Reinforcement Learning for Multi-Objective Task Placement on Heterogeneous Architectures with Real-Time Constraints

Bakhta Haouari[1,2,3][a], Rania Mzid[1,4][b] and Olfa Mosbahi[2][c]
[1]*ISI, University Tunis-El Manar, 2 Rue Abourraihan Al Bayrouni, Ariana, Tunisia*
[2]*LISI Lab INSAT, University of Carthage, Centre Urbain Nord B.P. 676, Tunis, Tunisia*
[3]*Tunisia Polytechnic School, University of Carthage, B.P. 743, La Marsa, Tunisia*
[4]*CES Lab ENIS, University of Sfax, B.P:w.3, Sfax, Tunisia*

Keywords: Real-Time, Task Placement, Multi-Objective, Refactoring, Pareto Q-Learning.

Abstract: This paper introduces a novel approach for multi-objective task placement on heterogeneous architectures in real-time embedded systems. The primary objective of task placement is to identify optimal deployment models that assign each task to a processor while considering multiple optimization criteria. Given the NP-hard nature of the task placement problem, various techniques, including Mixed Integer Linear Programming and genetic algorithms, have been traditionally employed for efficient resolution. In this paper, we explore the use of reinforcement learning to solve the task placement problem. We initially modeled this problem as a Markov Decision Process. Then, we leverage the Pareto Q-learning algorithm to approximate Pareto front solutions, balancing system extensibility and energy efficiency. The application of the proposed method to real-world case studies showcases its effectiveness in task placement problem resolution, enabling rapid adaptation to designer adjustments compared to related works.

1 INTRODUCTION

Real-time embedded systems (RTES) are computing system designed to perform specific functions with precise timing constraints. Nowadays, these systems are commonly found in various aspects of everyday life, ranging from customer electronics to healthcare applications (Akesson et al., 2020). They typically consist of both hardware and software components tightly integrated to execute tasks efficiently. Indeed, RTES are characterized by their ability to promptly respond to external stimuli within strict timing constraints, offering reliability and performance crucial for a wide array of applications. Developing such systems is inherently challenging, given that any failure could have critical implications for human safety.

The software engineering community is facing various challenges today during the development of RTES, including the task placement problem (Lassoued and Mzid, 2022). This problem involves assigning tasks to processors in a manner that satisfies real-time constraints while optimizing system performance metrics. As real-time embedded systems become more complex, solving the task placement problem becomes much harder. In fact, task placement turns into an NP-hard problem, which essentially means that finding the best solutions becomes extremely difficult within a reasonable amount of time. In response, researchers have explored various optimization techniques, including exact methods such as Mixed Integer Linear Programming (MILP) (Mehiaoui et al., 2019; Lakhdhar et al., 2020), Reinforcement Learning (RL) (Haouari et al., 2022; Haouari et al., 2023b), and Genetic Algorithms (GA) (Lassoued and Mzid, 2022), to tackle task placement efficiently.

In this paper, we explore the use of reinforcement learning to solve the multi-objective task placement problem in RTES. We introduce a new method called Pareto Q-learning Placement (PQP). This method relies on the Pareto Q-learning algorithm (Van Moffaert and Nowé, 2014), an emerging artificial intelligence technique known for its effectiveness in such scenarios. The PQP method is designed to generate a set of Pareto-optimal solutions for the task placement prob-

[a] https://orcid.org/0000-0002-5336-6300
[b] https://orcid.org/0000-0002-3086-370X
[c] https://orcid.org/0000-0002-0971-2368

Haouari, B., Mzid, R. and Mosbahi, O.
Reinforcement Learning for Multi-Objective Task Placement on Heterogeneous Architectures with Real-Time Constraints.
DOI: 10.5220/0012721500003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 179-189
ISBN: 978-989-758-696-5; ISSN: 2184-4895

lem in RTES, adhering to all specified constraints, particularly those related to real-time requirements, while simultaneously improving system extensibility and reducing energy consumption. In addition to its ability to produce Pareto-optimal solutions, the proposed PQP method addresses the challenge of refactoring. Refactoring refers to the process of modifying the system properties such as updating tasks properties (Haouari et al., 2022). The main contributions of this paper may be summarized as follows: (i) We model the task placement problem as a Markov Decision Process (MDP), (ii) We use the Pareto Q-learning algorithm to propose a multi-objective decision maker called PQP. The latter offers to the designer a set of deployment models referring to the possible task-to-processor assignments, balancing the system extensibility and energy efficiency. (iii) We apply and simulate the proposed PQP method to real-life case studies. The results obtained demonstrate the effectiveness of the proposed methods in efficiently identifying Pareto solutions compared to existing approaches. Additionally, they showcase the capability of the proposed methods to address the refactoring issue.

The rest of the paper is organized as follows: Section 2 discusses the relevant literature. Section 3 introduces key concepts related to multi-objective optimization. Section 4 presents the RTES formalization. The description of the task placement problem as a MDP is introduced in Section 5. Section 6 introduces the PQP method and provides detailed explanations of the RL algorithms. Experimental results are detailed in Section 7, and the paper concludes with Section 8, which discusses future directions of our research.

2 RELATED WORK

Several approaches have been proposed in the literature to deal with the task allocation problem in RTES. In (Zhu et al., 2013), the authors utilize MILP technique to optimize task and message allocation within a distributed system, aiming to meet end-to-end deadlines and minimize latencies. In (Vidyarthi and Tripathi, 2001), a GA-based method is proposed for maximizing reliability in the task allocation problem. In (Kashani et al., 2017), the authors deal with the same problem and propose a method based on GA to minimize the communication cost. In (Haouari et al., 2023a), RL techniques are applied to introduce a novel approach to real-time task placement and scheduling. Unlike previous approaches, this paper systematically explores all feasible placements before assigning a scheduling to each solution, defining the optimal scheduling as the one that minimizes response time. Additionally, in (Haouari et al., 2023b), the authors delve into self-adaptive systems, proposing a new method capable of effectively functioning under both predictable and unpredictable online system updates. Their proposed approach, represented by a set of RL algorithms, aims to optimize system extensibility while ensuring real-time feasibility. Another notable work by (Haouari et al., 2022) leverages the Q-learning algorithm for task placement, with the goal of reducing the number of utilized processors. This study also addresses the refactoring issue, demonstrating the effectiveness of RL methods in coping with updates to system parameters. Despite the significance of these works in addressing the placement problem in RTES, they are limited to single-objective optimization, which may not fully capture the complexities of real-world scenarios.

