MOANA: Modeling and Analyzing HPC I/O Variability

ABSTRACT

Exascale high-performance computing (HPC) systems are essential to address the grand challenges of scientific computing applications. Unfortunately variability threatens to make experimental systems research at scale impracticable due to an inordinate number of tunable parameters. Performance variability grows with the scale and complexity of the underlying system hardware and software. Moreover, large variabilities in HPC I/O are particularly common and formidable making it difficult and at times impossible to isolate the root cause of performance variations and optimize emergent data intensive applications. While an improved understanding of HPC I/O performance variability is essential, exascale domain scientists cannot be expected to design their own system tools and techniques to address variability. Hence, there is an urgent need to model, characterize, and analyze variability in high-performance parallel or distributed systems.

In this paper, we present MOANA, an approach for modeling and analyzing HPC I/O variability arising from interactions of different components, such as CPU frequency, number of I/O threads, I/O scheduler, etc. Our goal is to identify both root cause and magnitude of variability in our experiments and to determine whether variability is predictable to some extent. Our experiments demonstrate quantitatively that advanced non-linear statistical methods like the two studied (Linear Shepard and Multivariate Adaptive Regression Splines) are essential to accurate I/O variability prediction. To the best of our knowledge, this work is the first to attempt to predict HPC I/O variability. Our results indicate that for sequential and parallel I/O workloads some parameters (CPU freq. and file size) contribute substantially more to variability than others (I/O scheduler). We also show that while the studied techniques are useful and accurate for predicting variance (within 15%), there are tradeoffs between the input data set characteristics and predictor accuracy; fully automated hybrid non-linear statistical techniques will be necessary for use at runtime.

1 INTRODUCTION

Modern high performance computing (HPC) systems are required to meet the demands of many grand challenges for scientific computing as well as other domains such as e-commerce [27, 48]. To overcome these challenges HPC systems are both growing in scale and evolving 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 systems designs.

Highly variable observations in extremely complex systems can make it difficult or impossible to fully optimize for performance. Such variability is cited as a significant barrier to exascale computing [24, 38]. Unfortunately, variability is both ubiquitous and elusive as its causes pervade and obscure performance across the systems stack from hardware [8, 21, 31, 32] to middleware [5, 17, 28, 39, 47] to applications [18] to large-scale systems [24, 25, 37, 38, 45]. Additionally, as a number of recent reports attest [24, 38], performance variability at scale can significantly reduce performance and energy efficiency [9, 14, 18, 29].

Variability is a persistent challenge in HPC experimental systems research that 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 minimum average times. These techniques may work well on one system, but not hold true on another. They may work well initially but fail with increases in scale and complexity. Moreover, such techniques often fail 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. Furthermore, the experimental systems researcher, intent on proving out their new technology, has little incentive to stop and question their methods deeply since many are seemingly intuitive.

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.

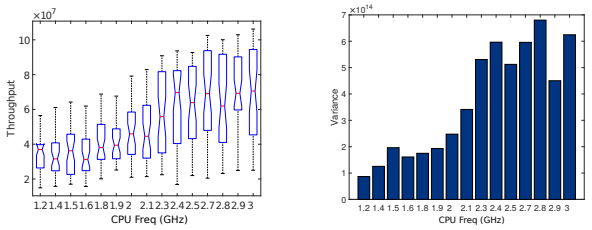
Figure 1(b) provides useful and potentially actionable variability information since it shows how much variability results from different configuration settings. However, an approach based on this type of pairwise analysis is not scalable. Table 1 shows the number of available parameters for the study 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 solution 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 40 times or more which would take two weeks. The use of brute force for validation of our techniques with longer

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

[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 have observed similar results on bare metal installs. In this paper we use VMs since it enables more robust parameterization.



(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.

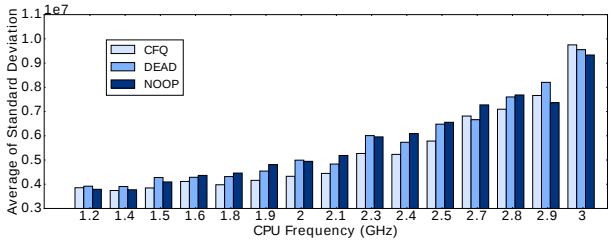


Figure 2: Non-linearity effect of CPU frequency and VM IO scheduler. We run the IOZone benchmark on a virtual operating system whose host system supports multiple CPU frequencies. The test is repeated 40 times under each single system configuration that includes the parameters: CPU frequency, I/O scheduler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

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 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 §3 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 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.

Figure 2 illustrates that changing just four parameters simultaneously leads to accuracy challenges for modeling, characterization, and analysis of variability. Commonly used statistical tools such as the analysis of variance (ANOVA) [35] focus on identifying statistically significant effects on variability. For the data set in Figure 2, the ANOVA experiments show statistically significant effects for all the parameters without any notion of causation or magnitude. This finding lacks the actionable information that could be used to steer system configurations and perhaps predict the best combination of CPU frequency and scheduler.

