Predicting the Flexibility of Dynamic Loop Scheduling Using an Artificial Neural Network

Srishti Srivastava*, Brandon Malone†, Nitin Sukhija*, Ioana Banicescu*, and Florina M. Ciorba‡

*Department of Computer Science and Engineering
Mississippi State University
Mississippi, USA
{ss878@,ioana@cse.,nitin@cavs.}msstate.edu

†Department of Computer Science
Helsinki Institute for Information Technology, University of Helsinki
Helsinki, Finland
brandon.malone@cs.helsinki.fi

‡Center for Information Services and High Performance Computing
Technische Universität Dresden
Dresden, Germany
florina.ciorba@tu-dresden.de

***Abstract*—In this paper, an artificial neural network (ANN) model is proposed to predict the *flexibility* (or robustness against system load fluctuations in heterogeneous computing systems) of dynamic loop scheduling (DLS) methods. The multilayer perceptron (MLP) ANN model has been used to predict the degree of robustness of a DLS method, given specific values for the problem size, the system size, and the characteristics of the system load fluctuations as a compound effect of the variations in the application's iteration execution times and the processor availabilities. The developed MLP ANN model can be useful in an effective selection of the most robust DLS technique for scheduling a certain type of scientific application onto a given set of non-dedicated heterogeneous processors, when their system load is expected to fluctuate unpredictably during the application's runtime.**

***Keywords*-dynamic loop scheduling; robustness; flexibility; artificial neural networks; multilayer perceptron; fluctuating system loads.**

I. Introduction

Recently, it has been shown that dynamic loop scheduling (DLS) algorithms are a powerful approach towards improving the performance of scientific and engineering applications via dynamic load balancing [1]. To guarantee a certain performance level of the DLS methods, metrics are required to measure their robustness against various unpredictable variations of perturbation factors in the heterogeneous computing environment, such as processor load fluctuations, processor failures, and others. Successful efforts have been made to develop such metrics and a methodology for measuring the robustness of the DLS techniques against system load fluctuations (using the flexibility metric) and processor failures (using the resilience metric) [2][3]. Further, the robustness metrics and the proposed methodology were used to evaluate the robustness of the DLS methods against variations in processor availability, where the experiments were performed in a C simulation environment [4]. In this paper, the focus is on forecasting the *flexibility* of the DLS methods via the use of an artificial neural network (ANN). Throughout the paper, the terms *flexibility* and *robustness against system load fluctuation* are used interchangeably. The system load is defined as the compound effect of the variations in the problem characteristics (loop iteration execution times) and the systemic characteristics (processor availabilities). In the previous work, the robustness of the DLS methods was only assessed with respect to the variations in the processor availabilities for a manageable number of test cases [4]. However, it is of interest to investigate the robustness of dynamic loop scheduling via an ANN analysis, to ensure the robustness of the DLS methods for a considerably larger number of experimental cases resulting from considering different combinations of problem sizes, number of processors, and scheduling methods. Forecasting robustness

978-0-7695-5018-3/13 $26.00 © 2013 IEEE
DOI 10.1109/ISPDC.2013.10

using traditional statistical specifications in the form of tables and equations for performing a comprehensive manual robustness analysis is a very time consuming and tedious task. Traditional statistical approaches often assume a linear functional relation between the actual parallel execution performance feature and the affecting perturbation parameters. However, the actual performance shows highly non-linear behavior in relation to the perturbation parameters. A multilayered perceptron (MLP) ANN model is used in this paper to predict the *flexibility* of the DLS methods in the presence of system load fluctuations, by learning the relation among the following attributes: loop scheduling methods, problem sizes, system sizes, characteristics of the system load fluctuations as a compound effect of the characteristics of the variations in the iteration execution times and the processor availabilities, and impact of system load fluctuations on the parallel execution time. The MLP ANN model generates a non-linear function from the perturbation parameters (i.e., system load) to the performance feature (i.e., execution time). The ANN does not require any prior assumption of the type of functional relation between the above mentioned attributes. The ANN is trained on a part of the input data set and tested with a different subset of the input data set. The results for generating the input dataset have been obtained in a similar manner as described in the work presented in [5]. Furthermore, when exposed to new data, the MLP ANN is capable of accurately predicting the DLS robustness. The work presented in this paper is a proof of concept that ANNs can be employed to predict the *flexibility* of DLS in the presence of fluctuating system load. Preliminary results obtained in [5], have been used to verify this concept. The results obtained from the work done in this paper provide a novel contribution towards predicting the *flexibility* of scheduling in parallel and distributed computing, considering that very limited work has been done in this area [6][7].

