An Autonomic Approach for the Selection of Robust Dynamic Loop Scheduling Techniques

Anthony Boulmier*
University of Geneva
Department of Computer Science
1227 Carouge, Switzerland
anthony.boulmier@unige.ch

Ioana Banicescu†
Mississippi State University
Department of Computer Science and Engineering
Mississippi State, MS 39762, USA
ioana@cse.msstate.edu

Florina M. Ciorba‡
University of Basel
Department of Mathematics and Computer Science
4051 Basel, Switzerland
florina.ciorba@unibas.ch

Nabil Abdennadher§
University of Applied Sciences Western Switzerland
Department of Computer Science and Engineering
1202 Geneva, Switzerland
nabil.abdennadher@hesge.ch

Abstract— **Parallel applications are highly irregular and high performance computing (HPC) infrastructures are very complex. The HPC applications of interest herein are time-stepping scientific applications (TSSA). Often, TSSA involve the repeated execution of multiple parallel loops with thousands of iterations and irregular behavior. Dynamic loop scheduling (DLS) techniques were developed over time and have proven to be effective in scheduling parallel loops for achieving load balancing of TSSA. Using a single particular DLS technique throughout the entire execution of a time-step, or even over the entire application, does not guarantee optimal performance due to the unpredictable variations in problem and algorithmic characteristics as well as those of the infrastructure capabilities. For that reason, an autonomic selection of DLS techniques as function of the parallel loop execution time has shown to improve application performance. Recently, a robustness metric of DLS techniques, named "flexibility", has been proposed to estimate the capability of a DLS technique to resist to variations in the loop iterations execution time. To improve the performance of TSSA, we propose in this work an approach that involves the autonomic selection of DLS techniques as function of the flexibility of DLS techniques. The first major novelty of our approach lies in the use of state-of-the-art reinforcement learning (RL) algorithms as smart agents. The second novelty lies in the design of a modified flexibility metric. The third major novelty resides in using the new modified flexibility metric as a reward for the smart agents. The fourth novelty is the evaluation of the proposed approach within a simulated environment, in particular using the SimGrid-SMPI interface to execute DLS algorithms. We discuss the advantages and the limitations of the new proposed flexibility metric as a reward.**

Keywords- High Performance Computing, Autonomic Computing, Reinforcement Learning, Performance Optimization, Robustness, Dynamic Loop Scheduling.

I. INTRODUCTION

Scientific applications cover various computational domains such as quantum physics, cosmology, molecular dynamics, landscape engineering, and others. These applications are often time and computationally intensive, complex and irregular. Scientific applications that require a high number of time-steps to converge toward a solution are considered *time-stepping scientific applications* (TSSA). These applications often involve one or several parallel loops that have a high number of loop iterations with irregular execution time. Parallel loops are naturally suitable candidates for parallelization. To achieve high performance, these loops are executed on tightly coupled parallel infrastructure. During the execution of the TSSA, the parallel loops iterations execution time is determined by the combined effect of the iteration workload and the infrastructure availability to compute. A straightforward way to distribute the workload is to simply allocate an equal number of loop iterations to each processing element (PE). Unfortunately, this method provides almost always poor performance.

The semantics of TSSA and their algorithmic structure often lead to an irregular execution behavior. Moreover, the high performance computing infrastructures used to execute these applications often suffer from unpredictable characteristics such as non-uniform memory access time, interrupts, cache misses, and others. TSSA irregularities combined with the unpredictable behavior of the infrastructure ultimately lead to load imbalance, and thus to severe performance degradation.

Over the years, various scheduling strategies have been developed to address the load imbalance problem. Those strategies use loop scheduling techniques to perform load balancing since parallel loops encapsulate the major part of the application workload. Dynamic loop scheduling (DLS) algorithms have been shown to be the most efficient for providing load balancing in environments with variable characteristics. For instance, Fixed Size Chunking (FSC), Guided Self Scheduling (GSS), Factoring (FAC), Weighted Factoring (WF), Adaptive Weighted Factoring (AWF) and its variants (AWF-B, C, D, E) and Adaptive Factoring (AF) [1]–[7] are algorithms used for DLS. All of these techniques use probabilistic analyses making them naturally capable of addressing the unpredictable variations arising from the algorithmic, problem and infrastructure characteristics.

Selecting the most appropriate DLS technique for a given application is not a trivial task. A DLS technique selected offline may perform well at the beginning of the application's lifetime and poorly later, due to the combined effect of application and infrastructure irregularities. Employing a single DLS technique throughout the execution of the entire TSSA, or even a single time-step does not guarantee optimal performance due to the unpredictable variations in algorithmic and problem characteristics together with those of infrastructure capabilities. To overcome this challenge, an *autonomic computing* (AC) approach was proposed for the autonomic selection of the dynamic scheduling technique during the execution of a TSSA as function of the parallel loop execution time [8], [9]. Therein, the goal was to integrate a *reinforcement learning* (RL) agent for all the parallel loops within a time-step, or one agent for each parallel loop of a time-step. Each RL agent selects a single DLS technique within a time-step based on its reward (discussed in detail in Section II). RL is an active area of research in machine learning used to solve sequential decision making problems via learning, planning and decision making [10]. Such an approach presents the advantage of being generic and easily integrable into the existing code.

Recently, various performance metrics have, however, been proposed to accurately measure the quantity of absorbed perturbation a DLS technique can handle. A metric such as the robustness [11] (both the flexibility and the resilience [12]) has shown useful properties, such as the ability to guarantee a certain level of performance. The resilience metric allows the estimation of the ability of a DLS technique to withstand PE failures. The flexibility metric allows to estimate the quantity of perturbations in the loop iterations execution time a DLS technique can absorb. Such metrics can greatly improve the quality of the DLS selection and, thus, increase the application performance.