In this paper, we create MOdeling and ANAlysis techniques (MOANA) to characterize and predict HPC I/O variability while simultaneously maintaining scalability and capturing the non-linear 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 non-linear statistical models) to predict, characterize, and analyze the performance variability of HPC I/O workloads. Specifically, we make the following contributions in this paper:

- We study the impact of different system and application parameters on I/O performance variability.
- We develop visualization tools to explore performance variability and enable improved understanding of the effects of variability on HPC user applications.
- We identify, implement, and evaluate the effectiveness of two non-linear statistical tools for variability modeling and prediction: interpolator linear Shepard algorithm and multivariate adaptive regression spline (MARS). We refer to the resulting modeling and analysis techniques as MOANA.
- We compare the accuracy of these first-of-their-kind models of HPC I/O variability to a non-scalable brute force approach. Our techniques consistently predict the throughput variability (e.g. variance) of dozens of system and application configurations within 16% over all the predicted data points.
- We make MOANA and our datasets open source and available to the community. [3]

2 BACKGROUND AND RELATED WORK

Computational variability, or jitter, is crucial to HPC performance. HPC systems research is not immune to the current trend of criticisms of presentations of scientific results of rigor. Hoefler et al. [20] recently summarize 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 [26], and Kramer et al. explored variation in large distributed memory systems more than a decade ago [22]. 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 [29] or variations caused by the competition for resources between background processes and applications. Some have simulated these effects at scale [14], while others have proposed applications [18] or systems [9] that can account

[3] URL redacted for paper review.

	Parameters	Number of levels	Levels
Hardware	CPU Clock Frequency	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, 128, 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.

for jitter. Additionally, schedulers are often identified as causes of significant variability in HPC systems [42]. 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 [13, 41, 44], profile-based [10, 36], and simulation-based [12, 30] approaches, or a combination of these [49], 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 interactions of different components in the stack.

Introducing determinism to achieve reproducibility has also been explored using environment categorization [33], statistical modeling [40], or variance-aware algorithm design [7]. Environment categorization considers the interactions and composition of hidden factors in the system such as the open benchmarking infrastructure or performance evaluation, DataMill [15]. Our focus on modeling and managing variability and the application to experimental design complements these approaches and may influence such best practices.

Runtime predictability is a key concern in Real-Time Operating System (RTOS) design, where mission critical application deadlines have to be met with high accuracy. As such, this paper shares the goal of achieving similar runtime performance and behavior guarantees with RTOSs. However, a key differentiator in this context is that RTOSs sacrifice performance for predictability [6, 34]. As high performance is the main concern in our work, the RTOS approach cannot be simply used in our target HPC use cases.

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 [8, 11, 21, 32, 46]. The resulting issues such as leakage current, cross-wire-signaling, and real-estate budgeting, result in expected performance to vary significantly. 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 as for RTOS, the fundamental nature of our approach could be applied to the programmatic design aspects of the chip such as mapping of on-chip decisions to reduce power and the resulting affect on the probability density of the observed variations.

3 A STUDY OF VARIABILITY USING ANOVA

As mentioned, the analysis of variance (ANOVA) tool is commonly used to quantify the variability (i.e. variance) of a data set. Using ANOVA and a thoughtful experimental setup, we can identify individual parameters that contribute in a statistically significant way to variability.

Experimentally, we are interested in the combined effects of both application and system parameters. We selected the IOZone benchmark due to its large parameter space and its focus on I/O performance throughput. By starting with a highly-configurable, I/O focused performance application, we believed we could concentrate on identifying appropriate statistical models that can capture the complicated, non-linear aspects of throughput performance variability. We also chose to run our experiments in virtualized systems since we observed similar variability behavior for this benchmark in bare-metal installations and we gained a highly configurable, easily reconfigurable substrate to automate experiments and data collection that could run for months.

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 observational data with regard to CPU frequency and upon the extant literature with regard to scheduling policies. 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 intensive behaviors. Overall, we limited our selections somewhat to ensure a brute force approach was possible in less than 6 months for a full set of data runs.

Our parameters result in a total of over 95K unique configurations. For each configuration, we conduct 40 runs. Based upon our experience, this provides a 95% statistical confidence in the resulting data set. We consider this a brute-force approach since it considers all permutations of the controllable parameters. We believe this approach is impracticable for exascale systems and applications. In this work, the standard deviation of these 40 runs is used as a proxy for variability without loss of generality.

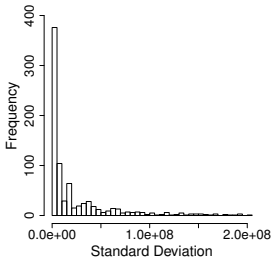


Figure 3: Distribution of measured standard deviations for 810 configurations with host I/O scheduler=CFQ, VM I/O scheduler=DEAD, mode=ReverseRead.

Figure 3 shows the distribution of the standard deviation across all possible configurations. We fix the host and VM I/O schedulers 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.

Table 2 shows the results after using ANOVA to identify system and application parameters that are statistically significant.[4] The key values in this table are: df or the degrees of freedom or number of parameters for a given type of variable in the row; P value is the

[4] If a single factor is statistically significant, it means that changing its value will affect the performance variability.