In the following section, the background and a review of relevant work are presented. The proposed prediction methodology is described in Section III. The results obtained from the generated MLP ANN model are illustrated and analyzed in Section IV. A discussion of the benefits and usefulness of the proposed methodology are highlighted in Section V. The conclusions and potential future work are summarized in Section VI.

II. Background and Related Work

Dynamic Loop Scheduling. The fundamental idea behind the DLS approach is to determine the workload for each of the P processors as a function of the loop iteration chunks (a collection of loop iterations or tasks) to obtain balanced processor loads. Extensive research in this area has produced the design and development of a number of loop scheduling algorithms. These algorithms are further classified into the following two categories: *non-adaptive* and *adaptive*. Non-adaptive algorithms determine workload sizes based on *a priori* information and assumptions made before the execution of the application. Examples of non-adaptive DLS algorithms include self scheduling (SS), guided self scheduling (GSS) [8], fixed size chunking (FSC) [9], factoring (FAC) [10], and weighted factoring (WF) [11]. The DLS algorithms belonging to the adaptive category determine the workload related information during the execution of the *current* chunk of loop iterations and from the *prior* execution of loop iterations. The adaptive DLS methods are: adaptive weighted factoring (AWF) and its variants AWF-Batch (AWF-B) and AWF-Chunk (AWF-C), and adaptive factoring (AF). These are listed in order of their increasing complexity. All DLS techniques are based on probabilistic analyses and they schedule chunks of loop iterations (either from a central ready queue or using a master-worker paradigm) so that they finish within the optimal time with high probability. For a detailed description of these algorithms, please refer to [1]. Some examples of real applications that have successfully incorporated these DLS algorithms include *Monte-Carlo simulations, radar signal processing,* N-*body simulations, computational fluid dynamics, and wave packet simulation using quantum trajectory method* [1]. The workload used in the simulation experiments, which were performed to generate the parallel execution

times for the calculation of the robustness of the DLS method being used, capture the characteristics similar to the characteristics of the aforementioned applications.

Robustness of DLS. Due to advances in computing platforms, robustness has become an emerging area of research and aims to guarantee performance in the presence of uncertainties in the computing environment. In the case of dynamic load balancing of scientific applications (which are large, irregular, computationally intensive, and often contain a large number of data parallel loop iterations) onto a set of computing resources via DLS, robustness analysis metrics and methodologies can be useful to measure the response of a scheduling method to possible erroneous inputs or unpredictably varying environmental factors. The robustness analysis of DLS enables selection of a robust DLS method for scheduling a certain loop of N iterations with certain loop execution characteristics, onto a set of P computing processors with unpredictably varying processor speeds. Recently, robustness metrics were formulated to study the robustness of the DLS methods against two perturbation parameters: processor load variation and processor failures [2][3]. The robustness of the DLS techniques against unpredictable variations in the processor loads was mathematically represented using a *flexibility metric*, whereas, the robustness of the DLS methods against processor failures was mathematically represented using a *resilience metric*. The two metrics collectively define the robustness or the reliability of a DLS method in the presence of perturbations. The related work on the formulation of the robustness metrics can be found in [2][3], which has been motivated by the work on analyzing the robustness of resource allocation heuristics, in [12]. A methodology has been developed which employs the metrics from [2][3] to measure the robustness of the DLS methods and a STATIC straight-forward parallelization technique against system load variations [4].

An MLP ANN. The multilayer perceptron (MLP) is one of the most widely used ANNs for prediction. The MLP ANN consists of an input layer, one or more hidden layers, and an output layer. The input parameters are provided into the neurons in the input layer. These values are then fed forward into the first hidden layer. Neurons in the hidden layer then perform some calculations based on the values they receive from the input layer. Common operations include summing the values and then performing a sigmoid transformation or thresholding. The neurons in the first hidden layer pass the result of their operations to neurons in the second hidden layer (if it exists), which perform similar calculations. This process continues until the neurons in the output layer produce the final calculations [13]. An example of a feed-forward MLP ANN can be seen in Figure 1.

A higher number of hidden layers introduces more weight parameters into the network, which increases the model complexity [14]. In general, an ANN with more hidden layers can learn more complex functions than one with fewer hidden layers; however, complex ANNs are more prone to overfitting and often perform worse on unseen (testing) examples. The increase in the number of hidden layers and/or neurons also increases the training time [15].