In addressing multi-objective optimization in the task allocation problem in RTES, the authors of (Mehiaoui et al., 2019) propose an approach aimed at addressing placement and scheduling challenges within embedded distributed architectures, where multiple objectives need to be optimized collectively. They employ both MILP and GA (Mirjalili, 2019) to handle the scalability issues associated with MILP. To address the set of objectives comprehensively, the authors suggest consolidating them into a single objective function, assigning weights to each objective. This process yields an optimal solution that strikes a balance among the objectives for the entire problem. However, as discussed in (Coello, 2007), the authors demonstrate that despite executing this process multiple times with varying weights, certain solutions that do not align with any combination of weights remain undetected. In (Lassoued and Mzid, 2022) a multi-objective algorithm based on evolutionary algorithms, specifically SPEA2 (Huseyinov and Bayrakdar, 2022), was proposed to optimize both the slack capacity and the consumed energy of the system for distributed processors. Nevertheless, the scalability of the GA model is evident. The authors of (Mehiaoui et al., 2019) conclude that evaluating and ensuring the quality of solutions generated by the GA is challenging, given its dependency on various factors such as encoding, crossover, and mutation operators. Despite their importance, these works do not address the refactoring issue, as the MILP and GA formulations must be re-executed from scratch after application properties changes.

Some studies have achieved success utilizing RL techniques to address problems akin to task placement in domains beyond RTES, including the work referenced as (Caviglione et al., 2021). In this paper, the authors introduce a multi-objective approach utiliz-

ing a deep reinforcement learning (RL) technique for addressing the online placement of virtual machines (VMs) across a group of servers within cloud data centers. Nevertheless, their methodology employs the weighted sum function, imposing constraints on its applicability to linear problems. The study presented in (Yang et al., 2020) introduces a multi-policy convex hull reinforcement learning algorithm designed for managing the operations of We-Energy in the context of the Energy Internet. The algorithm leverages Q-learning twice: initially, to construct a set of Pareto multi-objective solutions within a continuous state space, and subsequently, to select the optimal solution that minimizes We-Energy production costs while maximizing the security of each We-Energy unit in the system. This proposed algorithm incorporates multi-objective reinforcement learning techniques, and it incorporates human involvement in the adjustment of the application to ensure system control and enhance the confidence of intelligent systems. Inspired by the aforementioned approaches, this work aims to cope with the limitations of existing methods for task placement in RTES. In this paper, we aim to extend the work proposed in (Haouari et al., 2022) to deal with multi-objective task placement to be mapped in heterogeneous architectures.

3 BACKGROUND

The resolution of optimization problems where a single or non conflicting objective is considered leads to a single solution, which may be the optimal or near optimal one. In multi-objective optimization where objectives may be conflicting, the resolution step is not as simple. In fact, we are faced to a collection of solutions representing a trade-off between the objectives. The decision maker can navigate in this collection and choose the one that best fits their needs. In this section, some necessary definitions commonly used in multi-objective optimization problems are given :

- *Pareto dominance*: In a multi-objective optimization approach, optimization problems are defined by conflicting objectives. This implies that enhancing the solution with respect to one criterion may potentially worsen another. In these scenarios, there is no singular optimal solution ; instead, there exists a set of non-dominated solutions, each representing a trade-off between the objectives. For instance, if we consider two solutions, π_1 and π_2, for the same problem, to dominate π_2, it is sufficient and necessary for π_1 to be at least better than π_2 on all the considered objectives. In Figure 1, solutions depicted as red diamonds are dominated by solutions depicted as large blue circles.
- *Pareto optimality*: π_1 is deemed a Pareto optimal solution when there is no other solution π that dominates π_1. The set of Pareto-optimal solutions constitutes the Pareto set. In Figure 1, this collection is illustrated by the set of solutions shown as large blue circles.
- *The Pareto front*: Represents a specific subset of the Pareto set, encompassing only solutions along the frontier. These solutions are characterized by the trade-off phenomenon, wherein the enhancement of one objective results in the degradation of another. The Pareto front is the line that delineates the portion of the plane containing the dominated points. The solutions illustrated as large blue circles in Figure 1 comprise the Pareto front.

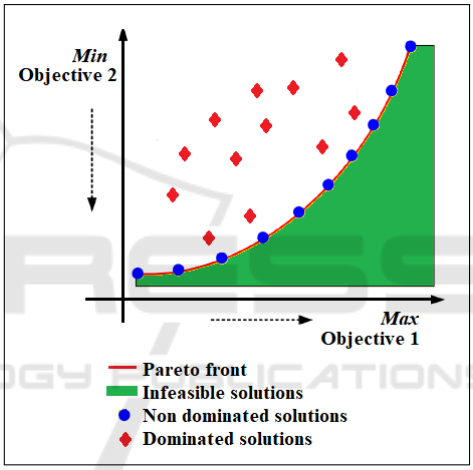

Figure 1: Pareto front curve.

