Prediction of High-Performance Computing Input/Output Variability and Its Application to Optimization for System Configurations

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

Abstract

Performance variability is an important measure for a reliable high performance computing (HPC) system. Performance variability is affected by complicated interactions between numerous factors, such as CPU frequency, the number of input/output (IO) threads, and the IO scheduler. In this paper, we focus on HPC IO variability. The prediction of HPC variability is a challenging problem in the engineering of HPC systems and there is little statistical work on this problem to date. Although there are many methods available in the computer experiment literature, the applicability of existing methods to HPC performance variability needs investigation, especially, when the objective is to predict performance variability both in interpolation and extrapolation settings. A data analytic framework is developed to model data collected from large-scale experiments. Various promising methods are used to build predictive models for the variability of HPC systems. We evaluate the performance of the methods by measuring prediction accuracy at previously unseen system configurations. We also discuss a methodology for optimizing system configurations that uses the estimated variability map. The findings from method comparisons and developed tool sets in this paper yield new insights into existing statistical methods and can be beneficial for the practice of HPC variability management. This paper has supplementary materials online.

Key Words: Approximation Methods; Computer Experiments; Design Analysis; Gaussian Process; Reliability; System Design.

1 Introduction

1.1 The Problem

High performance computing (HPC) commonly refers to the aggregation of computing power to obtain much higher performance than a typical desktop computer or workstation. HPC is widely used to solve large-scale problems in various areas such as science, engineering, and business. While improving the performance of HPC systems attracts lots of research, managing the performance variability of HPC systems is also an important dimension of HPC system management that can not be ignored. A common manifestation of performance variability is the variation from run to run in the execution time for a particular task.

In a more specific example, the middle panel of Figure 1 shows the performance variability of input/output (IO) throughput (data transfer speed, in units of 10^7 KB/s) as a function of CPU frequency with other system parameters fixed. Following Cameron et al. (2019), we use the standard deviation of IO throughput from multiple runs as a performance variability measure (PVM). While deferring the details of Figure 1 to later sections, Figure 1 shows that the performance variability increases as the CPU frequency increases. Although one can generally obtain higher throughput when increasing the CPU frequency as shown in the top panel of Figure 1, the variability of the throughput from different runs also increases when the CPU frequency increases. For high CPU frequencies, the variability is high and the standard deviation can exceed 50% of the mean performance as shown by the bottom panel of Figure 1, which plots the ratio of the standard deviation over the mean throughput. Thus, high performance variability can be a hindrance for system development because it makes the evaluation of a system challenging when the performance is inconsistent across individual runs. There are also other factors such as imbalanced tasks caused by variability that can affect overall system performance and efficiency.

In the 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 the IO 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 design/configurations, and how can we manage performance variability?

To answer these questions, computer scientists have completed large scale experiments to collect the necessary data. In this paper, we present the statistical modeling of HPC IO variability. The prediction and management of variability is a challenging problem in the
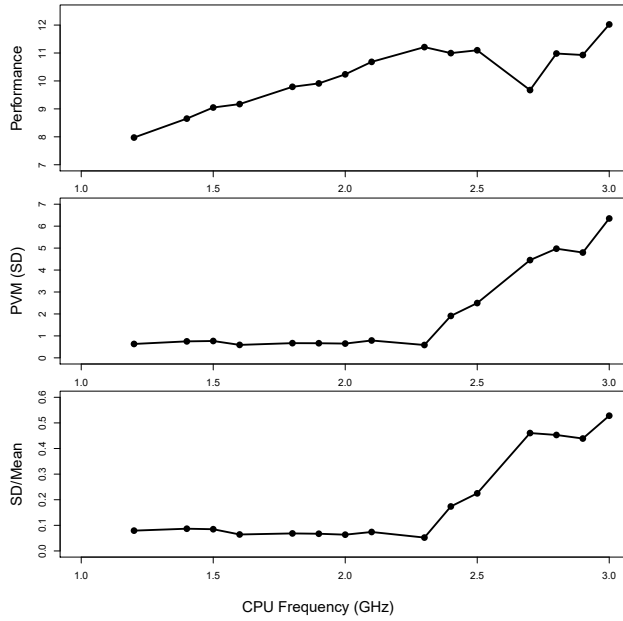
Figure 1: Example of performance variability of IO throughput (in units of KB/s$\times 10^7$) as a function of CPU frequency, while keeping other variables fixed. The top panel shows the mean throughput, which is the performance measure. The middle panel shows the standard deviation (SD), which is used as the performance variability measure (PVM). The bottom panel shows the ratio of the SD over the mean, which is referred to as the coefficient of determination in statistics.

engineering of HPC systems and there is little statistical work on this problem. Specifically, the objective is to conduct statistical modeling and make predictions about performance variability. In particular, we use various statistical methods to describe system variability with system configuration variables as inputs, which we refer to as a variability map construction. With the variability map, we make predictions for new design points (i.e., new system configurations). We examine two types of predictions: 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. This comparison provides insights into the effectiveness of different methods for predicting HPC performance variability. While it is impossible to eliminate variability, the statistical models and predictions can help manage the variability. Being able to predict variability allows one to identify system configurations that can lead to large variability. Thus, one can avoid these system configurations and design a system in which performance and variability are both optimized.

1.2 Related Literature and the Contribution of This Work

In the area of statistics, computer experiments are constructed to emulate a physical system. Because these are meant to replicate some aspects of a system in detail, they often do not yield an analytic solution. Due to the complexity and expense of evaluating system behavior, a surrogate model is usually used to describe the system behavior under some configuration. A surrogate model is an engineering method used when an outcome of interest can not be easily or directly measured, so a model of the outcome is used instead. Popular surrogate models (Chen et al. 2006) include response surface models, Kriging, gradient-enhanced Kriging, support vector machines, space mapping, and artificial neural networks (Bandler et al. 2004). In our work, we mainly focus on the following surrogate models: response surface models, Gaussian process based methods, inverse distance weighting methods, and non-parametric regression models. A good reference book on response surface models is Myers, Montgomery, and Anderson-Cook (2016).

The Gaussian process model is a popular tool in computer experiments (e.g., Santner, Williams, and Notz 2010). However, a Gaussian process only forms a stationary surface. When the underlying surface has steep behavior, the Gaussian process may not be able to capture those steep changes. Several methods have been proposed to extend the Gaussian process to form a non-stationary surface. Chipman, George, and McCulloch (2002) proposed a Bayesian treed Gaussian process that uses a reversible-jump Monte Carlo Markov chain (MCMC) to build the tree structure. Gramacy and Lee (2008) built a treed Gaussian process and Taddy, Gramacy, and Polson (2011) developed dynamic trees. Tree-based non-stationary Gaussian processes in Gramacy and Lee (2008) and Taddy, Gramacy, and Polson (2011) used similar priors to build tree structures, while the dynamic tree's prior extended the method in Chipman, George, and McCulloch (1998). Chipman, George, and McCulloch (2010) proposed an additive Gaussian process. In addition, Qian, Wu, and Wu (2008) proposed a framework for building Gaussian process models that can handle qualitative and quantitative factors. Zhang and Notz (2014) provided a comprehensive review of computer experiments with qualitative and quantitative variables. Deng, Hung, and Lin (2015) introduced the marginally coupled designs for computer experiments that can consider both qualitative and quantitative variables.