We propose a new AC approach for selecting the most robust DLS technique, at various time-steps of the execution of a TSSA on a tightly coupled parallel infrastructure. The major contributions of the work presented in this paper are as follows: (i) The design of a modified flexibility metric, (ii) The use of the modified flexibility metric as a RL reward and a study of its effectiveness, (iii) The successful implementation and the use of three state-of-the-art RL algorithms as smart-agent for the selection of robust DLS techniques, and (iv) The evaluation of the proposed approach using a generic simulation framework (i.e., SimGrid [13]) and use of the SimGrid-SMPI interface to execute the DLS techniques. To the best of our knowledge, this is the first effort to use SimGrid-SMPI to simulate the execution of DLS techniques. The present research provides valuable information extracted from forty-five sets of experiments on TSSA optimization using five RL techniques. It also highlights the advantages and limitations of the flexibility metric as a RL reward.

The rest of the paper is organized as follows. A review on the previous work on TSSA self-optimization via the selection of efficient DLS techniques together with a description of these techniques, their flexibility, and a description of reinforcement learning techniques are presented in Section II. The adaptation

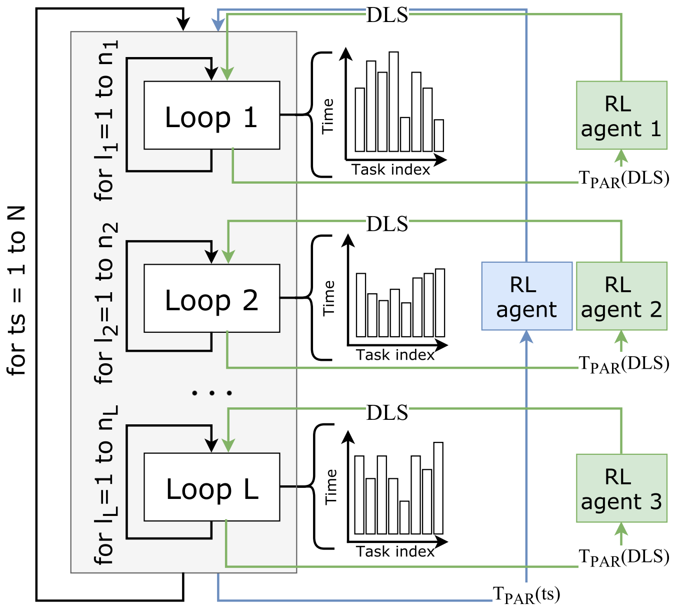

Figure 1: Autonomic selection of DLS techniques via RL in TSSA. T_{PAR} is the parallel execution time, DLS denotes the dynamic loop scheduling technique selected by the RL agent, and $r^{\text{RL}}_{\text{DLS}}$ is the robustness reward of the selected DLS technique. The blue RL agent represents the approach introduced in [8], whereas the green RL agents show the approach proposed in [9].

of the flexibility metric as reward, as well as the design of the approach proposed in this paper, are described in Section III. The evaluation of the effectiveness of the proposed approach within a generic simulation framework is presented in Section IV. The conclusions and future work are summarized in Section V.

II. Background and Related Work

Recently, efforts have been made to integrate autonomic computing with load balancing strategies. The idea is to leverage the ability of the reinforcement learning (RL) agents to learn optimal load balancing policies. This area of research has been explored in the literature. The goal of those works was to minimize the computation time of time-stepping scientific applications (TSSA).

TSSA is a class of scientific applications that perform a high number of iterations (time-steps) to converge to the desired solution. Inside the time-steps, several parallel loops (of various nesting levels) are disposed in sequential order. Parallel loops are a main source of parallelism in scientific applications. These loops involve a large number of loop iterations which can be considered independent tasks with variable execution time. N-Body simulations, wave-packet simulations, and heat solvers are well known examples of TSSA.

Previous work has proposed an autonomic computing approach for selecting, at various time-steps of a TSSA, the most appropriate DLS technique as function of a given criterion. In [8], a single RL agent uses the entire time-step performance to decide which DLS technique to use for every parallel loop. As pointed out in previous work, using a single DLS technique throughout an entire time-step is likely to provide low performance [8], [9]. Accordingly, in [9] a similar approach has been proposed that uses one RL agent per each parallel loop. Therein, the RL agents use the parallel loop execution time to decide which DLS technique to use for their assigned parallel loop. The RL algorithms which have been targeted for this approach are Q-Learning and SARSA. These algorithms will be introduced later in Section II-C as they are also considered in present work. Figure 1 illustrates the aforementioned approaches in red and green color.

The results provided in [9] show that the autonomic selection of DLS techniques is promising. However, the work had a limitation in generalizing the usefulness of the approach on a wide range of situations due to its experimental setup.

The present work proposes an autonomic approach for the selection of robust dynamic loop scheduling techniques. It aims at selecting using RL agents, for each parallel loop and at various time-step of a TSSA, the optimal DLS technique to minimize the application completion time, $T_{PAR}(\mathrm{TSSA})$, while maximizing the degree of absorbed perturbation arising from unpredictable problem, algorithmic and infrastructure characteristics. To increase the generic aspect of this approach, a simulation framework is considered and employed. Several state-of-the-art reinforcement learning algorithms have been tested as selection agents.

Table I: Glossary of notation.

