A Case Study on High-Performance Computing I/O Variability

Li Xu[1], Thomas Lux[2], Tyler Chang[2], Bo Li[2], Yili Hong[1], Layne Watson[2],

Ali Butt[2], Danfeng Yao[2], and Kirk Cameron[2]

[1]Department of Statistics, Virginia Tech, Blacksburg, VA 24061

[2]Department of Computer Science, Virginia Tech, Blacksburg, VA 24061

October 9, 2017

Abstract

Managing performance variability is an important issue in high-performance computing (HPC). The performance variability is affected by complicated interactions of numerous factors, such as CPU frequency, the number of I/O threads, and I/O scheduler. In this paper, we present statistical methods for modeling and analyzing HPC I/O variability. The objective is to predict performance variability both in interpolation and extrapolation settings. A complete analysis procedure is applied to deal with complex datasets. Statical and approximation methods are used to build predictive surface for the variability of HPC system. We evaluate the performance of the proposed method by using different criteria in prediction accuracy under new system configurations. We also discuss the methodology for future system configuration by using the estimated variability map.

Key Words: Computer Experiments; Linear Shepherd; MARS; Treed Gaussian Process; Performance Variability; System Variability.

Contents

1 Introduction

1.1 The Problem

High performance computing (HPC) is widely used to solve large-scale problems in various areas such as science, engineering, and business. While the performance of HPC attracts lots of research, the performance variability of HPC is also an important dimension of the problem that can not be ignored. For example, the time to finish a particular task may be varying from run to run. For another example, Figure 1 shows the performance variability of input/output (I/O) throughput (data transfer speed, in units of KB/s$\times 10^7$) as a function of CPU frequency, when keeping other variables of the system fixed. Here we use the standard deviation of I/O throughput from multiple runs as the performance variability measure (PVM). While defer the details of the figure to Section 2, the figure shows the performance variability increase as the CPU frequency increases. Although one can generally obtain higher throughput when increases the CPU frequency, the variability of the throughput from different runs also increases when the CPU frequency increases. For high CPU frequency, the variability is high and the standard deviation can exceed 50% of the mean.

In a big picture, performance variability in HPC systems is common and critical. Exponential increases in complexity and scale make variability a growing threat to sustaining HPC performance at exascale. However, the performance variability is affected by complicated interactions of numerous factors, such as CPU frequency, the number of threads, and I/O scheduler. The study of performance variability is an emerging area in computer science. The fundamental questions are: how do system variables affect the performance variability, can we predict performance variability for new system configurations, and can how to manage performance variability?

To answer those questions, computer scientists have done large scale experiments to collect necessary data. The objective of this paper is to conduct statistical modeling and predictions for performance variability. In particular, we use various statistical methods to describe system variability with system configuration variables as inputs, which we refer to as the variability map construction. With the variability map, we make prediction for new design points (i.e., new system configuration). We do two types of prediction: interpolation and extrapolation, both of which are of interest to computer scientists. Using prediction accuracy as the criterion, we evaluate and compare the performance of different methods, which yields insights in the use of different methods in the practice of HPC performance variability study. While it is impossible to eliminate variability, the statical models and prediction can help manage the variability. Being able to predict variability allows one to identify areas that can lead to large variability. Thus, one can avoid those system configurations that can lead to

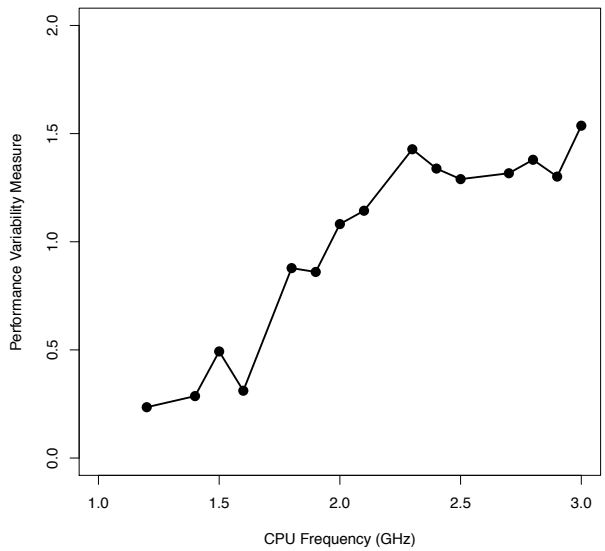


Figure 1: Example of performance variability of I/O throughput (in units of KB/s$\times 10^7$) as a function of CPU frequency, while keeping other variables fixed.

large performance variability, and design a system in the region where the performance and performance variability is optimized.

1.2 Related Literature

In the area of statistics, computer experiments are constructed to emulate a physical system. Because these are meant to replicate some aspect of a system in detail, they often do not yield an analytic solution. Due to the complexity and expensive of running, usually a surrogate model is used to describe the system behavior under some configuration. A surrogate model is an engineering method used when an outcome of interest cannot be easily directly measured, so a model of the outcome is used instead. Popular surrogate models (Chen et al. 2006) includes polynomial response surfaces, Kriging, gradient-enhanced Kriging, support vector machines, space mapping, and artificial neural networks (Bandler et al. 2004). In our work, we mainly use three classes of surrogate models, which are Gaussian process based methods (e.g., TGPlm, TGPcart, DynaTree, BART), inverse distance weighting methods (e.g., Shepard method) and non-parametric regression (e.g., MARS).

Gaussian process (GP) is a popular tools in computer experiments (e.g., Santner et al. 2010). However, GP forms a stationary surface. So when data has steep behavior, GP may not be able to capture the change. Several extensions has been make to let origin GP forms a

non-stationary surface. Gramacy and Lee (2008) build treed Gaussian Process and dynamic tree (Taddy et al. 2011). Monte Carlo Markov chain (MCMC) approach is used to build the tree structure based on algorithm CGM (Chipman et al. 2002). Another approach is called additive GP (Chipman et al. 2010). These tree-based non-stationary Gaussian Process use similar prior to build tree structures. Bayesian Treed Gaussian Process (TGPlm and TGPcart) uses a reversible-jump MCMC (Chipman et al. 2002). Additive GP uses prior defined in Chipman et al. 1998. Dynamic tree's prior extend the method in Chipman et al. 1998

Spline basis is widely used when the relation between input and response is highly nonlinear. In particular, the multivariate adaptive regression splines (MARS) (see, Friedman 1991) is a technique that uses an iterative approach to select B-spline basis for approximation of the unknown surface. The criterion used in selecting basis is related to the entire data. MARS does not have assumption of the type of the relation between response and input and can capture very complex relation adaptively. A generalized cross validation (GCV) procedure is used to determine the best model. For more details of GCV see Friedman (1991) Hastie, Tibshirani, and Friedman (2009). Adaptive splines are also adopted in other approximation to improve accuracy. For example in Neural network (Campolucci et al. 1996), Gaussian Process (Kim and Gu 2004) and so on.

