Towards A Robust Meta-Reinforcement Learning-Based Scheduling Framework for Time Critical Tasks in Cloud Environments

Hongyun Liu* Peng Chen*† Zhiming Zhao*
*Multiscale Networked Systems (MNS), University of Amsterdam, Amsterdam, the Netherlands
†School of Computer and Software Engineering, Xihua University, Chengdu, China
Email: {h.liu|p.chen2|z.zhao}@uva.nl

***Abstract*—Container clusters play an increasingly important role in cloud computing for processing dynamic computing tasks. The resource manager (i.e., orchestrater) of the cluster automates the scheduling of the dynamic requests, effectively manages the resources' utilization across distributing infrastructure resources. For many applications, the requests to the cluster are often with restricted deadlines. The scheduling of container clusters is often tricky, especially when the cluster's size is large and the load of the requests is dynamically changing. Machine learning-based approaches such as reinforcement learning have attracted lots of research attention during the past years; However, those approaches suffer from low robustness when the requests in an operational environment are changing and different from the training data sets. This paper investigates this problem by quantifying the robustness and proposing meta-gradient reinforcement learning to improve the robustness of classical reinforcement learning-based approaches. The proposed approach can lead to better deadline guarantees and faster adaptation for time-critical task scheduling under dynamic environments. We then empirically test the benefits of our method using both real-world and synthetic data sets. The evaluation results show that the proposed method outperforms the compared RL methods in scheduling performance and robustness.**

***Index Terms*—resource management, task scheduling, reinforcement learning, robustness, meta learning.**

I. Introduction

In cloud environments, container technologies provide lightweight OS-level virtualization solution which can effectively pack and deploy software components in remote infrastructures [1], [2]. The computing cluster for elastic container deployment and execution, e.g., Kubernetes [3], play a crucial role for not only automating software development and operation (DevOps) [4] but also in handling dynamic tasks driven by external events, e.g., sensors [5], or human interactions [6]. Container clusters have now become a standard service offered by most providers.

A container cluster's resources are often managed by a component called orchestrator, which schedules the dynamic container deployment requests based on the constraints such as resource demands of the request, available capacity in the cluster, and expected quality of service application. In many cases, requests with high-performance requirements, e.g. processing video [7] and critical time constraints, e.g. scaling services in cloud [8].

Empirical heuristics such as Shortest Job First(SJF) [9], and FIFO [10] are effective for small-scale clusters but not suitable for managing large infrastructures, specifically when the types of requests are diverse and changing. Advanced scheduling approaches based on multi-resource types [11], dependencies among requests [12], resource budgets [13], and deadlines of each requests [14] have attracted lots of research attention during past years. Based on specific models of the resources and requested tasks, those scheduling approaches often show significant success in handling a specific type of container requests but fail in large-scale handling clusters with diverse requirements for deployment. Since 2013, machine learning-based approaches have attracted lots of attention. Zhang et al. [15] [16], apply scheduling policies acquired through the former learning process; However, by revisiting empirical methods and machine learning approaches above, the goal always lies in achieving a scheduling model or set of policies for a fixed environment without taking the dynamics into account.

Reinforcement Learning (RL) is a typical example. In an RL-based solution, deployment requests, cluster resources are the action space. Scheduling decisions are based on the feedback of the decision made through the learning process, in which learning agents learn to make decisions through interacting with the environment built on action space [17]. Mao et al. reported the advances of their RL-based solutions [18]. However, in those works, the robustness of the solution is often a big obstacle for utilizing those solutions in a new operating environment where workloads of the requests are different from the training data [19].

To improve the robustness of scheduling, Yao et al. explicitly model the uncertainties in the task demands during the scheduling [20], and Singh et al. predict the future incoming workload and resource requirement based on time series information [21]. Guo et al. improve the robustness in the offloading strategy for computation with failure recovery (RoFFR) [22] when aiming to reduce energy consumption and shorten application completion time. Mireslami et al. tackle the robustness of dynamic user demands using a hybrid method to allocate cloud resources [23]. Most of those early works focus on the specific patterns of uncertainties but not the performance stability and the recovery under uncertainties.

Moreover, not all solutions tackle the time-critical aspects of the tasks, e.g., deadlines. We are motivated to investigate the robustness issues with a clear focus on the RL-based approach in the time-critical request scheduling in container clusters to tackle these issues.

In this paper, to address RL-based scheduling performance deviation and deadline guarantee failures to time-critical tasks under dynamic environment, enlightened by [24], we present a novel approach to improve the robustness of RL [25] by integrating Meta Learning [26] framework and demonstrate how the proposed method can handle time-critical task scheduling to new patterns of the tasks with just minor adaptation. Furthermore, the paper presents a Meta Learning-based framework integrated with RL, which improves scheduling performance, robustness, and quantitative robustness assessment metrics. The main contributions are as follows:

1) An improved time-critical task scheduling framework exploits meta-learning and RL to improve scheduling robustness.
2) A novel RL reward function and state representation achieves better deadline guarantee.
3) Fast adaptation and shorter training time to fit dynamic environments.

In the remainder of this paper, we will first review the existing RL-based solutions for scheduling time-critical container tasks in Section 2. Then, in Section 3, we present the detailed framework and algorithm of the proposed meta RL-based approach. Next, we evaluate the approach using both a real-world data-set from operational research infrastructure and synthetic data-sets in Section 4. Section 5 presents further discussions on the experimental results and potential future work. Finally, Section 6 summarizes the whole paper.

II. Problem Statement and Related Work

A. Time critical container deployment scheduling

In a container cluster, the orchestrator schedules the deployment requests based on specific policies and automates the resource allocation and deployment process, like shown in Fig. 1.

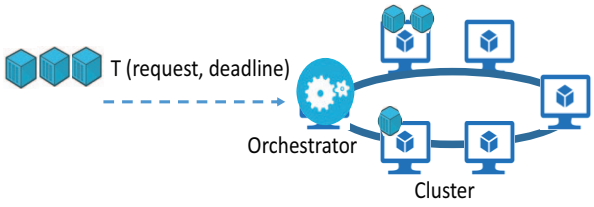

Fig. 1: Container scheduling processes.

A container cluster can be deployed on a set of virtual or physical networked machines. The deployment requests are generated by events, e.g., application controller or external data sources; those requests often come with resource demand, e.g., on the capacity of CPU or memory, and an optional deadline when needed. The orchestrator handles the incoming container deployment requests, allocates the available resources of the cluster, and makes decisions on the suitable actions, e.g., execute immediately, skip or discard the request. During this life-cycle, the orchestrator aims to continuously improve the scheduling quality to handle uncertainties in the coming requests and the stability of the resources.

B. Related work

During the past decades, task scheduling has been extensively studied in the context of resource management, e.g., cloud infrastructure services [27], IoT devices [28], and Edge computing [29]. Different scheduling solutions have been developed, e.g., using heuristics [30], or optimization methods such as genetic algorithm [31] [32], ant colony algorithm [33], and particle swarm algorithm [34] [34]. Those classical approaches often rely on explicit models on resources or workloads to design the scheduling policies and strategies and are often for a specific type of system. A scheduling solution must handle the complexity of highly customizable and dynamic cloud infrastructure services in their model in cloud environments, which requires profound optimization and often in low efficiency. Machine learning-based approaches have hence attracted lots of attention in the recent decade [15] [16] [35]. Machine learning approaches are tried to handle makespan of task flows [36], resource utilization rate [37], Quality of Service [38] and pricing models [39].