Notation	Description
$\lambda = \{\lambda_1, ..., \lambda_\mathrm{P}\}^t$	PE's availability to compute $\in [0, 100]$ at time t
λ^{orig}	Availability to compute at which the infrastructure is assumed to operate, $\leq 100\%$
θ	Loop performance threshold, $\tau \cdot T_{\mathrm{PAR}}^{\mathrm{FASTEST}}(\lambda^{\mathrm{orig}})$
τ	Tolerance factor, ≥ 1
r_{DLS}	Flexibility radius of a DLS technique
$r_{\mathrm{DLS}}^{\mathrm{RL}}$	Modified flexibility radius of a DLS technique
$T_{\mathrm{PAR}}(\mathrm{DLS}, \lambda)$	Parallel loop execution time of a given DLS and a given perturbation
$T_{\mathrm{PAR}}^{\mathrm{IDEAL}}(\mathrm{DLS}, \lambda^{\mathrm{orig}})$	*Ideal* parallel loop execution time of a given DLS
$T_{\mathrm{PAR}}^{\mathrm{FASTEST}}(\lambda^{\mathrm{orig}})$	$\min\limits_{\mathrm{DLS}} T_{\mathrm{PAR}}^{\mathrm{IDEAL}}(\mathrm{DLS}, \lambda^{\mathrm{orig}}),\ \forall \mathrm{DLS} \in A$
$T_{\mathrm{PAR}}(\mathrm{ts})$	$\sum\limits_{\mathrm{l}=1}^{\mathrm{L}} T_{\mathrm{PAR}}(\mathrm{DLS_l}, \lambda^{\mathrm{ts}}),\ \mathrm{DLS_l} \in A$
$T_{\mathrm{PAR}}(\mathrm{TSSA})$	$\sum\limits_{\mathrm{ts}=1}^{\mathrm{N}} T_{\mathrm{PAR}}(\mathrm{ts})$
S	Set of states, $s \in$ {running, stopped}
A	Set of actions, $a \in$ {STATIC, FSC, GSS, FAC, AWF, AWF-B, AWF-C, AWF-D, AWF-E, AF}
$\mathcal{R}(s, a)$	Reward function
$\mathcal{T}(s, a)$	State-transition function
$Q(s, a)$	Action-value function
$Q^*(s, a)$	Optimal action-value function
$V(s)$	State-value function
$\pi(s)$	Control policy
α	Learning rate
γ	Discount factor

A. DYNAMIC LOOP SCHEDULING

Load imbalance in scientific applications has always been a critical performance issue in high performance computing. Over the years, this issue has been addressed with various load balancing approaches. Among them, loop scheduling algorithms have shown to be very efficient [14].

Dynamic loop scheduling (DLS) techniques are load balancing algorithms that employ a master-worker execution model and are designed to address load imbalance for parallel loops execution. They aim at dividing the parallel loop iterations into chunks of variable size. These chunks are then scheduled and executed on worker processors in order to minimize the application makespan. Popular DLS algorithms can be identified as: FSC, GSS, FAC, WF, AWF and its variants AWF-B, AWF-C, AWF-D, AWF-E, and AF [1]–[7].

DLS techniques are based on probabilistic analyses. The chunks sizes depend on the number of loop iterations n, the number of processors p, the processor pace, and the probability of a given chunk to be completed within the optimal time [9], [14]. As such, DLS techniques address all sources of load imbalance arising from unpredictable problem, algorithmic and infrastructure characteristics.

Over the years, many DLS techniques have been studied and well described in the literature [14], [15]. Among them, two categories can be considered: non-adaptive and adaptive. In the non-adaptive techniques, the sizes of the chunks can be computed *before* the execution of the parallel loop and, thus, require prior knowledge about the application workload. FSC, GSS, FAC, WF [1]–[4] are three effective non-adaptive algorithms. In the adaptive techniques, the chunk sizes *online* are computed during the execution of the application. These techniques are efficient when dealing with highly unpredictable variations in applications and infrastructures. AWF and its variants (AWF-B, C, D, E) and AF [5]–[7] are examples of effective adaptive algorithms. For a comprehensive review of DLS techniques, the interested reader can refer to [14].

No DLS technique is well suited for all possible combinations of applications and infrastructures characteristics. Having a metric that quantifies the efficiency of how the DLS technique responds to perturbations is highly relevant due to variations in application and infrastructure characteristics [12].

Robustness metrics can offer a good indication on which DLS technique will respond best against various types of perturbations. The next section describes an approach to quantify the robustness of DLS techniques.

B. ROBUSTNESS

In the past few years, evaluation metrics have been developed to quantify the ability of DLS technique to resist to perturbations. These perturbations may appear in different forms both in the applications and the computing infrastructures. The robustness notation used throughout the present paper is introduced in the upper half of Table I.

The robustness “flexibility” has been proposed in [12], [16]

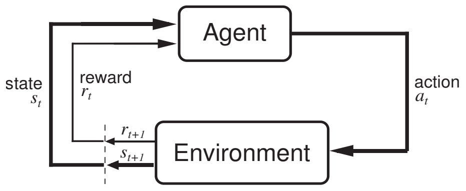

Figure 2: Reinforcement learning system structure [10].

to measure the capability of a DLS technique to resist to *variations in the loop iterations execution time* without exceeding the loop performance threshold $\theta = \tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$. The loop iterations execution time is a compound effect of the loop iterations computational effort (in flop) and the PE's availability to compute (in flop/s). The tolerance factor depends on the nature of the application (e.g., real-time, web, etc.). The flexibility of a DLS technique is related to the target application and can not be generalized. Throughout this paper, the terms "robustness" and "flexibility" are interchangeably used and refer to the robustness of a DLS technique with respect to the variations in the PE availability. The term PE availability is used throughout the paper as the inverse of the PE load and it represents the computing capability currently delivered by the PE.The procedure to compute the robustness flexibility radius r_{DLS} can be found in [12]. The lower the r_{DLS} the more robust the DLS technique. The classification of the DLS techniques in term of their flexibility is obtained by sorting them in ascending order in term of their r_{DLS}. Throughout the paper, the term flexibility metric is interchangeably used with the term flexibility radius and both refer to the flexibility of a given DLS technique in the presence of perturbations in PEs availability to compute.

C. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a sub-field of machine learning that allows learning how to make decisions under uncertainty for sequential decision problems modeled through a Markov decision process (MDP) [10]. The RL notation used throughout the present paper is introduced in the lower half of Table I.

MDPs are used to model decision making situations where consequences are partially under the control of a decision maker (e.g., a RL agent) and partly random [10]. An MDP is defined by a 4-tuple $\langle S, A, \mathcal{R}, \mathcal{T} \rangle$ defining the set of states S, the set of actions A, the reward function $\mathcal{R}(s,a)$, and the state-transition function $\mathcal{T}(s,a)$.

