A generalized statistics-based model for predicting network-induced variability

Sudheer Chunduri, Elise Jennings, Kevin Harms, Christopher Knight, Scott Parker
Argonne National Laboratory
{sudheer, ejennings, kharms, knightc, sparker}@anl.gov

***Abstract*—Shared network topologies, such as dragonfly, subject applications to unavoidable inter-job interference arising from congestion on shared network links. Quantifying the impact of congestion is essential for effectively assessing and comparing the application runtimes. We use network performance counter-based metrics for this quantification. We claim and demonstrate that by using a local view of congestion captured through the counters monitored during a given application run, we can accurately determine the run conditions and thereby estimate the impact on the application's performance. We construct a predictive model that is trained using several applications with distinctive communication characteristics run under production system conditions with a 91% accuracy for predicting congestion effects.**

***Index Terms*—variability, congestion, performance counters, Aries, tuning**

I. Introduction

Application performance analysis is a critical activity of high-performance computing (HPC). The presence of variability in application runtime [1], [2] causes difficulties with application performance optimization, scaling, and benchmarking analyses. The variability may come from many different sources, such as dynamic frequency scaling of the CPU, and from shared resources, such as CPU cache or network links. These effects have been the subject of many studies [3]–[5]. The effects can be mitigated or reduced [5], [6] but cannot be completely removed. The variable effects due to network sharing are also likely to continue or even increase, as exemplified by the use of shared network topologies in the current top systems, Summit and Sierra, and in the proposed future systems. HPC users will have to adapt to this environment and find techniques that allow for understanding application runtime and performance in production environments.

One aspect that can significantly contribute to variability of large-scale system jobs is the high-speed interconnect network. An HPC system that allocates distinct compute resources to jobs can potentially share network links, thus introducing contention between jobs. The interconnect does not provide quality of service guarantees to specifically allocate fractions of the bandwidth to each job. When multiple jobs demand peak bandwidth rates at the same time, congestion on the network links exacerbates. The contention which these jobs experience is a function of the other jobs running on the system and how the network usage of the jobs interacts. The contention impacts change over time based on the production workload that the system is running at any given time. As part of this study, we specifically examined the Cray XC Aries interconnect [7] using Dragonfly topology, which is currently used by many facilities and appears in 22 of the Top 100 systems in the world, based on the June 2019 Top500 list. The effects of inter-job contention within a Dragonfly network have been well studied [3], [8]–[10]. These effects can be completely mitigated only by running the job on an isolated system with no other jobs running. This method is not practical for frequent use within production facilities. Therefore, HPC users must learn to cope with variability introduced by inter-job contention on the interconnect. Tools which assist users in interpreting the performance of an execution and in comparing the performance across different executions of an application are essential. However, currently, no such tools are available to help users understand the variability that occurs from run to run.

Two key tasks involved in understanding the application performance variability are (i) predicting the congestion that occurred due to inter-job contention with the potential to influence the application performance and (ii) assessing the application sensitivity to that congestion. While the network performance counters can be used to estimate the congestion, careful aggregation of the counter data by filtering out unrelated data across different nodes allocated to the job is essential. The second task of assessing application sensitivity to congestion is crucial for accurately estimating the application performance variability.

We use the network performance counters for accomplishing both of the key tasks mentioned above. As part of this study, a set of applications were run periodically under production conditions, with network counter data collected and analyzed for each run. A thorough statistical analysis of this counter data is performed. Predictive machine learning models that can be used to understand the relative performance of any given run were built using these statistics. The relative performance shows if the run had mean performance or was $\pm n$ standard deviations from the mean. We construct a prototype software package to train a model that can determine the effect of congestion on the application run and provide an estimate of the impact based on the deviation from the expected mean runtime.

The main contributions of this work are: (i) demonstrating that network performance counter-based metrics can be used to assess the application sensitivity to congestion in variable production environments, (ii) an empirically validated machine

978-1-7281-5977-5/19/$31.00 ©2019 IEEE
DOI 10.1109/PMBS49563.2019.00013

learning model for estimating an application's runtime variability, (iii) an open source model with summary data, and (iv) adaptability of the approach to other systems (HPC systems typically provide an interface for network performance counters, making the insights and methods detailed in this study adaptable to other systems as well).

Section II discusses the experimental setup, systems and applications used, and the measurement methodology. Section III presents the analysis of the correlation between the application runtime variability and the variability in network counters. Also, validation of the production experiments is presented using controlled experiments. Sections IV and V respectively describe the construction and evaluation of a predictive model. A description of the practical use cases of the model is provided in Section VI. Section VII presents the related work, and Section VIII provides an overall summary and conclusion of this study.

II. Experimental Setup

A. Machine Details

Experiments were conducted on Theta at the Argonne Leadership Computing Facility (ALCF). Theta is an 11.69-petaflops Cray XC40 with 4,392 Intel Xeon Phi Knights Landing (KNL) 7230 compute nodes connected with the Cray Aries high-speed network in the Cray dragonfly topology.

Theta uses the Cray Aries interconnect configured with a three-level dragonfly topology. The first two levels (rank-1 and rank-2) are copper-based with 10.5 GB/s bidirectional bandwidth per link, and the rank-3 level is optical with 9.38 GB/s per link. The three rank-2 links are used to connect each pair of routers in the intragroup columns, and each router is connected to the 15 other routers in a row with a rank-1 link (refer figure 8 in [7]). Although the organization of rank-1 and rank-2 links is consistent across different Cray XC based installations, the arrangement of the active optical cables can vary. Theta has 12 groups with 12 active optical cables (3 lanes each) [7] between each group. The Cray default routing algorithm was used for all experiments presented. Theta prefers to allocate nodes to a job that span a large number of groups for a given job in order to maximize global bisection bandwidth.

B. Applications

For this study, seven application cases with a wide range of communication patterns and behaviors were used. Production-scale applications were used instead of benchmarks for representing real workload scenarios and production system conditions. Each application was configured and run with a realistic input case. The run configuration followed the best practices [5] in order to minimize any variability from other sources, such as the operating system, CPU, and memory subsystems. I/O phases were avoided for runtime considerations to reduce variability effects introduced from the storage system. Table I provides a brief overview of the communication behaviors of these applications.

TABLE I. High-level summary of communication properties for each application.

App	Point-to-point	Collectives	Approx. Comp.-to-Comm.Ratio
MILC	heavy	light allreduce	3:2
MILC REORDER	medium	light allreduce	2:1
Nekbone	medium	light	8:1
Nek5000	medium	light	4:1
Qbox	medium	medium	1:2
Rayleigh	none	heavy	2:1
LAMMPS	light	medium	2:1

MILC [11] is an MIMD lattice computation code for simulation of high energy and nuclear physics. **MILC REORDER** is the same MILC with an optimized rank-to-node mapping, which reduces off-node communication. This was done using Cray's grid_order tool with a $4 \times 4 \times 2 \times 2$ subgrid. **Nek5000** [12] is a computational fluid dynamics code with a high-order, incompressible Navier-Stokes solver based on the spectral element method. **Nekbone** is a miniapp that exposes the principal computational and communication kernels of Nek5000. **Rayleigh** [13] is a 3D convection code that evolves the incompressible and anelastic magnetohydrodynamics equations in spherical geometry by using a pseudo-spectral approach. Rayleigh is used for the study of planetary and stellar dynamos, and is a pure-MPI code that uses large-message-size all-to-alls. **Qbox** [14], used for electronic structure calculations, is an ab initio molecular dynamics code written in C and based on density functional theory. **LAMMPS** [15] is an open-source molecular dynamics code written in C/C++ to simulate systems spanning several science domains (e.g., liquids, biomolecules, materials, and mesoscopic systems).

