Lightweight Measurement and Analysis of HPC Performance Variability

Jered Dominguez-Trujillo, Keira Haskins, Soheila Jafari Khouzani, Christopher Leap, Sahba Tashakkori, Quincy Wofford, Trilce Estrada, and Patrick G. Bridges
Computer Science Department
University of New Mexico
{jereddt,wasp,sjafarikhouzani,cleap,stashakkori,wofford,trilce,patrickb}@unm.edu

Patrick M. Widener
Center for Computing Research
Sandia National Laboratories
pwidene@sandia.gov

***Abstract*—Performance variation deriving from hardware and software sources is common in modern scientific and data-intensive computing systems, and synchronization in parallel and distributed programs often exacerbates their impacts at scale. The decentralized and emergent effects of such variation are, unfortunately, also difficult to systematically measure, analyze, and predict; modeling assumptions which are stringent enough to make analysis tractable frequently cannot be guaranteed at meaningful application scales, and longitudinal methods at such scales can require the capture and manipulation of impractically large amounts of data. This paper describes a new, scalable, and statistically robust approach for effective modeling, measurement, and analysis of large-scale performance variation in HPC systems. Our approach avoids the need to reason about complex distributions of runtimes among large numbers of individual application processes by focusing instead on the maximum length of distributed workload intervals. We describe this approach and its implementation in MPI which makes it applicable to a diverse set of HPC workloads. We also present evaluations of these techniques for quantifying and predicting performance variation carried out on large-scale computing systems, and discuss the strengths and limitations of the underlying modeling assumptions.**

I. Introduction

Large-scale cloud and HPC systems and applications regularly experience performance variation due to dynamic hardware and software actions, configurations, and resource allocations common in modern systems. On dedicated nodes, sources of this variation include operating system management activities [12], processor clock speed changes to control system power and temperature [1], competition for memory bandwidth between processors and network cards [13], inconsistent system cooling patterns [22], changes in network links, switch capacities, and routes [6], and inherent variation or non-determinism in application behavior. On systems where applications share node-level resources, something common in cloud environments and increasingly prevalent in HPC systems, resource competition from other applications can also lead to performance variation [11], [18], [23], [28].

Accurately measuring, predicting, or otherwise assessing performance variation in large-scale system performance to vary is extremely difficult. First, because node and system behavior can widely vary over time [1], [16], any approach that seeks to quantify or predict variations in system performance must collect large numbers of performance samples from many nodes over significant time-scales, potentially requiring collection, storage, and analysis of vast amounts of data. In addition, any approach to quantifying variation in expected application or system performance assumes an underlying model of both application/system performance and necessarily makes simplifying assumptions about its scope and behavior. Variation in real-world parallel and distributed systems exhibits complex behaviors which can invalidate many common modeling and analysis assumptions (e.g. continuity, temporal invariance, independence, identical distribution) [14].

As a result, there is currently no general approach to measure, predict, or otherwise assess performance variation in real-world large-scale computing systems. Multiple authors have measured different performance variation sources on specific nodes [1] or systems [12], but these approaches do not generalize beyond a single system or predict how performance variation changes with application or system scale. Similarly, application performance prediction approaches predict only *average* performance [20] and do not attempt to predict the variation in performance common in real-world systems.

Because users, system architects, and system tools cannot accurately measure or predict performance variation, they must make pessimistic resource allocation and system configuration decisions. Users, for example, generally request maximum job lengths from the system scheduler because the cost of underestimating application runtime is job termination; this reduces the ability of the system scheduler to effectively allocate system resources. Similarly, users or system architects may disable dynamic resource allocation systems and policies (e.g. OS VM, thread, process, or memory migration; automatic processor power management) to achieve predictability at the cost of performance. In addition, it also makes truly anomalous system or application behavior difficult to detect.

In this paper, we describe a new approach to measuring, analyzing, and predicting performance variation in large-scale computing systems which combines a focus on the *maximum* length of distributed workload intervals with parametric and non-parametric statistical bootstrapping techniques. Compared to other approaches, focusing on estimating variation in maxima significantly simplifies measurement and analysis challenges because it avoids the need to reason about the

978-1-6654-2265-9/20/$31.00 ©2020 IEEE
DOI 10.1109/PMBS51919.2020.00011

complex distribution of runtimes on individual nodes. Relevant background material on the statistics of maxima are provided in Section II.

Our overall goal is to enable system and application architects to accurately assess and predict performance variation in large-scale systems. Overall, this paper describes the following contributions toward achieving this goal:

- A new approach to modeling, measuring, and analyzing performance and performance variation in large-scale systems based on bootstrapping either parametric or empirical maxima distributions (Section III);
- An MPI implementation of the measurement approach underlying this model that enables its application to a range of HPC workloads and system (Section IV); and
- An evaluation of these techniques for quantifying and predicting performance variation in large-scale systems, including an analysis of the strengths and limitations of the modeling assumptions underlying the different analysis techniques (Sections V and VI).

In addition, the paper also discusses related research in characterizing performance variation in large-scale computing systems (Section VII), and then directions for future research before concluding (Section VIII).

II. Background: Mathematics of Maximas and Bootstrapping

Our methodology uses a combination of extreme value theory and statistical bootstrapping to ensure two things, respectively: 1) that we avoid making assumptions about the workload as a whole (i.e., specifically about the type of distribution governing the different workloads, as intra-node variability can and will impact those assumptions), and 2) that we can achieve robust conclusions with a relatively small number of samples. The various methods that we use for this work make use of these two aspects of extreme value theory and statistical bootstrapping in different ways, as we explain in detail in Section III. In the remainder of this section we provide background on the different building blocks that provide the mathematical basis for our approach.

A. Statistical Distribution of Maximas

The most commonly used statistical tool for characterizing distributions of maxima is the Generalized Extreme Value (GEV) distribution. The GEV distribution is a set of probability distributions comprising Gumbel, Fréchet, and Weibull, which is characterized by a set of underlying i.i.d. random variables. The GEV approximates the distribution of the maximum value of a set of independent and identically distributed random variables. The cumulative distribution function (CDF) of the GEV distribution is:

$$F\left(x|\kappa,\xi,\alpha\right) = e^{-\left(1+\kappa\left(\frac{x-\xi}{\alpha}\right)\right)^{\frac{-1}{\kappa}}} \quad (1)$$

where κ is shape, ξ is location, and α is a scale parameter. When the shape parameter κ is 0, the extreme value distribution is type I or Gumbel. When ξ is greater than 0, it is type II or Fréchet, and if κ is less than 0, it is type III or Weibull.

