Why machine learning for HPC?

- Proliferation of performance data
  - On-node hardware counters
  - Switch/network port counters
  - Power measurements
  - Traces and profiles

- Supercomputing facilities’ data
  - Job queue logs, performance
  - Sensors: temperature, humidity, power
Types of ML-related tasks in HPC

• Auto-tuning: parameter search
  • Find a well performing configuration

• Predictive models: time, energy, …
  • Predict system state in the future
  • Time-series analysis

• Identifying root causes/factors
Network congestion

• Responsible for performance degradation, variability and poor scaling

• Congestion and its root causes not well understood

• Study network hardware performance counters and their correlation with execution time

• Use supervised learning to identify hardware components that lead to congestion and performance degradation
Life of a message packet
Life of a message packet

Processor
Injection Memory FIFOs (per task)

Router
Injection Network FIFOs (per node)

Source node

On the network
Life of a message packet

Source node

Intermediate node

Processor
Injection Memory FIFOs (per task)

Router
Injection Network FIFOs (per node)

On the network

Router
Intermediate buffers

Reception
Injection
Life of a message packet

Intermediate node

Source node

Processor

Injection Memory FIFOs (per task)

Injection Network FIFOs (per node)

Router

On the network

Router

Intermediate buffers

Reception

Injection

On the network
Life of a message packet

Source node -> Processor

Router

Intermediate node

Router

Destination node -> Processor

Reception Memory FIFOs (per task)

Reception Memory FIFOs (per node)

Injection Memory FIFOs (per task)

Injection Network FIFOs (per node)

Injection Network

Intermediate buffers

Life of a message packet
Life of a message packet

- Source node
  - Injection Memory FIFOs (per task)
  - Injection Network FIFOs (per node)

- Intermediate node
  - Router
    - Intermediate buffers
    - Injection
    - Reception

- Destination node
  - Router
    - Reception Network FIFOs (per node)
  - Processor
    - Reception Memory FIFOs (per task)

### Hardware resource

- Source node
- Network link
- Intermediate router
- All
Gathering data for machine learning

- Collect network hardware counters data on IBM Blue Gene/Q and use a functional simulator

- Use Rubik task mappings to get a range of execution times for the same application
Gathering data for machine learning

- Collect network hardware counters data on IBM Blue Gene/Q and use a functional simulator

<table>
<thead>
<tr>
<th>Hardware resource</th>
<th>Contention indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source node</td>
<td>Injection FIFO length</td>
</tr>
<tr>
<td>Network link</td>
<td>Number of sent packets</td>
</tr>
<tr>
<td>Intermediate router</td>
<td>Receive buffer length</td>
</tr>
<tr>
<td>All</td>
<td>Number of hops (dilation)</td>
</tr>
</tbody>
</table>

- Use Rubik task mappings to get a range of execution times for the same application

---

Abhinav Bhatle
In this section, we describe the process of gathering and preparing data for machine learning. We use applications like heavy applications, MILC and pF3D. We demonstrate our technique using various communication kernels and applications we use, and the step-by-step preparing the input data for machine learning, the community can also be an important indicator of congestion. The number of hops a message travels, also referred to as its route, increases the chance of network congestion. So, the intermediate component that a message passes through along cause congestion when these buffers become full. Finally, each packet is temporarily in receive buffers. Stalled packets may cause congestion due to link contention.

At the source node, a message is split into several packets. These pass through various hardware components, any or all of which can delay the communication [10]. We briefly explain the hardware components and the measurements that we would need to evaluate contention on each of them (see Table I).

We analyze the relative impact of different features on predicting execution times to identify hardware components that contribute the most to network congestion (Table I), we gather communication data from three network hardware counters: the number of packets sent on each link, the receive buffer length and the number of sent packets. All these indicators contribute to network congestion. All the experimental data for this study has been collected on Vulcan, an IBM Blue Gene/Q installation at LLNL. We use Rubik [14] to generate many different task mappings of the code running on a 5D torus.