For inverse distance weighting methods, the assigned values to unknown points are calculated with a weighted average of the values available at the known points. The known earliest IDW interpolation method was Shepard Method (Shepard 1968). Many extensions has been make based on Shepard's work such as Renka (1988), Berry and Minser (1999) and Gordon and Wixom (1978) .

1.3 Overview

The rest of the paper is organized as follows. Section 2 introduce the experiment setup and data collection. Section 3 introduce various methods that can be used for variability map construction. Section 4 compares different methods for their performance in interpolation and extrapolation. Section 5 presents the optimization of performance variability. Section 6 contains some concluding remarks and areas for future research.

2 Experiment Setup and Data Collection

2.1 Performance Variability Measure

While HPC performance can be a general term, in this paper, we focus on one specific metric, which is input/output (I/O) throughput. Thus, the performance variability measure (PVM) used in this paper is the I/O throughput. The modeling and analysis approach used in this paper, however, can be extended to other metrics. Here, the I/O throughput is defined as the data transfer speed in term of kilobytes per second (KB/s). Event under the same system configuration, different runs will end up with different I/O throughput. That is performance variability exits in I/O throughput. The objective is to study how system variables affect the I/O throughput variability.

2.2 Experiment Setup

The I/O throughput is affected by a list of variables such as hardware settings, operating system settings, and application settings. To study how the system configurations affect the I/O performance variability, computer scientists conducted a complicated large-scale experiments. In this section, we introduce the setup of the experiment. The response variable is the throughput variability of a single node running IOzone. Under the same system setting $\boldsymbol{x}$, a large number of runs were done and the standard deviation of the I/O throughput of those runs is used as the PVM of the I/O throughput, which is used as the quantitative measure of the performance variability.

For system configurations, seven important variables were selected by computer scientists, which consist of categorical and continuous variables. Table 1 shows the summary information for variables that are used in the IOzone throughput experiments. In general, the variables can be grouped into three categories: hardware settings, operating system settings, and application settings. The CPU clock frequency is the hardware setting. The Frequency variable has 15 levels, which are shown in Table 1, ranging from 1.2 GHz to 3.0 GHz. The I/O Scheduler and virtual machine (VM) I/O Scheduler are operating system settings. The I/O Scheduler has three types, which are CFQ, DEAD, and NOOP. VM I/O Scheduler has the same setting as the I/O Scheduler.

The applications setting has four variables, which are the I/O Operation Mode, the Number of Threads, File Size (KB), and Record Size (KB). The I/O Operation Mode has 13 levels, such as Fread and Fwrite, representing different types of reading and writing tasks. The Number of Threads has 9 levels, ranging from 1 to 256 on a basis of power of 2. The values of File Size and Record Size are chosen in pairs as there are some constraints on the selection of values. The File Size has to be larger than the Record Size, and has be a multiple of the Record Size.

Table 1: Summary information for variables that are used in the IOzone throughput experiments to collect training set. The points, at which the Frequency and the Number of Thread are (2.8, 256), (2.9, 256), (3.0, 256), (3.0, 64), and (3.0, 128), given the File Size and Record Size is (256, 32), are for testing purpose.

Category	Variable	No. of levels	Values
Hardware	CPU Clock Frequency (GHz)	15	1.2, 1.4, 1.5, 1.6, 1.8, 1.9, 2.0, 2.1 2.3, 2.4, 2.5, 2.7, 2.8, 2.9, 3.0
Operating System	I/O Scheduler	3	CFQ, DEAD, NOOP
	VM I/O Scheduler	3	CFQ, DEAD, NOOP
Application	I/O Operation Mode	13	Fread, Pread, Re-read, Randomread Read, ReverseRead, Strideread Fwrite, Pwrite, Randomwrite, Rewrite Initialwrite, Mixedworkload
	Number of Threads	9	1, 2, 4, 8, 16, 32, 64, 128, 256
	File Size (KB) by Record Size (KB)	6	(64, 32), (256, 32), (256, 128) (1024, 32), (1024, 128), (1024, 512)
Total no. of points: $3 \times 3 \times 13 \times 5 \times 15 \times 9 - 3 \times 3 \times 13 \times 1 \times 5 = 94185$			

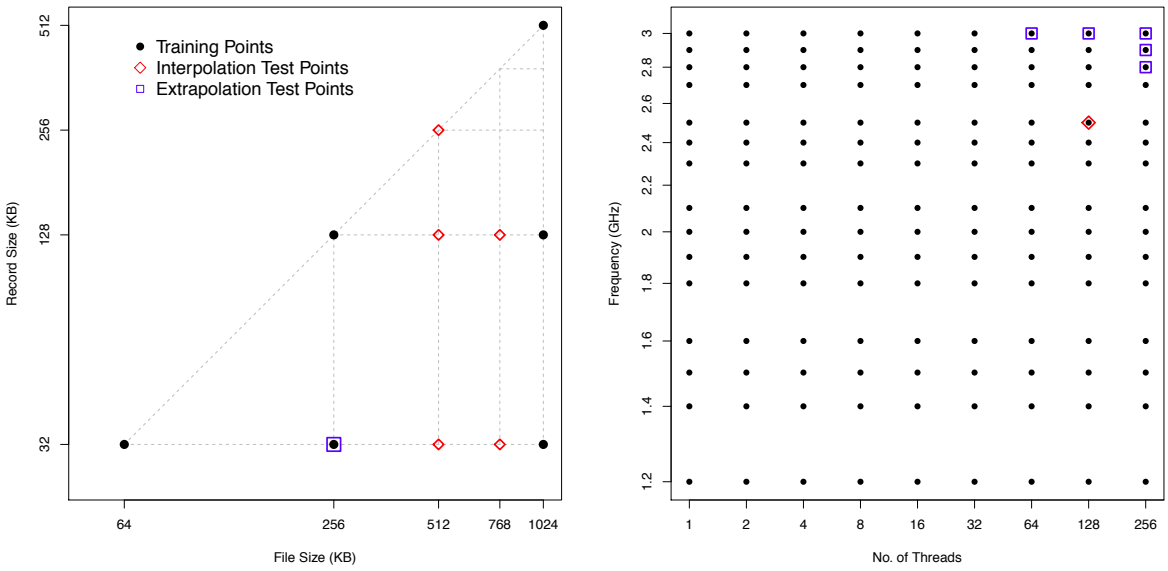
As shown in Table 1, six valid combinations were used in the experiment. Figure 2 illustrates of design points for File Size, Record Size, Number of Threads, and Frequency, as shown by black dots. Figure 2 also displays some points that are used for testing purpose, which will be introduced in next section.

2.3 Data Collection

Three sets of data were collected, which are the training set, the interpolation test set, and the extrapolation test set. Each dataset contains 8 columns with one column for the PVM, which is the response/output variable, and 7 columns for the explanatory/input variables. The I/O Scheduler, VM I/O Scheduler and I/O Operation Mode are treated as categorical variables, and the Frequency, Number of Threads, File Size and Record Size are treated as continuous variables.