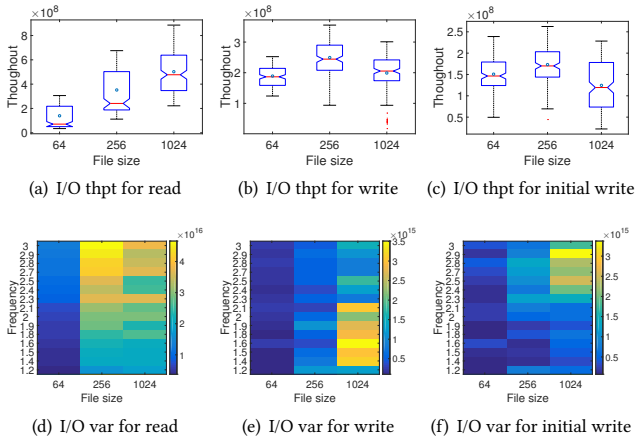
	Df	Sum of squares	Mean Squares	F value	P value	Sig.
Filesize	2	2.04E+15	1.02E+15	1.370E+02	≤2.2E-16	***
Recsize	2	3.29E+14	1.65E+14	2.208E+01	2.764E-10	***
IOsche	2	9.06E+13	4.53E+13	6.077E+00	2.307E-03	**
VMIOsche	2	2.12E+14	1.06E+14	1.423E+01	6.835E-07	***
Thread	8	3.64E+17	4.54E+16	6.093E+03	≤ 2.2E-16	***
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 influence by single or two-level parameter changes. Extremely low P values (e.g., ≤2.2e−16 for Thread) means it is almost certain the hypothesis is not true and therefore that a parameter (e.g. Thread) does in fact affect variability.

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 three different file sizes and nine 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 it is almost certain the hypothesis is not true and therefore that the parameter (e.g., Thread) does in fact affect variability. Since the P value for Thread is so low, the Sig value is set to three stars to denote the high likelihood the hypothesis is false.

Table 2 shows 13/21 entries received three stars and thus influence variability. 17/21 entries received two or more stars. This ANOVA experiment shows that nearly all of the parameters and their second factor configurations studied affect I/O variability in a statistically significant way. However, the experiment gives us little actionable information such as suggesting which configurations might result in less variability or how much an individual parameter is contributing to the overall variability. Furthermore, to achieve this level of information for the entire data set basically requires a brute force approach. Additionally, ANOVA is sensitive to the input data set and sampling a smaller portion of this data set for example might result in conflicting suggestions as to which parameters affect variability. This motivates the need for a deeper, more comprehensive analysis of variability that can help identify the root cause of variability.



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

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

Figure 4: I/O throughput 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 (c, d, and e). Record size = 32 bytes, Threads = 256.

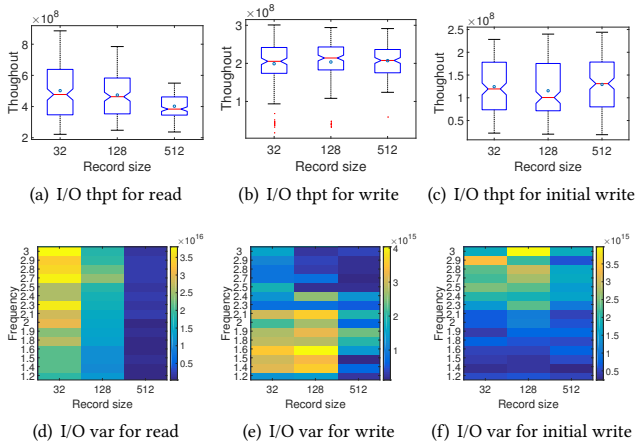
4 EXPERT-DRIVEN VARIABILITY ANALYSIS

In contrast to the ANOVA modeling approach, as advanced systems researchers we can leverage our experience to design thoughtful experiments and manually analyze I/O variability. This technique is representative of common experimental approaches to system analysis in the extant literature.

The full set of experiments (see Table 1) is too large and cumbersome to analyze using manual inspection. Once again, we use I/O throughput variance as a proxy for variability without loss of generality. For systems settings, we will consider four CPU frequencies (1.5 GHz, 2.0 GHz, 2.5 GHZ, and 3.0 GHZ) and fix the host I/O scheduler (CFQ) and the VM I/O scheduler (NOOP). For application settings, we will vary only one of file size, record size, and number of threads in isolation in our experiments.

Effect of file size. Figure 4 shows experiments designed to examine the effects of file size on I/O variability. Figure 4(a) shows raw I/O throughput increases with file size for file read operations. For file write operations (Figure 4(b)) and file initial write operations (Figure 4(c)), the I/O throughput initially increases but eventually decreases. Figure 4(d) shows that I/O variance for file read operations is highest for smaller file sizes and higher CPU frequency. For file write operations, variance is higher for medium sized files at lower CPU frequency as shown in Figure 4(e). Figure 4(f) shows that for initial write operations, larger file sizes with higher frequency exhibit the most variance.