Different machine learning methods have been tried: supervised learning methods [40], unsupervised methods [41], and Reinforcement Learning [17]. Among those approaches, supervised learning-based approaches require well-curated labeled training data set, which is not always the case in many infrastructures. Moreover, the quality of the training data set directly influences the quality of the scheduling decision processes. RL-based approaches do not require such labeled data for training but rely on continuous feedback from the trial decisions to improve the scheduling decision processes. RL-

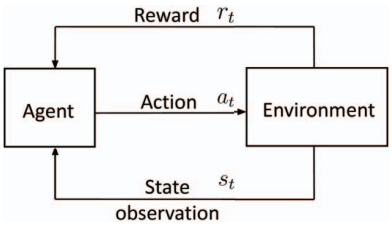

Fig. 2: A basic Reinforcement Learning model

based approaches have been used for different purposes, e.g., for scheduling resources in data centers [18], for workflow applications in container clusters [19] [42]. Those approaches provide a continuous decision-making process to flexibly handle the dynamic aspects of the cloud infrastructure during the scheduling process [17]. Figure 2 shows a typical RL framework [17], in which the *agent* acts as scheduler which applies scheduling policies; the *environment* where tasks are executed is represented as the state of the resources and tasks;

the *action* is the chosen scheduling policies for tasks. The reward operation gives feedback to the agent on the effect of the previous action.

The process of RL is straightforward [18] [36]; However, the performance, in terms of learning time and accuracy of the decision, highly depends on the design and configuration of the learning pipeline. When applying a mature model to a new environment, we can observe some performance variation in classical approaches. This issue is also called the robustness of the scheduling performance. These are some work addressed this issue: robustness of heuristic methods [43], measurements of scheduling robustness [44], graphical approach for improvement of robustness [45]. However, compared with the efficiency and performance of the scheduling algorithm itself, the problem of performance robustness in different environments has gained much less attention during past years.

C. Problem statement

The performance robustness of a scheduler also refers to the stability of its performance in a new environment, which may have different resource models or task workloads. For an RL-based scheduling solution, the high robustness of the performance is ideal.

The performance robustness can be assessed from two aspects:

1) The performance deviation right after the change of the environment (e.g., change of the resource model or the task workloads) can be measured as performance loss compared to the stable performance.
2) The recovery time spent on adaptation or retraining shows the ability of the algorithm to adapt to new environments or changes.

Among machine learning-based approaches, the *robustness* has been discussed mainly in the context of the transfer learning approach [46]. On the other hand, lots of existing researches focus on learning methods against adversarial attacks by using approaches such as Fast Gradient Sign Method (FGSM) [47], Projected Gradient Descent (PGD) [48], Carlini and Wagner(C&W) attack [49], and Adversarial patch attack [50]. In this context, robustness refers to learning quality against the simulated data sources, which is not the same as performance robustness.

In this paper, we specifically focus on the issue of performance robustness. The critical question is *how to improve the performance robustness of an RL-based scheduler for different types of time-critical task requests in a container cluster?* The solution should minimize the performance deviation and retrain time while achieving the best scheduling (with minimal deadline missing rate) for critical tasks. In the next section, we will discuss a solution based on the meta-gradient RL approach.

III. Robust Meta-RL for Time Critical Scheduling

As is shown in Figure 3, the whole process starts when a new task arrives cloud platform, it will be sorted into a task queue, depicted in the "State of Tasks" blank at the bottom of the figure, waiting for execution actions from the learning process in the center of the figure. Inside the learning process, depicted in the middle named "Discrete Markov Decision Process," each state has a scheduler learning policy model, whose structure is depicted in the red block on the left named "Deadline-Guarantee Task Scheduling"; the scheduler receives resource availability information and task queue information to calculate deadline critical reward function following Meta-Learning framework in parallel via N RL learning agents. After the learning process by RL agents, as depicted in the middle green part, the meta learner adapts to the gradients updates learned by RL agents to achieve the action for each state to execute a chosen task. The meta-learning process is depicted in the green block at the top of the figure, named "Meta Learning-Based Adaptation." Finally, the framework updates the system's policy model and related parts of the system: task queue and resource availability.

TABLE I: Notation of the system framework

Task properties:	
T_j	the task *j* ($j \in \{1, ..., n\}$)
T_j^{arr}	arrival time of task *j*
d_j	resource demand of task *j*
T_j^{len}	theoretical execution length of task*j*
$\sigma_{T_j}(t)$	arrival rate of task *j*
T_j^{fin}	execution finishing time of task *j*
$\mathrm{T}_j^{ex}(t)$	execution update of task *j* at time *t*
$\mathrm{Q}_{T_j}^{fu}(t)$	QoS fuse variable of task *j* at time *t*
$\mathrm{Q}_{T_j}(t)$	QoS variable of task *j* at time *t*
Resource availability Parameters:	
$\mathrm{M}^i(t)$	available amount of resource *i* at time *t*
$\mathrm{Re}_{ma}^i(t)$	upper bound of resource *i* at time *t*

A. RL-Based Deadline-Guarantee Scheduling for Time-critical Tasks

Firstly we present the deadline-guarantee scheduling based on Reinforcement Learning to fit our problem formulation. As a continuous decision making process, the scheduling process can be seen a finite discrete-time Markov Decision Process (MDP), where Reinforcement Learning is feasible to be formulated. More specifically, a learning process $\mathcal{M}$ can be denoted as a tuple $(\mathcal{S}, \mathcal{A}, \pi, \mathcal{R}, \gamma, H)$, where $\mathcal{S}$ represents state space, $\mathcal{A}$ represents action space, H represents the number of tasks to be calculated in each training iteration. The reward function $\mathcal{R}$ is defined as the sum of rewards $\sum_{t=0}^{H-1} r(s_t, a_t)$ from each trajectory $\tau := (s_0, a_0, ..., s_{H-1}, a_{H-1}, s_H)$, π denotes the policy $\mathcal{S} \times \mathcal{A} \xrightarrow{\pi} \mathbb{R}^+$, which is characterized by θ, defined as a probability distribution over actions: $\pi(s, a) \rightarrow [0, 1]$ and $\gamma \in (0, 1]$ denotes discount factor in cumulative rewards.