In RL, the agent learns how to behaves optimally in an unknown environment by acting and learning from the consequences of the actions it takes. Each time the agent acts, the environment rewards the agent with a scalar value (i.e., $\mathcal{R}(s,a)$), and communicates the new environment state to the agent (i.e., $\mathcal{T}(s,a)$). The agent has to decide which action to use given the current state of the environment to maximize its rewards, and to achieve a particular goal. Figure 2 illustrates the decision flow of a RL system.

RL algorithms can be divided into two groups: model-free and model-based. Model-free algorithms learn the optimal action-value function $Q^*(s,a)$ and use it to derive a control policy. This function represents the expected reward of taking an action a in a state s. Model-based algorithms learn the model of the environment (i.e., $\mathcal{R}(s,a)$ and $\mathcal{T}(s,a)$) and use it to derive the control policy. Because it is difficult to predict how $\mathcal{R}(s,a)$ and $\mathcal{T}(s,a)$ behaves in an unpredictable environment, this paper focus on model-free methods.

To learn the problem optimal control policy, model-free agents iteratively approximate $Q^*(s,a)$ via temporal difference (TD) learning. The agents updates the current approximation of $Q^*(s,a)$ after each action-reward cycle [10]. TD learning uses two parameters: $\alpha \in [0,1]$ (learning rate) and $\gamma \in [0,1]$ (discount factor). $Q(s,a)$ is used by the agent's behavioral policy to drive the selection of next actions. This composes the *current control policy* (i.e., $\pi(s)$).

Each model-free RL agent implements a behavioral policy that determines how to choose the next action. This policy decides whether the agent should operate with the current best choice (i.e., exploit) or test alternatives (i.e., explore). In the present research, we have considered the *Greedy policy*, which has no exploration phase and only exploits the best action so far.

Herein, we have considered the following five RL algorithms for the selection of robust DLS techniques:

QLearning It is the most popular RL algorithm which has been introduced in [17]. It uses the next *greedy* action as the current estimation of $Q^*(s,a)$ to update the action-value function [10], [17].

Sarsa It is another well-known RL algorithm which has been introduced in [18]. It uses the expected reward of the next action a' as the current estimation of $Q^*(s,a)$.

Expected Sarsa (ESARSA) This algorithm tends to reduce the stochastic aspect that may appear in the "behavioral" policies. For that purpose, it weighs the "value" of an action (i.e., $Q(s,a)$) by its probability to be chosen [19].

QVLearning The particularity of this algorithm is that it keeps track of both the state-value function $V(s)$ and the action-state function $Q(s,a)$. The idea of this algorithm is to use $V(s)$ as the current estimation of $Q^*(s,a)$ to converge faster to the optimal control policy [20].

Double QLearning It tends to avoids the learning overestimations that may appear in QLearning when dealing with stochastic MDP. Double QLearning uses two approximations of $Q^*(s,a)$ instead of one, and randomly updates one using the other [21].

The autonomic computing approach for selecting, at various time-steps of a TSSA, the most robust DLS technique is presented in the next section as well as the design of the reward signal. In this paper, we consider a modified flexibility metric as reward signal.

III. AN AUTONOMIC COMPUTING APPROACH FOR THE SELECTION OF ROBUST DLS

The approach proposed herein uses RL agent to learn optimal load balancing policy to optimize TSSA performance via robust scheduling. This section presents the adaptation of

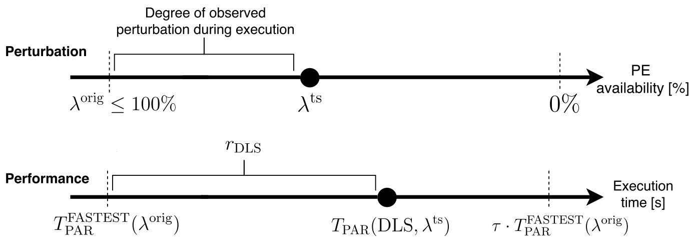
Degree of observed perturbation during execution
Perturbation
$\lambda^{\text{orig}} \leq 100\%$
λ^{ts}
0%
PE availability [%]
r_{DLS}
Performance
$T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$
$T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}})$
$\tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$
Execution time [s]

Figure 3: The traditional use of the flexibility metric, r_{DLS}, in TSSA [12], [16].

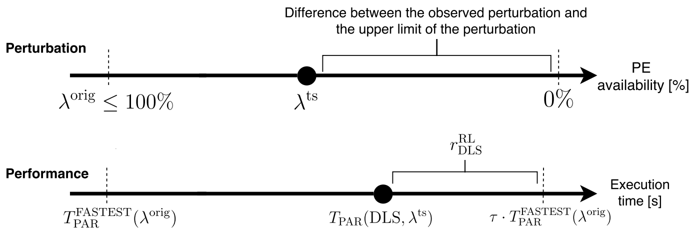
Difference between the observed perturbation and the upper limit of the perturbation
Perturbation
$\lambda^{\text{orig}} \leq 100\%$
λ^{ts}
0%
PE availability [%]
$r_{\text{DLS}}^{\text{RL}}$
Performance
$T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$
$T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}})$
$\tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$
Execution time [s]

Figure 4: The use of the modified flexibility metric, $r_{\text{DLS}}^{\text{RL}}$, as a reward.

the flexibility metric as a reward for RL agent, the formal definition of the sequential decision problem using MDP, and the description of the applications as well as the execution environments used to assess the effectiveness of the flexibility metric as a reward.