Spline bases are widely used when the relationship between input and response is highly nonlinear. In particular, the multivariate adaptive regression splines (MARS) in Friedman (1991) is a technique that uses an iterative approach to select spline bases for the approximation of an unknown surface. The criterion used in selecting the bases is related to the entire data. MARS only assumes a polynomial relationship between the response and the input variables, and can capture very complex relationships adaptively. A generalized cross validation procedure is used to determine the best model. More details on the generalized cross

validation can be found in Hastie, Tibshirani, and Friedman (2009). Adaptive splines are also adopted in other approximations to improve accuracy, for example, in neural networks (e.g., Campolucci et al. 1996), and Gaussian process modeling (e.g., Kim and Gu 2004). Ben-Ari and Steinberg (2007) compared Kriging with MARS and projection pursuit regression in the modeling of data from computer experiments.

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 earliest inverse distance weighting method is Shepard's method (Shepard 1968). Many extensions have been made based on Shepard's work (e.g., Gordon and Wixom 1978, Renka 1988, and Berry and Minser 1999).

Although there are many methods available in the computer experiment literature, the applicability of existing methods to HPC performance variability is not clear. We investigate the suitability of several methods in the literature and provide new insights into the practical application of those methods in HPC variability management. The findings of method comparisons and developed tool sets in this paper can be beneficial for the practice of HPC variability management and system building.

1.3   Overview

The rest of the paper is organized as follows. Section 2 introduces the experiment setup and describes the data collection process. Section 3 introduces 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, we focus on one specific metric, which is the input/output (IO) throughput to persistent memory (i.e., hard disk drives and solid state drives). The modeling and analysis approach used in this paper, however, can be extended to other metrics. Here, the IO throughput is defined as the data transfer speed in terms of kilobytes per second (KB/s). Even under the same system (i.e., hardware and software) configuration, different runs of the same task will end up with different IO throughput. That is, performance variability exists in IO throughput. Specifically, the performance variability measure (PVM) used in this paper is the standard deviation of the IO throughput from

multiple runs of the same task. Here, runs refer to the replicates of IOzone experiments under a given system configuration. For each run, we run the IOzone benchmark under a fixed configuration to obtain an IO throughput measurement. This process is repeated to obtain multiple measurements. We use the standard deviation of those throughput measurements as the PVM. The objective is to study how system variables affect the IO throughput, through statistical modeling of the relationship between system variables (including hardware and software configurations) and the performance variability.

2.2 Experiment Setup

The IO throughput to persistent memory is affected by three categories of variables: hardware configurations, operating system configurations, and application configurations. To study how the system configurations affect IO performance variability, complicated large-scale experiments were conducted by computer scientists. In this section, we introduce the setup of these experiments. The response variable is the throughput variability tested by running the IOzone benchmark on a server under the same system configuration $\boldsymbol{x}$, with a relatively large number of repeated runs. In particular, 40 runs were used under budget constraints. The server is configured with a dedicated 2TB HDD on a 2 socket/4 core Intel Xeon E5-2623 v3 (Haswell) platform. The memory is 32 GB DDR4. The number of hyperthreads is 2 per core. The Linux operating system used is Ubuntu 14.04. More details can be found in Cameron et al. (2019).

For system configurations, seven important variables were selected by computer scientists, which consist of both categorical and continuous variables. Table 1 shows the summary information for the seven variables that were used in the IOzone throughput experiments. On the hardware configurations, changing CPU clock frequency is an important method for adjusting system performance and energy efficiency. On the experiment server, there are 15 levels of frequencies ranging from 1.2 to 3.0 GHz. On the operating system configuration, one can change the IO scheduling policy of the host system as well as the hypervisor by which virtual systems are controlled. Both the host system and virtual machine (VM) have 3 levels of IO schedulers, which are CFQ, DEAD, and NOOP.

The application configuration has four variables, which are the IO Operation Mode, the Number of Threads, File Size (KB), and Record Size (KB). The IO Operation Mode has 13 levels, which cover almost all the common IO operations on HPC systems. For example, Fread and Fwrite represent different types of reading and writing tasks, respectively. The Number of Threads has 9 levels, ranging from 1 to 256 by powers of 2. The values of File Size and Record Size are chosen in pairs as there are some constraints on the selection of values. In particular, the File Size has to be larger than the Record Size, and has to be a multiple of the

Table 1: Summary information for variables that are used in the IOzone throughput experiments to collect the training set and extrapolation test set. Note in the File Size by Record Size row, the first number is File size and the second number is Record Size. The number of all the combinations in this table is $13 \times 3 \times 3 \times 15 \times 9 \times 6 = 94770$. The settings, at which the Frequency and the Number of Threads 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 set aside for extrapolation test set, which excludes $13 \times 3 \times 13 \times 5 \times 1 = 585$ settings with large values on the Frequency and the Number of Threads. Thus, the number of points in the training set is $94770 - 585 = 94185$.

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	IO Scheduler	3	CFQ, DEAD, NOOP
	VM IO Scheduler	3	CFQ, DEAD, NOOP
Application	IO 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)

Record Size. For example, the sixth row of Table 1 shows the valid combinations of File Size and Record Size used in the training set.

2.3   Prediction Problems and Data Collection

Two kinds of predictions arise in the setting of HPC variability management, which are interpolation and extrapolation. To test the predictability of different methods, we collect three sets of data: the training set ($\mathcal{S}_t$), the interpolation test set ($\mathcal{S}_i$), and the extrapolation test set ($\mathcal{S}_e$). Let $\boldsymbol{x}$ be the vector representing the system variables (i.e., system parameters). We build a prediction model based on the training set $\mathcal{S}_t$, and we then use the built prediction model to make a prediction for the PVM under a new configuration $\boldsymbol{x}_w$. A prediction problem is called an *interpolation* if $\boldsymbol{x}_w$ lies within the convex hull of $\mathcal{S}_t$, and a prediction problem is called an *extrapolation* if $\boldsymbol{x}_w$ falls outside the convex hull of $\mathcal{S}_t$.