Time-Critical Task State Space $\mathcal{S}$: As is shown in Fig 4: the state space representation consists of resource availability, nominated tasks pool, waiting for the queue, and discarded queue. We convert the information of states into quantified two-dimensional units with different colors in a coordinate system. The example of state representation of one kind of

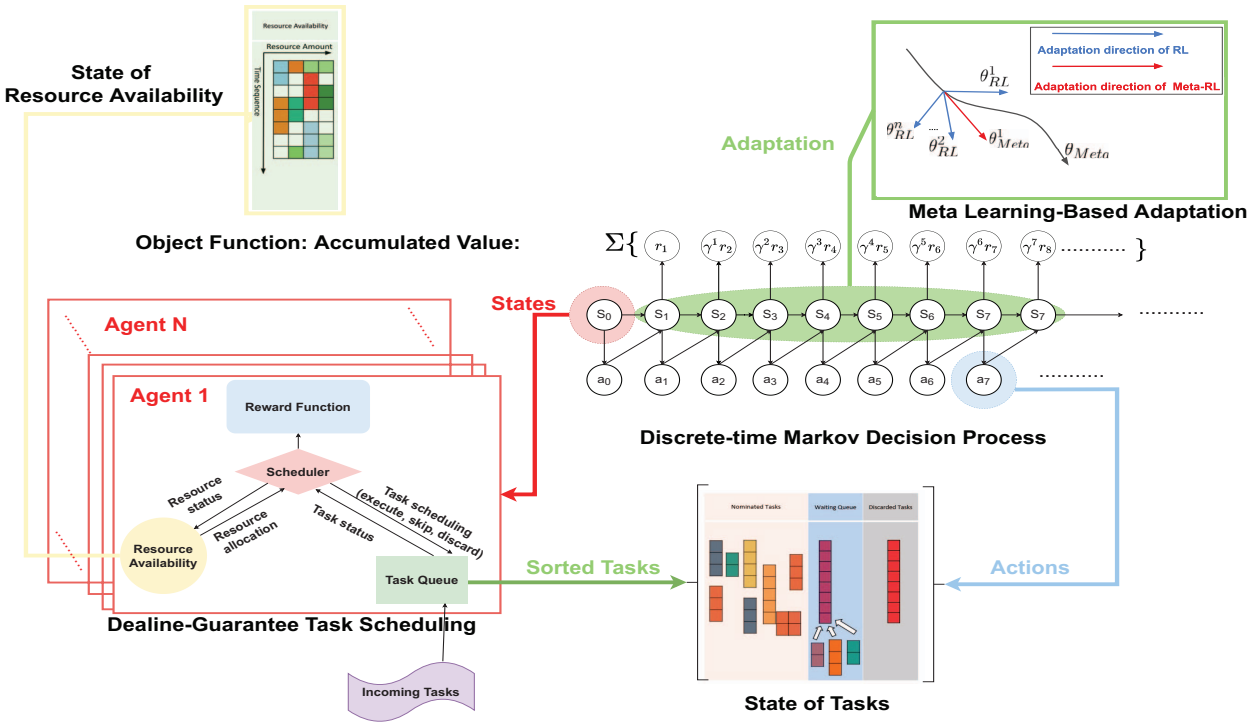

Fig. 3: The Meta-Gradient RL Scheduling Framework

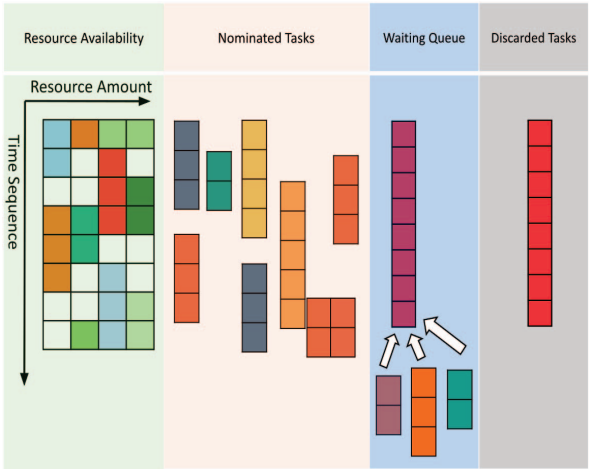

Fig. 4: State representation

resource is illustrated in Fig 4. In the representation, the colors of units in the coordinate system represent tasks. The length of units along the y axis represents time length, while the length x axis represents the number of resources. The left green area is the resource availability, where blank units represent available units, while units in different colors represent these resources being dominated by different tasks. The middle orange area is full of nominated tasks, which are H task samples chosen for the scheduler to pick. The right blue area is the waiting queue of tasks, including new tasks. Moreover, the waiting queue tasks are in proper order by its challenging execution slowdown variable, which helps the agent schedule more efficiently. The right gray part is the queue of tasks discarded by the scheduler. The details of each part are as follows:

Scheduling Action Space $\mathcal{A}$: Action space is the set of scheduling choices. During each time step, the scheduler calculates the choices of H nominated tasks to make the action decision. Thus each iteration H tasks (here we set it up to 50) get nominated from the waiting queue to be candidates of allocation. The decisions of allocation are made according to the policy model. The choices for each task include: executing, skipping, and discarding. After action gets taken, a new task will be nominated to the H tasks nominated queue. With the hard deadline scheme, the agent firstly checks $\mathcal{Q}_{T_j}^{fu}(t)$ of a task; if it is negative, the agent will discard this task. After each iteration of calculation, if the task T_j does not get execution action, its variable $\mathcal{Q}_{T_j}^{fu}(t)$ gets deducted by one. By so doing $\mathcal{Q}_{T_j}^{fu}(t)$ shows if a task missed deadline. There is a final check with the executive action to ensure current resource availability meets the selected task's request. If not, the scheduler will skip that task.

Calculation of Scheduling Policy π: The scheduling policy is trained by a two-layer fully-connected neural network [51]. The activation function is rectified linear unit (ReLU), *softmax*. The *input* to the NN is the state representation. The hidden layer has 30 neurons. The output of the first layer is the probability distribution of selecting each task of the nominated

queue in the number of H for execution; based on this probability distribution, the second layer outputs the selection of the chosen task to execute. Within this process, the action space decrease from 3^H choices to H.

After achieving scheduling policies π, the learning for the current environment is finished. However, there exist dynamics in the environment. Every time some part of the setup or environment changes, the model needs proper time to retrain, even subject to the change's influence. When it comes to scheduling problems for time-critical tasks, the time of retraining needs to be as short as possible. To this end, more general and robust learning is our ideal approach instead of conventional ones. As a learning approach to achieve a more general model, Meta-Learning is adopted to integrate with Reinforcement Learning.

Objective Function and The Gradient: The objective function of RL to optimize policy π, is formulated as maximization of the cumulative rewards:

$$\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\big[\sum_{t=0}^{\infty}\sum_{j=1}^{H}\gamma r(s_t,a_t)\big] \tag{1}$$

where (s_t, a_t) represents state and action among different samples (with size of H) respectively.

Then the gradient of this objective is given as follows:

$$\begin{aligned}&\nabla_\theta\mathbb{E}_{\in(s_t,a_t)\sim\pi_\theta}\big[\sum_{t=0}^{\infty}\sum_{j=1}^{H}\gamma r(s_t,a_t)\big]\\&=\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\Big[\nabla_\theta\log\pi_\theta(s_t,a_t)\mathcal{R}_{\pi_\theta}(s_t,a_t)\Big]\end{aligned} \tag{2}$$

Then the gradient descent update for policy parameters is as follows:

$$\theta' = \theta + \alpha\sum_{t=0}^{\infty}\nabla\log\pi_\theta(s_t,a_t)r(s_t,a_t) \tag{3}$$

where α is the step size, this equation is derived from a well-known RL equation from [17]. Based on the RL formulation, we integrate a deadline guarantee scheme.