The training set is used to build model for prediction of variability. The training set consists of

$$3 \times 3 \times 13 \times 5 \times 15 \times 9 - 3 \times 3 \times 13 \times 1 \times 5 = 94185$$



(a) File Size and Record Size (b) Number of Threads and Frequency

Figure 2: Illustration of design points for File Size, Record Size, Number of Threads, and Frequency. The points, at which the Frequency and the Number of Thread are (2.8, 256), (2.9, 256), (3.0, 256), (3.0, 64), and (3.0, 128), given the File Size and Record Size is (256, 32), are for testing purpose.

~~number of~~ points. The training set essentially is a full factorial of all levels for the variables with 585 points excluded.

The excluded points are at which the Frequency and the Number of Thread are (2.8, 256), (2.9, 256), (3.0, 256), (3.0, 64), and (3.0, 128), given the File Size and Record Size is (256, 32). Figure 2 displays the points that are used for extrapolation testing purpose, which are marked by blue squares. The points are on the right upper corner of the combination of Frequency and the Number of Threads. Thus, the main purpose is to test the extrapolation of the variables of Frequency and the Number of Threads.

There are an additional $3 \times 3 \times 13 \times 5 \times 1 \times 1 = 585$ points used for interpolation testing purpose. Those points are marked by red diamonds in Figure 2. Specifically, the chosen points are the File Size and Record Size at (512, 32), (512, 128), (512, 256) (768, 32), and (768, 128), given the Number of Threads is 128 and the Frequency is 2.5 GHz. The points lies in the interior ~~region or boundary~~ of the convex hull of File Size and Record Size testing points. Thus, the main purpose is to test the interpolation of the variables of File Size and Record Size. Table 2 shows the summary of experiment setup for collection of test sets.

Table 2: Summary of experiment setup for collection of test sets.

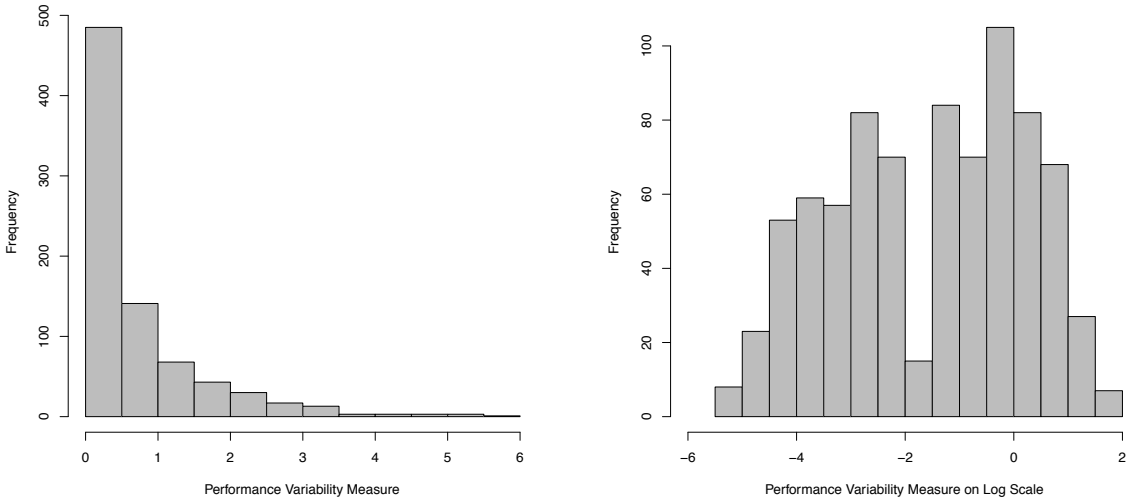
Variable	Interpolation Test Set	Extrapolation Test Set
I/O Scheduler	All 3 levels	All 3 levels
VM I/O Scheduler	All 3 levels	All 3 levels
I/O Operation Mode	All 13 levels	All 13 levels
File Size by	(512, 32), (512, 128), (512, 256)	256
Record Size	(768, 32), (768, 128)	32
Frequency	2.5	(2.8, 256), (2.9, 256), (3.0, 256)
Number of Threads	128	(3.0, 64), (3.0, 128)
Total No. of Points	585	585

3 Variability Map Construction

3.1 Data Visualization and Variability Map

In this section, we first do some data visualization to explore the data. In the dataset, we have seven explanatory variables. The I/O operation Mode, I/O Scheduler, and VM I/O Scheduler are categorical variables, and GPU Frequency, File Size, Record Size, and the Number of Threads are considered to be continuous variables. Figure 3 shows one example for a set of system variables before and after log transformation using the training set. Note that the scale of the I/Othroughput is in the magnitude of 10^7. The system configuration for the dataset shown in the figure is chosen as follows. The IO Operation Mode is Fwrite, IO Scheduler is CFQ, and the VM IO Scheduler is NOOP. With this setting, Figure 3 shows the PVM of the 805 points is right skewed. After the log transformation, the PVM of the 805 points shows a bimodal behavior, which shows for certain configurations, the variability tends to be small, while for other configurations, the variability tends to be large.

To further visualize how PVM changes across different system settings, Figure 4 shows the perspective plots of PVM as a function of CPU Frequency and the Number of Threads, before and after taking a log transformation. The IO Operation Mode is Fwrite, IO Scheduler is CFQ, the VM IO Scheduler is NOOP, the File Size is 1024, and the Record Size is 512. The figure shows that, in general, the PVM increases as the Frequency and the Number of Threads increase. However, the exact relationship is complicated, and can not be described by simple functional relationships. While the figure can only shows the pattern for two variables, the actual relationship is more complicated due many other variables can affect the PVM.



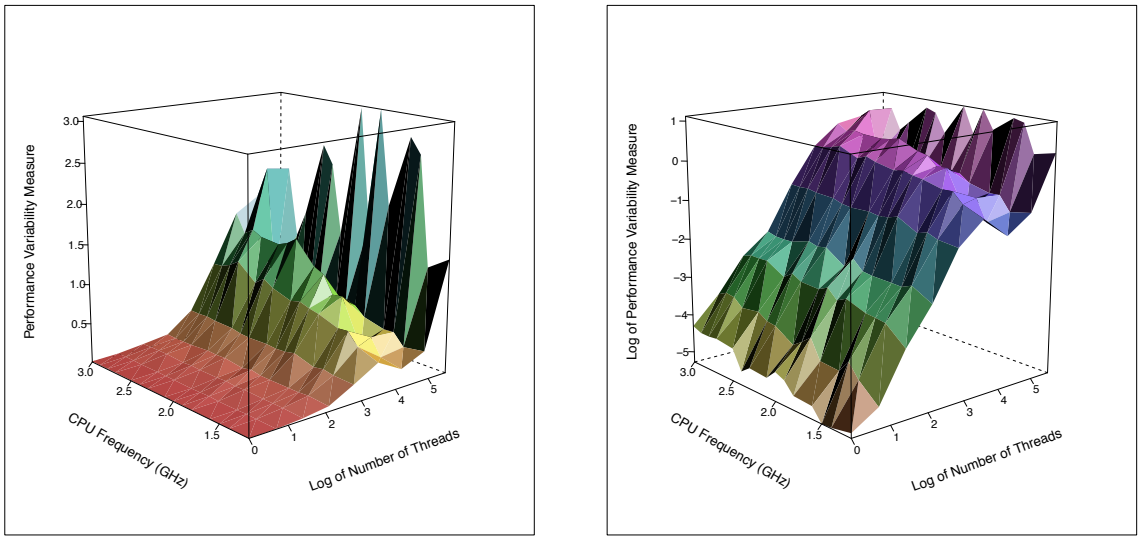
(a) PVM (b) Log of PVM

Figure 3: One example for a set of system variability before and after log transformation (in a scale of 10^7). The IO Operation Mode is Fwrite, IO Scheduler is CFQ, and the VM IO Scheduler is NOOP.