In this paper, we adopt the widely used *back-propagation* algorithm to learn the weights of the edges. Backpropagation begins by randomly initializing all edge weights. The learning algorithm consists of a number of two phase iterations. In the first phase, a training input sample is fed into the input layer and propagated through the layers until output is generated. Error is calculated as the difference between the desired (known) output for that sample and the calculated output. In the second phase, the error is propagated backwards, and link weights are modified to reduce the error between the desired and calculated output [16]. Each iteration comprises both phases for a portion of the input training samples. For example, one iteration may consist of using each input training sample once. Training stops upon reaching a specified stopping criterion, such as only a small change in the weights from one iteration to the next.

In our experiments, we use *10-fold cross-validation*. In this technique, the entire data set is divided into 10 equal-sized smaller data sets

(*folds*). In particular, we used *stratified* cross-validation, in which each fold contains a similar number of instances of each class, to minimize the bias caused by the skewed class distribution in our dataset. Among these 10 data sets, 9 are used as the input training set for backpropagation. After training reaches the stopping criteria, the final data set is used as a test set.

We calculate the *accuracy* and *balanced error rate* to evaluate the predictive performance of the MLP ANN. We chose to collect both statistics because we wish to investigate both how well the MLP performs at predicting runtimes across all instances as well as its ability to distinguish between different robustness classes. We also report the confusion matrix of the MLP.

The learning and evaluation process is repeated 10 times, such that each of the smaller data sets is used as the test set once. This results in 10 values for accuracy and balanced error rate. These are further averaged to obtain the final prediction statistics for evaluation of the proposed MLP ANN.

III. Design of the MLP ANN Model

In this paper, an MLP ANN model is developed to predict the flexibility of the loop scheduling methods in the presence of system load fluctuations. The model is generated using the MLP classifier of the open source data mining tool, Weka [17]. The input data set for the proposed MLP consists of parallel execution times obtained using a simulation toolkit as described in [5] and using the robustness analysis methodology proposed in [4] These execution time values have been obtained for various problem sizes, system sizes and probability distributions characterizing the fluctuations in system load. The degree of robustness is defined in the MLP input data set as the range of values [1, ... 5], where 5 is the highest degree of robustness denoting the most robust scheduling method for a particular execution scenario, and 1 denotes a non-robust scheduling method. For a given number of processors, a given number of loop iterations, and a particular probability distribution for the variations in the iteration execution times and the processor availabilities, the degree of robustness is calculated as follows:

- Calculate the parallel execution time for the ideal case T_{PAR}^{ideal}, for each scheduling method, where the computing system has dedicated processors with 100% availability, for each scheduling method.
- Calculate the parallel execution time T_{PAR}, where the application has variable iteration execution times, and the computing system has non-dedicated processors with variable availability.
- Set the values of the tolerance factor, τ, enforcing an upper limit to the impact of fluctuating system load on T_{PAR}, as 1, 1.25, 1.5, and 1.75.
- A scheduling method is robust if it satisfies the condition $T_{PAR} \leq \tau \cdot T_{PAR}^{ideal}$. Thus, the degree of robustness is 5 for $\tau = 1$, 4 for $\tau = 1.25$, 3 for $\tau = 1.5$, 2 for $\tau = 1.75$, and 1 for all values of $\tau > 1.75$.

For the calculation of the degrees of robustness, the values of T_{PAR}^{ideal} and T_{PAR}, were obtained using the SimGrid simulation framework and the approach in [5] for all possible combinations of the values of the following parameters: *schedulingMethod* $= \{$STATIC, FSC, GSS, FAC, WF, AWF-B, AWF-C, AF$\}$, $P = \{2048, 4096, 8092\}$, $N = \{1048576, 4194304, 16777216\}$, *iterationDistribution* $= \{Gaussian, Gamma, Exponential\}$ and *availDistribution* $= \{Uniform, Exponential\text{-}constant, Exponential\text{-}variable\}$.

The final input dataset for our experiment contained 1,152 samples (or instances). The input dataset is preprocessed using the *NumericToNominal* preprocessor in Weka, which converts all of the numeric values, such as $8,092$ and $4,194,304$, into categorical values. Thus, during the ANN learning, these values represent categories. This step prevents the magnitude of N from dominating all of the calculations.