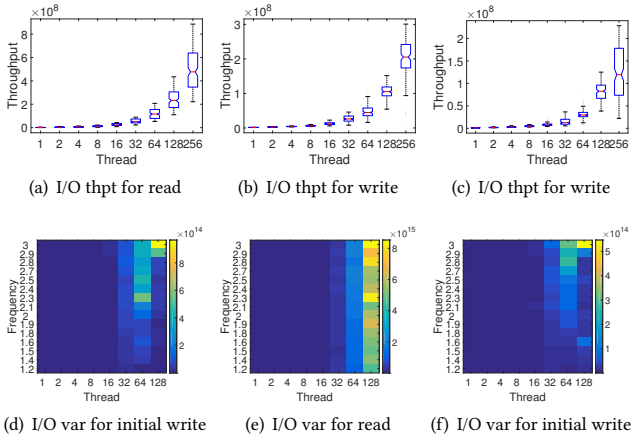
Effect of record size. Figure 4 shows experiments designed to examine the effects of record size on I/O variability. Figure 5(a) shows raw I/O throughput decreases with increases in record size for file read operations. For file write operations (Figure 5(b)) and file initial write operations (Figure 5(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



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

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

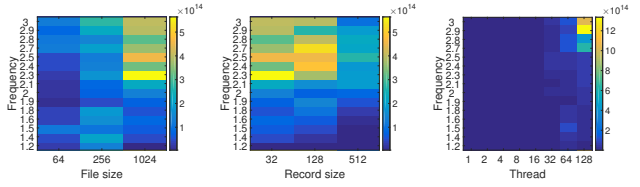
Figure 5: I/O throughput as a function of record size for three different I/O op modes (a, b, and c). Heat map of I/O throughput variance (y-axis-right) as a function of CPU frequency (y-axis-left) and record size (x-axis) for three different I/O op modes (c, d, and e). File size = 1024 Kbytes, Threads = 256.



(a) I/O thpt for read (b) I/O thpt for write (c) I/O thpt for write

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

Figure 6: I/O throughput 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 (c, d, and e). File size = 1024 Kbytes, Record size = 32 bytes.



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

Figure 7: Heat map of change (a and b) in I/O throughput variance (y-axis-right) from 256 threads (Figure 4(e) and Figure 5(e)) down to 64 threads as a function of CPU frequency (y-axis-left) and file size (x-axis) for three different I/O op modes (c, d, and e). Heat map of change (c) in I/O throughput variance (y-axis-right) from 1024 Kbytes file size (Figure 6(e)) down to 64 Kbytes.

the highest variation for write operations. Medium record size and higher frequency give the highest variation for initial write operations. See Figures 5(d), 5(e), and 5(f) respectively.

Effect of number of threads. Figure 6 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 6(a), 6(b), and 6(c). I/O throughput variance also increases with the number of threads; highest for file write operations (Figure 6(d)) and file initial write operations (Figure 6(f)) at the highest frequency. File read operations have high variance in the lower frequency range as well (Figure 6(e)).

Key Insight These experiments highlight the usefulness of expert-driven inspection and analysis. However, these results also show that clear cause and effect patterns are not obvious. This makes generalizing beyond the measured results difficult. For example, Figure 7 shows small experimental changes can lead to unexpected results. We repeat the file size and record size experiments with a decrease in the number of threads from 256 to 64. The resulting heat map flips leading to almost the opposite trends in variance with regard to frequency. In the third case, where number of threads is changing, we reduce the file size to gauge the effects on variance. While variance was tied to CPU frequency, now variance is high only for higher CPU frequency.

Effect of Compound Factors To further illustrate the difficulty of the expert-driven approach, consider the effects of multivariate changes on variability. Figure 8 shows the per thread I/O throughput for an increasing number of threads. 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.

There is a stark contrast between these two columns of subfigures. There seems no discernible pattern to the resulting change in the variance. This again highlights the non-linear aspects to modeling variability with a large number of variables. And in combination with the results from the ANOVA model analysis, it is apparent that models that account for non-linearity in the cause-effect relationships that are also scalable will be essential to modeling and analyzing variability at exascale.

5 MOANA PREDICTION MODELS

In this section, we first describe our modeling and analysis approach (MOANA); namely two advanced statistical models that we use to predict the I/O performance variability—modified linear Shepard Algorithm, and Multivariate Adaptive Regression Splines (MARS). The strength of these algorithms lie in their ability to scale up to high dimensions, as well as uncovering connections between complex variables and their interactions, and capturing complicated local nonlinear relationship. Thus, they are strong candidates for uncovering interactions between factors that affect complex systems such as I/O. Next, we present the process of model