B. Estimate of Maxima Distributions

Several methods exist for estimating the parameters for the GEV distribution. We compared three different methods for our work: maximum likelihood estimators (MLE), probability weighted moments (PWM), and the method of moments (MOM). Maximum likelihood estimators are commonly used in parametric distribution fitting, such as in the Python SciPy Stats library [24]. However, multipe studies have found that that MLE is unstable with smaller sample size ($15 < n < 100$) [15] when compared to PWM. Madsen et al. also found that MOM has lower root mean square error when dealing with smaller sample sizes [19]. We have found that MOM and PWM are more reliable than MLE for fitting GEV distribution parameters in our experiments.

The method of moments [19] for estimating the parameters of the GEV distribution is as follows: given the sample mean, standard deviation, and skewness: $\hat{\mu}$, $\hat{\sigma}$, and $\hat{\gamma}$, it estimates empirically the shape parameter $\hat{\kappa}$ by minimizing the following equation

$$\hat{\gamma} = sign(\hat{\kappa}) * \frac{-\Gamma(1+3\hat{\kappa}) + 3\Gamma(1+\hat{\kappa})\Gamma(1+2\hat{\kappa}) - 2[\Gamma(1+\hat{\kappa})]^3}{\{\Gamma(1+2\hat{\kappa}) - [\Gamma(1+\hat{\kappa})]^2\}^{3/2}}$$

with respect to $\hat{\kappa}$ using the Nelder-Mead method and an initial guess of $\hat{\kappa} = 0$ and where $\Gamma(n) = (n-1)!$. After approximating the shape, the scale $\hat{\alpha}$ and location $\hat{\xi}$ are estimated as:

$$\hat{\alpha} = \frac{\hat{\sigma}|\hat{\kappa}|}{\{\Gamma(1+2\hat{\kappa}) - [\Gamma(1+\hat{\kappa})]^2\}^{3/2}}$$

$$\hat{\xi} = \hat{\mu} - \frac{\hat{\alpha}}{\hat{\kappa}}\{1 - \Gamma(1+\hat{\kappa})\}$$

The method of probability weighted moments [15] considers the first three moments, mean, variance, and skewness: $\hat{\mu}$, $\hat{\sigma^2}$, and $\hat{\gamma}$ respectively. It computes the shape parameter $\hat{\kappa} = 7.8590c + 2.9554c^2$, where $c = \frac{2\hat{\sigma^2}-\hat{\mu}}{3\hat{\gamma}-\hat{\mu}} - \frac{log2}{log3}$. The scale parameter $\hat{\alpha}$, and the location parameter $\hat{\xi}$ are calculated as:

$$\hat{\alpha} = \frac{(2\hat{\sigma^2} - \hat{\mu})\hat{\kappa}}{\Gamma(1+\hat{\kappa})(1-2^{-\kappa})}$$

$$\hat{\xi} = \hat{\mu} + \hat{\alpha}\frac{\Gamma(1+\hat{\kappa}) - 1}{\hat{\kappa}}$$

C. Bootstrapping Maxima Confidence Intervals

Given a sufficiently large sample, and the assumption that the empirical distribution of the observed data is representative of the true population distribution, bootstrapping randomly resamples with replacement the empirical distribution. Although this process yields only an approximate estimate, it has been proven [7], [10] to produce more accurate population parameter estimates than if these parameters were calculated based on an initial incorrect assumption of a specific distribution.

The parametric version of bootstrapping randomly generates samples from a parametric distribution fitted to the data (e.g., a GEV distribution). The non-parametric bootstrap method performs resampling with replacement from the given sample

and calculates the required statistic from a large number of repeated samples. In both cases, standard errors and confidence intervals can be calculated. Parametric and non-parametric approaches have been compared as to their performance in estimating uncertainties in extreme-value models. The non-parametric bootstrap leads to confidence intervals that are too narrow and underestimate the real uncertainties involved in the frequency models [17], unless a large number of samples are used.

III. Maxima-based Performance Variation Assessment

In addition to being a source of unpredictable behavior, performance variance inherently limits the performance of large-scale applications. This happens because periodic application-wide communication in distributed applications requires all processes to wait for the slowest process before any process can proceed. As a result, even variability which manifests in only a single process eventually affects every application process. In traditional bulk-synchronous HPC applications, for example, this frequently happens through explicit MPI collective communication. Data-centric compute systems often synchronize implicitly during global compute phases (e.g. Hadoop reductions or Spark shuffle transformations) as well. Our methodology focuses on efficiently and accurately modeling this type of variability.

Our metholodology has two goals:

1) To reduce the amount of data that needs to be collected to accurately measure system and application performance characteristics
2) To accurately quantify performance variation in large-scale systems and applications

In the remainder of this section, we discuss the basic performance quantification and prediction model (and its assumptions) on which we base our methodology, how we leverage this model to measure system performance, and the bootstrapping methodology we use to quantify variation in measured and predicted workload performance.

A. Modeling Approach

Our approach measures, models, and analyzes the distribution of the *maximum* length of a fixed distributed compute/communication workload at a given system scale. Such a workload could be a simple computational kernel fenced by synchronization operations, including a complex application workload delineated with `MPI_Init`/`MPI_Finalize`, or a Hadoop MapReduce kernel.

We then model workload performance as a generally-distributed random variable of which a sample is the length of one execution of this workload. We assume that this random variable is the maximum of N random variables, each of which describes the length of program runtime on an individual node. We further assume that these distributions are stationary (not time-varying). We do *not*, however assume that the distributions of individual node runtimes are independent or identical, or that either the node or overall system runtime distributions are continuous.

This model allows us to measure only either (1) the time between the end of successive synchronizing collectives, which is useful if we are analyzing only the maximas of workloads such in non-parametric bootstrap described below, or (2) distribution of individual node inter-collective times such as in the inter-node and intra-node parametric bootstraps described below. In either case the amount of data needed to quantify system variation is significantly less than that required by approaches that rely on fine-grain tracing of all applications.

B. Analytical Methods

In this section we describe how we use statistical bootstrapping to augment an existing performance prediction technique for quantifying expected changes in workload variation when the size of the system or application under examination is increased. More specifically, we use non-parametric and parametric bootstrapping approaches.