The goal of this paper is to find correlations of network congestion and their corresponding indicators. The table below shows the features that we use as input to the machine learning model.

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg dilation AO</td>
<td>Avg. dilation of average outliers (AO)</td>
</tr>
<tr>
<td>max dilation</td>
<td>Maximum dilation</td>
</tr>
<tr>
<td>sum dilation AO</td>
<td>Sum of dilation of AO</td>
</tr>
<tr>
<td>avg bytes</td>
<td>Avg. bytes per link</td>
</tr>
<tr>
<td>avg bytes AO</td>
<td>Avg. bytes per link for AO</td>
</tr>
<tr>
<td>avg bytes TO</td>
<td>Avg. bytes per link for top outliers (TO)</td>
</tr>
<tr>
<td>max bytes</td>
<td>Maximum bytes on a link</td>
</tr>
<tr>
<td>#links AO bytes</td>
<td>No. of AO links w.r.t. bytes</td>
</tr>
<tr>
<td>avg stalls</td>
<td>Avg. receive buffer length</td>
</tr>
<tr>
<td>avg stalls AO</td>
<td>Avg. receive buffer length for AO</td>
</tr>
<tr>
<td>avg stalls TO</td>
<td>Avg. receive buffer length for TO</td>
</tr>
<tr>
<td>max stalls</td>
<td>Maximum receive buffer length</td>
</tr>
<tr>
<td>#links AO stalls</td>
<td>No. of AO links w.r.t. recv buffer length</td>
</tr>
<tr>
<td>avg stallspp</td>
<td>Avg. number of stalls per rcv’d packet</td>
</tr>
<tr>
<td>avg stallspp AO</td>
<td>Avg. no. of stalls per packet for AO</td>
</tr>
<tr>
<td>avg stallspp TO</td>
<td>Avg. no. of stalls per packet for TO</td>
</tr>
<tr>
<td>max stallspp</td>
<td>Maximum number of stalls per packet</td>
</tr>
<tr>
<td>#links AO stallspp</td>
<td>No. of AO links w.r.t. stalls per packet</td>
</tr>
<tr>
<td>max inj FIFO</td>
<td>Maximum injection FIFO length</td>
</tr>
</tbody>
</table>
Experimental Setup

- Three benchmarks: 5-point 2D Halo, 15-point 3D Halo, All-to-all over sub-communicators
- Two scientific applications: pF3D, MILC

### Table

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>2D Halo 16 KB</th>
<th>4 MB</th>
<th>3D Halo 16 KB</th>
<th>4 MB</th>
<th>Sub A2A 16 KB</th>
<th>4 MB</th>
<th>MILC 16 KB</th>
<th>4 MB</th>
<th>pF3D</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>208</td>
<td>94</td>
<td>806</td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>103</td>
<td>103</td>
<td>710</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td>168</td>
<td>311</td>
<td>197</td>
<td>1516</td>
<td></td>
</tr>
</tbody>
</table>

- Regression methods in scikit-learn: extremely randomized trees, gradient boosted regression trees
Predicting the execution time

- Scale the input features to values between 0 and 1
- Split samples into training and testing set (2/3 : 1/3)
- Generate all possible combinations ($2^{19}$) of the 19 input features
- Parallel runs to try all combinations and report prediction scores
Evaluation criteria

• Kendall rank correlation coefficient

\[
RCC = \left( \sum_{0<i<n} \sum_{0<j<i} \text{concord}_{ij} \right) / \left( \frac{n(n-1)}{2} \right)
\]

\[
\text{concord}_{ij} = \begin{cases} 
1, & \text{if } x_i \geq x_j \text{ & } y_i \geq y_j \\
1, & \text{if } x_i < x_j \text{ & } y_i < y_j \\
0, & \text{otherwise}
\end{cases}
\]

• Coefficient of determination, \(R^2\)

\[
R^2(y, \hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}
\]
Prediction on individual datasets

RCC and $R^2$ (GBRT, Huber loss function)

![Graph showing prediction scores for different datasets with RCC and $R^2$ values.](image-url)
Feature importance (individual datasets)

In this section, we analyze the possibility of using the data gathered for a few different codes to predict novel scenarios or discuss this in detail in Section VI.