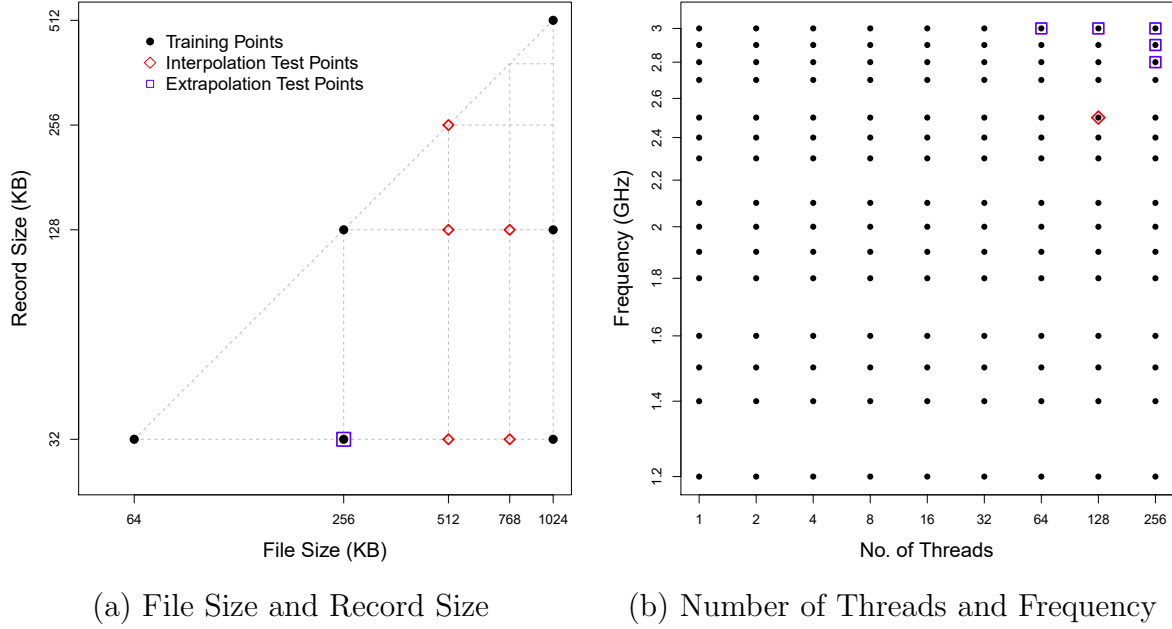(a) File Size and Record Size   (b) Number of Threads and Frequency

Figure 2: Illustration of design points for File Size, Record Size, Number of Threads, and Frequency. The points marked by black dots are for the training set. The points marked by black dots with a square are for the extrapolation test set, and the points marked by diamonds are for the interpolation test set.

Regarding $\mathcal{S}_t$, $\mathcal{S}_i$, and $\mathcal{S}_e$, each dataset contains eight columns with one column for the PVM, which is the response/output variable, and seven columns for the explanatory/input variables. The IO Scheduler, VM IO Scheduler, and IO 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, $\mathcal{S}_t$, is used to build a model for predicting variability. As explained in the caption of Table 1, the training set consists of

$$13 \times 3 \times 3 \times 15 \times 9 \times 6 - 13 \times 3 \times 3 \times 5 \times 1 = 94185 \tag{1}$$

points. The training set consists of all combinations of the variables (i.e., $13 \times 3 \times 3 \times 15 \times 9 \times 6 = 94770$) with $13 \times 3 \times 3 \times 5 \times 1 = 585$ points excluded for the extrapolation test set. Figure 2 illustrates the design points for File Size, Record Size, Number of Threads, and Frequency, as shown by black dots.

The subtracted points in (1) are those at which the Frequency and the Number of Threads 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), which are used for extrapolation testing. Figure 2 displays the points that are used for extrapolation testing purposes, which are marked by squares. Note that the points are on the right upper corner of the combination of Frequency and the Number of

Table 2: Summary of the experimental setup for the collections of the two test sets.

Variable	Interpolation Test Set $\mathcal{S}_i$	Extrapolation Test Set $\mathcal{S}_e$
IO Scheduler	All 3 levels	All 3 levels
VM IO Scheduler	All 3 levels	All 3 levels
IO 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

Threads, representing large values of the two variables. Thus, the main purpose is to test the extrapolation of the Frequency and the Number of Threads to large values.

There are an additional $3 \times 3 \times 13 \times 5 \times 1 \times 1 = 585$ points used for interpolation testing purposes. Those points are marked by 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 lie in the interior of or on the convex hull of File Size and Record Size points. Thus, the main purpose is to test the interpolation of the File Size and Record Size.

Table 2 shows the summary of the experimental setup for the collections of the two test sets, $\mathcal{S}_i$ and $\mathcal{S}_e$. In our data collection, we have more points on the Frequency and Number of Threads combinations than that of the File Size and Record Size combination. Because extrapolation is usually more challenging than interpolation, we chose to test the ability of extrapolation on the Frequency and Number of Threads combinations. For interpolation, we chose to test the ability on the File Size and Record Size combination. Although there were only six combinations for the File Size and Record Size combination, interpolation can still be made based on this information provided by the data.

3 Variability Map Construction

3.1 Data Visualization and Variability Map

In this section, we first do some data visualization to explore the data. Figure 3 shows one example of a set of system variables before and after log transformation using the training

set. Note that the scale of the IO throughput is in the magnitude of 10^7. Because a variable and its standard deviation have the same scale, the PVM (i.e., the standard deviation of IO throughputs) is also 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. These categorical configurations were chosen by computer scientists as the most likely selection to be used in practice. With this configuration, Figure 3 shows that the PVM of the 805 points (i.e., configurations) is right skewed. After the log transformation, the PVM of the 805 points shows a bimodal behavior. Figure 3(b) also allows us to see that for certain configurations, the variability tends to be small, while for other configurations, the variability tends to be large.

To further visualize how the PVM changes across different system configurations, Figure 4 shows the perspective plots of the 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 cannot be described by simple functional relationships. While the figures can only show the pattern for two variables, the actual relationship is more complicated due to many other variables that affect the PVM. Note that Figure 4 only shows the pattern for one slice of the data. The shapes of the perspective plots could differ for different configurations.

We introduce the concept of the variability map to describe the functional relationship between the system configuration and the PVM. The variability map is defined as a function $f(\boldsymbol{x})$ that gives the PVM at $\boldsymbol{x}$. With the variability map $f(\boldsymbol{x})$ estimated from the training dataset, one can use it to characterize and predict variability for a given configuration $\boldsymbol{x}$.

We focus on the modeling of continuous variables in this paper. For categorical variables, we do a separate estimation for the variability map for each unique combination of the variables for all candidate prediction methods, except for the categorial Gaussian process (CGP) model. In particular, the IO operation Mode, IO Scheduler, and VM IO Scheduler have 117 unique combinations for their levels. Thus, 117 separate variability maps will be constructed. For certain prediction techniques, such as the linear Shepard (LSP) method, they are not designed for categorical variables. Thus, we need to handle each level of categorical variables separately. This treatment, however, does not limit their practical use because one has to specify the level of the categorical variable in the prediction. That is, we can still use the separate variability maps to generate the prediction, given the level of the categorical variable. The CGP model is able to handle categorical variables to some extent.