We introduce the concept of variability map to describe the functional relationship between the system configuration and the PVM. Let $\boldsymbol{x}$ be a vector of parameters. The variability map is defined as a function $f(\boldsymbol{x})$ that gives PVM at $\boldsymbol{x}$. With the variability map $f(\boldsymbol{x})$ estimated from the training dataset, one can use it to characterize and to predict variability for a given configuration $\boldsymbol{x}$. Because most approximation methods work for continuous variables, we only consider the modeling of continuous variable in this paper. For categorical variables, we do a separate estimation for the variability map for each unique combination of the variables. For example, the I/O operation Mode, I/O Scheduler, and VM I/O Scheduler has 117 unique combinations for their levels. Thus, 117 separate variability maps will be constructed. Without loss of generality, we only describe only to construction one particular variability map. ~~In particular,~~ we define $\boldsymbol{x}$ as follows,

$$\boldsymbol{x} = (\text{Log of File Size, Log of Recored Size, Log of No. of Threads, CPU Frequency})'. \quad (1)$$

The training set is denoted by $\{y_i, \boldsymbol{x}_i\}$, $i = 1, \cdots, n$, where y_i is the PVM under the system configuration $\boldsymbol{x}_i$ for a particular combination of the I/O operation Mode, I/O Scheduler, and VM I/O Scheduler. Here $n = 805$ is the number of data points for each variability map.



(a) PVM (b) Log of PVM

Figure 4: Perspective plots of PVM as a function of CPU Frequency and the Number of Threads, before and after taking a log transformation. The IO Operation Mode is Fwrite, IO Scheduler is CFQ, the VM IO Scheduler is NOOP, the File Size is 1024, and the Record Size is 512.

3.2 Description of Candidate Methods

In this section, we give a short description of candidate methods that will be used for the construction of the variability map. We first consider multiple linear regression model, which is a simple model. We also consider several popular classes of surrogate models as the candidate methods, which are the linear Shepard (LSP) method that is based on distance inverse weight, the multivariate adaptive splines (MARS), Gaussian process based methods and their extensions, and tree based model such as the Bayesian additive regression trees (BART). Those methods are flexible and have the potential to capture complicated local nonlinear relationship, providing the possibility to uncover interactions between variables that affect complex I/O variability.

Linear Model. For the linear model, the response y_i is modeled as

$$y_i = \beta_0 + \boldsymbol{x}_i'\boldsymbol{\beta} + \varepsilon_i,$$

where the variability map $f(\boldsymbol{x})$ is modeled as $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}'\boldsymbol{\beta}$, β_0 and $\boldsymbol{\beta}$ are the regression coefficients, and ε_i is the error term. The parameter estimation can be done by using the least-squares method.

The Linear Shepard Method. The LSP is approximation algorithm based a modification of the original Shepard algorithm (e.g., Thacker et al. 2010). To introduce the LSP,

we need to define some notation first. Let p be the length of $\boldsymbol{x}$ and $m = \min\{n, 3p/2\}$. Let $d_i(\boldsymbol{x}) = \|\boldsymbol{x}-\boldsymbol{x}_i\|_2$ be the distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$ and let $d = \max_{i,j} d_i(\boldsymbol{x}_j)$ be the maximum distance among all pairs of $\boldsymbol{x}_i$. Let r_i be the smallest radius of the closed ball around $\boldsymbol{x}_i$ such that at least m number of data points are included in the ball. Let $u_i = \min\{d/2, r_i\}$ and $v_i = 1.1 \cdot r_i$.

In the LSP, for an arbitrary point $\boldsymbol{x}$, the variability $f(\boldsymbol{x})$ is estimated as $\widehat{f}(\boldsymbol{x})$. In particular,

$$\widehat{f}(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} w_i(\boldsymbol{x}) p_i(\boldsymbol{x})}{\sum_{i=1}^{n} w_i(\boldsymbol{x})},$$

where $w_i(\boldsymbol{x}) = \{[1/d_i(\boldsymbol{x}) - 1/u_i]_+\}^2$. We use the locally weighted least-squares fit to obtain the $p_i(\boldsymbol{x})$ with the following form, $p_i(\boldsymbol{x}) = y_i + (\boldsymbol{x} - \boldsymbol{x}_i)'\boldsymbol{\beta}_i$, where $\boldsymbol{\beta}_i$ is the vector for coefficients. The weight is defined as $\omega_{ij} = \{[1/d_i(\boldsymbol{x}_j) - 1/v_j]_+\}^2$ for those points within the ball for $\boldsymbol{x}_i$ and $\omega_{ij} = 0$ for those points are outside the ball. Then, the coefficient $\boldsymbol{\beta}_i$ can be obtained by solving the weighted least-squares problem.

Multivariate Adaptive Regression Splines. Let $\boldsymbol{x}_i = (x_{i1}, \cdots, x_{ij}, \cdots, x_{ip})'$. For any $\boldsymbol{x} = (x_1, \cdots, x_j, \cdots, x_p)'$, the MARS bases are defined as, $\mathcal{C} = \left\{(x_j - t)_+, (t - x_j)_+\right\}$ for $t \in \{x_{1j}, x_{2j}, \ldots, x_{nj}\}$, $j = 1, 2, \ldots, p$. Using the MARS method, the variability map $f(\boldsymbol{x})$ is estimated as,

$$\widehat{f}(\boldsymbol{x}) = \widehat{\beta}_0 + \sum_{l=1}^{m} \widehat{\beta}_l h_l(\boldsymbol{x}),$$

where β_0 and β_l are coefficients, and $h_l(X)$ is a function in $\mathcal{C}$ or a product of two or more functions in $\mathcal{C}$. A step-wise procedure is used for the selection of bases from the set $\mathcal{C}$. One starts with constant function $h_0(\boldsymbol{x}) = 1$ in the model. At each step, one adds a term of the following form which minimizes the square of error most to the current model,

$$\widehat{\beta}_{m+1} h_l(\boldsymbol{x}) \cdot (x_j - t)_+ + \widehat{\beta}_{m+2} h_l(\boldsymbol{x}) \cdot (t - x_j)_+,$$

where $h_l(\boldsymbol{x})$ are the terms in the current model. At the end of the step-wise procedure, a large model may result and thus a backward procedure is performed to eliminate some terms. To determine the best model, the generalized cross-validation procedure is ~~usually~~ used.