The relative importance of features drawn from these plots is that the number of bytes flowing over the network has a significant impact on execution time, which changes depending on the code and on whether RCC or R is used for testing. Table IV shows the best prediction scores together are used for training and the four application datasets are used for testing. In the second scenario, we group the datasets by kernels and production applications, and the 4K-node data for testing. In the first scenario, we combine the sixteen datasets by node count into two groups and use the 1K-node data for training set and the 4K-node data for testing. Table IV presents the relative importance or ranks of different features in the models that yield the highest RCC and R scores (right plot) for individual datasets using Gradient Tree Boosting (loss function ‘Huber’). Each stacked bar represents the ranks of the nineteen features (colored by categories) for one of the sixteen datasets.

**TABLE IV**

<table>
<thead>
<tr>
<th></th>
<th>All kernels pF3D (1K + 4K)</th>
<th>All kernels MILC (1K + 4K)</th>
<th>All 1K samples</th>
<th>All 4K samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set</td>
<td>0.874</td>
<td>0.772</td>
<td>0.865</td>
<td>0.673</td>
</tr>
<tr>
<td>Testing set</td>
<td>0.0</td>
<td>0.0</td>
<td>0.63</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Feature ranks for RCC** (GBRT, Huber loss function)
Identifying important features

- Use quantile loss function in the GBRT regressor
Identifying important features

Feature subset selection based on Kernels

0.1 quantile 0.9 quantile

0.4
0.3
0.2
0.1
0
avg dilation AO
max dilation AO
sum dilation AO
avg bytes AO
avg bytes AO
max bytes AO
#links AO bytes
avg stalls AO
avg stalls TO
max stalls AO
max stalls TO
avg stallspp AO
avg stallspp TO
max stallspp
#links AO stallspp
max inj FIFO

Abhinav Bhavele
Identifying important features

Feature subset selection based on Kernels

Abhinav Bhavele
Identifying important features

Feature subset selection based on Kernels

0.1 quantile  0.9 quantile

Applying the feature selection technique explained above for each of the larger datasets. Note that the feature ranks obtained using the technique described above for the feature importance of the features in this identified subset. The marker colors for each row are scaled independently (maroon/red is high and yellow is low).

Fig. 8. Comparison of the feature ranks obtained using the feature selection technique applied to the eight larger datasets. Note that the marker colors for different regions in the function space asymmetrically weights different regions in the function space asymmetrically.

Fig. 7. Ranks of different features obtained using GBRT with quantile loss functions at 0.1 quantile and not used by the regression function optimized for 0.9 quantile and not used by the regression function optimized for the feature that the features show significant importance for prediction, and the inferred model (Figure 7). This means that only a few features have a high rank and be relevant across multiple datasets. Figure 8 presents the results for training and testing sets (Figure 1). The goal is to identify a common set of features that might be relevant across multiple datasets. Figure 8 presents the relative importance of the features in this identified subset.

4) All (all 16 datasets added together)
3) MILC (2 datasets)
2) Kernels (combination of (1), (2), and (3), 12 datasets)
1) 3D Halo (4 datasets)
Sub A2A (4 datasets)
Identifying important features

Feature subset selection based on Kernels

0.1 quantile          0.9 quantile

Abhinav Bhatele
Technique for feature selection

- Create split of dataset into training and testing set
- Learn GBRT regressor with quantile loss function at 0.1 quantile and 0.9 quantile
- Identify feature subsets that are important at different quantiles
- Use the subsets to identify new feature importances
The causes of network congestion
The causes of network congestion

Fig. 7. Ranks of different features obtained using GBRT with quantile loss functions at $\alpha = 0.1$ and $\alpha = 0.9$ respectively: left plot is for a combined set of the three communication kernels (twelve datasets) and the right plot is for a combined set of the two applications (four datasets).