A. FLEXIBILITY METRIC VERSUS REWARD

A key challenge in the selection of robust DLS techniques via RL is to determine the objective of the agent. Determining the objective of the agent implies to define its reward after performing an action (i.e., selecting a DLS technique). In this work, the goal of the agent is to select the most robust DLS technique at each time-step 'ts' of a TSSA for a given parallel loop. The flexibility metric is used as a selection criterion. The flexibility metric measures the largest increase in the completion time of a DLS technique, for a given parallel loop, a given perturbation will cause. For an entire TSSA involving ts time-steps, it is formally given by:

$$r_{\text{DLS}} = \max_{\text{ts}} \left(T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}}) - T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})\right) \; \forall \text{ts} = 1, \text{N}. \tag{1}$$

where $T_{\text{PAR}}(\lambda^{\text{ts}}, \text{DLS})$ is the parallel loop execution time with the perturbation λ^{ts} at the 'ts'-th iteration and $T_{\text{PAR}}^{\text{FASTEST}}$ is the fastest ideal parallel loop execution time across all DLS techniques (Section II-B). Figure 3 illustrates the traditional use of the flexibility metric in TSSA.

A given DLS technique is said to be robust if the following equation holds for any level of perturbation that may occur during the execution of the TSSA:

$$T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}}) \leq \tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}}). \tag{2}$$

As mentioned in Section II-B, the lower the r_{DLS} the more robust the DLS technique. It is counterproductive to use the flexibility metric as a reward in its present form, as the aim of the RL agent is to maximize the long-term rewards. Hence, the flexibility metric needs to be adapted. Instead of measuring the robustness as the largest difference in completion time between $T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}})$ and $T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$ $\forall$ ts $= 1, \text{N}$, the robustness is measured as the smallest observed difference between $T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}})$ and $\tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}})$. Formally, the flexibility of a DLS technique for an entire TSSA is given by:

$$r_{\text{DLS}}^{\text{RL}} = \min_{\text{ts}} \left(\tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}}) - T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}})\right) \; \forall \text{ts} = 1, \text{N}. \tag{3}$$

The modified flexibility metric, $r_{\text{DLS}}^{\text{RL}}$, does not maintain the same scalar values as r_{DLS}, however, it preserves the same ranking in terms of flexibility. The greater the $r_{\text{DLS}}^{\text{RL}}$ the more robust the DLS technique. Figure 4 illustrates the use of the modified flexibility metric as a reward.

If a DLS technique has proven not to be robust (i.e., it has exceeded the loop performance threshold, at least once), the environment rewards the agent with a negative value to penalize this particular DLS technique. Hence, the reward function is given by:

$$\mathcal{R}(s, a) = \begin{cases} r_{\text{DLS}}^{\text{RL}}, & \text{if } \max\limits_{\text{ts}} T_{\text{PAR}}(\text{DLS}, \lambda^{\text{ts}}) \leq \tau \cdot T_{\text{PAR}}^{\text{FASTEST}}(\lambda^{\text{orig}}), \\ -1, & \text{otherwise.} \end{cases} \tag{4}$$

where $s \in \{\text{running, stopped}\}$, and $a \in \{\text{STATIC, FSC, GSS, FAC, AWF, AWF-B, AWF-C, AWF-D, AWF-E, AF}\}$ (Section III-B).

B. MARKOV DECISION PROCESS DEFINITION

Sequential decision problem solved by reinforcement learning are usually formalized through Markov decision processes (MDP). MDP are defined with a 4-tuple $\langle \text{S}, \text{A}, \mathcal{R}, \mathcal{T} \rangle$. In the present approach, it is defined as follows:
S: The set of states is composed of two states: {running,stopped}. Running is the entry state and stopped is the terminal state. The environment performs a state transition after the last time-step of the TSSA.
A: The set of DLS techniques is the set of actions, composed of ten DLS techniques: {STATIC, FSC, GSS, FAC, AWF, AWF-B, AWF-C, AWF-D, AWF-E, AF}. We used the load balancing tool proposed in [22] to have access to a DLS library implemented in C and using MPI.
$\mathcal{R}$: The reward function, $\mathcal{R}(s, a)$. In our approach, it returns the flexibility (as defined in Section III-A) of a particular DLS technique.
$\mathcal{T}$: The state-transition function, $\mathcal{T}(s, a)$. In the present approach, the environment starts in the "running" state when a new scenario (i.e., a combination of a TSSA and an infrastructure). The action of selecting a particular DLS technique $\in$ A does not produce any state-transition until the last time-step of the application. At the last time-step of the application, selecting an action leads the environment to transit to the "stopped" state.

The proposed approach works as follows: A RL agent is immersed in the target application context. One agent is implemented for each parallel loop. The agent is charged with choosing the DLS technique for executing its parallel loop.

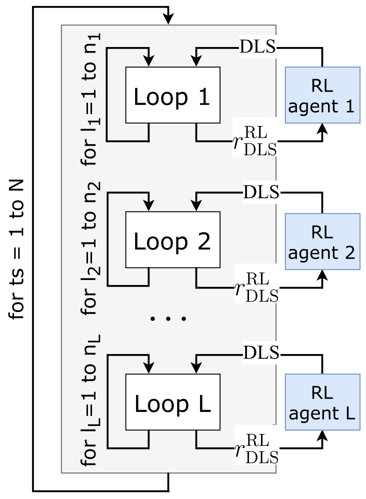

Figure 5: Autonomic selection of robust DLS techniques using the modified flexibility as a reward.

When the agent has chosen the DLS technique to use, it informs the loop scheduling library with its choice. Then, the loop scheduler executes the algorithm provided by the loop scheduling library on the computing environment. After the execution of the parallel loop, the agent is rewarded with the modified flexibility radius of the selected DLS technique as presented in Section III-A. The goal of the agent is, therefore, to maximize the long-term flexibility [10]. This approach aims at minimizing $T_{\mathrm{PAR}}(\mathrm{TSSA})$. Figure 5 illustrates the approach presented herein.