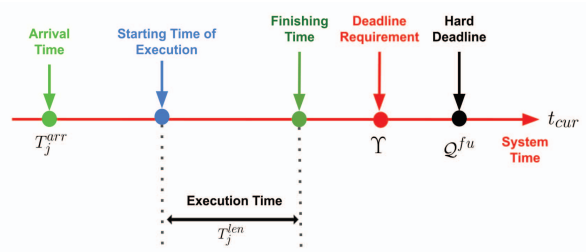

Fig. 5: Deadline Definition for Each Task: the execution slowdown value is the ratio between real execution time and ideal execution time

Deadline-Guarantee Reward Function $\mathcal{R}$: For the aim of deadline guarantee, we propose diverse reward functions characterized by valuables depicted in Figure 5: the axis in red depicts the system time t_{cur}; Each task is with its arrival time T_j^{arr} and execution time T_j^{len}, then we propose the execution slowdown [18] value of each task $\mathcal{Q}_{T_j}(t)$ as follows:

$$\mathcal{Q}_{T_j}(t) = \frac{t_{cur} - T_j^{arr}}{T_j^{len}} \tag{4}$$

Based on this, we formulate the function $f(\mathcal{Q}_{T_j}(t)$ to judge if execution slowdown of a task is within the deadline requirement:

$$f(\mathcal{Q}_{T_j}(t),\Upsilon) = \begin{cases}1, & \text{if } \mathcal{Q}_{T_j}(t)\le\Upsilon\\ 0, & \text{otherwise}\end{cases} \tag{5}$$

Then the objective of RL process is formulated as follows:

$$\max\sum_{T_j\sim\mathcal{D}}[f(\mathcal{Q}_{T_j}(t),\Upsilon)] \tag{6}$$

where $\mathcal{D}$ is the task sample data set, Υ denotes the deadline guarantee threshold, according to Service Level Agreement (SLA) made between cloud user and provider, the execution slowdown variable of each task shall be bounded. For *nominated* tasks:

$$r_{T_j\in(s_t,a_t)} = \begin{cases}\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a + P_a^{fuse},\\ \text{if to be discarded}\\ \max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a + B_o,\\ \text{if to be executed}\\ \max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a,\\ \text{if to be skipped}\end{cases} \tag{7}$$

where, P_a denotes a constant representing penalty, Υ is deadline requirement threshold as aforementioned, T_j^{arr} denotes task arrival time; For tasks chosen to be *discarded*, whose execution slowdown goes beyond hard deadline $\mathcal{Q}_{T_j}^{fu}$, their reward function includes execution slowdown penalty $\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a$ and extra penalty P_a^{fuse} *discard* action. For tasks chosen to be *executed*, whose demands meet the resource availability while execution slowdown does not violate hard deadline, their reward function includes execution slowdown penalty $\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a$ and a constant bonus B_o for successful allocating a task; For tasks to be *skipped*, whose demands does not meet resource availability while execution slowdown does not violate hard deadline, their reward function includes only execution slowdown penalty $\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a$. Overall, every time a task gets executed successfully, it will be removed from nominated queue while a new task will be added to nominated queue from waiting queue; The skipped tasks stay in the nominated queue; The discarded tasks go to the discarded queue.

For tasks in the *waiting* queue:

$$r_{T_j\in(s_t,a_t)} = \max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a\mathcal{Q}_{T_j}(t)T_j^{len} \tag{8}$$

Another penalty related to waiting time length is given for waiting tasks, forcing the learning process to allocate those tasks faster.

After we formulate a time-critical RL scheduling framework, we could find that the scheduling policy model it achieves is learned from the data set $\mathcal{D}$. What if $\mathcal{D}$ changes? What will happen to the scheduling performance? If there will be a performance deviation, how to decrease it? We will talk about these problems in the following section.

B. Robust Meta-RL scheduling paradigm

When it comes to dynamics in the data set, in other words, the learned model needs to expand its feasibility scope, to be more general and robust to the environment. Based on the model of cloud platform and Reinforcement Learning-based scheduling method built as aforementioned, in this section, we will propose our framework of the algorithm for achieving a robust scheduling model.

Robustness of Meta-Learning paradigm

Meta-learning aims to learn a more general model instead of learning a specific one. The robustness of Meta-Learning comes at the randomization of the training environment, which is thus the dynamics. The model trained by meta-learning can effectively exploit and adapt to changes brought incurred by dynamics faster than re-training the model from scratch. [24]. In a typical Meta-Learning setting, the task distribution Λ provides the training set and adaption set (new tasks). The training process is to learn a policy model, denoted as π_θ, characterized by θ. π_θ optimizes the objective function while minimizing learning loss denoted as $\mathcal{L}_\mathcal{D}$.

In this paper, we introduce the gradient-based meta-learning into RL, which updates the learning parameter in two steps:

1) Inner layer update: Doing training on the sample drawn for training $\mathcal{D}_{tr}$ from task distribution $\mathcal{D}$ to calculate the updated θ' according to the following update function:

$$\theta' = \Phi_\beta(\mathcal{D}_{tr}, \theta) \tag{9}$$

2) Outer layer update: Using the updated θ' to apply testing procedure among tasks $\mathcal{D}_{te}$, which is from the same data set with $\mathcal{D}_{tr}$, to update the parameter of the model when achieving minimal of loss, we will demonstrate the definition of the loss function in next section.

$$\min_{\theta,\beta} \mathbb{E}_\mathcal{D}[\mathcal{L}(\mathcal{D}_{te}, \theta')] \tag{10}$$

For the inner layer update, we adopt the gradient descent method to update θ as follows:

$$\Phi_\beta(\mathcal{D}_{tr}, \theta) = \theta - \alpha \nabla \mathcal{L}(\mathcal{D}_{tr}, \theta) \tag{11}$$

Repeatedly, according to the convergence, after specific times mutation of environment [26], a general model is achieved. When a learned model encounters a new data set or a new environment, it just needs to adapt itself by few times learning new features.

Moreover, the inner layer learns a specific scheduling model for a specific data set from a cloud log period. As is known, all kinds of dynamics exist in resource availability and task demands. The data trajectories consequently change dynamically, where the outer layer is working on learning across different data trajectories to achieve more features improving robustness. As to the inner layer, the learning goal is to learn the scheduling model, which acts as a scheduler in the system interacting with task model and cloud platform models. Therefore, the inner layer's learning approach has decent interactive ability while learning for this role. Reinforcement Learning is ideal for this mission among different learning approaches as its structure fits the problem set up in this work. The details of the RL approach's feasibility and its design will be introduced in the following subsection.

Meta-Gradient Reinforcement Learning Formulation

In this section, we propose the inner layer RL designed for scheduling model learning. According to the problem formulation and models built as aforementioned, the reward We aim to use a gradient-based meta-learning framework to upgrade policy learned via RL to be more robust to the environment's dynamics. Those uncertainties influence scheduling performance and the deadline missing rate.

The RL takes the role of the inner loop of meta-learning to learn the update from trajectories of data set samples; thus, we have a more specific version of the equation: 9

$$\theta' = \Phi_\beta(\mathcal{D}_{tr}, \theta) = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\mathcal{A}_t, \mathcal{S}_t \sim \pi(\theta)}[\sum_{t=0}^{N} \gamma^t \mathcal{R}_t] \tag{12}$$

where, $\mathcal{D}_{tr}$ is data set sampled from Λ for training; $\mathcal{S}_t, \mathcal{A}_t$ represents state and action among different samples (with size of N sampled points) respectively.