It turns out that for the datasets used in this paper, optimizing for the conditional quantiles inherently promotes sparsity in the inferred model (Figure 7). This means that only a few features show significant importance for prediction, and the ranks of different features vary considerably in the case of lower versus higher quantiles. This results in different features being more important for the two quantiles. Figure 7 shows the feature importance for the extreme quantiles for all the kernel datasets combined together (left plot) and all the application datasets combined together (right plot). In the left plot, we see that the features `avg bytes` and `avg stalls AO` have a high rank only when predicting at the higher quantile. On the other hand, the feature `avg stallspp` is prominent in predicting at the lower quantile and not used by the regression function optimized for the higher quantile. In the right plot, we can observe similar things about `sum dilation AO`, `max inj FIFO`, `#links AO bytes` and `avg stallspp`.

We exploit these observations by selecting the most relevant features from the models at different quantiles, and using this subset of features to predict the execution time for different applications. The steps involved in this proposed technique for feature selection for a dataset are as follows:

1. Create random splits of the dataset into training and testing sets (70% for training and the rest for testing).
2. Learn regression models using GBRT with quantile loss functions at $\alpha = 0.1$ and $\alpha = 0.9$. We denote the feature ranks in the two cases by $\alpha_{0.1}$ and $\alpha_{0.9}$ respectively.
3. Repeat the above steps 50 times to avoid overfitting and compute the average feature ranks for the extreme quantiles from the 50 iterations.
4. Identify the relevant features as those with either $\alpha_{0.1}$ or $\alpha_{0.9}$ greater than a pre-defined threshold $t$. In our experiments, we fixed $t$ at 0.1.

B. Results and discussion

We employ the feature selection technique explained above on the following larger datasets formed by combining the individual datasets in Table III:

1) 2D Halo (4 datasets)
2) 3D Halo (4 datasets)
3) Sub A2A (4 datasets)
4) Kernels (combination of (1), (2), and (3), 12 datasets)
5) MILC (2 datasets)
6) pF3D (2 datasets)
7) Apps (combination of (5) and (6), 4 datasets)
8) All (all 16 datasets added together)

The goal is to identify a common set of features that might be relevant across multiple datasets. Figure 8 presents the feature ranks obtained using the technique described above for each of the larger datasets. Note that the importance/rank of each feature is obtained by first identifying the smallest subset of important features for each dataset and then performing another cycle of training and testing to obtain the relative importance of the features in this identified subset. The marker colors for each row/dataset are scaled independently (maroon/red is high and yellow is low).
The causes of network congestion
The causes of network congestion
The causes of network congestion

- Average and maximum length of receive buffers
The causes of network congestion

- Average and maximum length of receive buffers
- Average load on network links
The causes of network congestion

• Average and maximum length of receive buffers
• Average load on network links
• Maximum length of injection FIFOs
Interference from other jobs

Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019

Bhatele et al. The case of performance variability on dragonfly-based systems, IPDPS 2020
Interference from other jobs

Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019

Concurrently running jobs can contend for shared resources: network, filesystem

Bhatele et al. The case of performance variability on dragonfly-based systems, IPDPS 2020
Data analytics study to understand variability

- Primarily focus on variability arising from sub-optimal communication on the network
- Set up controlled experiments on a dragonfly-based Cray system:
  - Submit jobs of the same applications periodically in the batch queue for ~4 months
- Collect network hardware counters per iteration for each job and other data described later
- Use machine learning to analyze the gathered performance data
Run four applications in control jobs

- Gather network hardware counters on Aries routers connected to my jobs’ nodes
- Hardware counters and execution time recorded per iteration