C. *SIMULATED EXECUTION ENVIRONMENT*

During the past decade, the need in simulating real world computing environment has grown. Several simulation tools have been developed over time, such as SimGrid [13], GridSim [23], MicroGrid [24]. SimGrid provides core functionality to build complex applications and infrastructure simulators. Different interfaces are available in SimGrid for simulating and evaluating scheduling algorithms under complex scenarios. SimDag (SD) allows the modeling and the simulation of applications structured as directed acyclic graphs. MetaSimGrid (MSG) provides a fine-grain control of the simulation for sending, receiving, and scheduling tasks. SMPI is used for executing existing MPI codes on a simulated distributed memory infrastructure.

These interfaces are implemented on top of SURF which is the core model of simulation. Herein, we used the SimGrid-SMPI interface to simulate the execution environment. The main motivation was to stay as close as possible from real world applications and to leverage our existing MPI code. SimGrid-SMPI provides an easy way to use simulation platforms and a realistic parallel programming paradigm (MPI).

SimGrid models simulation platforms via a description file. This file specifies the cluster characteristics such as the number of PE, the PE speed, network topology, bandwidth, latency, and PE speed variations. Variations in the PE speed are modeled through *trace files*. Each line of a trace file specifies the time at which a percentage of the resource will be available and the percentage of available resources at this time. This file is read sequentially and cyclically. Each cycle is separated by an user-defined number of seconds.

In the proposed approach, we decided to use several simulated computing infrastructures via SimGrid-SMPI. In particular, we described four infrastructures with perturbation in the PE availability to compute modeled via trace files. The PE availability is the percentage of the computing speed currently delivered by the processing element. The percentages of PE availability are chosen randomly following certain probability distributions (Table II). We also described five simulated TSSA via SimGrid-SMPI that use one parallel loop each during a thousand time-steps. Each parallel loop of these simulated TSSA behaves irregularly. SimGrid-SMPI allowed us to use the DLS library without effort.

IV. EVALUATION

To test and evaluate the approach presented herein, we chose to use SimGrid-SMPI for simulating both the TSSA and the computing infrastructures. The structure of the TSSA involved a single parallel loop executed during a thousand time-steps. Three parallel loop sizes have been considered: 4^{10}, 4^{11}, and 4^{12} iterations. The computational sizes (in flop) of the parallel loop iterations were generated randomly following probability distributions. It is worth to remind that during the execution of the application, the parallel loops iterations execution time is determined by the combined effect of the iteration workload and the PE availability to compute. We considered five probability distributions: (i) constant, (ii) uniform, (iii) normal, (iv) gamma and (v) exponential. Table II summarizes the parallel loops and the infrastructure probability distributions.

We targeted computing infrastructures of 2^{10}, 2^{11}, 2^{12} PE. The characteristics of these infrastructures are showed in Table II. The infrastructure characteristics have been chosen for a definite objective of testing the proposed approach on a realistic cluster which stress out the inter-processor communication overhead. We considered four types of infrastructure PE perturbations modeled via SimGrid-SMPI trace files. When a PE has an availability to compute less than 100%, it is said to be perturbed. PEs availability to compute varies over time every 5 seconds. After 500 seconds, PEs availability to compute restarts at time 0 (i.e., cyclic process). The PE availability to compute were chosen randomly following certain probability distribution. We took into account four probability distributions: (i) constant, (ii) uniform, (iii) normal, and (iv) gamma.

Due to limited space, only a selection of the most relevant results, representative of the observations we made and the patterns we identified, are presented in this paper. Results which belong to the setup involving a TSSA with a single parallel loop of 4^{10} iterations executing on a cluster of 2^{10} PE are presented in this paper.

Table II: Summary of the simulation parameters used for the applications and infrastructures, as well as the infrastructures characteristics.

TSSA loop distribution	Parameter	Min [flop]	Max [flop]
Constant	-	$2.3 \cdot 10^8$	$2.3 \cdot 10^8$
Uniform	-	10^3	$7 \cdot 10^8$
Normal	$\mu = 9.5 \cdot 10^8$, $\sigma = 7 \cdot 10^7$	$6 \cdot 10^8$	$1.3 \cdot 10^9$
Gamma	$k = 2$, $\theta = 10^8$	$4.1 \cdot 10^6$	$2.7 \cdot 10^9$
Exponential	$\lambda = 1/3 \cdot 10^8$	948	$4.5 \cdot 10^9$
PE availability distribution	**Parameter**	**Min [%]**	**Max [%]**
Constant	-	100	100
Uniform	-	10	100
Normal	$\mu = 0.5$, $\sigma = 0.1$	10	94
Gamma	$k = 2$, $\theta = 2$	10	100
Infrastructures characteristics		**Value**	
PE count		2^{10}, 2^{11}, 2^{12}	
PE speed		1Gflop/s	
Bandwidth		20Gb/s	
Latency (up/down)		10ms	

A. METHODOLOGY

To analyze the effectiveness of the approach proposed herein, we apply it on several TSSA executed on various simulated clusters that run applications programmed using MPI. Three cluster sizes have been taken into account. Different types of cluster perturbations and application behaviors have been employed. Five reinforcement learning algorithms have been tested as smart agents. The agents followed a greedy behavioral policy with the following learning parameters: $\alpha = 0.15$, $\gamma = 0.9$ and $\beta = 0.15$. The optimization of the learning parameters (i.e., hyper-parameters optimization) is left for future work as well as evaluating the proposed approach using different behavioral policies. The robustness tolerance factor τ was 1.5. This value was chosen on the basis of the relevant values proposed in [12].

One experiment has been launched for each combination of parameters and the experiments are grouped by case scenario (A TSSA comprised one parallel loop of N iterations following a loop distribution 'ld' and a computing infrastructure composed of P PE with a PE perturbation 'cp'). Hence, forty-five sets of five (i.e., one per each RL algorithm) experiments have been launched in total, and two are included in this paper. Additional results are hosted in a public repository and are available online.