Then for the meta testing part, the learned parameter θ' will be used to calculate the loss function and do the gradient-based update:

$$\theta \leftarrow \theta - \alpha \sum_{j=1}^{N} \mathcal{L}_j(\mathcal{D}_{te}, \theta') \tag{13}$$

C. Robust Meta-RL Scheduling algorithm with Deadline-Guarantee

In this section, we describe the overall algorithm, merging all components mentioned in previous sections. As is shown in algorithm 1, distribution over tasks, learning rates of the outer and inner loop are required. The first step is the initialization of the algorithm: setting the initial parameters of the policy model and reset the data set $\mathcal{D}$. We also set a sliding learning window with a size of h to improve the algorithm to be a continuous learning process. From line 2 to line 4 is the sampling step: based on the number of environments, N data trajectories are sampled from the distribution Λ according to the current policy model and added to the data set $\mathcal{D}$. The following inner layer learning loop from lines 5 to 6 re parallel RL learning agent, sampling data sets τ_H inside $\mathcal{D}$. From lines 7 to 9, each learning agent samples a smaller sliding learning window with a size of h to calculate updated θ'_h based on each loss function. Unlike conventional RL or other learning methods, the overall policy model does not get updated by any parallel inner layer learning agent. After achieving updated θ'_h, each RL agent uses θ'_{h_i} model to sample new data samples τ'_{h_i} from $\mathcal{D}$. In line 14, each agent finishes a sliding window learning and

Algorithm 1 Robust Meta-RL Scheduling algorithm with Deadline-Guarantee

Require: Distribution over tasks: Λ,
Require: Environment number: N, sliding window size h
Require: Learning rate: $\alpha, \beta \sim \mathbb{R}^+$
1: Initialize the policy π_θ and $\mathcal{D} \leftarrow \varnothing$
Require: Number of environments: N
2: **for** $i = 1, ...N$ **do**
3: Use pre-adapted policies $\pi_{\theta'_H}$ to sample $\mathcal{D}' \sim \Lambda$
4: Add samples: $\mathcal{D} \leftarrow \mathcal{D}'$
5: **for** $j = 1, ..., H$ **do**
6: Set a sliding door with the size $h_i \in (0, H)$
7: **for** h_i **do**
8: Use policies π_θ to sample trajectories
9: within first h_i samples $\tau_{h_i} \sim H$
10: Use τ_H to calculate adapted parameters:
11: $\theta'_{h_i} = \theta + \alpha \sum_{j=1}^{h_i} \nabla \log \pi_\theta(s_t, a_t) r_{T_j}(s_t, a_t)$
12: Use adapted policy $\pi_{\theta'_{h_i}}$ sample trajectories:
13: $\tau'_{h_i} \sim H$
14: **end for**
15: Use τ_H to calculate adapted parameters:
16: $\theta'_H = \theta + \alpha \sum_{h_i} \nabla \log \pi_{\theta_{h_i}}(s_t, a_t) r_{T_j}(s_t, a_t)$
17: Use adapted policy θ'_H sample trajectories $\tau'_H \sim \mathcal{D}$
18: **end for**
19: Calculate update:
20: $\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{H} \sum_{j=1}^{H} \mathcal{L}_j(\theta'_H)$ using τ'_H
21: **end for**
22: **return**θ updating policy model

then continues to the next sliding window. The learned τ'_{h_i} are used to calculate θ'_H and update τ'_H. From lines, 19 to 20 is the calculation of overall parameter update for scheduling policy model based on the gradient of θ'_H learned by RL agent from each sliding window within each environment. After each iteration, the sliding window continues to sample another data set then the whole algorithm stays continuous learning process.

IV. Experiment

In this section, we conduct a serial of experiments to evaluate our approach while comparing it with several other methods to prove that our approach achieves state-of-the-art performance.

A. Experimental Settings

Real-world data-sets

Euro-Argo Data Service log: The Euro-Argo research infrastructure is the European contribution to the global Argo program, which currently has more than 3500 autonomous float instruments globally deployed over the world ocean to measure and report temperature salinity and other properties of the oceans. The collected raw data from deployed floats is processed into scientific research data, critical assets for conducting environmental and interdisciplinary scientific research. They are then made available via the Euro-Argo data portal, and research communities can access them from various methods. To guarantee this, the Euro-Argo infrastructure needs to allocate sufficient resources to store data, execution of service requests, and bandwidth for down and uploads. The Euro-Argo Data Service Log data is collected for one month's continuous data services 24 hours per day. There are 14 variables from different data services measured for one month. There are 43200 samples collected by sampling every 1 minute from the 4094157 raw log data, including request numbers, requested transfer time, and requested transfer size.

Synthetic Data

For synthetic data-set in each environment, we assume two types of resources, i.e., CPU cores and memory, both with a total capacity of m. Tasks are further classified into two categories: light and heavy tasks. The duration of light tasks is uniformly chosen between $1t_0$ and $3t_0$, while the duration of heavy tasks is chosen uniformly from $10t_0$ to $15t_0$, t_0 is a one-time step in the system. Each task has a dominant resource which is picked randomly. The demand for the dominant resource is selected uniformly between $0.25m$ and $0.5m$, and the demand for the other resource is between $0.05m$ and $0.1m$. Finally, we set up the deadline threshold $\Upsilon = 3$ and the hard execution slowdown variable to 5.

General environment set up				
	Parameters			
Modes	light tasks	heavy tasks	dominant resource	execution slowdown
General	$1t_0 - 3t_0$	$10t_0 - 15t_0$	$0.25m - 0.5m$	$3x$

TABLE II: General Setup: We assume two types of resources, i.e., CPU cores and memory, both with a total capacity of m. Tasks are further classified into two categories: small and big tasks. The duration of small tasks is uniformly chosen between $1t_0$ and $3t_0$, while the duration of big tasks is chosen uniformly from $10t_0$ to $15t_0$, t_0 is a one-time step in the system, the execution slowdown is set to three times of each task length.

Evaluation Metrics

TABLE III: Table of Indicators

Deadline Guarantee Indicators		
DRMR	$\triangleq$	Deadline Requirement Missing Rate
HDMR	$\triangleq$	Hard Deadline Missing Rate
NAI	$\triangleq$	Necessary Adaptation Iterations
CSP	$\triangleq$	Converged Scheduling Performance
CW10000	$\triangleq$	Convergence within 1000 times of iterations
Robustness Indicators		
SPD	$\triangleq$	Scheduling Performance Deviation
AIDURP	$\triangleq$	Adaptation Iteration and Data Usage for Performance Recovery

Deadline Requirement Missing Rate(DRMR) [52]: total percentage of scheduled tasks that violate deadline requirement. DRVP indicates the level of deadline guarantees for each method.

Hard Deadline Missing Rate(HDMR) [52]: total percentage of scheduled tasks that violate hard deadlines. HDVP shows

several tasks which violate strict limitation of deadline requirement.

Necessary Adaptation Iterations(NAI): the iteration needed for convergence after the influence of dynamics.

Converged Scheduling Performance(CSP): the scheduling performance after the convergence of retraining, which describes under newly converged scheduling policies, the portion of scheduled tasks that meet the deadline requirement:

$$CSP = 1 - [\mathcal{Q}_{conv} - \Upsilon]/\Upsilon \tag{14}$$

Convergence within 1000 times of iterations(CW10000): wither learning converged within 10000 iterations of adaptation in a new environment (yes/no).