C. Experiment Details

The majority of the experiments in this study were run under typical production environments on Theta. Each application was run under a range of node sizes (128, 256, and 512) that are 3 to 11 percent of the system size of Theta. These sizes were selected based on the likelihood that, because of their smallness, they would be affected by other applications on the system. Each application was run on the order of $\sim$200–500 times to sample the distribution space accurately and to ensure that the statistics are robust. Also, the number of runs allowed us to assess the presence and impact of a spectrum of congestion scenarios. For each run, the execution time, network performance counters, and job placement information were recorded. The node placement information recorded for an Aries topology is the (group, chassis, blade, and node) for all nodes allocated to the job.

The remainder of the experiments were performed under controlled conditions. The controlled experiments are further described in section III-F.

D. Aries Router Performance Counters

The performance counters [16] used for analysis are custom counters provided by Cray as part of the Aries high-speed

interconnect. A router is composed of 48 "tiles" where each tile has a set of counters. Aries provides a large number of possible network counters, some of which are configurable, to monitor traffic on the network. These counters primarily measure FLITs and STALLs across row and column buffers and virtual channels. The FLIT counters record the atomic unit of data transfer on the Aries interconnect. A STALL counter increments each FLIT time that a ready-to-forward FLIT is unable to forward. This is normally due to a lack of credits in the router buffers or arbitration policies. The counters INQ_PRF_INCOMING_FLIT_VC{0..7} capture the flits on the eight virtual channels. The eight router tiles directly connected to the nodes (also referred to as processor tiles or proctiles) use only two virtual channels: INQ_PRF_INCOMING_FLIT_VC0 and INQ_PRF_INCOMING_FLIT_VC4. The incoming FLITs are aggregated together for each tile across virtual channels. The stalls are captured by the counter INQ_PRF_ROWBUS_STALL_CNT. Stalls are an indication of the back pressure limiting the rate at which flits are able to be forwarded. A high ratio of stalls to flits indicates possible network congestion. These counters were also used by [17], [18], and [3].

A job on the system is allocated to a set of nodes that are distributed across groups, chassis, and blades of the Aries network. Network counters are only captured from the router tiles that are connected to the nodes of the job. Depending on the job size and allocation scheme, either some or all of the nodes on a blade will be allocated to a job. The counter data from the network tiles reflect either request or response traffic from both the application and from other jobs running at the same time. The nature of the dragonfly routing may also result in traffic from the job being routed via network tiles that are not connected to any of the nodes within the job. In this case, this information is not captured.

The counters are recorded by using the PAPI [19] interface either at the job startup or before the computational section begins. The counters are then collected again at the end of the computational section or end of the job. The collection of the counters takes less than a second, and writing the log files takes about one second. The resulting overhead is trivial for jobs with runtimes on the order of minutes or hours.

III. Methodology

Before construction of a model based on the data, an initial analysis is performed on the metrics used to better understand the data and determine if a generalizable model could be constructed. The correlation between the variability in runtime and the metrics collected is examined.

A. Runtime Variability

Figure 1 shows the absolute runtimes of all data collected for the MILC application on both the normal and the network optimized (Reorder) configurations. This is representative of the runtime variability seen by applications on Theta. For both MILC and Reorder on all node sizes, the distribution

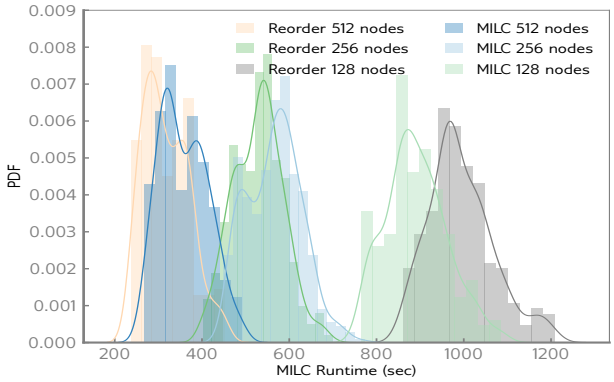

Fig. 1. Probability distribution functions (PDF) for MILC and MILC Reorder runtimes in seconds for 128 nodes, 256 nodes and 512 nodes.

of runtimes with an extended tail can be seen. The extended tail indicates there are runtimes which are multiple standard deviations from the mean. This behavior is consistent across all of the application cases studied. Table II summarizes the runtime statistics for all the applications running at 256 nodes. This table shows that the applications were configured for a variety of different total runtimes and that in all cases significant variability occurred. The maximum runtime divided by the minimum runtime shows between 1.18x to 1.74x difference. The maximum runtimes recorded show that the MILC, Nekbone, MILC REORDER, and Qbox applications display significant deviations from their mean in a way that is not captured by the CV statistic. This result is not surprising because CV measures the ratio of the standard deviation to the mean and can be uninformative for skewed or long-tailed distributions. Hence, statistics that represent the tails must be used for better characterizing the runtime variability.

TABLE II. Mean (μ), standard deviation (σ), maximum, and coefficient of variation (CV) of runtime (seconds) and the ratio of maximum to minimum runtime for the applications at 256 nodes on Theta.

App	$\mu \pm \sigma$	Max	CV	Max/Min
MILC	566.2 $\pm$ 61.90	771.8	0.11	1.71
MILC Reorder	528.7 $\pm$ 57.03	699.5	0.11	1.74
Nekbone	1585.1 $\pm$ 101.8	1960.5	0.06	1.41
Nek5000	429.9 $\pm$ 12.16	472.3	0.03	1.13
Qbox	675.8 $\pm$ 43.90	824.4	0.07	1.39
Rayleigh	680.9 $\pm$ 18.68	763.5	0.03	1.18
LAMMPS	721.6 $\pm$25.50	779.5	0.04	1.12

B. Network Counter Characteristics

As stated in subsection II-D, counter values from an Aries router tile are either classified as a "network" tile or a "processor" tile. Network tiles represent request or response traffic from both the measured application and other jobs running at the same time. These counters represent the global traffic occurring on the system from a local application viewpoint. The processor tiles counter data are data generated or received only at the network interface of the compute node. The

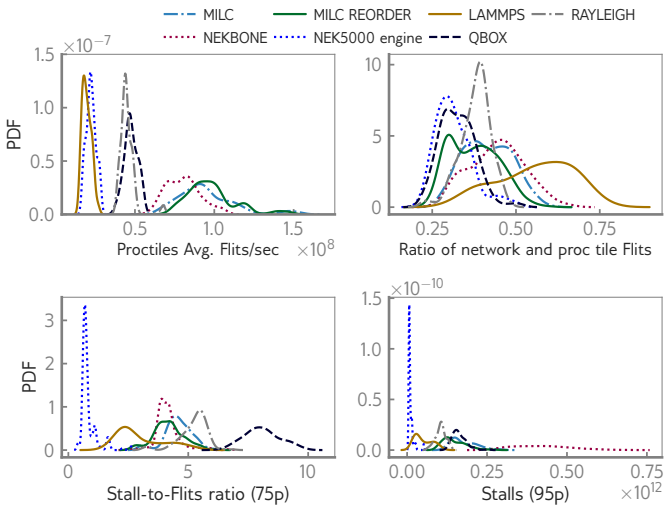

Fig. 2. PDFs for four top features for applications using 256 nodes: MILC (light blue), Nekbone (purple), MILC reorder (green), Nek5000 (blue), LAMMPS (yellow), Qbox (black), and Rayleigh (gray).

processor tiles represent the application-specific characteristics that are unique to that application.

C. Network Counter Collection

Since it is not feasible to distinguish between which network tiles are used by the measured application, all router tile counter data from all 40 network tiles and 8 processor tiles are treated with equivalent weight. The job scheduler will pack the application allocation onto the same compute blade when possible, but there are cases when the measured application occupies a fraction (0.25, 0.5 or 0.75) of the compute blade. In these cases, we examined scaling the values on these particular nodes by the fraction of the blade used but did not find this significantly impacted the results, and hence data from all 8 processor tiles were used.