The non-parametric bootstrap makes minimal assumptions about the distribution of maxima on nodes, focusing on the empirical distribution of maxima. In contrast, the parametric bootstrap assumes that either individual nodes or sets of nodes have independent, identically-distributed (i.i.d.) workload lengths. The non-parametric bootstrap's weaker assumptions, however, require a large number of global maximum samples to be effective, as described in Section II, while some parametric bootstrap approaches require performance samples from individual nodes to avoid assuming that all workload samples are i.i.d.

1) Non-parametric bootstrapping: For the non-parametric version of our approach, we do not perform explicit model fitting. Instead, we re-sample maxima data to simulate the effects of scaling-up the workload. In particular, we implement the non-parametric bootstrapping by randomly resampling the original maxima data k times with replacement and taking the maximum of these k values as a new sample on a system k times larger. We repeat this process n times to generate bootstrap empirical samples for the larger system being modeled. The 95% confidence interval is then directly computed from the resulting set of maxima scaled up via resampling.

2) Parametric bootstrapping: In the parametric approach, we fit a GEV distribution to a collection of maxima, perform a prediction of expected growth, and calculate its confidence intervals. To perform the GEV fitting, we use the two methods described in Section II: the method of probability weighted moments (PWM) and the method of moments (MOM). Although mixed methods combining MLE and MOM have been explored before to fit GEV [3], PWM has shown greater stability compared to MLE even though it can still break down in the presence of outliers [9]; thus, the need to combine them. By using both, PWM and MOM, in conjunction with bootstrapping, we seek to leverage their strengths while reducing the likelihood of instability in the fitting. After obtaining

a good fitting, we can use the Expected Mean Maximum Approximation [26] (EMMA) technique, which can be used to predict the performance correctly when the distribution of maxima are known a *priori*. The EMMA function estimates the expected value of the maximum of *n* observations of a set of i.d.d random variables X_i with distribution *F* as:

$$E\left(max_{i=1}^{m} X_i\right) \approx F^{-1}\left(P\right)$$

where $P \approx 0.570376002^{\frac{1}{m}}$, *m* is the projection scale, and *F* is the GEV distribution. If we use EMMA for n iterations, we will have a distribution of maximum values in the extrapolated scale. As we are interested in exposing the variability that happens naturally within a node, and across the system, we systematically bootstrap in two ways:

- **Inter-node Parametric**, where we use one node at a time to generate the 50 bootstrap replicas and repeat for the total number of nodes in our sample (e.g., for 8 nodes we generate $8 \times 50 = 400$ replicas per sample workload).
- **Intra-node Parametric**, where we select one rank for each node to generate the 50 replicas, and repeat for the total number of ranks per node (e.g., for 32 ranks per node we generate $32 \times 50 = 1600$ replicas per sample workload.

To obtain the 95% bootstrap confidence interval we extract the $2.5th$ and $97.5th$ percentiles of the distribution of EMMA projected runtimes. For a sample size n and a confidence interval ci, the position in the ordered collection of resampled projections corresponding to the bounds of the empirical bootstrap confidence interval are given by $[n(1-ci)/2, n(1+ci)/2]$. As for the number of bootstrap replicas that are needed in practice to compute a stable 95% confidence interval, Efron et al. [10] suggested 200 replicas for calculating the bootstrap standard error but 1000 or more for computing the bootstrap confidence interval.

IV. Implementation and Workloads

We have implemented the general methodology described in Section III in order to evaluate its ability to measure, analyze, and predict performance variation in modern systems when combined with different statistical performance prediction techniques. In addition, we have also examined its ability to characterize the potential performance variation of different workloads on several modern HPC systems. This section describes both our implementation for collecting runtime and maxima samples from different workloads on HPC systems.

A. Measurement Implementation

To collect maxima samples from a wide range of workloads, we designed and implemented a simple MPI application that can be used to execute and measure the performance and performance variation of a wide range of HPC workloads. This system is based on a the well-known bulk-synchronous parallel model (BSP) model of large-scale applications. As such, the measurement program executes *k intervals* in which *n processes* compute. Each interval begins with an `MPI_Barrier`, after which the test program runs a specified MPI/compute workload on each process (which may include communication through an MPI subcommunicator), finally executing a second `MPI_Barrier`. Each process logs the time taken to compute the workload, and the rank 0 MPI process logs the time between the completion of these two MPI barriers as the maximum time taken to execute the specified workload across all processes as the length of the kth interval. These logs are stored in memory and written to a parallel file system at program completion to minimize runtime overhead.

B. Synthetic and Application Workloads

We have implemented synthetic workloads that mimic common HPC computation/communication patterns and library versions of several HPC applications in this framework, as summarized in Table I. Specifically, we support parameterized versions of multiple common synthetic single-node compute workloads, and each of these synthetic compute workloads can also be supplemented with a simple 2D halo exchange of a configurable size in each iteration to mimic BSP applications with different computational characteristics. In addition, we have integrated two different distributed MPI applications into this system, HPCG [8] and LAMMPS [21]. These application workloads perform a fully distributed solve or simulation, adding complex communication characteristics to our framework.

For each of these workloads, the global synchronization step guarantees that the distribution of runtimes is the maxima of the local compute/communication times on each node. When all local distributions are identical Gaussian distributions, either given directly or established by the central limit theorem, this distribution approaches a Gumbel distribution. More generally, when the local distributions are all continuous, and independent and identically distributed (IID), the maximum distribution is the generalized extreme value (GEV) distribution. It is important to note, however, that in many cases the distributions of local compute/communication times are *not* IID in real systems. For example, explicit communication or indirect interference (e.g. L3 cache conflicts) between processes can violate independence assumptions. Similarly, differences in resource allocations between cores and nodes (e.g. dynamic clock frequency management) can violate identical distribution assumptions. These challenges can have potentially significant impact on the suitability of different analysis approaches, as discussed in the next section and as evaluated in Section VI.

V. Experimental Setup

To evaluate our statistical prediction techniques and their ability to quantify and predict performance variation on various workloads, we ran the six workloads outlined in Table I at various processor counts on two supercomputing platforms (Table II) to collect maxima timings from each of the *k* intervals of each workload. We then use the non-parametric and parametric bootstrapping methods described in Section III-B to evaluate the ability of each method to predict performance variation at various scales.