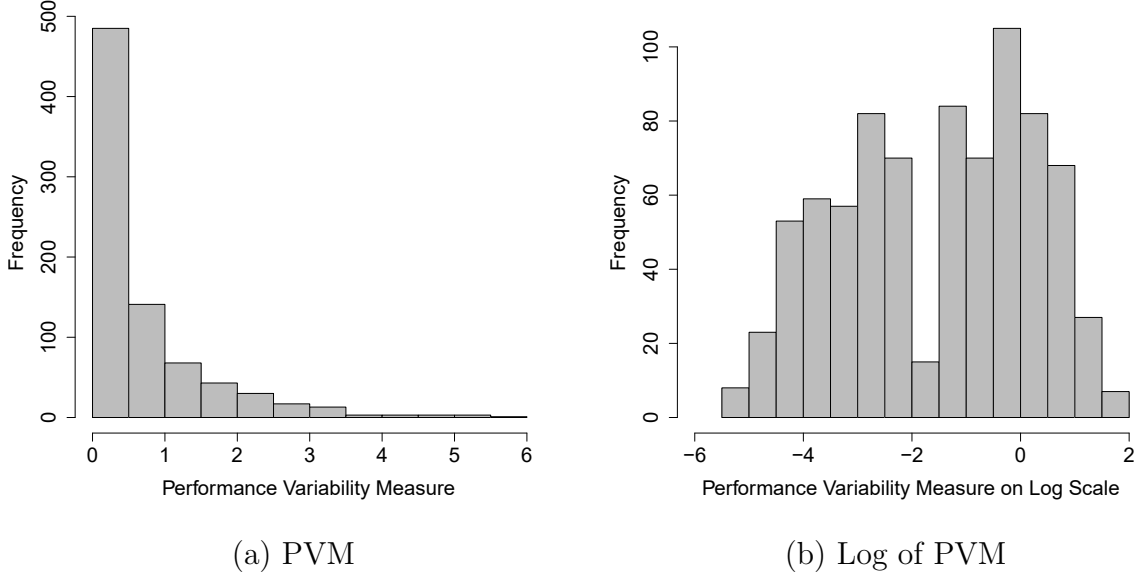Without loss of generality, we only describe the construction of one particular variability

(a) PVM

(b) Log of PVM

Figure 3: One example of 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, with all combinations of File Size and Record size.

map. We define $\boldsymbol{x}_i$ as follows,

$$
\begin{aligned}
\boldsymbol{x}_i =& (\text{Log of File Size, Log of Record Size, Log of No. of Threads, CPU Frequency})' \\
=& (x_{i1}, x_{i2}, x_{i3}, x_{i4})'.
\end{aligned}
\tag{2}
$$

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 $\boldsymbol{x}_i = (x_{i1}, x_{i2}, x_{i3}, x_{i4})'$ for a particular combination of the IO operation Mode, IO Scheduler, and VM IO Scheduler. Here $n = 805$ is the number of data points for each variability map.

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 a 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 Gamma generalized linear model (GLM) with a polynomial structure, the linear Shepard (LSP) method that is based on inverse distance weight, the multivariate adaptive regression splines (MARS), Gaussian process based methods, tree based models such as the Bayesian additive regression trees (BART), and categorical Gaussian process (CGP). These methods are flexible and can automatically capture globally complex

11

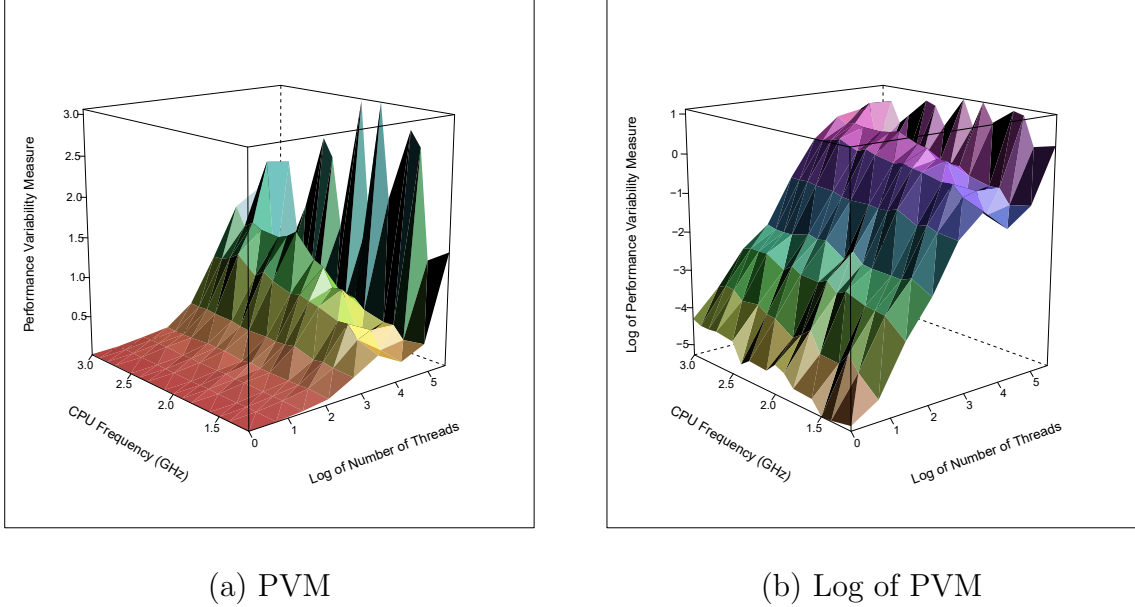(a) PVM   (b) Log of PVM

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.

relationships, potentially uncovering interactions between variables that have dominant effects on IO variability.

Linear Model. For the linear model (LM), 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 is done by using the least-squares method.

Gamma GLM. In some applications, the LM can be insufficient for modeling standard deviation as mentioned in Chapter 11 of Myers, Montgomery, and Anderson-Cook (2016). Hence, we further consider a Gamma GLM (GAMGLM) with a polynomial mean structure. In the GAMGLM setting, we assume y_i follows a Gamma distribution with mean μ_i. Through the log link function, the mean μ_i is linked to the linear predictor η_i as $\mu_i = \log(\eta_i)$. We use a polynomial structure for η_i as follows for the HPC dataset,

$$\eta_i = \beta_0 + \sum_{j=1}^{4} \beta_j x_{ij} + \sum_{j=3}^{4} \gamma_j x_{ij}^2 + \sum_{j=3}^{4}\sum_{k=1}^{j-1} \beta_{jk} x_{ij} x_{ik} + \sum_{j=3}^{4} \delta_j x_{ij}^3.$$

Here, the covariates are defined in (2) and $\{\beta_0, \beta_j, \gamma_j, \beta_{jk}, \delta_j\}$ are regression coefficients. Note that we only consider the linear terms for x_{i1} (Log of File Size) and x_{i2} (Log of Record Size)

12

because there are only six unique combinations for the two variables. The variability map $f(\boldsymbol{x})$ is estimated as μ, which is the mean of the Gamma distribution at point $\boldsymbol{x}$. The parameters can be estimated by using the maximum likelihood method.

The Linear Shepard Method. The LSP method is an approximation algorithm based modification of the original Shepard algorithm (e.g., Thacker et al. 2010). The LSP approximation is constructed using linear combinations of locally linear fits. To introduce LSP, we need to define some notation first. Let p be the length of $\boldsymbol{x}$, n be the number of data points and $m = \min\{n, 3p/2\}$ as set in Thacker et al. (2010). 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 $v_i = 1.1r_i$, also as designed in Thacker et al. (2010).