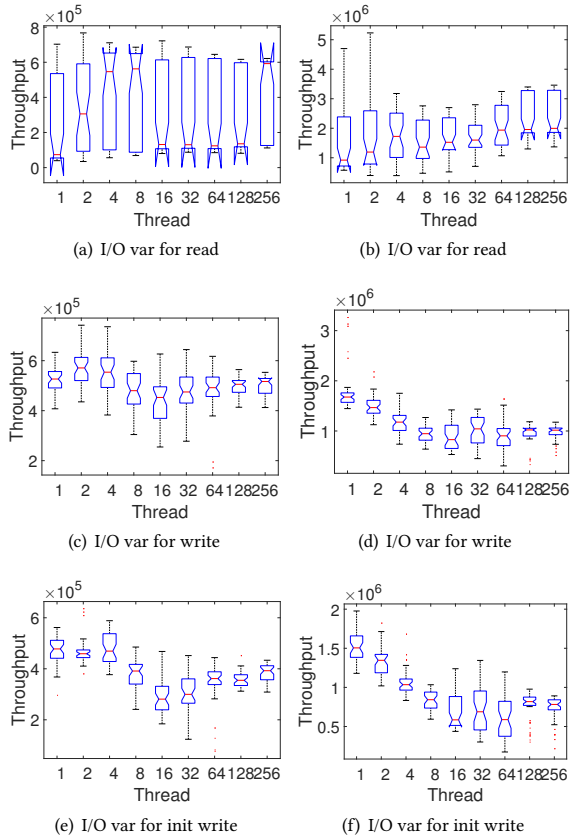


Figure 8: Each subfigure shows the per thread I/O throughput as the number of threads increases. All of the subfigures on the left (a, c, and e) use 1.2 GHz for CPU frequency and 64 Kbyte file size. All of the subfigures on the right (b, d, and f) use 3.0 GHz and 1024 Kbyte file size. Other fixed parameters: host scheduler=CFQ, VM I/O scheduler = NOOP, record size = 32 Kbytes.

building and training. In particular, we use the concept of variability map to describe and predict variability. Let x be a vector of parameters. For a given configuration of x, the variability map is a function $f(x)$ that gives the variability measure (i.e., standard deviation in our context) at x. With the variability map $f(x)$ trained from experimental data, one can use it to characterize and to predict variability for any given configuration x.

5.1 Modified Linear Shepard Algorithm

The linear Shepard algorithm is derived from the modified Shepard algorithm [43]. The modified Shepard algorithm approximate $f(x)$ by $\tilde{f}(x)$ at point x, which is obtained as,

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

Here,

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

The function $P_k(x)$ is the local linear approximation function of around the k^{th} data point. For the linear Shepard algorithm, we use the local weighted least-squares fitting to obtain the $P_k(x)$. That is, the weight is defined as,

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

Let $N_p = \min\{n, 3m/2\}$ and $D = \max_{i,j} \|x^{(i)} - x^{(j)}\|_2$. The values of $R_p^{(k)}$ and $R_\omega^{(k)}$ are specified as,

$$R^{(k)} = \min\left\{ r \,\middle|\, \overline{B(x^{(k)}, r)} \text{ contains at least } N_p \text{ points} \right\},$$
$$\text{where } \overline{B(x^{(k)}, r)} = \left\{ x \,\middle|\, \|x^{(k)} - x\|_2 \le r \right\},$$
$$R_\omega^{(k)} = \min\left\{ \frac{D}{2}, R^{(k)} \right\}, \quad \text{and } R_p^{(k)} = 1.1 R^{(k)}.$$

Then, one can compute $R_p^{(k)}$ and w_{ij}. The function $P_k(x)$ has the following form,

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

Let $S = \{i_1, i_2, i_3, \cdots, i_{N_{p-1}}\}$ 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 $R^{(k)}$. Let A be an $(N_p - 1) \times m$ matrix with the j^{th} row to be

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

and b be an $(N_p - 1) \times 1$ vector with the j^{th} element to be

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

The coefficients $a^{(k)}$ of $P_k(x)$ are obtained as the minimum norm solution of the least-squares problem,

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

Where E is the real number set.

5.2 Multivariate Adaptive Regression Splines (MARS)

The collection of basis functions of the MARS is defined as,

$$C = \left\{ \left(X_j - t\right)_+, \left(t - X_j\right)_+ \right\}_{t \in \{x_{1j}, x_{2j}, \ldots, x_{Nj}\},\ j = 1, 2, \ldots, p}.$$

We use a procedure that is similar to the step-wise regression by using the functions generated in set C as the candidate predictors. The regression model has the following form,

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X). \tag{1}$$

Here, $H_m(X)$ is a function in C or a product of two or more functions in C. We start with constant function $H_0(X) = 1$ in the model. At each step we add a term of the following form which minimizes the square of error most to the model $\mathcal{M}$,

$$\hat{\beta}_{M+1} h_l(X) \cdot (X_j - t)_+ + \hat{\beta}_{M+2} h_l(X) \cdot (t - X_j)_+, \; h_l \in \mathcal{M}, \quad (2)$$

Where $h_l(X)$ is one item in current model $\mathcal{M}$. $\left(X_j - t\right)_+$ and $\left(t - X_j\right)_+$ are in C. At the end of the step-wise procedure, we have a large model of the form (1), which may over-fit the data. Thus, a backward procedure is used. To determine the size of the best model, we use the generalized cross-validation (GCV) as the criterion, which is defined as

$$\mathrm{GCV}(\lambda) = \frac{\sum_{i=1}^{N} (y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2}.$$

Here, λ is the number of terms in the current model and $M(\lambda)$ is the effective number of parameters in the model. For more details of GCV see [16, 19]. We use the λ that attains the least GCV(λ). We continue the backward process until the number of terms in the model is λ.

6 VARIANCE MODELING

To compare the performance of algorithms, we use a leave-one-out method and calculate the sum of squares of errors, with relative error, as the criterion.