Workload	Description	Parameters
FTQ [2]	Spin for a statistically distributed length of time	Distribution and distribution parameters
FWQ [2]	Perform a statistically distributed number of integer additions	Distribution and distribution parameters
DGEMM	Single-Node Dense General Matrix Multiply	Size of matrix and number of multiplications per iteration
SPMV	Single-Node Sparse Matrix Vector Multiply	Size of matrix and number of multiplications per iteration
HPCG [8]	Distributed Pre-conditioned Conjugate Gradient Linear Solver	Global number of rows in matrix being solved
LAMMPS [21]	Molecular Dynamics Solver	Global number of molecules and timesteps to simulate

TABLE I
SYNTHETIC AND APPLICATION WORKLOADS CURRENTLY SUPPORTED BY MEASUREMENT TOOL

Platforms	Cori	Attaway
Processor	Intel Xeon E5-2698 v3 (Haswell) Intel Xeon Phi Processor 7250 (Knights Landing)	Intel Xeon Gold 6140
Clock Speed	2.3 GHz (Haswell) / 1.4 GHz (Knights Landing)	2.3 GHz
Cores per node	32 (Haswell) / 68 (Knight's Landing)	36
Total Nodes	2388 (Haswell) / 9688 (Knights Landing)	1488
Memory per Node	128 GB (Haswell) / 96 GB (Knights Landing)	192 GB

TABLE II
PLATFORM HARDWARE DETAILS

We generated the data used in our evaluation on two supercomputing systems: Cori at the National Energy Research Scientific Computing Center, and Attaway at Sandia National Laboratories. Each workload was run on some combination of 8, 16, 32, and 64 nodes, with 32 MPI ranks per node, corresponding to 256, 512, 1024, and 2048 MPI ranks respectively. Generally, we performed 20 runs on 256 ranks, 5-10 runs on 512 ranks, 5-10 runs on 1024 ranks, and 3-5 runs on 2048 ranks. This provided ample data at "smaller" scales to feed our bootstrapping techniques, while still providing test data at "larger" scales to test the accuracy of our predictive techniques in our limited system allocations. Additionally, the FTQ, FWQ, DGEMM, and SPMV workloads were each run with and without a 1MB 2-D halo exchange, as described in Section IV-B. The halo exchange component was not included for the HPCG and LAMMPS workloads as each already contains communication and synchronization operations.

Each workload was controlled with several input parameters (Table I). These were tuned to allow each workload to run in reasonable amounts of time, to generate enough data for bootstrapping, and to utilize appropriate amounts of memory to minimize cache effects. Specifically, the FTQ workload was run with 100 ms normally distributed intervals, while the FWQ workload was similarly run with normally distributed loop counts. Additionally, the SPMV and HPCG workloads were initialized to utilize 1GB per rank of memory to minimize cache effects, and the LAMMPS workload was performed on 64,000 atoms per rank for 250 timesteps utilizing the Lennard-Jones potential.

Overall, we performed 467 runs on Cori and 330 runs on Attaway, broken down between the 6 workloads of interest and with or without the inclusion of the additional 2D halo exchange as seen in Table III. With these runs, we were able to collect data and characterize the performance of a significant portion of the Cori and Attaway systems. Shown in Figure 1, our experiments utilized 1461 of the 2388 unique Haswell nodes on Cori and 319 of the 1488 unique Xeon Gold nodes on Attaway. On both Cori Haswell nodes and Attaway nodes, we used 32 cores per node. Cori nodes are configured to allow hardware use of Intel Turbo Boost by default and we did not explicitly disable it. Attaway nodes do not have have Turbo Boost enabled.

System	Cori (# of experiments)		Attaway (# of experiments)	
Workload	*No Halo*	*Halo*	*No Halo*	*Halo*
FTQ	38	38	30	30
FWQ	53	53	30	30
DGEMM	53	53	40	40
SPMV	53	53	35	35
HPCG	38	0	30	0
LAMMPS	35	0	30	0

TABLE III
EXPERIMENT OVERVIEW

VI. EVALUATION

We evaluated the non-parametric and parametric bootstrapping techniques described in Section III with the measurement and analysis framework described in Section IV for its ability to accurately quantify and predict variation of the six chosen

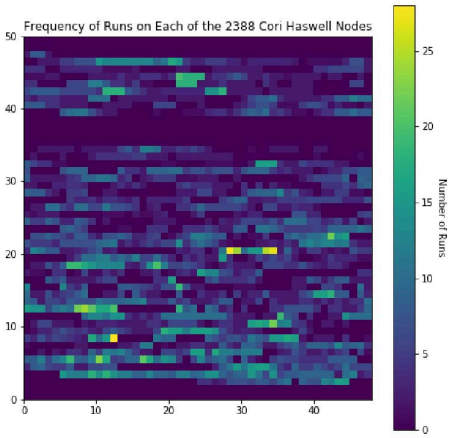


Fig. 1. Heatmap of Nodes Utilized for Cori Experiments. Overall Coverage of 61% of Haswell Nodes Available on Cori.

workloads in Table I. We find that both non-parametric and parametric bootstrapping can reliably predict the performance and variation of controlled workloads and commonly used kernels such as DGEMM and SPMV, which only synchronize after each iteration, but do not internally communicate. For workloads that contain internal communication and network dependencies, the parametric bootstrapping is able to capture the scaling behavior, but would require more calibration data to robustly narrow and test its confidence intervals due to their sensitivity to the internal synchronization and communication overheads of these workloads.

The remainder of this section presents validation that our measurement technique is scalable, a quantification of the performance variability observed, our evaluation approach, and an evaluation of the ability of each bootstrapping technique to capture the performance variation of each workload on the two supercomputing systems.

A. Validation of Data Collection

To validate the scalability of our measurement technique, we measured the time spent in the barrier after each iteration of a workload. We chose to look at a sample of the DGEMM workload at 1024 ranks in order to characterize the average amount of time spent in the barrier after each iteration. On average, the DGEMM workload took 790 milliseconds on Cori and 820 milliseconds on Attaway, while the average time spent in the barrier was 24 microseconds and 15 microseconds on each system respectively. This indicates that a negligible amount of the workload is spent in the barrier at some of the largest scales we ran at, showing our measurement technique to be highly scalable on both systems.

B. Assessment of Performance Variation

We evaluated the overall runtimes of each run to assess the performance variation observed for each workload on each platform. We present the runtimes both with and without halo exchanges, but do not specifically discuss variance in the halo exchange runs. As seen in Figure 2, the runtime performance on Cori showed minimal variability for the FTQ workload (less than 1%), while compute-bound workloads (e.g. DGEMM) showed up to 10% variability across identical runs. The Attaway runtimes in Figure 3 showed less variability for the FTQ, FWQ, DGEMM, and SPMV workloads, but experienced 15% and 94% variability for the HPCG and LAMMPS workloads, respectively, most likely due to significant internal network dependencies.