In 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 $w_i(\boldsymbol{x}) = \{[1/d_i(\boldsymbol{x}) - 1/u_i]_+\}^2$. That is, the weight is set as $w_i(\boldsymbol{x})$ for those $\boldsymbol{x}$'s are within the ball for $\boldsymbol{x}_i$ with radius u_i and is set to 0 for those $\boldsymbol{x}$'s are outside the ball for $\boldsymbol{x}_i$. Here $[x]_+ = x$ if $x \geq 0$ and $[x]_+ = 0$ if $x < 0$.

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 used in the locally weighted least-squares fit is defined as $\omega_{ij} = \{[1/d_i(\boldsymbol{x}_j) - 1/v_j]_+\}^2$. That is, the weight is set as ω_{ij} if $\boldsymbol{x}_i$ is within the ball for $\boldsymbol{x}_j$ with radius v_j and is set to zero if $\boldsymbol{x}_i$ is outside the ball for $\boldsymbol{x}_j$. Then, the coefficient $\boldsymbol{\beta}_i$ can be obtained by solving the weighted least-squares problem.

Multivariate Adaptive Regression Splines (MARS). Let $\boldsymbol{x}_i = (x_{i1}, \cdots, x_{ij}, \cdots, x_{ip})'$. For any $\boldsymbol{x} = (x_1, \cdots, x_j, \cdots, x_p)'$, 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 squared errors 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 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 CART model, and it partitions the parameter space into disjoint regions using a tree structure. In each region, a local Gaussian process is fit 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 in this paper, which are TGP with a linear trend (denoted as TGPlm), TGP with a 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 in previously divided regions. Other tree construction methods such as swapping, pruning, rotation can take place after the splitting.

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

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

Here $\mu(\boldsymbol{X}_j, \boldsymbol{\theta}_j)$ is the mean structure which is a function of $\boldsymbol{X}_j$ with unknown parameters $\boldsymbol{\theta}_j$, and $\boldsymbol{\Sigma}_j$ is the variance-covariance matrix. With the 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 a linear trend Gaussian process in local partition while TGPcart uses a constant mean. Dynamic trees can explore posterior tree space by stochastically proposing incremental modifications to the tree structure (such as splitting, pruning, and swapping) in each terminal leaf.

Bayesian Additive Regression Trees. The 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

$g(\boldsymbol{x}; T_j; \boldsymbol{\theta}_j)$ 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.

Gaussian process with qualitative and quantitative factors. The categorical Gaussian process (CGP) is used to handle qualitative and quantitative factors input variables, which is a modification to the classic Gaussian process. We use the CGP model in Zhou, Qian, and Zhou (2011). Starting from the original Gaussian process model, $\boldsymbol{y} \sim \mathrm{N}[\mu, \boldsymbol{\Sigma}(\boldsymbol{X}, \boldsymbol{\theta})]$, where μ is the global mean, $\boldsymbol{X}$ is the design matrix for continuous variables, and $\boldsymbol{\Sigma}(\boldsymbol{X}, \boldsymbol{\theta})$ is the covariance matrix with parameters $\boldsymbol{\theta}$. We consider two categorical variables, the IO scheduler and the VMIO scheduler, as the categorical input variables, in addition to those continuous variables. With the consideration of the IO scheduler and the VMIO scheduler, the length of $\boldsymbol{y}$ is $805 \times 3 \times 3 \times 3 = 7245$. Due to computational constraints, we still model the levels of the IO Operation Mode separately. Thus, we have 13 separate variability maps under CGP model.

Let $\boldsymbol{z}^{io}$ and $\boldsymbol{z}^{vmio}$ be the corresponding coded vectors for IO scheduler and VMIO scheduler (i.e., CFQ, 1; DEAD, 2; and NOOP, 3). Let z_i^{io} and z_i^{vmio} be the corresponding ith elements in $\boldsymbol{z}^{io}$ and $\boldsymbol{z}^{vmio}$, which takes values in $\{1, 2, 3\}$. For CGP model, the covariance between two points $(\boldsymbol{x}_i', z_i^{io}, z_i^{vmio})'$ and $(\boldsymbol{x}_j', z_j^{io}, z_j^{vmio})'$ is further modeled as,

$$\boldsymbol{\Sigma}(\boldsymbol{X}, \boldsymbol{\theta})_{ij} \times \tau_{z_i^{io} z_j^{io}}^{io} \times \tau_{z_i^{vmio} z_j^{vmio}}^{vmio},$$

where τ_{kl}^{vmio} and $\tau_{kl}^{io}, k = 1, 2, 3, l = 1, 2, 3$ are correlation parameters to be estimated. All the parameters are estimated by using the maximum likelihood method.

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 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 functions in each iteration. For the DynaTree method, the linear terminal Gaussian process is used.

For each distinct combination of these categorical variables, we generate separate variability maps, resulting in 117 variability maps, except for CGP which has 13 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 maps as a function of CPU Frequency and the Number of Threads, using the MARS algorithm. The system configuration of
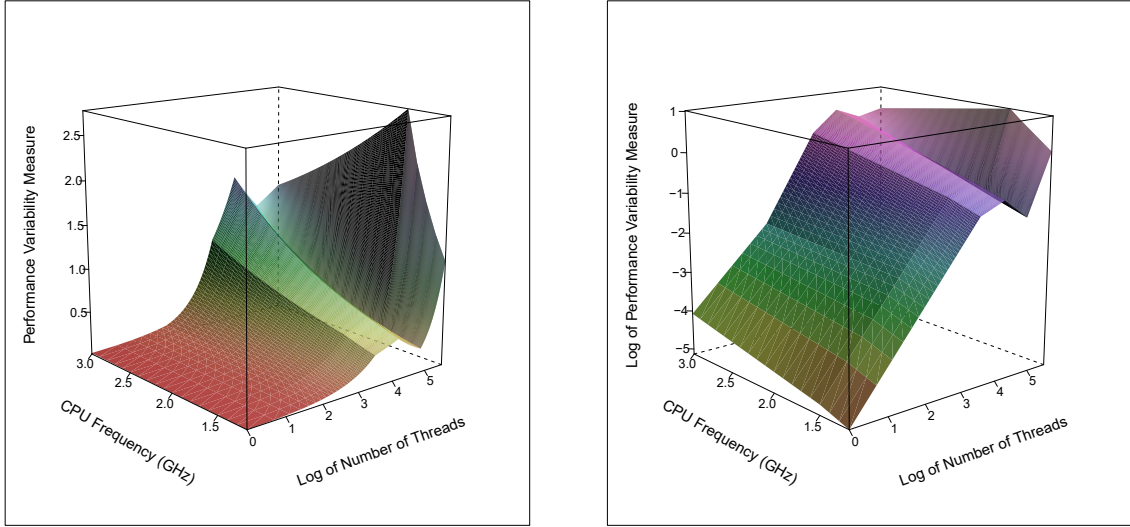
| (a) PVM | (b) Log of PVM |

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.