Bayesian Treed Gaussian Process. Treed Gaussian processes (TGP) are widely used methods in modeling and prediction of data collected from computer experiments. In general, TGP uses Bayesian methods to build a classification and regression model (CART), and it partitions the parameter space into disjoint regions using a tree structure. In each region, a local Gaussian process is fitted to the data from that region. Thus, a TGP typically consists of a tree structure construction and local Gaussian process fitting, which provide the potential to capture complex relations between system configuration $\boldsymbol{x}$ and the performance variability

$f(\boldsymbol{x})$. We consider three types of TGP are considered in this paper, which are TGP with linear trend (denoted as TGPlm), TGP with constant mean (denoted as TGPcart), and dynamic trees (denoted as DynaTree).

The treed partition models use binary splits on the value of univariate variables to divide the input space. Each partition is recursive, and a new partition can occur on previous divided regions. Other tree construction methods such swapping, pruning, rotation can take place after the splitting.

For a specific region, the data from a specific region j are denoted by X_j and $\boldsymbol{y}_j$, where each row of X_j contains the one system configuration $\boldsymbol{x}$ and $\boldsymbol{y}_j$ is a vector of response y_i from the region. For each region we use the following Gaussian process model,

$$\boldsymbol{y}_j \sim \mathrm{N}[\mu(X_j, \boldsymbol{\theta}_j), \Sigma_j].$$

Here $\mu(X_j, \boldsymbol{\theta}_j)$ is the mean structure which is function of X_j with unknown parameters $\boldsymbol{\theta}_j$, and Σ_j is the variance-covariance matrix. With specification of priors, the parameters can be estimated through Gibbs sampling. The estimation and prediction of $f(\boldsymbol{x})$ at any $\boldsymbol{x}$ can be done using the posterior distribution.

TGPlm uses linear trend GP process in local partition while TGPcart uses constant mean. Dynamical Trees can explore posterior tree space by stochastically proposing incremental modifications to tree structure (such as splitting, pruning, and swapping) in each terminal leaf.

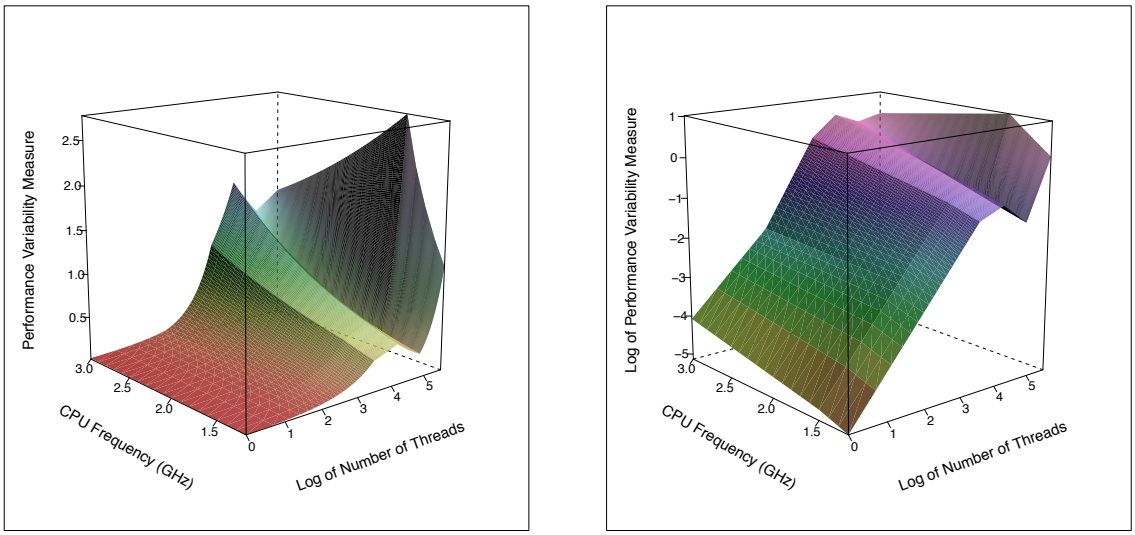
Bayesian Additive Regression Trees. The Bayesian additive regression trees (BART) consists of two components, which are a sum-of-trees model and a regularization prior on the sum-of-tree model parameters. For a BART model with m trees, the variability map $f(\boldsymbol{x})$ can be modeled as

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i = \sum_{j=1}^{m} g(\boldsymbol{x}_i;\, T_j;\, \boldsymbol{\theta}_j) + \varepsilon_i.$$

Here the error term $\varepsilon_i \sim \mathrm{N}(0, \sigma^2)$, and T_j is a binary regression tree that partitions the parameter space. The associated terminal leaf parameters for tree T_j is denoted by $\boldsymbol{\theta}_j$. The function $g(\boldsymbol{x};\, T_j;\, M_j)$ provides the output values for $\boldsymbol{x}$ according to tree T_j. With the specification of the prior distributions, the Bayesian back-fitting MCMC algorithm can be used for model estimation and prediction.

3.3 Estimation and Visualization of Variability Maps

In this section, we use the training set in Table 1 to estimate the variability map. The candidate methods described in Section 3.2 are used. The log-transform is applied to the response variable. For the linear regression model , we use four predictors, namely, the log of



(a) PVM (b) Log of PVM

Figure 5: Perspective plots of estimated variability map as a function of CPU Frequency and the Number of Threads, using the MARS algorithm. The IO Operation Mode is Fwrite, IO Scheduler is CFQ, the VM IO Scheduler is NOOP, the File Size is 1024, and the Record Size is 512.

File Size, the log of Record Size, the log of the Number of Threads, and the CPU frequency. For MARS, the order of the highest basis function is set to three and the exclusive method is used to add new basis function in each iteration. For the DynaTree method, the linear terminal GP is used.

For each distinct combination of those categorical variables, we generate a separate variability maps, resulting 117 variability maps. Each variability map is a four dimensional surface. To visualize the estimated variability map, Figure 5 shows two perspective plots of estimated variability map as a function of CPU Frequency and the Number of Threads, using the MARS algorithm. The system setting of Figure 5 is the same as in Figure 4, with IO Operation Mode fixed at Fwrite, IO Scheduler at CFQ, VM IO Scheduler at NOOP, File Size at 1024, and Record Size at 512. From the figure, we see that the estimated variability map by MARS can capture most patterns in raw data as shown in Figure 4.

4 Prediction and Method Comparisons

4.1 Prediction Problems and Criteria for Comparisons

Although there are many methods available in computer experiment literature, the applicability of ~~the~~ existing methods to the performance variability study is not clear. In this section, we conduct extensive comparisons to study the properties of existing methods. In particular, we are interested in whether the models can predict the performance variability well for a new system configuration $\boldsymbol{x}$. The prediction problem can be categorized into two types, which are interpolation and extrapolation. For interpolation, the prediction is to be made for a $\boldsymbol{x}$ that lies inside the input space of the training set. For extrapolation, the prediction is to be made for a $\boldsymbol{x}$ that lies outside the input space of the training set. Both problems are important to the variability study of HPC systems.