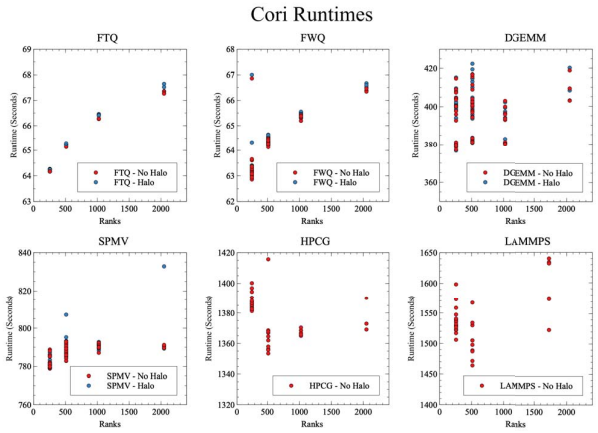


Fig. 2. Experiment runtimes of each workload at various scales on the NERSC Cori supercomputer

While the controlled workloads of FTQ and FWQ were well-behaved, the DGEMM and SPMV workloads exhibited larger variances since they are real compute-bound and memory-bound kernels subject to processor speed control and memory contention. Similarly, the HPCG and LAMMPS workloads exhibited more unpredictable behavior due to their internal communication patterns and network dependencies.

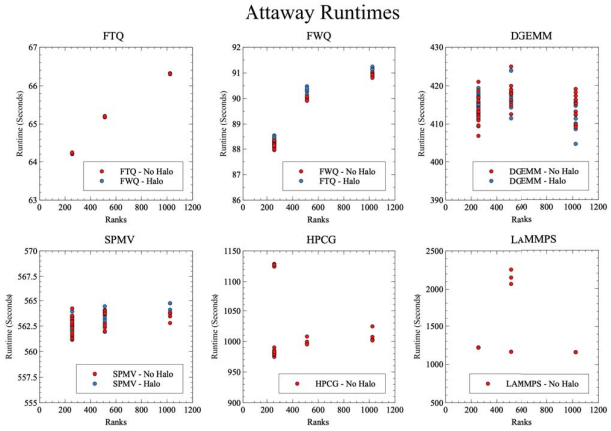
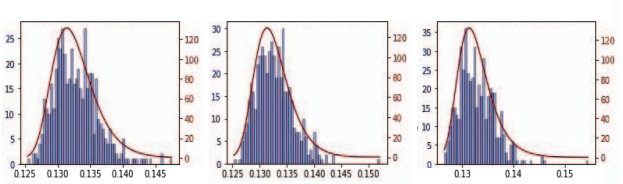


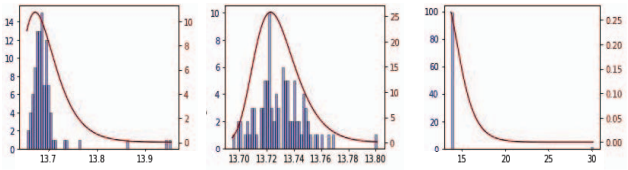
Fig. 3. Experiment runtimes of each workload at various scales on the SNL Attaway supercomputer

Beyond the visual inspection, we can characterize the vari-

ability of workloads across runs by analyzing their GEV fitting parameters. For example, Figure 4 shows MOM fittings for the very stable FTQ workload, and for the more unpredictable HPCG. For space purposes we depict only MOM fittings; however we compare both PWM and MOM. For FTQ, both PWM and MOM agree across runs that this is a Type II distribution. On the other hand, for HPCG the fitting methods do not reach an agreement. PWM characterizes the distribution as Type III, while MOM characterizes runs 1 and 3 as Type I, and run 2 as Type II (see Section II for the definition of the different types). This assessment of distribution type and agreement between fitting methods provides us with valuable insight into whether we can trust the predictive capabilities of our methods for specific workloads.



a) FTQ workload variation and MOM fitting



b) HPCG workload variation and MOM fitting

Fig. 4. Characterization of maxima for FTQ and HPCG across Cori runs

C. Evaluation Approach

Our evaluation approach consists of four steps:

- Utilization of the data collected to calculate the total runtime of each experiment.
- Prediction of performance variation at larger scales with the bootstrapping techniques described in Section III-B applied to the smallest data set collected, the 256 rank data.
- Evaluation of the ability of each technique to accurately predict and quantify performance variation.
- Discussion of the trade-offs of the various techniques, their underlying assumptions, and their performance.

For each method, the data resulting from running each workload with 256 ranks (the smallest data set collected) on Cori and Attaway was supplied as input to the performance variation prediction techniques of Section III-B. These methods were then used to generate medians and confidence intervals of the predicted runtimes at various scales. The actual runtimes at 256, 512, 1024, and 2048 ranks were then used to evaluate whether each method was able to successfully quantify and predict the performance variation observed. Evaluation criteria were then applied in order to determine whether the observed runtimes fell within the predicted confidence intervals, and whether the confidence interval reasonably reflected the observed variance at larger scales.

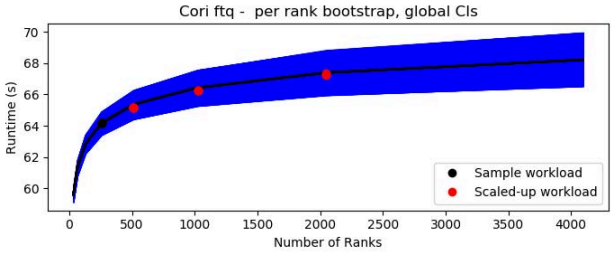
D. Prediction of Performance Variation

Workloads that are more controlled (e.g. FTQ) can be predicted well by techniques (e.g. EMMA, GEV) that further characterize variance, but those techniques make strong assumptions (e.g. known distribution or i.i.d.) which don't work well on workloads with more complicated behaviors. On those, more general techniques that cannot bound variation as tightly are necessary.