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   Criteria for Comparisons

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. To evaluate the performance of the candidate methods, we need to discuss our comparison 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 all 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,

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

Note that the ER_1 is the average of the point-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 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. Note that ER_2 is also referred to as the coefficient of variation in the statistical literature. The ER_3 is the MAE divided by the 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, variability maps are constructed by using the candidate methods. The log transformation for the thread count, file size, record size and 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 candidate methods based on the interpolation test set. The RMSE 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 in the smallest error rate. The LSP and BART generate results that are best after MARS. The three TGP based methods and CGP are in the middle based on ER_1. The LM and GAMGLM tend to have large errors based on ER_1 but their error rates are smaller based on ER_2 and ER_3. 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 candidate methods. One can see that the LSP method has the least spread, and MARS and TGPlm have the least median error rates. Overall, the LSP method tends to work well for interpolation prediction problems as it has the least error spread and a relatively low error.

4.3   Extrapolation

In this section, we make a similar comparison for predictability of different methods when a new design point falls outside the convex hull of the input space (i.e., extrapolation). Similar to the setting of interpolation tests, 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.

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

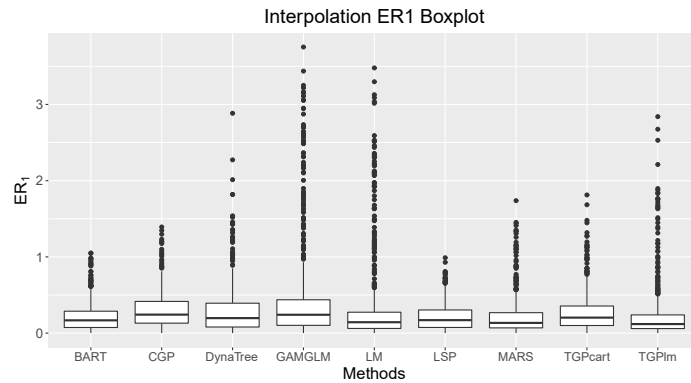| Method	ER_1	RMSE	ER_2	MAE	ER_3
LM	0.357	1.135	0.227	0.827	0.165
GAMGLM	0.481	1.407	0.281	1.116	0.223
LSP	0.216	1.269	0.254	0.891	0.178
MARS	0.223	0.913	0.183	0.695	0.139
TGPlm	0.245	1.024	0.205	0.707	0.141
TGPcart	0.268	1.519	0.304	1.072	0.214
DynaTree	0.296	2.462	0.492	1.305	0.261
BART	0.220	1.222	0.244	0.878	0.175
CGP	0.298	1.770	0.354	1.300	0.261

Figure 6: Box plot for the 585 individual relative error rates for the candidate methods based on the interpolation test set.

Table 4: The RMSE, MAE, and three error rates for the 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	0.412	7.509	0.712	5.272	0.500
GAMGLM	0.142	1.490	0.141	1.087	0.103
LSP	0.089	1.402	0.094	0.634	0.060
MARS	0.148	0.995	0.133	0.952	0.090
TGPlm	0.236	3.851	0.365	2.561	0.243
TGPcart	0.289	4.701	0.446	3.376	0.320
DynaTree	0.184	3.390	0.321	1.927	0.183
BART	0.087	0.946	0.090	0.621	0.059
CGP	0.066	0.630	0.060	0.434	0.041

We find that the models with log transformed responses tend to generate nonsensically large errors for extrapolation due to the effect of exponentiation. 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 CGP shows the best result. The BART and LSP methods are the next after CGP. The MARS and GAMGLM methods are in the middle, and the TGP methods tend to generate large errors. Figure 7 shows the box plot for the 585 individual relative error rates. The figure shows that the CGP and LSP methods have the least amount of spread in terms of individual errors. Most surrogate models in computer experiments are built for interpolation and their extrapolation ability is not well studied. Surprisingly, we find that both the CGP and LSP methods have good extrapolation ability under our current test setting. Overall, the CGP shows potential as a good extrapolation technique. However, one should note that extrapolation is always a difficult task and should be exercised with caution.

5 Management of Performance Variability

5.1 The System Optimization Problem

In this section, we address the problem of variability management. One method for balancing overall performance with performance variability is to minimize performance variability, while maintaining acceptable overall performance. 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
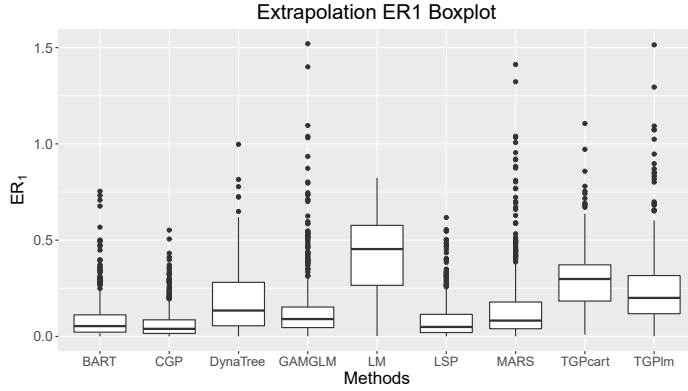
Figure 7: Box plot for the 585 individual relative error rates for the candidate methods based on the extrapolation test set.

performance surface $m(\boldsymbol{x})$ at any $\boldsymbol{x}$. The performance surface is the mean throughput at $\boldsymbol{x}$. With the training set, $m(\boldsymbol{x})$ 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 optimization problem can be formulated as the following optimization problem. That is

$$\min_{\boldsymbol{x}\in\mathcal{D}} \widehat{f}(\boldsymbol{x}), \quad \text{subject to} \quad \widehat{m}(\boldsymbol{x}) \geq m_0. \tag{3}$$

Based on the results in the comparison study, we build the system variability map using LSP, and the performance surface is estimated using LSP as well, due to its good prediction performance and computational efficiency. For illustration, the performance requirement threshold m_0 is specified as the mean of the throughput from the 805 points in the training set for each variability map. One can, however, specify any values for the threshold.

The objective function $\widehat{f}(\boldsymbol{x})$ is a 4-dimensional surface. To visualize the objective function, we show the 2-dimensional cross sections of the 4-dimensional surface. That is, one fixes two of the four dimensions (e.g., File Size and Record Size), and draws the contour of $\widehat{f}(\boldsymbol{x})$ as a function of the rest two dimensions (e.g., the Number of Threads and CPU Frequency). Figure 8 shows the contour plots of the cross sections of the objective function under the configuration: File Size of 1024, Record Size of 128, Frequency of 2.4, Number of Threads of 32, Operation Mode of Fread, IO Scheduler of DEAD and VM IO Scheduler of CFQ. The plots in Figure 8 show that the objective function is complicated.