To evaluate the performance of the candidate methods, we need to discuss our comparisons criteria first. Let y_k be the true value and $\widehat{y}_k$ be the predicted value, for $k = 1, \cdots, n$. Here n is both 585 for the interpolation and extrapolation test sets, after combining the points from the 117 variability maps. The root mean squared error (RMSE) is defined as $\sqrt{\sum_{k=1}^{n}(\widehat{y}_k - y_k)^2/n}$, while the mean absolute error (MAE) is defined as $\sum_{k=1}^{n}|\widehat{y}_k - y_k|/n$. The RMSE and MAE have the same scale as the original data. Several other relative errors are defined as,

$$\mathrm{ER}_1 = \frac{\sum_{k=1}^{n}|\widehat{y}_k/y_k - 1|}{n}, \quad \mathrm{ER}_2 = \frac{\sqrt{\sum_{k=1}^{n}(\widehat{y}_k - y_k)^2/n}}{\sum_{k=1}^{n} y_k/n}, \quad \text{and} \quad \mathrm{ER}_3 = \frac{\sum_{k=1}^{n}|\widehat{y}_k - y_k|/n}{\sum_{k=1}^{n} y_k/n}.$$

The ER_1 is the average of the individual-wise error rate. However, ER_1 tends to be large when the true value is small. The ER_2 is the RMSE divided by the estimated mean $\sum_{k=1}^{n} y_k/n$ to provide a scale-free measure of error. However, RMSE tends to be dominated by those terms with large errors. The ER_3 is the MAE divided by the estimated mean to provide a scale-free measure. Because each error measure has its pros and cons, we will present our results under different error measures to provide a comprehensive comparison.

4.2 Interpolation

In this section, we make a comparison for predictability of different methods when a new design point lies within the input space (i.e., interpolation). Using the training set with 94185 data points, 117 variability map are constructed by using the seven candidate methods. The log transformation for the responses is used. For each variability map, there are five points to be predicted, as shown in Table 2. In total there are $117 \times 5 = 585$ points in the test set for interpolation.

Various error measures are then computed. Table 3 shows the RMSE, MAE, and three error rates for the seven candidate methods based on the interpolation test set. The RMSE

Table 3: The RMSE, MAE, and three error rates for the seven candidate methods based on the interpolation test set. The RMSE and MAE are in the magnitude of 10^7.

Method	ER_1	RMSE	ER_2	MAE	ER_3
LM	0.357	1.128	0.226	0.822	0.164
LSP	0.216	1.269	0.254	0.891	0.178
MARS	0.220	0.911	0.182	0.693	0.138
TGPlm	0.450	2.097	0.419	1.331	0.226
TGPcart	0.439	3.103	0.621	1.767	0.353
DynaTree	0.351	4.037	0.807	1.562	0.312
BART	0.215	1.165	0.233	0.829	0.166

and MAE are in the magnitude of 10^7. Even under different error measures, the MARS method turns out to be the best one, resulting the smallest error rate. ~~The~~ LSP and BART generate results that are next to the the MARS. The linear model (LM) is in the middle. The three TGP based methods tend to have larger errors than other methods. To investigate the spread of error rates, Figure 6 shows a box plot for the 585 individual relative error rates ($\widehat{y}_k/y_k$) for the seven candidate methods. One can see that the LSP method has the least spread, and the MARS and LM have the least median error rates. Overall, the MARS method tends to work well for interpolation prediction problems.

4.3 Extrapolation

In this section, we make a similar comparison for predictability of different methods when a new design point lies outside the input space (i.e., extrapolation). Similar to the setting of interpolation test, for each variability map, there are five points to be predicted, as shown in Table 2. In total there are $117 \times 5 = 585$ points in the test set for extrapolation. We find that the models with log transformation the responses tend to generate nonsensical large error for extrapolation due to the effect of anti-log. Thus, for the extrapolation, we model the response at its original scale.

Table 4 shows the various error rates based on the extrapolation test set. Among all error measures, the LSP shows the best result. The MARS is next to the LSP and the TGP methods tend to generate larger errors. Figure 7 shows the box plot for the 585 individual relative error rates. The figure shows that ~~the~~ LSP has the least amount of spread in term of individual errors. Most ~~of~~ surrogate models in computer experiments are built for interpolation and their extrapolation ability is not well studies. Surprisingly, we find that the LSP method

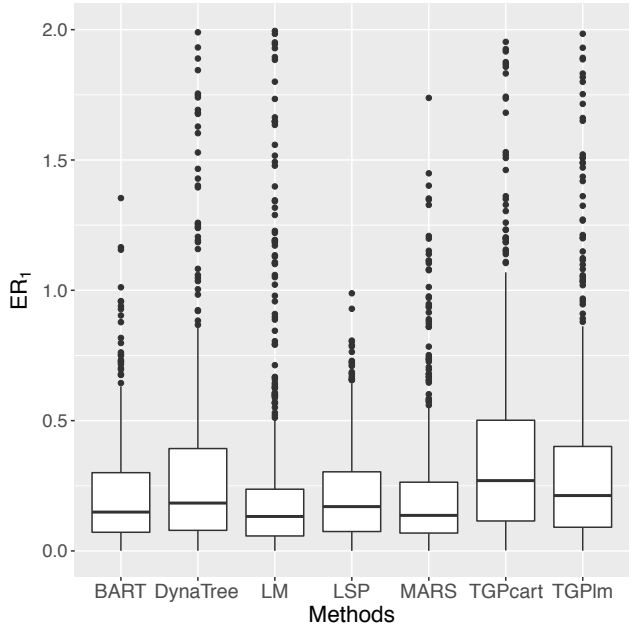


Figure 6: Box plot for the 585 individual relative error rates for the seven candidate methods based on the interpolation test set. Some outliers are above the y-axis limit because of too high error rate for methods: LM, TGPlm, TGPcart, DynaTree.

has good extrapolation ability under our current test setting. The advantage of the LSP in extrapolation could be due to its weighted sum of the local linear fits. Overall, the LSP shows good . However, one has notice that extrapolation is always a difficult task and should be exercised with caution.

5 Management of Performance Variability

5.1 The System Design Problem

In this section, we address the question of how to do variability management. One important task is to design a system ~~that~~ with minimum performance variability, while the performance of the system still meets the requirement. Let $\boldsymbol{x}$ be the system configuration, and $\widehat{f}(\boldsymbol{x})$ be the estimated variability map. To do the system design, one also needs to know the performance surface $m(\boldsymbol{x})$ at at any $\boldsymbol{x}$. The performance surface is the mean throughput at $\boldsymbol{x}$. With the training set, ~~the~~ $m(\boldsymbol{x})$ can be estimated by using the methods discussed in Section 3.2, which is denoted by $\widehat{m}(\boldsymbol{x})$. Let $\mathcal{D}$ be the feasible domain of $\boldsymbol{x}$. Let m_0 be the minimum performance requirement. The system design problem can be formulated as the following optimization

Table 4: The RMSE, MAE, and three error rates for the seven candidate methods based on the extrapolation test set. The RMSE and MAE are in the magnitude of 10^7.