B. EXPERIMENTAL RESULTS

For each set of experiments, we are interested in comparing the performance of the "robustness-RL approach" with a "fixed-DLS approach" for a given case scenario (i.e., an application and a perturbed infrastructure). We also want to determine which RL algorithm provides the best TSSA performance. Among the forty-five sets of experiments we have

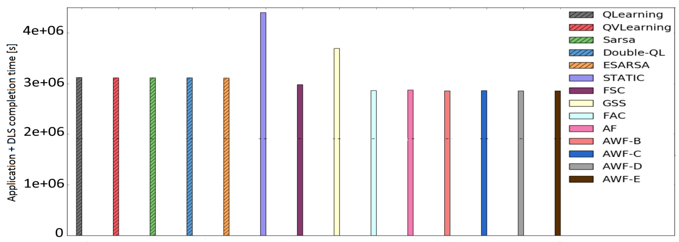

Figure 6: Performance results for the "exponential" application behavior on the "gamma" infrastructure.

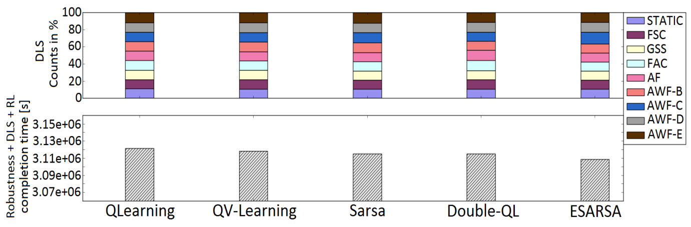

Figure 7: DLS selection results and RL performance for the "exponential" application behavior on the "gamma" infrastructure.

conducted, we present two sets that are highly representative of two patterns we identified.

Pattern 1: In the experiments involving extreme perturbations (e.g., a "gamma" infrastructure, or even a combination of an "exponential" parallel loop behavior and a "uniform" infrastructure), the RL agents are unable to identify an optimal load balancing strategy. They select the DLS techniques almost uniformly. The set of experiment representative of this pattern involves a TSSA with a parallel loop of 4^{10} iterations with task sizes randomly generated following an exponential probability distribution. The cluster size is 2^{10} PE. The availability of the PEs is randomly generated following a gamma probability distribution. The performance of the approach proposed herein using various RL algorithms compared to a fixed DLS approach is presented in Figure 6, whereas RL performance and the DLS selection rate are shown in Figure 7.

Pattern 2: In the experiments involving low perturbations (e.g., a "constant" infrastructure, or the a combination of a "normal" parallel loop behavior and a "uniform" infrastructure.), the RL agent is able to identify a single DLS technique that withstands the perturbations. The RL agent keeps using this technique throughout the execution of the TSSA even if it is not the optimal one. The set of experiments representative of this pattern involves a TSSA with a parallel loop of 4^{10} iterations with sizes randomly generated following a normal distribution. The cluster size is 2^{10} PE. The availability of the PEs is randomly generated following an uniform probability distribution. The performance of the approach proposed herein using various RL algorithms compared to a fixed DLS approach is presented in Figure 8, whereas RL performance and DLS selection rate are shown in Figure 9.

From these experiments, three observations are drawn. (i) The modified flexibility does not take into account the

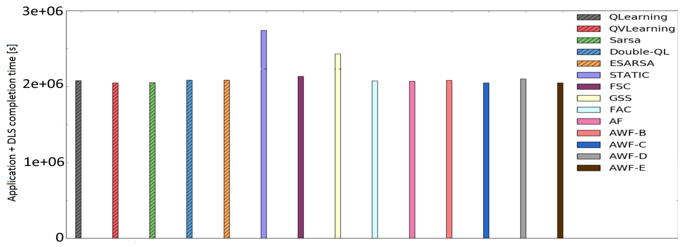

Figure 8: Performance results for the uniform application behavior on the normal infrastructure.

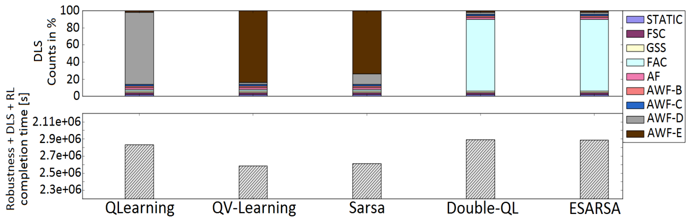

Figure 9: DLS Selection results and RL performance for the uniform application behavior on the normal infrastructure.

current performance of the DLS technique to compute $r^{\mathrm{RL}}_{\mathrm{DLS}}$. Conversely, it uses the slowest parallel loop completion time over all TSSA time-steps and the loop performance threshold to compute $r^{\mathrm{RL}}_{\mathrm{DLS}}$ (Equation (3)). Thus, if a DLS technique suddenly performs better, the flexibility metric will not take this change into account. This lack of adaptability makes the flexibility, in its modified form, a poor representation of the current environment state. This issue leads the agent to be likely to use a sub-optimal DLS technique. (ii) When no DLS technique is robust enough to withstand the perturbation, the agent begins to select the DLS techniques uniformly (Figure 7). (iii) QVLearning proves to outperform the others RL algorithms considered herein in a vast majority of experiments.

V. CONCLUSIONS AND FUTURE WORK

This paper presents an autonomic computing approach for performance optimization of the TSSA via an autonomic selection of robust DLS techniques. This approach employs a RL agent for each parallel loop of a TSSA. At each time-step, each RL agent selects the DLS technique to use for their assigned parallel loop. The present work proposes a modified version of the flexibility metric to reward the choices of the RL agents. The goal of every RL agent is to maximize the long-term flexibility. This paper uses a generic simulation framework to study the effectiveness of the modified flexibility metric as a RL reward. Various TSSA and infrastructures have been simulated via SimGrid-SMPI. Five reinforcement learning algorithms (QLearning, Sarsa, Expected Sarsa, QVLearning, Double QLearning) have been studied for the robust-RL approach. Forty-five sets of experiments using five RL techniques have been performed, and the most relevant two sets have been presented herein, on the basis of two patterns we identified.

