A Learning-based Selection for Portfolio Scheduling of Scientific Applications on Heterogeneous Computing Systems

Nitin Sukhija[*1], Brandon Malone[2], Srishti Srivastava[3], Ioana Banicescu[4], and Florina M. Ciorba[5]

[1.3.4]Department of Computer Science and Engineering, Mississippi State University, Mississippi State, USA

[2]Department of Computer Science, Helsinki Institute for Information Technology, University of Helsinki, Finland

[5]Center for Information Services and High Performance Computing, Technische Universität Dresden, Dresden, Germany

[*1]nitin@cavs.msstate.edu; [2]brandon.malone@cs.helsinki.fi; [3]ss878@msstate.edu;[4]ioana@cse.msstate.edu;[5]florina.ciorba@tu-dresden.de

***Abstract*-The execution of computationally intensive parallel applications in heterogeneous environments, where the quality and quantity of computing resources available to a single user continuously change, often leads to irregular behavior, in general due to variations of algorithmic and systemic nature. To improve the performance of scientific applications, loop scheduling algorithms are often employed for load balancing of their parallel loops. However, it is challenging to select the most robust scheduling algorithms that guarantee an optimized execution of scientific applications on large-scale computing systems. The computing systems comprise resources that are widely distributed, highly heterogeneous, often shared among multiple users, and have computing availabilities that cannot always be guaranteed or predicted. To address this challenge, in this work we focus on a portfolio-based approach to enable the dynamic selection and use of the most robust dynamic loop scheduling (DLS) algorithm from a portfolio of DLS algorithms, depending on the given application and current system characteristics, including the workload conditions. We provide a solution to the algorithm selection problem and experimentally evaluate its quality. We propose the use of supervised machine learning techniques to build empirical robustness prediction models that are used to predict a DLS algorithm's robustness for given scientific application characteristics and system availabilities. Using simulated scientific applications characteristics and system availabilities, along with empirical robustness prediction models, we show that the proposed portfolio-based approach enables the selection of the most robust DLS algorithm that satisfies a user-specified tolerance on the given application's performance degradation obtained in the particular computing system with a certain variable availability. We also show that the portfolio-based approach offers higher guarantees regarding the robust performance of the application using the automatically selected DLS algorithms when compared to the robust performance of the same application using a manually selected DLS algorithm.**

Keywords-Dynamic loop scheduling; Robustness; Algorithm selection; Empirical robustness prediction models; Machine learning techniques; Variable system availability

I. INTRODUCTION

As the mainstream computing technology enters into the post petascale era, the number and complexity of its computing components is sharply increasing[1]. With this advancement in the computing systems, promising exascale performance by the beginning of the next decade, the development and use of large-scale scientific applications is becoming increasingly prevalent for solving a multitude of problems in various science, engineering, and society areas. Scientific applications are often computationally intensive and comprise of repetitive computations in the form of program loops with iterations. Such loops are a major source of parallelism in scientific applications and can be executed concurrently on available computing resources. However, running these applications in dynamic heterogeneous computing environments yields an irregular behavior, in general due both to variations of algorithmic and systemic nature, leading to a degradation of the parallel execution performance. Major factors that account towards the performance degradation of scientific applications include inter-processor communication, workload imbalance among processors, overhead of managing parallelism, and others. In general, load imbalance is due to unpredictable problem, algorithmic, and systemic variances, and often is the major factor for degradation of application performance.

Dynamic loop scheduling (DLS) techniques are considered to be the key solution for achieving and preserving the best performance of computationally intensive scientific applications in dynamic heterogeneous computing environments[2]. However, to guarantee an optimal level of performance for a scientific application executing in a dynamically changing and unpredictable environment, it is required to appropriately select the scheduling technique which will enable the most robust execution of the application against various perturbations that may arise in the computing environment during its execution. A DLS algorithm employed for scheduling the computations of applications on a heterogeneous computing system is considered to be robust, if it can handle the largest variation in the existing perturbations with a lowest impact on performance. The perturbation factor considered in this paper is the *fluctuation in system load*, which results in the variation in the delivered computation speed of the processor expressed as percentage availability of the processor. A processor is considered to be unloaded if it is 100% available to compute, i.e., there is no other task running on it and this processor is, therefore, fully available to process a new task. A processor is considered to be loaded when it is available for processing only for a fraction of

its overall delivered computational speed. The fluctuation in system load results in the variation in the availability of the system. Thus, the variation in system availability may be a result of many factors, such as, the variance in processor speeds, architecture, power, and others[3,4]. The variation of the execution times of individual application tasks is a combined effect of the variations in the computational cost of a particular task and of the system availability during its execution.

In earlier work, the robustness of the DLS methods has been studied and two metrics have been mathematically formulated to quantify the robustness of DLS techniques in the presence of unpredictable variations in systemic perturbations[5] inspired by existing literature on robustness of resource allocation[6]. One of these metrics is the *flexibility metric, which* represents the robustness of an application employing a certain DLS in the presence of variations in system availability (due to fluctuations in system workload) during the running time of the application on the system. The second *metric is the resilience metric, which* denotes the robustness of an application using a certain DLS method in the presence of processor failures that occur during the execution of the application. The flexibility metric and the developed robustness analysis have been successfully employed to evaluate the robustness of a number of DLS algorithms against system load fluctuations using discrete event simulation[7,8]. Throughout this paper, the terms *flexibility* and *robustness in presence of system availability variation* are used interchangeably. The expressions "flexibility (or robustness) of an application using a certain DLS" and "flexibility (or robustness) of a certain DLS" have equivalent meanings and are also used interchangeably.

The flexibility of scheduling algorithms often varies among execution instances, where an instance is an assembly of four attributes: problem size, system size, characteristics of the variations in the application task execution times, and those of the processor availabilities. Since the DLS algorithms employ probabilistic rules, the use of these rules to parallelize and execute applications in dynamic environments results in application runtimes that vary from run to run for a single DLS and among the DLS algorithms. Consequently, this poses a challenge of deciding *a priori* which algorithm is most robust in obtaining the best performance of a given scheduled application onto a given computing system in the presence of variable system availability. This motivates the investigation into the use of machine learning techniques to address the challenge of selecting effective scheduling algorithms for a broad spectrum of scientific applications, thus underscoring their relevance in large-scale computing.

In recent work[9], a proof of concept engaging a multilayer perceptron (MLP) artificial neural network (ANN) was presented for predicting the flexibility of individual DLS algorithms in parallel and distributed computing environments. However, the problem of deciding at runtime which DLS algorithm to use for a given instance was not addressed. The goal of the present work is not only to retrospectively predict the performance of an algorithm on a particular execution instance, but also to *enable a dynamic selection* of the most robust algorithm from a *portfolio* of DLS algorithms for scheduling scientific applications for any new instance.

The problem we aim to solve is stated as follows: given a scientific application, a collection of DLS algorithms, and a computing system, which algorithm should be employed to guarantee the robust execution of the application in this system in the presence of algorithmic and systemic variances? This problem translates into the *algorithm selection problem*[10] for a portfolio of DLS algorithms. The most widely adopted solution to this problem is the winner-take-all approach[11]. Using this approach, the robustness of each DLS algorithm can be measured on a representative set of instances, and the algorithm which has the highest robustness value can be selected for scheduling. However, this approach can lead to overlooking of some algorithms that are not robust on certain instances but offer optimal performance on others.

To solve the problem of selecting a scheduling algorithm, in this work we use machine learning algorithms to learn empirical hardness models[12] which act as predictors of a DLS algorithm's robustness for a given instance based on the learning acquired through the four attributes of the instances and the algorithm's past robustness values. Significantly extending previous work[9], herein we formally define *empirical robustness prediction models*, which predict the robustness of a DLS algorithm on a given instance. This is a regression problem; consequently, we explore the vast space of regression model classes, their hyperparameters and parameters which solve the regression problem, and select the model which best predicts the robustness of a scheduling algorithm on any instance. The model offers the basis for selecting the most robust algorithm from a DLS *algorithm portfolio* for a given instance based on its characteristics. The major contributions of this paper are threefold: (i) Employing state-of-the-art machine learning techniques to explore a large space of empirical robustness prediction models. Consequently, the learned models predict robustness much more accurately than the previous models[9]; (ii) Selecting DLS algorithms from a portfolio with the developed empirical robustness prediction models. The prediction models enable an algorithm selection on a per-instance basis; (iii) Evaluating experimentally the performance of the per-instance selections compared to the simpler winner-take-all approach. The results show that algorithm selections based on the empirical robustness prediction models yield more robust performance on large number of instances than the selections using winner-take-all.

The rest of the paper is organized as follows. A review of the DLS algorithms and their robustness, together with a description of representative empirical robustness prediction models are presented in Section II. The design and an outline of a methodology to select the most robust DLS algorithm are described in Section III. The simulation analysis and the evaluation of the experimental results are discussed in Section IV. The related work is presented in Section V. The conclusions and possible future directions are summarized in Section VI.

II. BACKGROUND

A. Dynamic Loop Scheduling

Dynamic loop scheduling (DLS) algorithms have been developed with the goal of achieving dynamic load balancing through optimized workload assignment. For a given problem size N and system size P, the DLS algorithms dynamically determine the work for each of the P processors as a function of chunks of application tasks (loop iterations). For achieving good load balancing on variably loaded resources when executing tasks with variable execution times, the DLS algorithms provide two alternative approaches: *non-adaptive* and *adaptive*. A detailed survey and comparison of the non-adaptive and the adaptive DLS algorithms can be found in[2,13]. Non-adaptive algorithms determine workload sizes based on *a priori* information and assumptions made before the loop executions. Examples of non-adaptive DLS algorithms include self scheduling(SS), guided self scheduling(GSS), fixed size chunking (FSC), factoring (FAC), and weighted factoring (WF). Subsequent efforts gave birth to more elaborate DLS algorithms, called adaptive DLS algorithms. Examples of adaptive DLS algorithms include adaptive weighted factoring (AWF) and its variants AWF-B and AWF-C, and adaptive factoring (AF). The adaptive algorithms use a combination of runtime information about the application and the system, in order to predict the system's capabilities for the next computational assignments, or to estimate the time future tasks will require to finish execution, in order to achieve the best allocation possible for optimizing application performance via load balancing. These techniques use different probabilistic models to dynamically compute the size of chunks at runtime, such that the execution of the application completes before the optimal time with high probability. The type of probabilistic model used by different DLS algorithms describes the differences in their adaptivity, complexity, and generality. Therefore, different DLS algorithms may give different performance results when executing in various environments. Our present portfolio consists of a set of eight scheduling algorithms, $\mathbf{s}$ = {STATIC, FSC, GSS, FAC, WF, AWF-B, AWF-C, AF}. STATIC is the straightforward parallelization method.

B. Robustness of Dynamic Loop Scheduling

Scheduling scientific applications onto large-scale platforms, where chances of uncertainties are high, requires an approach that insures achieving optimal performance via the algorithms employed for parallelizing these applications. This motivates the robustness study of the DLS algorithms. The robustness analysis enables quantifying the performance of the DLS algorithms against perturbations or environmental variations, in order to allow the selection of the most robust DLS algorithm. The selection of robust DLS algorithms for dynamic load balancing of scientific applications insures that the total parallel execution time does not exceed a user-specified tolerable limit, defined as $\tau_{user} \geqslant 1$, when processors comprising a computing system have variable availabilities.

In earlier and in the present work, the robustness of a DLS algorithm is evaluated by measuring the impact of a specific variation in processor availabilities on the total parallel execution time, T_{PAR}, of the application executing on the (non-dedicated) computing system using the particular DLS algorithm. Thus, for a DLS algorithm to be robust, T_{PAR} must not exceed the smallest expected parallel execution time of the application, T_{PAR}^{ideal} by a factor of τ_{user}.

Mathematically, this is expressed by the following inequalities:

$$T_{PAR}^{ideal} \leqslant T_{PAR} \leqslant \tau_{user} \cdot T_{PAR}^{ideal} \quad (1)$$

In the above inequalities, varying the lower and upper bounds on the values of the tolerance factor, τ_{user}, leads to imposing different constraints on applications of different types, such as time critical or non-time critical applications. As stated earlier, robustness metrics have been proposed to study the robustness of the DLS algorithms[5]. A simulation model implementing DLS algorithms in the SimGrid simulation framework[14] was proposed in[15], and a procedure to investigate the robustness of the DLS algorithms against variations in system availability using this model has been demonstrated in[8].

C. MLP ANN as an Empirical Model for Robustness Prediction

Forecasting robustness 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. To insure the robustness of the DLS methods for a considerably larger number of execution instances resulting from considering different combinations of problem sizes, number of processors, and scheduling methods, an artificial neural network (ANN) analysis has been performed to investigate the robustness of dynamic loopusing Weka[9]. A multilayered perceptron (MLP) ANN model has been used 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 attributes mentioned above. The ANN is trained on a part of the input data set and tested with a different subset of the input data set. Furthermore, when exposed to new data, the MLP ANN is capable of accurately predicting the DLS robustness. This work has been presented as a proof of

concept that ANNs can be employed (as an empirical learning model) to predict the flexibility of DLS in the presence of fluctuating system load.

The robustness prediction problem has been presented as a classification problem, where each technique is classified into a robustness category based on the impact on T_{PAR} that results from employing the particular technique for a given or specified value of τ_{user}. The robustness categories are 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. The prediction results obtained from using the MLP ANN have been compared to the naïve 0-R classifier as shown in Fig. 1. 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 allow a scheduler to effectively decide which loop scheduling algorithm will best adhere to the user constraints. In contrast, the incorrect predictions by the 0-R classifier could lead to very poor decisions about which scheduling algorithm to choose. The MLP ANN was unable to distinguish robustness classes 3 and 4 because of the limited training. A larger training set would allow the learning algorithm to better distinguish the characteristics of these classes. 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. The limitations of using MLP ANN form the motivation to experiment with the other empirical models of learning and their associated learning parameters for achieving a better level of prediction accuracy. The different empirical prediction models and their hyperparameters are explained in detail in the following section.

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)

Fig. 1 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.

D. Identifying Good Empirical Robustness Prediction Model Classes and Hyperparameters

It is difficult to *a priori* select a particular prediction model class (e.g., neural networks, decision trees, etc.) and its hyperparameters (e.g., the number of nodes to use in the middle layers of a neural network) that will best predict the tolerance of a DLS algorithm on an instance. Given the model class and hyperparameters, the parameters for a model can be learned using standard techniques (e.g., backpropagation for learning the weights in neural networks).Auto-WEKA[16] addresses this problem by searching for model classes and hyperparameters that best explain a given dataset. In particular, if provided with a sequential model-based optimization (SMBO) strategy, a training set, and a validation set, Auto-WEKA uses a local search through the model classes and hyperparameters exposed by Weka[17] to find good model classes and hyperparameters to predict the class variable of a given dataset. We briefly describe the SMBO strategy and Auto-WEKA in following subsection, while we refer the reader to [16] and the references therein for more details.

1) Auto-WEKA and Sequential Model-Based Optimization:

The sequential model-based optimization algorithm (SMBO), the pseudocode of which is listed in Algorithm 1, is an optimization framework which searches through the combined state space of model classes and their hyperparameters, collectively referred to as hyperparameters $\mathcal{E}$, to minimize an objective function, f, on a dataset D. The best hyperparameters found so far are referred to as $\mathcal{E}^*$. First, the SMBO algorithm constructs a model, M_f to estimate the dependence of f on $\mathcal{E}$ and D (Algorithm 1, line 1).Then, M_f is used to select a new set of hyperparameters $\mathcal{E}'$(Algorithm 1, line 4), and f is evaluated on $\mathcal{E}'$ and D(Algorithm 1, line 5).Next, if $f(\mathcal{E}', D)$ is better than $f(\mathcal{E}^*, D)$, then $\mathcal{E}^*$ is updated to be $\mathcal{E}'$ (Algorithm 1, line 6). Finally, the new information, $(\mathcal{E}', f(\mathcal{E}', D))$, is used to update M_f(Algorithm 1, line 8).

Algorithm 1 Sequential model-based optimization (SMBO)

Input: objective function f, training dataset D
Output: hyperparameters $\mathcal{E}^*$
1: initialize M_f › M_f gives estimates of $f(\mathcal{E}, D)$
2: $\mathcal{E}^* \leftarrow$ hyperparameters from SMAC using M_f
3: **while** not out of time **do**
4: $\mathcal{E}' \leftarrow$ hyperparameters from SMAC using M_f
5: **if** $f(\mathcal{E}', D) < f(\mathcal{E}^*, D)$ **then**
6: $\mathcal{E}^* \leftarrow \mathcal{E}'$
7: **end if**
8: update M_f with $(\mathcal{E}', f(\mathcal{E}', D))$
9: **end while**
10: **return** $\mathcal{E}^*$

The next set of hyperparameters $\mathcal{E}'$ are selected using the *positive expected improvement* (PEI)[18] of $\mathcal{E}'$ over $f(\mathcal{E}^*, D)$. The PEI calculation uses M_f to estimate the probability distribution $p(f(\mathcal{E}', D) < f(\mathcal{E}^*, D))$. This estimation primarily requires $p(f(\mathcal{E}', D) = c|\mathcal{E}')$. We configure Auto-WEKA to use sequential model-based algorithm configuration (SMAC)[19], the default configuration, to learn this probability distribution. SMAC assumes $p(f(\mathcal{E}', D) = c|\mathcal{E}')$ is Gaussian-distributed. It represents M_f as a random forest and uses it to estimate the mean and variance of the Gaussian distribution. In addition to selecting models which optimize $p(f(\mathcal{E}', D) = c|\mathcal{E}')$, SMAC also sometimes selects hyperparameters at random to better explore the search space. This randomness insures that good models can be found even if M_f does not model $p(f(\mathcal{E}', D) = c|\mathcal{E}')$ very well.

TABLE I PARAMETERS USED IN SIMGRID'S SIMULATION TOOLKITFOR GENERATING THE INPUT DATASET

Parameter: notation (unit of measure)				**Values used in the simulation**				**#comb.**
Problem size: N (tasks)				1,048,576	4,194,304	16,777,216		3
System size: P (processors)				2048	4096	8092		3
Load imbalance	**Problem**		**Algorithmic Variance: T (flops)**	Constant $2X10^8$	Gaussian $\mu=9.5X10^8$, $\sigma=7X10^7$	Gamma $\mu=1.1X10^8$	Exponential $\mu=3X10^8$	4
	System	**Processor**	**Systemic Variance: A(%)**	Constant 100	Uniformly distributed bound within [10,100]	Exponentially distributed $\mu=0.3$		3
			Power rating : pow (flops)	10^9				1
		Network	**Link latency: lat (s)**	$16.6X10^{-6}$				1
			Link bandwidth: bw (B/s)	$1.25X10^9$				1
Scheduling algorithm: S				STATIC, FSC, GSS, FAC,WF, AWF-B, AWF-C, AF				8

III. METHODOLOGY FOR SOLVING THE DLS SELECTION PROBLEM

This section gives a detailed overview of the methodology for developing a DLS portfolio to solve the DLS algorithm selection problem. The basic idea is to construct a portfolio of dynamic scheduling algorithms, and to select from it the most robust algorithm for the current application and system characteristics. A detailed workflow illustrating the design and organization of the methodology used to solve the DLS algorithm selection problem is shown in Fig.2. This workflow elucidates the problem of selection of an algorithm for scheduling individual tasks of a scientific application executing on dedicated or non-dedicated computing systems (where the availability of the computing resources allotted to a given application may vary during runtime). The steps comprising the proposed methodology and the algorithm used for selection are discussed in detail in the following subsections.

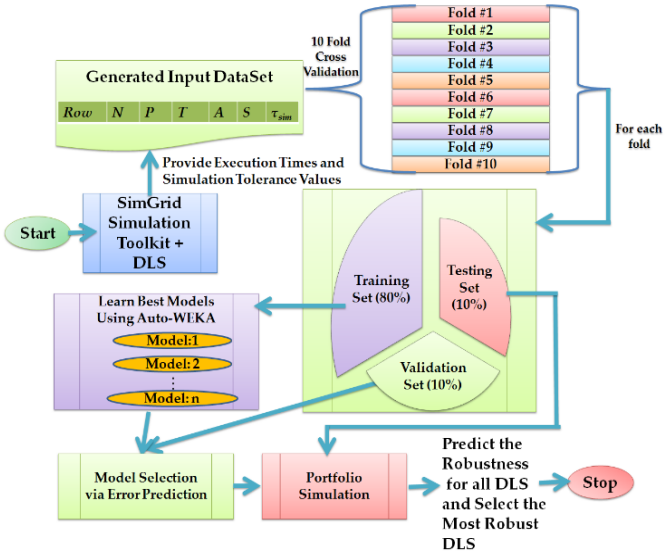

Fig. 2 The proposed workflow for selecting the most robust DLS algorithm

A. Constructing the Input Dataset of Instances

Algorithm 2 shows the pseudocode of the algorithm employed to construct the input dataset comprising the instances as well as other attributes. In this work we used the SimGrid simulation framework to model, control, reproduce, and monitor large-scale computing systems toanalyze the performance of the DLS algorithms on computationally intensive scientific applications. We have chosen SimGrid to implement the DLS algorithms and to evaluate their robustness because it is a versatile tool and is also validated for the evaluation of scheduling algorithms in distributed and heterogeneous computing systems, such as grids, clouds or peer-to-peer systems[20-22]. SimGrid's deployment and platform files facilitate the creation of the execution scenarios reflecting load imbalance for performing an experimental evaluation of the robustness of the DLS algorithms. The deployment file contains information on modeling both the scientific applications and the heterogeneous, non-dedicated computing systems.

Algorithm 2 Creating the training dataset

Input: problem size N, computing system P, algorithmic variances T, systemic variances A, DLS algorithm portfolio **s**
Output: dataset D
1: dataset D ← Ø
2: **for** each combination c of N, P, T and A **do**
3: T_{PAR}^{ideal} ← ideal parallel execution time of c
4: **for** each algorithm s_c **s do**
5: T_{PAR} ← total parallel execution time of instance c using s
6: $\tau_{sim} \leftarrow T_{PAR}/T_{PAR}^{ideal}$
7: $D_{s_c} \leftarrow \tau_{sim}$
8: **end for**
9: **end for**
10: **return** D

The description of hosts (or processing resources), processor availabilities, network links, and routing paths exhaustively describing a highly irregular computing system is given via the platform file.

The computing system being modeled is comprised of P processors where the availability to compute or the computing speed of each processor either has no variation, or it varies over time according to the uniform or exponential probability distributions, and the computing speed of a processor in both cases (constant or variable) differs from the speed of other processors; thus, we are considering heterogeneous computing systems. The network bandwidth and latency values considered for modeling the computing system are 1.25×10^9 bits per second and 16.6×10^{-6} seconds, respectively.

The systemic variance in the computing system, denoted as A, is modeled by varying the availabilities of the processors during the runtime of the application. The choice of the probability distributions representing the parameters' variability is made to designate highly regular and irregular applications and systems, and to allow the study of the effects of such irregularities on the robustness of the DLS techniques. The dataset given as input to the candidate empirical robustness prediction models is organized as rows. The instances used to populate each row of the input dataset were generated using different combinations of (shown in Table I):

- Problem sizes, N, comprising 1,048,576, 4,194,304, and 16,777,216 individual tasks,
- Computing system sizes, P, containing 2048, 4096, and 8092 processors,
- Probability distributions underlying T, modeled as constant with each task computation time of 2×10^8 flops, as Gaussian with mean $\mu = 9.5\times10^8$ flops and standard deviation $\sigma = 7\times10^7$ flops, as gamma with mean $\mu = 1.1\times10^8$ flops, or as exponential with mean $\mu = 3\times10^8$ flops, and
- Probability distributions underlying A, constant-availability where the availability of each processor follows a uniform distribution in the interval [0.1,...1.0] (*const-uni*) or is exponentially distributed with mean $\mu = 0.3$ but does not vary with time i.e., remains unchanged during the entire course of the simulation(*cont-exp*). Furthermore, variable availability where the availability of a processor drawn either from the uniform (*var-uni*) or exponential distributions (*var-exp*) with the above parameters periodically varies during the execution time of the application.

As shown in Algorithm 2(line 3), the algorithm computes the T_{PAR}^{ideal} values for each scientific application comprising N independent tasks with **constant** T or **variable** T as:

$$T_{PAR}^{ideal} = T_{SEQ} / P, \tag{2}$$

where the sequential time T_{SEQ} represents the total execution time of the N tasks on the fastest available processor of the computing system considered to have a computing speed of 10^9 flops/s. Also, in equation (2), P denotes the number of processors in our simulated system that are **100% available (dedicated)** throughout the simulation, each with maximum speed (or power attribute) of 10^9 flops/s.

Then, the total parallel execution time, T_{PAR} for a particular $s \in \mathbf{s}$ (Algorithm 2, line 5) is obtained upon employing a certain DLS algorithm s from the DLS portfolio $\mathbf{s}$ to schedule a scientific application consisting of N independent tasks with **constant** T (fixed task computation costs) or with **variable** T (task computation costs following Gaussian, gamma, or exponential distributions) onto a non-dedicated heterogeneous system with P processors with **variable** A (processor availabilities varying over time according to the uniform or exponential distributions with the parameters mentioned above). T_{PAR} acquired via simulation captures the compound effect of the major sources of load imbalance, i.e., algorithmic (T) and systemic variances (A) on the performance of a particular s.

The robustness of s defines its performance in the presence of both T and A. The tolerance value observed via simulation, τ_{sim}, reflects the robustness of each s for a particular instance c represented by the 4-tuple: (N, P, T, A), and is computed (Algorithm 2, line 6) as:

$$\tau_{sim} = T_{PAR} / T_{PAR}^{ideal} \tag{3}$$

The τ_{sim} denotes the simulated impact of variable system availability on the total parallel execution time, T_{PAR}. Various τ_{sim} values reflect different degrees of robustness for a certain s. For instance, $\tau_{sim}=1$ implies that the use of s results in a T_{PAR} that satisfies the inequality on the right hand side of (1), and that s has the maximum degree of robustness. Achieving the maximum degree of robustness implies that the performance of the application is unaffected by runtime perturbations when it is scheduled using s. The value of τ_{sim} reflects an increase or decrease in the robustness of a DLS algorithm, i.e., for a given instance c and a given DLS algorithm s, a lower value of τ_{sim} implies a higher degree of robustness for the corresponding DLS algorithm.

To insure best performance, the total parallel execution time, T_{PAR}, of a scientific application running on a system must not exceed the best sequential time of that application and mathematically this is written as:

$$T_{PAR} \leqslant T_{SEQ} \tag{4}$$

From (1) and (4) the following inequalities can be derived:

$$T_{PAR}^{ideal} \leqslant T_{PAR} \leqslant \min(T_{SEQ}, \tau_{user} \cdot T_{PAR}^{ideal}) \tag{5}$$

Therefore, an upper bound on τ_{sim} can be obtained from (3) and (5) as:

$$\tau_{sim} \leqslant \min(P, \tau_{user}) \tag{6}$$

The input dataset used in this work contained 1,152 samples (one per row) and an illustration of its structure is presented below:

Row	N	P	T	A	S	τ_{sim}
1	4,194,304	8,092	Gaussian	expo.	AF	1.2
2	1,048,576	2,048	gamma	uniform	FSC	1.65
…						
1,152	16,777,216	4,096	constant	expo.	AWF-B	2.9

B. Learning DLS Empirical Robustness Prediction Models:

Algorithm 3 shows the pseudocode of the algorithm we use to learn the empirical robustness prediction models from the dataset described in Section III-A.

The empirical robustness prediction models $\mathcal{E}_i$ give a mapping from an instance c, the 4-tuple (N, P, T, A), and an algorithm $s \in S$ to τ_{pred}, the predicted tolerance of s on those 4 attributes. This algorithm also gives the logic for making the tolerance predictions. Our dataset contains 144 distinct instances, so we cannot afford to hold out a sizeable portion of our dataset for testing. We address this limitation with the standard procedure of stratified 10-fold cross-validation. That is, we partition our dataset into 10 subsets (Algorithm 3, line 1) and run the workflow 10 separate times (Algorithm 3, lines 2 to 8). Due to stratification, the distribution of robustness values in each subset is similar to the distribution in the entire dataset.

Algorithm 3 Learning empirical robustness prediction models

Input: dataset D, random seeds V
Output: empirical robustness prediction models $\mathcal{E}$
1: $D_1, \ldots, D_{10} \leftarrow$ stratified 10-fold split of D
2: **for** $i \in \{1, \ldots, 10\}$ **do** › 10-fold cross-validation
3: $D^i_{training} \leftarrow D \setminus D_i \setminus D_{i+1}$
4: $D^i_{validation} \leftarrow D_i$, $D^i_{testing} \leftarrow D_{i+1}$
5: $\mathcal{E}_1, \ldots, \mathcal{E}_{|V|} \leftarrow$ run Auto-WEKA with V on $D^i_{training}$
6: $\mathcal{E}^* \leftarrow arg\ min_{\mathcal{E} \in \mathcal{E}_1, \ldots, \mathcal{E}_{|V|}} f(\mathcal{E}', D_{validation})$
7: $\mathcal{E}_i \leftarrow \mathcal{E}^*$
8: **end for**
9: **return** $\mathcal{E}$

Each time, we use 8 subsets for training (Algorithm 3, line 3), one subset for validation and the final subset for testing (Algorithm 3, line 4).Each subset is used once for validation and once for testing.

In our previous work, we trained a multilayer perceptron with hand-tuned hyperparameters to serve as an empirical robustness prediction model [9]. Our new workflow (shown in Fig. 2) adopts a much more general strategy: we run Auto-WEKA on the training set to search both for good model classes and hyperparameters. As objective function f within the SMBO component of Auto-WEKA, we use the root mean squared error, which is the average square root of the sum of the squares of the differences between the observed tolerance τ_{sim} and the predicted tolerance τ_{pred} for a given instance c and DLS algorithm s. That is, for model $\mathcal{E}$ and dataset D,

$$\mathbf{f}(\mathcal{E}, \boldsymbol{D}) = \frac{\mathbf{1}}{|\boldsymbol{D}|}\sqrt{\sum_{(\boldsymbol{S},\boldsymbol{c})\in \boldsymbol{D}}(\boldsymbol{\tau_{sim}} - \boldsymbol{\tau_{pred}})^{\mathbf{2}}} \quad (7)$$

We selected this objective function because it penalizes large errors more heavily than smaller ones. Consequently, incorrect predictions with small errors are still useful for algorithm selection. As a simple example, suppose the observed τ_{sim} values for dataset D are {1.5, 2.5, 1.75} and that the respective τ_{pred} values using model $\mathcal{E}$ are {1.8, 2.2, 1.8}. Then, using root mean squared error as f, $f(\mathcal{E}, D) = \sqrt{(1.5-1.8)^2 + (2.5-2.2)^2 + (1.75-1.8)^2} = 2.5$. The SMBO component of Auto-Weka will use this information when selecting the next set of hyperparameters $\mathcal{E}'$.

As described in Section II-D, Auto-WEKA internally uses some randomization; consequently, its developers suggest running it several times with different random seeds and selecting the best model it finds. For each training set, we used 8 random seeds (in particular, V = {1, 5, 7, 10, 1001, 1005, 1007, 1010}) and allowed Auto-WEKA to run for 2 hours for each seed (Algorithm 3, line 5).

The above mentioned training phase results in 8 candidate models, $\mathcal{E}_1, \ldots, \mathcal{E}_{|V|}$. As a final step to limit overfitting, the root mean squared error, f, of each model is calculated on the validation set, and we select the model with the lowest error, $\mathcal{E}$ (Algorithm 3, line 9). We collect $\mathcal{E}^*$ from each fold to use in the algorithm selection (Algorithm 3, line 9), described in Section III-C.

C. Selecting an Algorithm from the Portfolio during Online Simulation:

Algorithm 4 shows the pseudocode of the algorithm employed for the selection of DLS algorithms from the portfolio. The algorithm loops over each distinct instance c (Algorithm 4, lines 3 to 8). It then uses the predictions made in Algorithm 3 to select the algorithm with the lowest predicted robustness for c (Algorithm 4, line 5).This algorithm selection process simulates real online behavior because the models are used to make predictions on instances not seen in training. Due to the cross-validation, we always make predictions with the model $\mathcal{E}_I$ such that $(S, c) \in D^i_{testing}$. Finally, these selections are used to evaluate the efficacy of the empirical robustness prediction models and the performance of the portfolio (Algorithm 4, line 9).

Furthermore, Algorithm 4 describes algorithm selections for the *virtual best solver* (VBS).The VBS gives the theoretical best behavior of a portfolio comprising the algorithm set s. In particular, it uses the complete dataset D as an oracle to select the most robust algorithm for each instance (Algorithm 4, line 7).We also use these selections in our evaluation to compare the

algorithms selected with the empirical robustness models to the theoretical best selections (Algorithm 4, line 9)

Algorithm 4 Selecting robust algorithms during online simulation

Input:: empirical robustness prediction models $\mathcal{E}$, dataset D, problem size N, computing system P, algorithmic variances T, systemic variances A, DLS algorithm portfolio **S**

Output: algorithm selection $S^{portfolio}$ and S^{VBS}

1: portfolio algorithm selection $S^{portfolio} \leftarrow \emptyset$

2: VBS algorithm selections $S^{VBS} \leftarrow \emptyset$

3: **for** each combination $c = (N, P, T \text{ and } A)$ **do**

4: › pick the algorithm with the lowest τ_{pred} for the portfolio

5: $S_c^{portfolio} \leftarrow arg\ min_{S \in \mathbf{S}} \mathcal{E}(S, c)$

6: › pick the algorithm with the lowest τ_{sim} for the VBS

7: $S_c^{VBS} \leftarrow arg\ min_{S \in \mathbf{S}} D_{S,c}$

8: **end for**

9: **return** $S^{portfolio}$, S^{VBS}

TABLE II CHARACTERISTICS OF THE MODELS SELECTED BY AUTO-WEKA DURING EACH CROSS-VALIDATION STEP

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
Attributes	**Decision Table**	**Bagging (22 REPTrees)**	**REP Tree**	**Additive Regression (2 REPTrees)**	**REP Tree**	**REP Tree**	**Multilayer Perceptron**	**REP Tree**	**REP Tree**	**Decision Table**
P	1	555	28	18	16	18	15	14	40	1
N	1	445	17	18	18	14	16	15	20	1
T=gamma	0	492	22	24	14	18	10	22	24	0
T=exp	0	712	42	40	30	38	36	22	40	0
T=const	0	356	12	14	16	14	10	6	14	0
T=Gauss	0	274	6	8	8	6	6	4	12	0
A=var-uni	0	294	14	22	14	16	14	14	14	1
A=var-exp	0	278	8	20	12	12	12	10	10	1
A=const-uni	1	192	10	8	4	4	6	6	10	1
A=const-exp	1	166	10	8	4	4	2	6	6	0
Static	1	44	2	4	2	2	2	2	2	1
GSS	1	44	2	4	2	2	2	2	2	1
FSC	0	254	8	16	6	8	8	12	12	0
AF	0	158	6	4	2	4	6	4	8	0
FAC	1	210	10	20	6	8	6	16	16	1
WF	0	150	12	4	2	2	2	4	10	1
AWF-B	0	84	2	0	0	0	2	0	4	0
AWF-C	0	110	0	2	0	0	0	2	4	0

IV. ANALYSIS AND RESULTS OF THE PORTFOLIO-BASED APPROACH

This section describes the results of learning the empirical robustness models and their efficacy in selecting robust DLS algorithms.

A. Empirical Robustness Model Classes Selected with Auto-WEKA

As described in Section III-B, Auto-WEKA is run with eight different random seeds and produces eight different models, $\mathcal{E}_1, \ldots, \mathcal{E}_8$ on line 5 in Algorithm 3, for each training dataset. Auto-WEKA was run for two hours for each seed on computing nodes with a dual quad-core Intel Xeon processor (2833 MHz) and 32 GB of RAM. Four seeds were run concurrently in different processes. Then, the validation dataset is used to select the best model among those, $\mathcal{E}^*$on line 6 in Algorithm 3.That process is repeated for each of the ten datasets (the union of which forms the input dataset described in Section III-A) created during cross-validation. Thus, in total, eighty model classes, hyperparameters and parameters are learned with Auto-WEKA, and ten of those are selected based on their root mean squared error on the validation set. Somewhat surprisingly, Fig. 3 shows that simple model classes like decision trees and decision tables were selected far more often than complex model classes like

Gaussian processes (never selected as an $\mathcal{E}^*$) or support vector machines (never even selected as $\mathcal{E}_i$, so they do not appear in the figure).The superior performance of the simple models over the more complex ones on the validation datasets suggests that the hyperparameters learned by Auto-WEKA may overfit the training data.

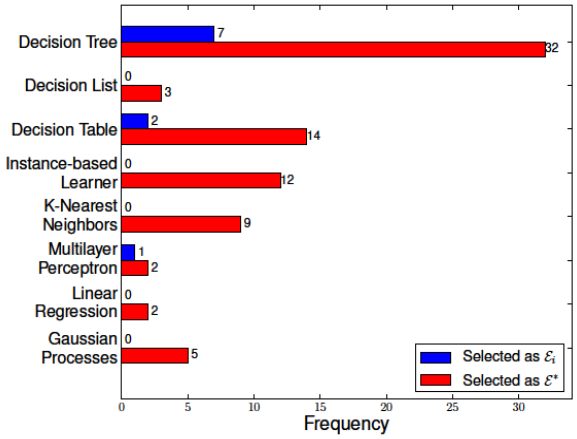

Fig. 3 Frequency of model classes learned by Auto-WEKA and selected based on performance on the validation set. The best models learned by Auto-WEKA were aggregated by their model class, regardless of hyperparameters. Sometimes, Auto-WEKA used a "meta" model, such as boosting. These meta models extend one of the base models, such as a decision table. We grouped those according to the base model.

1) Discussion:

In order to better understand how the system and application characteristics affect tolerance, we inspected the models in more detail. In particular, in Table II, we show how often each attribute was selected in the best model from each cross-validation fold. This table shows the characteristics of the models selected by Auto-Weka during each cross-validation step. Each column corresponds to one fold. The first row gives the fold number, and the second row describes the model class selected for that fold. The remaining rows give the number of times the indicated attribute was used in that model. All of the models selected during our validation phase were either based on decision trees or decision tables. The following discussion focuses on the decision trees, but the tables exhibit similar trends.

Typically, the decision trees first split on algorithms and then the other characteristics of the problem. This suggests that, in most cases, little information is shared among the subtrees corresponding to the different algorithms. Thus, future research could focus on learning distinct models for each algorithm. Nevertheless, though, the AWF methods are used much less often in the trees than the other algorithms. This suggests that some information is shared among the models about these algorithms. FSC and FAC are both used quite often in the trees, so their behavior may sometimes offer insight into the behavior of the other algorithms.

Possibly as expected, N and P are used many times in all of the learned models. For the processor distributions (A), the decision trees also frequently split on *var-uni* and *var-expo*. This suggests that many refinements of those variables are necessary because of the variance introduced by these processor behaviors. On the other hand, the *const* variables appear in the decision trees much less often, so those variables do not require as much refinement and are easier for the trees to model accurately. The gamma and, especially, exponential task distributions (T) appear frequently in the learned trees. Compared to the constant and Gaussian distributions, these distributions induce a high variance in the work for each task. Thus, more fine-grained inspection of these task distributions is required to insure accurate trees.

B. *Flexibility Predictions of DLS Algorithms*

We next consider how well the empirical tolerance prediction models learned with Auto-WEKA are able to predict the tolerance for each instance and provide algorithm selection. The results shown in Fig. 4 indicate that for almost all of the instances and algorithms, the predicted tolerance τ_{pred} and observed tolerance τ_{sim} are close. Each point in the scatter plot corresponds to one of the predictions made on line 5 in Algorithm 4.

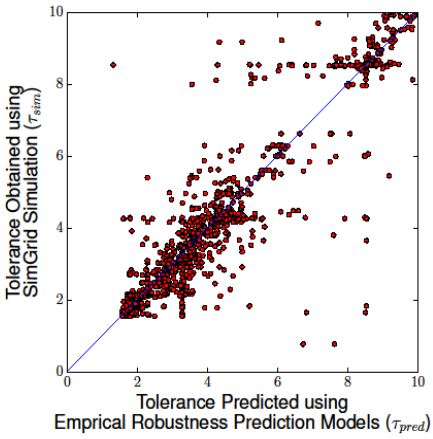

Fig. 4 Prediction performance of the empirical robustness prediction models. Each point in the plot represents one combination of a 4-tuple (N, P, T, A) instance and an algorithm s∈s. There are a total of 1,152 points. Points closer to the main diagonal represent more accurate predictions.

Furthermore, many of the incorrect predictions are higher than observed tolerances (beneath the diagonal). Of course, all of the predictions should be as accurate as possible, but these over-predictions have less impact on the final portfolio behavior because they will only discourage selection of a "good" (more robust) algorithm. In contrast under-predictions (above the diagonal) will encourage selection of a "bad" (less robust) algorithm.

C. Algorithm Selections and Portfolio Behavior

1) Algorithm Selections:

Using the tolerance predictions τ_{pred}, an algorithm was selected from the portfolio for each instance using Algorithm 4. Fig. 5 (left) shows that, other than STATIC (not selected at all), GSS and FAC (both selected only rarely), all of the algorithms were selected by the portfolio for over 10% of the 144 instances. Fig. 5 (right) shows a similar story for the virtual best solver (VBS), which acts as an oracle and always selects the algorithm with the best observed tolerance τ_{sim}. The primary difference between the VBS and the prediction-based selections is that the VBS selects WF quite a bit more frequently, and the AF-based

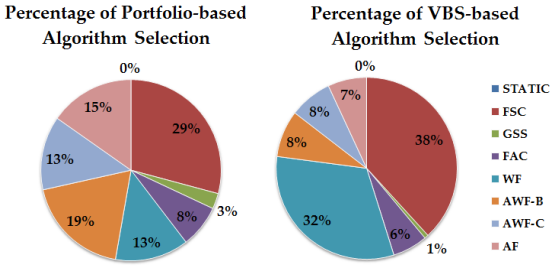

Fig. 5 Algorithm selections from the portfolio using the empirical tolerance prediction models (left) and the virtual best solver (right). Algorithms are selected using the empirical tolerance prediction models by predicting τ_{pred} for all algorithms on an instance and selecting the one with the lowest predicted value. Algorithms are selected using the virtual best solver by selecting the algorithm with the lowest τ_{sim} for each instance. The percentage is based on selections on 144 instances, where each instance represents the 4-tuple (N, P, T, A).

algorithms less frequently, than the prediction-based selections. Nevertheless, these results show that no single algorithm is best "most" of the time. Consequently, good instance-based algorithm selections from the portfolio can offer improvements over a winner-take-all approach which simply uses the same algorithm for all instances.

2) Comparison of VBS and Prediction-based Algorithm Selections without τ_{user}:

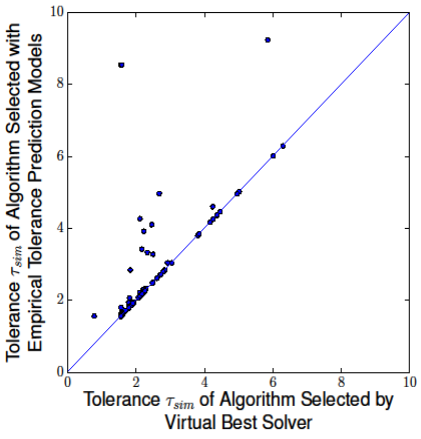

Fig. 6 Comparison of the tolerance τ_{sim} of the algorithms selected with the empirical tolerance prediction models and the virtual best solver. Each point in the plot represents one instance. There are a total of 144 instances. Points close to the main diagonal represent prediction-based selections that are similar to the best selection made by the virtual best solver.

In order to verify the quality of the selections based on the empirical tolerance prediction models, we compared the observed tolerance τ_{sim} of the prediction-based and VBS algorithm selections in Fig. 6. The maximum observed tolerance value τ_{user} for any algorithm selected by the portfolio was 9.23. As the figure shows, the algorithms selected based on the predictions almost always achieve similar tolerance compared to the best algorithm selections made by the VBS. Out of the 144 instances, the τ_{sim} of the prediction-based algorithm is more than $(1 + \tau_{sim})$ of the VBS algorithm only 10 times. On 4 out of those 10 times, including all three of the times in which the difference was greater than 3.5, GSS was predicted to be the best algorithm; furthermore, GSS was not predicted to be the best on any other instances, and the VBS only selected it one time. That means, GSS does not significantly improve the VBS behavior, and occasionally significantly degrades the prediction-based behavior. Consequently, we recommend, for time being, removing GSS from the portfolio to prevent its erroneous selection.

3) Algorithm Selections Incorporating τ_{user}:

Previous work on algorithm selection has always considered the problem of selecting the best algorithm for a given instance. In the case of certain hard-to-meet requirements (e.g., a τ_{user} very close to 1), though, no algorithm may perform well. In these cases, the user may prefer to consider other options even though they do not meet a desired robustness requirement.

TABLE II IDENTIFICATION OF INSTANCES FOR WHICH NO ALGORITHM WILL SATISFY THE USER-SPECIFIED TOLERANCE

τ_{user}	*tp*	*fp*	*fn*	*tn*	Sensitivity	specificity
1	0	0	1	143	0	1
2	70	9	0	65	1	0.88
3	123	2	3	16	0.98	0.89
4	128	1	1	14	0.99	0.93
5	139	2	1	2	0.99	0.5
6	141	2	1	0	0.99	0
7	144	0	0	0	1	0
8	144	0	0	0	1	0
9	144	0	0	0	1	0
10	144	0	0	0	1	0

We address this problem setting by considering a user-specified allowable tolerance value τ_{user}. We use τ^*_{sim} and τ^*_{pred} to denote the best observed τ_{sim} and best predicted τ_{pred}, respectively, for a given instance. Given τ_{user}, a certain algorithm $s \in \mathbf{s}$ is *robust* for a given instance c if and only if $\tau^*_{sim} < \tau_{user}$ or *non-robust* otherwise. Similarly, a certain algorithm $s \in \mathbf{s}$ is *predicted robust* for a given instance c if and only if $\tau^*_{pred} < \tau_{user}$ or *predicted non-robust* otherwise. Then, for a given threshold, each instance-DLS pair i.e. (c, s) can be labeled with the standard information retrieval terms (true positive (*tp*), false positive (*fp*), true negative (*tn*), false negative (*fn*)) based on its predicted and actual robustness. Those labels allow us to calculate the sensitivity (tp / (tp + fn)) and the specificity (tn / (tn + fp)).Both sensitivity and specificity range from 0 to 1.Intuitively, the sensitivity measures how well we identify instances for which some algorithm is robust. The specificity reflects the degree of reliability that some algorithm is robust for an instance, given that the algorithm has been already predicted to be robust.

Table III shows how the sensitivity and specificity vary as we change τ_{user}. The first column gives τ_{user}. The second column gives the true positives for that tolerance (instances for which $\tau^*_{sim} < \tau_{user}$ and $\tau^*_{pred} < \tau_{user}$). The third column gives the false positives ($\tau^*_{sim} \geqslant \tau_{user}$ but $\tau^*_{pred} < \tau_{user}$). The fourth column gives false negatives ($\tau^*_{sim} < \tau_{user}$ but $\tau^*_{pred} \geqslant \tau_{user}$). The fifth column

gives true negatives ($\tau^*_{sim} \geqslant \tau_{user}$and $\tau^*_{pred} \geqslant \tau_{user}$). The sixth and seventh columns give the sensitivity and specificity, respectively. We use the range 1 to 10 for τ_{user}. For all values of τ_{user}, the sensitivity is very close to 1; this means that if some algorithm is robust, then we accurately identify it with our predictions. The specificity is also close to 1 for most values of τ_{user}, which means that we rarely select an algorithm when none of them is robust. Occasionally, though, the specificity is low; these cases occur when very few instances cannot be solved robustly and even a small number of false positives yields a low specificity.

D. Comparison of Empirical Robustness Prediction Models to Winner-Take-All Behavior

As mentioned earlier in Section I, winner-take-all is the most common approach to algorithm selection. In this approach, each algorithm in a portfolio is executed on the training set, and the algorithm with the best performance (according to some metric, such as average robustness) is used for all instances in the testing set.

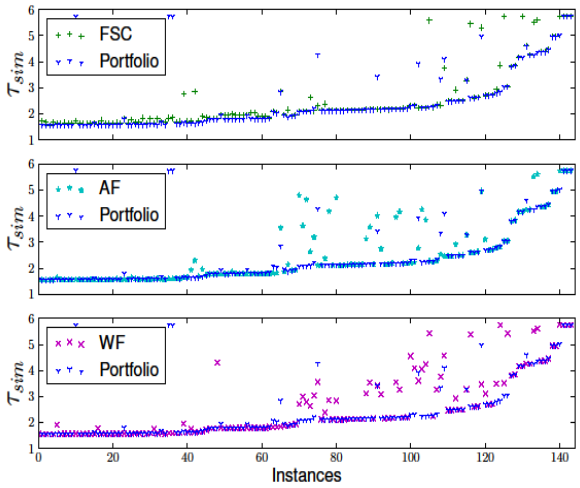

Fig. 7 Comparison of the observed tolerance τ_{sim} of the algorithms selected with the empirical tolerance prediction models and several DLS algorithms. The τ_{sim} of the selected algorithm is given as a blue 'Y'. The other glyph (i.e., +, *, and x) represent the observed τ_{sim} of the respective DLS algorithm. Instances for which τ_{sim}> 6 were rounded down to 6 for clarity of presentation. Each column represents one instance. There are a total of 144 instances. Smaller tolerance values are better.

In order to assess the behavior of our instance-based algorithm selection using the empirical robustness prediction models to a winner-take-all strategy, we show in Fig. 7 the robustness of the algorithm we select from the portfolio and the robustness of selected DLS algorithms (the behavior of AWF-B, AWF-C and FAC is similar to AF). The figure shows that FSC is clearly less robust on many of the instances for which AF and WF perform well (approximately instances 0 to 60).The empirical robustness prediction models learn this pattern. Consequently, one of the more robust algorithms is selected for use on those instances. On the other hand, FSC outperforms the other algorithms on instances for which the VBS has a high robustness. Our models also learn this trend and result in selecting FSC for most of those instances. Furthermore, in some cases, other DLS algorithms, such as AWF-B, outperform all of the visualized algorithms. Our models select these algorithms, as well, such as on several instances from 130 to 140.Taken together, these observations show that our instance-based algorithm selection strategy outperforms any winner-take-all approach.

E. Generalizing the Algorithm Selection to System Sizes

As a final set of experiments, we tested the generalization of the learning procedure to system sizes completely unseen during training. Specifically, rather than using stratified cross-validation to split the instances into training, validation and testing, we instead used all instances for which $P = \{4096, 8092\}$ as the training set and all instances for which $P = \{2048\}$ as the test set. The validation was performed using the training set. Thus, during testing, the learned models were presented with a system size never seen during training. In contrast, in the previous experiments, all system sizes were seen during training. (Note that all system, application and algorithm configurations were *never* seen during training of any models in this work.)

Fig. 8 shows that the τ_{pred} values were not as accurate compared to the previous experiments. This reflects a typical supervised learning shortcoming: the learned model typically does not generalize well to data completely different from that seen in training. Nevertheless, Fig. 9 shows that the algorithm selection using the models was still comparable to that of the VBS. So, despite the poor individual predictions, the models are still useful in distinguishing between robust and non-robust DLS algorithms.

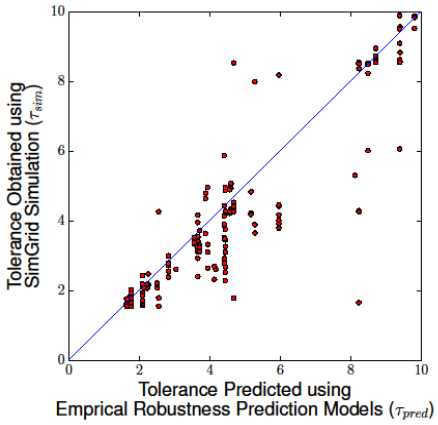

Fig. 8Prediction performance of the empirical robustness prediction models on the experiment in which instances where P=2048 were not used during training. Each point in the plot represents one combination of a 4-tuple (N, P=2048, T, A) instance and an algorithm s∈S. There are a total of 384 points. Points closer to the main diagonal represent more accurate predictions.

We carried out two similar experiments in which we used P = {4096} and P ={8092} instances as the testing sets. The results obtained showed that the robustness predictions were often inaccurate, thus not allowing robust algorithm selection. Employing empirical robustness prediction models provides a capability to make correct flexibility predictions of scheduling algorithms (high real time prediction speed (~ 1 second)). However, a limitation of the proposed method is that it may not assist in selecting robust DLS algorithms for system sizes much different from those seen in training.

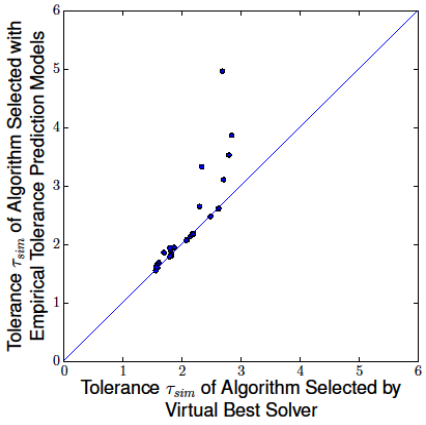

Fig. 9Comparison of the tolerance τ_{sim} of the algorithms selected with the empirical tolerance prediction models and the virtual best solver on the experiment in which instances where P=2048 were not used during training. Each point in the plot represents one instance. There are a total of 48 instances. Points close to the main diagonal represent prediction-based selections that are similar to the best selection made by the virtual best solver.

V. RELATED WORK

The algorithm selection problem was identified as the problem of selecting the most appropriate algorithm among the many available algorithms, such that it can solve a specific problem while optimizing some performance objective[10]. In contrast to the winner-take-all approach[11] which is the most widely adopted solution to the selection problem, Huberman[23] introduced a general procedure for computational portfolio design. Closest to our work, the authors in[24] advocate a widely-adopted solution to such algorithm selection problems by measuring the runtime of every candidate solver on a representative set of instances for solving the propositional satisfiability (SAT) problem using an approximate runtime predictor. The authors in[25] employ supervised learning techniques to predict the performance (execution time) of an application for different mappings aimed at minimizing communication.

Another research group proposed a combination of scheduling and solver selection to significantly boost performance of algorithm selection using a k-nearest neighbor based algorithm selection strategy[26]. The authors in[27] investigated the long-term execution of parallel scientific workloads on Infrastructure as a Service (IaaS) cloud resources via time-constrained portfolio scheduling. Historical simulations to adopt the scheduling policy and adaptive provisioning policies for changing

workloads, and future predictions have been proposed in a number of recent publications[28-31].

In contrast to these related studies, to the best of our knowledge, the present work is the first to apply a portfolio-based approach for predicting the robustness *and* selecting the most robust dynamic loop scheduling algorithm. Significantly differing from the body of previous work, we use state-of-the-art machine learning techniques to build empirical robustness prediction models for predicting the DLS robustness; and for a user specified tolerance value, we use that model to select the most robust DLS algorithm (if one exists), for various known or unknown sets of instances.

VI. CONCLUSIONS

In this article, a methodology was proposed for selecting the most robust DLS algorithm for enabling portfolio scheduling of scientific applications with large, computationally intensive, and data parallel loops on heterogeneous computing systems with unpredictably fluctuating load. A wide range of machine learning techniques were investigated to obtain an empirical hardness model which predicts the robustness of DLS algorithms with better accuracy than previous models. The prediction model enables selection of the most robust DLS algorithm for a given problem instance (application size, system size, probability distribution of underlying algorithmic variance and probability distribution of underlying systemic variance). Our results demonstrate that, typically the robustness predictions were quite accurate. We obtain such accurate results because the training, validation and testing sets represent similar populations due to stratified cross-validation. A shortcoming of our approach, common to supervised machine learning, is that the learned models do not generalize well to completely new populations. Concretely, we constructed a training set using all instances with $P \in \{2048, 4096\}$ and used all instances with $P = 8092$ as the testing set. Preliminary results suggested that the robustness predictions were often not very accurate. Nevertheless, the overall portfolio behavior remained competitive with other strategies because the predictions were still able to often separate robust algorithms from non-robust ones. By comparing the statistical results, we have shown evidence that the proposed portfolio-based approach enables selection of the most robust DLS algorithm for a wide range of application types and computing environments.

Our work in the near future will emphasize the assessment and multi-objective optimization of a utility function for achieving the desired performance, at a holistic level, for scientific applications executing on heterogeneous computing environments. In addition to parallel execution time and robustness, the utility function may contain other additional metrics, such as cost and power consumption. These additional metrics can also be introduced into the models of learning to derive the functional relation between these attributes for achieving a desired performance at a holistic level.

ACKNOWLEDGMENT

This work is in part supported by the National Science Foundation under grant number NSF IIP-1034897, by the Center for Advanced Vehicular Systems (CAVS), by the German Research Foundation (DFG) in the Collaborative Research Center 912 ``Highly Adaptive Energy-Efficient Computing", and by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

REFERENCES

[1] Top 500 supercomputer sites. [Online]. Available: http://www.top500.org

[2] I. Banicescu and R. L. Cari˜no, “Addressing the stochastic nature of scientific computations via dynamic loop scheduling,” Transactions on Numerical Analysis, Special Issue on Combinatorial Scientific Computing, vol. 21, pp. 66–80, 2005.

[3] F. Allen, M. Burke, P. Charles, R. Cytron, and J. Ferrante, “An overview of the PTRAN analysis system for multiprocessing,” Journal of Parallel and Distributed Computing, vol. 5, no. 5, pp. 617–640, 1988. http://dx.doi.org/10.1016/0743-7315(88)90015-9

[4] N. R. Satish, “Compile Time Task and Resource Allocation of Concurrent Applications to Multiprocessor Systems,” Ph.D. dissertation, University of California at Berkeley, Berkeley, CA, USA, 2009.

[5] I. Banicescu, F. M. Ciorba, and R. L. Cari˜no, “Towards the robustness of dynamic loop scheduling on large-scale heterogeneous distributed systems,” in IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2009), pp. 129–132, 2009.

[6] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, “Measuring the robustness of a resource allocation,” IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 7, pp. 630 – 641, Jul. 2004. http://dx.doi.org/10.1109/TPDS.2004.24

[7] S. Srivastava, N. Sukhija, I. Banicescu, and F. M. Ciorba, “Analyzing the robustness of dynamic loop scheduling for heterogeneous computing systems,” in IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2012), pp. 156–163, June 2012. http://dx.doi.org/10.1109/ISPDC.2012.29

[8] 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 Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, May 2013.

[9] S. Srivastava, B. Malone, N. Sukhija, I. Banicescu, and F. M. Ciorba, “Predicting the flexibility of dynamic loop scheduling usingan artificial neural network,” in IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2013), 2013.

http://dx.doi.org/10.1109/ISPDC.2013.10

[10] J. R. Rice, "The algorithm selection problem," in Advances in Computers, 1976, pp. 65–118.

[11] J. Lazzaro, S. Ryckebusch, M. A. Mahowald, and C. A. Mead, "Winner-take-all networks of O(n) complexity," in Advances in Neural Information Processing Systems, 1989, pp. 703–711.

[12] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Empirical hardness models: Methodology and a case study on combinatorial auctions," Journal of the ACM (JACM), vol. 56, no. 4, p. 22, 2009.
http://dx.doi.org/10.1145/1538902.1538906

[13] R. L. Cari˜no and I. Banicescu, "Dynamic load balancing with adaptive factoring methods in scientific applications," The Journal of Supercomputing, vol. 44, pp. 41–63, 2008.
http://dx.doi.org/10.1007/s11227-007-0148-y

[14] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A Generic Framework for Large-Scale Distributed Experiments," in 10th IEEE International Conference on Computer Modeling and Simulation, Mar. 2008.

[15] M. Balasubramaniam, N. Sukhija, F. M. Ciorba, I. Banicescu, and S. Srivastava, "Towards the scalability of dynamic loop scheduling techniques via discrete event simulation," 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum, vol. 0, pp. 1343–1351, 2012.

[16] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto- WEKA: Combined selection and hyperparameter optimization of classification algorithms," in 19th ACM International Conference on Knowledge Discovery and Data Mining, 2013, pp. 847–855.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10–18, Nov. 2009.
http://dx.doi.org/10.1145/1656274.1656278

[18] M. Schonlau, W. J. Welch, and D. R. Jones, "Global versus local search in constrained optimization of computer models," New Developments and Applications in Experimental Design, vol. 34, pp. 11–25, 1998.
http://dx.doi.org/10.1214/lnms/1215456182

[19] F. Hutter, H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," vol. 6683, pp. 507–523, 2011.

[20] R. Bertin, S. Hunold, A. Legrand, and C. Touati, "Fair scheduling of bag-of-tasks applications using distributed lagrangian optimization," Journal of Parallel and Distributed Computing, vol. 74, no. 1, pp. 1914 – 1929, 2014.
http://dx.doi.org/10.1016/j.jpdc.2013.08.011

[21] P. Velho, L. Schnorr, H. Casanova, and A. Legrand, "On the validity of flow-level TCP network models for grid and cloud simulations," ACM Transactions on Modeling and Computer Simulation, vol. 23, no. 3, Oct. 2013.

[22] P. Bedaride, A. Degomme, S. Genaud, A. Legrand, G. Markomanolis, M. Quinson, L. Stillwell, Mark, F. Suter, and B. Videau, "Toward Better Simulation of MPI Applications on Ethernet/TCP Networks," in 4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Nov. 2013.

[23] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," Science, vol. 27, pp. 51–53, 1997.
http://dx.doi.org/10.1126/science.275.5296.51

[24] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," Journal of Artificial Intelligence Research, vol. 32, pp. 565–606, 2008.

[25] N. Jain, A. Bhatele, M. P. Robson, T. Gamblin, and L. V. Kale, "Predicting application performance using supervised learning on communication features," in Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM, 2013, pp. 95:1–95:12.

[26] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in CP, 2011, pp. 454–469.

[27] K. Deng, J. Song, K. Ren, and A. Iosup, "Exploring portfolio scheduling for long-term execution of scientific workloads in IaaS clouds," in Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: ACM, 2013, pp. 55:1–55:12.

[28] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on, 2010, pp. 89–96.

[29] Y. Gao, H. Rong, and J. Z. Huang, "Adaptive grid job scheduling with genetic algorithms," Future Gener. Comput. Syst., vol. 21, no. 1, pp. 151–161, Jan. 2005.
http://dx.doi.org/10.1016/j.future.2004.09.033

[30] R. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in ICPP, 2011, pp. 295–304.

[31] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, ser. ScienceCloud '12. New York, NY, USA: ACM, 2012, pp. 31–40.

Nitin Sukhija received his BS degree (with honors) in Computer Science Engineering from Institute of Technology and Management, India (2002), Post Graduate Diploma in Financial Management from Symbiosis, India (2005), MBA degree majoring in Information Systems from

San Diego State University (2009), and MS degree in Computer Science majoring in Computing from National University, San Diego (2010).

Nitin has worked in industry and academia for over 7 years. He is currently pursuing his PhD in Computer Science from Mississippi State University and is working for Center for Advanced Vehicular Systems, Mississippi State University. His current research interests include autonomic computing, grid computing, dynamic load balancing, performance optimization, prediction and modeling, resilience analysis, self-healing systems, cyberinfrastructure and virtual organizations and high performance computing for scientific and engineering applications. He has authored and co-authored a number of articles published in IEEE and ACM conferences and journals.

He is a member of the IEEE computer society and ACM and also a member of the American Society of Engineering Education (ASEE)and the honor society of Upsilon Pi Epsilon (UPE). He has been a member of the Extreme Science and Engineering Discovery Environment (XSEDE-NSF-TeraGrid) Campus Champions program since 2013 and a member of the XSEDE Campus Champions Leadership Team since 2014.

Brandon M. Malone was born in Murfreesboro, TN in 1983. He graduated with a Ph.D. in computer science and engineering from Mississippi State University in 2012. He also has a Master's degree (2008) and a Bachelor's degree (2006) in computer science from Tennessee Technological University, Cookeville, TN.

He is currently a postdoctoral researcher in the Finnish Centre of Excellence in Computational Inference Research at the University of Helsinki, Finland. Previously, he was a graduate research assistance at Mississippi State University. He has published many articles on machine learning, including a recent publication entitled "Learning Optimal Bayesian Networks: A Shortest Path Perspective" in the Journal of Artificial Intelligence Research.

Srishti Srivastava is a PhD student at the Department of Computer Science and Engineering at Mississippi State University since August 2010. Her research interests include dynamic load balancing, high performance computing, performance and reliability analysis, optimization, and prediction, and autonomic computing. Presently, she is also a graduate research assistant at the Center for Autonomic Computing at Mississippi State University, which is also one of the four National Science Foundation sites for autonomic computing. Srishti has authored and co-authored a number of articles published in renowned IEEE and ACM conferences.

She is the member of the IEEE computer society, ACM, Society for Industrial and Applied Mathematics (SIAM), Computing Research Association (CRA, CRA-W), Anita Borg Institute Grace Hopper Celebration (ABI-GHC), and an honor society of Upsilon Pi Epsilon (UPE).

IoanaBanicescu is a professor in the Department of Computer Science and Engineering at Mississippi State University (MSU), a Director of the Center for Autonomic Computing at MSU, and also a Co-Director of the National Science Foundation Center for Autonomic Computing. She received the Diploma in Engineering (Electronics and Telecommunications) from Polytechnic University - Bucharest, and the M.S. and the Ph.D. degrees in Computer Science from New York University - Polytechnic Institute.

Professor Banicescu's research interests include parallel algorithms, scientific computing, performance analysis, evaluation, and prediction. Currently, her research focus is on autonomic computing and performance optimization for problems in computational science. She is an Associate Editor of the Cluster Computing journal and the International Journal on Computational Science and Engineering, and she has given many invited talks at universities, government laboratories, and at various national and international forums in the United States and overseas. Professor Banicescu is the recipient of a number of awards and distinctions for her scholarly contributions and has authored and co-authored over 100 articles published in journals, books, and conference proceedings.

Florina M. Ciorbais born in Zalau, Romania. She holds a diploma in computer engineering from the University of Oradea, Romaniasince 2001, and a doctoral degree in computer engineering from National Technical University of Athens, Greece since 2008.

She was Postdoctoral Research Associate at Mississippi State University, USA between 2008-2010. Since 2010 she is a Postdoctoral Research Associate at TechnischeUniversität Dresden, Germany. She has authored/co-authored 7 journal articles and over 25 conference papers. Her research interests include high performance computing, performance optimization of scientific applications, andperformance modeling and analysis of parallel applications on adaptive and energy-efficient computing platforms.

Dr. Ciorba is a member of ACM (Association for Computing Machinery) and the IEEE (Institute of Electrical and Electronics Engineers).