5.2   Methods for System Configuration Optimization

The optimization problem is a challenging problem due to the complexity of the multi-dimension surface. In this paper, we use the augmented Lagrange method (e.g., Hestenes
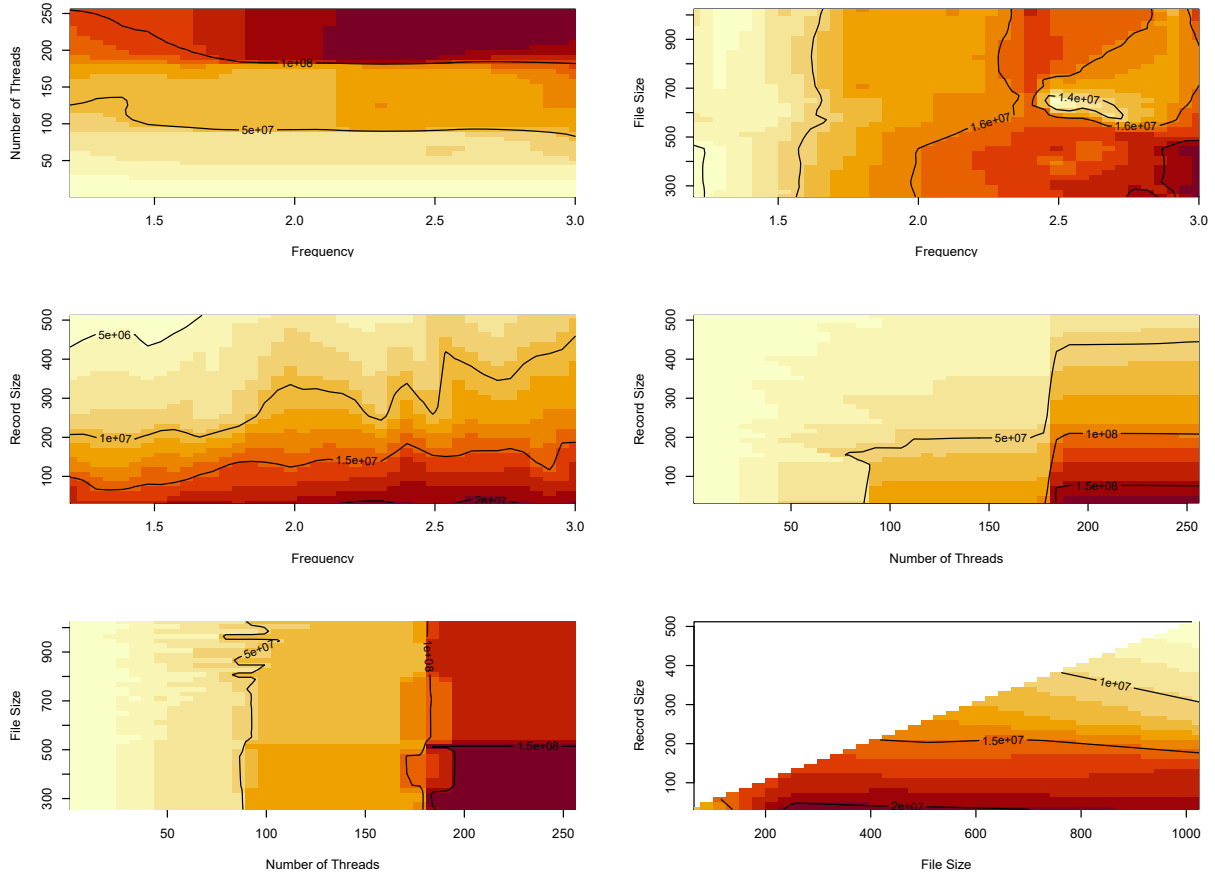
Figure 8: Contour plots of $\widehat{f}(\boldsymbol{x})$ under the configuration that File Size is 1024, Record Size is 128, Frequency is 2.4, Number of Threads is 32, Operation Mode is Fread, IO Scheduler is DEAD and VM IO Scheduler is CFQ.

1969) to solve the optimization problem in (3). By introducing a slack variable z, (3) is equivalent to

$$\min_{\boldsymbol{x} \in \mathcal{D}, \, z} \widehat{f}(\boldsymbol{x}), \quad \text{subject to} \quad m_0 - \widehat{m}(\boldsymbol{x}) + z^2 = 0.$$

Using the augmented Lagrange method, we have the following new objective function,

$$L(\boldsymbol{x}, z, u, c) = \widehat{f}(\boldsymbol{x}) + u \left[m_0 - \widehat{m}(\boldsymbol{x}) + z^2 \right] + \frac{c}{2} \left[m_0 - \widehat{m}(\boldsymbol{x}) + z^2 \right]^2, \quad (4)$$

where u is the Lagrange multiplier and c is an additional parameter introduced by the algorithm. Note that $\boldsymbol{x}$ and z are separable. We first minimize z and obtain,

$$L_z(\boldsymbol{x}, u, c) = \widehat{f}(\boldsymbol{x}) + \frac{1}{2c} \max\{0, u + c[m_0 - \widehat{m}(\boldsymbol{x})]\}^2 - u^2.$$

In theory, the problem in (3) can be solved in an iterative way. One can pick a sequence of values for c, denoted by c_k, such that $\lim_{k \to \infty} c_k = \infty$. Then $\boldsymbol{x}_k$ and u_k can be updated in the following way, $\boldsymbol{x}_k = \arg\min L_z(\boldsymbol{x}, u_k, c_k)$ and $u_{k+1} = \max\{0, u_k + c_k [m_0 - \widehat{m}(\boldsymbol{x}_k)]\}$. The sequence $\boldsymbol{x}_k$ will approach to the solution for (3). The iteration stops when the distance between two consecutive updates of $\boldsymbol{x}_k$ is small enough (e.g., $1e-6$).

In this paper, we use the VTDIRECT95 package (He, Watson, and Sosonkina 2009) which can solve the problem with one global iteration run. We set the c parameter to be a large value (e.g., $c = 1000$), and use VTDIRECT95 to solve (4) directly.

Let $\boldsymbol{x}^*$ be the solution to (3). Due to some system requirements, some adjustments are needed for $\boldsymbol{x}^*$. We need to find proper file size and record size such that the ratio of file size and record size is an integer. For a given pair of file size and record size, we find the nearest corrected file size and record size such that the corrected file size and record size are integers, and the ratio of corrected file size and record size is also an integer. We also round the number of threads to the nearest integer.

5.3 System Optimization Results

For system design optimization, we aim to find the best system configurations (i.e., IO Scheduler, VM IO Scheduler, File Size, Record Size, Number of Threads, and CPU Frequency) under each operation mode. For each operation mode, there are nine IO and VM IO scheduler combinations, we optimize the nine configurations separately and select the IO and VM IO schedulers with the least cost function value.

Table 5 shows the optimal system configurations for the 13 operation modes. We found that most write operations need a high number of threads to have optimal performance in terms of stability and speed. While for the read operations, the system has the best performance when

Table 5: Examples of system configurations for each operation mode after optimization to minimize performance variability subject to minimum performance requirements. Here "Fs" means File Size, "Rs" means Record Size, and "Freq" means the CPU Frequency.