We then converted each instance into a 22-dimension binary vector with the *NumericToBinary* preprocessor in Weka. In this step, a binary variable is introduced for each value of each parameter. For example, there is a binary variable corresponding to $availDistribution = Gaussian$.

Each instance is converted to a binary vector by setting the binary variables corresponding to the parameter values for that instance to 1 and the others to 0. We perform this transformation to avoid suggesting an ordering to the parameter values. For example, if for $availDistribution$, we encoded $Gaussian$ = 1, $Gamma$ = 2 and $Exponential$ = 3, then that implies to the learning algorithm that $Gamma$ is "closer" to $Gaussian$ than $Exponential$. The binary encoding avoids this implication. The $degreeOfRobustness$ was similarly transformed. In order to minimize any bias in the distribution of samples, due to the skewed nature of the collected data, during our cross-validation evaluation, we randomized the ordering of the instances. Different number of iterations of training were used during backpropagation. However, the results reported in this paper have been generated with 500 iterations of training (or epochs). This offers a good tradeoff between the computing time required to train the MLP and overfitting the training set. Empirically, other numbers of iterations resulted in similar prediction accuracy and increased or decreased the computation time linearly. The preprocessed data is then fed as the input dataset to the MLP classifier in Weka. We use stratified 10-fold cross-validation, as discussed in Section II, to divide the dataset into the respective training and testing sets. The goal of the MLP is to predict the degree of robustness given scheduling method, P, N, iteration distribution and availability distribution, and as such we select degree of robustness as the *class variable*. During training, we use the class variable to adjust the edge weights. Backpropagation is then used to train the MLP. After training for each cross-validation fold, we use the MLP to predict the degree of robustness for the test data set. While testing, the class variable is treated as unknown and predicted with the MLP. Based on the predictions, we calculate the accuracy and balanced error rate of the model.

IV. Experimental Analysis and Evaluation

The MLP ANN model generated using Weka is used to predict the *flexibility* of the scheduling methods as a measure of their degree of robustness against system load fluctuations in a given execution scenario. The structure of the MLP ANN, generated using Weka, is illustrated in Figure 1. The MLP ANN has an input layer with 22 neurons, one hidden layer with 13 neurons, and one output layer with 5 neurons.

The accuracy of the MLP ANN for predicting the degree of robustness was 0.95 on the test dataset samples, which were not used during training. The balanced error rate of the MLP ANN was 0.58. The time taken by Weka to build and train the MLP ANN model was 9.55 seconds, and the time taken to test and validate the ANN model was 0.94 seconds. These timings were recorded by Weka on an Apple® computer with an Intel® Core i5-2.3GHz processor and 4.00 GB RAM.

For comparison, we show the prediction errors of the learned MLP ANN compared to the errors of a simple 0-R classifier, which simply predicts the most common value (robustness class 1, in this case) for all samples. As expected, and confirmed in Figure 2, the MLP ANN outperforms the naive 0-R classifier. Of course, the 0-R classifier correctly predicts all of the samples for class 1, but does not distinguish between any of the other four robustness classes. For comparison, this gives the 0-R classifier a seemingly impressive accuracy of 0.90 (5% worse than the MLP ANN, though); however, its balanced error rate was 0.8. These results call attention to the skewed distribution of our data and highlight the difficulty and importance of distinguishing between robustness classes.

Qualitatively, as shown in the confusion matrices in Figure 2, the MLP ANN only mis-classifies 10 instances (out of 62) from class 5 and correctly identifies 8 instances (out of 31) from class 2. These correct predictions are important for downstream planning because they allow a scheduler to effectively decide which loop scheduling algorithm will best adhere to user constraints. In contrast, the incorrect predictions by the 0-R classifier could lead to very poor decisions about which scheduling algorithm to prefer. Both the MLP ANN and the 0-R classifier were unable to correctly predict any instances from classes 3 or 4. The MLP ANN was