- *Hypervolume*: In a multi-objective scenario, we encounter a collection of solutions that approximate the Pareto front. It has been demonstrated in (Zitzler and Thiele, 1998; Fonseca et al., 2006) that hypervolume serves as the most effective metric for evaluating the quality of a given set of solutions. Figure 2 illustrates the region (colored in green) to which the hypervolume measurement pertains. This region delineates the space occupied by solutions dominated by the approximated Pareto front, bounded by this front and a reference point (i.e., a designated point dominated by all solutions). Therefore, a higher hypervolume value corresponds to superior solution quality.

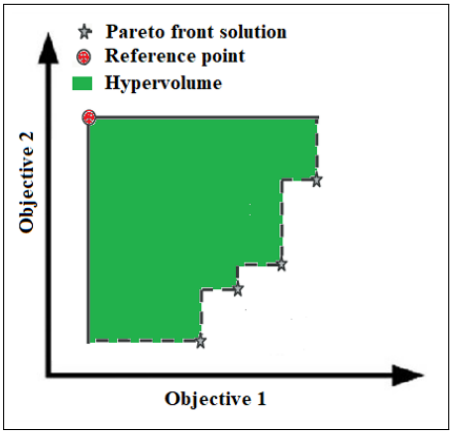

Figure 2: Hypervolume.

4 SYSTEM FORMALIZATION

The task placement problem, involves three different types of models: the task model, the platform model, and the deployment model. It is assumed in this work that the task model, that we denote by τ, is composed of n synchronous, periodic, and independent tasks (i.e.,$\tau = \{T_1, T_2 \dots T_n\}$). Each task T_i is characterized by static parameters $T_i = (\mathcal{C}_i, Pr_i)$ where $\mathcal{C}_i = (c_1, \dots, c_m)$ such as c_{ij} represents an estimation of the worst case execution time of the task T_i on the processor P_j; $i \in \{1 \dots n\}$ and $j \in \{1 \dots m\}$, and Pr_i is the activation period of the task T_i. The platform model, that we denote by $\mathcal{P}$, represents the execution platform of the system. We assume that this model is composed of m heterogeneous processors (i.e., $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$). Each processor has its own memory and runs a Real-Time Operating System (RTOS). The task placement step produces a deployment model that we denote by $\mathcal{D}$ in this work. The deployment model consists of a set of tuples $\mathcal{D} = \{(P_1, \xi_1), (P_2, \xi_2), \dots (P_k, \xi_k)\}$ where k represents the number of active (or used) processors such as $k \leq m$ and ξ_j represents the subset of tasks allocated to the processor P_j after the placement step. The primary objective of the approach developed in this paper is to minimize the energy consumption (denoted by E) of the processors responsible for executing the designated tasks. To quantify E for a specific deployment model, we rely on the energy model outlined in (Lakhdhar et al., 2018). In this model, each processor P_j is associated with a constant ζ_j representing its capacitance (assumed to be equal to 1 in this work), a frequency f_j denoting the processing speed, and a voltage v_j corresponding to the electrical supply. The energy required for the proper operation of processor P_j, denoted as E_j, is calculated using Equation 1

$$E_j = \sum_{i \in \{K_j\}} E_{ij} = \sum_{i \in \{K_j\}} \zeta_j f_j V_j^2 * c_{ij}, j \in \{1 \dots m\} \quad (1)$$

where K_j is the set of tasks allocated to P_j, E_{ij} is the energy required for processor P_j to perform task T_i and the global energy E is computed as Equation 2

$$E = \sum_{j=1}^{m} E_j \quad (2)$$

For real-time embedded systems, it is imperative that the deployment model is deemed feasible. Feasibility, in this context, entails that placing real-time tasks on various processors ensures the timing specifications of the system are met. In this regard, Liu and Layland (Liu and Layland, 1973) defined a schedulability test that is both necessary and sufficient when the task model satisfies the Rate Monotonic (RM) optimality conditions. The feasibility test serves to ascertain whether a given set of tasks will consistently meet all deadlines under all release conditions. This test relies on the calculation of the processor utilization factor U_p and is defined as follows:

$$U_{P_j} = \sum_{i=1}^{n} \frac{c_{ij}}{Pr_i} \leq 0.69 \quad (3)$$

In tackling refactoring, our approach involves maximizing the system's adaptability, referred to as extensibility. Extensibility denotes the system's ability to accommodate changes while preserving feasibility. We measure this extensibility through the concept of "slack capacity," which signifies the minimum remaining processor capacity post task placement over all the considered processors. The second objective pursued by PQP is the maximization of slack capacity, defined as:

$$SlackCapacity(P_j) = \min_{1..m}(0.69 - U_{P_j}) \quad (4)$$

Essentially, the greater the available capacity on the processors, the more adaptable the system is to potential changes.

5 TASK PLACEMENT PROBLEM AS AN MDP

In this study, the task placement challenge is designed as a sequential decision problem. Specifically, at each time step t, the system is characterized by the states of the processors (i.e., the list of tasks placed on each processor which is required to compute its available utility) and the list of tasks that has not yet

been placed. Consequently, the new task placement is contingent solely upon the current state of the system. This adherence to the Markov property (Bellman, 1957), defined as an essential condition for applying RL (Barto, 2021), justifies the utilization of this technique in our context. The application of RL techniques necessitates the establishment of key concepts, such as:

- *State (S_t)* : denoted by S, it reflects the state of the system at time step t. It undergoes changes based on the actions performed by the agent at each step until reaching the final stage. In the context of the task placement problem, S_t consists of sets, each composed of a processor and its currently assigned tasks, along with the set of unplaced tasks. The final stage aligns with the completion of the task placement process, involving the assignment of all tasks.
- *Epoch of Decision* : It aligns with the completion of the placement of the entire set of tasks on the designated processors. It concludes with the generation of a deployment model.
- *Agent* : It serves as the decision-maker for the system, responsible for selecting an action to transit from state S_t to S_{t+1} with the goal of maximizing its reward in accordance with a policy π. The decision-maker undergoes a learning process from one epoch to another to fulfill its mission, and its performance is contingent upon both the action selection process and the received reward. In the task placement problem, the objective is to discover an optimal deployment model within a multi-objective context.
- *Action Space* : Denotes the set of feasible actions available to the agent when it is in state S_t following a policy π. For the task placement problem, the action involves determining the optimal processor for a given task from the list of unplaced tasks, adhering to an ε-greedy policy (Barto, 2021), where ε is a small number that have to be initialized. Initially, a high value of ε is assigned to promote environmental exploration in the early stages. The ε-value is systematically reduced after each epoch by an ε-decrease factor. This decline in ε is justified by the agent's learning progression from one epoch to the next. As the learning process advances, new states are explored, leading to a preference for exploitation over exploration.
- *Reward R* : The reward holds paramount significance in system modeling, serving as the agent's motivator that influences its choices during each action selection. In the context of the task placement problem, the reward functions as positive reinforcement, resembling a gift, when the agent makes an appropriate selection, or as a penalty in alternative scenarios. Additionally, given the multi-objective nature of our approach, it becomes essential to designate specific rewards for each objective. In the ensuing discussion, we define two rewards: one for addressing slack capacity and another for consumed energy.

Slack-Capacity Reward.

$$Rsc = SP_j - U_{jt} + U_{ij} \tag{5}$$

Where

- U_{jt} is the available utilization of processor P_j at time step t
- U_{ij} is the required utilization for a task T_i to turn on P_j(i.e., $U_i = \frac{C_i}{Pr_i}$)
- SP_j reflects the processor state such as

$$SP_j = \begin{cases} -(m+\delta), & \text{When there is not enough space on } P_j \text{ to support } T_i \text{ (with } \delta > 0) \\ \text{-m}, & \text{When } T_i \text{ is placed on } P_j \text{ and it is not the processor with the most free capacity} \\ 0, & \text{Otherwise} \end{cases}$$

Where m denotes the number of processors defined in the hardware model.

Energy Consumption Reward.

$$Re = \begin{cases} \frac{1}{E_{ij}}, & \text{when } E_{ij} \text{ is the minimum over all } P_j \\ -m & \text{Otherwise} \end{cases} \tag{6}$$

Where

- E_{ij} represents the energy needed for processor p_j to execute task T_i
- m is the number of processors defined in the hardware model.

6 PQP DESCRIPTION

The proposed Pareto Q-learning Placement (PQP) method aims to address placement problems characterized by a set of tasks that need to be assigned to a set of heterogeneous processors (as discussed in Section 4). This method comprises a series of algorithms designed to generate the Pareto-optimal set.

Initially, the RL Initialization algorithm (Algorithm 1) is called to start with essential initialization steps. The objective of this algorithm is to configure various data structures accommodating inputs such as the task model and the platform model for the given problem and some initialization variables such as γ that denotes the discounting factor, which reflects the importance given to expected rewards, ε, and ε-decrease (as defined in Section 5).

After the initialization step, Algorithm 3 uses an ε-greedy technique for the action selection step, which consists of randomly generating a number *nb* in [0, 1]. This algorithm compares the *nb* number with ε; if *nb* is equal to or less than ε then the action ((task, processor) pair) is randomly chosen; otherwise, it invokes the Task Selection algorithm, referred to as Algorithm 2, to determine the optimal task placement. Following the execution of this procedure, the agent receives a reward pair $(r.sl, r.e)$, where *r.sl* and *r.e* correspond to the slack-capacity and energy objectives, respectively. This pair of immediate rewards is employed to update the average reward R, which is in turn combined with the non-dominated set $ND_t(S,a)$ to compute the $Qset$. Here, $ND_t(S,a)$ signifies the reward pairs of non-dominated solutions at time step t for the state S and the action a. It undergoes updates based on the non-dominated vectors of S' with all possible actions a'. Similarly, $Qset(S,a)$ represents a collection of vectors for the state-action pair (S,a). It stores the non-dominated set of Q-values obtained by adding the average immediate reward R to the non-dominated set $ND_t(S,a)$ expected in state S' and discounted by γ. Subsequently, the agent transits to state S', removes the placed task from the list of unplaced tasks, updates the deployment model D_t with the new placement, and the agent is replenished for another task placement until all tasks in the unplaced list are exhausted. At this moment, the final D_t is added to $\mathcal{D}$, and a new epoch searching for another non-dominated optimal model is stated. The algorithm concludes by generating a set of non-dominated deployment models $\mathcal{D}$. Underlining its importance, it must be emphasized that the epoch count is a critical parameter that needs to be carefully chosen to ensure the convergence of PQP. Convergence is realized when no new deployment model is generated that differs from those already present in.

The strategy employed for action selection plays a pivotal role as it significantly impacts both the speed and quality of the agent's learning process. In the context of multi-objective optimization, this step becomes intricate, deviating from the straightforward process in classical Q-learning, where the action evaluation considers a single optimal solution. Here, the evaluation must account for non-dominated solutions concerning two conflicting objectives, namely, the slack capacity and the consumed energy. Various strategies for the action selection process have been proposed in (Van Moffaert and Nowé, 2014). We specifically adopt the Pareto action selection method, identified as the most efficient in (Van Moffaert and Nowé, 2014). The implementation of Pareto action selection is detailed in Algorithm 2. Initially, the Q-values associated with each pair (action and its corresponding Q-set) for all available actions at state S are collected into a list called Qsg. Subsequently, the ND operator is applied to Qsg, eliminating dominated elements and retaining only the non-dominated ones in the NDQ list. From this list, the agent randomly selects an action. This random choice allows, for the agent, more exploration of the NDQ set.

Algorithm 1: RL Initialization.