Mode	IO	VMIO	Fs	Rs	Thread	Freq
Fread	CFQ	CFQ	752	376	16	1.73
Fwrite	DEAD	CFQ	64	32	30	1.33
Initialwrite	DEAD	NOOP	64	32	30	1.31
Mixedworkload	CFQ	NOOP	780	390	16	1.97
Pread	DEAD	CFQ	704	352	16	1.20
Pwrite	DEAD	CFQ	712	356	30	1.20
Randomread	NOOP	NOOP	1024	512	16	1.34
Randomwrite	NOOP	NOOP	308	154	16	1.73
Re-read	NOOP	DEAD	848	424	16	1.70
Read	NOOP	NOOP	752	376	16	1.20
ReverseRead	CFQ	NOOP	990	495	16	1.30
Rewrite	DEAD	NOOP	832	416	16	1.88
Strideread	DEAD	DEAD	752	376	16	1.66

the number of threads is 16, which is the number of physically available threads from the server. When the number of threads is more than the number of the physically available threads, the system typically shows more variability. The optimal file size doubles the record size, which means smaller file size and record size ratio leads to smaller variability. Lower frequencies typically lead to smaller variability, given that the requirement for the performance is met. The results in Table 5 can provide useful information for configuring new systems.

6 Concluding Remarks and Areas for Future Research

In this paper, we investigate HPC variability using various statistical techniques. We fit the variability data with different surrogate models. We show that the MARS, BART, and L-SP methods have good performance in interpolation, and the CGP and LSP methods have good performance in extrapolation, under several error measures. The users can choose the corresponding methods based on the purpose of the analysis (i.e., interpolation versus extrapolation). Overall, the LSP method has small ER_1 error rates with small error spreads for both scenarios. In addition, we also develop an optimization procedure to manage system variability and select the best system configuration under different modes. The tools developed in

this paper can be useful for the practice of HPC variability management and HPC system building.

We want to point out that our paper serves as an initial step in analyzing HPC performance variability statistically. We demonstrate the usefulness of the particular prediction methodologies in understanding the variability in IO. However, we want to emphasize that more extensive work is needed in the future for studying HPC performance variability. Here we discuss some possible areas for future research. We currently use a grid-based design for selecting design points in the data collection stage. In the future, we can consider sequential design techniques to select the design points. Space filling design techniques can also be considered for data collection (e.g., Joseph 2016).

Even though we can predict performance variability well, it is still important to understand the root causes of variability. In the future, it will be interesting to collect more data, such as the performance counter statistics which provide finer details on the IO operation, and analyze such data to reveal the root causes of performance variability. Variable selection techniques will be useful in finding important factors in performance variability.

We focus on the prediction of the standard deviation of the throughput in this paper. Xu et al. (2020) show that the distribution of the throughput is multi-modal and thus it is complicated. A more ambitious goal is to predict the system throughput distribution generally (e.g., Lux et al. 2018). Hung, Joseph, and Melkote (2015) developed a Gibbs sampling-based expectation-maximization algorithm for efficiently fitting a Kriging model to the functional responses. Techniques in functional data analysis can also be considered in modeling functional responses. In literature, single index models are used in Gaussian processes to make predictions in the context of computer experiments (e.g., Choi, Shi, and Wang 2011, and Gramacy and Lian 2012). It will be interesting to consider Gaussian processes with single index models to predict performance variability in HPC setting.

In this paper, we consider a Gaussian process with categorical and continuous inputs, which can handle two categorical variables (i.e., the IO scheduler and the VMIO scheduler). However, we found estimating a model using all categorical variables (i.e., including IO Operation Mode) to be challenging. Because the IO Operation Mode has 13 levels, in order to show all interactions, we need to estimate $13 \times 12/2 = 78$ parameters. In addition, all observations will be correlated (i.e., we need to invert a covariance matrix with dimension 94185×94185). In the future, it will be interesting to investigate methods that can handle this large scale of data.

The developed data analytical framework for variability can be applied to 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 large corporations 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 may impact system performance enough to create noticeable changes in performance variability. Variability management can potentially increase the effectiveness of these types of malware detection techniques.

Supplementary Materials

The following supplementary materials are available online.

Code and data: Computing code for data analysis, comparisons, and optimization. The HPC datasets used in the paper are also included (zip file).

Acknowledgments

References

Bandler, J. W., Q. S. Cheng, S. A. Dakroury, A. S. Mohamed, M. H. Bakr, K. Madsen, and J. Sondergaard (2004). Space mapping: the state of the art. *IEEE Transactions on Microwave Theory and Techniques 52*, 337–361.

Ben-Ari, E. N. and D. M. Steinberg (2007). Modeling data from computer experiments: An empirical comparison of Kriging with MARS and projection pursuit regression. *Quality Engineering 19*, 327–338.

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.

Cameron, K. W., A. Anwar, Y. Cheng, L. Xu, B. Li, U. Ananth, J. Bernard, C. Jearls, T. Lux, Y. Hong, L. T. Watson, and A. R. Butt (2019). MOANA: Modeling and analyzing I/O variability in parallel system experimental design. *IEEE Transactions on Parallel and Distributed Systems 30*, 1843–1856.

Campolucci, P., F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini (1996). 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.

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.

Choi, T., J. Shi, and B. Wang (2011). A Gaussian process regression approach to a single-index model. *Journal of Nonparametric Statistics 23*, 21–36.

Deng, X., Y. Hung, and C. D. Lin (2015). Design for computer experiments with qualitative and quantitative factors. *Statistica Sinica 25*, 1567–1581.

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.

Gramacy, R. B. and H. Lian (2012). Gaussian process single-index models as emulators for computer experiments. *Technometrics 54*, 30–41.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. New York: Springer.

He, J., L. T. Watson, and M. Sosonkina (2009). Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm direct. *36, article no. 17*.

Hestenes, M. R. (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications 4*, 303–320.

Hung, Y., V. R. Joseph, and S. N. Melkote (2015). Analysis of computer experiments with functional response. *Technometrics 57*, 33–54.

Joseph, V. R. (2016). Space-filling designs for computer experiments: A review. *Quality Engineering 28*, 28–35.

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.

Lux, T. C., L. T. Watson, T. H. Chang, J. Bernard, B. Li, X. Yu, L. Xu, G. Back, A. R. Butt, K. W. Cameron, Y. Hong, and D. Yao (2018). Nonparametric distribution models for predicting and managing computational performance variability. In *SoutheastCon 2018*, pp. 1–7. IEEE.

Myers, R. H., D. C. Montgomery, and C. M. Anderson-Cook (2016). *Response Surface Methodology: Process and Product Optimization Using Designed Experiments* (4th ed.). Hoboken, NJ: John Wiley & Sons.

Qian, P. Z. G., H. Wu, and C. F. J. Wu (2008). Gaussian process models for computer experiments with qualitative and quantitative factors. *Technometrics 50*, 383–396.

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*. New York: Springer.

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, pp. 517–524. ACM.

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.

Xu, L., Y. Wang, T. Lux, T. Chang, J. Bernard, B. Li, Y. Hong, K. Cameron, and L. Watson (2020). Modeling I/O performance variability in high-performance computing systems using mixture distributions. *Journal of Parallel and Distributed Computing 139*, 87–98.

Zhang, Y. and W. I. Notz (2014). Computer experiments with qualitative and quantitative variables: A review and reexamination. *Quality Engineering 27*, 2–13.

Zhou, Q., P. Z. G. Qian, and S. Zhou (2011). A simple approach to emulation for computer models with qualitative and quantitative factors. *Technometrics 53*(3), 266–273.