The network counters are collected by each process running on every node, and up to four nodes may be connected to the same router tile. Applications in our test were run with either 63 or 64 processes per node. As a result, different processes collect the same counter data in many instances. The network counters are used from a single node on each uniquely allocated compute blade. Ideally, each process sharing the same router tile should have exactly the same counter values, but skew in the initial and final counter collection time will result in small variations of the counter values. Therefore, when reporting counter measurements on a tile, we average the values over all the processes.

D. Network Counter Metrics

The network counter values were used to construct several metrics that describe the application and environment the application was running under. Table III summarizes these metrics.

For each metric except nodes, the probability distribution was characterized by a set of summary statistics which are the mean, variance, 75^{th}, 95^{th} and 99^{th} percentiles of the metric.

TABLE III. METRICS AND DEFINITIONS

Metric	Definitions
processor tile flits per second	application communication intensity
network tile stall-to-flit ratio	relative congestion on network
network tile to processor tile flit ratio	relative communication on the global network versus the injection rate of the application
network stalls	the total stalls, a high absolute value indicates extreme congestion events
network flits	the total traffic, an indication of how busy the network is
nodes	number of nodes used for a run

This results in five metrics multiplied by five summary statistics each for twenty-five total statistics that were analyzed. Eventually we selected a subset of these statistics to be used as features in our model. Section IV-A discusses the feature construction and evaluation. Figure 2 shows the distribution of four key metrics from the above for each of the applications studied. The average flits/second from the processor tiles (top-left) defines the communication rate for the application as measured from NIC. The flits are averaged over the runtime of the application. The ratio of network and processor tile flits (top-right) defines the total flit traffic measured within the network versus the measured flits from the application at the NIC. This metric gives insight into the relative network utilization between the target application and all other applications running on the system during the same time period. The seventy-fifth percentile of stall-to-flit ratio (bottom-left) examines the relative congestion seen on all network tiles in tail of the distribution. The tail of the distribution emphasizes the importance of the extreme congestion occurring in the network which can have a more profound impact on the application synchronization stages. The ninety-fifth percentile of stalls (bottom-right) is an absolute measure of conditions where network data is ready but no network resources are available to send. Again, this extreme tail indicates cases where certain network links maybe be impacted that can cause significant effects on application synchronization stages.

The Pearson correlation coefficient, r, was used as a measure of the degree of correlation between an application's runtime and a network counter metric. Table IV presents the measured r values of three key metrics for each application. Table IV shows that for the MILC, Reorder, Nekbone, Qbox and LAMMPS applications there is at least one metric that has an $r > 0.6$. This indicates that the chosen metrics show correlation with runtime and thus can be used as a proxy to estimate the effects of variability caused by network congestion (details on the exact feature selection method used are provided in Section IV-A). The applications Nek5000 and Rayleigh do not show any significant correlation and we investigate if a combination of metrics can be used to deduce the effects of network congestion. To determine the number of samples needed for this evaluation, we ran an initial set of $\sim$500 MILC simulations in varying production environments and node placements on Theta. The runtimes and counter metric

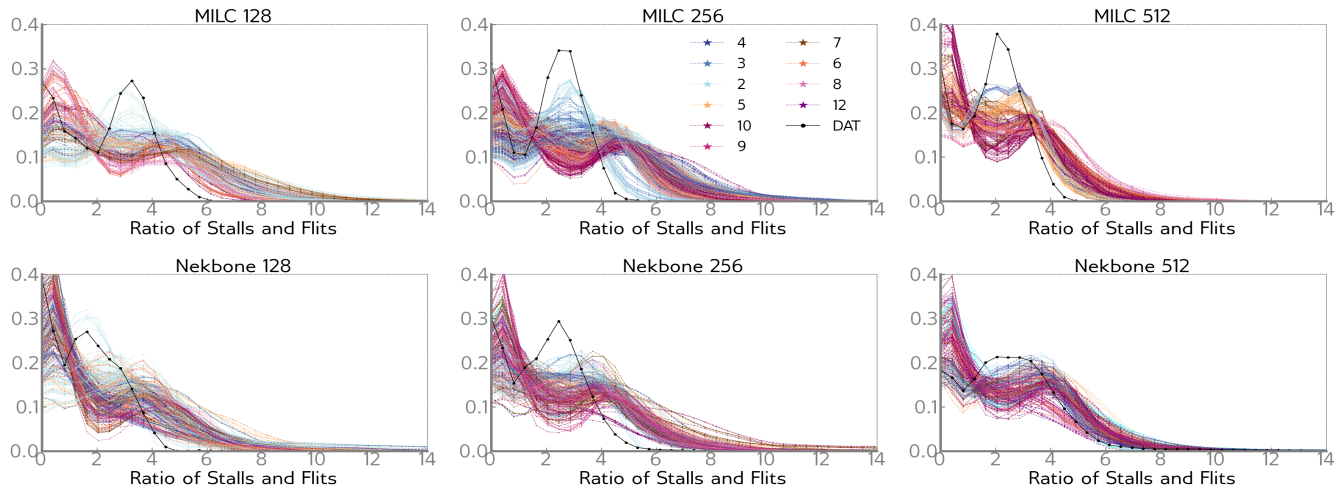

Fig. 3. Normalized PDFs for the stall-to-aggregated flit ratio on a network tile for MILC (top row) and Nekbone (lower row). Columns 1, 2, and 3 show data from runs using 128, 256, and 512, respectively. All production runs are represented by colored dotted lines and are colored according to the number of groups as shown in the legend. The DAT (Dedicated Access Time) runs are shown as a solid black line in each panel.

values for each run were recorded. The measured correlation coefficient converged in 250–300 runs, and therefore fewer than 500 runs were used for other applications.

TABLE IV. PEARSON CORRELATION COEFFICIENT, r, BETWEEN THE APPLICATION'S RUNTIME AND THE HARDWARE COUNTER STATISTIC GIVEN IN THE COLUMN HEADER FOR 256 NODE JOBS.

App	Ratio of Flits on networktiles and proctiles	Ratio of Stalls and Flits	Stalls (95 percentile)
MILC	0.65	0.69	0.79
Reorder	0.69	0.67	0.79
Nekbone	0.53	0.42	0.64
Nek5000	0.10	0.29	0.17
Qbox	0.61	0.17	0.72
Rayleigh	0.19	0.32	0.32
LAMMPS	0.56	0.84	0.76

E. Metric Analysis

Figure 3 shows the distribution of the normalized stall-to-flit ratio of the network tile for two applications (MILC and Nekbone) at three different job sizes. The colored lines on the graph represent data from a run with a particular node placement known as the dragonfly group size. The larger the number of dragonfly groups provide more global links for the application to use which implies greater bisection bandwidth. Every graph in the panel has the same scale. The two applications shown, MILC and Nekbone, have very different communication characteristics, as seen in figure 2. However, we can see from the panel of graphs that each case exhibits a very similar distribution. This critical finding is the key factor in forming the foundation to construct a generalizable model for estimating the variability of application runtime in congested network conditions. The extended tails indicate the higher congestion conditions that lead to the variability in runtime. These tails are present at the same scale, regardless of the application, the number of the nodes, or the job placement that was used.

F. Metric Evaluation

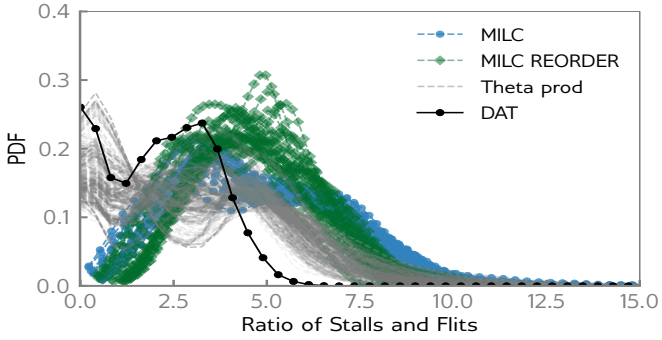

Fig. 4. PDF of the average stall to aggregated flit ratio for the MILC (blue circles) and MILC REORDER (green diamonds) applications. Production runs of MILC on Theta are shown as grey lines, and a DAT run of MILC is shown as a solid black line.