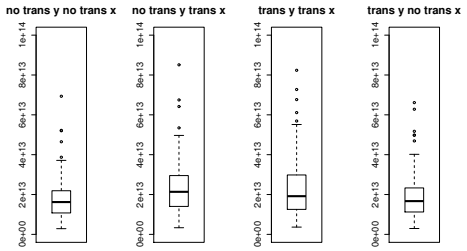
To precisely describe the error calculation, consider the 2D linear Shepard algorithm. From Table 3, we have $13 \times 9 \times 6 = 702$ datasets in total. For $i = 1, 2, \ldots, 702$, we represent the i^{th} dataset with $\left[X^{2d,\,i},\; Y^{2D,\,i}\right]$ where $X^{2d,\,i}$ is a 135×2 matrix of configuration and $Y^{2D,\,i}$ is the 135-dimensional vector of corresponding standard deviations. Compute the linear Shepard interpolant to this data with the j^{th} row of $\left[X^{2d,\,i},\; Y^{2D,\,i}\right]_j$ deleted, and let $\hat{y}_j^{2D,\,i}$ be the value of this interpolant at the deleted point $X_j^{2D,\,i}$. Let $\overline{Y^{2D,\,i}}$ denote the mean of the vector of standard deviation $Y^{2D,\,i}$. Then the mean squares error (MSE) mse_i and relative error are defined by

$$mse_i = \frac{1}{135} \sum_{j=1}^{135} (y_j^{2D,\,i} - \hat{y}_j^{2D,\,i})^2, \; i = 1,\, 2,\, 3,\, \cdots,\, 702. \quad (3)$$

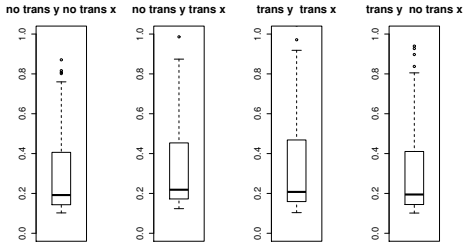
$$r_i = \frac{mse_i}{Y^{2D,\,i}}, \; i = 1,\, 2,\, 3,\, \cdots,\, 702. \quad (4)$$

For practical reasons, the data is transformed before the computations are done. For the 2D linear Shepard interpolation, only thread and frequency are varied. I/O schedule, virtual I/O schedule, file size and record size are fixed. Because of the exponential spread of the thread values (Table 4), $\log_2$ is taken of the thread values, and then these $\log_2$ values are normalized to [0, 1]. The frequency is normalized to [0, 1]. The [0, 1] normalization is done per dataset.

Note that the linear Shepard interpolant can predict a (meaningless) negative standard deviation. Furthermore, since $\log_2$ transforming a linear relationship changes it to a nonlinear relationship, the value of $\log_2$ transformation is moot. All four combinations of $\log_2$ transforming the thread values and the standard deviation values were tried.



(a) mse_i box plots for 2D linear Shepard interpolation.



(b) Relative error box plots.

Figure 9: Box plots for four classes of pre-processing, from left to right: (no transforming on Y and no transforming on X), (no transforming on Y and transforming on X), (transforming on Y and transforming on X) and (transforming on Y and no transforming on X). The no pre-processing method provides the best result.

For the 4D linear Shepard interpolation and MARS least-squares fitting, we do not use any pre-processing of the data. Table 5 gives the training information for each method.

Method	Changing variables in each dataset	# of points in each training dataset	# of training datasets
2D Shepard	Thread, frequency	9 × 15=135	13 × 9 × 6
4D Shepard	File size, record size, thread, frequency	6 × 9 × 15=810	13 × 9
MARS	File size, record size, thread, frequency	6 × 9 × 15=810	13 × 9

Table 3: Training dataset summary for the three considered methods.

Figure 9(a) shows the box plots for 2D linear Shepard interpolation of mse_i based on different data pre-processing methods. We see that transforming on X or Y do make a difference in the accuracy of the model. The model with no data pre-processing seems to work best. The box plots for the relative error are shown in Figure 9(b), again no pre-processing provides for the best outcome. The summary statistics are shown in Figure 10, where dark green

Model	X dimension	Transform on Y	Transform on X	Min	25% quantile	Median	Mean	75% quantile	Max	Sd
Shepard	2	no	no	0.1023	0.1433	0.1918	0.2913	0.4062	0.8707	0.2051586
	2	no	yes	0.1235	0.1723	0.2185	0.3248	0.4536	0.986	0.2204899
	2	yes	yes	0.1036	0.1591	0.2076	0.3384	0.4684	1.312	0.2708023
	2	yes	no	0.1014	0.1443	0.1946	0.3071	0.4105	1.106	0.235266
	4	no	no	0.08566	0.1187	0.1565	0.2382	0.3379	0.7044	0.1673215
MARS	4	no	no	0.1102	0.1361	0.1609	0.2506	0.404	0.5858	0.1465333

Figure 10: Relative error summary statistics of the studied methods.

means a smaller quantile. We see that no transformation is about 10% more accurate than the other three pre-processing methods.

The average relative error is 29%. The 2D linear Shepard using only thread and frequency as the training data. The information of file size and record size is not used. Treating file size and record size continuous gives a much bigger training set with 6 (file size and record size) ×9 (thread) ×15 (frequency) = 810 points. We now have 117 datasets to perform leave-one-out testing, giving MSE and relative error for a box plot. We already know that transformation of X or Y will not improve the accuracy, so no transformation is used.

Note that the linear Shepard interpolant, compared with linear regression, does not provide a simple analytic model of the data. We also tried multivariate adaptive regression splines (MARS) on the same interpolant 4-dimensional training set for the 4-dimensional linear Shepard algorithm. No pre-processing was applied in this setting as well.

The summary statistics for the 4-dimensional Shepard interpolant and MARS least-squares fit are shown in Figure 10. The 4D linear Shepard interpolant is the best model with the least relative error, but MARS has a smaller maximum error rate. Most error rates are around 20%. We also display the relative error box plots for the 2D Shepard, 4D Shepard and MARS models in figure 11.

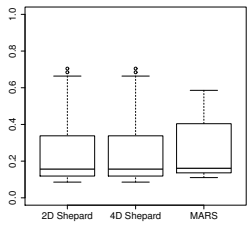


Figure 11: Relative error box plots for 2D, 4D Shepard interpolant and MARS fit. More dark green means smaller error rate quantile.

Comparison of 4D Shepard and MARS. Comparing the values from a leave-one-out methodology for an interpolator algorithm and a least squares algorithm, for any sparse data, is misleading. Other than that both algorithms do reasonably well. Moreover, a general conclusion can be drawn from Figure 12. When the standard deviation is small, linear Shepard algorithm works better. When the standard deviation is large, MARS has much better accuracy. The linear Shepard algorithm does local interpolation, whereas MARS offers global fitting of the data.

7 MODEL VALIDATION

In this section, we evaluate the accuracy of our linear Shepard and MARS model in predicting I/O variability. Specifically, we perform additional experiments with new configurations and compare the

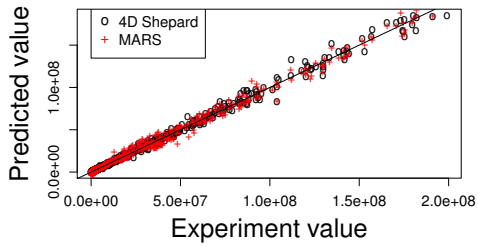


Figure 12: Scatter plot of fitting values and true values using data in Figure 3. The X-axis shows the predicted standard deviation from our two models using the former data, while the Y-axis shows the empirical standard deviation of 40 runs.

results with predicted values calculated using the models. We set CPU frequency as 2.5 GHz and the number of threads as 128. We validate six new combinations of file size and record size with all possible combinations of I/O scheduler, VM I/O scheduler, and I/O operation modes. More configuration details of our experimentation space are shown in Table 4.

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

Table 4: New configurations used for model validation.

First, we compare the accuracy of our two models in predicting I/O variability. With all new configurations, we have $3 \times 3 \times 13 \times 6 = 702$ prediction points in total. This is depicted as a scatter plot in Figure 13. The X-axis shows the predicted standard deviation from our two models trained by the former data, while the Y-axis shows the empirical standard deviation of 40 runs of new configuration. The $y = x$ diagonal line represents ideal prediction. The MARS points (cross point type in Figure 13) are clustered closer to the ideal diagonal compared to the interpolator linear Shepard ones

	MSE
Linear Shepard	2.881224×10^{14}
MARS	1.372443×10^{14}

Table 5: MSE table.

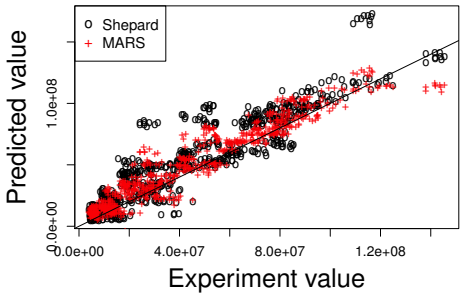


Figure 13: Comparison of the two proposed prediction models on new configurations. The X-axis shows the predicted standard deviation from our two models using the former data, while the Y-axis shows the empirical standard deviation of 40 runs.

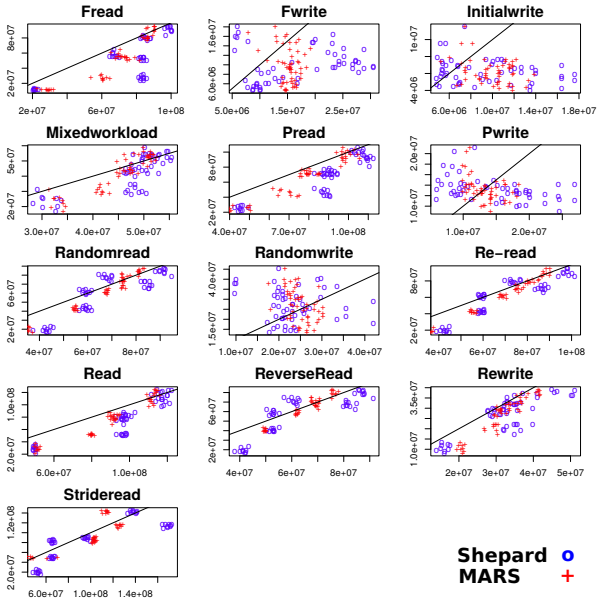


Figure 14: Prediction scatter plots of different I/O operation modes. Points in the same plots have same modes. The Y-axis is the empirical standard deviation observed from the 40 runs. The X-axis is the predicted standard deviation obtained from our two models.

(circle point type), which indicates that MARS achieves higher accuracy in predicting the I/O variability. The observation is consistent with the conclusion of Figure 10: MARS's most relative errors are smaller than linear Shepard algorithm's. Similar results are also shown in Table 5. As an ongoing effort, we are still exploring new models for better results of prediction.