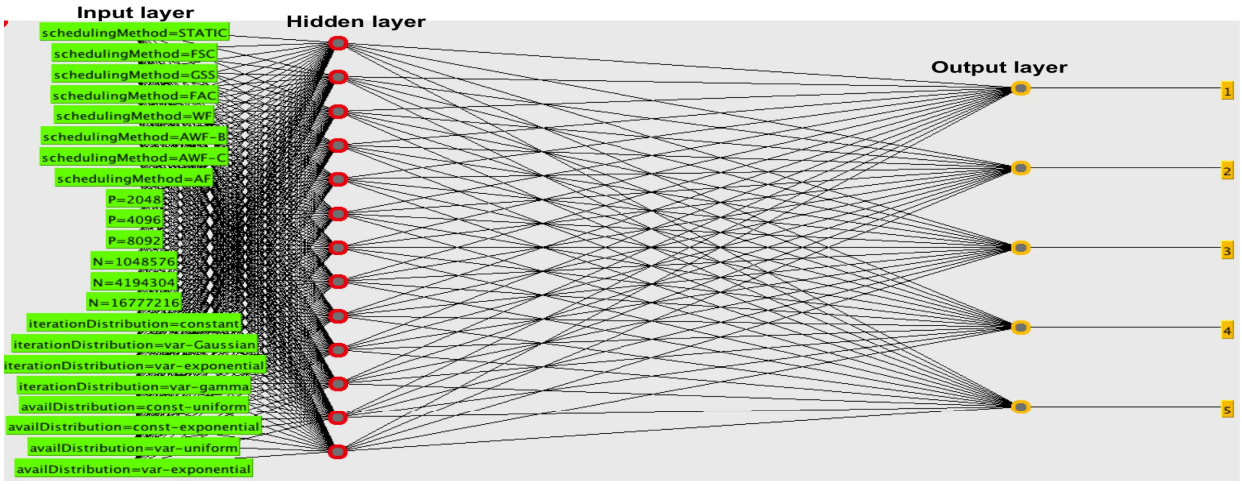

Figure 1: Weka-generated MLP ANN with five input attributes: scheduling methods, system size, problem size, probability distributions for iteration execution time variation, and probability distributions for varying processor availabilities. One output class attribute: degree of robustness [1, ... 5].

unable to distinguish these classes because of the limited number of training instances available for them (13 and 7, respectively). A larger training set would allow the backpropagation algorithm to better learn the characteristics of these classes. The use of a larger training dataset is potential future work to the preliminary work done in this paper for predicting the flexibility of DLS with ANNs.

A scatter plot of the predicted values of the degree of robustness for all scheduling methods is in Figure 3. Furthermore, the visual interface of the output result in Weka allows a detailed view of any coordinate point on the scatter plot in a textual mode. This visual interface is shown in Figure 3 as the two overlaid GUI windows (one showing correctly predicted degree of robustness for AF, and the other showing incorrectly predicted degree of robustness for STATIC) on top of the scatter plot. For a desired degree of robustness and a particular scheduling method, this detailed view helps in identifying the required execution scenario. Similarly, the visual output of the ANN model in Weka enables the identification of the required values of all input parameters in the execution scenario, when the output class attribute (degree of robustness) is plotted against against any other input attribute. For example, this can help in selecting the most flexible scheduling method to achieve a desired level of robustness, for a given problem size, system size and assumed variations in the system load characteristics.

Additionally, the MLPs learned by Weka can be exported and used online. For a given execution scenario, they can be used to predict the degree of robustness of each scheduling method in real time. Then, the scheduling method predicted to be the most robust can be selected to run the desired job.

MLP Confusion Matrix

Actual Class \ Predicted Class	1	2	3	4	5
1	1033	4	0	0	2
2	19	8	3	0	1
3	9	1	0	1	2
4	6	1	0	0	0
5	9	2	0	0	52

(a)

0-R Confusion Matrix

Actual Class \ Predicted Class	1	2	3	4	5
1	1039	0	0	0	0
2	31	0	0	0	0
3	13	0	0	0	0
4	7	0	0	0	0
5	62	0	0	0	0

(b)

Figure 2: The confusion matrices of (a) the MLP ANN and (b) the 0-R classifier. Each cell, i, j, gives the number of instances that were actually robustness class i (rows), but were predicted to be class j (columns). Thus, the main diagonal represents correct predictions.

As illustrated by the areas highlighted via the red circles in Figure 3, the STATIC, FSC and GSS scheduling methods contribute towards the largest number of incorrectly predicted degree of robustness values by the MLP ANN. The statistical results generated by Weka show that STATIC, FSC and GSS together contribute to 79% of the total number of incorrectly predicted values by the proposed MLP ANN model. This indicates that STATIC, FSC and GSS are less predictable compared to the other scheduling methods because, even though the MLP can model highly non-linear functions, it was still unable to consistently predict

the robustness of these three methods. All results are in confirmation with the theoretical and the experimental results obtained for these scheduling techniques in related previous work [1][4][5].

