Machine Learning and HPC

Alan Sussman, Department of Computer Science
Announcements

• Quiz 3 will be posted on tomorrow, Wednesday, May 10, 11AM
  • In ELMS, for 24 hours
  • Mainly on topics since last quiz

• Course evaluation: https://www.courseevalum.umd.edu
Why machine learning for parallel computing/HPC?

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

• Supercomputing facility and data center 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
  - For errors, failures (hardware/software), performance, …
Example 1 - 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

https://www.osti.gov/servlets/purl/1184730
Life of a message packet
Experiment - 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

• Rubik is a tool from LLNL that maps tasks to nodes in torus or mesh-connected cluster
# Gathering data for machine learning - features

<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>

**Source Node**

**Network Link**

**Intermediate Router**

**All Resources**
Experimental Setup

• Three benchmarks: 5-point 2D Halo, 15-point 3D Halo, All-to-all over sub-communicators – MPI codes

• Two scientific applications: pF3D, MILC – number of task mappings (from Rubik)

<table>
<thead>
<tr>
<th>#Nodes</th>
<th>2D Halo 16 KB</th>
<th>2D Halo 4 MB</th>
<th>3D Halo 16 KB</th>
<th>3D Halo 4 MB</th>
<th>Sub A2A 16 KB</th>
<th>Sub A2A 4 MB</th>
<th>MILC 208</th>
<th>pF3D 94</th>
<th>Total 806</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>
</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>
</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>
</tr>
</tbody>
</table>

• Regression methods in scikit-learn (a Python ML library)
  • 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

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

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

• 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)
Feature importance (individual datasets)

Feature ranks for RCC (GBRT, Huber loss function)

Rank

0 0.2 0.4 0.6 0.8 1

2D Halo 3D Halo Sub A2A MILC pF3D

InjFIFO Stallsp Stall Bytes Dilation
Identifying important features

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

Feature subset selection based on Kernels
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

Feature ranks (maroon/red is high and yellow is low)
The causes of network congestion

- Average and maximum length of receive buffers
- Average load on network links
- Maximum length of injection FIFOs
Example 2 - 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 (Lightweight Distributed Metric Service)
  - 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
- Create models to predict the deviation of the execution time instead of the absolute time
- 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.

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
- Use the popular scalar dot-product attention model along with a fully connected neural network
- 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
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

Control Job ‘x’

← 5 mins prior to job
Results: Predicting perf. of unseen jobs


Based on global routers
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)](image1)

![Distribution of likely fast vs. slow jobs (MILC 512 nodes)](image2)
How to 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
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
Example 3 - 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 running a scheduler within a job partition allocated by the system scheduler (slurm)
Application performance

Variation Occurrences Comparison ADAA

Average Number of Runs with Variation

Control
Experiment

ADAA Application Performance

FCFS + EASY
RUBS

Laghos AMG Kripke SWFFT PENNANT sw4lite LBANN

Time (sec)

Laghos LBANN

AMG Kripke SWFFT PENNANT sw4lite
Scheduler throughput

![Graph showing Makespan and Average Wait Time for different experiments with FCFS + EASY and RUBS]

**Makespan**
- ADAA: 46.1, 45.8
- ADPA: 44.5, 44.0
- PDPA: 44.4, 44.0
- WS: 50.1, 49.7
- SS: 50.1, 49.0

**Average Wait Time**
- Laghos: 12.2, 12.2
- AMG: 12.3, 12.3
- Kripke: 12.1, 12.1
- SWFFT: 12.6, 12.6
- PENNANT: 12.3, 12.3
- swallize: 12.1, 12.1
- LBANN: 12.4, 12.4
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
Auto-tuning with limited training data

Kripke: Performance variation due to input parameters

Number of configurations

Execution time (s)
Auto-tuning with limited training data

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