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of nodes</th>
<th>Input Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG 1.1</td>
<td>128</td>
<td>-P 32 16 16 -n 32 32 32 -problem 2</td>
</tr>
<tr>
<td>AMG 1.1</td>
<td>512</td>
<td>-P 32 32 32 -n 32 32 32 -problem 2</td>
</tr>
<tr>
<td>MILC 7.8.0</td>
<td>128</td>
<td>n128_large.in</td>
</tr>
<tr>
<td>MILC 7.8.0</td>
<td>512</td>
<td>n512_large.in</td>
</tr>
<tr>
<td>miniVite 1.0</td>
<td>128</td>
<td>-f nlpkkt240.bin -t 1E-02 -i 6</td>
</tr>
<tr>
<td>UMT 2.0</td>
<td>128</td>
<td>custom_8k.cmg 4 2 4 4 4 0.04</td>
</tr>
</tbody>
</table>
Other sources of data for analytics

• Job queue logs
  • Information about jobs running concurrently with a specific control job

• Job placement
  • Number of unique groups and routers to which a control job is assigned

• System-wide counters for all Aries routers gathered using LDMS
  • All routers: all routers connected to compute or I/O nodes
  • I/O routers: only routers connected to I/O servers
Analysis I: Identifying predictors of deviation

• Execution times and network counters data are available for each iteration of the application
  • Each iteration is treated as an independent sample

• We create models to predict the deviation of the execution time instead of the absolute time

• We use gradient boosted regression to generate a predictive model and recursive feature elimination (RFE) to study feature importances
Results: Identifying predictors of deviation

Relevance scores of each counter in predicting the deviation from mean behavior for the different datasets.

AMG 128 nodes
AMG 512 nodes
MILC 128 nodes
MILC 512 nodes
miniVite 128 nodes
UMT 128 nodes
Results: Identifying predictors of deviation

Relevance scores of each counter in predicting the deviation from mean behavior for the different datasets.

Network switch congestion important for some apps while end-point congestion more important for others
Analysis II: Forecasting within-run variation

- Idea is to predict next \( k \) time steps based on knowledge of \( m \) previous time steps
- Use a sliding window approach to create the training set
- We use the popular scalar dot-product attention model along with a fully connected neural network
- We explore using different groups of features to understand the impact on model accuracy
Results: Forecasting within-run variation

MAPE = Mean Absolute Percentage Error, m = temporal context, k = predicting future time steps
Results: Forecasting within-run variation

MAPE = Mean Absolute Percentage Error, m = temporal context, k = predicting future time steps
Analysis III: Using only system data

- Use system state before a job starts running to predict performance
- No application-specific features are used
- Train a 2-layer neural network that combines multiple datasets
- Goal: develop application-agnostic models

LDMS gathers data every second

5 mins prior to job

Control Job ‘x’
Results: Predicting perf. of unseen jobs

MAPE comparison for application-agnostic models

<table>
<thead>
<tr>
<th>Testing dataset</th>
<th>All routers</th>
<th>My routers</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG 128</td>
<td>4.23%</td>
<td>3.61%</td>
</tr>
<tr>
<td>AMG 512</td>
<td>8.21%</td>
<td>7.52%</td>
</tr>
<tr>
<td>MILC 128</td>
<td>7.52%</td>
<td>6.90%</td>
</tr>
<tr>
<td>MILC 512</td>
<td>7.52%</td>
<td>6.90%</td>
</tr>
</tbody>
</table>

PSLE comparison for application-agnostic models

<table>
<thead>
<tr>
<th>Testing dataset</th>
<th>% Samples with Large Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMG 128</td>
<td>0.00%</td>
</tr>
<tr>
<td>AMG 512</td>
<td>5.00%</td>
</tr>
<tr>
<td>MILC 128</td>
<td>10.00%</td>
</tr>
<tr>
<td>MILC 512</td>
<td>10.00%</td>
</tr>
</tbody>
</table>

