Meta-Balancer: Automating Load Balancing Decisions

Harshitha Menon, Kavitha Chandrasekar, Laxmikant V. Kale
University of Illinois Urbana Champaign

***Abstract*—HPC applications are increasingly becoming complex and dynamic. Many applications require dynamic load balancing to achieve high performance and system utilization. Different applications have different characteristics and hence need to use different load balancing strategies. There are many load balancing algorithms available. However, invocation of an unsuited load balancing strategy can lead to inefficient execution. Most commonly, the application programmer decides which load balancer to use based on some educated guess. We propose Meta-Balancer, a framework to automatically decide the best suited load balancing strategy. Meta-Balancer monitors application characteristics and based on that, it chooses an ideal load balancing algorithm to use. In order to predict the best load balancing strategy, Meta-Balancer uses a supervised random forest machine learning technique with the application characteristics as the features. Using this, we are able to achieve high prediction accuracy of 82% on the test set to demonstrate performance benefits of up to 3X.**

I. Introduction

Many applications require dynamic load balancing to achieve good performance and increase the scalability. Different applications need to use different load balancing strategies. There are numerous load balancing algorithms available. Each of these load balancers are suitable for certain applications. Using a load balancer that is not well suited for the application will result in loss of performance. Typically, the application behavior depends on the machine characteristics and system being simulated. 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 my result in suboptimal solutions.

In this poster, Meta-Balancer framework is proposed which automatically decides the best suited load balancing strategy. This runtime system component continuously monitors the system and based on the observed characteristics automatically chooses the load balancing strategy. Work on quantifying load balancing cost and choosing the load balancing strategy was done in [1]. They consider only two load balancing strategies. Whereas, we consider 6 different load balancing strategies that can be used. Our previous work [2] predicts the ideal load balancing frequency for a given application. In this work, we extend the predictive capability to automate the decision of ideal load balancing strategy. The main contributions are:

- Use of randomized decision forest machine learning technique to select the best performing load balancing strategy
- Identification and automatic collection of statistics that capture application characteristics
- Demonstration of the effectiveness of the strategy selection model on Lassen, a LLNL proxy application.

II. Meta-Balancer

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.

A. 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. Meta-Balancer gathers these statistics via an asynchronous reductions. Some of the statistics that are collected are shown in Table I.

Statistics	Description
Num_PEs	Number of PEs in the system
Num_Ovld_PEs	Number of overloaded PEs in the system
Avg_PE_Load	Average load of the PEs in the system
Max_PE_Load	Maximum load of the PEs in the system
PE_Load_STD	Standard deviation of the PE loads
Avg_Utilization	Average utilization of the PEs
Total_Msgs	Total number of messages transferred
Total_Bytes	Total amount of bytes transferred
Total_Ext_Msgs	Total number of messages transferred outside a PE
Total_Ext_Bytes	Total amount of bytes transferred outside a PE
Avg_Hops	Average number of hops per message
Avg_Hopbytes	Average number of hop bytes per message
Alpha	Cost of latency per message
Beta	Per link 1/bandwidth
Rate_Max_Load	Rate of change of maximum load of PEs across iterations
Migration_Size	Average amount of data per object to be transferred during migration

TABLE I: Meta-Balancer collected statistics

The statistics that are collected are normalized and used as features for the random forest machine learning technique.

B. 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. Some of these strategies are:

- **GreedyLB**: A strategy that uses greedy heuristic to assign heaviest tasks onto least loaded processors iteratively. GreedyLB is most suitable for applications which has high compute load imbalance.
- **RefineLB**: A strategy that carries out load balancing by incrementally transferring the load away from the overloaded processors.
- **MetisLB:** A strategy that passes the load information and the communication graph to METIS, a graph partitioning library. MetisLB is most suitable for communication intensive applications.
- **ScotchLB**: A strategy that uses SCOTCH graph parition-ing library to make load balancing decisions. ScotchLB is suitable for applications that is affected by communi-cation as well as the computation load imbalance.
- **HierarchicalLB**: 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 bal-ancing strategy that does probabilistic work transfer.

C. 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. In this work, a supervised random forest technique is used to predict the load balancing strategy for a particular run. 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 is flexible and can generate various scenarios representative of a real application. 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.

III. EXPERIMENTAL RESULTS

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

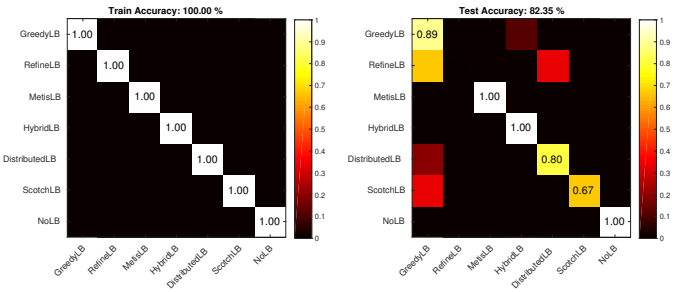

Fig. 1: Accuracies on the training and test set along with confusion matrix.

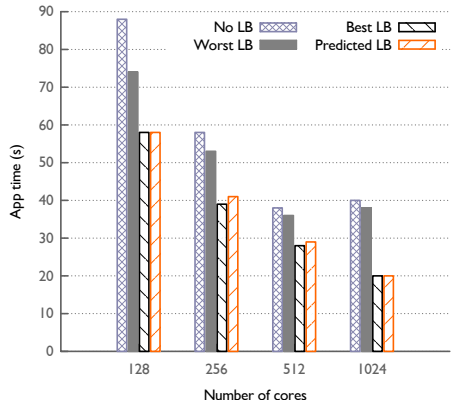

Fig. 2: Performance of Lassen using the trained model

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%.

IV. CONCLUSION

Load imbalance is a very important factor affecting the performance of various applications. Application programmers have to deal with the complexity of choosing load balancing strategy under dynamic conditions. In this poster, we presented a runtime system module that automatically predicts a suitable load balancing strategy to employ based on the runtime characteristics of the application as well as the system. A random forest machine learning technique was used to predict the load balancer based on those features. We were able to achieve a good accuracy of 82% with the model that was trained.

REFERENCES

[1] 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*, ser. ICS '12, 2012, pp. 185–194.

[2] H. Menon, N. Jain, G. Zheng, and L. V. Kalé, "Automated load balancing invocation based on application characteristics," in *IEEE Cluster 12*, Beijing, China, September 2012.