Robustness Indicators We adopt the two indicators to compare robustness performance: Scheduling Performance Deviation(SPD) and Adaptation Iteration and Data Usage for Performance Recovery(AIDUPR). More specifically, the performance values PER_{after} and PER_{before} are defined as the average execution slowdown value $\mathcal{Q}_{T_j}(t)$ of scheduled tasks before and after environment changes, respectively. Requests and which are according to resource requirements are adjusted after algorithm convergence to check the deviation of algorithms and then the recovery and adaptation process of each approach. Thus the instant performance deviation right after the influence of dynamic from a workload or resource availability is one of the criteria, which is formulated as follows:

$$SPD = \frac{|PER_{after} - PER_{before}|}{PER_{before}} \tag{15}$$

where, PER_{after} denotes the instant average execution slowdown value after the influence of dynamic, PER_{before} indicates previous converged execution slowdown value .

Besides the instant performance deviation, algorithm convergence speed also reflects the robustness, which shows how fast the algorithm adapts to dynamics. AIDUPR is proposed to describe the time needed for iterations and data needed for training of adaptation after performance deviation incurred by dynamics:

$$AIDUPR = SPD * ITER * t \tag{16}$$

where $ITER$ demonstrates the iteration time, t describes time spent for each iteration.

Baseline Settings To compare Meta-RL and conventional RL methods, we pre-train four different RL-based scheduling methods with the same input data as Meta-RL. The structure and parameters are shown in Table IV. The pre-training details follow in the next part.

Platform settings: We implement the experiments on the hardware platform consists of 18 nodes, each node has: 4 x GTX 1080 Ti, 2 x Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz (12 cores per cpu), 128 GB memory, 2 x 10 TB local HDD, 2 x 4 TB local SSD. The software environment includes: Anaconda, python-numpy, python-scipy, python-dev, python-pip, python-nose, g++ libopenblas-dev, git, Theano 1.4 version, Lasagne 0.1 version, and python-matplotlib.

Fine-tuned RL Approaches						
Parameters / RL Approaches	NN Layers	Neuron Number	Optimizer	ρ	Learning Rate	Activation Function
RL1	2	20	*RMSProp*	0.9	0.00001	ReLU, Softmax
RL2	3	30	*RMSProp*	0.8	0.00001	ReLU, Softmax
RL3	3	40	*Adam*	–	0.00001	ReLU, Softmax
RL4	3	50	*RMSProp*	0.7	0.00001	ReLU, Softmax

TABLE IV: Fine-tuned RL approaches: we train four different RL approached as baselines of our proposed Meta-RL. Those four fine-tuned are with different neuron network structures: different neural network layers, different number of neurons for each layer, and different update rules combinations

Before making an adaptation comparison with RL, we apply our approach, Meta-RL, to obtain a generic policy going through 30 sets of environments trajectories, each one has $4 \times 4 \times 4$ kinds of combinations: σ_{T_j}, d_j, T_j^{max} with respect of general set up. New trajectories are fed to the learning process only after the algorithm reach convergence among the last trajectories. Then apply the learned model to new environment training. Thus prior meta-learning includes 1920 continuous environments data sets training. For comparison

Workload Setups			
	Parameters		
Workload modes	σ_{T_j}	d_j	T_j^{max}
Light	(0.1,0.3)	$< 0.3m$	$< 5t_0$
Medium	(0.4,0.5)	$< 0.5m$	$< 10t_0$
Heavy	(0.5,1)	$< 0.8m$	$< 15t_0$

TABLE V: Workload Setups: We assume two types of resources, i.e., CPU cores and memory, both with a total capacity of m, t_0 is a one-time step in the system.

with RL, we set up three kinds of environments with respect to workload:*Easy workload:* with $\sigma_{T_j} \in (0.1, 0.3)$, $d_j < 0.3m$, $T_j^{max} < 5t_0$. *Medium workload:* with $\sigma_{T_j} \in (0.4, 0.5)$, $d_j < 0.5m$, $T_j^{max} < 10t_0$. *Heavy workload:* with $\sigma_{T_j} \in (0.5, 1)$, $d_j < 0.8m$, $T_j^{max} < 15t_0$. Each set we sampled $3 \times 3 \times 3$ environments to do compression between Meta-RL and normal reinforcement learning, and other heuristics approach.

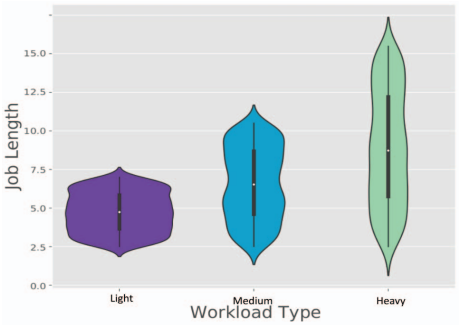

Fig. 6: Distribution of different workload.

B. Experimental Results

As is shown in Table VI, we compare the scheduling performance of our Meta-RL algorithm with four fine-tuned RL methods based on the synthetic data. We change the same portion of the workload for each method in the same workload

Workload	Light		Medium		Heavy	
Indicators / Approaches	DRMR	HDMR	DRMR	HDMR	DRMR	HDMR
Meta-RL	**1.5±0.5%**	**1.2±0.7%**	**2.2±0.6%**	**2.3±0.4%**	**5.2±1.1%**	**3.5±0.9%**
RL1	$10.1 \pm 2.4\%$	$5.5 \pm 3.6\%$	$12.2 \pm 5.3\%$	$8.2 \pm 3.3\%$	$22.9 \pm 6.1\%$	$11.4 \pm 5.2\%$
RL2	$15.2 \pm 4.3\%$	$7.3 \pm 3.1\%$	$21.7 \pm 10.3\%$	$12.3 \pm 5.5\%$	$33.2 \pm 7.9\%$	$17.4 \pm 11.2\%$
RL3	$13.3 \pm 3.1\%$	$10.5 \pm 3.1\%$	$14.7 \pm 5.5\%$	$10.6 \pm 4.6\%$	$15.3 \pm 2.5\%$	$12.5 \pm 3.3\%$
RL4	$13.5 \pm 5.3\%$	$8.2 \pm 2.1\%$	$14.1 \pm 6.4\%$	$7.4 \pm 3.3\%$	$16.3 \pm 5.2\%$	$10.2 \pm 4.3\%$

TABLE VI: Deadline Guarantee Comparison: we compare Meta-RL with four fine-tuned RL approaches in deadline guarantee based on synthetic data sets, which shows Meta-RL outperform the others in deadline missing rate, offering better deadline guarantee

Indicators / Workload Modes / Approaches	Light			Medium			Heavy		
	NAI	CSP	CW10000	NAI	CSP	CW10000	NAI	CSP	CW10000
Meta-RL	**1210±300**	**96.13±3.24%**	✓	**2677±657**	**95.22±2.82%**	✓	**4366±1211**	**91.23±5.15%**	✓
RL1	10423 ± 2145	$91.45 \pm 5.21\%$	X	22754 ± 1203	$75.31 \pm 11.46\%$	X	8422 ± 2102	$67.62 \pm 14.71\%$	X
RL2	13473 ± 1133	$85.36 \pm 5.17\%$	X	17621 ± 3125	$76.65.45 \pm 4.81\%$	X	10125 ± 1014	$82.45 \pm 6.33\%$	X
RL3	6623 ± 765	$92.12 \pm 3.36\%$	X	12500 ± 1423	$88.9 \pm 7.53\%$	X	15032 ± 600	$77.8 \pm 5.76\%$	X
RL4	5477 ± 675	$87.25 \pm 7.56\%$	✓	9644 ± 1325	$73.23 \pm 9.77\%$	–	9642 ± 1742	$85.67.45 \pm 5.21\%$	–