The results show that the RL agent was unable to select the optimal DLS technique during the execution of a TSSA facing extreme perturbations. The robust-DLS-with-RL performed either as good as, or worse than, the fixed-DLS approach. The main problem of this approach lies in the unsuitable representation of the environment state by the modified flexibility metric as a reward. This is due to the fact that the flexibility of a particular DLS technique is monotonically decreasing over time. Therefore, the flexibility metric in both its previous (Section II-B) and present form (Section III-A) are not good indications of the actual performance of a DLS technique. These results refute the effectiveness of the flexibility metric as a suitable reward for an autonomic selection of robust DLS techniques in TSSA via RL. A better approach requires a criterion that is able to precisely represent the current state of the system (e.g., $T_{\mathrm{PAR}}(\mathrm{DLS}, \lambda^{\mathrm{ts}})$). Besides the conclusions on the effectiveness of the flexibility metric, the results of this work have also pointed out that, between the five reinforcement learning algorithms considered herein, QVLearning proves to be the most effective. The overhead of the RL techniques is less than 0.01% of the entire application execution.

A generic RL approach for optimizing TSSA via the autonomic selection of DLS techniques presents many advantages, pointed out in earlier works. The flexibility metric can improve the quality of the DLS selection, given its capability to measure the quantity of absorbed perturbations. Hence, the investigations on how to use the flexibility metric as a RL reward is still highly relevant and will be pursued in future work. Making the robustness a better state representation could be beneficial. For instance, considering the flexibility of a particular DLS technique at a particular time-step as a reward could provide better results. Or, combining the robustness and another metric, such as the execution time via multi-objective reinforcement learning [25], is of great interest. Moreover, tuning the RL agent hyper-parameters as well as using more complex behavioral policies would improve learning the benefits of the load balancing policy, and increase the performance of the TSSA.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grant number NSF IIP-1034897 and the Swiss National Science Foundation in the context of the Multi-level Scheduling in Large Scale High Performance Computers (MLS) grant, number 169123.

REFERENCES

[1] C. P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Trans. Softw. Eng.*, pp. 1001–1016, Oct 1985.
[2] C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Trans. Comput.*, vol. C-36, no. 12, pp. 1425–1439, Dec 1987.
[3] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: A method for scheduling parallel loops," *Commun. ACM*, pp. 90–101, Aug. 1992.
[4] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring," in *Proc. 8th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1996, pp. 318–328.
[5] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," in *Proc. IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, Apr. 2001, pp. 791–801.

[6] I. Banicescu, V. Velusamy, and J. Devaprasad, “On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring,” *Cluster Computing*, vol. 6, no. 3, pp. 215–226, 2003.

[7] I. Banicescu and V. Velusamy, “Load Balancing Highly Irregular Computations with the Adaptive Factoring,” in *Proc. IEEE Int. Parallel and Distributed Processing Symposium*, Apr. 2002, pp. 195–.

[8] S. Dhandayuthapani, I. Banicescu, R. L. Carino, E. Hansen, J. R. Pabico, and M. Rashid, “Automatic selection of loop scheduling algorithms using reinforcement learning,” in *Proc. IEEE Challenges of Large Applications in Distributed Environment.*, July 2005, pp. 87–94.

[9] I. Banicescu, F. M. Ciorba, and S. Srivastava, *Scalable Computing: Theory and Practice*. John Wiley & Sons, Inc., 2013, no. Chapter 22, ch. Performance Optimization of Scientific Applications using an Autonomic Computing Approach, pp. 437–466.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[11] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, “Measuring the robustness of a resource allocation,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 7, pp. 630–641, 2004.

[12] I. Banicescu, F. M. Ciorba, and R. L. Carino, “Towards the Robustness of Dynamic Loop Scheduling on Large-Scale Heterogeneous Distributed Systems,” in *IEEE Int. Symposium on Parallel and Distributed Computing (ISPDC).*, June 2009, pp. 129–132.

[13] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” *J. Parallel Distrib. Comput.*, pp. 2899–2917, 2014.

[14] I. Banicescu and R. L. Cariño, “Addressing the stochastic nature of scientific computations via dynamic loop scheduling,” *Electron. Trans. Numer. Anal.*, vol. 21, pp. 66–80, 2005.

[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,” in *26th Int. Parallel and Distributed Processing Symposium Workshops PhD Forum*, May 2012, pp. 1343–1351.

[16] S. Srivastava, I. Banicescu, and F. M. Ciorba, “Investigating the Robustness of Adaptive Dynamic Loop Scheduling on Heterogeneous Computing Systems,” in *Int. Symposium on Parallel and Distributed Processing, Workshop Proceedings, IPDPSW*, April 2010, pp. 1–8.

[17] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, UK, May 1989.

[18] G. A. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems,” University of Cambridge, Department of Engineering, Technical Report, 1994.

[19] H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering, “A Theoretical and Empirical Analysis of Expected Sarsa,” in *Proc. IEEE Symposium on ADPRL*, March 2009, pp. 177–184.

[20] M. A. Wiering and H. van Hasselt, “Two Novel On-Policy Reinforcement Learning Algorithms based on TD(λ)-Methods,” in *Proc. IEEE Symposium on ADPRL*, 2007, pp. 280–287.

[21] H. V. Hasselt, “Double Q-learning,” in *Advances in Neural Information Processing Systems 23*, 2010, pp. 2613–2621.

[22] R. L. Cariño and I. Banicescu, “A load balancing tool for distributed parallel loops,” *Cluster Computing*, vol. 8, no. 4, pp. 313–321, 2005.

[23] R. Buyya and M. Murshed, “Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[24] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, “The Microgrid: A scientific tool for modeling computational grids,” in *Proc. IEEE Supercomputing (SC)*, Nov. 4-10.

[25] C. Liu, X. Xu, and D. Hu, “Multiobjective reinforcement learning: A comprehensive overview,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 3, pp. 385–398, 2015.