V. Benefits of the MLP ANN Flexibility Prediction Model

In previous work, a methodology to assess the robustness of the scheduling methods has been proposed to test their robustness against variations in processor availabilities at runtime. The methodology proposed in [4] establishes a general procedure to analyze the robustness of the dynamic loop scheduling (DLS) methods when they are used to execute scientific applications on non-dedicated heterogeneous computing systems. The robustness metric is used to quantify the robustness of the DLS methods, and to compare them with respect to their robustness values; however, the work presented in [4] is only a preliminary step towards analyzing the robustness of the the scheduling methods. The statistical calculations in that work are restricted to smaller test data sets.

In this work, an MLP ANN model is used to predict the robustness of the scheduling methods against fluctuating system load and captures more realistic execution scenarios. The advantages of using an ANN model over traditional statistical methods for predicting the *flexibility* of the scheduling methods are: (i) a capability to handle larger input datasets, (ii) a high real time prediction speed (approximately 1 second) as jobs are presented to a computing system, (iii) a capability to learn non-linear relations among input and class attributes (does not require any prior information related to the functional relation among the input and output attributes), and (iv) a capability to make correct predictions of robustness (of DLS) on data unexplored during training (verified by using 10-fold cross validation technique).

The MLP ANN model developed in this paper can be useful in the appropriate selection of the most *flexible* DLS method for achieving a desired level of robustness for a given application and execution scenario. The proposed ANN model can also be adapted to include additional input attributes (such as makespan, cost, power consumption, and others) for predicting the robustness, performance and execution cost of using a specific DLS method.

VI. Conclusions and Future Work

An MLP ANN model for predicting the flexibility of the scheduling methods for achieving robust dynamic load balancing has been developed in the work presented in this paper. The MLP classifier function of the Weka data mining tool was used to develop the ANN model. The input data set was generated from the experiments performed using the SimGrid simulation framework as reported in [5]. The test cases used to generate the data sets were generated using a combination of problem size, system size, DLS method, probability distribution showing the variation in iteration execution time, and probability distribution showing the variation in processor availability. The MLP ANN is trained and tested using the input data set, with the 10-fold cross-validation technique. The ANN learns the relation between the degree of robustness of the scheduling methods and the other input parameters through the backpropagation learning algorithm. The generated MLP ANN delivers a prediction accuracy of 94.9%. The visual output interface provided by Weka enables the analysis of the required values for the input parameters (N, P, the scheduling method, and others) for achieving a desired degree of robustness. The obtained MLP ANN has high computing speed. Therefore, it can be used online to determine the best DLS method for a job as it is submitted to a computing system. It has the ability to handle large input data sets, to adapt to new input data, and does not presume any functional relation between the input and the output attributes. This makes our ANN model generic and applicable to a wide range of application types and computing environments. Further, work is required to experiment with different values of the ANN parameters and to compare the prediction performance of ANNs with other learning techniques in AI. The immediate future work includes generating larger input data sets for achieving a higher prediction accuracy than that of the proposed MLP ANN. Additional metrics,

