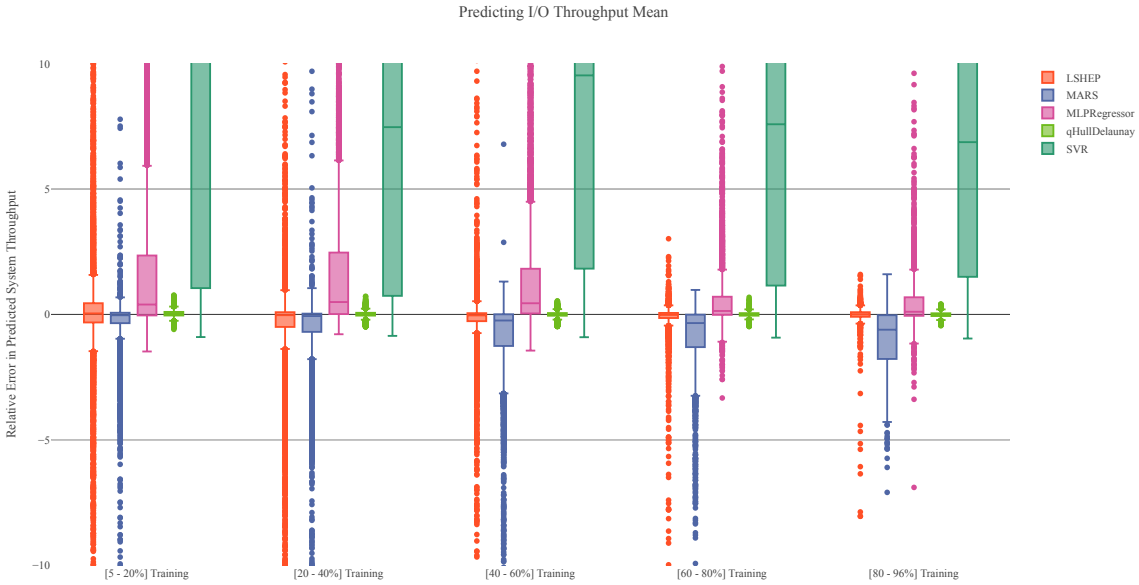

Figure 2: These box plots show the prediction error of means with increasing amounts of training data provided to the models. Notice that the response values being predicted span 3 orders of magnitude and hence relative error in that range would not be unexpected. For SVR the top box whisker goes from around 100 to 50 from left to right and is chopped in order to maintain focus on the more performant models.

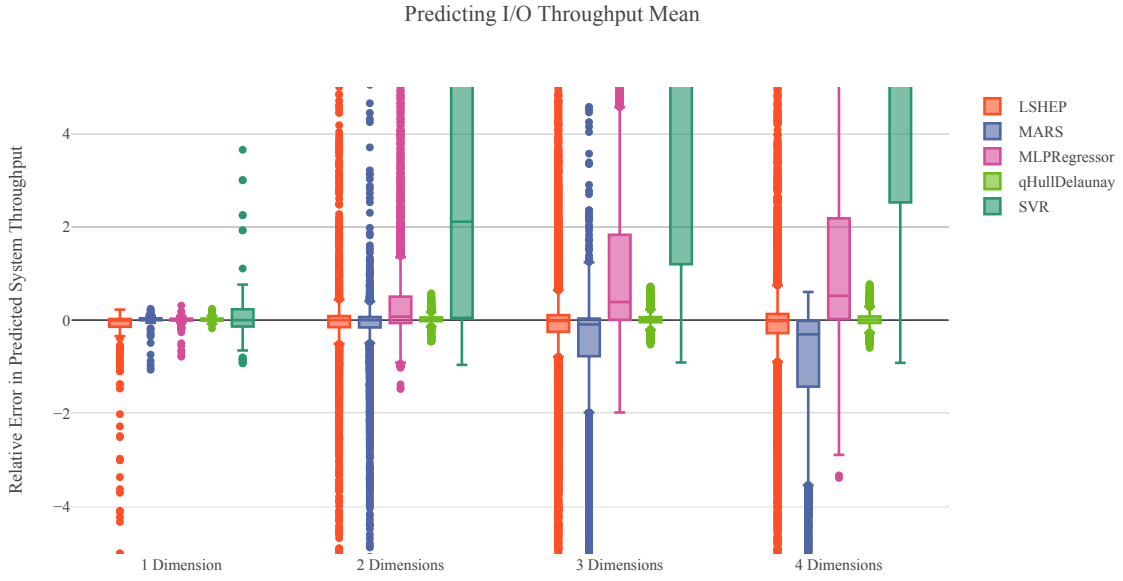

Figure 3: These box plots show the prediction error of means with increasing dimension.

5 DISCUSSION

5.1 Modeling the System

1. Promise of the models, especially Delaunay
2. Possible reasons for the performance differences between models
3. The need for as few as 50 experiments to get less than 5% error
4. The unfortunate inability of Delaunay to scale to higher dimension

5.2 Extending the Analysis

1. Measuring different system characteristics other than I/O
2. Recording system performance constantly to maintain model accuracy

6 CONCLUSION

1. Delaunay is the best model, capable of predicting the performance of 800 different system configurations accurately after only being trained on a well-spaced set of 50 configurations.
2. More comparisons of models similar to Delaunay should be done to evaluate what mathematical characteristics of multivariate models are most effective.

6.1 Future Work

The most severe limitation to the presented work is the present inability to model the relationship between categorical system parameters. This could be seen as a limitation of the models selected, because only MARS is capable of handling non-numeric variables, or as a limitation of the MDA framework. Future works could attempt to identify the viability of reducing categorical variables to numerical ones to expand set of predictable system configurations.

1. Using models to predict more complex responses, such as functions

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".

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 *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8609–8613. IEEE.

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

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

Friedman, J. H., and C. S. L. o. 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, Eric and Oliphant, Travis and Peterson, Pearu and others 2001–. "SciPy: Open source scientific tools for Python". [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 *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, 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, William D. 2017. "IOzone Filesystem Benchmark".

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, Jason and Cherti, Mehdi and others 2017. "Py-Earth: A Python Implementation of Multivariate Adaptive Regression Splines".

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 2002 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 *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, 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 *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, 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 *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, pp. 273–280. IEEE.