We observe that when multiple datasets are combined for training, the models perform better in terms of predicting the execution times, compared to training on a portion of the training. The MAPE for predicting the execution times, compared to training on a portion of the training, the models perform better in terms of predicting the execution times, compared to training on a portion of the training. We observe that AMG has the lowest errors, followed by MILC, and then miniVite. AMG has the lowest performance variability and miniVite the highest. We believe that this is the reason for the models having better success with predicting AMG's performance than MILC and then miniVite. On average, AMG has the lowest percentage of communication with respect to its total execution time, followed by MILC, and then miniVite. AMG reduces from 4.23 to 3.61 and that for AMG 512 reduces from 8.21 when used by itself for training to 7.52 when the other three datasets are combined for training a model. This improvement is likely due to the larger training dataset (⇠450 samples versus 150) allowing for more robust training of models. We see this as a promising sign for future models which could include tens of applications and more generalizable predictions. We also observe that certain features might be more important for the My routers data, while RTS_PKT_RSP (processor tile flits) are more important when using All routers, while RT_PKT_RSP (stalls on global links), and RTS_PKT_REQ (stalls on router tiles) are more important when using All routers, while PT_PKT_RSP (stalls on router tiles) is more important with the My routers data.

For example, when we combine all AMG and MILC datasets for training, we use the miniVite dataset for testing. Figure 9 shows how these application-agnostic models perform in terms of predicting perf. of unseen jobs. We observe that AMG has the lowest errors, followed by MILC, and then miniVite. AMG has the lowest performance variability and miniVite the highest. We believe that this is the reason for the models having better success with predicting AMG's performance than MILC and then miniVite. On average, AMG has the lowest percentage of communication with respect to its total execution time, followed by MILC, and then miniVite. AMG reduces from 4.23 to 3.61 and that for AMG 512 reduces from 8.21 when used by itself for training to 7.52 when the other three datasets are combined for training a model. This improvement is likely due to the larger training dataset (⇠450 samples versus 150) allowing for more robust training of models. We see this as a promising sign for future models which could include tens of applications and more generalizable predictions. We also observe that certain features might be more important for the My routers data, while RTS_PKT_RSP (processor tile flits) are more important when using All routers, while RT_PKT_RSP (stalls on global links), and RTS_PKT_REQ (stalls on router tiles) are more important when using All routers, while PT_PKT_RSP (stalls on router tiles) is more important with the My routers data.
Results: Predicting perf. of unseen jobs

Based on global routers


Abhinav Bhavele
Results: Potential impact on job schedulers

- Classify jobs into likely fast or likely slow based on values of three most important features
- Based on whether values of these features are above or below the median

![Distribution of likely fast vs. slow jobs (AMG 512 nodes)](image)
Results: Potential impact on job schedulers

- Classify jobs into likely fast or likely slow based on values of three most important features
- Based on whether values of these features are above or below the median

**Fig. 9. MAPE and PSLE scores for the NN model when combining datasets by application type. Two applications are used for training and the third**

**Fig. 10 shows the relative feature importances for three different training datasets (AMG+MILC, AMG+miniVite, and MILC+miniVite), and two filterings (All routers and My routers). Surprisingly,**

- "important feature. In principle, one would expect that the system-wide values of these three counters were below the median, but this was not observed in the previous plots. We also observe that more groups, the likelihood of encountering congestion on incoming jobs in the queue as likely to run relatively fast or slow. The hypothesis is that by selecting a small number of incoming jobs in the queue as likely to run relatively fast or slow, the job scheduler can quickly determine if the fast versus slow jobs in each dataset. We can see that the significant results below the 1% threshold for all applications. Note that a one-way ANOVA test yields statistically significant difference between the likely fast and slow execution times. The value was below 3e-05, indicating a statistically significant value of likely fast and slow jobs in each application dataset.**

**Table 4 compares the mean execution times of the likely fast and slow subsets of**

**Fig. 11. Distribution of actual runtimes of likely fast versus slow jobs of AMG 512 nodes**

**Fig. 12. Distribution of actual runtimes of likely fast versus slow jobs of MILC 512 nodes**

- "Mean execution times (in seconds) of the likely fast and slow subsets of**

**Note that a one-way ANOVA test yields statistically significant difference between the likely fast and slow execution times. The value was below 3e-05, indicating a statistically significant value of likely fast and slow jobs in each application dataset.**

**Results: Potential impact on job schedulers**

Abhinav Bhavele
Can we minimize performance variability?