Data: τ: The task model;
$\mathcal{P}$: The platform model;
ε, $\varepsilon - decease$, γ : Initialization parameters;
Result: Q : The Q-table;
S: the initial state;
$Q \leftarrow$ *Create* $Q - table(S,A)$;
Initialize Q[Qset] to empty sets;
Initialize S to initial state;
return Q, S

Algorithm 2: Task Selection.

Data: S: Current state;
Q: Q-table;
Result: a: action (task,processor);
foreach a ***in*** S_t **do**
  $Qsg \leftarrow$ empty set;
  foreach *vector v* ***in*** *Qset(S,a)* **do**
    Append (a,v) to Qsg ;
  end
end
NDQS $\leftarrow$ ND(QSG);
$a \leftarrow$ choose randomly one element from NDQS;
return a

7 CASE STUDY

This section illustrates the applicability of the proposed PQP method based on the case study outlined in (Haouari et al., 2022). The case study involves an RTES implemented in a car to assist the driver in vehicle control. The task model of the RTES consists of five independent, periodic, and synchronous tasks:

Algorithm 3: Pareto Q-learning Placement (PQP).

Data: S: The initial state;
τ: The task model;
$\mathcal{P}$: The platform model;
ε, $\varepsilon - decease$, γ: Initialization parameters;
Result: $\mathcal{D}$: the deployment models;
Notations:
D_t: deployment model at time step t;
a: the action (Task,Processor);
for t ***from*** 1 ***to*** $number\ of\ epochs$ **do**
  (S, Q) ← RL Initialization (τ, $\mathcal{P}$) ;
  while τ <> ∅ **do**
    $nb \leftarrow$ random number $\in [0,1]$;
    if $nb \leq \varepsilon$ **then**
      Select a random possible action a ;
    else
      $a \leftarrow Task\ Selection(S,Q)$;
    end
    Perform placement a, Observe state $S\prime$;
    $r.sc \leftarrow Rsc$;
    $r.e \leftarrow Re$;
    $Q.nb(S,a) \leftarrow Q.nb(S,a) + 1$;
    $R(S,a) \leftarrow R(S,a) + \frac{r - R(S,a)}{nb(S,a)}$;
    $ND_t \leftarrow ND(\cup_{a\prime}(Qset(S\prime,a\prime)))$;
    $Qset(S,a) \leftarrow R(S,a) \oplus \gamma\, ND_t(S,a)$;
    $S \leftarrow S'$;
    update τ /* /* remove the placed task from τ */ */ ;
    update D_t with the new placement a;
  end
  $\mathcal{D} \leftarrow \mathcal{D} \cup D_t$;
end
return $\mathcal{D}$

three dedicated to sensor measurements such as the car's speed, the car's temperature, and the car's GPS location, and two for displaying the measured values such as the measurement summary and the map of the current car location. In this paper, we expand upon the platform model described in (Haouari et al., 2022) by incorporating three heterogeneous processors instead of homogeneous ones. Table 1 offers a detailed description of the case study where T_i denotes the task names ($i \in [1..5]$), Pr_i corresponds to the task periods, and C_1, C_2, and C_3 match the worst-case execution time of T_i on P_j.

In order to determine the appropriate deployment model for the specified case study, the designer employs the PQP method, implemented in Python 3 with the NumPy library. This analysis is conducted using the task model outlined in Table 1 on the platform model described in Table 2. Certain parameters need to be initialized to ensure the proper functioning of the PQP algorithms, including γ, which serves as the discount factor quantifying the significance assigned to future rewards. In our approach, we deem future task placements important for generating the final deployment model, and therefore, we assign a sufficiently high value to γ (= 0.9). For the action selection, ε is assigned a value of 0.9, and we consider an ε-decrease value of 0.001 (c.f. Section 5). Table 3 illustrates the non-dominated solutions (deployment models) derived from the PQP algorithm's execution. There are four ways for the designer to allocate tasks across various processors, offering a trade-off between system extensibility (slack capacity) and energy consumption. These four solutions provide the designer with valuable insights on how to implement the car system under optimal conditions. Importantly, these insights are presented without any a posteriori imposition of designer preferences. Indeed, the designer has the opportunity to analyze the various solutions and select from Table 3 the one that aligns best with his preferences and constraints related to the two objectives. For instance, the first deployment model enables maximum system extensibility, albeit at the expense of consuming the highest amount of energy. Nevertheless, the third model exhibits the least slack capacity in exchange for minimizing energy consumption. It prohibits any updates to the task model due to the overload on processor P_1. Moreover, extending the properties of the tasks would render the current deployment model infeasible. The second solution is also noteworthy as it maintains an acceptable extensibility level while having a very reasonable energy consumption value.

In multi-policy RL techniques like Pareto Q-learning, when multiple solutions are generated for a given problem, algorithm convergence signifies the conclusion of the agent's learning process, indicat-

Table 1: Task model description.

T_i	Pr_i	C_1	C_2	C_3
T_1	10	1	1.5	2
T_2	10	2	3	4
T_3	40	3.2	4.8	6.4
T_4	12	2	3	4
T_5	6	1	1.5	2

Table 2: Hardware model description.

P_j	$Capacitance$	$Voltage$ (V)	$Frequency$ (GHz)
P_1	1	2	2
P_2	1	4	3
P_3	1	6	4

Table 3: Deployment models description (The Pareto front).

