Predicting the Performance of Jobs in the Queue using Machine Learning

Ian Costello, Abhinav Bhatele

Department of Computer Science, University of Maryland, College Park, MD 20742 USA
E-mail: ianjc@umd.edu, bhatele@cs.umd.edu

I. Introduction

In recent years, several HPC facilities have started continuous monitoring of their systems and jobs to collect performance-related data for understanding performance and operational efficiency. Such data can be used to optimize the performance of individual jobs and the overall system by creating data-driven models that can predict the performance of pending jobs. In this paper, we model the performance of representative control jobs using longitudinal system-wide monitoring data to explore the causes of performance variability. Using machine learning, we are able to predict the performance of unseen jobs before they are executed based on the current system state. We analyze these prediction models in great detail to identify the features that are dominant predictors of performance. We demonstrate that such models can be application-agnostic and can be used for predicting performance of applications that are not included in training.

II. Background

In this paper, we analyze longitudinal system monitoring data to explore the causes of performance variability in certain *control* jobs. The monitoring data is collected by the Lightweight Distributed Metric Service (LDMS [1]) on Cori, a ~30 Pflop/s Cray XC40 system at NERSC. We run some control jobs on Cori to document the impact of varying resource usage on application performance. We run three codes – AMG (parallel algebraic multigrid solver), MILC (MIMD Lattice Computation), and miniVite (community detection in large distributed graphs), which are representative of common workloads on HPC systems. AMG and MILC are both run in 128 and 512 node variants and miniVite is solely run in 128 node variant.

LDMS provides a framework for system-wide collection of performance related data, albeit at a single frequency configured for the entire system. This may be every second to every minute, depending on the resources available to analyze and store the data, for all nodes on the system.

III. Methodology

A. Reduction in Data Dimensionality

We perform the following reductions of LDMS data to make them suitable for consumption by ML-models:

- We select solely the last five minutes of LDMS data strictly prior to start of the job execution.
- We perform selections and aggregations across routers and ports to reduce the per-node features from 100,000 features to a set of 25 raw features.
- We additionally create a set of human-interpretable features by manually selecting logical groupings of counters.
- We filter by router type (routers assigned to a job, IO routers, and all routers) and perform aggregations across these entire groups.

B. Machine Learning Prediction Models

In this work, we utilize gradient boosted regression (GBR) both in our prediction models and for assigning importances to input features and neural networks solely for performance prediction.

The inputs to the machine learning (ML) algorithms for creating the dataset-specific ML models are: (1) for each sample (job) in the training set, values of the aggregated LDMS counters for the five minutes prior to the start of that job are provided as the input features, and (2) execution time of each sample (job) is provided as the dependent variable to be modeled. Given a set of samples in the testing set, the model outputs the predicted execution time of each testing sample based on the values of the independent features (counter values) for that sample.

We create multiple datasets by combining different applications, and leaving some datasets for testing entirely. For example, in one instance, we combine the following four datasets – AMG 128, AMG 512, MILC 128, MILC 512 – train a model using the combined data, and use the trained model to predict the performance of miniVite 128.

C. Feature Evaluation

To analyze the relative importance of the derived system-wide counters in forecasting job runtime, we use the technique of recursive feature elimination (RFE). We train an initial model and identify the worst performing feature based on feature importances, drop that feature from the training data, and train again with the smaller set of features. This process continues until all features are eliminated. Finally, based on the ranking of when each feature is eliminated, we select the five best features and compute a relevance score for each selected feature.

IV. Results

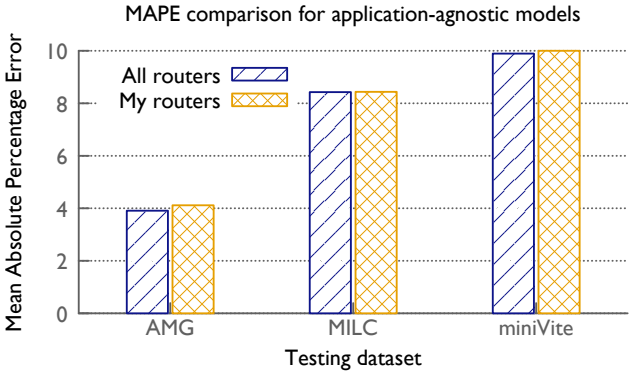

Fig. 1. MAPE scores for the NN model when combining datasets by application type. Two applications are used for training and the third application is used for testing (x-axis label.)

