Portfolio-based Selection of Robust Dynamic Loop Scheduling Algorithms Using Machine Learning

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

*Mississippi State University
Department of Computer Science and Engineering
Mississippi State, MS 39762, USA
{nitin@cavs., ioana@cse., ss878@}msstate.edu

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

‡Technische Universität Dresden
Center for Information Services and High Performance Computing
01062 Dresden, Germany
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 a challenge to select the most robust scheduling algorithms for guaranteeing optimized performance of scientific applications on large-scale computing systems that comprise resources which 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 workload conditions. Thus, in this paper 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 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 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 end of this decade, the development and use of large-scale scientific applications is becoming increasingly prevalent for solving a multitude of problems in various science and engineering fields, as well as society domains. 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 computational 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 is considered 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 environments [2]. However, to guarantee the performance of scientific applications 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 applications against various perturbations that may arise in the computing environments during the execution of the application. A DLS algorithm used to schedule computations of applications on a heterogeneous computing system is considered to be robust, if it can handle the largest variation in the existing perturbations that results in the lowest impact on its 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 this processor and therefore this processor is 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 affects 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 execution time 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 were mathematically formulated to study 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]. The *flexibility metric* 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 *resilience metric* 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. Furthermore, the flexibility metric and the developed robustness analysis have been successfully employed to evaluate the robustness of the DLS algorithms against system load fluctuations using discrete event simulations [7] [8]. Throughout the 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 from instance to instance, 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 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 which algorithm is most robust in accomplishing the best performance of a given scheduled application onto a 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 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 instance but also to *enable a dynamic selection* of the most robust algorithm 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, what 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]. Applying this approach, the robustness of each DLS algorithm can be measured on a representative set of instances, and the algorithm which has the best 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 a predictor 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 three-fold: (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 techniques and their robustness, together with a description of representative empirical models and related work are presented in Section II. The design and organization of the methodology to select the most robust DLS algorithm is described in Section III. The simulation analysis and the evaluation of the experimental results are discussed in Section IV. The conclusions and possible future directions are summarized in Section V.

II. Background and Related Work

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, $\mathcal{S}$ = {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 variances, 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 tolerance limit, defined as τ_{user}, where $\tau_{user} \geq 1$, 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 best expected parallel execution time of the application, T_{PAR}^{ideal} by a factor of τ_{user}. Mathematically, this is expressed by the following inequalities:

$$T_{PAR}^{ideal} \leq T_{PAR} \leq \tau_{user} \cdot T_{PAR}^{ideal} \tag{1}$$

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. Identification of Appropriate Empirical Models for Robustness Prediction

A priori, it is difficult to 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) which 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., back-propagation for learning the weights in neural networks). Auto-WEKA [16] addresses this problem by searching for model classes and hyperparameters which best explain a given dataset. In particular, if a sequential model-based optimization (SMBO) strategy, a training set and a validation set are provided, 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 Section III-B, while we

refer the reader to [16] and the references therein for more details.

D. 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 [18] introduced a general procedure for computational portfolio design. Closest to our work, the authors in [19] 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 [20] 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 [21]. The authors in [22] 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 [23] [24] [25] [26].

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.

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 Figure 1. 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.

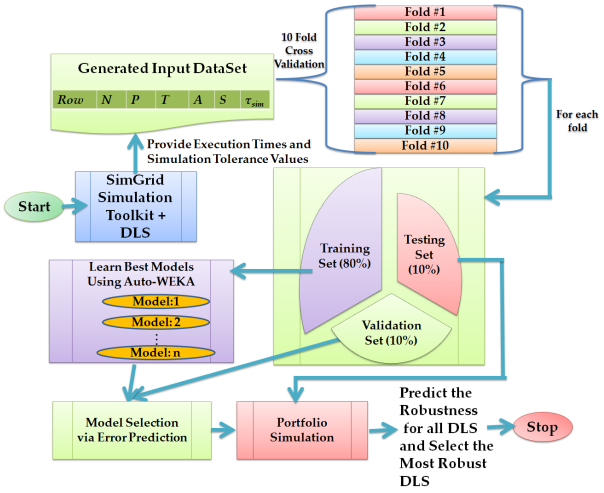

Figure 1. The proposed workflow for selecting the most robust DLS algorithm

A. Constructing the Input Dataset of Instances

The Algorithm 1 shows the pseudocode of the algorithm employed to construct the input dataset comprising the instances as well as other attributes.

Algorithm 1 Creating the training dataset

Input: problem size N, computing system P, algorithmic variances $\mathcal{T}$, systemic variances $\mathcal{A}$, DLS algorithm portfolio $\mathcal{S}$
Output: dataset $\mathcal{D}$

```
dataset D ← ∅
for each combination c of N, P, T and A do
    T_PAR^ideal ← ideal parallel execution time of c
    for each algorithm S ∈ S do
        T_PAR ← total parallel execution time of instance c using S
        τ_sim ← T_PAR / T_PAR^ideal
        D_{S,c} ← τ_sim
    end for
end for
return D
```

In this work we have used the SimGrid simulation framework to model, control, reproduce, and monitor large-scale computing systems to analyze 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 [27] [28] [29]. 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 informationon modeling both, the scientific applications and the heterogeneous, non-dedicated computing

systems. 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 correspondingly. The systemic variance in the computing system, denoted as $\mathcal{A}$, is characterized by varying the availabilities of the processors during the runtime of the application. The choice of the probability distributions representing the parameter 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:

- *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* $\mathcal{T}$ − modeled as constant with each task computation time of 2×10^8 flops, as Gaussian with mean $\mu = 9.5 \times 10^8$ flops and standard deviation $\sigma = 7 \times 10^7$ flops, as gamma with mean $\mu = 1.1 \times 10^8$ flops, or as exponential with mean $\mu = 3 \times 10^8$ flops, and
- *Probability distributions underlying* $\mathcal{A}$ − constant-availability where the availability of each processor follows a uniform distribution in the interval [0.1, ... 1.0] or is exponentially distributed with mean $\mu = 0.3$ but does not vary with time (remains unchanged during the entire course of the simulation), or variable availability where the availability of a processor drawn either from the uniform or exponential distributions with the above parameters periodically varies during the execution time of the application.

As shown in Algorithm 1 (line 3), the algorithm computes the T_{PAR}^{ideal} values for each scientific application comprising *N* independent tasks with **constant** $\mathcal{T}$ or **variable** $\mathcal{T}$ 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 which 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 $S \in \mathcal{S}$ (Algorithm 1, line 5) is obtained upon employing a certain DLS algorithm S from the DLS portfolio $\mathcal{S}$ to schedule a scientific application consisting of *N* independent tasks with **constant** $\mathcal{T}$ (fixed task computation costs) or with **variable** $\mathcal{T}$ (task computation costs following Gaussian, gamma, or exponential distributions) onto a non-dedicated heterogeneous system with *P* processors with **variable** $\mathcal{A}$ (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 ($\mathcal{T}$) and systemic variances ($\mathcal{A}$) on the performance of a particular $\mathcal{S}$.

The robustness of S defines its performance in the presence of both $\mathcal{T}$ and $\mathcal{A}$. 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*, $\mathcal{T}$, $\mathcal{A}$), and is computed (Algorithm 1, 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:

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

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

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

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

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

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	$\mathcal{T}$	$\mathcal{A}$	$\mathcal{S}$	τ_{sim}
1	4,194,304	8,092	Gaussian	exponential	AF	1.2
2	1,048,576	2,048	gamma	uniform	FSC	1.65
…	…	…	…	…	…	…
1,152	16,777,216	4,096	constant	exponential	AWF-B	2.9

B. Learning DLS Empirical Robustness Prediction Models

Auto-WEKA and Sequential Model-Based Optimization: The sequential model-based optimization algorithm (SMBO), the pseudocode of which is listed in Algorithm 2, 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 $\mathcal{D}$. The best hyperparameters found so far are referred to as $\mathcal{E}^*$. First, the SMBO algorithm constructs a model, $\mathcal{M}_f$ to estimate the dependence of f on $\mathcal{E}$ and $\mathcal{D}$ (Algorithm 2, line 1). Then, $\mathcal{M}_f$ is used to select a new set of hyperparameters $\mathcal{E}'$ (Algorithm 2, line 4), and f is evaluated on $\mathcal{E}'$ and $\mathcal{D}$ (Algorithm 2, line 5). Next, if $f(\mathcal{E}', \mathcal{D})$ is better than $f(\mathcal{E}^*, \mathcal{D})$, then $\mathcal{E}^*$ is updated to be $\mathcal{E}'$ (Algorithm 2, line 6). Finally, the new information, $(\mathcal{E}', f(\mathcal{E}', \mathcal{D}))$, is used to update $\mathcal{M}_f$ (Algorithm 2, line 8).

The next set of hyperparameters $\mathcal{E}'$ are selected using the *positive expected improvement* (PEI) [30] of $\mathcal{E}'$ over $f(\mathcal{E}^*, \mathcal{D})$. The PEI calculation uses $\mathcal{M}_f$ to estimate the probability distribution $p(f(\mathcal{E}', \mathcal{D}) < f(\mathcal{E}^*, \mathcal{D}))$. This estimation primarily requires $p(f(\mathcal{E}', \mathcal{D}) = c|\mathcal{E}')$. We configure Auto-WEKA to use sequential model-based algorithm configuration (SMAC) [31], the default configuration, to learn this probability distribution. SMAC assumes $p(f(\mathcal{E}', \mathcal{D}) = c|\mathcal{E}')$ is Gaussian-distributed. It represents $\mathcal{M}_f$ as a random forest and uses it to estimate the mean and variance of the Gaussian distribution. In addition to selecting models which optimize $p(f(\mathcal{E}', \mathcal{D}) = c|\mathcal{E}')$, SMAC also sometimes selects hyperparameters at random to better explore the search space.

Algorithm 2 Sequential model-based optimization(SMBO)

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

This randomness insures that good models can be found even if $\mathcal{M}_f$ does not model $p(f(\mathcal{E}', \mathcal{D}) = c|\mathcal{E}')$ very well.

Learning 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, $\mathcal{T}$, $\mathcal{A}$), and an algorithm $S \in \mathcal{S}$ 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. 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.

Algorithm 3 Learning empirical robustness prediction models

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

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 Figure 1) 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 $\mathcal{D}$,

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

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 described in Section II-C, 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, $\mathcal{V} = \{1, 5, 7, 10, 1001, 1005, 1007, 1010\}$) and allowed Auto-WEKA to run for 2 hours for each seed (Algorithm 3, line 5). This training phase results in 8 candidate models,

$\mathcal{E}_1, \ldots, \mathcal{E}_{|\mathcal{V}|}$. 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 6). We collect $\mathcal{E}^*$ from each fold to use in the algorithm selection (Algorithm 3, line 9), described in Section III-C.

C. Algorithm Selection during Online Simulation

Algorithm 4 shows the pseudocode of the algorithm employed for simulating the selection of robust DLS algorithms from the portfolio. The algorithm loops over each distinct instance c (Algorithm 4, lines 3 to 8). It then uses the empirical robustness prediction models learned in Algorithm 3 to predict the robustness of each algorithm S on instance c; the algorithm with the lowest predicted robustness for c is selected for use by the portfolio (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 $(S, c) \in \mathcal{D}^i_{testing}$. 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).

Algorithm 4 Selecting robust algorithms during online simulation

Input: empirical robustness prediction models $\mathcal{E}$, dataset $\mathcal{D}$, problem size N, system size P, algorithmic variances $\mathcal{T}$, systemic variances $\mathcal{A}$, DLS algorithm portfolio $\mathcal{S}$
Output: algorithm selections $\mathcal{S}^{portfolio}$ and $\mathcal{S}^{VBS}$
1: portfolio algorithm selections $\mathcal{S}^{portfolio} \leftarrow \varnothing$
2: VBS algorithm selections $\mathcal{S}^{VBS} \leftarrow \varnothing$
3: **for** each combination c = (N, P, $\mathcal{T}$ and $\mathcal{A}$) **do**
4: ▷ pick the algorithm with the lowest τ_{pred} for the portfolio
5: $\mathcal{S}^{portfolio}_c \leftarrow \arg\min_{S \in \mathcal{S}} \mathcal{E}(S, c)$
6: ▷ pick the algorithm with the lowest τ_{sim} for the VBS
7: $\mathcal{S}^{VBS}_c \leftarrow \arg\min_{S \in \mathcal{S}} \mathcal{D}_{S,c}$
8: **end for**
9: **return** $\mathcal{S}^{portfolio}$, $\mathcal{S}^{VBS}$

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 $\mathcal{S}$. In particular, it uses the complete dataset $\mathcal{D}$ 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).

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, Figure 2 shows that

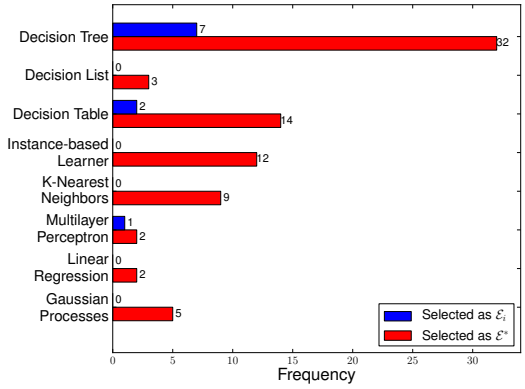

Figure 2. Frequency of model classes learned by Auto-WEKA and selected based on performance on the validation set. The best models learned by Auto-WEKA were aggregated by their model class, regardless of hyperparameters. Sometimes, Auto-WEKA used a "meta" model, such as boosting. These meta models extend one of the base models, such as a decision table. We grouped those according to the base model.

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.

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 Figure 3 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. 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.

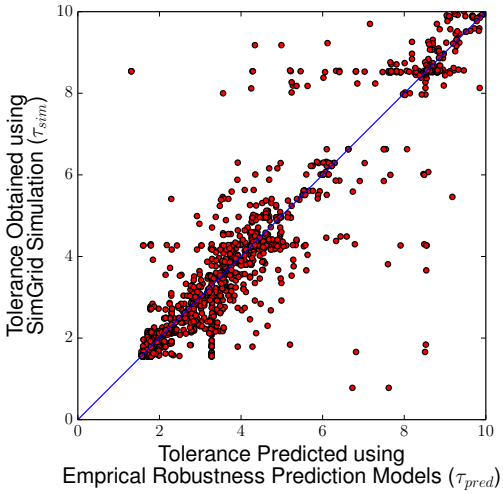

Figure 3. Prediction performance of the empirical robustness prediction models. Each point in the plot represents one combination of a 4-tuple (N, P, $\mathcal{T}$, $\mathcal{A}$) instance and an algorithm $S \in \mathcal{S}$. There are a total of 1,152 points. Points closer to the main diagonal represent more accurate predictions.

C. Algorithm Selections and Portfolio Behavior

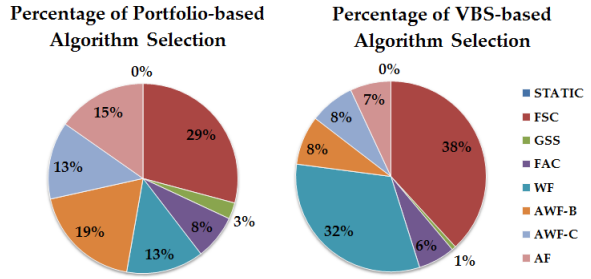

Figure 4. Algorithm selections from the portfolio using the empirical tolerance prediction models (left) and the virtual best solver (right). Algorithms are selected using the empirical tolerance prediction models by predicting τ_{pred} for all algorithms on an instance and selecting the one with the 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, $\mathcal{T}$, $\mathcal{A}$).

Algorithm Selections: Using the tolerance predictions τ_{pred}, an algorithm was selected from the portfolio for each instance using Algorithm 4. Figure 4 (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. Figure 4 (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 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.

Comparison of VBS and Prediction-based Algorithm Selections without τ_{user}: In order to verify the quality of the selections based on the emprical tolerance prediction models, we compared the observed tolerance τ_{sim} of the prediction-based and VBS algorithm selections in Figure 5. 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

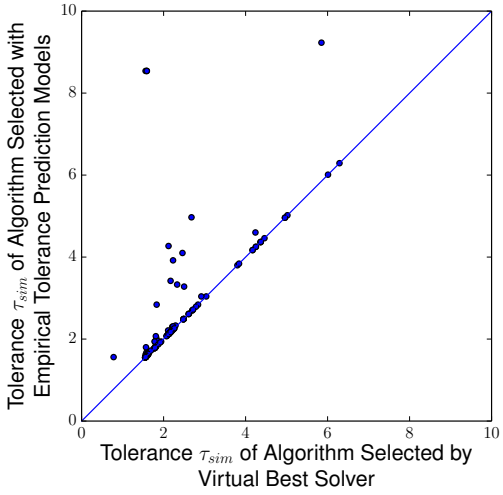

Figure 5. Comparison of the tolerance τ_{sim} of the algorithms selected with the emprical tolerance prediction models and the virtual best solver. Each point in the plot represents one instance. There are a total of 144 instances. Points close to the main diagonal represent prediction-based selections that are similar to the best selection made by the virtual best solver.

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.

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.

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 $S \in \mathcal{S}$ is *robust* for a given instance

c if and only if $\tau^*_{sim} < \tau_{user}$ or *non-robust* otherwise. Similarly, a certain algorithm $S \in \mathcal{S}$ 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.

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

Table I

IDENTIFICATION OF INSTANCES FOR WHICH NO ALGORITHM WILL SATISFY THE USER-SPECIFIED TOLERANCE 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 ($\tau^*_{sim} \geq \tau_{user}$ BUT $\tau^*_{pred} < \tau_{user}$). THE FOURTH COLUMN GIVES FALSE NEGATIVES ($\tau^*_{sim} < \tau_{user}$ BUT $\tau^*_{pred} \geq \tau_{user}$). THE FIFTH COLUMN GIVES TRUE NEGATIVES ($\tau^*_{sim} \geq \tau_{user}$ AND $\tau^*_{pred} \geq \tau_{user}$). THE SIXTH AND SEVENTH COLUMNS GIVE THE SENSITIVITY AND SPECIFICITY, RESPECTIVELY. WE USE THE RANGE 1 TO 10 FOR τ_{user}.

Table IV-C shows how the sensitivity and specificity vary as we change 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 II-D, 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. 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 Figure 6 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.

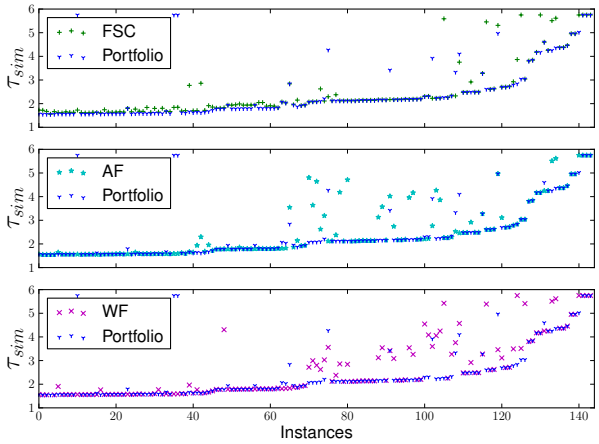

Figure 6. Comparison of the observed tolerance τ_{sim} of the algorithms selected with the emprical tolerance prediction models and several DLS algorithms. The τ_{sim} of the selected algorithm is given as a blue 'Y'. The other glyph (i.e., $+$, $\star$, and $\times$) represent the observed τ_{sim} of the respective DLS algorithm. Instances for which $\tau_{sim} > 6$ 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.

V. CONCLUSIONS AND FUTURE WORK

A portfolio-based approach for selecting the most robust DLS algorithm was proposed in this paper. 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 the robustness predictions were typically 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 (not shown here due to space limitations) suggested that the robustness predictions were often not very accurate. Nevertheless, the overall portfolio behavior remained competitive with other strategies because the predictions still were 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 of the utility function for achieving the desired level of performance of scientific applications executing on heterogeneous computing environments.

Acknowledgments: This work is in part supported by the National Science Foundation under grant number NSF IIP-1034897, 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. Cariño, "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.

[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. Cariño, "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.

[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.

[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 using an artificial neural network," in *IEEE International Symposium on Parallel and Distributed Computing (ISPDC 2013)*, 2013.

[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.

[13] R. L. Cariño and I. Banicescu, "Dynamic load balancing with adaptive factoring methods in scientific applications," *The Journal of Supercomputing*, vol. 44, pp. 41–63, 2008.

[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.

[18] B. A. Huberman, R. M. Lukose, and T. Hogg, "An economics approach to hard computational problems," *Science*, vol. 27, pp. 51–53, 1997.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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