P_j	P_1	P_2	P_3	***Slack-capacity***	***Energy***
$DeploymentModel_1$	$\{T_1, T_3, T_4\}$	$\{T_2\}$	$\{T_5\}$	0.34	349.6
$DeploymentModel_2$	$\{T_2, T_3, T_4\}$	$\{T_1, T_5\}$	$\{\}$	0.24	165.6
$DeploymentModel_3$	$\{T_1, T_2, T_3, T_4\}$	$\{T_5\}$	$\{\}$	0.14	119.6
$DeploymentModel_4$	$\{T_1, T_4, T_5\}$	$\{T_2, T_3\}$	$\{\}$	0.26	312.8

ing that the agent is no longer able to discover new or superior solutions compared to the current ones, regardless of the number of epochs used for training (Van Moffaert and Nowé, 2014). To investigate the convergence of the PQP algorithm, we conduct multiple PQP executions while varying the number of epochs (incrementing by 50 each time) and observe the generated solutions as illustrated in Table 4. The NB NG in this table represents the total number of solutions generated by the PQP, whereas the Nb NDS refers to the number of non-dominated ones. It is noteworthy that stability in solutions and convergence are observed from 250 epochs onward. The spike recorded at 150 epochs in the number of generated solutions (7 solutions) can be attributed to the agent's immaturity at this stage, where it is still exploring its environment through random task selection. Figure 3 depicts the learning progress of the agent. This learning progress is represented by the ratio of the number of non-dominated solutions (NB NDS) to the total number of generated solutions (NB GS) at each algorithm iteration. We can see from this figure that, after 250 epochs, PQP achieves convergence, yielding a set of deployment models that align with the Pareto front described in Table 3. From the data in Table 4 and the curve trend depicted in Figure 3, we observe that as agents converge towards Pareto-optimal solutions, the total number of generated placement models decreases. The decrease in overall count can be attributed to the dominance of non-dominated solutions during generation, leading to a reduction in the total number of solutions. This trend is particularly noticeable at epoch 150, where merely 7 solutions exist, and only one among them is Pareto-optimal. Conversely, at epoch 100, despite the agent's less advanced learning stage, there are still 5 solutions in total. This occurrence can be explained by the fact that the agent stumbles upon two non-dominated solutions randomly, resulting in the effective elimination of the majority of dominated ones.

Despite its significance, convergence alone is insufficient to determine the efficiency of the algorithm. Even if the algorithm converges and produces a set of feasible task placements, these placements may be dominated and far from the Pareto front solutions. To address this issue and ensure a faithful evaluation of PQP, we computed the Pareto front for this case

Table 4: PQP results with respect to the number of epochs.

Number of epochs	*Nb GS*	*Nb NDS*
50	8	1
75	6	1
100	5	2
125	5	2
150	7	1
175	5	2
200	4	3
225	4	3
250	4	4
275	4	4
300	4	4

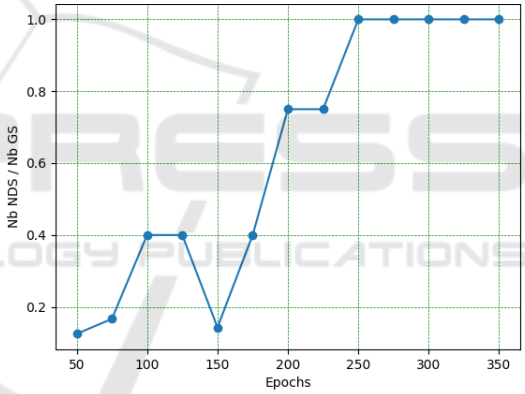

Figure 3: Process of agent learning.

study. Initially, we generated the complete set of potential task placements (finding 230 possible deployment models), then filtered out only the feasible ones (132 deployment models are feasible). Subsequently, we applied the non-dominated function to retain only the task placements that constitute the Pareto front (4 deployment models are non-dominated). Figure 4 compares the Pareto solutions produced by the PQP algorithm, which correspond to the deployment models described in Table 3, with the true Pareto solutions already computed for the considered case study. As we can see from the figure, the obtained solutions from the PQP execution (indicated by red diamonds) perfectly match the actual deployment models (represented by black marks), proving the efficiency of our method in producing accurate solutions. It's worth noting that determining the Pareto front is an NP-hard

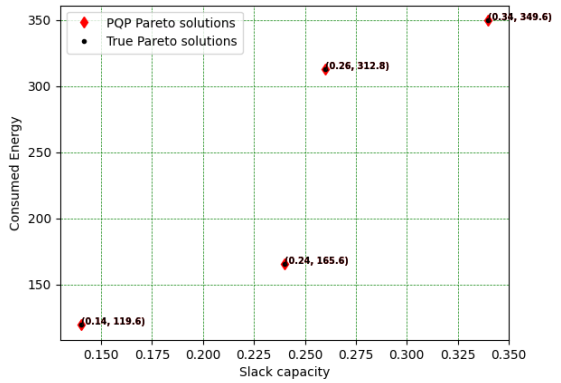

Figure 4: Non dominated solutions for PQP approach Vs true Pareto optimal solutions.

problem, and our success in computing it for this case study is attributed to the small number of tasks and processors considered.

8 EXPERIMENTAL RESULTS

We experiment PQP algorithm on a more extended general case (for simplicity reason we call it GC) that has been defined in (Lassoued and Mzid, 2022). In the study by (Lassoued and Mzid, 2022), a methodology (that we call GA based algorithm) employing genetic algorithms was presented to generate placement models that strike a balance between system extensibility and energy consumption. The eight black

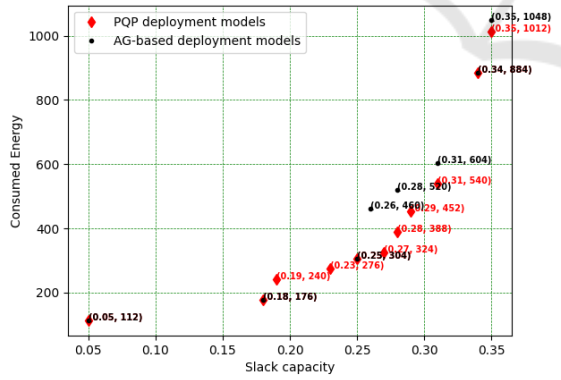

Figure 5: Non dominated solutions for PQP approach Vs GA based approach (Lassoued and Mzid, 2022).