Method	ER_1	RMSE	ER_2	MAE	ER_3
LM	0.337	1.970	0.361	1.542	0.282
LSP	0.099	0.597	0.109	0.392	0.072
MARS	0.194	1.417	0.259	0.920	0.169
TGPlm	0.509	10.62	0.650	8.462	0.518
TGPcart	0.534	11.22	0.687	9.055	0.554
DynaTree	0.462	10.17	0.622	7.856	0.481
BART	0.441	9.802	0.600	7.451	0.456

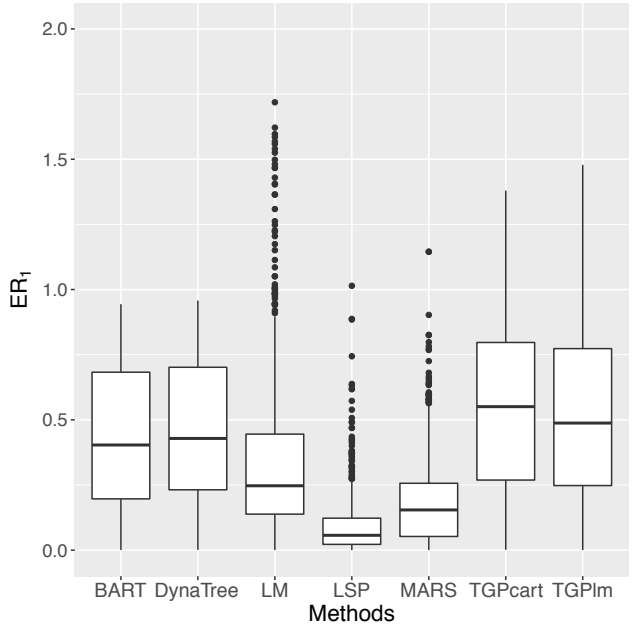


Figure 7: Box plot for the 585 individual relative error rates for the candidate methods based on the extrapolation test set. Some outliers are above the y-axis limit because of too high error rate for method: TGPlm.

problem. That is

$$\min_{\boldsymbol{x}\in\mathcal{D}} \widehat{f}(\boldsymbol{x}), \quad \text{subject to} \quad \widehat{m}(\boldsymbol{x}) \geq m_0. \tag{2}$$

The optimization problem in (2) is equivalent to minimize the following cost function,

$$c(\boldsymbol{x}) = \begin{cases} \infty, & \widehat{m}(\boldsymbol{x}) \leq m_0 \\ \widehat{f}(\boldsymbol{x}) & \widehat{m}(\boldsymbol{x}) > m_0 \end{cases}.$$

Based on the results in Section 4, we build the system variability map using the MARS, and the performance surface is estimated using MARS as well. For illustration, the performance requirement threshold m_0 is specified as the median of the throughput from the 805 points in the training set for each variability map. One can, however, specific any values for the threshold.

The cost function $c(\boldsymbol{x})$ is a 4-dimension surface. To visualize the cost function, we show the 2-dimension cross sections of the 4-dimension surface. That is one fixes two of the four dimensions (e.g., File Size and Record Size), and draws the contour of $c(\boldsymbol{x})$ as a function of thread and frequency. Figure 8 shows the contour plots of the cross sections of the the cost function under the configuration that File Size is 1024, Record Size is 204, Frequency is 2.4, Number of Threads is 15, Operation Mode is randomread, IO Scheduler is NOOP and VM IO Scheduler is CFQ.

5.2 System Configuration Optimization

We use global optimization algorithm in R to find the best configuration $\boldsymbol{x}$ that minimizes the cost function $c(\boldsymbol{x})$. The plots in Figure 8 show that it is possible to have more than one local minima points. The optimization problem is a challenging problem. Commonly used optimization solvers such as gradient-based methods, simulated annealing (Kirkpatrick, Gelatt, and Vecchi (1983)) and other stochastic optimization can easily get trapped in local minima points

In this paper, we use the Differential Evolution algorithm in Storn and Price (1997). Differential Evolution randomly select some starting points to fill the input space evenly (we call these points agents). Then for each points, we do a simplified annealing simulation on only one dimension of input space to update each agent. Updating each agent is much cheaper than simulated annealing so these can be done quickly in parallel. After several iterations, the minimal output agent is selected as optimal solution. Due to Table 2 and Table 3 in Storn and Price (1997), Differential Evolution is very robust to objective function with many local extrema but it needs more evaluations of the objective function. We can tell that Differential Evolution works better when the objective function has complex surface,