The evaluated network counter data were collected over the course of six months. The congestion in a production environment is non-deterministic since it depends on several factors, such as the other jobs running, the node placement for all jobs, and the routing schemes used. The collected test samples cover a wide range of possible network conditions, but to gain confidence that the metrics are within the possible extremes, a set of controlled congestion experiments were run. Two types of controlled experiments were performed in order to establish a baseline for the study and to determine bounds on potential congestion. In order to have an idea of the best possible run environment for an application, individual applications were run in an isolated condition where no other jobs were running on the system. These runs are referred as Dedicated Access Time (DAT) runs. In order to assess the maximal congestion seen by an application, multiple identical copies (jobs) of MILC were run on the whole machine, with each run configured to use a different node size combination. MILC was used to assess the maximal congestion as it is the application

with the highest communication intensity (refer Figure 2). For each of the individual MILC jobs, the performance counters were collected and analyzed to find the job with the greatest congestion. These runs are referred to as controlled congestion runs. Both sets of controlled environment runs were performed on Theta under whole-system reservations.

Figure 4 shows the controlled congestion normalized PDFs of the average stall-to-aggregated flit ratio for the MILC (blue circles) and MILC REORDER (green diamonds) applications run on 256 nodes. Production runs of MILC on Theta are shown as grey lines, and a DAT run of MILC is shown as a solid black line (similar to Fig. 3). The figure shows that the data from the production runs (grey) lie within the bounds of the minimal congestion DAT (black) run and the maximal congestion runs (blue and green). Clearly, the extended tails of grey distributions for the Theta production runs are less than maximum network congestion (blue and green distributions), indicating that the maximum congestion is greater than what is currently observed on Theta. These results provide confidence that our data captures the possible congestion conditions on Theta.

IV. Model Construction

A. Feature Engineering and Evaluation

In the previous section, we demonstrated empirically that the network performance counter-based metrics can be used as reliable indicators of network congestion. Based on this, we conjecture that these metrics can be used to estimate an application's sensitivity to the congestion. We construct a set of models that consider the multiplicity of features needed for accurately estimating the application sensitivity to congestion.

The statistics of our counter-based metrics are used as the feature set for building regression models that, given the feature set as input, can estimate the application impact due to congestion. The feature set is comprised of the statistics that represent congestion as well as those that represent the application communication characteristics. The list below provides all seven features listed in order of importance with the first being most important. The feature selection used in this study is based on impurity reduction technique using the Random Forest variable importance measures. The features such as "average stalls per networktile", "ratio of stalls to flits" and "average flits per networktile" represent the congestion. The features such as "average proctile flits per second" and "ratio of flits on network tiles to processor tiles" represent the application communication characteristics. The number of nodes used by the application run is also used as one feature, though it is not found to be dominant compared to other features.

1) Ratio of Average networktile Flits to Average proctile Flits
2) Average proctile Flits per second
3) 75th percentile of Ratio of networktile Stalls to Flits
4) 95th percentile of networktile Stalls
5) Average networktile Flits
6) Average networktile Stalls
7) Number of Nodes

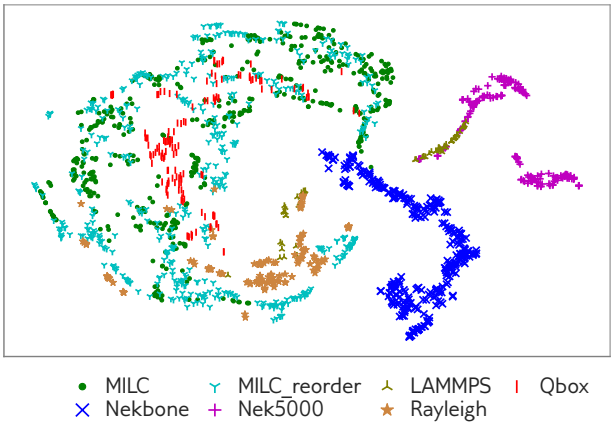

Fig. 5. TSNE plot for all applications with 256 nodes.

Figure 2 shows the distribution by application for each of the top four features of the model. The features "ratio of flits on network tiles to processor tiles" and "average proctile flits per second" are the two most dominant features used by the model. These features demonstrate that there exists common overlap between applications, implying that a general model can be formed but also showing that distinct application characteristics are captured.

When the different features have a wide range of scales, Standardization of feature dataset is essential for robust operation of many machine learning estimators. Given the huge diversity of scale in the features ($\mathcal{O}(1 - 10^{11})$), the feature data are scaled by removing the median and scaling the data according to the 25-75$^{\text{th}}$ quantile range. The scaler for each feature removes the median and scales the corresponding feature data according to the interquartile range. The target space is constructed via the application runtimes normalized with the Z-score normalization. Since different applications have different runtimes, the runtime is normalized (standardized) by removing the mean and scaling to unit variance.

B. Comparison of Application Feature Spaces

Given that the feature space is a 7-dimensional datum, we use the t-Distributed Stochastic Neighbor Embedding (t-SNE) for visualizing the feature distributions across the seven applications. The t-SNE is a nonlinear dimensionality reduction algorithm used for mapping high-dimensional space to 2D space suitable for human observation.

Figure 5 shows the feature data for the seven applications used in this study. While the feature spaces for some applications such as MILC, MILC Reorder, Qbox and Rayleigh overlap, the feature spaces for Nekbone and Nek5000 are quite distinct from other applications.

Figure 6 shows the how the four dominant features (the standardized counter metrics) are related to the normalized runtime for four applications. Results from the jobs using 128, 256, and 512 nodes are shown as light, medium, and dark blue circles for MILC; light, medium, and dark red

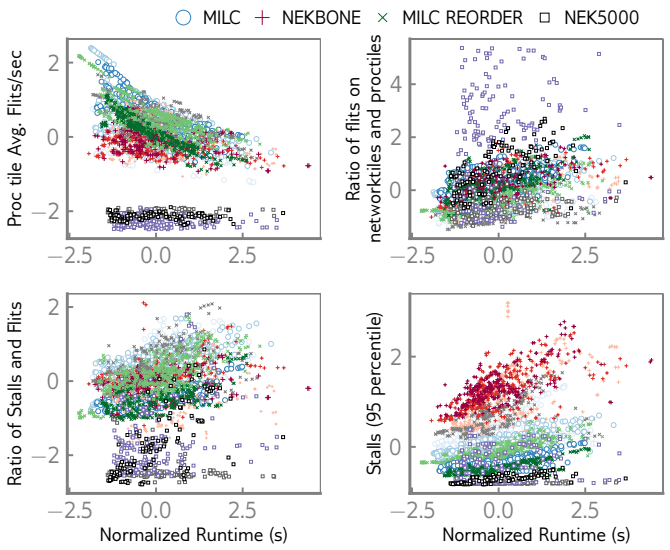

Fig. 6. Normalized runtime for the four dominant features for the applications.

pluses for Nekbone; grey, medium, and dark green 'x'es for MILC REORDER; and grey, purple, and black rectangles for Nek5000, respectively. Clearly, once the data is normalized, the counter metrics display significant overlap across the applications. Because of this overlap, a model that can predict the effects of network congestion can be built by only using these local network counter based metrics. As observed in Figure 5, Figure 6 also shows the Nek5000 application demonstrating distinct communication characteristics from the rest of applications.

C. Regression methods used to build a Model

We have experimented with various regression models such as RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, and Support Vector Regression (SVR). The Python machine learning package SCI-KIT LEARN [20] was used to train several models. We first present the results from the SVR model which performed and generalized the best.