TABLE VII: Scheduling Adaptation Iteration: we compare Meta-RL with four fine-tuned RL approaches based on data sets from real-world Argo log data, which shows Meta-RL outperform the others in performance stability and adaptation speed through different workload modes

setup to get the average results of two deadline missing rates to compare the proposed Meta-RL deadline guarantee performance. As can be seen, all numbers in bold are the ones with the best performance; Meta-RL stably offers the deadline guarantee in each environment, outperforming the other four fine-tuned RL methods in terms of the deadline guarantee. Only $3 \pm 1.5\%$ tasks scheduled violate hard execution slowdown variable 5. In contrast, other tasks scheduled by RL methods have at least 10% missing rate of deadline requirement. Moreover, for heavy tasks, shown in Table VI, our method Meta-RL offers deadline guarantees among under different workloads. As is shown in Table VII, we implement adaptation comparison between Meta-RL and other four fine-tuned RL approaches based on real-world Argo log dataset. Meta-RL converges with the least iteration times and with the best-converged scheduling performance. Thus Meta-RL outperforms the other four fine-tuned RL in performance stability and adaptation speed.

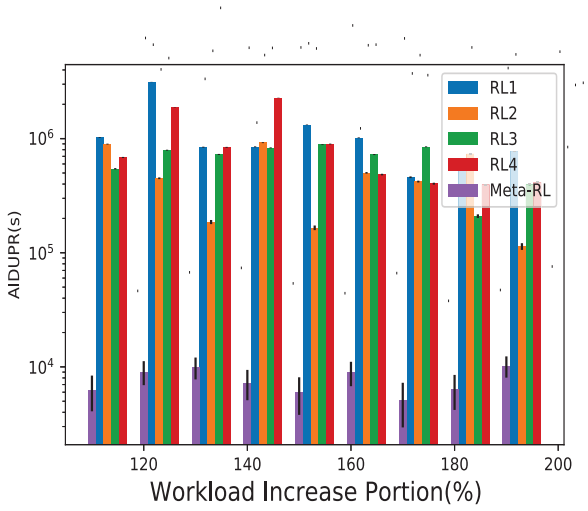

Fig. 8: Adaptation Speed Comparison

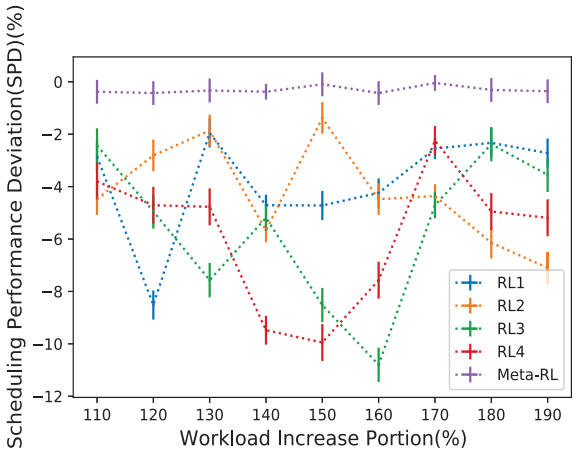

Fig. 7: Performance deviation Comparison

The performance deviation calculated here is right after the change of the environments; it could demonstrate the robustness of different approaches. The task requests are from the log data, with increasing resource demands. As shown in Figure 7: with the increasing workload, the performance deviation of Meta-RL remains stable within 50%, for some range even under 20%. In comparison, RL approaches' performance decreases starting from more than 50% even beyond 250% with the increased workload. From this, the robustness of our Meta-RL outperforms the conventional RL approach. Figure 7 shows the adaptation speed or performance recovery speed discounted by the performance deviation portion, which balances the adaptation speed and robustness performance. As is shown, we cloud see that the adaptation speed of Meta-RL is more than five times faster than RL averagely after every time increase of workload, at some point, even more, proving its robustness to dynamics of the environment.

V. Discussion

As shown in the previous section, our proposed approach outperforms the conventional Reinforcement Learning approach at the robustness and converging speed after changing workloads. Our approach's scheduling deviation and adaptation speed change in a similar trend during the increase of workload. Firstly they both increase within the range of 10%-30% workload increase then decrease within range of 30%-50% which shows then increase again. For other fine-tuned RL methods, their scheduling stability and adaptation speed do not show accordingly changing trends. Thus when given a 50% or minor workload increase, our approach could still keep robustness without lower than 30% performance deviation. The robustness starts to decrease when workload increase beyond 50% but still with lower than 50% performance deviation, much lower than fine-tuned RL methods (more than 200%). We are currently working on expanding the robustness range where our framework's framework could keep robustness with lower performance deviation to improve the overall scheduling robustness further.

The performance deviation of our Meta-RL approach can be minimized within -30% to -50%, which is much smaller than -200% to -1000% observed from conventional RL approaches. However, there is still room for further improvement, particularly for reducing the deviation right after the dynamic workload changes. Furthermore, by adding a sliding learning window, our framework stays continuous online learning to adapt to dynamics in the environment. We are currently profiling performance changes in different workload patterns, considering changes in workload types, failures among cloud infrastructures, or pricing models used. Combing those studies with well-refined online learning strategies will be an important future work.

Above all, further improving the robustness of container cluster scheduling will be the goal in future work.

VI. Conclusion

In this work, Meta-RL, a robust task scheduling framework, is presented to offer deadline-guaranteed scheduling for time-critical tasks and improve the schedule's robustness in the meantime. We propose a meta-gradient robust reinforcement learning framework to quickly adapt a scheduling policy model to a newly changed environment while using a deadline critical scheme to maintain the deadline guarantee. Experimental results show that our approach can provide the deadline guarantee, which outperforms fine-tuned RL methods. Furthermore, our Meta-RL approach finishes adaptation in new environments using fewer training iterations, 200%-500% faster than the fine-tuned RL approach, achieving better robustness while offering deadline guarantees.

Acknowledgment

This work has been partially funded by the European Union's Horizon 2020 research and innovation program by the ARTICONF project grant agreement No 825134, by the ENVRI-FAIR project grant agreement No 824068, by the BLUECLOUD project grant agreement No 862409, by the LifeWatch ERIC, by China Scholarship Council, Science and Technology Program of Sichuan Province under Grant No.2020YFG0326, and Talent Program of Xihua University under Grant No.Z202047.

References

[1] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "Concurrent container scheduling on heterogeneous clusters with multi-resource constraints," *Future Generation Computer Systems*, vol. 102, pp. 562–573, Jan. 2020.

[2] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "ECSched: Efficient Container Scheduling on Heterogeneous Clusters," in *Euro-Par 2018: Parallel Processing* (M. Aldinucci, L. Padovani, and M. Torquati, eds.), vol. 11014, pp. 365–377, Cham: Springer International Publishing, 2018. Series Title: Lecture Notes in Computer Science.

[3] T. Goethals, F. DeTurck, and B. Volckaert, "Extending kubernetes clusters to low-resource edge devices using virtual kubelets," *IEEE Transactions on Cloud Computing*, 2020.