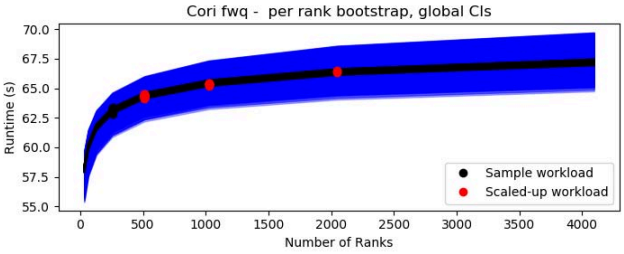
For controlled workloads, such as FTQ and FWQ, both non-parametric and parametric methods exhibited good predictive performance. As seen in Figure 5, the intra-node parametric technique was able to successfully predict the runtimes of the FTQ and FWQ workloads at 512, 1024, and 2048 ranks successfully while maintaining reasonable confidence intervals. Note that LAMMPS was run at 1728 ranks instead to satisfy exact weak scaling at 64000 atoms per rank. The inter-node results were excluded for brevity, but exhibit the same behavior.

For less-controlled workloads that are representative of common kernels in HPC applications, the parametric methods again provided good predictive power. Since the DGEMM and SPMV workloads were run such that each rank was responsible only for its own operations, with no internal communication except for the synchronization after each iteration was performed. The kernels exhibit almost perfect weak scaling, and the parametric methods are able to accurately capture this as seen in Figure 6. However, since the DGEMM kernel is compute bound and subject to CPU frequency variation and other sources of noise, and because SPMV is memory bound and subject to memory latency variations, there can be substantial variation of up to 10% in runtimes across identical runs, as seen in Figures 2 - 3. As seen in Figure 6, the inter-node parametric method also captures this variation accurately. This time, we have excluded the intra-node results for brevity, but they exhibit similar behavior.

The resampling non-parametric prediction method and the intra-node and inter-node parametric prediction methods can behave differently in evaluation of real-world applications such as the HPCG and LAMMPS workloads. The communication and synchronization within each iteration of HPCG and LAMMPS result in an additional dependency on network performance which may produce extreme outliers as network traffic fluctuates. This internal communication and synchronization occurs before our measurement framework runs at each step, effectively masking the variation in performance between the various ranks. The same synchronization ahead of our measurement point can cause all ranks of the application to be reported as outliers in an iteration if even one rank is delayed due to communication overhead. The net effect is that the measured time of each rank is equal to the maximum time of all the ranks since synchronization has already occurred within the HPCG and LAMMPS applications at the time of

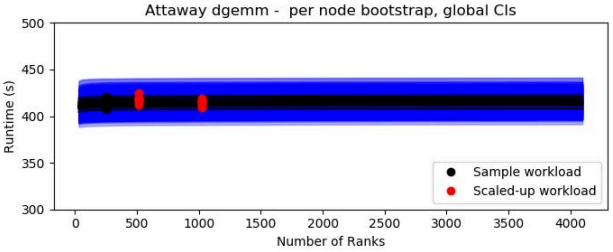


(a) Parametric intra-node prediction of performance variation of Cori FTQ workload

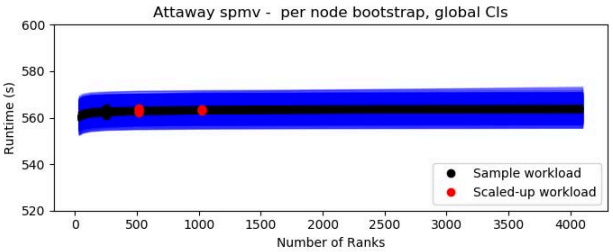


(b) Parametric intra-node prediction of performance variation of Cori FWQ workload

Fig. 5. Cori FTQ and FWQ performance variation prediction

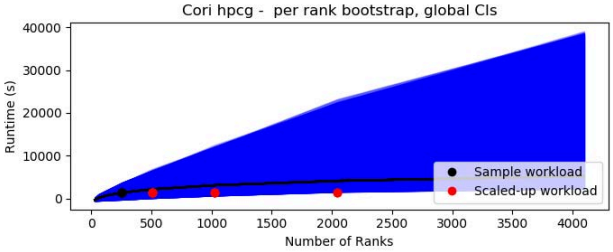


(a) Parametric inter-node prediction of performance variation of Attaway DGEMM workload

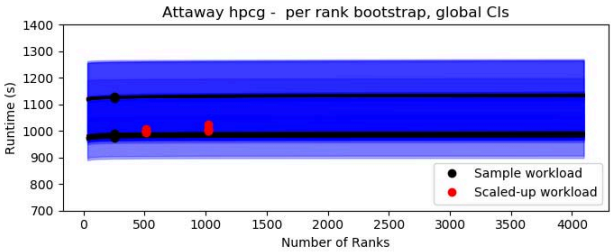


(b) Parametric inter-node prediction of performance variation of Attaway SPMV workload

Fig. 6. Attaway DGEMM and SPMV performance variation prediction



(a) Parametric intra-node prediction of performance variation of Cori HPCG workload



(b) Parametric intra-node prediction of performance variation of Attaway HPCG workload

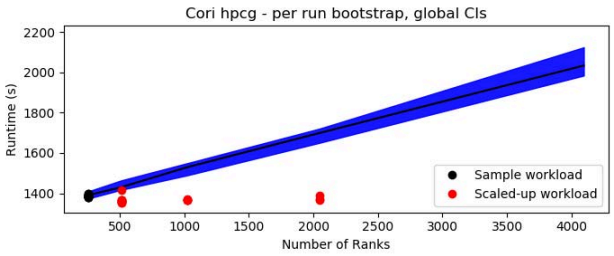
Fig. 7. Parametric performance variation prediction of HPCG workload

measurement. Instrumenting and estimating variation using the bulk-synchronous communication points in the applications themselves would also potentially address this.

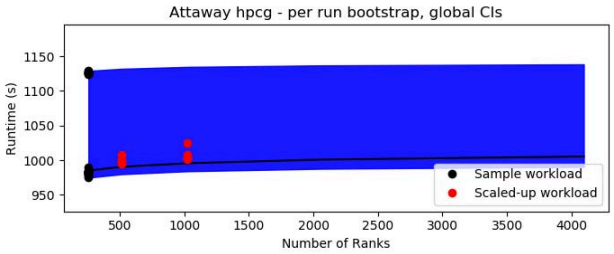
As seen in Figure 7, a consequence of this is increased sensitivity of the parametric methods to tail behavior and larger predicted confidence intervals and variation than observed. This may happen more frequently on larger systems whose networks are more likely to experience performance variations; in the specific case of our results, Cori is much larger than Attaway. Hence, the likelihood of having large outliers due to internal synchronization barriers within HPCG is higher on Cori. This results in an overestimation of variability on Cori, compared to the accurate prediction of variability on Attaway for the HPCG workload. Similar behavior was observed for both parametric methods, but only the intra-node results are provided for brevity.