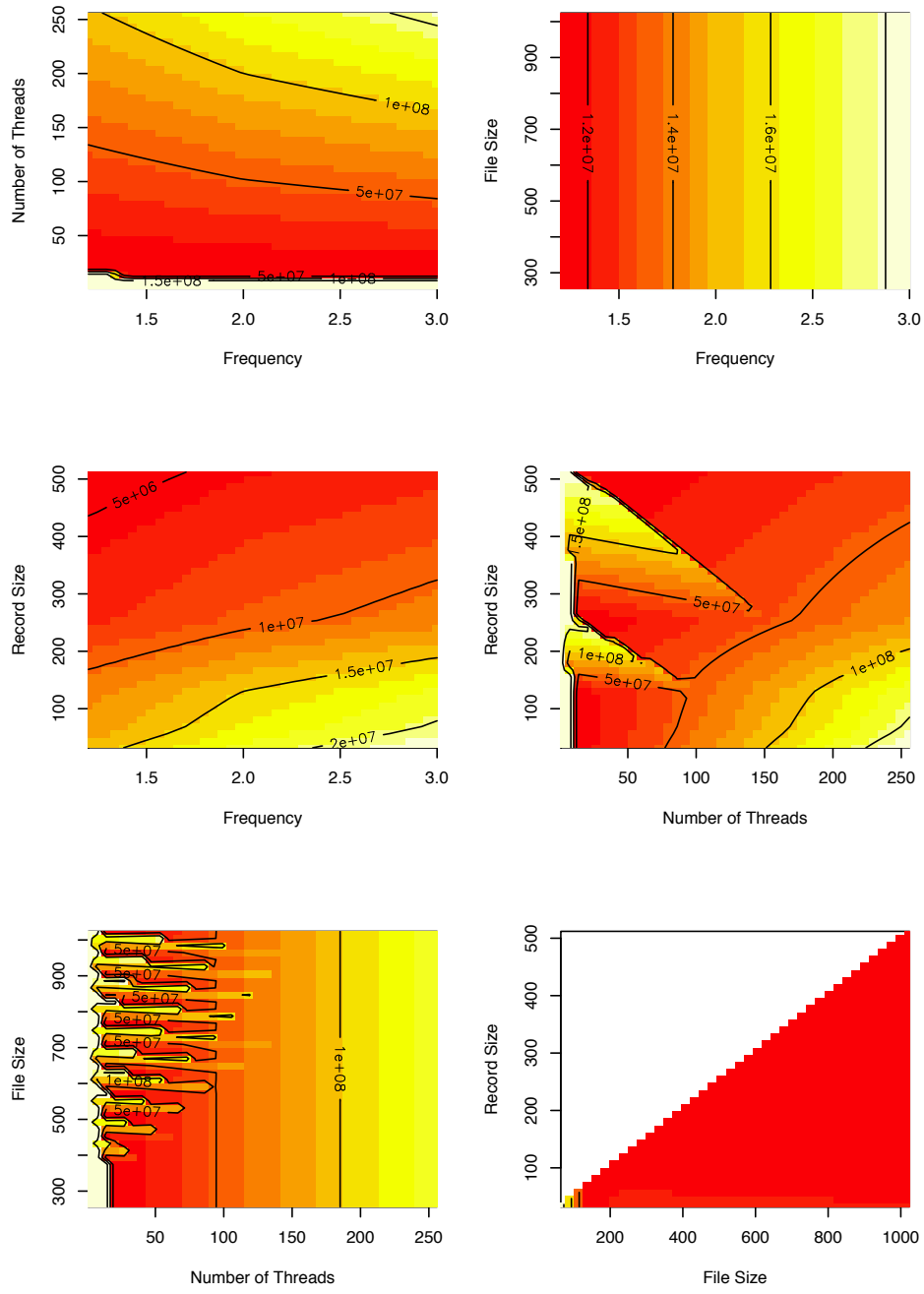


Figure 8: Contour plots of the cost function under the configuration that File Size is 1024, Record Size is 128, Frequency is 2.4, Number of Threads is 32, Operation Mode is Fread, IO Scheduler is DEAD and VM IO Scheduler is CFQ.

Table 5: Examples of system configurations after optimization to minimize performance variability subject to minimum performance requirements.

Operation Mode	IO Sch.	VM IO Sch.	File Size	Record Size	Threads	Freq	Score
Fread	DEAD	CFQ	64	5.33	1	1.2	168475.2
Fwrite	NOOP	CFQ	1024	512	1	1.2	44666.6
Initialwrite	DEAD	CFQ	768	128	1	1.2	37867.7
Mixedworkload	CFQ	DEAD	1024	512	1	1.2	287893.7
Pread	CFQ	DEAD	1024	512	1	1.2	215879.1
Pwrite	DEAD	DEAD	64	10.67	1	1.2	35301.2
Randomwrite	NOOP	NOOP	1024	512	1	1.2	280299.2
Re-read	NOOP	CFQ	64	5.82	1	1.2	82389.0
Read	CFQ	DEAD	1024	512	1	1.2	316501.8
Pwrite	NOOP	CFQ	1024	512	1	1.2	266366.2
Reverseread	DEAD	DEAD	64	32	1	1.2	211962.9
Rewrite	CFQ	NOOP	1024	512	1	1.2	97225.6
Strideread	DEAD	NOOP	1024	512	1	1.2	240535.1

relative low-dimension input space but it is cheap to evaluate. From Figure (8), the bottom left subplot shows lots of potential local extrema so Differential Evolution is promising in this scenario.

We set 300 starting points, which turns out to work well. Table 5 shows examples of system configurations after optimization to minimize performance variability subject to minimum performance requirements. We aim to find the best system configuration settings (IO scheduler, VM IO scheduler, File Size, Record Size, number of Threads, Frequency) under each mode. For each mode, there are 9 IO and VM IO scheduler combinations, we optimize the 9 settings separately and select the IO, VM IO scheduler with least cost function value. The results in Table 5 can provide useful information for designing new systems.

6 Concluding Remarks and Areas for Future Research

In this paper, we present a case study of HPC variability based on statistical and approximation techniques. We fit the variability data with different surrogate models. We show that the MARS method has good performance in interpolation and the LSP method has good performance in extrapolation, under several error measures. In addition, we also develop an optimization procedure to manage system variability to select the best system configuration under different modes.

The following are possible areas for future research. To further improve prediction accuracy, one may need more design points. We currently use a grid based method for picking design points. In the future, we can consider sequential design techniques to pick experiment points~~, which can be an interesting topic for future research~~. In this paper, we focus on the prediction of the standard deviation of the throughput, one more general goal is to predict the system throughput distribution. Techniques in functional data analysis can be used in this direction.

The developed data analytical frame work for variability can be applied to other areas other than HPC in computer science. The two major areas are cloud computing and system security. In cloud computing, large scale clouds operated by Amazon and others seek to provide a guaranteed level of service to a client. The variability management framework presented in this paper provides new opportunities for tradeoffs between system stability and performance. In system security, malware attacks can possibly change the system performance that can lead to a change in performance variability. ~~The~~ variability management can potentially increase the effectiveness of these types of malware detection techniques.

References

Bandler, Q., S. Cheng, A. Dakroury, M. Mohamed, K. Bakr, J. Madsen, and Sondergaard (2004). Space mapping: the state of the art.

Berry, M. W. and K. S. Minser (1999). Algorithm 798: High-dimensional interpolation using the modified shepard method. *ACM Transactions on Mathematical Software 25*, 353–366.

Campolucci, P., F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini (1996, May). Neural networks with adaptive spline activation function. In *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*, Volume 3, pp. 1442–1445 vol.3.

Chen, V. C., K.-L. Tsui, R. R. Barton, and M. Meckesheimer (2006). A review on design, modeling and applications of computer experiments. *IIE Transactions 38*, 273–291.

Chipman, H. A., E. I. George, and R. E. McCulloch (1998). Bayesian cart model search. *Journal of the American Statistical Association 93*, 935–948.

Chipman, H. A., E. I. George, and R. E. McCulloch (2002). Bayesian treed models. *Machine Learning 48*, 299–320.

Chipman, H. A., E. I. George, and R. E. McCulloch (2010). BART: Bayesian additive regression trees. *The Annals of Applied Statistics 4*, 266–298.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics 19*, 1–67.

Gordon, W. J. and J. A. Wixom (1978). Shepard's method of "metric interpolation" to bivariate and multivariate interpolation. *Mathematics of Computation 32*, 253–264.

Gramacy, R. B. and H. K. H. Lee (2008). Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association 103*, 1119–1130.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Kim, Y.-J. and C. Gu (2004). Smoothing spline gaussian regression: more scalable computation via efficient approximation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 66*, 337–356.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science 220*, 671–680.

Renka, R. J. (1988). Algorithm 660: Qshep2d: Quadratic shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software 14*, 149–150.

Santner, T., B. Williams, and W. Notz (2010). *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer New York.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, New York, NY, USA, pp. 517–524. ACM.

Storn, R. and K. Price (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 341–359.

Taddy, M. A., R. B. Gramacy, and N. G. Polson (2011). Dynamic trees for learning and design. *Journal of the American Statistical Association 106*, 109–123.

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 37*, 34:1–34:20.