[4] H. Zhou, Y. Hu, X. Ouyang, J. Su, S. Koulouzis, C. Laat, and Z. Zhao, "CloudsStorm: A framework for seamlessly programming and controlling virtual infrastructure functions during the DevOps lifecycle of cloud applications," *Software: Practice and Experience*, vol. 49, pp. 1421–1447, Oct. 2019.

[5] I. L. Santos, L. Pirmez, F. C. Delicato, S. U. Khan, and A. Y. Zomaya, "Olympus: The cloud of sensors," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 48–56, 2015.

[6] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[7] W. Chen and D. McDuff, "Deepmag: Source-specific change magnification using gradient ascent," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 1, pp. 1–14, 2020.

[8] J. Wei, X. Chen, J. Wang, X. Hu, and J. Ma, "Enabling (end-to-end) encrypted cloud emails with practical forward secrecy," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[9] J. Ru and J. Keung, "An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems," in *2013 22nd Australian Software Engineering Conference*, pp. 78–87, IEEE, 2013.

[10] L. George and P. Minet, "A fifo worst case analysis for a hard real-time distributed problem with consistency constraints," in *Proceedings of 17th International Conference on Distributed Computing Systems*, pp. 441–448, IEEE, 1997.

[11] C. Wang, S. Zhang, Z. Qian, M. Xiao, J. Wu, B. Ye, and S. Lu, "Joint server assignment and resource management for edge-based mar system," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2378–2391, 2020.

[12] X. Tang, "Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems," *IEEE Transactions on Cloud Computing*, 2021.

[13] S. Huang, X. Huang, and N. Ansari, "Budget-aware video crowdsourcing at the cloud-enhanced mobile edge," *IEEE Transactions on Network and Service Management*, 2021.

[14] L. Niu and D. Zhu, "Fixed-priority scheduling for reliable and energy-aware (m, k)-deadlines enforcement with standby-sparing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[15] Y. Zhang, J. Yao, and H. Guan, "Intelligent cloud resource management with deep reinforcement learning," *IEEE Cloud Computing*, vol. 4, no. 6, pp. 60–69, 2017.

[16] W. Zhang and T. G. Dietterich, "Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling," *Journal of Artificial Intelligence Reseach*, vol. 1, pp. 1–38, 2000.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, pp. 50–56, 2016.

[19] Y. Hu, C. de Laat, and Z. Zhao, "Learning workflow scheduling on multi-resource clusters," in *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–8, IEEE, 2019.

[20] J. Yao, Q. Lu, H.-A. Jacobsen, and H. Guan, "Robust multi-resource allocation with demand uncertainties in cloud scheduler," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 34–43, IEEE, 2017.

[21] P. Singh, A. Kaur, P. Gupta, S. S. Gill, and K. Jyoti, "Rhas: robust hybrid auto-scaling for web applications in cloud computing," *Cluster Computing*, pp. 1–21, 2020.

[22] S. Guo, M. Chen, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Transactions on Mobile Computing*, 2020.

[23] S. Mireslami, L. Rakai, M. Wang, and B. H. Far, "Dynamic cloud resource allocation considering demand uncertainty," *IEEE Transactions on Cloud Computing*, 2019.

[24] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[25] R. B. Slaoui, W. R. Clements, J. N. Foerster, and S. Toth, "Robust domain randomization for reinforcement learning," 2019.

[26] D. Li, Y. Yang, Y.-Z. Song, and T. Hospedales, "Learning to generalize: Meta-learning for domain generalization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[27] A. Karthick, E. Ramaraj, and R. G. Subramanian, "An efficient multi queue job scheduling for cloud computing," in *2014 World Congress on Computing and Communication Technologies*, pp. 164–166, IEEE, 2014.

[28] T. Qiu, K. Zheng, M. Han, C. P. Chen, and M. Xu, "A data-emergency-aware scheduling scheme for internet of things in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2042–2051, 2017.

[29] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4225–4234, 2019.

[30] A. Spachis and J. King, "Job-shop scheduling heuristics with local neighbourhood search," *International Journal of Production Research*, vol. 17, no. 6, pp. 507–526, 1979.

[31] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *ieee transactions on cybernetics*, 2020.

[32] Z.-G. Chen, K.-J. Du, Z.-H. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 708–714, IEEE, 2015.

[33] X. Lu and Z. Gu, "A load-adapative cloud resource scheduling model based on ant colony algorithm," in *2011 IEEE international conference on cloud computing and intelligence systems*, pp. 296–300, IEEE, 2011.

[34] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *2010 24th IEEE international conference on advanced information networking and applications*, pp. 400–407, IEEE, 2010.

[35] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pp. 1–6, IEEE, 2017.

[36] S. Sahoo, B. Sahoo, and A. K. Turuk, "A learning automata-based scheduling for deadline sensitive task in the cloud," *IEEE Transactions on Services Computing*, 2019.

[37] A. Asghari, M. K. Sohrabi, and F. Yaghmaee, "Online scheduling of dependent tasks of cloud's workflows to enhance resource utilization and reduce the makespan using multiple reinforcement learning-based agents," *Soft Computing*, vol. 24, no. 21, pp. 16177–16199, 2020.

[38] X. Zuo, G. Zhang, and W. Tan, "Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 2, pp. 564–573, 2013.

[39] X. Zhang, C. Wu, Z. Huang, and Z. Li, "Occupation-oblivious pricing of cloud jobs via online learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 2456–2464, IEEE, 2018.

[40] M. Wajahat, "Costefficient dynamic management of cloud resources through supervised learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 47, no. 3, pp. 28–30, 2020.

[41] W. Iqbal, M. N. Dailey, and D. Carrera, "Unsupervised learning of dynamic resource provisioning policies for cloud-hosted multitier web applications," *IEEE Systems Journal*, vol. 10, no. 4, pp. 1435–1446, 2015.

[42] Y. Hu, J. Wang, H. Zhou, P. Martin, A. Taal, C. de Laat, and Z. Zhao, "Deadline-Aware Deployment for Time Critical Applications in Clouds," in *Euro-Par 2017: Parallel Processing* (F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, eds.), vol. 10417, pp. 345–357, Cham: Springer International Publishing, 2017. Series Title: Lecture Notes in Computer Science.

[43] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics," in *Grid Computing*, pp. 73–84, Springer, 2008.

[44] S. Goren and I. Sabuncuoglu, "Robustness and stability measures for scheduling: single-machine environment," *IIE Transactions*, vol. 40, no. 1, pp. 66–83, 2008.

[45] F. Ghezail, H. Pierreval, and S. Hajri-Gabouj, "Analysis of robustness in proactive scheduling: A graphical approach," *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 193–198, 2010.

[46] W. Kuang, L. Brown, and Z. Wang, "Transfer learning-based co-run scheduling for heterogeneous datacenters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[47] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[48] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[49] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, IEEE, 2017.

[50] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," *arXiv preprint arXiv:1712.09665*, 2017.

[51] T. M. Moerland, J. Broekens, and C. M. Jonker, "A framework for reinforcement learning and planning," *arXiv preprint arXiv:2006.15009*, 2020.

[52] R. Khorsand, F. Safi-Esfahani, N. Nematbakhsh, and M. Mohsenzade, "Atsds: adaptive two-stage deadline-constrained workflow scheduling considering run-time circumstances in cloud computing environments," *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2430–2455, 2017.