The non-parametric prediction method exhibits a similar behavior, but to a lesser degree. Rather than utilizing GEV estimation and iteration runtime data from each rank, it simply re-samples the maximas. The result is that the tail is still amplified due to the internal communication and synchronization overhead within HPCG and LAMMPS, but the tail is not over-counted from each rank since the non-parametric re-sampling method discards all non-maxima data. The end result is Figure 8, where the Cori prediction still overpredicts runtime due to the communication and synchronization overhead, but by a much lesser degree than the parametric methods. Similarly, since Attaway is a quieter system it is less affected by internal network overheads and still provides a relatively accurate characterization and prediction of performance variation.

Evaluating the quantification and prediction of performance variation on LAMMPS workloads, a similar pattern is ob-

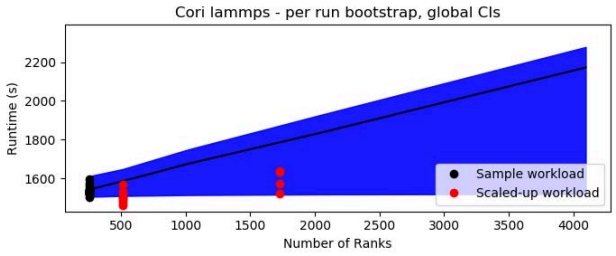


(a) Non-parametric re-sampling prediction of performance variation of Cori HPCG workload

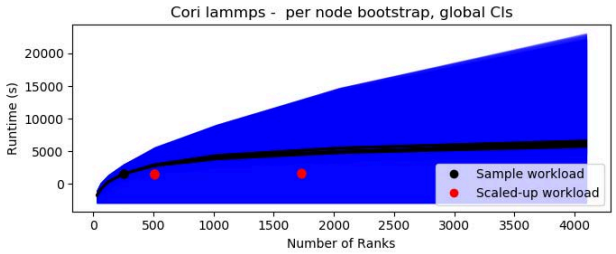


(b) Non-parametric re-sampling prediction of performance variation of Attaway HPCG workload

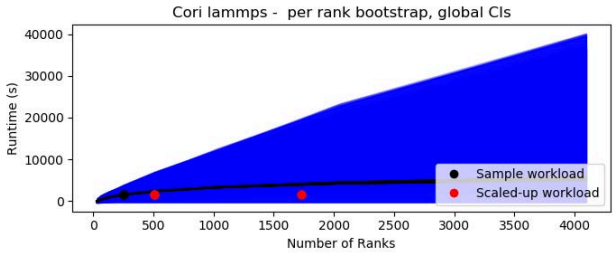
Fig. 8. Non-parametric performance variation prediction of HPCG workload



(a) Non-parametric re-sampling prediction of performance variation of Cori LAMMPS workload



(b) Parametric inter-node prediction of performance variation of Cori LAMMPS workload



(c) Parametric intra-node prediction of performance variation of Cori LAMMPS workload

Fig. 9. Performance variation prediction of LAMMPS workload

served. The parametric methods over-estimate the performance variability, resulting in large confidence intervals. The inter-node method also results in non-physical negative runtime predictions as seen in Figure 9(b), since the GEV estimation is unable to negotiate the amplified tail. Increasing the granularity of GEV estimation to per-rank data alleviates this problem (Figure 9(c)) as the GEV estimation is able to provide higher fidelity parameters. On the other hand, the non-parametric re-sampling method slightly overestimates performance variability, but does an acceptable job of capturing and predicting the performance variation of the LAMMPS workload. It does this despite the internal communication and synchronization overhead present within the LAMMPS workload. It is worth noting that even though the parametric methods overestimate confidence intervals for workloads like HPCG and LAMMPS, the predicted medians are still very close to the actual scaled-up workloads. Following our discussion of assessment of performance variation, we can use the insight provided by the GEV parameter estimation to determine when these confidence intervals are likely to be overestimated.

E. Discussion

The performance of our parametric and non-parametric methods on these six workloads performed on two different systems leads us to conclude the following:

- The performance variation of trivial and controlled workloads, such as FTQ and FWQ, can be accurately modeled and predicted by parametric and non-parametric methods.
- Computation and memory-bound kernels without any only local (e.g. halo) communication can be accurately measured, and their performance variation accurately quantified and predicted using both parametric and non-parametric methods.
- Parametric methods that rely on GEV estimation and either node or rank level granularity of data rather than iteration maxima granularity of data overestimate performance variation on workloads with internal communication and synchronization overhead. However, their median estimates remain accurate.
- Parametric methods can still be useful in evaluation and predicting performance variation on workloads with internal communication and synchronization overhead if and only if the network behaves predictably and with minimal fluctuations.
- Non-parametric methods can provide a technique to characterize and predict real workloads with internal communication and synchronization overhead with less overestimation that parametric methods.

VII. Related Work

In addition to the many evaluations of the underlying mechanisms that cause performance variation, there are several benchmarking studies of HPC systems in the literature. CPU throttling and power capping mechanisms, such as Intel Turbo Boost, can amplify the effect of manufacturing variability on CPU frequency and has been shown to degrade performance of HPC applications [1]. OS jitter and interference have been shown to affect application performance and to limit application scalability [12]. There is also evidence that nearby jobs are another source of performance variation that is difficult to model and predict as HPC jobs increase in scale [5]. As a result, libraries that simulate common interference patterns that stress the CPU, cache, memory, and network subsystems have been developed to aid the evaluation, benchmarking, and reproducibility of performance variation [4].

One of the many challenges of modeling performance variability is that HPC systems and applications have both spatial and temporal variability [16]. Many authors have measured and modeled performance variation using machine learning techniques [27], statistical analysis [20], and neighborhood analysis [6], but do not provide a framework to predict the scaling behavior of variability in applications. A machine learning model has been developed to identify runtime anomalies and performance variation in near real-time [27]. Furthermore, a method of performance analysis and extrapolation for bulk synchronous programs which utilizes extreme value theory was developed [20].

Network counters and neighborhood analysis have also been used to investigate the causes of performance variability in HPC systems and forecast execution time [6]. Researchers have also demonstrated broader sources of software and hardware architecture and application level variability, measured the performance loss caused by variability, and tried to control and decrease these sources and measure the resulting performance improvement [25].

Similar performance variability studies have also been undertaken on EC2 and FutureGrid/Eucalyptus cloud platforms [11]. They showed that the performance variation between their experiments is mostly related to the communication time and the type of solver/preconditioner they applied, rather than OS load.

HPC system performance variability also poses challenges to the evaluation of the effectiveness of performance tuning, specifically in the case when the amplitude of variability, due to differences in hardware manufacturing and HPC center cooling patterns, masks the change in performance caused by certain tuning enhancements [22].

VIII. Conclusions and Future Work

In this paper, we describe a new approach to quantifying and measuring performance variation in HPC applications. Overall, the proposed combination of bootstrapping and a maxima-based approach to measuring, quantifying, and predicting performance variation significantly reduces the scale of the measurement and analysis problem compared to handling and integrating performance variation from individual nodes. The parametric and non-parametric bootstrap methods we examined for predicting changes in performance variance under this model have different strengths. The non-parametric bootstrap is more effective at predicting applications with complex communication behaviors. The parametric bootstrap can quantify application variance and scaling with fewer parameters and samples, but is also less accurate for complex applications whose communications can violate GEV distribution modeling assumptions.

In future work, we plan to examine hybrid parametric and non-parametric bootstrapping approaches which can predict application scaling for complex applications while minimizing data collection requirements. Our statistical approach to quantifying application performance and scaling could also serve as the basis for performance anomaly detection in large scale systems; it could also be used to inform and optimize system schedulers and other resource allocators. We also plan to examine the use of this technique in predicting and mitigating performance variation in cloud and big data systems.

Acknowledgements

This paper was supported in part by the National Science Foundation under Grant No. OAC-1807563, and by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the United States Department of Energy.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231, resources at the UNM Center for Advanced Research Computing, and from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562 through allocation ASC190036.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2020-9636C.

This work was funded in part by Los Alamos National Laboratory, supported by the US Department of Energy contract DE-FC02-06ER25750 (Los Alamos Publication Number LA-UR-20-28021).

References

[1] B. Acun, P. Miller, and L. V. Kale. Variation among processors under Turbo Boost in HPC systems. *Proceedings of the International Conference on Supercomputing*, 01-03-June, 2016.

[2] Advanced Simulation and Computing, Lawrence Livermore National Laboratory. ASC Sequoia Benchmark Codes, 2008. https://asc.llnl.gov/sequoia/benchmarks/FTQ_summary_v1.1.pdf.

[3] P. Ailliot, C. Thompson, and P. Thomson. Mixed methods for fitting the gev distribution. *Water Resources Research*, 47, 2011.

[4] E. Ates, Y. Zhang, B. Aksar, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun. HPAS: An HPC performance anomaly suite for reproducing performance variations. *ACM International Conference Proceeding Series*, 2019.

[5] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2013.

[6] A. Bhatele, J. J. Thiagarajan, T. Groves, R. Anirudh, S. A. Smith, B. Cook, and D. K. Lowenthal. The case of performance variability on dragonfly-based systems. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 896–905, 2020.

[7] G. Calmettes, G. B. Drummond, and S. L. Vowler. Making do with what we have: use your bootstraps. *The Journal of Physiology*, 590:3403–3406, 8 2012.

[8] J. Dongarra, M. A. Heroux, and P. Luszczek. High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems. *The International Journal of High Performance Computing Applications*, 30(1):3–10, 2016.

[9] D. J. Dupuis and C. A. Field. A comparison of confidence intervals for generalized extreme-value distributions. *Journal of Statistical Computation and Simulation*, 61:341–360, 1998.

[10] B. Efron. Better bootstrap confidence intervals. *Journal of the American Statistical Association*, 82:171, 3 1987.

[11] Y. El-Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of hpc workloads on clouds. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 383–387. IEEE, 2010.

[12] K. B. Ferreira, P. Bridges, and R. Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection. *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008*, pages 1–20, 2008.

[13] T. Groves, R. E. Grant, and D. Arnold. Nimc: Characterizing and eliminating network-induced memory contention. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 253–262, 2016.

[14] T. Hoefler and R. Belli. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2015.

[15] J. R. M. Hosking, J. R. Wallis, and E. F. Wood. Estimation of the generalized extreme-value distribution by the method of probability weighted moments. *Technometrics*, 27(3):251–261, 1985.

[16] B. Kocoloski. *Scalability in the Presence of Variability*. PhD thesis, University of Pittsburgh, 2018.

[17] J. Kyselý. A cautionary note on the use of nonparametric bootstrap for estimating uncertainties in extreme-value models. *Journal of Applied Meteorology and Climatology*, 47:3236–3251, 2008.

[18] P. Leitner and J. Cito. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. *ACM Transactions on Internet Technology (TOIT)*, 16(3):1–23, 2016.

[19] H. Madsen, P. F. Rasmussen, and D. Rosbjerg. Comparison of annual maximum series and partial duration series methods for modeling extreme hydrologic events. *Water Resources Research*, 33(4):747–757, 1997.

[20] O. H. Mondragon, P. G. Bridges, S. Levy, K. B. Ferreira, and P. Widener. Understanding Performance Interference in Next-Generation HPC Systems. *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 0(November):384–395, 2016.

[21] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1):1–19, Mar. 1995.

[22] A. Porterfield, S. Bhalachandra, W. Wang, and R. Fowler. Variability: A tuning headache. *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, pages 1069–1072, 2016.

[23] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1-2):460–471, 2010.

[24] Scipy. scipy.stats.rv_continuous.fit, 2020. Last accessed September 11, 2020.

[25] D. Skinner and W. Kramer. Understanding the causes of performance variability in HPC workloads. *Proceedings of the 2005 IEEE International Symposium on Workload Characterization, IISWC-2005*, 2005:137–149, 2005.

[26] J. Sun and G. D. Peterson. An effective execution time approximation method for parallel computing. *Parallel and Distributed Systems, IEEE Transactions on*, 23(11):2024–2032, 2012.

[27] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun. Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning. *IEEE Transactions on Parallel and Distributed Systems*, 30(4):883–896, 2019.

[28] Y. Ueda and T. Nakatani. Performance variations of two open-source cloud platforms. In *IEEE International Symposium on Workload Characterization (IISWC'10)*, pages 1–10. IEEE, 2010.