markers shown in Figure 5 illustrate the results obtained from running their algorithm on GC. Similarly, in the same Figure 5, the eleven red diamonds correspond to the solutions derived from executing the PQP algorithm under the same conditions. As a first interpretation, we observe that the PQP algorithm produces a larger number of solutions compared to GA based algorithm. Then the two algorithms share four solutions (denoted by red diamond and a black mark): (0.05, 112), (0.18, 176), (0.25, 304), and (0.34, 884). However, the remaining four solutions from the other algorithm are dominated by additional solutions from PQP. Specifically, the solutions (0.26, 460), (0.28, 520), (0.31, 604), and (0.35, 1048) from the other algorithm are respectively dominated by the PQP solutions (0.29, 452), (0.29, 452), (0.31, 540), and (0.35, 1012). Additionally, PQP provides three solutions not present in the GA based algorithm's results: (0.19, 240), (0.23, 276), and (0.27, 324).

Delving deeper into the analysis, it would be advantageous to compare the results against the true Pareto front. However, due to the relatively complex system involved, this task is not as straightforward as outlined in section 7. In such circumstances, where the true Pareto front is unavailable, many studies (Zitzler and Thiele, 1998; Cao et al., 2015) have advocated for the use of the hypervolume as an effective measure to evaluate multi-objective solutions. The hypervolume, a scalar value, quantifies the area bounded by a reference point (i.e., a solution that is dominated by all generated solutions) and the non-dominated solutions discovered by an algorithm (see Section 3). What sets the hypervolume apart from other multi-objective metrics is its ability to simultaneously quantify both the extent and diversity of a set of solutions provided by an algorithm. Hence, The higher the hypervolume value, the larger space of dominated solutions covered by the non-dominated solutions. Figures 6 and 7 exhibit the hypervolumes

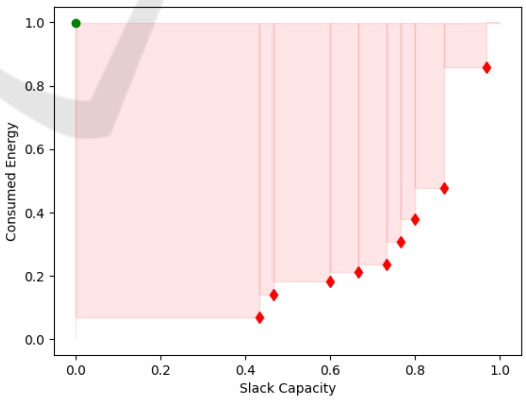

Figure 6: (Lassoued and Mzid, 2022) Hypervolume.

of the solution spaces for the GA-based algorithm and PQP, respectively. Figure 8 presents the juxtaposition of the two hypervolume figures under consideration to facilitate comparison. It is evident that PQP covers a larger extent compared to (Lassoued and Mzid, 2022). However, since visual interpretation alone may not suffice, we compute the hypervolume values of the

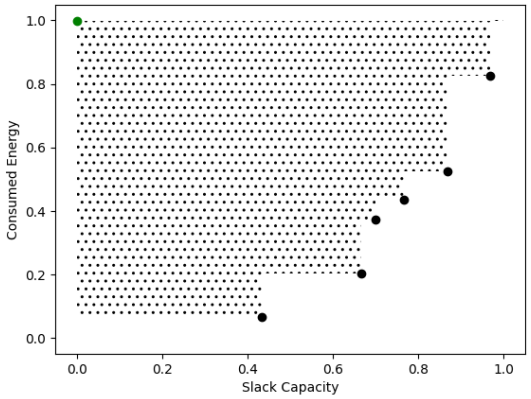

Figure 7: PQP algorithm Hypervolume.

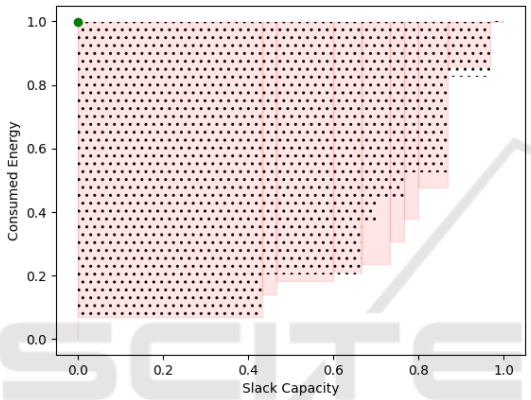

Figure 8: Hypervolume comparison.

two algorithms following the methodology outlined in (Van Moffaert and Nowé, 2014; Cao et al., 2015). The computed values are 0.714 for GA based algorithm and 0.734 for PQP, confirming the superior performance of the latter.

To more show the capabilities of PQP algorithm and the utility of maximising the extensibility (slack capacity) of the system, we maintain a set of new experiments where we tackle the refactoring issue in a multi-objective setting. For that we refer to the two random systems data used for experiments in (Haouari et al., 2022) and we run the refactoring process many times to compare PQP and GA based algorithm performances. To measure the time of refactoring we compute the $T_{generation}$ factor (defined in (Haouari et al., 2022)) which refers to the necessary time spent to take into consideration the system updates and to generate the new task placement models. Equation 7 precises the compute of $T_{generation}$:

$$T_{generation} = T_{initial} + rf * T_{refactoring} \quad (7)$$

Where $T_{initial}$ represents the duration required to supply the deployment models for the initial system version, $T_{refactoring}$ denotes the time necessary to generate new solutions for considering system updates, and rf is the number of times the designer updates the system properties. Figure 9 illustrates the $T_{generation}$