Next, we evaluate the impact of different configuration parameters on the accuracy of our prediction models. Figure 14 plots the results of varying I/O operation modes. The predictions are relatively less accurate for both models for Fwrite, InitialWrite, Pwrite, and RandomWrite. Both models give good prediction for

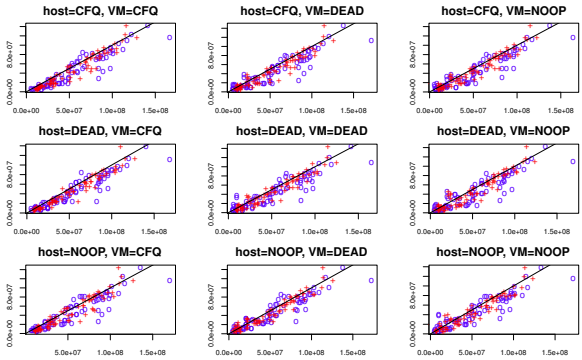


Figure 15: Prediction scatter plots of different host and VM I/O scheduler combinations. Points in the same plots have fixed other parameters.

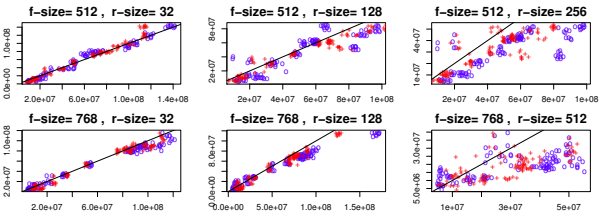


Figure 16: Prediction scatter plots of different file and record size combinations. Points in the same plots have fixed other parameters.

read-related I/O operations (e.g. Randomread, Re-read, Read, ReverseRead, StrideRead). MARS model further outperforms Linear Shepard model for the read-related operations. This is because: (1) the operating system file system's page cache is trying to commit many updates to disk, and (2) interactions of nested (host and VM) file systems impose unexpected performance implications [23]. Both models work reasonably well in capturing the variability trend, while MARS is slightly better than linear Shepard. Figure 15 shows the results when fixing both the host and VM I/O scheduler (each plot with a fixed combination of host and VM I/O schedulers). We observe that, across all possible I/O scheduler combinations (all 9 plots), changing other parameters (file/record size and I/O operation modes) do not cause discernible influence on the models' predictability. This demonstrates the superiority of our non-linear models in prediction accuracy.

Figure 16 shows the models' predictability under new file size and record size combination. For fixed file size, prediction accuracy becomes worse as the record size increases. This is again because of unexpected performance implications caused by nested file system interactions. We also observed that prediction accuracy is good for both of the two file sizes when the record size is set to be relatively small (e.g. 32 and 128). However, if the record size is large enough, effects of other system parameters not in our current scope, such as buffer cache, may come into play. From the model perspective, those two configurations are out of the range of our sample data used for training the models. We are actually doing extrapolation for the prediction of these two configurations. This is another reason of the relatively bad accuracy. To improve the

accuracy of prediction for larger record sizes, we can retrain our model using additional system parameters and sample data points. Such retraining may be cumbersome, but the effort is amortized over the longer term benefits the approach can help realize.

Overall, our two proposed models work well in predicting the I/O variability, and can be used to quantify the runtime performance variance.

8 CONCLUSIONS

In this paper, we investigated the impact of software and hardware parameters on HPC I/O throughput. We demonstrated quantitatively the limitations of classic statistical analysis tools such as ANOVA and expert-driven approaches common in the HPC community. We proposed a modeling and analysis approach (MOANA) that compared and contrasted the use of two non-linear statistical techniques to develop accurate and scalable models of I/O variability suitable for exascale systems and for potential use by broad HPC audiences. The resulting models were accurate to within 16% of the impracticable brute force approach on average while providing previously unavailable functionality including creation of actionable information that can be integrated in runtime tools. Our future plans include addressing some existing accuracy limitations through an integrated hybrid modelling approach that leverages the strengths of the two techniques studied. We are also exploring other non-linear modeling techniques and integration of our techniques in runtime systems to automate management of variability.

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] Siro Arthur, Carsten Emde, and Nicholas Mc Guire. 2007. Assessment of the realtime preemption patches (RT-Preempt) and their impact on the general purpose performance of the system. In *Proceedings of the 9th Real-Time Linux Workshop*.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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*.

[16] Jerome H. Friedman. 1991. Multivariate Adaptive Regression Splines. *Ann. Statist.* 19, 1 (03 1991), 1–67. DOI: http://dx.doi.org/10.1214/aos/1176347963

[17] 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*.

[18] 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.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA. 321–329 pages.

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

[21] 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.

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

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

[24] 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).

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

[26] 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.

[27] 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).

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

[29] 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*.

[30] 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.

[31] 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*.

[32] 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.

[33] 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.

[34] Steven Rostedt and Darren V Hart. 2007. Internals of the RT Patch. In *Proceedings of the Linux symposium*, Vol. 2. Citeseer, 161–172.

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

[36] 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.

[37] 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*.

[38] 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*.

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

[40] 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.*

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

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

[43] 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.

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

[45] 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).

[46] 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.*

[47] 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.

[48] 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).

[49] 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.