In SVR, the goal is to find a function that has at most ϵ deviations from the target objects y_i. Here, the target is standard deviation from the mean runtime for all applications. The *rbf* kernel is used with the SVR model including the hyperparameters, C and γ. The C parameter is a regularization parameter that controls the trade off between the amount up to which deviations larger than ϵ are tolerated and the ability to generalize the model to unseen data [21]. γ is a parameter of the *rbf* kernel and can be thought of as the spread of the kernel and therefore the decision region. The hyperparameter tuning is performed using the *gridSearchCV* method. To measure the prediction accuracy of the model, we use the R^2 coefficient of determination metric as a scoring function.

Figure 7 shows the performance of the SVR model for the data set consisting of all the seven applications: MILC, MILC REORDER, Nekbone, Nek5000, LAMMPS, Qbox, and Rayleigh. The figure shows the proximity of the predicted runtime to the actual runtime. The dashed black line in the figure indicates a perfect prediction of runtime. The left (right) blue shaded regions represent areas where both the actual and predicted runtime are less (greater) than the mean, which would indicate that the application ran in a congestion light (heavy) environment. The prediction results are from the case where 75% of the data is used for training and 25% of data is used as a test set.

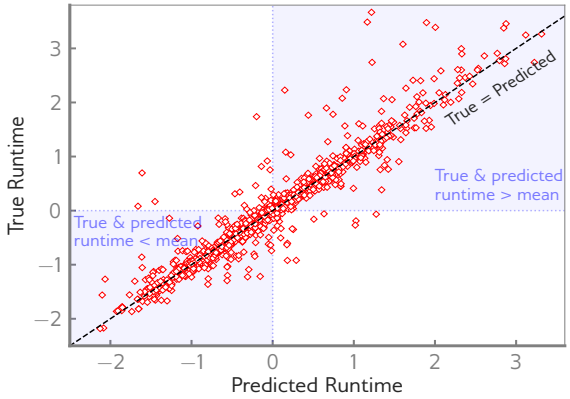

Fig. 7. Predicted versus true (actual) normalized runtimes for the dataset with the seven applications.

The model is validated using the data from the seven applications. Initially, the 10-fold cross-validation was used to determine the hyperparameters of the SVR model. These hyperparameters were used for all the models used in this study. The model validation performed using 10-fold cross-validation shows a score of $R^2 = 0.91$. The training and the cross-validation scores for the model as a function of the number of samples is shown in the Figure 8. Given that around 2000 samples are required for improved accuracy, constructing generic models as performed in this work, as opposed to building application-specific models, is the feasible approach.

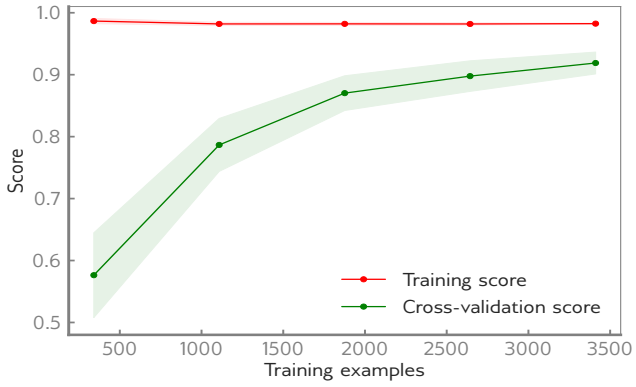

Fig. 8. SVR model training and cross-validation scores with increasing training samples.

The Jupyter notebooks and summary statistics input files

are open sourced and available online. This material provides a detailed information about the features, exact model, and training method used to reproduce all of the model and feature data presented. The next section provides more details on generalization of the model.

V. Generalization of the Model

A. Accuracy versus Training Size

Using the hyperparameters determined building the regression model, we examine accuracy using different ratios of the data used for training and testing. When the data is split into 75% training and 25% testing, the model performs well, predicting normalized application runtimes with an $R^2 = 0.91$. When using the train-test split as 50% each, the model's accuracy is reduced to $R^2 = 0.83$. When using the train-test split as 25%-75%, the accuracy is again reduced to $R^2 = 0.79$. Using only 10% of the data to train, an accuracy of $R^2 = 0.62$ is achieved. A model trained with only 25% of the data still achieves a reasonable accuracy of $R^2 = 0.79$.

B. Accuracy on an Unseen Application

The generalization of the model can be evaluated by testing the performance of the model on data from an application (unseen) that is not present in training set. We consider the case where the unseen application is similar to an existing application in the training set. MILC REORDER was used as the unseen application, with the model trained on the remaining six applications. When using the 512 node runs of MILC REORDER as the test set, the model shows an accuracy of $R^2 = 0.82$. Figure 9 shows the prediction quality of the model with using MILC REORDER jobs (three different node sizes) as the unseen data. As expected the model performance is reasonable because the model was trained on MILC which shows similar communication characteristics as MILC REORDER.

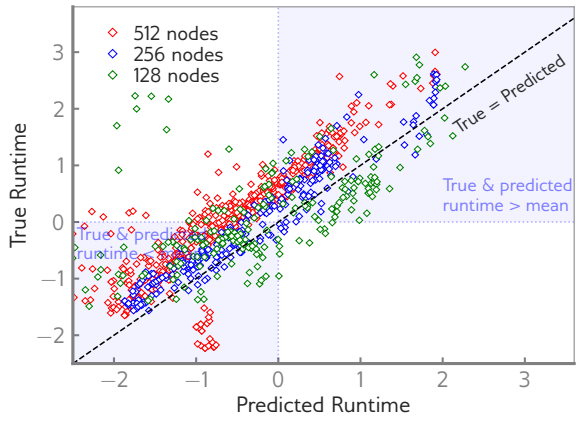

Fig. 9. Predicted versus true runtimes for target application - MILC reorder.

C. Accuracy with Sparse Samples

In the second test, the Nekbone application is used as an application with a limited set of samples used for retraining the model. The model was trained on the other six applications and also includes only 40 random samples of Nekbone. The model shows an accuracy of $R^2 = 0.52$. Figure 10 shows the prediction quality of model using Nekbone jobs with three different node sizes as the unseen data. The poor performance of the model is explained by noting the unique characteristics of Nekbone compared to rest of the applications as shown in Figure 5. This demonstrates that the core training set with a limited number of samples can being to provide reasonable results even for applications not initially represented in the training set.

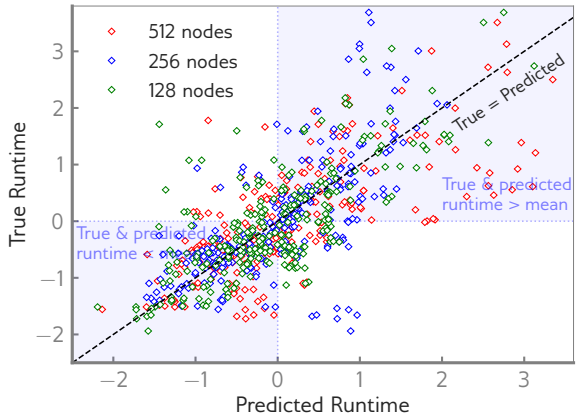

Fig. 10. Predicted versus true runtimes for target application - Nekbone.

As shown in Table V, the prediction accuracies are low for models constructed with Nekbone, Nek5000, and LAMMPS as unseen applications. These applications, especially the specific input problem cases used here, exhibit distinct communication characteristics that were not covered by the rest of the applications. The accuracies for the models that use any of the applications MILC, MILC reorder, Qbox, and Rayleigh as unseen application are good. These four applications exhibit overlapping feature space, as shown in Figure 5, potentially possessing some common communication characteristics. These results indicate that the model is able to accurately estimate the normalized runtimes of applications with these characteristics. We see that with small numbers of samples, the model rapidly improves in accuracy. By including 20 samples of MILC REORDER into the training set, the model's accuracy is increased to $R^2 = 0.92$.

D. Robustness Improvements

In order to identify conditions where the model does not have sufficient training data and to avoid reporting mispredictions, the model would report the predictions only when there is sufficient confidence. The SVR model can only provide the prediction as a single point estimate and does not provide any estimate of the error bounds. However, a model based on Random Forest can provide both the point estimate and the error bounds of the prediction.

Thus, we also train a Random Forest (RF) regression model that can report the estimations as well as the corresponding estimation intervals. Figure 11 shows the predicted standard deviation including all the estimators using the Random Forest

TABLE V. ACCURACIES FOR SVR MODEL TESTED ON DIFFERENT TARGET APPLICATIONS.

Target App	Training	Accuracy with 20 runs of Target App	Accuracy with 40 runs of Target App
MILC	(All w/o MILC)	0.94	0.97
Reorder	(All w/o Reorder)	0.92	0.96
Nekbone	(All w/o Nekbone)	0.21	0.52
Nek5000	(All w/o Nek5000)	-0.09	0.19
LAMMPS	(All w/o LAMMPS)	0.12	0.44
Rayleigh	(All w/o Rayleigh)	0.34	0.63
Qbox	(All w/o Qbox)	0.63	0.87

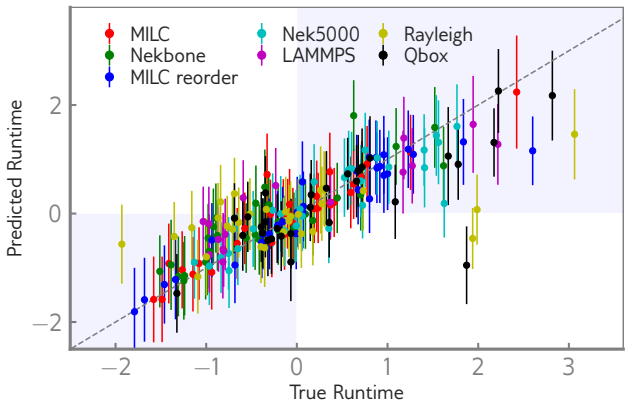

Fig. 11. Random Forest Model estimation and estimation intervals.

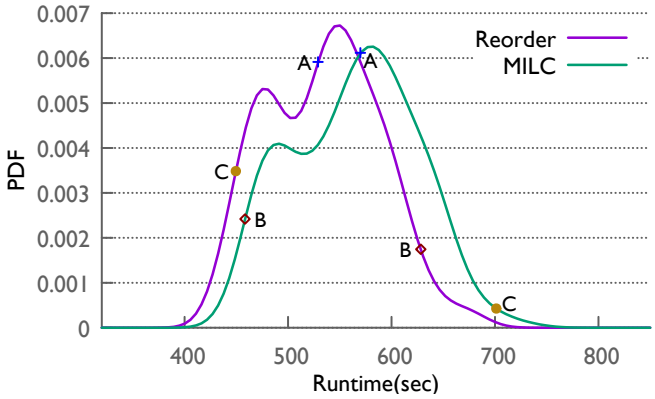

Fig. 12. MILC and MILC Reorder runtimes in seconds for 256 node runs.

model. The combined model compares the two models using a heuristic to determine if the prediction produced is of high quality. If we compare the SVR prediction with the RF estimation intervals, 90% of the predicted SVR values fall within the RF estimation intervals. If the SVR prediction value is within the range (RF prediction +/- error bounds), meaning the absolute difference between the predicted values of SVR and RF is less than RF error range, then we use the (SVR prediction +/- the error bounds of RF) for reporting the *Variability Estimate*. Otherwise, the combined model will report that no prediction can be provided. We evaluate this heuristic on the models trained from the data of all seven applications. When the value predicted by SVR is within RF error bounds, the measured value (True) is within the error bounds for 96.1% of the data points. However, the values predicted by SVR not within the RF error bounds are only 9.8% of the data points, and the combined model will not report the predictions for these cases.

VI. MODEL USE CASES

We now describe how the model (described in the previous section) could be used in practice with a few concrete examples. To illustrate the use of model, we use the samples from applications, MILC and MILC Reorder, running on 256 nodes. The PDFs for MILC and Reorder are shown in the Figure 12. We choose three runs from MILC and MILC Reorder for the use cases. Table VI lists the *Variability Estimate* that is generated by the model for each for these runs. The corresponding actual runtimes are marked in Figure 12.

TABLE VI. PREDICTIVE MODEL USE CASE: MILC PERFORMANCE TUNING.

RUN	MILC		REORDER	
	Runtime	Variability Estimate	Runtime	Variability Estimate
A	569.05	-0.05 (+/- 0.58)	529.73	-0.07 (+/- 0.42)
B	458.13	-1.56 (+/- 0.76)	626.86	1.26 (+/- 0.65)
C	703.09	2.54 (+/- 0.90)	449.33	-1.50 (+/- 0.49)

A. Performance Benchmarking

A user is interested in benchmarking the performance of the MILC application to establish a baseline for comparison to other machines, for setting the walltime or for use in scaling studies. Let us assume the users's first benchmark run is Run C at 703.09 seconds. The *Variability Estimate* for this run is 2.54 ± 0.90 which indicates the run experienced high congestion and is not characteristic of an average run on the system. Run C should not be used as a benchmark time. The user runs again and the time for Run A, 569.05 seconds, is obtained. Examining the *Variability Estimate* for Run A, -0.05 ± 0.58, shows that the estimate is near zero and represents average network conditions. Run A is suitable as a benchmark and is representative of the expected average runtime for MILC.

B. Performance Tuning

A common use case users might want to perform is code optimization. In order to understand if the modified code is performing better, it is necessary to understand both the average runtime of the original code and the average runtime

of the modified code. This can require a large number of runs if the user has no additional information about each run. As an example, the MILC Reorder application is identical to MILC but has an optimized MPI rank placement which reduces off-node communication. From II, we see that MILC Reorder has improved performance with an average runtime of 93%, $528.7/566.2 = 0.93$, of MILC.

Let us assume the user were to run MILC and obtain the runtime of Run B, 458.13 seconds, and run MILC Reorder and obtain the runtime of Run B, 626.86 seconds. The user may erroneously believe that MILC is faster than MILC Reorder. However, with using the model, the user will have the *Variability Estimate* for each of these runs and will see that MILC ran in very favorable network conditions, -1.56 ± 0.76 while MILC Reorder ran in very unfavorable network conditions, 1.26 ± 0.65. These two runs should not be compared. If the runtimes for Run A are obtained for both MILC and MILC Reorder, both of these have a *Variability Estimate* which is near zero indicating average network conditions. Comparing the runtimes, we see MILC Reorder is 93% of the runtime of MILC, $529.73/569.05 = 0.93$.

In essence, in production environments that experience potential network congestion effects, mere comparison of the runtimes does not provide an accurate attribution of performance tuning benefits. The *Variability Estimates* reported by model augmented with the runtimes can be used to differentiate between the tuning benefits and the congestion effects on an application performance.

VII. Related Work

Application performance can be significantly impacted by node-level performance variability due to OS noise effects [22], dynamic frequency scaling effects [23], shared cache contention [24], and power consumption variability [25]. Effective mitigation strategies such as core specialization [26], power capping and frequency selection [27] are some approaches to address variability at this level. Many researchers have studied application runtime variability effects due to various system-scale factors such as shared network and I/O in production HPC systems [5], [28] and commercial data centers [29]. Network resource contention has been identified as the prime contributor for run-to-run variability at system scale. Several studies [4], [5], [10] have observed that shared network resource contention is the major contributing factor for variability on dragonfly networks. They have also highlighted that the effects of node placement on the variability were not dominant compared with network congestion effects.

Jain et al. [6] proposed fabric isolation schemes to address the inter-job effects in dragonfly networks, and they quantified the benefits through simulation. While simulator-based studies [9], [30] can provide valuable insights for exploring the design decisions in future system architectures, simulators are most often built on top of general-purpose models (such as LogP) and thus often cannot represent all possible scenarios in a production system. For example, the adaptive routing schemes implemented on Cray XC [7] systems are not public and hence are not represented in any existing simulator. The varying sitewide scheduling policies, which could influence node placement and thus performance, are also not straightforward to represent within a generic simulator.

Tools have been developed to quantify the extent of inter-job effects in HPC systems. For instance, the tool mentioned in [31] measures the effects of inter-job contention by continually running the benchmarks and provides insights into using network performance counter data. Benchmarks on production systems are used to measure and understand variability [3], [32]; however, benchmarks often do not represent the characteristics of the production applications. Monitoring tools such as LDMS (Lightweight Distributed Metric Service) [33] were developed to obtain system-wide resource utilization information on production HPC systems. While such tools help monitor global events such as network congestion [18], they do not have access to the application context and thus cannot correlate application effects with network effects [34]. This is in contrast to our work where variability is estimated by using a local view of congestion captured through network performance counters.

VIII. Conclusions

Network resources shared across multiple competing jobs can potentially lead to unfair resource provisioning to some applications, thus impacting their performance. Given this lack of an assured provisioning of resources, the runtimes of an application can vary from run to run. The application runtime by itself is insufficient to conjecture anything of a specific execution, such as its performance drop due to network congestion. The effects of network congestion can be lead to variability between runs up to 74% depending on the application and environment. With this level of effect, users require an approach to understand the effects of variability in order to evaluate application performance and scaling.

A prototype software package was developed to provide potential users with an estimate including error bounds of the impact of network congestion on their application run. A set of metrics was developed and analyzed which demonstrate the ability to measure network congestion as set of probability distributions. We validated that these distributions are effective in measuring the specific network congestion that affected an application. Using these metrics, a set of seven features were constructed and a combined SVM and RF model was trained using well-known HPC applications that are commonly run at many HPC facilities. The resulting models show a prediction quality of 91% based on the 10-fold cross-validation score. We propose this model trained with a diverse set of application communication patterns as a basis for building a general model for use in production systems. Any HPC interconnect that provides similar network counters that measure traffic and congestion will be able to adapt our work to their systems. This enables the entire HPC community to benefit from the metrics, training techniques and data collected considering that network congestion impacts on application performance is prevalent in all the current-generation networks.

ACKNOWLEDGEMENT

This research used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357. Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

[1] A. Porterfield, S. Bhalachandra, W. Wang, and R. Fowler, "Variability: A tuning headache," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1069–1072.

[2] J. Chen and J. Revels, "Robust benchmarking in noisy environments," in *20th Annual IEEE High Performance Extreme Computing Conference*, Aug 2016.

[3] T. Groves, Y. Gu, and N. J. Wright, "Understanding performance variability on the Aries dragonfly network," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 809–813.

[4] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs," *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 41:1–41:12, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2503210.2503247

[5] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran, "Run-to-run variability on Xeon Phi based Cray XC systems," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis - SC '17*, pp. 1–13, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3126908.3126926

[6] N. Jain, A. Bhatele, X. Ni, T. Gamblin, and L. V. Kale, "Partitioning low-diameter networks to eliminate inter-job interference," in *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium, IPDPS 2017*. IEEE, 5 2017, pp. 439–448.

[7] B. Alverson, E. Froese, and D. Roweth, "Cray XC Series network," *Cray*, pp. 1–28, 2012. [Online]. Available: www.cray.com

[8] A. Bhatele, N. Jain, W. D. Gropp, and L. V. Kale, "Avoiding hot-spots on two-level direct networks," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 2011, pp. 1–11.

[9] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P. T. Bremer, "Analyzing network health and congestion in dragonfly-based supercomputers," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 93–102.

[10] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully!: Job interference study on dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 64:1–64:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014990

[11] (2016) MILC collaboration. [Online]. Available: http://www.physics.utah.edu/ detar/milc/

[12] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier, "Nek5000 web page," *Web page: http://nek5000.mcs.anl.gov*, 2008.

[13] N. A. Featherstone and B. W. Hindman, "The spectral amplitude of stellar convection and its scaling in the high-Rayleigh-number regime," *The Astrophysical Journal*, vol. 818, no. 1, p. 32, 2016. [Online]. Available: http://stacks.iop.org/0004-637X/818/i=1/a=32

[14] F. Gygi, "Architecture of Qbox: A scalable first-principles molecular dynamics code," *IBM Journal of Research and Development*, vol. 52, no. 1.2, pp. 137–144, Jan 2008.

[15] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, Mar. 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1039

[16] C. Inc, "Aries hardware counters," *Cray Doc S-0045-20*, 2017. [Online]. Available: http://docs.cray.com/books/S-0045-20/S-0045-20.pdf

[17] A. Deconinck, A. Bonnie, K. Kelly, S. Sanchez, C. Martin, M. Mason, J. Brandt, A. Gentile, B. Allan, A. Agelastos, M. Davis, and M. Berry, "Design and implementation of a scalable network monitoring system for Trinity," *Cray User's Group*, 2016.

[18] A. Deconinck, H. Nam, D. Morton, A. Bonnie, C. Lueninghoener, J. Brandt, A. Gentile, K. Pedretti, A. Agelastos, C. Vaughan, S. Hammond, B. Allan, M. Davis, and J. Repik, "Runtime collection and analysis of system metrics for production monitoring of Trinity Phase II," *Cray User's Group*, 2017.

[19] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Tools for High Performance Computing 2009*, M. S. Müller, M. M. Resch, A. Schulz, and W. E. Nagel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078195

[21] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, Aug 2004.

[22] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj, "Benchmarking the effects of operating system interference on extreme-scale parallel machines," *Cluster Computing*, vol. 11, no. 1, pp. 3–16, Mar. 2008.

[23] B. Acun, P. Miller, and L. V. Kale, "Variation among processors under turbo boost in hpc systems," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016, pp. 6:1–6:12.

[24] A. Sandberg, A. Sembrant, E. Hagersten, and D. Black-Schaffer, "Modeling performance variation due to cache sharing," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 155–166.

[25] B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal, "Accurate characterization of the variability in power consumption in modern mobile processors," in *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems*, ser. HotPower'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 8–8.

[26] H. Pritchard, D. Roweth, D. Henseler, and P. Cassella, "Leveraging the Cray Linux environment core specialization feature to realize MPI asynchronous progress on Cray XE systems," in *Proceedings of the Cray User Group Conference*, 2012.

[27] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 78:1–78:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807638

[28] E. C. Inacio, P. A. Barbetta, and M. A. R. Dantas, "A statistical analysis of the performance variability of read/write operations on parallel file systems," *Procedia Computer Science - Special Issue: International Conference on Computational Science, ICCS 2017*, vol. 108, pp. 2393–2397, 2017.

[29] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with Heracles," *ACM Transactions on Computer Systems*, vol. 34, no. 2, pp. 1–33, 5 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2912575.2882783

[30] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim: Simulating large-scale applications in the LogGOPS model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 597–604. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851564

[31] R. E. Grant, K. T. Pedretti, and A. Gentile, "Overtime," in *Proceedings of the 3rd Workshop on Exascale MPI - ExaMPI '15*, 2015, pp. 1–10.

[32] N. J. Wright, S. Smallen, C. M. Olschanowsky, J. Hayes, and A. Snavely, "Measuring and understanding variation in benchmark performance," in *Proceedings of the 2009 DoD High Performance Computing Modernization Program Users Group Conference*, ser. HPCMP-UGC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 438–443.

[33] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The Lightweight Distributed Metric Service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, vol. 2015-Janua, no. January, 2014, pp. 154–165.

[34] A. Bonnie, M. Mason, and D. Illescas, "Monitoring infrastructure: the challenges of moving beyond petascale," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 9 2017, pp. 785–788. [Online]. Available: http://ieeexplore.ieee.org/document/8049017/

APPENDIX

A. Abstract

This artifact contains links to the source code for the scientific applications used in this study along with the corresponding input datasets and the build instructions. It also describes the specific experimental setup for the controlled environment runs. Moreover, we provide scripts and Jupyter notebooks for processing the data, generating the summary statistics, and running the predictive model. The instructions and the code should help in replicating the results we have presented in this study. Although the range of variability is not exactly reproducible on a system different from Theta, since it depends on several factors such as workload characteristics, node placement, and interconnect used, the basic approach should still be relevant and applicable to different systems.

B. Description

1) Checklist (artifact metainformation):

- **Applications:** The application codes are open source and thus can be accessible publicly.
- **Scripts:** The PAPI-based code for counter measurement; scripts for gathering node placement information; and a set of Jupyter notebooks for statistical analysis that are used to build the model.
- **Compilation:** CC (Cray wrapper around Intel compiler [intel 17.0.4.196])
- **Runtime environment:** Cray MPI (cray-mpich:7.6.0), Cray OS (SLES 12)
- **Hardware:** Argonne Theta (Intel Xeon Phi 7230) and NERSC Cori (Intel Xeon Phi 7250) systems.
- **Execution:** Details in the run scripts
- **Output:** Run time and a output of the from the model
- **Experiment workflow:** Download source code, build the tools, run the applications, collect performance counters, parse the output files, and analyze the summary. See below for more detailed information.
- **Publicly available?:** Yes

2) How delivered: The artifact content (source code, predictive mode [a set of Jupyter notebooks], and various other processing and plotting scripts used in this study) will be hosted on a publicly accessible github web site.

3) Hardware dependencies: The experiments shown were performed on Argonne's Theta and NERSC's Cori supercomputers, both are based on Cray XC systems. Theta consists of Intel Xeon Phi 7230 nodes interconnected with a Cray Aries in a Cray Cascade dragonfly topology. Cori consists of two partitions, one with Intel Xeon Haswell processors and another with Intel Xeon Phi 7250 processors, all on the same Cray Aries network in a Cray Cascade dragonfly topology.

Cray XC systems are constructed from four node blades with each blade having a single Aries network router. Aries NIC is connected to a node over a 16X PCI-e Gen3 interface. Each chassis has 16 such 4-node blades and each cabinet has three chassis (192 nodes). The dragonfly network is constructed from two-cabinet electrical groups with 384 nodes per group. So, essentially, 1 blade - 4 nodes; 1 chassis - 16 blades; 1 group - 6 chassis.

The topology used is a three-level dragonfly topology. The first two levels (rank-1 and rank-2) are copper based with 10.5 GB/s peak bi-directional bandwidth per link. The chassis backplane provides connectivity in the rank-1 (green) dimension; each Aries is connected to each of the other 15 Aries in the chassis. The rank-2 (black) dimension is formed by connecting three links from each Aries to its peer. The rank-3 level, connectivity between the groups, is optical with 9.38 GB/s per link. The node placement for a job on Dragonfly is arbitrary, and perfect isolation of traffic is not possible. Hence, while a 256 node job could in principle be placed within a single group, there is no guarantee that it would not be spread across multiple groups.

Only flat-quad memory mode of KNLs was used in all the experiments to avoid the variability effects due to the MCDRAM cache-mode effects possible with the cache-quad mode.

Although PAPI has limitations on the number of counters that can be read in compute processors such as Intel's Skylake, there are no such limitations with reading the large selection of Aries router counters. Two basic categories of hardware performance counters are available to the user: NIC counters and router tile counters. Each Aries router has more than 1,000 counters; many of these differ only in the router tile and virtual channel. We used around 384 counters per process in our study. In addition, some counters are configurable, meaning not only can they be read but also written to, in order to configure and control data collection. Essentially, all the network counters can be collected in each run. The reading and configuring can be accomplished by using PAPI. For a complete list of the available Aries counters, use the papi_native_avail command.

4) Software dependencies: All benchmarks were built by using Cray's compiler wrappers with Intel compiler and other default parameters. Theta uses the Cobalt job scheduler, and Cori uses Slurm. All software dependencies specific to each benchmark will be noted in a README file within each subdirectory.

5) Datasets: A brief description of the applications used in this study is provided here. MILC is a MIMD lattice computation code for simulation of high energy and nuclear physics, such as the study of the mass spectrum of strongly interacting particles and the weak interactions of these particles.

MILC REORDER is MILC with a simple optimization applied to optimize the rank-to-node mapping. This was done using Cray's grid_order tool with a $4 \times 4 \times 2 \times 2$ subgrid.

Nek5000 is a computational fluid dynamics code with a high-order, incompressible Navier-Stokes solver based on the spectral element method designed for large eddy simulation and direct numerical simulation of turbulence in complex domains.

Nekbone is a miniapp that exposes the principal computational and communication kernels of Nek5000, which solves

a standard Poisson equation in a 3D box domain with a block spatial domain decomposition.

LAMMPS is an open-source molecular dynamics code written in C/C++ to simulate systems spanning several science domains (e.g., liquids, biomolecules, materials, and mesoscopic systems). The benchmark examined was a water droplet containing 87 million particles modeled by using a short-range forcefield with in situ analyses included.

Rayleigh is a 3D convection code that evolves the incompressible and anelastic magnetohydrodynamics equations in spherical geometry by using a pseudo-spectral approach.

Qbox is an ab initio molecular dynamics code written in C based on density functional theory using the plane waves and pseudopotentials formalism for electronic structure calculations.

A brief descriptions of the input decks used, and processes per node used are provided in the table below:

Application	Description of Input	Processes per Node
MILC	su rmd 16,384 grid points per process	64 MPI
Nekbone	511 spectral elements per process; 8^{th} poly order	63 MPI
HACC	3200^3 grid	64 MPI
NEK5000	Engine Case	64 MPI
LAMMPS	Water Droplet	64 MPI x 2 OpenMP
Rayleigh	Christensen et al. 2001 (MHD, Case 1)	64 MPI
Qbox	SiC512	64 MPI

The specific input data files and the run scripts used are provided in the Github site.

C. Installation

Each application used in the paper has a separate directory that will contain source code, build scripts, run scripts, and output parsing scripts (if required). Since these experiments are designed primarily for Theta, the build commands provided give examples that should be adapted if they were to be used on other systems.

D. Experiment Workflow

Each benchmark will have a "./run.sh" script that details how to run the application on Argonne's Theta system. Additionally, a README file will explain the various options of the run script and output. A prototype implementation of the performance counter (Aries counters used in this study and the memory-related counters) monitoring tool exists and is integrated as a submodule of the Darshan package.

A few short-hour system reservations are used for conducting the controlled experiments. For establishing the lower-bound on the potential congestion, one application instance was run in a contiguous partition on the system; This lower bound helps establish the intra-application contention because no other interference is possible on the system. For establishing an upper bound on the potential congestion, multiple instances of the same application are run on the system with each instance capturing respective performance counters. In this run, all the application instances start execution at the same time by having appropriate idle times before the application executions.

The scripts and software can be used directly on a Cray XC system. The data capturing tools could be substituted for similar tools by Mellanox to capture PortXmitWait, PortXmitPackets, etc. on Infiniband networks and then processed into the same statistical format for processing and training.

E. Evaluation and Expected Results

The key takeaway in this paper is an approach to establish the correlation between the application runtime and job-specific network performance counters, and use that insight to assess application sensitivity to predicted congestion. This idea should hold true for any production network, although the exact procedure to measure the counters varies and may be non-portable for a non-Cray system. However, any system that has similar counters to Stalls and Flits should be able to adapt the techniques mentioned in this study. Once the correlation of counters and runtime is established on a new machine, the derived statistics developed in this study can be used to develop predictive models to assess the variability. The predictive model is an invaluable for application developers to optimize codes in these variable run environments. On a similar Cray system with a different application workload set, the model accuracy depends on the diversity of the application set used in the training.

There is no specific requirement for using a specific number of applications, we have used six applications only to demonstrate the entire approach of model building and prediction as a proof-of-concept.