Automated Load Balancer Selection Based on Application Characteristics

Harshitha Menon, Kavitha Chandrasekar, Laxmikant V. Kale
University of Illinois Urbana Champaign
gplkrsh2@illinois.edu, kchndrs2@illinois.edu, kale@illinois.edu

Abstract

Many HPC applications require dynamic load balancing to achieve high performance and system utilization. Different applications have different characteristics and hence require different load balancing strategies. Invocation of a suboptimal load balancing strategy can lead to inefficient execution. We propose Meta-Balancer, a framework to automatically decide the best load balancing strategy. It employs randomized decision forests, a machine learning method, to learn a model for choosing the best load balancing strategy for an application represented by a set of features that capture the application characteristics.

1. Introduction

HPC applications are increasingly becoming complex and frequently use dynamic structures and adaptive refinement techniques. Such complex techniques introduce load imbalance and therefore, require dynamic load balancing to achieve good performance. Different applications have different characteristics requiring the use of different load balancing strategies. Optimal strategy depends on a complex combination of various application characteristics making it difficult to select the best one. Further, load balancing algorithms have also grown in number making this choice harder. Using a load balancer that is not well suited for the application will result in loss of performance. Typically, the runtime behavior of the application depends on the problem being simulated and the machine characteristics. As a result, choosing the load balancing strategy that gives the best performance becomes difficult. Most commonly, the application programmer decides which load balancer to use and when to do load balancing based on some educated guess. This may result in a suboptimal solution.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PPoPP '17 February 04-08, 2017, Austin, TX, USA

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4493-7/17/02...$15.00
DOI: http://dx.doi.org/10.1145/3018743.3019033

We propose an introspective run-time system component called Meta-Balancer to automate the decision of choosing the best load balancing strategy for an application. Previous works (Pearce et al. 2012; Siegell and Steenkiste 1996) proposes models to predict load balancing period based on the cost benefit of load balancing and (Pearce et al. 2012) predicts load balancing strategy but from a limited set. In constrast our work proposes a framework that makes a selection from six different strategies and uses machine learning technique to do so. Meta-Balancer continuously monitors the application and based on the observed characteristics automatically selects the load balancing strategy. Randomized decision forests, a machine learning method, is used to learn a model for choosing the best load balancing strategy for an application represented by a set of features that capture the application characteristics. Meta-Balancer is implemented on Charm++ runtime system and takes advantage of its support for load balancing.

2. Meta-Balancer Strategy Selection

Meta-Balancer framework monitors the application and system characteristics by automatically collecting statistics about it. It then chooses the load balancing strategy based on the observed characteristics. It is implemented as a part of the Charm++ runtime system. Meta-Balancer consists of two major components, namely, asychronous statistics collection and decision making module for the ideal load balancing strategy.

2.1 Meta-Balancer Statistics Collection

Meta-Balancer collects load and communication information about the application frequently. These statistics are collected at the iteration boundary. The load balancing framework in Charm++ automatically collects the load and communication at each PE and stores it in a distributed manner. The Meta-Balancer uses some of the information stored in the load balancing framework. The statistics that are collected are normalized and used as features for the random forest machine learning technique.

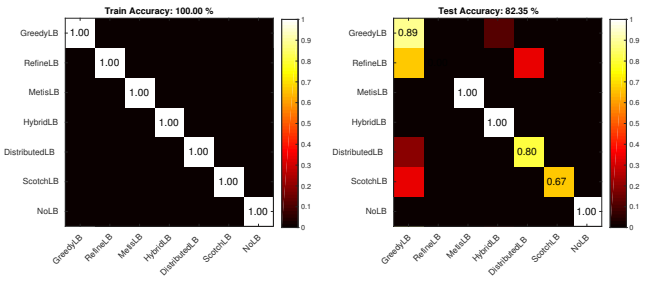

Figure 1: Accuracies on the training and test set along with confusion matrix.

2.2 Load Balancing Strategy Selection

There are a number of load balancing strategies in Charm++. These strategies handle load imbalance for different application characteristics. When no load balancing is required, we refer to it as NoLB. The strategies used are:

GreedyLB: A strategy that uses greedy heuristic to assign heaviest tasks onto least loaded processors iteratively.

RefineLB: A strategy that incrementally transfers the load away from the overloaded processors.

MetisLB: A strategy that passes the load information and the communication graph to METIS graph partitioner.

ScotchLB: A strategy that uses SCOTCH graph paritioning library to make load balancing decisions.

HybridLB: A hierarchical strategy in which at each level of the hierarchy, the root node performs the load balancing for the processors in its sub-tree.

DistributedLB: A refinement based distributed load balancing strategy that does probabilistic work transfer.

2.3 Load Balancing Strategy Selection Using Machine Learning

Machine learning techniques have been used to do pattern recognition. These techniques operate by building a model from a sample input set and learn to predict the outcome. We use a supervised random forest technique to predict the load balancing strategy. In supervised learning algorithms, the tool is presented with example inputs and their desired output.

A benchmark, called *lbtest*, was used to generate the training set for the random forest. This benchmark was run with different parameters to generate different configurations. For each configuration, all the load balancers were run to identify the best performing load balancing strategy. For the training set, the statistics collected were used as input features to the machine learning algorithm. The desired output for the training is the best performing load balancing strategy.

3. Experiments

The benchmark used for the strategy selection is *lbtest*. It was run on Blue Waters, a hybrid XE/XK system located

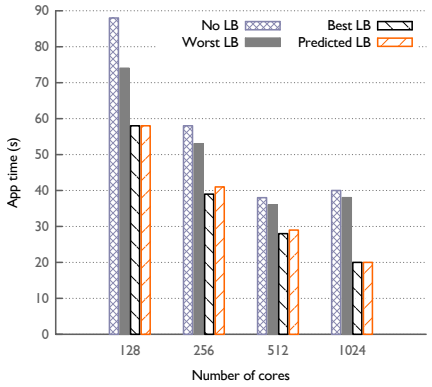

Figure 2: Performance of Lassen using the trained model

at NCSA.More than 200 samples were collected to generate the training set for the machine learning model. The samples were divided into training and test set and the learning algorithm was given the training set and evaluated on the test set. The expected outcome is the best performing load balancing strategy. Figure 1 shows the performance of the machine learning model in predicting the expected outcome. We can see that it is able to achieve an accuracy of 82%. The confusion matrix shows how the expected outcome varied.

We used the trained model on a new application, Lassen, which is a LLNL proxy app to study detonation shock dynamics, to predict the ideal load balancing strategy. Figure 2 shows the total application time with NoLB, worst, best and the predicted load balancing strategies. Meta-Balancer accurately predicts the best performing load balancer for 128 cores to be GreedyLB and for 1024 to be DistributedLB. For 256 and 512 core runs, the predicted (GreedyLB) is very close to the best performing load balancing strategy (DistributedLB). The predicted load balancing strategy gives an improvement ranging from 30% to 100%.

Acknowledgments

This research was supported in part by NCSA PAID - LB (NSF OCI 07-25070) NSF ChaNGa (AST 13-12913) and NSF OpenAtom (ACI 13-39715). This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

O. Pearce, T. Gamblin, B. R. de Supinski, M. Schulz, and N. M. Amato. Quantifying the effectiveness of load balance algorithms. In *26th ACM international conference on Supercomputing*, ICS '12, pages 185–194, 2012.

B. S. Siegell and P. A. Steenkiste. Automatic selection of load balancing parameters using compile-time and run-time information, 1996.