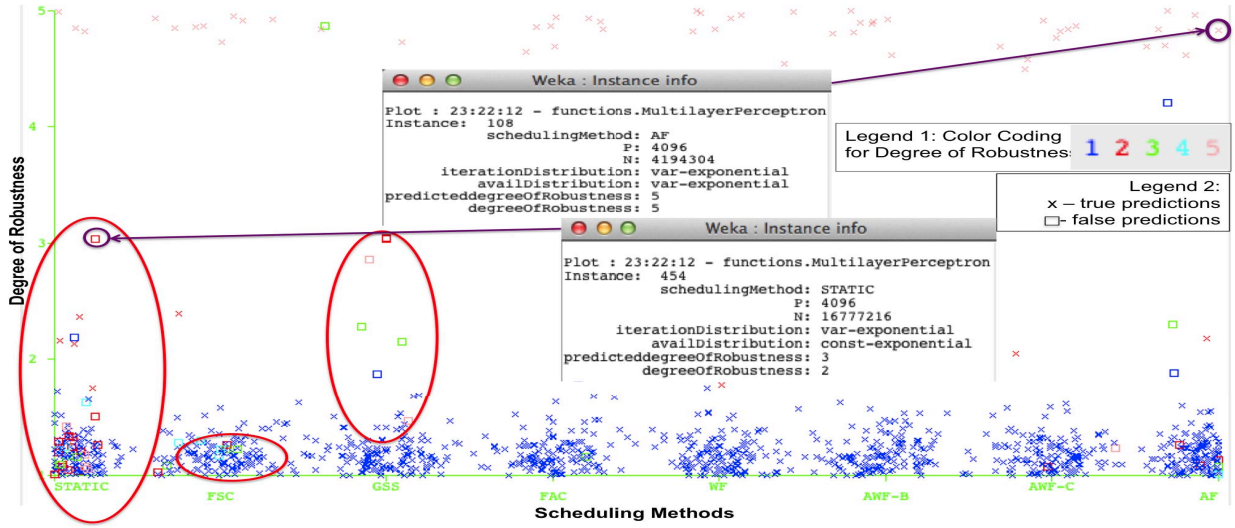

Figure 3: Degree of robustness predictions obtained from the MLP ANN model. The overlaid windows show: (top) correctly predicted value of the degree of robustness for AF scheduling method and (bottom) incorrectly predicted value of the degree of robustness for STATIC scheduling method. The areas highlighted with red circles show that STATIC, FSC and GSS contribute to 79% of the total number of incorrectly predicted values of the degree of robustness by the MLP ANN model.

such as cost and power consumption, can also be introduced into the proposed ANN for learning the functional relation between these attributes for achieving a desired level of performance.

Acknowledgments: This work is in part supported by the National Science Foundation under grant number NSF IIP-1034897, and by the German Research Foundation (DFG) in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

REFERENCES

[1] I. Banicescu and R. L. Cariño, "Addressing the stochastic nature of scientific computations via dynamic loop scheduling." *Electronic Transactions on Numerical Analysis*, vol. 21, pp. 66–80, 2005.

[2] I. Banicescu, F. M. Ciorba, and R. L. Cariño, "Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems," *International Symposium on Parallel and Distributed Computing*, 2009.

[3] S. Srivastava, I. Banicescu, and F. M. Ciorba, "Investigating the robustness of adaptive dynamic loop scheduling on heterogeneous computing systems," in the Proceedings of the *International Parallel and Distributed Processing Symposium*, Apr. 2010, pp. 1–8.

[4] S. Srivastava, N. Sukhija, I. Banicescu, and F. M. Ciorba, "Analyzing the robustness of dynamic loop scheduling for heterogeneous computing systems," *Parallel and Distributed Computing, IEEE Int. Symp.*, pp. 156–163, 2012.

[5] N. Sukhija, I. Banicescu, S. Srivastava, and F. M. Ciorba, "Evaluating the flexibility of dynamic loop scheduling on heterogeneous systems in the presence of fluctuating load using simgrid," in the Proceedings of the *International Parallel and Distributed Processing Symposium (IPDPSW-PDSEC 2013)*, 2013.

[6] F. Xia and Y. Sun, "Neural network based feedback scheduling of multitasking control systems," in *Proc. KES2005, LNCS, Springer-Verlag*, 2005, pp. 237–246.

[7] T. Eguchi, F. Oba, and S. Toyooka, "A robust scheduling rule using a neural network in dynamically changing job-shop environments." *IJMTM*, vol. 14, pp. 266–288, 2008.

[8] C. D. Polychronopoulos and D. J. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Transactions on Computers*, vol. 36, no. 12, pp. 1425–1439, Dec. 1987.

[9] C. P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Transactions on Software Engineering*, vol. 11, no. 10, 1985.

[10] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A method for scheduling parallel loops," *Communications of the ACM*, vol. 35, no. 8, pp. 90–101, Aug. 1992.

[11] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring," in *in Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997, pp. 318–328.

[12] S. Ali, A. A. Maciejewski, and H. J. Siegel, "Perspectives on robust resource allocation for heterogeneous parallel systems", *Handbook of Parallel Computing: Models, Algorithms, and Applications*, S. Rajasekaran and J. Reif, Ed. Chapman and Hall/CRC Press, 2008.

[13] E. Rich and K. Knight, *Artificial Intelligence*. McGraw-Hill Science/Engineering/Math, December 1990.

[14] G. Zhang, "Neural networks for classification: a survey," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 30, no. 4, 2000.

[15] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Comput.*, vol. 4, no. 1, pp. 1–58, Jan. 1992.

[16] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, 3rd ed. Pearson Education, 2010.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, Nov. 2009.