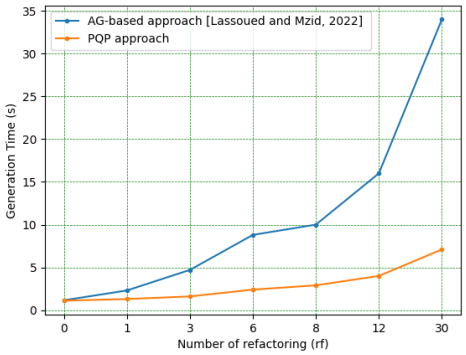

Figure 9: Evaluation of the generation time for randomly generated system.

evolution during the refactoring process for both PQP and AG based methods on the two random systems (e.i system1 and system2). The curves depicted in Figure 9 illustrate that AG-based algorithm requires more time to accommodate adjustments in both systems. Furthermore, this necessary time becomes increasingly substantial with the rise in refactoring frequency, as well as with the increase in tasks and processors within the system, as seen in system 2. These results can be explained by the fact that in AG based approach, a new system is established for every refactoring request, requiring the algorithm to be executed anew each time. In the PQP approach, system updates are handled differently. Specifically, to define the external structure (i.e., tasks and processor numbers) of the system, the PQP algorithm utilizes the Q-table. As in the refactoring process the aspect of the system is preserved, the construction of the Q-table is also maintained, thereby saving the time initially invested in this phase of the algorithm. Consequently, to address updates related to tasks properties, only the computation of Q-values needs to be recalculated, requiring a time of $T_{refactoring}$.

9 CONCLUSION

In this paper, we have presented a novel approach leveraging reinforcement learning to address the multi-objective optimization challenges inherent in task placement problems. The optimization objectives entail maximizing system extensibility while minimizing energy consumption. To achieve this, we introduce the PQP method, rooted in Pareto Q-learning and tailored for the task placement problem in real-time embedded systems. The proposed

method approximates the Pareto front for mapping tasks to heterogeneous processors. The Pareto front represents the optimal deployment models that strike a balance between the optimized objectives within real-time constraints. Through empirical evaluation, PQP demonstrates its efficacy compared to genetic algorithms while also providing a solution to the refactoring problem, enabling designers to efficiently explore system configurations and adjustments.

As future work, we aim to extend PQP's applicability to more diverse case studies, incorporating additional objectives and refactoring scenarios. Additionally, we plan to address the task scheduling process from a multi-objective perspective, aiming to minimize both worst-case response time and energy requirements simultaneously.

REFERENCES

Akesson, B., Nasri, M., Nelissen, G., Altmeyer, S., and Davis, R. I. (2020). An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 3–11. IEEE.

Barto, A. G. (2021). Reinforcement learning: An introduction by Richards' Sutton. *SIAM Rev*, 6(2):423.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

Cao, Y., Smucker, B. J., and Robinson, T. J. (2015). On using the hypervolume indicator to compare pareto fronts: Applications to multi-criteria optimal experimental design. *Journal of Statistical Planning and Inference*, 160:60–74.

Caviglione, L., Gaggero, M., Paolucci, M., and Ronco, R. (2021). Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. *Soft Computing*, 25(19):12569–12588.

Coello, C. A. C. (2007). *Evolutionary algorithms for solving multi-objective problems*. Springer.

Fonseca, C. M., Paquete, L., and López-Ibánez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *2006 IEEE international conference on evolutionary computation*, pages 1157–1163. IEEE.

Haouari, B., Mzid, R., and Mosbahi, O. (2022). On the use of reinforcement learning for real-time system design and refactoring. In *International Conference on Intelligent Systems Design and Applications*, pages 503–512. Springer.

Haouari, B., Mzid, R., and Mosbahi, O. (2023a). Psrl: A new method for real-time task placement and scheduling using reinforcement learning. In *Software Engineering and Knowledge Engineering*, pages 555–560. ksi research.

Haouari, B., Mzid, R., and Mosbahi, O. (2023b). A reinforcement learning-based approach for online optimal control of self-adaptive real-time systems. *Neural Computing and Applications*, 35(27):20375–20401.

Huseyinov, I. and Bayrakdar, A. (2022). Novel nsga-ii and spea2 algorithms for bi-objective inventory optimization. *Studies in Informatics and Control*, 31(3):31–42.

Kashani, M. H., Zarrabi, H., and Javadzadeh, G. (2017). A new metaheuristic approach to task assignment problem in distributed systems. In *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pages 0673–0677. IEEE.

Lakhdhar, W., Mzid, R., Khalgui, M., and Frey, G. (2018). A new approach for optimal implementation of multi-core reconfigurable real-time systems. In *ENASE*, pages 89–98.

Lakhdhar, W., Mzid, R., Khalgui, M., Frey, G., Li, Z., and Zhou, M. (2020). A guidance framework for synthesis of multi-core reconfigurable real-time systems. *Information Sciences*, 539:327–346.

Lassoued, R. and Mzid, R. (2022). A multi-objective evolution strategy for real-time task placement on heterogeneous processors. In *International Conference on Intelligent Systems Design and Applications*, pages 448–457. Springer.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Mehiaoui, A., Wozniak, E., Babau, J.-P., Tucci-Piergiovanni, S., and Mraidha, C. (2019). Optimizing the deployment of tree-shaped functional graphs of real-time system on distributed architectures. *Automated Software Engineering*, 26:1–57.

Mirjalili, S. (2019). Genetic algorithm. In *Evolutionary algorithms and neural networks*, pages 43–55.

Van Moffaert, K. and Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512.

Vidyarthi, D. P. and Tripathi, A. K. (2001). Maximizing reliability of distributed computing system with task allocation using simple genetic algorithm. *Journal of Systems Architecture*, 47(6):549–554.

Yang, L., Sun, Q., Zhang, N., and Liu, Z. (2020). Optimal energy operation strategy for we-energy of energy internet based on hybrid reinforcement learning with human-in-the-loop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(1):32–42.

Zhu, Q., Zeng, H., Zheng, W., Natale, M. D., and Sangiovanni-Vincentelli, A. (2013). Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4):1–30.

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer.