MOANA: Modeling and Analyzing HPC I/O Variability

ABSTRACT

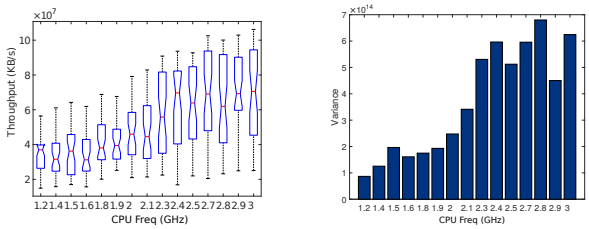
Exponential increases in complexity and scale make variability a growing threat to sustaining HPC performance at exascale. Performance variability in HPC I/O is common, acute, and formidable. As a result, it is difficult and sometimes even impossible to isolate the root cause of parallel application performance variability. We take the first step towards comprehensively studying linear and nonlinear approaches to modeling variability in HPC systems, especially the mission critical I/O subsystems. We create a modeling and analysis approach (MOANA) that predicts HPC I/O variability for thousands of software and hardware configurations on highly parallel shared-memory systems. Our findings indicate that highly dimensional, nonlinear models are essential to address fundamental limitations in linear approaches to variability prediction. Specifically, we identify two nonlinear approaches, linear Shepard interpolation (LSP) and multivariate adaptive regression (MARS), that are each an order of magnitude more accurate than linear regression for variability prediction.

1 INTRODUCTION

Modern high performance computing (HPC) systems are required to meet the evergrowing demands of many grand challenges for scientific computing as well as other domains such as e-commerce [25, 46]. Thus, HPC systems continue to grow in scale as well as evolve in terms of the underlying hardware and software substrates. This scale and evolution contribute to performance variability[1], which in turn challenges our ability to rigorously examine and improve system designs.

Highly variable observations in large complex systems can make it difficult, and sometimes even impossible, to fully optimize for performance. Such variability is cited as a significant barrier to exascale computing [22, 35]. Unfortunately, variability (which includes OS jitter[27]) is both ubiquitous and elusive as it causes pervade and obscure performance across the systems stack from hardware [7, 18, 29, 30] to middleware [5, 15, 26, 36, 44] to applications [16] to extreme-scale systems [22, 23, 34, 35, 42]. Additionally, as a number of recent reports attest [22, 35], performance variability at scale can significantly reduce performance and energy efficiency [8, 13, 16, 27].

Variability is a persistent challenge in HPC experimental systems research, which becomes impracticable at exascale. Experimental HPC systems research uses sound methods generally, but often relies upon experiential techniques to address variability. Such techniques include disabling subsystems not under study, discounting outliers, and using average values without statistical confidence. These techniques may be appropriate for testing one system, but not another. They may work well initially, but fail with increases in scale and complexity. Moreover, such techniques can fail, and may even mislead, when the system under study itself exhibits high variability. This makes it often impossible to discern why the methods work or why they do not, which is crucial for drawing high confidence inferences from experimentally observed results.



(a) I/O throughput vs. CPU freq (b) I/O variance vs. CPU freq

Figure 1: Throughput (a) and throughput variance (b) vs. CPU frequency for small write intensive workloads (file size of 512 KB and 64 threads of the IOZone benchmark) on a lone guest Linux operating system (Ubuntu 14.04 LTS over XEN 4.0) with a dedicated 2TB HDD on an Intel (Haswell) platform. Each result for a single CPU frequency configuration is generated from 40 separate executions.

Figure 1(a) illustrates the impact of system parameterization on performance variability under the IOZone benchmark [3]. In this example, I/O performance variability generally increases with CPU frequency for a series of small, intense write operations in a virtualized environment[2]. Figure 1(b) shows the measured throughput variance for the same data set. The figure provides useful and potentially actionable variability information since it shows how much variability results from different configuration settings. However, an approach based on this type of pairwise analysis is not scalable.

Table 1 shows the number of available parameters for the study in this paper. In this example, one would need $\binom{7}{2} = 21$ pairwise plots to display the two-way relationship of parameter effects on variability. For higher order (e.g., three way or more) relationships and a larger number of parameters (e.g., 100 to 200) that are common in parallel and distributed systems, the parameter space explodes quickly to tens of thousands or millions of possible configurations. Assuming it takes 30 seconds for an IOZone run, it would take over 8 hours to test just 1,000 configurations. For 95% statistical significance, we would run this configuration tens of times or more which would take weeks. The use of brute force for validation of our techniques with longer average IOZone runs means the experiments in this paper took months.

Figure 2 shows the difficulties apparent when attempting to model, characterize, and analyze variability in HPC I/O. For each bar in the graph, we measure IOZone throughput for all available permutations of file size, record size, thread number, and hypervisor scheduler for all pairings of three I/O schedulers (Completely Fair Queuing (CFQ) [1], Deadline (DEAD) [2], and NOOP [4]) and 15

[1]We use the term *performance variability* to describe the general spread or clustering of a measured performance metric such as execution time or throughput. Variability can be expressed in standard statistical terms such as standard deviation.

[2]We began with extensive bare-metal experiments and found that observed variabilities (and predictions) on bare metal were not significantly different from VM lone guest OS configurations for our I/O studies. VMs enabled fully-automated, month-long experiments without the manual reboots required to reconfigure the bare-metal OS installation for some variables.

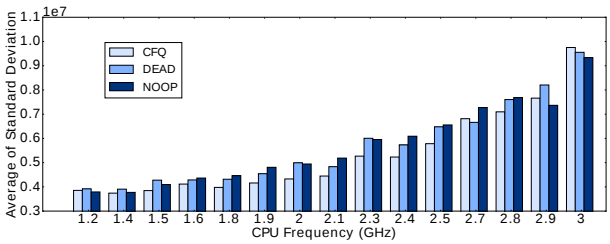


Figure 2: Nonlinearity effect of CPU frequency and VM I/O scheduler. We run the IOZone benchmark on a virtual operating system whose host system supports multiple CPU frequencies. The test is repeated 40 times under each single system configuration that includes the parameters: CPU frequency, I/O scheduler for host system, I/O scheduler of virtual machine, file size of IOZone, record size of IOZone, and number of IOZone threads.

CPU frequencies. Standard deviation (y-axis) is calculated across 40 runs for each permutation and these standard deviations are averaged to provide a single result shown in the figure for each pairing of I/O scheduler and CPU frequency (x-axis).

While §2 provides more details for these experiments, here we focus on just three sets of data points at 1.2 GHz, 2.3 GHz, and 2.7 GHz. In these three cases, the lowest average standard deviation—or "best configuration" if we are interested in minimizing or controlling standard deviation through system parameterization—is a different I/O scheduler for each CPU frequency. Additionally, while the variability again generally increases with CPU frequency, the trend lines for each I/O scheduler intersect and cross. This makes accurate prediction of standard deviation using linear approximation techniques problematic, since for example, NOOP is a clear winner from 2.4 GHz to 2.8 GHz but this trend does not persist at 2.9 GHz and 3.0 GHz.

In this paper, we create MOdeling and ANAlysis techniques or MOANA to characterize and predict HPC I/O variability while simultaneously maintaining scalability and capturing the nonlinear aspects of variability. Our focus on I/O is driven and motivated by the data-intensive nature of modern and emerging HPC applications. The experimental findings and observations motivate us to use advanced analytical tools (i.e., two nonlinear approximation methods) to predict, characterize, and analyze the performance variability of HPC I/O workloads. Specifically, we make the following contributions in this paper:

- a first-of-its-kind detailed study of the strengths and weakness of linear and nonlinear techniques to model, predict, and analyze HPC I/O variability for thousands of configurations on highly-parallel shared-memory systems;
- design, implementation, and testing of MOANA, a nonlinear variability prediction methodology;
- demonstration of nonlinear variability prediction techniques that result in an order of magnitude accuracy improvement over best-in-class linear regression;
- three detailed MOANA use cases providing analysis of the magnitude and causality of variability for hundreds of previously unseen system configurations; and
- a commitment to make the MOANA techniques and our datasets open source and available to the community. [3]

[3]URL redacted for paper review.

2 EMPIRICAL VARIABILITY ANALYSIS

2.1 Experimental Setup

A brute force methodology where we measure the variability effects of every combination of variable empirically is impractical at exascale given the experimental time required; however, the brute force approach can yield some insights to variability and can provide a baseline for comparison of new methods for analyzing variability.

Brute force experiments using all valid permutations of the parameters from Table 1 result in a total of over 95K unique configurations. For each configuration, we conduct 40 runs. Assuming data normality, which remains to be shown but is commonly used in practice, this results in a 95% statistical confidence in the resulting data set. The standard deviation of these 40 runs is used as a proxy for variability without loss of generality. We limited our parameterizations somewhat to ensure our brute force approach could provide baseline performance measurements in less than six months of runtime for a full set of data.

We selected the IOZone benchmark due to its large parameter space and its focus on I/O performance throughput. IOZone application parameters include: I/O operation modes (file write, file read, file initial write, etc.), file size, record size, and number of threads. These were selected, based on our past experience, to enable representation of a large set of HPC I/O behaviors.

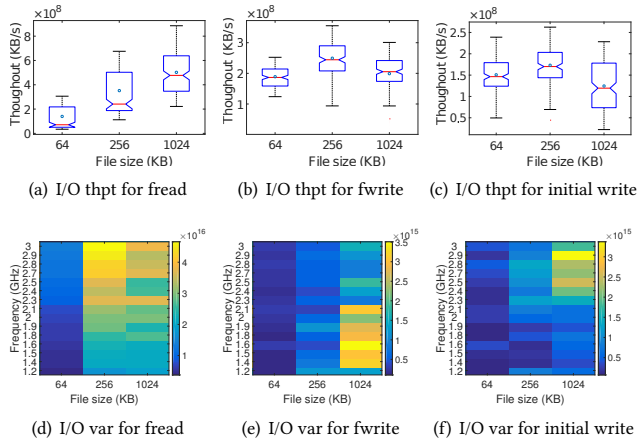
We focus on parallel shared-memory systems used as a common building block in HPC. Following extensive bare-metal experimentation, we found no statistically significant performance variability difference between bare-metal and lone-guest virtualized OS configurations. To minimize the number of parameter-required reboots during month-long experiments, all experiments were performed on a lone guest Linux operating system (Ubuntu 14.04 LTS//XEN 4.0) on a dedicated 2TB HDD on a 2 socket, 4 core (2 hyperthreads/core) Intel Xeon E5-2623 v3 (Haswell) platform with 32 GB DDR4. System parameters include: CPU frequency, host I/O scheduling policy (CFQ, DEAD, NOOP), and VM I/O scheduling policy (CFQ, DEAD, NOOP). These were selected based upon their common availability, their accessible user-level control, the likelihood they would affect variability, and our own familiarity and expertise.

We also studied parallel file systems (e.g., Lustre) and observed high performance volatility in the network that masks nodal contributions to variability. Thus, our initial focus is on localized techniques in highly-parallel, shared-memory clusters (4 cores x 2 sockets x 2 hyperthreads) up to 256 threads.

2.2 Empirical Analysis

As it would be impossible to manually inspect, compare and analyze all 95K possible application and system configurations, we focus on a small subset to demonstrate that while reasonable conclusions can be drawn from some experiments, the high-dimension, nonlinear aspects of the data set make actionable information difficult to discern, especially at scale.

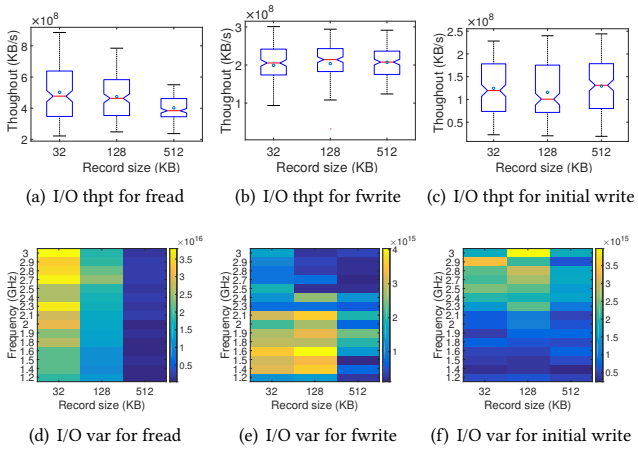
Effect of file size Figure 3 shows experiments designed to examine the effects of file size on I/O variability. Figure 3(a) shows raw I/O throughput increases with file size for file read operations. This is expected as the major I/O time is for seek operations, and



(a) I/O thpt for fread (b) I/O thpt for fwrite (c) I/O thpt for initial write

(d) I/O var for fread (e) I/O var for fwrite (f) I/O var for initial write

Figure 3: I/O throughput (@freq: 1.5 GHz, 2.0 GHz, 2.5 GHZ, and 3.0 GHZ) as a function of file size for three different I/O op modes (a, b, and c). Heat map of I/O throughput variance (y-axis-right) as a function of CPU frequency (y-axis-left) and file size (x-axis) for three different I/O op modes (d, e, and f). Record size = 32 KBytes, Threads = 256.



(a) I/O thpt for fread (b) I/O thpt for fwrite (c) I/O thpt for initial write

(d) I/O var for fread (e) I/O var for fwrite (f) I/O var for initial write

Figure 4: I/O throughput (@freq: 1.5 GHz, 2.0 GHz, 2.5 GHZ, and 3.0 GHZ) as a function of record size for three different I/O op modes (a, b, and c). Heat map of I/O throughput variance (y-axis-right) as a function of CPU frequency (y-axis-left) and record size (x-axis) for three different I/O op modes (d, e, and f). File size = 1024 KBytes, Threads = 256.

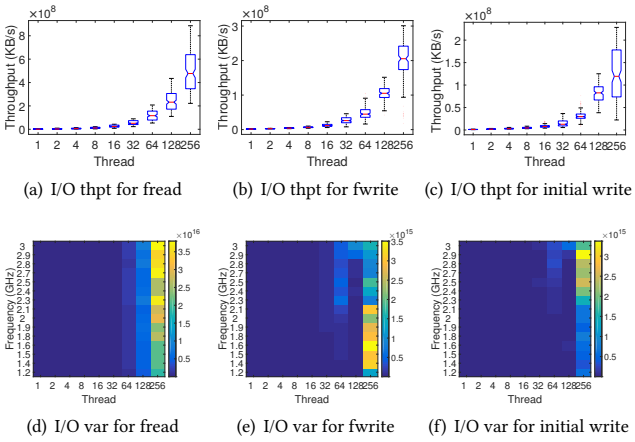
increasing file sizes imply each seek is followed by a sequential read of larger data. For file write operations (Figure 3(b)) and file initial write operations (Figure 3(c)), the I/O throughput initially increases but eventually decreases. Here, first most writes are absorbed in the cache, but as the cache becomes full, the disk flush operations come into play and reduce overall throughput. Figure 3(d) shows that I/O variance for file read operations is highest for medium file sizes and higher CPU frequency. A reason for this is that initially, the throughput is driven by disk seeks, but as the role of seeks decrease, the role of I/O-system interactions become more pronounced. As these interactions are the effect of CPU frequency and scheduling decisions, the observed variance increases. For file write operations, variance is higher for larger sized files at lower CPU frequency as shown in Figure 3(e). Figure 3(f) shows that for initial write operations, larger file sizes with higher frequency exhibit the most variance. This is potentially due to the need for more disk flushes, and the higher CPU frequencies triggering faster scheduling decisions, resulting in increased non-deterministic interactions between the I/O sub-components, which in turn can increase measure variance.

Effect of record size Figure 4 shows experiments designed to examine the effects of record size on I/O variability. Figure 4(a) shows raw I/O throughput decreases with increases in record size for file read operations. For file write operations (Figure 4(b)) and file initial write operations (Figure 4(c)), the I/O throughput remains unchanged. I/O variance results vary extensively. Smaller record size and higher CPU frequency give the highest variation for file read operations. Medium record size and lowest CPU frequency give the highest variation for write operations. Medium record size and higher frequency give the highest variation for initial write operations. See Figures 4(d), 4(e), and 4(f), respectively. Once again, the interaction of different record sizes and the I/O sub-system yield complex non-deterministic interactions that are not straight-forward to explain via understanding of system-level implementation details only. Consequently, there is a need for designing better approaches to capturing, modeling, and mitigating variance in such systems.

Effect of number of threads Figure 5 shows experiments designed to examine the effects of number of threads on I/O variability. In this case, raw I/O throughput increases with the number of threads for all three modes (file read, file write, and file initial write)—see Figures 5(a), 5(b), and 5(c). I/O throughput variance also increases with the number of threads: highest for file read operations (Figure 5(d)) and file write operations (Figure 5(e)) at the highest frequency. File read operations have high variance in the lower frequency range as well (Figure 5(d)).

Variability Trends A meta-analysis of Figures 3- 5 is appealing in search of trends in the data. Consider the following experiment shown in detail for fwrite in Figure 6. We repeat the file size (Figure 6(a)) and record size (Figure 6(b)) experiments calculating the change in variability when the number of threads decreases from 256 to 64. The resulting heat maps show that the variability when changing thread counts (at large and small file sizes) is sensitive to CPU frequency variations. Figure 6(c) shows that variability when changing file sizes is not particularly sensitive to CPU frequency variations (i.e., only the highest frequency seems to matter). For a system where file size and CPU frequency variations are common (e.g., most shared-memory HPC systems), there is no clear optimal operating configurations. This points to the challenge identified above that understanding the runtime interactions requires more than just the system-level implementation details, mainly due to the many dynamic interactions between various sub-systems.

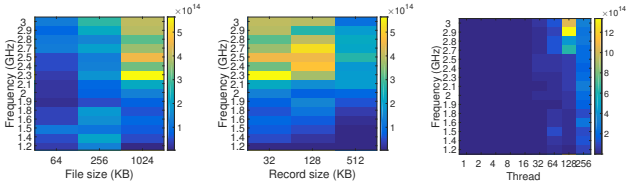
To illustrate further, consider Figure 7 showing the per thread I/O throughput as the number of threads increases (x-axis). All of the subfigures on the left of Figure 7 (a, c, and e) use 1.2 GHz for



(a) I/O thpt for fread (b) I/O thpt for fwrite (c) I/O thpt for initial write

(d) I/O var for fread (e) I/O var for fwrite (f) I/O var for initial write

Figure 5: I/O throughput (@freq: 1.5 GHz, 2.0 GHz, 2.5 GHZ, and 3.0 GHZ) as a function of number of threads for three different I/O op modes (a, b, and c). Heat map of I/O throughput variance (y-axis-right) as a function of CPU frequency (y-axis-left) and number of threads (x-axis) for three different I/O op modes (d, e, and f). File size = 1024 KBytes, Record size = 32 KBytes.

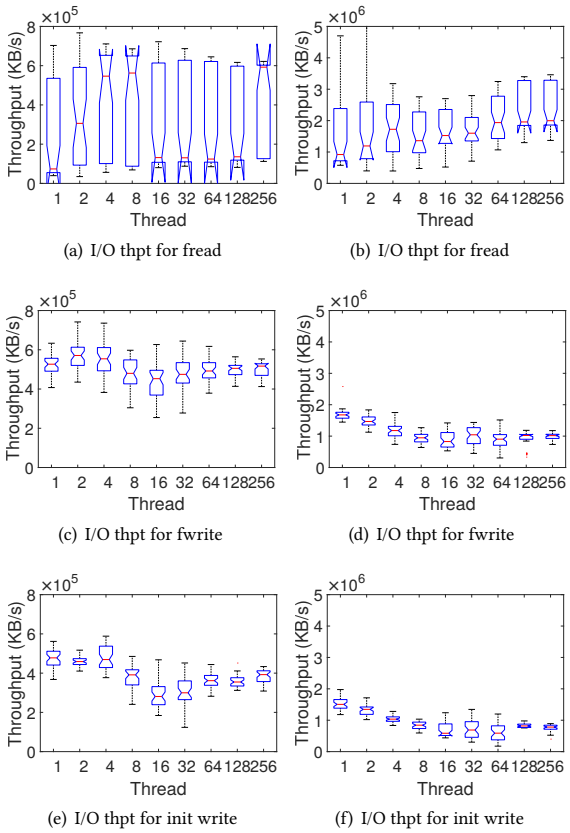


(a) file size (b) record size (c) # of threads

Figure 6: Heat map of change ((a) and (b)) in I/O throughput variance (y-axis-right) from 256 threads (Figure 3(e) and Figure 4(e)) down to 64 threads as a function of CPU frequency (y-axis-left) and file size (x-axis) and record size (x-axis). Heat map of change (c) in I/O throughput variance (y-axis-right) from 1024 KBytes file size (Figure 5(d)) down to 64 KBytes. Results for fwrite are shown.

CPU frequency and 64 Kbyte file size. All of the subfigures on the right of Figure 7 (b, d, and f) use 3.0 GHz and 1024 Kbyte file size. There is a stark contrast between these two columns of subfigures. There seems no discernible pattern to the resulting change in the variance and no clear optimal operating configuration.

Limitations In these examples, we are only considering a few hundred permutations of I/O modes, CPU frequency, thread count, file size, and record size from among over 95K. This limits this type of analysis to a very small subset of the experimental data set. Analyses of the combined effects of multiple variables and their non-linear interactions is severely limited. While some causality can be inferred from the analyzed data, any conclusions lack the full context of the data set and cannot be easily generalized. For these reasons, and the manual nature of this approach, we next consider methods for automating analysis of variability.



(a) I/O thpt for fread (b) I/O thpt for fread

(c) I/O thpt for fwrite (d) I/O thpt for fwrite

(e) I/O thpt for init write (f) I/O thpt for init write

Figure 7: Each subfigure shows the per thread I/O throughput as the number of threads increases. All of the subfigures on the left ((a), (c), and (e)) use 1.2 GHz for CPU frequency and 64 Kbyte file size. All of the subfigures on the right ((b), (d), and (f)) use 3.0 GHz and 1024 Kbyte file size. Other fixed parameters: host scheduler = CFQ, VM I/O scheduler = NOOP, record size = 32 KBytes.

	Parameters	Number of levels	Levels
Hardware	CPU Clock Frequency (GHz)	15	1.2, 1.4, ⋯ , 2.9, 3.0
Operating System	I/O Scheduling Policy	3	CFQ, DEAD, NOOP
	VM I/O Scheduling Policy	3	CFQ, DEAD, NOOP
Application	Number of Threads	9	1, 2, 4, 8, ⋯ , 256
	File Size (KB)	3	64, 256, 1024
	Record Size (KB)	3	32, 128, 512
	I/O op mode	13	fread, fwrite, ⋯

Table 1: Parameters used in our study of I/O variability. Note that both File Size and Record Size have three levels; but there are only 6 distinct, valid combinations of File Size and Record Size.

3 ANOVA VARIABILITY ANALYSIS

The analysis of variance (ANOVA) [32] method is commonly used to quantify the variability (i.e., variance) of a data set. Using an ANOVA-based experimental setup, we can identify parameters that significantly affect variability. The process can be automated to some extent and can be used to examine large sets of data. Once again, we use standard deviation as a measure for variability without loss of generality.

Figure 8 shows the distribution of the standard deviation across all possible configurations. We fix the host and VM I/O schedulers

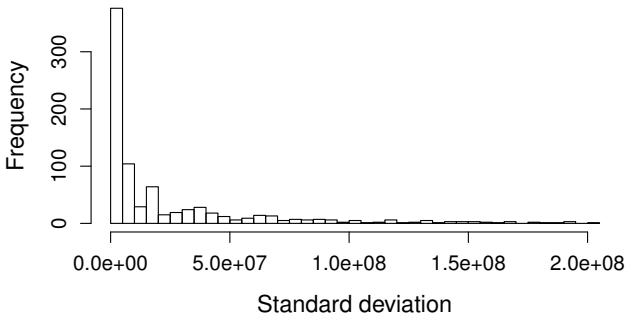


Figure 8: Distribution of measured standard deviations for 810 configurations with host I/O scheduler=CFQ, VM I/O scheduler=DEAD, mode=ReverseRead.

in this figure to reduce the data shown to 810 unique configurations. Although the histogram exhibits a long tail pattern because most standard deviations are low, some configurations can reach a value as high as 2.0×10^8 (i.e., an order of magnitude or more greater than the average throughput).

Table 2 shows the results after using ANOVA to identify system and application parameters that are statistically significant.[4] The parameters in Table 2 are treated as categorical variables: the number of degrees of freedom "Df" is one less than the number of levels (sampled values) for that parameter; "P value" is the probability that this parameter does not affect variability; and "Sig" is a notation to categorize the relative score of the P value where more stars indicate a lower P value.

The Df value describes the parameter space for each variable in our experiments. For example, in this data set we use 3 different file sizes and 9 thread combinations. This ANOVA experiment hypothesizes that the row parameter does not affect variability. Extremely low P values (e.g., ≤2.2e−16 for Thread) means we reject our hypothesis and it is almost a statistical certainty the parameter (e.g., Thread) affects variability. Since the P value for Thread is very low, the significance value (Sig.) is set to three stars to denote the high likelihood the hypothesis is false. Table 2 shows 19 out of 21 rows received at least one star in the "Sig." column, in which a term with a star indicates significant influence on variability. This ANOVA experiment shows that nearly all of the parameters studied (and their second factor configurations, e.g., Filesize x Thread) affect I/O variability in a statistically significant way.

Limitations In some ways, the ANOVA approach to studying variability is an improvement over the empirical approach. ANOVA techniques can analyze a large set of information for single and combined effects of parameter settings on variability. Causality can be inferred from this data, however the question of causality is limited by a simple hypothesis: whether a variable is a likely contributor to variability in our study.

The effective conclusion is that nearly all the observed variables and their combinations cause variability. While this is likely because we selected variables that we had previously observed empirically as having impact on variability, it is not particularly useful

[4] A single factor (or row in Table 2) is statistically significant if changing its value will affect the performance variability in a statistically significant way.

	Df	Sum of squares	Mean Squares	F value	P value	Sig.
Filesize	2	2.04E+15	1.02E+15	1.370E+02	≤2.2E-16	***
Recsize	2	3.29E+14	1.65E+14	2.208E+01	2.764E-10	***
IOsche	2	9.06E+13	4.53E+13	6.077E+00	2.307E-03	**
VMIOsche	2	2.12E+14	1.06E+14	1.423E+01	6.835E-07	***
Thread	8	3.64E+17	4.54E+16	6.093E+03	≤ 2.2E-16	***
Freq	14	1.97E+16	1.41E+15	1.889E+02	≤ 2.2E-16	***
Filesize×Recsize	1	5.43E+13	5.43E+13	7.276E+00	7.007E-03	**
Filesize×IOsche	4	2.56E+14	6.40E+13	8.585E+00	6.593E-07	***
Filesize×VMIOsche	4	1.15E+14	2.88E+13	3.860E+00	3.896E-03	**
Filesize×Thread	16	1.78E+16	1.11E+15	1.492E+02	≤ 2.2E-16	***
Filesize×Freq	28	6.64E+14	2.37E+13	3.180E+00	3.234E-08	***
Recsize×IOsche	4	9.07E+13	2.27E+13	3.041E+00	1.622E-02	*
Recsize×VMIOsche	4	1.09E+14	2.73E+13	3.657E+00	5.568E-03	**
Recsize×Thread	16	2.04E+15	1.27E+14	1.708E+01	≤ 2.2E-16	***
Recsize×Freq	28	4.54E+14	1.62E+13	2.175E+00	3.259E-04	***
Iosche×VMIOsche	4	5.30E+13	1.32E+13	1.776E+00	1.307E-01	
Iosche×Thread	16	1.23E+15	7.66E+13	1.027E+01	≤ 2.2E-16	***
Iosche×Freq	28	3.25E+14	1.16E+13	1.559E+00	3.052E-02	*
VMIOsche×Thread	16	3.10E+15	1.94E+14	2.598E+01	≤ 2.2E-16	***
VMIOsche×Freq	28	2.68E+14	9.57E+12	1.283E+00	1.454E-01	
Thread×Freq	112	3.37E+16	3.01E+14	4.035E+01	≤ 2.2E-16	***
Residuals	6950	5.18E+16	7.46E+12			

Table 2: An ANOVA experiment for a fixed IOZone mode (initialwrite) that examines whether variability is influenced by single or two-level parameter changes. An extremely low P value (e.g., ≤2.2e−16 for Thread) means it is almost certain statistically that the parameter in the associated row affects variability.

to determine causality by variable. Furthermore, this conclusion does not provide any indication as to how much each variable contributes to variability. We could certainly be more creative in our statistical hypothesis, but regardless, ANOVA will not expose the relative magnitude of the variability contribution for a given variable. Nor will ANOVA, a linear technique, capture the nonlinear combined effects of more than a few variables. Knowing relative magnitude could enable managing variability by, for example, selecting configurations with high likelihood of low variability. This is exactly the type of actionable information we need to affect system design. Understanding high-order, nonlinear effects can make these techniques more robust as systems increase in scale and complexity. In the next three sections, we propose and evaluate scalable, nonlinear techniques for their ability to analyze variability and reveal magnitude and causality.

4 MOANA OVERVIEW

Our modeling and analysis approach (MOANA) leverages advanced approximation methods to predict I/O performance variability. The methods we have selected—modified linear Shepard (*LSP*) algorithm, and multivariate adaptive regression splines (*MARS*)—are capable of approximating nonlinear relationships in high dimensions. This increases the likelihood that given a system and application configuration, the resulting models will accurately predict the variability. We provide a detailed mathematical description of the LSP and MARS predictors in Section 6. For now, we provide an overview of MOANA for use in our variability analyses discussed in Section 5.

We propose the concept of a *variability map* to describe and predict variability. Let the configuration x be an m-dimensional vector of parameters. The variability map is a function $f(x)$ that gives the variability measure (i.e., standard deviation in our context) at x. The variability map approximation $\tilde{f}(x)$ is constructed from experimental data using the LSP and MARS methods and can be used to predict variability for any given configuration x.

We use the data collected from our brute force approach (Section 2.1) to train our LSP and MARS models. Recall from Table 1 the total number of measured configurations is:

$$15(\text{Freq}) \times 9(\text{Thread}) \times 6(\text{Filesize x Recsize}) \times 3(\text{I/O Sche}) \times 3(\text{VM I/O Sche}) \times 13(\text{I/O Op Mode}) = 94770.$$

The LSP and MARS methods require a numeric value, x, defined in our experiments as:

$$x = (\text{Frequency, Threads, File Size, Record Size}),$$

which has $15 \times 9 \times 6 = 810$ distinct configurations assuming fixed values for I/O Scheduler, VM I/O Scheduler, and I/O Operation Mode. For each distinct I/O Scheduler, VM I/O Scheduler, and I/O Operation Mode combination, we will build a variability map approximation $\tilde{f}(x)$. In total, we will construct $3 \times 3 \times 13 = 117$ variability maps.

Thus, we can denote the configuration as $x^{(k,\ l)}$, and denote the corresponding variability value as $f_k^{(l)}$, where $k = 1, \cdots, 810$ and $l = 1, \cdots, 117$. For a given l, the dataset is $\{x^{(k,\ l)}, f_k^{(l)}\}, k = 1, \cdots, 810$, and the corresponding variability map approximation $\tilde{f}^{(l)}(x)$ can be obtained by using the LSP or MARS algorithms described in Section 6.

Predictor evaluation We define the following relative error as an evaluation criterion. That is

$$r = \frac{|\tilde{f} - f|}{f},$$

where $\tilde{f}$ is the predicted variability at a x, and f is the true variability (obtained from direct measurements).

We use statistical cross validation to compute the average relative error (*ARE*). For a given dataset $\{x^{(k,\ l)}, f_k^{(l)}\}, k = 1, \cdots, 810$, we randomly divide the dataset into two parts where the proportion of one part is p and the remaining proportion is $(1 - p)$. We use all configurations from the p portion of the data set as samples to train our predictive models. We use our trained predictors to predict all configurations from the $(1 - p)$ portion of the data set. We compute the ARE value for each data point in the test using the average of the relative error, r, for each data point. We repeat this random data set division procedure to construct 117 variability maps. The ARE is averaged again over the 117 trained models, and it will be assumed to be the true variability for the method.

Predictor comparisons By varying p we can observe the tradeoffs between predictor accuracy and the ratio of the training set to the predicted data points. Table 3 shows the ARE for linear regression (LR), LSP, and MARS as functions of the training sample proportion p, averaged over 117 variability maps. We are unaware of any existing methods to predict performance variability. Hence, we compare the proposed techniques to the general linear regression model. From the results in the table, it is clear that the proposed nonlinear methods out predict the linear regression by an order of magnitude. In this random division testing, the LSP method consistently outperforms MARS. For example, the ARE is around 15% under LSP for a setting that uses 30% of the data for training and predicts 70% of the data. The ARE is around 30% if one uses 10% data for training and 90% data for LSP. If we use half of the data set to predict the other half of the data set ($p = .5$) the ARE of the LSP method is about 12%. This set of experiments implies we could design a runtime system that learns from the system over time and gains accuracy with more experience.

Training Prop. p	Testing Prop.	LR (%)	LSP (%)	MARS (%)
0.9	0.1	323.09	11.13	56.61
0.8	0.2	323.47	11.27	57.59
0.7	0.3	324.00	11.47	58.35
0.6	0.4	324.58	11.72	59.94
0.5	0.5	324.59	12.22	62.35
0.4	0.6	325.82	13.25	66.52
0.3	0.7	327.02	15.44	71.49
0.2	0.8	329.56	19.33	79.27
0.1	0.9	339.32	30.44	100.14

Table 3: ARE for linear regression (LR), LSP, and MARS as functions of p, averaged over 117 variability maps and based on $B = 200$ repeats for random division.

5 MOANA VARIABILITY ANALYSIS

MOANA is the first-of-its-kind attempt to model and predict variability. Our MOANA approach can be used to predict configurations that have not been observed in the training set. While accuracy is important, we are additionally interested in whether MOANA is scalable and captures both the magnitude of variability and the causality.

In this section we attempt to predict 585 configurations not considered in the full, 95K-configuration training set described in Section 2.1. We calculate the average relative error (ARE) as discussed in the previous section for these new sets of experiments for comparison. Without loss of generality, we limit the experiments to a fixed CPU frequency (2.5 GHz) and a fixed number of threads (128). We select 5 valid combinations of file size and record size with all possible permutations of I/O scheduler, VM I/O scheduler, and I/O operation modes. Table 4 lists all the parameters in this study.

We selected these parameters for two reasons. First, we wanted to study the appropriateness of MOANA for answering variability magnitude- and causality-related questions. We consider these questions in our use cases throughout this section. Second, we wanted to challenge our techniques with potentially difficult configurations to gauge predictor accuracy and robustness. While several of our variables are categorical (e.g., I/O operation modes), a number of them are continuous (e.g., file size). In systems research it is common to measure exponentially distributed data points for a continuous variable (e.g., file size = 64, 256, 1024 KB). Reasons for this include time limitations, scalability studies, and system design considerations. In statistics however, evenly distributed data points are generally preferred to maximize model accuracy. In this paper, we took a systems approach to data collection (see Table 1). While this will inevitably lead to some model inaccuracies, it is more generally applicable to the common experimental system studies in the extant literature. To challenge MOANA, we selected relatively large file sizes (e.g., 512 and 768 KB), record sizes (e.g., 128 and 256 KB), and threads (e.g., 128) since the data in the training models is somewhat sparse surrounding these larger, previously unseen data points.

Throughout this section, we use scatter plots (e.g., Figure 9) to discuss the accuracy of our MOANA methodology for the 5×3×3×

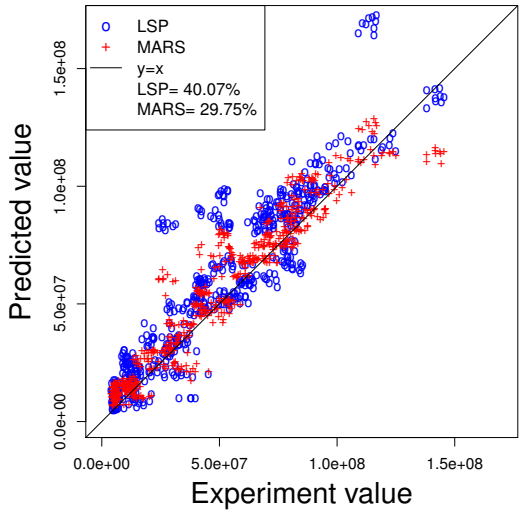


Figure 9: Comparison of the two proposed prediction models on new configurations. The y-axis shows the predicted standard deviation from our two models, while the x-axis shows the empirical standard deviation from 40 runs. The ARE for each method is also shown on the legend.

13 = 585 configurations described in Table 4. In Figures 9 — 11 and all subfigures, the y-axis shows the predicted standard deviation for both LSP and MARS models. The x-axis shows the empirical (measured) standard deviation with the unseen configuration. The $y = x$ diagonal line is a reference line that represents predicted values equal to the empirical (measured) values.

File size	Record size	I/O scheduler	VM I/O scheduler	I/O Mode
512	32, 128, 256	CFQ, DEAD, NOOP	CFQ, DEAD, NOOP	All 13 levels
768	32, 128	CFQ, DEAD, NOOP	CFQ, DEAD, NOOP	All 13 levels

Table 4: New configurations used for model validation.

Use Case I: A general model In this use case, we attempt to determine whether the MOANA approach results in a single model that predicts well generally. Figure 9 shows a scatter plot of the general prediction accuracy of our derived models for the 5×3×3×13 = 585 configurations described in Table 4. The MARS data points (cross point type in Figure 9) are generally clustered closer to the diagonal compared to the LSP data points (circle point type in Figure 9). This is confirmed with average relative error (ARE) calculations: MARS has a 29.75% ARE while LSP has a 40.07% ARE. This is the opposite finding from the previous section where LSP consistently outperforms MARS. Upon deeper inspection, we were able to determine that MARS is more sensitive to file size and record size parameters in the training set. Since much of the accuracy gains come from tight data points for continuous variables (due to the sparseness described previously at larger sizes), MARS likely gains accuracy due to its sensitivity to these continuous variables. LSP likely performs better when the variables predicted are randomly selected from within the population as done in the training set experiments in the previous section. As expected, the ARE for both methods is generally larger than observed values for LSP and MARS from the previous section (see Table 3) but consistently much better than linear regression.

We use an example to illustrate the detailed analysis for two data points. Consider the cases of file size=512 and file size=768. Depending on the record size variable settings, the ARE values using MARS for file size=512 vary from 15.54% to 48.29% with an average ARE of 29.4% and for file size=768 from 28.85% to 31.60% with an average ARE of 30.23%. These errors are close to the average for the general model, but they would likely be improved with more data points. Upon deeper inspection, it is the record size=256 configuration for the file size=512 that causes the error to be higher in that case. The inaccuracy comes from the sparsity of points at the higher record sizes as mentioned previously.

Use Case II: Variability by application (I/O Mode) In this use case, we attempt to determine whether the MOANA approach can be used to classify the predicted applications for the previously unseen configurations (i.e., I/O Mode in Table 4) by the magnitude of their variability. Figure 10 plots the prediction results for varying I/O operation modes. In all but 3 cases (Fread, Fwrite, and Initial Write), the LSP or MARS predictions beat the average ARE values for the general model from Use Case I (MARS=29.75%, LSP=40.07% in Figure 9). For the relatively accurate cases, the magnitude of variance is lowest and tightly clustered for Pwrite, Rewrite, Random Write. As the variance grows it generally becomes less tightly clustered for Reverse Read and Random Read. Pread, Read and Stride Read show the highest variances and the least clustered results. For 10 of the 13 applications examined, this constitutes a reasonably accurate rank ordering of variability by magnitude and the isolation by application provides some notion of causality by variable.

MOANA's strength, illustrated by this example, is classifying variables by magnitude of variability across a large set of experiments accounting for the nonlinear effects of high-order variables. We leave it to our future work to determine exactly why inaccuracies occur in the Fwrite, and Initial Write (i.e., file write) predictions, but we speculate there are two reasons: (1) page cache operations result in flushes of commit operations to update the disk store and the variability introduced is unaccounted for in our model; and (2) though we did not observe differences in the bare-metal versus VM variability studies, it is possible that under certain extreme write conditions (e.g., the buffer cache fills) the interactions of the nested (host and VM) file systems result in unanticipated performance changes [20]. For Fread, the inaccuracies are not as drastic as Fwrite and Initial Write but still exceed the average ARE for the general model. We speculate these inaccuracies are due to a combination of caching and prefetching effects that can be influenced significantly by the experimental setup (e.g., multi-level cache buffers).

Use Case III: Variability by file and record size In this use case, we attempt to determine whether the MOANA approach can be used to classify the predicted variability for previously unseen configurations of file and record size by the magnitude of their variability. Figure 11 plots the prediction results for valid file and record size combinations from Table 4. In all but 1 case, the LSP or MARS predictions beat the average ARE values for the general

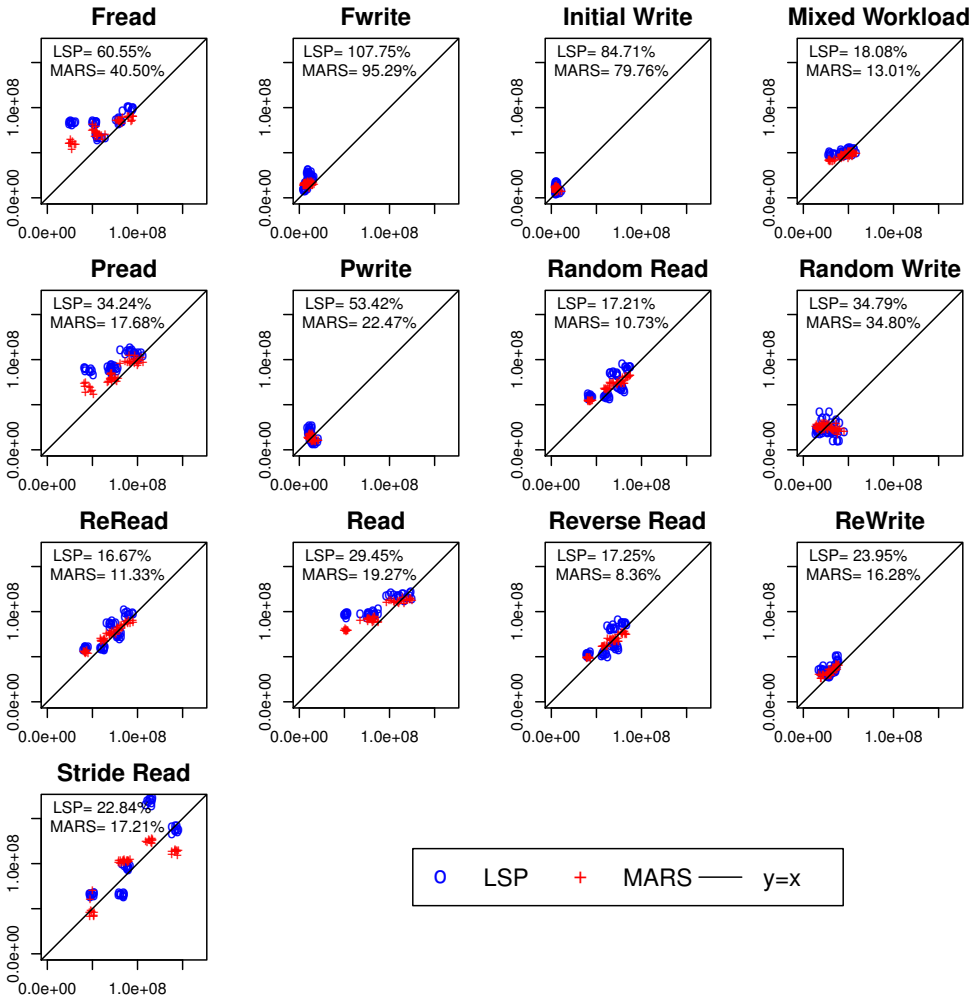


Figure 10: Prediction scatter plots of different I/O operation modes. Points in the same plot have the same mode. The x-axis is the empirical standard deviation observed from 40 runs. The y-axis is the predicted standard deviation obtained from our two models. The ARE for each method is also shown on the legend.

model from Use Case I (MARS=29.75%, LSP=40.07% in Figure 9). For the inaccurate case (file size=512, record size=256), we know from Use Case I that the inaccuracy comes from the sparsity of points at the higher record sizes; in other words, the best accuracies occur at the smallest record sizes for both file sizes.

Recall that for this experiment, for each file and record size pair, we vary I/O Scheduler, VM I/O Scheduler, and I/O Mode (or application). The results in Figure 11 indicate that second order effects are influential in the magnitude of the predicted variability. For example, for file size=768 and record size=32, LSP ARE is under 20% and 2-3 distinct clusters are observable in the data. These clusters are affected by the application class (i.e., they are clustered by I/O modes that share characteristics such as writes). Identification of these higher order effects are another valuable contribution of the MOANA approach to analyzing variability. For 4 of the 5 scenarios studied, with consideration of the higher order effects, we can (with reasonable accuracy) rank order variability by magnitude and the isolation by file size, record size, and mode classification provides some notion of causality for a set of variables.

6 MOANA PREDICTION MODELS

For completeness, we provide details for LSP (modified linear Shepard algorithm) and MARS (multivariate adaptive regression splines) approximation methods. Recall in Section 4 we described a variability map as a function $f(x)$ that gives the variability measure (i.e., standard deviation in our context) at x. The variability map approximation $\tilde{f}(x)$ is constructed from experimental data and can be used to predict variability for any given configuration x.

6.1 Modified Linear Shepard Algorithm

The linear Shepard algorithm is derived from the modified Shepard algorithm [40]. Given n distinct data points $x^{(1)}, \ldots, x^{(n)}$ and values $f_k = f\left(x^{(k)}\right)$, the linear Shepard algorithm constructs an

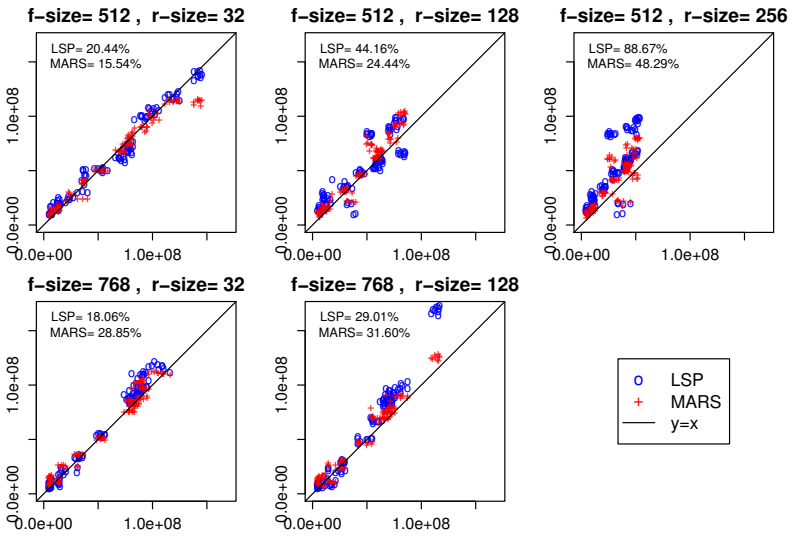


Figure 11: Prediction scatter plots of different file and record size combinations. Points in the same plot have other parameters fixed. The x-axis is the empirical standard deviation observed from 40 runs. The y-axis is the predicted standard deviation obtained from our two models. The ARE for each method is also shown on the legend.

interpolant to f of the form

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

where the locally supported weights are

$$W_k(x) = \left[\frac{\left(R_w^{(k)} - d_k(x)\right)_+}{R_w^{(k)} d_k(x)} \right]^2, \quad d_k(x) = \left\| x - x^{(k)} \right\|_2,$$

and the local (linear, here) approximations $P_k(x)$ have the form

$$P_k(x) = f_k + \sum_{j=1}^{m} a_j^{(k)} \left(x_j - x_j^{(k)} \right).$$

The function $P_k(x)$ is the local linear weighted least squares fit around $x^{(k)}$ with the ith data point having weight

$$\omega_{ik} = \left[\frac{\left(R_p^{(k)} - d_i\left(x^{(k)}\right)\right)_+}{R_p^{(k)} d_i\left(x^{(k)}\right)} \right]^2, \qquad i \neq k.$$

Let $N_p = \min\{n, \lceil 3m/2 \rceil\}$ and $D = \max_{i,j} \left\| x^{(i)} - x^{(j)} \right\|_2$, and define

$$R^{(k)} = \min\left\{ r \;\middle|\; \overline{B\left(x^{(k)}, r\right)} \text{ contains at least } N_p \text{ points} \right\},$$

where $\bar{B}$ is the closure of the open ball $B\left(x^{(k)}, r\right) = \left\{ x \;\middle|\; \left\| x - x^{(k)} \right\| < r \right\}$. The values of $R_p^{(k)}$ and $R_w^{(k)}$ are then specified as

$$R_w^{(k)} = \min\left\{ \frac{D}{2}, R^{(k)} \right\}, \qquad R_p^{(k)} = 1.1 R^{(k)}.$$

Let $S = \{i_1, i_2, i_3, \ldots, i_{N_p - 1}\}$ be the set of indices corresponding to the $N_p - 1$ points that are closest to $x^{(k)}$, which determine the local least squares approximation $P_k(x)$. The weights satisfy $\omega_{i_j k} > 0$ because $R_p^{(k)}$ is slightly larger than $R^{(k)}$. Define an $(N_p - 1) \times m$ matrix A by

$$A_{j\cdot} = \sqrt{\omega_{i_j k}} \left(x^{(i_j)} - x^{(k)} \right)^t,$$

and $(N_p - 1)$-vector b by

$$b_j = \sqrt{\omega_{i_j k}} (f_{i_j} - f_k).$$

The coefficients $a^{(k)} \in E^m$ of $P_k(x)$, where E^m is m-dimensional real Euclidean space, are the minimum norm solution of the least squares problem

$$\min_{a \in E^m} \|Aa - b\|_2.$$

6.2 Multivariate Adaptive Regression Splines (MARS)

Let

$$C = \Big\{ 1, (x_j - t)_+, (t - x_j)_+ \;\Big|\; t = x_{kj}, \; 1 \le k \le n, \; 1 \le j \le m \Big\}$$

The basis functions $\mathcal{B}$ for MARS are chosen from products of functions in the set C:

$$\mathcal{B} = C \cup C \otimes C \cup C \otimes C \otimes C \cup \cdots.$$

At each iteration the model of the data is a C^0 m-dimensional spline of the form

$$\sum_{h_\alpha \in \mathcal{M}} \beta_\alpha h_\alpha(x),$$

where $\mathcal{M} \subset \mathcal{B}$, $|\mathcal{M}| \le n$, and the coefficients β_α are determined by a least squares fit to the data. $h_\alpha \in \mathcal{M}$ is constrained to always

be a spline of order ≤ 2 (piecewise linear) in each variable x_j. The initial model is $\tilde{f}(x) \equiv 1$. Let $\mathcal{M} \subset \mathcal{B}$ be the set of basis functions in the model at iteration q. The basis at iteration $q+1$ is that basis

$$\mathcal{M} \cup \left\{ h_\ell(x)(x_j - t)_+,\ h_\ell(x)(t - x_j)_+ \right\}$$

which minimizes the least squares error of the model using that basis over all $h_\ell(x) \in \mathcal{M}$ and $t = x_j^{(k)}$, $1 \leq j \leq m$, $1 \leq k \leq n$, subject to the constraint that $h_\ell(x)(x_j - t)_+$ is a spline of order 2 in x_j, and a spline of degree at most n_I in x (where n_I is the most variable interactions permitted). The iteration continues for some given number n_B of iterations or until the data are overfit, at which point the generalized cross-validation criterion

$$\mathrm{GCV}(\lambda) = \frac{\sum_{k=1}^{n} \left(\tilde{f}_\lambda\left(x^{(k)}\right) - f_k \right)^2}{(1 - M(\lambda)/n)^2},$$

(where λ is the number of basis functions in the model $\tilde{f}_\lambda$, $M(\lambda)$ is the effective number of parameters in $\tilde{f}_\lambda$), having been computed for each λ, is used to choose the final approximation $\tilde{f}_\lambda(x)$ that minimizes GCV(λ) with $\lambda \leq n_B$. This C^0 m-dimensional spline $\tilde{f}_\lambda(x)$ is the multivariate adaptive regression spline (MARS) approximation to the data. The constraints and greedy way $\mathcal{M}$ is constructed mean that $\tilde{f}_\lambda(x)$ is not necessarily the best approximation to the data by a spline of degree n_I generated from $\mathcal{B}$, or by a spline with n_B basis functions from $\mathcal{B}$.

7 RELATED WORK

HPC systems research is not immune to the current trend of criticisms of presentations of scientific results of rigor. Hoefler et al. [17] recently summarized the state of the practice for benchmarking in HPC and suggest ways to ensure repeatable results. Despite the recent kerfuffle, HPC researchers have been examining variance for a long time, e.g., in the 1990s IBM observed variance in uniprocessors [24], and Kramer et al. explored variation in large distributed memory systems more than a decade ago [19]. Such well-cited studies establish the existence of variability, yet variability is not typically factored into modern HPC application design.

Much of the work emanating from the HPC community has been focused on OS jitter [27], variations caused by the competition for resources between background processes and applications, or application interference [21, 45]. Some have simulated these effects at scale [13], while others have proposed applications [16] or systems [8] that can account for these types of variability. Additionally, schedulers are often identified as causes of significant variability in HPC systems [39]. Our focus is on identifying the controllable aspects of variability. For this first-of-its-kind study, we focus on variability in I/O with enough variables to observe high-dimension, nonlinear effects while enabling brute force evaluation of the viability of our approach. We leave isolation of the effects of additional variables, background jitter, and cross-application interference to future work.

At the heart of any runtime system lies an analytical engine that uses some form of predictor to determine whether and how to adapt the systems use of resources. Performance prediction of HPC and distributed applications is a well-studied field and recent works have used analytical [12, 38, 41], profile based [9, 33], and simulation based [11, 28] approaches, or a combination of these [47], to accurately predict overall performance. In contrast, detailed studies like ours that result in models or predictors that consider variability are almost nonexistent. Our work focuses on quantifying and modeling I/O variability due to application thread interaction and resource demand in parallel, shared memory systems.

Introducing determinism to achieve reproducibility has also been explored using environment categorization [31], statistical modeling [37], or variance-aware algorithm design [6]. Environment categorization considers the interactions and composition of hidden factors in the system (e.g., DataMill [14]). Our focus on modeling and managing variability and the application to experimental design complements these approaches and may influence these types of best practices.

A number of projects in computer architecture research have explored variability in studying and realizing new hardware, arising mainly from heterogeneity in the chip-level architecture. These projects are mainly focused on the consequences of the chips having a limited amount of power [7, 10, 18, 30, 43]. The resulting issues such as leakage current, cross-wire-signaling, and power budgeting, will likely contribute to observed system variability. If these techniques affect software performance variability, then they can be captured and are orthogonal to our proposed work. Otherwise, such variations are likely to be small relative to the variations observed much higher in the systems software stack. Similarly, the fundamental nature of our complementary approach, isolating cause and magnitude of variability in software, could be applied to architectural designs. For example, observing or anticipating the magnitude of controllable variability in the software layer can provide actionable information (e.g., alter processor speed or enable/disable a functional unit) to the hardware in service of system operating constraints (e.g., minimize performance margins).

8 CONCLUSIONS AND FUTURE WORK

MOANA uses nonlinear statistical techniques to capture the high order effects of systems and application configurations on HPC I/O variability. We showed that by building variability maps, or correlation vectors between configurations and variance, we can predict both the magnitude and causality of variability for hundreds of unseen configurations. We demonstrated that we can use MOANA to create a single model for all parameter settings (>70% accuracy for MARS) that is an order of magnitude more accurate than best available linear regression techniques. Our analyses showed MOANA can readily identify both first order effects (rank ordering modes by the variability magnitude) and higher order effects (clustering variability by mode for file size and record size studies) through comparison of variability maps.

MOANA is just the first step towards improving our understanding of HPC system variability. Our current study is limited to 95K configurations and highly-parallel shared-memory systems, which form the building blocks of high-performance systems and supercomputers. In future work, we plan to improve the variability map accuracy by altering our continuous variable training strategy to be more linear than exponential. We plan to also incorporate clustering analysis of variability to improve identification of higher order effects. Also, the introduction of network variability would extend MOANA to distributed systems.

REFERENCES

[1] Complete Fairness Queueing. https://www.kernel.org/doc/Documentation/block/cfq-iosched.txt

[2] Deadline I/O scheduler. https://www.kernel.org/doc/Documentation/block/deadline-iosched.txt

[3] Iozone benchmark. http://www.iozone.org/

[4] Noop I/O scheduler. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/ch06s04s03.html

[5] Hakan Akkan, Michael Lang, and Lorie M. Liebrock. 2012. Stepping Towards Noiseless Linux Environment. In *Proceedings of the 2Nd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM.

[6] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. 2009. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science* 410, 19 (2009), 1876–1902.

[7] Anys Bacha and Radu Teodorescu. 2013. Dynamic reduction of voltage margins by leveraging on-chip ecc in itanium ii processors. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 297–307.

[8] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, Susan Coghlan, and Aroon Nataraj. 2008. Benchmarking the Effects of Operating System Interference on Extreme-scale Parallel Machines. *Cluster Computing* 11, 1 (March 2008), 3–16.

[9] Julien Bourgeois and François Spies. 2000. Performance prediction of an NAS benchmark program with ChronosMix environment. In *Euro-Par 2000 Parallel Processing*. Springer, 208–216.

[10] Keith Bowman, James W Tschanz, Shih-Lien L Lu, Paolo Aseron, Muhammad M Khellah, Arijit Raychowdhury, Bibiche M Geuskens, Carlos Tokunaga, Chris B Wilkerson, Tanay Karnik, and others. 2011. A 45 nm resilient microprocessor core for dynamic variation tolerance. *Solid-State Circuits, IEEE Journal of* 46, 1 (2011), 194–208.

[11] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *UKSIM 2008*. IEEE.

[12] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. 1993. *LogP: Towards a realistic model of parallel computation*. Vol. 28. ACM.

[13] Pradipta De and Vijay Mann. 2010. jitSim: A Simulator for Predicting Scalability of Parallel Applications in Presence of OS Jitter. In *Euro-Par 2010 - Parallel Processing*. Lecture Notes in Computer Science, Vol. 6271. Springer Berlin Heidelberg, 117–130.

[14] Augusto Born de Oliveira, Jean-Christophe Petkovich, Thomas Reidemeister, and Sebastian Fischmeister. 2013. Datamill: Rigorous performance evaluation made easy. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*.

[15] M. Giampapa, T. Gooding, T. Inglett, and R.W. Wisniewski. 2010. Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK. In *IEEE SC*.

[16] Adam Hammouda, Andrew R. Siegel, and Stephen F. Siegel. 2015. Noise-Tolerant Explicit Stencil Computations for Nonuniform Process Execution Rates. *ACM Trans. Parallel Comput.* 2, 1 (April 2015), 7:1–7:33.

[17] Torsten Hoefler and Roberto Belli. 2015. Scientific Benchmarking of Parallel Computing Systems. In *SC*. ACM/IEEE.

[18] Youngtaek Kim, Lizy Kurian John, Sanjay Pant, Srilatha Manne, Michael Schulte, W Lloyd Bircher, and Madhu Saravana Sibi Govindan. 2012. AUDIT: Stress testing the automatic way. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*. IEEE, 212–223.

[19] William TC Kramer and Clint Ryan. 2003. *Performance variability of highly parallel architectures*. Springer.

[20] Duy Le, Hai Huang, and Haining Wang. Understanding Performance Implications of Nested File Systems in a Virtualized Environment. In *USENIX FAST'12*.

[21] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing variability in the IO performance of petascale storage systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*. IEEE.

[22] Robert Lucas, James Ang, Shekhar Borkar, William Carlson, Laura Carrington, George Chiu, Robert Colwell, William Dally, Jack Dongarra, Al Geist, Gary Grider, Rud Haring, Jeffrey Hittinger, Adolfy Hoisie, Dean Klein, Peter Kogge, Richard Lethin, Vivek Sarkar, Robert Schreiber, John Shalf, Thomas Sterling, and Rick Stevens. 2014. ASCAC Subcommittee for the Top Ten Exascale Research Challenges. (2014).

[23] Alessandro Morari, Roberto Gioiosa, Robert W Wisniewski, Francisco J Cazorla, and Mateo Valero. 2011. A quantitative analysis of os noise. In *IEEE IPDPS*.

[24] Ronald Mraz. 1994. Reducing the variance of point to point transfers in the IBM 9076 parallel computer. In *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press.

[25] J.T. Oden, O. Ghattas, J.L. King, B.I. Schneider, K. Bartschat, F. Darema, J. Drake, T. Dunning, D. Estep, S. Glotzer, M. Gurnis, C.R. Johnson, D.S. Katz, D. Keyes, S. Kiesler, S. Kim, J. Kinter, G. Klimeck, C.W. McCurdy, R. Moser, C. Ott, A. Patra, L. Petzold, T. Schlick, K. Schulten, V. Stodden, J. Tromp, M. Wheeler, S.J. Winter, C. Wu, and K. Yelick. 2011. Cyber Science and Engineering: A Report of the National Science Foundation Advisory Committee for Cyberinfrastructure Task Force on Grand Challenges. (2011).

[26] Jiannan Ouyang, Brian Kocoloski, John R. Lange, and Kevin Pedretti. 2015. Achieving Performance Isolation with Lightweight Co-Kernels. In *HPDC*. ACM.

[27] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. 2003. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *ACM ICS*.

[28] Sundeep Prakash and Rajive L Bagrodia. 1998. MPI-SIM: using parallel simulation to evaluate MPI programs. In *Proceedings of the 30th conference on Winter simulation*. IEEE Computer Society Press.

[29] A. Rahimi, D. Cesarini, A. Marongiu, R.K. Gupta, and L. Benini. 2015. Task scheduling strategies to mitigate hardware variability in embedded shared memory clusters. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*.

[30] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D Smith, Gu-Yeon Wei, and David Brooks. 2010. Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In *MICRO*. IEEE/ACM.

[31] Robert Ricci, Gary Wong, Leigh Stoller, Kirk Webb, Jonathon Duerig, Keith Downie, and Mike Hibler. 2015. Apt: A Platform for Repeatable Research in Computer Science. *ACM SIGOPS Operating Systems Review* 49, 1 (2015), 100–107.

[32] Andrew Rutherford. Introducing Anova and Ancova: A GLM Approach.

[33] Rafael H Saavedra and Alan J Smith. 1996. Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems (TOCS)* 14, 4 (1996), 344–384.

[34] B.W. Settlemyer, S.W. Hodson, J.A. Kuehn, and S.W. Poole. 2010. Confidence: Analyzing performance with empirical probabilities. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*.

[35] John Shalf, Sudip Dosanjh, and John Morrison. 2011. Exascale Computing Technology Challenges. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*.

[36] Kai Shen. 2010. Request behavior variations. *ACM SIGARCH Computer Architecture News* 38, 1 (2010), 103–116.

[37] David Skinner and William Kramer. 2005. Understanding the causes of performance variability in HPC workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*.

[38] David Sundaram-Stukel and Mary K Vernon. 1999. Predictive analysis of a wavefront application using LogGP. *ACM SIGPLAN Notices* 34, 8 (1999), 141–150.

[39] Vahid Tabatabaee, Ananta Tiwari, and Jeffrey K Hollingsworth. 2005. Parallel parameter tuning for applications with performance variability. In *ACM/IEEE SC*.

[40] William I. Thacker, Jingwei Zhang, Layne T. Watson, Jeffrey B. Birch, Manjula A. Iyer, and Michael W. Berry. 2010. Algorithm 905: SHEPPACK: Modified Shepard Algorithm for Interpolation of Scattered Multivariate Data. *ACM Trans. Math. Softw.* 37, 3 (Sept. 2010), 34:1–34:20.

[41] Arjan JC Van Gemund. 2003. Symbolic performance modeling of parallel systems. *Parallel and Distributed Systems, IEEE Transactions on* 14, 2 (2003), 154–165.

[42] Sarp Oral Feiyi Wang, David A Dillow, Ross Miller, Galen M Shipman, Don Maxwell, and Dave Henseler Jeff Becklehimer Jeff Larkin. 2010. Reducing application runtime variability on Jaguar XT5. (2010).

[43] Paul N Whatmough, Shidhartha Das, Zacharias Hadjilambrou, and David M Bull. 14.6 An all-digital power-delivery monitor for analysis of a 28nm dual-core ARM Cortex-A57 cluster. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*.

[44] Nicholas J Wright, Shava Smallen, Catherine Mills Olschanowsky, Jim Hayes, and Allan Snavely. 2009. Measuring and understanding variation in benchmark performance. In *HPCMP-UGC*. IEEE.

[45] Orcun Yildiz, Matthieu Dorier, Shadi Ibrahim, Rob Ross, and Gabriel Antoniu. 2016. On the root causes of cross-application I/O interference in HPC storage systems. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE.

[46] Thomas Zacharia, Jim Kinter, Rob Pennington, Ron Cohen, Larry Davis, Tiziana Di Matteo, Bill Harrod, George Karniadakis, Rubin Landau, Rich Loft, Michael Macy, Dick McCombie, Dave Randall, Steve Scott, Horst Simon, Thomas Sterling, Theresa Windus, and Rob Pennington. 2011. Report of the High-Performance Computing Task Force. (2011).

[47] Jidong Zhai, Wenguang Chen, and Weimin Zheng. 2010. Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. In *ACM Sigplan Notices*, Vol. 45. ACM, 305–314.

Artifact Description: MOANA: Modeling and Analyzing HPC I/O Variability

ABSTRACT

This description contains the information needed to launch some experiments of the SC'17 paper "MOANA: Modeling and Analyzing HPC I/O Variability". More precisely, we explain how to setup the experiment environment and execute the tests to reproduce the results used for model creation and verification.

A DESCRIPTION

A.1 How delivered?

(1) MOANA test scripts can be cloned or downloaded from GitHub repository using the following URL: Redacted for paper review.
(2) Dataset collected can be directly downloaded from URL: Redacted for paper review.
(3) Visualization tool can be accessed via web using URL: Redacted for paper review.

A.2 Hardware dependencies

(1) Intel (Haswell) platform capable of DVFS and has Virtualization instructions available and enabled.
(2) A dedicated 2 TB HDD.

A.3 Software dependencies

(1) Ubuntu 14.04.02.
(2) List of 2164 packages provided in Package.list file present inside GitHub repository.

B INSTALLATION

B.1 OS installation

(1) The setup is built on the 64 bit build of Ubuntu 14.04.02 desktop.
(2) The ISO is downloadable from URL:http://old-releases.ubuntu.com/releases/14.04.2/ubuntu-14.04-desktop-amd64.iso.
(3) The same version is used for both the hypervisor and VM.
(4) The hypervisor and the base OS live together.

B.2 Hypervisor package installation

Execute following commands for hypervisor (XEN) package selection and installation:

```
dpkg --get-selections > ~/setup/Package.list
sudo cp -R /etc/apt/sources.list* ~/setup/
sudo apt-key exportall > ~/setup/Repo.keys
sudo apt-key add ~/setup/Repo.keys
sudo cp -R ~/setup/sources.list* /etc/apt/
sudo apt-get update
sudo apt-get install dselect
sudo dselect update
sudo dpkg --set-selections < ~/setup/Package.list
sudo apt-get dselect-upgrade -y
```

C CONFIGURATION

C.1 VM configuration and creation

(1) Test VM is configured with 2 VCPUs, 2 GB of memory and 8/16 GB of Hard Disk space.
(2) Test VM is configured to run on a separate disk mounted as */vdrives*.
(3) *TestVM.xml*, a configuration XML is provided inside the GitHub repository.
(4) Execute following command to create VM:

```
xl create TestVM.xml
```

C.2 VM setup

(1) Each VM is setup with as few packages as necessary to run iozone in an automated framework.
(2) VM package list is provided inside repository along with hypervisor package list.
(3) Ensure ssh keys being setup in all the VMs and head nodes to allow automated login and execution of tests.

D EXPERIMENT WORKFLOW

(1) Before any execution of test scripts, copy the authentication keys to VM to ensure passwordless login.
(2) Clone the scripts and execute setup.sh to build your test case.
(3) Change directory to *results_scripts*.
(4) If you need to make a smaller execution, you can use targeted.sh.