While we use control jobs to create training data, to be useful in a job scheduler our models must perform well on applications they have never seen before. Figure 1 shows the success of the trained models in predicting performance of unseen applications in terms of their MAPE scores. For each application listed, we reserve all node counts of that application entirely for testing and train our models on the other application sets. We observe strong prediction results even though these applications were excluded entirely from the training set with the MAPEs universally below 10% despite significant variability within datasets. We see this as a promising sign for future models which could include tens of applications with hundreds of samples each and likely even stronger and more generalizable predictions. The end goal is to train a single model that can accurately predict the standardized runtime for any application even if we do not have data for that application in the training dataset.

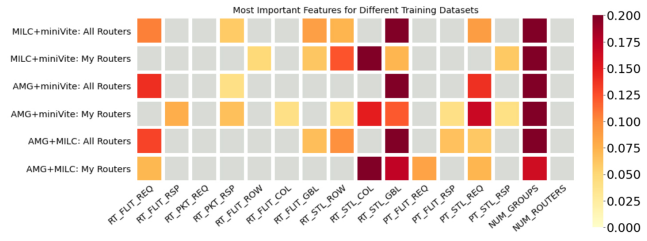

Fig. 2. Relative importances of the most important counters obtained using RFE for different router groups in the application-agnostic model.

We next evaluate which LDMS-derived features are the most important in terms of contributing to the overall execution time of a job. Figure 2 shows the relative feature importances when training the application-agnostic model. `NUM_GROUPS` emerges as a highly important feature suggesting it is possible that when a job is spread over more groups, the likelihood of encountering congestion on global links increases. `RT_STL_GBL` (stalls on global links between node groups) and `RT_STL_COL` (stalls of column links within node groups) both have relatively strong performance impact.

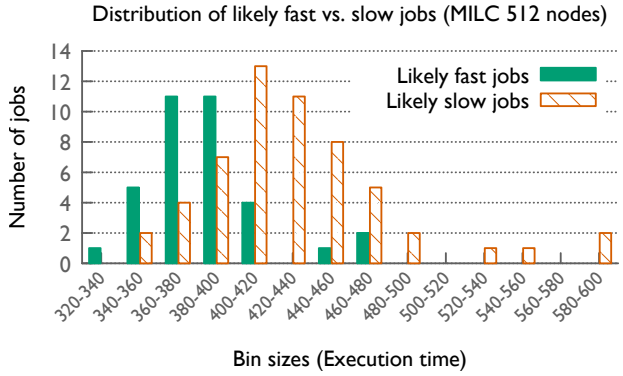

Fig. 3. Distribution of actual runtimes of likely fast versus slow jobs of MILC when considering above median values of three features: RT_STL_COL, RT_STL_GBL, and NUM_GROUPS

We demonstrate by using a small set of these "significant counters" and trivially looking at their median values, we could classify jobs in a dataset into likely fast and likely slow with statistically significant results. Figure 3 shows the execution results for these two groups for MILC 512, the mean of the likely-fast MILC 512 jobs is 389.61 seconds and the mean of the likely-slow MILC 512 jobs is 445.80 seconds.

For MILC 512 and all applications tested, we found the difference between these two groups to be statistically significant below a 1% threshold. This demonstrates that an intelligent job scheduler could use a similar simple mechanism or a more complex model-based approach to forecast the performance of a pending job.

V. Discussion

We foresee two immediate applications of our prediction system. Intensive HPC workloads may be continually resubmitted over the period of a few months for long-running simulations. While the inputs will change slightly as computation progresses, the overall runtime profile and networking requirements of an application usually do not vary significantly, providing an optimal use case for our prediction system. Individual can use this prediction to decide decide whether to go ahead with launching their application or to give the allocation back and request another job later when conditions improve.

Second, these results suggest that an intelligent job scheduler can monitor a handful of counters, and use their current values to determine if, for example, communication-heavy jobs will perform well or poorly if scheduled right away. Figure 3 demonstrates the power of predicting job execution time solely based on the median aggregation of just three network counters While in this paper, we analyzed these jobs after they had run, a job scheduler would have access to similar counter data through LDMS at the time when a job is ready to be scheduled.

ACKNOWLEDGMENT

This work was supported by funding provided by the University of Maryland College Park Foundation.

REFERENCES

[1] 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 *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 154–165.