- Topology-aware job scheduling
- Self-tuning systems
  - Adaptive congestion-aware routing
  - Adaptive scheduling of jobs
Availability of large-scale monitoring data

- Several Department of Energy laboratories are using LDMS to record monitoring data: LLNL/LC, LBNL/NERSC, ANL/ALCF

- Vast quantities of rich but noisy data: on-node (flops, memory, caches), network, filesystem, power, cooling

Image from Kathleen Shoga’s slides at LLNL

Abhinav Batele
Variability prediction

- Ran a large number of control jobs (hundreds per application): 7 different applications
- Train a classifier (AdaBoost) to predict if an app will experience variation

<table>
<thead>
<tr>
<th>Input source</th>
<th>Counters</th>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysclassib</td>
<td>62</td>
<td>186</td>
<td>infiniband counters</td>
</tr>
<tr>
<td>opa_info</td>
<td>62</td>
<td>186</td>
<td>Omni-Path switch counters</td>
</tr>
<tr>
<td>lustre2_client</td>
<td>44</td>
<td>132</td>
<td>Lustre client metrics</td>
</tr>
<tr>
<td>MPI benchmarks</td>
<td>3</td>
<td>9</td>
<td>Execution time</td>
</tr>
<tr>
<td>Proxy applications</td>
<td>-</td>
<td>1</td>
<td>Compute Intensive</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>1</td>
<td>Network Intensive</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>1</td>
<td>I/O Intensive</td>
</tr>
</tbody>
</table>

Comparing F1 Scores with All vs. Job-only System Data
Self-tuning job scheduler

- Modify the job scheduler to:
  - Obtain recent values of system counters
  - Predict if the next job in the queue will experience variability
  - If yes, put it back in the queue and try scheduling the next job

- Leverage the Flux scheduler framework developed at LLNL

- Enables us to run a scheduler within a job partition allocated by the system scheduler (slurm)
Application performance

Variation Occurrences Comparison ADAA

ADAA Application Performance

Abhinav Bhatele
Fig. 10. This figure displays the percent improvement in max run time for each application in experiment SS under strong scaling with the control and proposed scheduling policy.

Fig. 11. Scheduler makespans. For each experiment this figure displays the control and new scheduler's makespans averaged over their 5 runs. RUBS outperforms the control in each experiment.

Fig. 12. This figure displays the average wait time per app in experiment ADAA. We the proposed scheduler has a larger range of wait times and is often higher.
Identifying best performing code variants

- Many computational science and engineering (CSE) codes rely on solving sparse linear systems
- Many choices of numerical methods
- Optimal choice w.r.t. performance depends on several things:
  - Input data and its representation, algorithm and its implementation, hardware architecture

Many computational science and engineering (CSE) codes rely on solving sparse linear systems. Many choices of numerical methods exist, and the optimal choice with respect to performance depends on several factors:

- Input data and its representation
- Algorithm and its implementation
- Hardware architecture

Choosing an optimal method for a given problem is challenging.
Auto-tuning with limited training data

Kripke: Performance variation due to input parameters

Number of configurations vs. Execution time (s)
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters

![Chart showing performance variation due to input parameters](chart.png)
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters

- Performance also depends on:
  - Code changes, linked libraries
  - Compilers, architecture
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters

- Performance also depends on:
  - Code changes, linked libraries
  - Compilers, architecture

- Surrogate models + transfer learning
Questions?

Abhinav Bhatel

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu