Robustness challenges in Reinforcement Learning based time-critical cloud resource scheduling

A Meta-Learning based solution

Liu, H.; Chen, P.; Ouyang, X.; Gao, H.; Yan, B.; Grosso, P.; Zhao, Z.

DOI
10.1016/j.future.2023.03.029

Publication date
2023

Document Version
Final published version

Published in
Future Generation Computer Systems

License
CC BY

Future Generation Computer Systems 146 (2023) 18–33

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Robustness challenges in Reinforcement Learning based time-critical cloud resource scheduling: A Meta-Learning based solution

Hongyun Liu [a,b,*], Peng Chen [c,**], Xue Ouyang [d], Hui Gao [e], Bing Yan [f], Paola Grosso [a,**], Zhiming Zhao [a,**]

[a] Informatics Institute, University of Amsterdam, Amsterdam, 1098 XH, The Netherlands
[b] Graduate School of Informatics, University of Amsterdam, Amsterdam, 1098 XH, The Netherlands
[c] School of Computer and Software Engineering, Xihua University, Chengdu, 610039, China
[d] School of Computer Science, National University of Defense Technology, Changsha, 410073, China
[e] College of Electrical and Control Engineering, Shaanxi University of Science and Technology, Xi'an, 710021, China
[f] School of Electrical and Electronic Engineering, University of Adelaide, Adelaide, South Australia, 5005, Australia

ARTICLE INFO

Article history:
Received 10 October 2022
Received in revised form 9 February 2023
Accepted 21 March 2023
Available online 13 April 2023

Keywords:
Robustness
Reinforcement Learning
Meta Learning
Resource management
Task scheduling
Cloud computing

ABSTRACT

Cloud computing attracts increasing attention in processing dynamic computing tasks and automating the software development and operation pipeline. In many cases, the computing tasks have strict deadlines. The cloud resource manager (e.g., orchestrator) effectively manages the resources and provides tasks Quality of Service (QoS). Cloud task scheduling is tricky due to the dynamic nature of task workload and resource availability. Reinforcement Learning (RL) has attracted lots of research attention in scheduling. However, those RL-based approaches suffer from low scheduling performance robustness when the task workload and resource availability change, particularly when handling time-critical tasks. This paper focuses on both challenges of robustness and deadline guarantee among such RL, specifically Deep RL (DRL)-based scheduling approaches. We quantify the robustness measurements as the retraining time and investigate how to improve both robustness and deadline guarantee of DRL-based scheduling. We propose MLR-TC-DRLS, a practical, robust Meta Deep Reinforcement Learning-based scheduling solution to provide time-critical tasks deadline guarantee and fast adaptation under highly dynamic situations. We comprehensively evaluate MLR-TC-DRLS performance against RL-based and RL advanced variants-based scheduling approaches using real-world and synthetic data. The evaluations validate that our proposed approach improves the scheduling performance robustness of typical DRL variants scheduling approaches with 97%–98.5% deadline guarantees and 200%–500% faster adaptation.

1. Introduction

Cloud computing attracts increasing attention in scheduling and processing dynamic and complex computational tasks [1,2], elastic service scaling [3,4] and automating the software development and operation pipeline [5,6]. A cloud environment can be customized based on application requirements using different resources, e.g., Virtual Machines (VM), docker containers, storage, and network. To automate the provisioning of cloud infrastructure and deployment of software services, a resources manager has to handle dynamic application requests with quality of service (QoS), e.g., video processing [1,7], time-critical tasks, scaling services in cloud [8,9]. At an abstract level, the management of the resources is often seen as a scheduling problem, e.g., allocating and scheduling the placement of VMs in data centers [10,11], deploying dynamic docker containers on a docker cluster [12,13], handling events in a big data framework [14,15]. However, such problems are complex and NP-hard; the request patterns from diverse users are highly dynamic, and resource availability constraints vary.

Time-critical tasks, for example, disaster forecast, often have very diverse time requirements in the context of execution, including data communication, processing, and calculation [9,16,17]. When scheduling time-critical tasks, Infrastructure as a Service (IaaS) is a common approach to optimizing the allocation of VMs (virtual machines), improving the network topology, and optimizing the execution process on the virtual infrastructure [18,19]. To this end, scheduling optimization, as an NP-hard problem, is proposed to investigate in this field. Furthermore, with the

* Corresponding author at: Informatics Institute, University of Amsterdam, Amsterdam, 1098 XH, The Netherlands.
** Corresponding authors.
E-mail addresses: h.liu@uva.nl (H. Liu), chenpeng@mail.xhu.edu.cn (P. Chen), ouyangxue08@nudt.edu.cn (X. Ouyang), gaohui@sust.edu.cn (H. Gao), bing.yan@adelaide.edu.au (B. Yan), p.grosso@uva.nl (P. Grosso), z.zhao@uva.nl (Z. Zhao).

https://doi.org/10.1016/j.future.2023.03.029
0167-739X/

development of cloud platforms, the scheduling methods have evolved from heuristics and artificial intelligence methods to machine learning-based methods.

Many works have been done on scheduling optimization by investigating Machine Learning-based approaches. A common solution for the scheduling optimization problem is to retrain the model to adapt to the dynamics. For small-scale clusters empirical heuristics [20,21] are feasible. With the growing capability of infrastructures, especially the requests and resources are becoming diverse and dynamic, empirical heuristics are not ideal for applying anymore. To this end, more scheduling approaches address issues from perspectives: multi-resource [22,23], task dependencies [24,25], resource budgets [26], and task deadlines [27]. Regarding different models combining calculations resources and related tasks, the effectiveness of those approaches is mostly counting on a specific kind of computational task. However, it has no scheduling performance guarantee for scalable clusters with diverse tasks and resources. Then during the past decades, Reinforcement Learning as its interactive learning property has been widely applied and investigated in the context of scheduling. Zhang et al. [28–30] optimize the scheduling process by using a policy model learned beforehand by RL algorithms. Many different RL variants-based algorithms are also dealing with scheduling problems: DQN [31] applies a deep Q learning algorithm to optimize scheduling problems by calculating the Q value of each scheduling action. Double-DQN [32] algorithm applies extra network to calculate optimization target. CEM [33] applying cross-entropy training methods to improve the optimization efficiency.

However, the dynamics of the scheduling environment have rarely been addressed in previous work, including empirical methods and Machine Learning-based approaches. The uncertainty of the dynamics leads to the scheduling performance deviation of the learned model, which is not beneficial for scheduling. An increase in tasks' deadline missing rate is one of the severe consequences. Intuitively, it is a standard solution to do retraining to compensate for this part of performance loss. However, retraining time [34] could lead to degradation of QoS for time-critical tasks and many consequences. Retraining time spent on adaptation to a new environment shows the ability of scheduling approaches to adapt to dynamics. We adopt this ability as the robustness of scheduling. The less retraining time it spends in a new environment, the better robustness it has. Scheduling robustness in this paper is described as the retraining time spent adapting to the new environments. Namely, the less retraining time means the better robustness. If the quantified changes among environments with the same changes are given, less retraining time means better robustness. So for better robustness, one of the aims is to reduce retraining time; we aim to investigate more details of methods coping with reducing retraining time and how to adjust it according to the requirements and the performance deviation.

There are also many works aiming to improve scheduling performance: Yao et al. explicitly model the task uncertainties through the scheduling process [35], and Singh et al. optimize scheduling by predicting workload, and resource availability [36]. Guo et al. improve the robustness of offloading by proposing failure recovery (RoFFR) [37], reducing energy consumption and application completion time. Mireslami et al. focus on a specific resource allocation scenario, where their proposed hybrid method [38] could handle the dynamics. Among those early works, the issues addressed are specific patterns of the uncertainties, not the scheduling performance stability or the adaptation speed after influences of dynamics. Moreover, the other missing part of the goal regarding scheduling is the time-critical factor, i.e., deadlines. To this end, the motivation of this work is the improvement of scheduling performance robustness and time-critical task scheduling.

The absence of robustness lies in cloud task scheduling optimization processes, their definition and evaluation process. There are rare systematic, comprehensive definitions and evaluations for scheduling performance robustness. Little attention and work have been dedicated to retraining time reduction and robustness improvement without the theoretical basis. However, as aforementioned, with the running of a cloud cluster platform, dynamics can appear at any part of it: from requests, orchestrator, and cluster. These dynamics directly influence the scheduling performance of the cloud cluster system. Consequently, the QoS it offers is not guaranteed anymore, which negatively influences cloud service providers.

Meta-learning [39], with its focus on adaptation against dynamics, takes dynamics into account, acquiring a more general model, which could help keep a compared more stable scheduling model and also help to adapt to newly changed environments.

Thus, in this work, inspired by [39], we investigate improving the robustness of the RL-based scheduling approach [40] by optimizing and integrating Meta Learning [41] framework. In the meantime, we optimize the reward system of the RL-based scheduling approach to guarantee deadlines for time-critical task scheduling. The main contributions of the paper include:

1. *Improving the robustness of Reinforcement Learning-based task scheduling* by optimizing and integrating the Meta-Learning framework. We also propose measurements for the robustness of RL-based scheduling.
2. *Improving Reinforcement Learning-based scheduling* with support for time-critical tasks by integrating an optimized deadline-guaranteed reward system.
3. *Optimizing and integrating the robustness and time critical guarantee into our proposed Meta-Learning-based robust Deep Reinforcement Learning scheduling (MLR-TC-DRLS) algorithm*, which integrates both optimized Meta-Learning-based scheduler and our proposed deadline guaranteed RL scheduler. In addition, we implement a comprehensive evaluation to compare our proposed MLR-TC-DRLS with state-of-the-art RL variants-based scheduling approaches.

In the remainder of this paper, we will first formulate the problem and go through state-of-the-art scheduling research in Section 2. Then, we elaborate our methodology step by step and formulate the complete algorithm MLR-TC-DRLS in Section 3. Next, in Section 4, we evaluate MLR-TC-DRLS using a real-world data set obtained from operational research infrastructure platform log data and a synthetic scheduling data set. Then, in Section 5 we have further analysis and discussion regarding the evaluation results. The possible future work also follows in this section. Finally, Section 6 concludes the whole paper.

2. Problem formulation and related work

2.1. Robustness problem

Firstly, we introduce a typical cloud computing system as shown in Fig. 1, the orchestrator schedules the deployment computational requests or tasks, following specific policies, automating the following deployment process.

The deployment environment of a cloud platform consists of many calculation resources, which are virtualized infrastructures or networked machines. The deployment requests or tasks are triggered by user events, e.g., remote file storage or data analysis; those requests or tasks are with specific calculation resource requirements, e.g., CPU or memory usage, and execution deadlines if those are time-critical tasks. The orchestrator manages

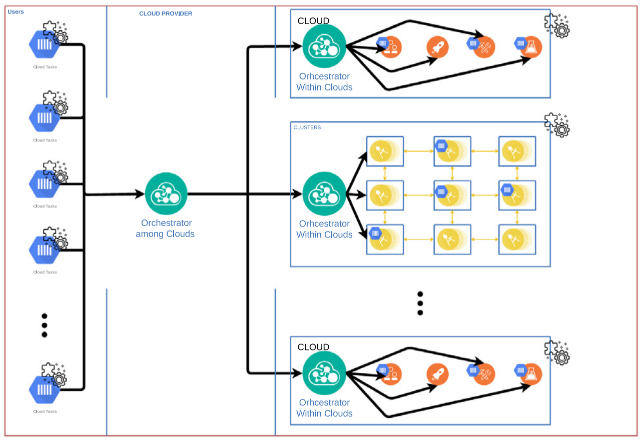

Users
CLOUD PROVIDER
Cloud Tasks
Cloud Tasks
Cloud Tasks
Cloud Tasks
Cloud Tasks
Orchestrator among Clouds
CLOUD
Orhcestrator Within Clouds
CLUSTERS
Orhcestrator Within Clouds
CLOUD
Orhcestrator Within Clouds

Fig. 1. Cloud scheduling processes.

Table 1
Notation list.

T_j	Number j task ($j \in \{1, \dots, n\}$)
T_j^{arr}	The time when task j arrives the platform
d_j	Required calculation resource from task j
t_{cur}	Current system time
T_j^{len}	Ideal time length spent on execution of task j
$\sigma_{T_j}(t)$	Arrival probability of task j
T_j^{fin}	The time when the execution of task j finishes
$\mathcal{Q}_{T_j}^{fu}(t)$	The hard deadline value of task j at time t
$\mathcal{Q}_{T_j}(t)$	The execution slowdown value of task j at time t
Υ	Expected deadline value
r_{T_j}	Reward value of task j at time t
P_a	Scheduling penalty constant (reward value calculation)
B_o	Scheduling bonus constant (reward value calculation)
P_a^{fuse}	Extra penalty constant (reward value calculation)

the incoming deployment tasks, making decisions on allocating available resources to execute tasks, e.g., to execute, hold or abandon the task. During this scheduling process, the orchestrator's objective is to continuously schedule tasks efficiently, meeting the requirements given by the requests. However, with the system running, dynamics brought by each component can appear at any time, inducing but not limited to: requests pattern changes, scheduling algorithm switching, hardware failures or unexpected performance deviation, as are shown in Fig. 1. For the orchestrator part, there are some scenarios where scheduling algorithms got switched due to requirements when the scheduler experiences performance deviation subject to the dynamics. As to the cluster resources part, hardware failures, maintenance, and many other factors could influence the resource availability, which incurred the scheduling performance deviation. Consequently, the cloud platform experiences uncertain performance deviation, which might violate LSA (Legal Service Agreement). Therefore, we need a more robust scheduling approach working against dynamics in all those scenarios to cope with this undesired working mode (see Table 1).

Many RL-based scheduling approaches optimize the scheduling process by learning a policy model to perform scheduling missions after training reaches the convergence requirements. In other words, the training process and the eventual model are all based on the source of the training data, namely the environment shown in Fig. 1, and the convergence requirements. However, these two parts of scheduling can change from time to time, as we aforementioned. Consequently, the model does not fit the new environment or training data anymore. To this end, retraining is necessary to adapt the model to a newly changed environment. However, little attention has been paid to reducing the retraining time. Consequently, most of the retraining is performing the training from scratch again, which leads to a long time of retraining. Time-consuming retraining is not always acceptable, especially for time-critical task scheduling. This work addresses the importance of retraining time reduction and the methods to improve scheduling robustness.

The robustness of scheduling algorithm performance we refer to in this work describes how stable the algorithm can remain against the dynamics of the environment. More specifically, with the inevitable change in the task workload or resource availability, the less scheduling performance deviation after the influence, the better robustness the algorithm has. For an RL-based scheduling approach, the higher robustness of the scheduling performance is the better it is for platform maintenance and QoS. We first need a clear definition of scheduling performance robustness. Then based on the definition, can we propose improvement solutions.

We define the assessment methods of scheduling performance robustness from following aspects:

1. The scheduling performance deviation between stable scheduling mode and instant scheduling performance after the dynamic influences from the environment (e.g., the resource availability dynamics or the task workloads change). The performance deviation can be represented as a ratio in positive or negative numbers depending on whether the performance decreases or increases compared with the previous performance.
2. The scheduling performance adaptation time, which describes time spent to converge again in a newly changed environment or retraining time recovering to the performance before dynamics' effects. Besides stability, we also want this adaptive ability to improve the overall stability of the scheduling process.

2.2. Deadline missing problem for time-critical tasks

When scheduling time-critical tasks, cloud platforms often use Infrastructure as a Service (IaaS) to optimize overall system-level performance by following methods: selecting the most suitable VMs (virtual machines), customizing their network topology, and optimizing the scheduling of execution on the virtual infrastructure [18,19]. Furthermore, the scheduler must be aware of time constraints, e.g., deadlines, as shown in Fig. 2, required for acceptable system performance. Scheduled executions, which fail to finish within specific deadlines, directly influence the deadline missing rate of tasks, furthermore, QoS, and serial consequences. However, current cloud providers lack explicit support for deploying time-critical applications where users need to manually deploy their applications step by step and have no guarantee regarding execution deadlines. In this paper, we address the deadline missing issue by integrating a deadline guarantee scheme into the scheduling approach to improving the deadline guarantee of task scheduling. In this work we define the deadlines in a more detailed manner, from the time when the tasks arrive in the platform to the final execution finishing time. With the more detailed deadline definition, we can better differentiate different phases of task execution process and schedule the tasks in a more precise way to improve the scheduling performance and robustness. Moreover, we have corresponding optimization targets with the more detailed deadline definition. We will elaborate the detailed optimization in Section 3.

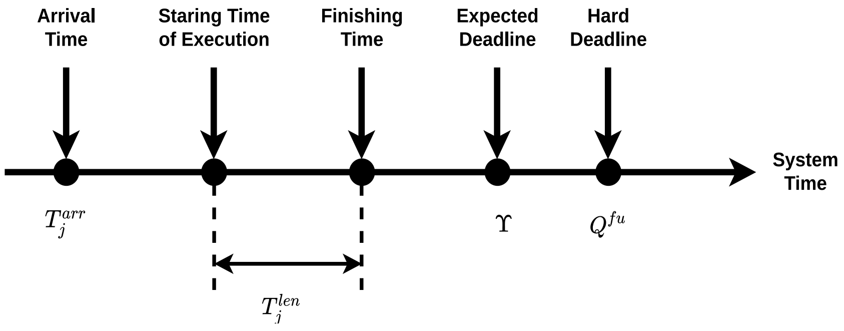


Fig. 2. Deadlines definition for each task.

2.3. Related work

For the past ten to twenty years, task scheduling for cloud computing platforms has been a popular topic, attracting many researchers to investigate. More specifically, relevant works include scheduling among cloud infrastructure services [42], Internet of Thing (IoT) [43], and scheduling of Edge platform [44]. Different strategies of optimizing scheduling have been investigated, e.g., empirical heuristics methods [45], artificial intelligence-based approaches such as genetic-based scheduling [46,47], ant colony-based scheduling [48], and particle swarm-based scheduling [49]. Those scheduling approaches are often designed based on particular resource and workload patterns, namely certain scheduling platforms or environment models. An ideal scheduling policy also needs to manage the dynamics among the cloud infrastructure services in cloud environments, which requires profound pruning and optimization. There are also some work [50] address the scheduling from a robust perspective and proposed a Canonical Particle Swarm Optimization (CPSO) based algorithm to solve the problem of resource allocation and management from the perspective of considering different policies for upcoming tasks, in both homogeneous and heterogeneous IoT Cloud Computing.

Another direction of investigating scheduling related problems is to formulate scheduling into decision-making process (MDP) then apply machine learning to do optimization [28–30], where the optimization objectives include handling make-span of workflows [51], optimizing resource utilization rate [52], Quality of Service (QoS) optimization [53] and pricing models optimization [54]. As is known, supervised learning-based methods ask for accurately labeled training data, limiting their feasibility in many scenarios. Consequently, the quality of the data labeling for training determines the learned scheduling policies. Different from reliance on labeled data, RL-based approaches [55] learn the scheduling policy through continuous interaction with the environment, including make-span optimization for data centers [56], workflow applications scheduling optimization [57]. A well-pruned scheduling model experiences a certain extent of performance deviation when applied to cloud platforms, depending on the dynamics. We have started to pay more attention to scheduling performance robustness issues. There has been effort involved in investigating this issue: robust heuristic approach [58], measuring scheduling robustness [59], improving robustness with graphical methods [60]. For work researching machine learning-based scheduling methods, there has also been much work done focusing on robustness lately. Transfer learning has been tried [61] in terms of scheduling stability. Also, adversarial attacks have been taken into account for scheduling to design more robust algorithms, which include Fast Gradient Sign Method (FGSM) [62], Projected Gradient Descent (PGD) [63], Carlini and Wagner (C&W) attack [64], and adversarial patch attack [65]. However, robustness issues addressed here are mostly about learning efficiency, where the dynamics in the environment have rarely been taken into account.

Among many RL-based approaches, DQN [66], Double-DQN [32], CEM [33], are representatives of them. DQN applies Q-network to calculate the Q value of different policies to optimize them. Double-DQN applies an extra neural network to improve the estimation of Q values. CEM adjusts methods of selecting different data entropy to improve optimization efficiency. Many works related to these approaches have been done: the DQN approach easily overestimates the Q values; Double-DQN corrects the overestimation but needs more calculation; CEM improves optimization by changing sampling methods, which requires more calculation and also brings more dynamics. Above all, most of the works contribute to scheduling efficiency but rarely to the robustness of scheduling performance, that is, retraining time, which influences the stability and generality of the scheduling a lot. We adopt three typical RL variants as comparisons in this work. The first, Deep Q Learning (DQN), is a popular approach to training machine learning models, aiming to optimize each action by calculating the Q value of each action. The second double DQN is a variant of DQN with an extraneural network to optimize the Q value calculation. Finally, we also consider CEM to be the approach to retaining time reduction.

DQN [67] aims to learn the estimation value of each action, defined as the expected sum of future rewards when taking that action and following the optimal policy after that. For example, under a given policy π, the actual value of action a in a state s is:

$$Q_\pi(s,a) \equiv \mathbb{E}[R_1 + \gamma R_2 + \cdots \mid S_0 = s, A_0 = a, \pi] \tag{1}$$

where $\gamma \in [0,1]$ is a discount factor that trades off the importance of immediate and later rewards. The optimal value is then $Q_*(s,a) = max_\pi Q_\pi(s,a)$. Estimates for the optimal action values can be learned using Q-learning, a form of temporal difference learning. The standard Q-learning update for the parameters after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is then

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \tag{2}$$

where α is a scalar step size and target Y_t^Q is defined as

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t) \tag{3}$$

This update resembles stochastic gradient descent, updating the current value $Q(S_t, A_t; \theta_t)$ towards target value Y_t^Q.

To prevent overoptimistic value estimates, double Q-learning [32], learns two value functions by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights, θ, and θ'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can untangle the selection and evaluation in Q-learning and rewrite its target (3) as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{4}$$

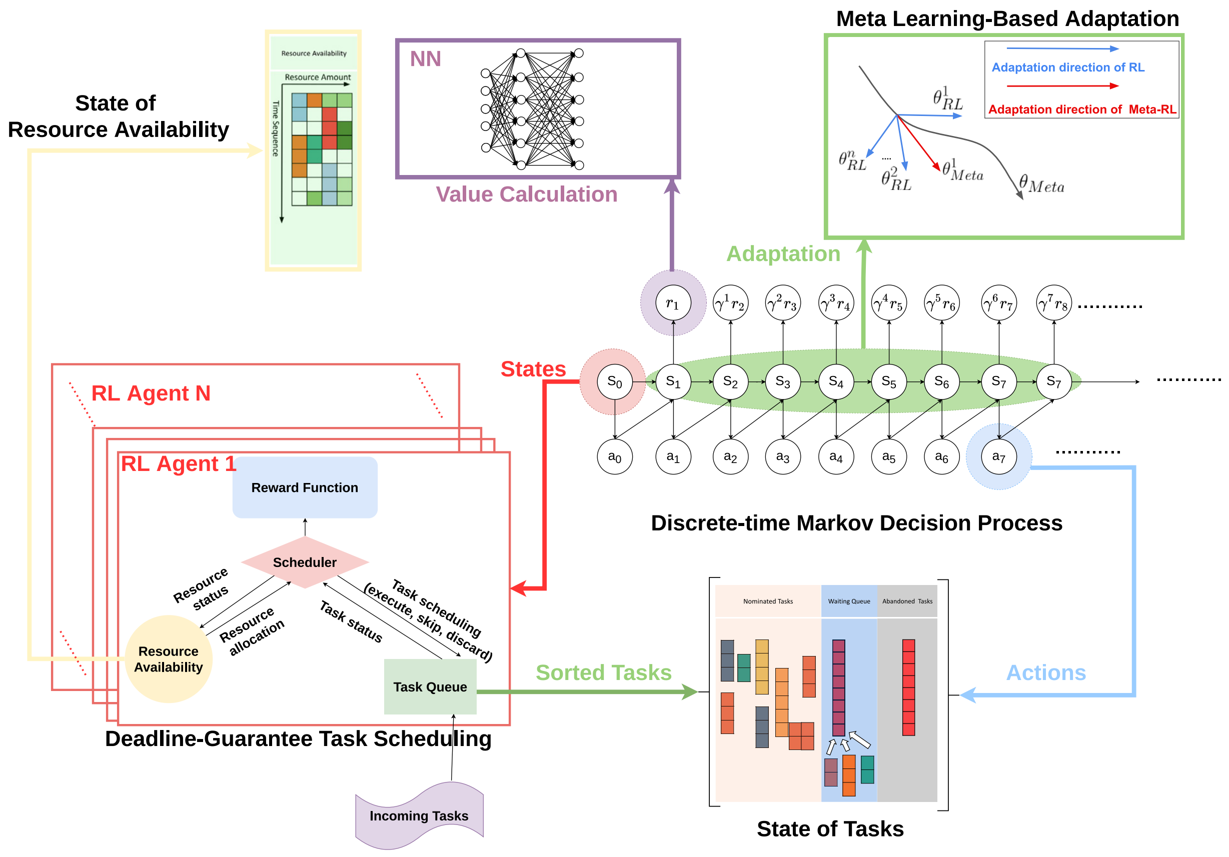


Fig. 3. The MLR-TC-DRLS approach.

The double Q-learning error can then be written as

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a; \theta_t); \theta_t') \tag{5}$$

We use the second set of weights θ_t' to evaluate the value of this policy fairly. This second set of weights can be updated symmetrically by switching the roles of θ and θ'.

The basic idea behind the Cross-Entropy Method(CEM) [33] is to tackle the original optimization problem with an adaptive sampling algorithm. Once the associated stochastic optimization is defined, the CE method alternates the following two phases:

1. Generation of a sample of random data (including trajectories and vectors) according to a specified random mechanism.
2. Update the parameters of the random mechanism based on the data to produce a "better" sample in the next iteration.

Since the Meta Learning method we integrate in our algorithm is sampling and learning in a more general manner, to better compare the sampling manner, we also integrated the CEM into one of the RL-based scheduling baselines. We use CEM here as a reference, to prove the sampling of our proposed method is more efficient.

By revisiting the approaches mentioned above: DQN aims to estimate the Q values of each action candidate, but because of its overestimation, as is shown in the results, the scheduling performance decreases with the increase of the workload and dynamics; the Double-DQN algorithm adds an extra neural network and experience memory to improve the optimization; CEM aims to adjust the sampling process to improve the optimization, because of the dynamics' direct influence on the sampling process the scheduling, the scheduling performance degrades dramatically with the increase of the dynamics.

3. Methodology

As aforementioned, in this work, we address the time-critical scheduling from robustness and deadline guarantee perspectives. As to scheduling performance robustness, we integrate the RL-based scheduling approach with the Meta-Learning approach and then optimize it; as to tasks deadline guarantee, we integrate a novel deadline guarantee scheme with the scheduling approach.

As shown in Fig. 3 we describe our methodology in the order of the scheduling process, which starts when the cloud platform receives a new task request. The new incoming task gets sorted into a queue within the "State of Tasks" area waiting for scheduling action, depicted in the middle of the figure, the red area. The execution actions are made by the learning process in "Deadline-Guarantee Task Scheduling". The process of learning scheduling policy, formulated as an MDP, depicted in the middle, consists of states, actions, rewards, and the transitions among states. First, each agent trains a policy model with a scheduler. The scheduler receives information, including resource availability and task queue information, then makes allocation decisions based on the current policy model. Afterward, the agent calculates the deadline critical reward function based on the scheduling decision to update the policy model. There are N RL learning agents running in parallel simultaneously. After the inner learning circle of the Meta-Learning framework, the meta learner adapts the scheduling policy model learned by RL agents into a more general scheduling policy model and updates the original policy model. This process is depicted at the top right of Fig. 3

in the green area. Then the scheduler follows this updated policy model for each state to execute the chosen task. Finally, the meta learner, i.e., the outer learning circle, updates the policy model and related state information: tasks status in different queues and platform resource availability.

3.1. Robust scheduling

It is familiar to train a model that fits the current situation among RL-based methods. The learning process relies on the interaction with environments in the short term. However, the environment could change interacting manners, namely distribution, consequently changing the target model. Hence the former learned model gets performance deviation. Retraining is a straightforward and standard solution to cope with this performance deviation.

Instead of learning a specific optimized model for each environment, Meta-Learning generalizes the learning process across different data sets and environments to adapt quickly to a new environment. The randomization within the training environment is the key to its robustness, which is also regarded as dynamics. In this way, a model trained accounting dynamics can effectively exploit and adapt to changes faster than the conventional manner, repeating the training process from scratch once again [39].

As is formulated in Meta-Learning, the training data distribution is denoted as Λ. By calculating the objective function based on π_θ, the learner agent aims to optimize the policy model at the same time minimizing training loss: $\mathcal{L}_\mathcal{D}$.

In this work, we integrate the conventional RL-based scheduling algorithm with gradient-based meta-learning, where the learning process updates the model as follows:

1. Inner circle learning: The learning agent samples training data-set denoted as $\mathcal{D}_{tr}$ from task distribution $\mathcal{D}$ for training and eventually calculating the updated model θ' for the next step outer circle training as follows:

$$\theta' = \Phi_\beta(\mathcal{D}_{tr}, \theta) \tag{6}$$

2. Outer circle learning: The learning agent applies the updated θ' from the inner circle to sample test data-set denoted as $\mathcal{D}_{te}$, then uses $\mathcal{D}_{te}$ to calculate the loss function and update the scheduling policy model.

$$\min_{\theta,\beta} \mathbb{E}_\mathcal{D}[\mathcal{L}(\mathcal{D}_{te}, \theta')] \tag{7}$$

For the inner circle learning update, we have:

$$\Phi_\beta(\mathcal{D}_{tr}, \theta) = \theta - \alpha \nabla \mathcal{L}(\mathcal{D}_{tr}, \theta) \tag{8}$$

Under objective function, after training across mutated environments [41], the learning agent achieves a more general model. Therefore, a model learned in this way can adapt to a changed scheduling environment quickly.

The scheduling policy model learned within the inner circle learning process is based on a data trajectory sampled from a cloud log file—the dynamics among resource availability and task workload change among training data sets or environments. The outer circle learns across different data trajectories to adapt to a more generalized model with better robustness. As to the inner circle learning process, the training objective is to learn the scheduling policy model, followed by the scheduler choosing a task and allocating resources. Therefore, the inner circle learning agent must continuously interact with the cloud environment to learn the updates from different components. As its interactive learning pipeline, Reinforcement Learning reasonably fits the scheduling environment in this work. The following subsection will introduce the detailed methodology, optimization, and numerical formulation of the RL-based scheduling part.

3.2. Meta-reinforcement learning method

This section proposes the inner circle RL pipeline to learn the scheduling policy model. We optimize and integrate a Meta-Learning based approach to update the scheduling policy model learned via RL agents. The final model is then more robust against the uncertainties incurred by dynamics.

The RL learning agents within the inner circle learning process updates the scheduling policy model as follows:

$$\theta' = \Phi_\beta(\mathcal{D}_{tr}, \theta) = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{\mathcal{A}_t, \mathcal{S}_t \sim \pi(\theta)}[\sum_{t=0}^{N} \gamma^t \mathcal{R}_t] \tag{9}$$

where, $\mathcal{D}_{tr}$ is sampled from tasks distribution Λ; $\mathcal{S}_t$ denotes state $\mathcal{A}_t$ indicates action (in size of N sampled points).

Then when it comes to the outer circle learning, the updated models from agents θ' will be used to calculate the final update as follows:

$$\theta = \theta - \alpha \sum_{j=1}^{N} \mathcal{L}_j(\mathcal{D}_{te}, \theta') \tag{10}$$

3.3. Time-critical scheduling

After integration with Meta-Learning, the robustness concern gets addressed, but the deadline missing issue is still not addressed. RL-based methods still lack a scheme of deadline guarantee. For time-critical tasks, especially after dynamics from resource availability, hardware failure, or even requests tide, the deadline is easily violated without a guarantee scheme.

We formulate the deadline-guarantee part of scheduling with Reinforcement Learning. As a serial decision making on resources allocation, the scheduling process is a perfect match with Markov Decision Process (MDP) based optimization, where we formulate Reinforcement Learning-based scheduling step by step later. In this MDP process denotes as $\mathcal{M}(\mathcal{S}, \mathcal{A}, \pi, \mathcal{R}, \gamma, H)$, where $\mathcal{S}$ denotes task scheduling state space, $\mathcal{A}$ represents scheduling action space, H represents the tasks number waiting to be calculated in each training iteration. The reward $\mathcal{R}$ is the sum of rewards $\sum_{t=0}^{H-1} r(s_t, a_t)$ from each trajectory $\tau := (s_0, a_0, \dots, s_{H-1}, a_{H-1}, s_H)$, which is defined in next section characterized as Eq. (17), π indicates the scheduling policy: $\mathcal{S} \times \mathcal{A} \xrightarrow{\pi} \mathbb{R}^+$, which is a probability distribution model characterized by θ over actions: $\pi_\theta(s, a) \in [0, 1]$, $\gamma \in (0, 1]$ is the discount factor in cumulative rewards.

Time-Critical Scheduling State Space $\mathcal{S}$: As shown in Fig. 4: there are four parts of information in each scheduling state: resource availability information, nominated tasks information, waiting for tasks information, and abandoned tasks information. We represent all states in a coordinate system, where each state's information is formulated as a certain amount of two-dimensional units in different colors. Fig. 4 also offers an example of the representation for each state. In the left green area of the coordinate system, the area of resource availability: the units in different colors indicate different tasks using this resource. In this coordinate system, the y axis denotes the spent time length, and the x axis represents the number of resources. The blank units show the number of available resources, while the colored units denote the number of resources dominated by different tasks. The nominated tasks are in the middle orange area, which are H tasks selected from the waiting area. The right blue area is the area of tasks waiting for the queue, including tasks waiting for nomination and new tasks. Moreover, the waiting queue tasks are sorted in the decreasing order of the execution slowdown, which helps the agent schedule more efficiently. The right gray area is

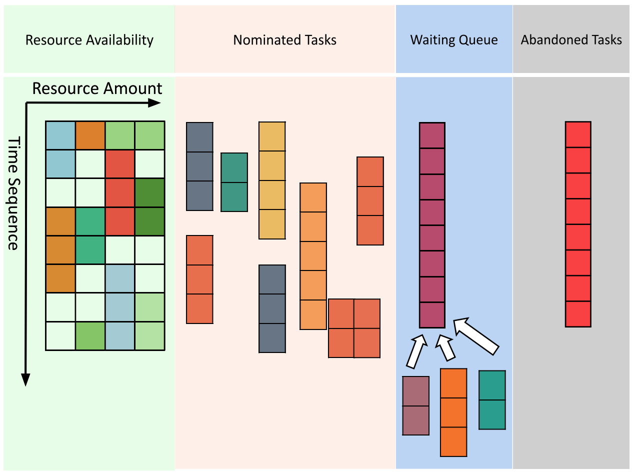


Fig. 4. Time-critical scheduling state representation.

the area of tasks abandoned by the scheduler. The details of each part of the learning process follow.

Scheduling Action Space $\mathcal{A}$: Scheduling action space is the collection of scheduling choices, i.e., nominated tasks here. As aforementioned, for each iteration, H (we set it to 50) tasks are nominated to be this action space according to the execution slowdown. Then the learning agent calculates each task's reward for deciding the action for each task. The decisions are made based on the policy model π_θ. The actions for each task are: executing, holding, and abandoning. After the action is executed, a new task from the waiting queue will be added to the nominated tasks pool. There is a check of $\mathcal{Q}^{fu}_{T_j}(t)$, the tasks with negative $\mathcal{Q}^{fu}_{T_j}(t)$ get abandoned to ensure the deadline guarantee. While for each task, no matter what kind of action it gets, its $\mathcal{Q}^{fu}_{T_j}(t)$ variable decreases one to push the learning efficiency. With the calculation of each iteration, $\mathcal{Q}^{fu}_{T_j}(t)$ becomes a deadline missing indicator of each task. Before finally executing the chosen task, the agent must check if the available resource matches the required amount. If not, the agent takes the action of holding upon that task for the next iterations.

Scheduling Policy Model π_θ: The scheduling policy is based on looking up Q value table calculated via a neural network (NN) [68]: two fully-connected layers, the activation function: rectified linear unit (ReLU), *softmax*, hidden layer: 30 neurons. The time-critical state representation we formulated previously is the *input* of the NN. Then the output of the first layer of NN is the probability distribution of selecting each task to execute (from the nominated task pool); Then, the second layer outputs the final choice of the task to execute. Via this two-step trick, the action space gets decreased from 3^H choices to H, which saves much of calculation capability.

The Objective Function: In this work, we aim to formulate an objective function to navigate the RL learning agent to optimize the scheduling policy π, which is interpreted into the maximization of the cumulative rewards as follows:

$$\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\sum_{j=1}^{H}\gamma r(s_t,a_t)\Big] \tag{11}$$

where (s_t, a_t), as aforementioned, denote state and action respectively.

The gradient of the scheduling policy model is as follows:

$$\begin{aligned}&\nabla_\theta\mathbb{E}_{\in(s_t,a_t)\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\sum_{j=1}^{H}\gamma r(s_t,a_t)\Big]\\&=\mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\Big[\nabla_\theta\log\pi_\theta(s_t,a_t)\mathcal{R}_{\pi_\theta}(s_t,a_t)\Big]\end{aligned} \tag{12}$$

Then we apply gradient descent [55] to update policy model parameters as follows:

$$\theta' = \theta + \alpha\sum_{t=0}^{\infty}\nabla\log\pi_\theta(s_t,a_t)r(s_t,a_t) \tag{13}$$

Based on this RL-based scheduling framework formulation, we optimize it with a deadline guarantee reward system.

Time-Critical Reward System: We formulate the whole reward system to guarantee the task deadline. The detailed formulation of the rewards system is depicted in Fig. 2: the horizontal axis in black color indicates the system time t_{cur}; Each task arrives at the platform with its arrival time T^{arr}_j and execution time T^{len}_j. To achieve the goal of deadline guarantee, we define an execution slowdown value $\mathcal{Q}_{T_j}(t)$ [56] for each task, whose calculation is as follows:

$$\mathcal{Q}_{T_j}(t) = \frac{t_{cur} - T^{arr}_j}{T^{len}_j} \tag{14}$$

The $f(\mathcal{Q}_{T_j}(t))$ function is meant to check whether execution slowdown of a task exceed the expected deadline or not. It is formulated as follows:

$$f(\mathcal{Q}_{T_j}(t),\Upsilon) = \begin{cases}1, & \text{if } \mathcal{Q}_{T_j}(t)\le\Upsilon\\ 0, & \text{otherwise}\end{cases} \tag{15}$$

Then the objective of RL process is formulated as follows:

$$\max\sum_{T_j\sim\mathcal{D}}[f(\mathcal{Q}_{T_j}(t),\Upsilon)] \tag{16}$$

where $\mathcal{D}$ indicates the sampled data trajectory, Υ represents the expected execution deadline threshold, which is defined according to Service Level Agreement (SLA) made between cloud user and provider. With the reward system, the execution slowdown variable of each task shall be bounded to guarantee that the tasks do not miss the deadlines.

More specifically, the execution slowdown variable calculation for a different group of the tasks is as follows for *nominated* tasks:

$$r_{T_j\in(s_t,a_t)} = \begin{cases}\max[0,(\mathcal{Q}_{T_j}(t)-\mathcal{Q}^{fu}_{T_j})T^{len}_j]P_a + P^{fuse}_a,\\ \text{if to be abandoned}\\ \max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T^{len}_j]P_a + B_o,\\ \text{if to be executed}\\ \max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T^{len}_j]P_a,\\ \text{if to be held}\end{cases} \tag{17}$$

where, P_a is a penalty constant threshold, Υ denotes expected deadline threshold. For tasks to be *abandoned*, whose execution slowdown value violates hard deadline $\mathcal{Q}^{fu}_{T_j}$, in their reward function, $\max[0,(\mathcal{Q}_{T_j}(t)-\mathcal{Q}^{fu}_{T_j})T^{len}_j]P_a$ is an execution slowdown penalty, while P^{fuse}_a is an extra penalty. For tasks to be *executed*, which do not exceed hard deadline while the current resource availability is enough according to their resource requirement, in their reward function, $\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T^{len}_j]P_a$ is an execution slowdown penalty, and B_o is a constant bonus value; For tasks to be *held*, which do not exceed hard deadline but the current resource availability is enough according to their resource requirement, in their reward function, $\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a$ is execution slowdown penalty. Overall, as shown in Fig. 4, the dynamics after each iteration are: successfully executed tasks get removed from nominated pool while new tasks get added to nominated pool; The held tasks remain in the nominated queue; The abandoned tasks get sorted in the abandoned queue.

Rewards of tasks in the *waiting* queue:

$$r_{T_j\in(s_t,a_t)}=\max[0,(\mathcal{Q}_{T_j}(t)-\Upsilon)T_j^{len}]P_a\mathcal{Q}_{T_j}(t)T_j^{len} \tag{18}$$

We also guide the learning agent to schedule waiting for tasks faster by giving them a waiting time related to their reward function.

3.4. MLR-TC-DRLS algorithm description

Algorithm 1 MLR-TC-DRLS algorithm

```
Require: Tasks distribution: $\Lambda$,
Require: Environment number: $N$, trajectory size: $h$
Require: Learning rate: $\alpha, \beta \sim \mathbb{R}^+$
1: Initialize the policy $\pi_\theta$ and $\mathcal{D} \leftarrow \varnothing$
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:     Sample the trajectories in the size $h_i \in (0, H)$
7:     for $h_i$ do
8:       Use policies $\pi_\theta$ to sample trajectories within first $h_i$ samples $\tau_{h_i} \sim H$
9:       Use $\tau_H$ to calculate adapted parameters: $\theta'_{h_i} = \theta + \alpha\sum_{j=1}^{h_i}\nabla\log\pi_\theta(s_t,a_t)r_{T_j}(s_t,a_t)$
10:      Use adapted policy $\pi_{\theta'_{h_i}}$ sample trajectories $\tau'_{h_i} \sim H$
11:    end for
12:    Use $\tau_H$ to calculate adapted parameters: $\theta'_H = \theta + \alpha\sum_{h_i}\nabla\log\pi_{\theta_{h_i}}(s_t,a_t)r_{T_j}(s_t,a_t)$
13:    Use adapted policy $\theta'_H$ sample trajectories $\tau'_H \sim \mathcal{D}$
14:  end for
15:  Calculate update: $\theta \leftarrow \theta - \beta\nabla_\theta\frac{1}{H}\sum_{j=1}^{H}\mathcal{L}_j(\theta'_H)$ using $\tau'_H$
16: end for
17: return$\theta$ as $\theta'$
```

After two phases of scheduling design described above: robustness and deadline guarantee of schedule, in this section, we integrate them to propose our approach MLR-TC-DRLS (Time-Critical Meta-DRL-Based Robust Scheduling). First, we will integrate the deadline guarantee scheme into the RL-based scheduling approach and then integrate the scheduling approach into the Meta-Learning paradigm. We formulate and explain the details of our proposed algorithm, MLR-TC-DRLS, by integrating inner circle learning, the RL-based approach, and the outer circle learning, which are described previously. As is shown in Algorithm 1, the input of the algorithm includes task distribution: Λ, learning rates: $\alpha,\beta\sim\mathbb{R}^+$, of the inner circle learning process and outer circle learning process, respectively. The initialization of the MLR-TC-DRLS in line 1: initialization of the scheduling policy model and resetting $\mathcal{D}$. Then the training data-set is pre-processed, denoted from lines 2 to 4: sampling N data trajectories from the distribution Λ by using the current scheduling policy model, then adding these trajectories to $\mathcal{D}$. The next step is the inner scheduling policy learning circle, described from lines 5 to 9, where the RL learning agents are ruining in parallel iterations. Each of them is an independent RL-based learning agent. The sample data sets τ_H inside $\mathcal{D}$ to calculate updated θ'_H. After achieving updated θ'_H, each RL agent uses θ'_H model to sample new data samples τ'_H from $\mathcal{D}$. After this, the algorithm turns to the outer learning circle, as shown in line 10, and the meta learner uses θ'_H to calculate the loss function based on τ'_H to achieve an update of the overall policy model. The time complexity of MLR-TC-DRLS is $\mathcal{O}(N\times H\times h)$. We will evaluate MLR-TC-DRLS in different aspects to see its performance.

4. Evaluation

We conduct a series of implementations to evaluate MLR-TC-DRLS by comparing it with basic RL-based scheduling approaches and state-of-the-art RL variants-based ones. It demonstrates that our approach achieves better time-critical scheduling performance and scheduling robustness.

4.1. Setup

Training Platform Settings: The details of the hardware platform where we implement all evaluations are: 18 $\times$ nodes, each node: GTX 1080 Ti$\times$4, Intel(R) Xeon(R) Gold 5118 CPU @ 2.30 GHz (12 cores per cpu)$\times$2, 128 GB memory, 10 TB local HDD$\times$2, 4 TB local SSD$\times$2. The details of software environment are: 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.

Scheduling Platform Settings: The platform we implement our scheduling algorithm is from EauroArgo, BlueCloud projects, whose structure is shown in Fig. 5. The platform consists of: Resource Management System, Information System(IS), Security System, and the MicroService Orchestrator System. These are complex ICT systems that exploit tailored persistence technologies managed via web services. The Resource Management System supports the creation of a Virtual Research Environment and its exploitation via the registration, management, and utilization of the resources assigned to it. The Information System supports the registration, discovery, and access of the resources profile. The Security System ensures the correct exploitation, auditing, and Auditing of the resources under the policies defined at registration time and customized at VRE definition time. It is orthogonal to all services operating in the infrastructure, and its components are deployed on all computing nodes. The MicroService Orchestrator System allows for a declarative definition of workflows, which are then executed by an engine. Decoupling orchestration logic of complex management tasks from the internals of single services enables a more scalable and manageable approach to complex procedures, facilitates tracking, monitoring and inspection of service interactions and finally provides a non-opinionated location for concentrating cross service logic (see Tables 2 and 3).

4.2. Data sets

Real-World Data Sets

The real-world data sets used in the evaluation are log files from the Euro-Argo Data Service, which includes more than 3500 autonomous float instruments worldwide. The purpose is to measure, collect and deliver temperature-salinity and related properties. Then those data are stored at the Euro-Argo data portal for different research communities to access and analyze. To ensure data availability, the Euro-Argo data portal has to efficiently manage resources for data storage, service request execution, and bandwidth allocation for downloads and uploads tasks. Therefore, the log data is a series of resource requests in this work, ideal for our scheduling use case. The data is collected 24 h per day, one-month-long recording continuous services. The data sampling frequency is 43200 samples taken per minute from the

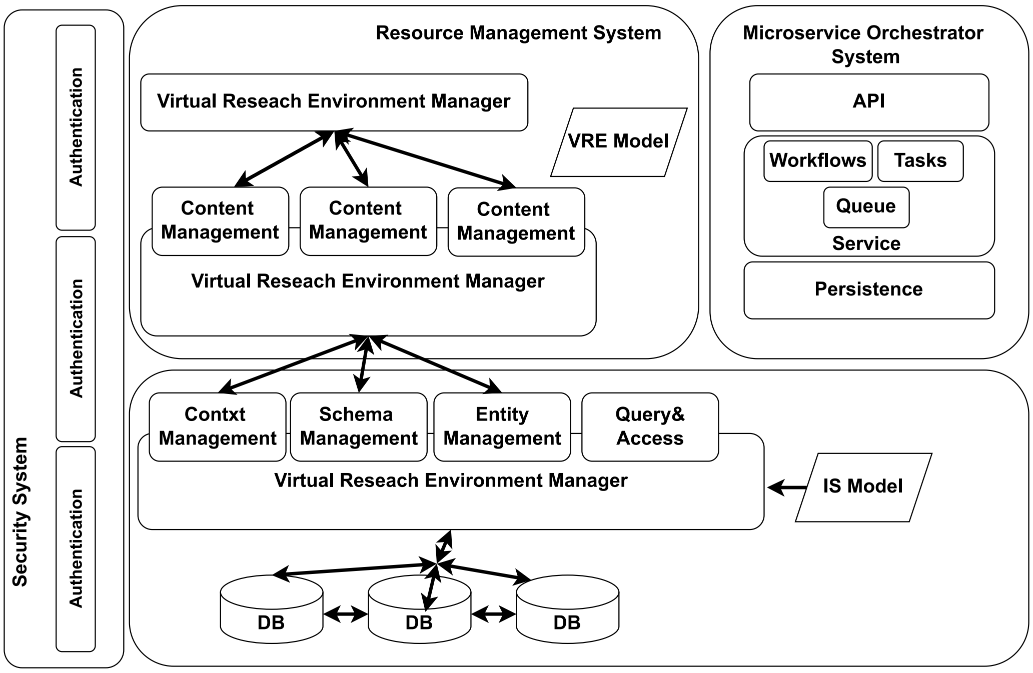


Fig. 5. Workload platform.

Table 2
Configuration of Fine-tuned RL baselines: We do fine-tuning of different RL-based scheduling approaches to be baselines. The configuration details of these baselines include different neural network layers, amount of neurons in each layer, optimizer, learning rate, discounted rate, and activation functions.

Fine-tuned RL approaches						
Parameters / RL approaches	NN layer number	Neuron amount	Optimizer	Discounted rate	Learning rate	Activation function
RL1	2	20	*RMSProp*	0.9	1e−5	ReLU, Softmax
RL2	3	30	*RMSProp*	0.8	1e−5	ReLU, Softmax
RL3	3	40	*Adam*	–	1e−5	ReLU, Softmax
RL4	3	50	*RMSProp*	0.7	1e−5	ReLU, Softmax

Table 3
Configuration of Fine-tuned RL Advanced Variants Baselines: We also fine-tune advanced RL variants DQN, Double-DQN, and CEM-based scheduling approaches to be baselines. The configuration details of these baselines include different neural network layers, replay buffer size, optimizer, learning rate, discounted rate, and activation functions.

Fine-tuned RL approaches						
Parameters / Baseline approaches	NN layer number	Replay buffer size	Optimizer	Discounted rate	Learning rate	Activation function
DQN	4	–	*Adam*	0.95	1e−3	ReLU, Softmax
Double-DQN	3	500	*Adam*	0.95	1e−3	ReLU, Softmax
CEM	3	—	*Adam*	0.95	1e−3	ReLU, Softmax

Table 4
Table of scheduling performance measurements.

Scheduling performance measurements		
MRED	≜	Missing Rate of Expected Deadline
MRHD	≜	Missing Rate of Hard Deadline
NAI	≜	Necessary Adaptation Iterations
CSP	≜	Converged Scheduling Performance
CW1,0000	≜	Convergence within 1,0000 times of iterations

Table 5
Table of robustness measurements.

Robustness measurements		
SPD	≜	Scheduling Performance Deviation
AIDURP	≜	Adaptation Iteration and Data Usage for Performance Recovery

4094157 raw log data. The data samples include task numbers, task transfer time, and requested data size.

Synthetic Data

Besides real-world data, we also adopt synthetic data, which includes two types of resources, i.e., CPU cores and memory, each with a capacity of m. There are three kinds of tasks depending on workload: light, medium, and heavy. More specifically, the ideal execution duration of light tasks follows uniform distribution between $1t_0$ and $5t_0$; the medium tasks follow uniform distribution from $5t_0$ and $10t_0$, and the duration of heavy tasks follows uniform distribution from $10t_0$ to $15t_0$, t_0 is one unified system time step. To be more practical, each task is configured with a randomly chosen dominant resource [56]. The desired amount follows uniform distribution between 0.25 m and 0.5 m, and the desired amount of other resources follows uniform distribution between 0.05 m and 0.1 m. The expected deadline threshold Υ is configured to be three, and for the hard execution slowdown variable, the configuration is 5. According to serial implementations and the prior work, [69] to compare the different results of optimization and robustness.

Table 6
General Configuration: We configure resources, CPU cores, and memory with a total capacity of m. Tasks are further divided into easy, medium, and heavy tasks. The duration of the tasks is chosen uniformly. For example, t_0 is a single step in the system. The execution slowdown is set to three times the duration of each task.

General 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)	$< 0.8m$	$< 15t_0$

4.3. Evaluation measurements

We define two groups of measurements to indicate and compare different experimental results and investigate different metrics (see Tables 4 and 5). The first group is related to deadline missing rate and scheduling performance:

Missing Rate of Expected Deadline (MRED) [70]: the proportion of executed task, which violate expected deadline, to all of tasks. MRED indicates the performance of time-critical scheduling for each method.

Missing Rate of Hard Deadline (MRHD) [70]: the proportion of executed task, which violate hard deadlines, to all of tasks.

Necessary Adaptation Iterations (NAI): the number of iterations each learning approach takes to reach a new convergence after the dynamic change in the environment.

Converged Scheduling Performance (CSP): a scheduling performance deviation measurement, which describes the portion of scheduled tasks that meet the expected deadline after retraining convergence:

$$CSP = 1 - [\mathcal{Q}_{conv} - \Upsilon]/\Upsilon \tag{19}$$

where, $\mathcal{Q}_{conv}$ is converged slowdown value.

Convergence within 1,0000 times of iterations (CW1,0000): whether retraining converged within 1,0000 iterations of adaptation in a new environment or not (yes/no).

In addition to defining the missing deadline metrics, we define two measurements to compare the robustness of scheduling performance: Schedule Performance Deviation (SPD) and Adaptive Iteration and Data Usage for Performance Recovery (AIDUPR). Therefore, the instantaneous performance deviation, just after the influence of workload dynamics or resource availability dynamics. The formulations are as follows:

$$SPD = \frac{|PER_{after} - PER_{before}|}{PER_{before}} \tag{20}$$

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

In addition to the instant performance deviation, we also show the retraining convergence speed, namely how quickly each scheduling approach adapts to dynamics. We use AIDUPR to indicate the time needed for iterations and the data needed for retraining adaptation and performance recovery:

$$AIDUPR = SPD * ITER * t \tag{21}$$

where $ITER$ denotes the iteration number of adaptation, t describes time length spent by each iteration.

Baseline Configuration

We build up the RL-based scheduling baselines to compare their scheduling performance against our proposed MLR-TC-DRLS to investigate MLR-TC-DRLS's performance from different aspects. To this end, we also optimize the RL-based algorithms in a similar scheduling manner as follows:

- **Scheduling State Space** We represent and store all scheduling states in the same way as we do for MLR-TC-DRLS, in a coordinate system, where each state's information is formulated as a certain amount of two-dimensional units in different colors. Similarly, as shown in Fig. 4, there are three areas of information areas in the coordinate system: the area of resource availability, in this coordinate system, the y axis denotes the spent time length, and the x axis represents the number of resources; The nominated tasks area, which are H tasks selected from the waiting area; The awaiting tasks area, includes tasks waiting for nomination and new tasks; Moreover, the awaiting tasks are sorted in the decreasing order of the execution slowdown, which helps the agent schedule more efficiently.
- **Scheduling Action Space** We formulate the scheduling action space as the collection of scheduling choices, i.e., nominated tasks here. As aforementioned, for each iteration, H (we set it to 50) tasks are nominated to be this action space according to the execution slowdown. Then the learning agent calculates each task's reward for deciding the action for each task. The decisions are made based on the learned scheduling policy model π_θ.
- **Scheduling Policy Model** The scheduling policy is based on different learning methods calculating via a neural network (NN): the NN structure and the configurations vary among the perspectives which we compare with our proposed algorithm, which will be elaborated in the baseline configuration section.

All RL-based scheduling baselines are all based on the work [56] with the integration of DQN, Double-DQN, and CEM methods to better compare their scheduling performance.

Before the adaptation comparison with RL-based approaches, we apply our approach, MLR-TC-DRLS, to learn a generic scheduling policy model by training across 30 trajectories sampled from each environment. Each environment has 4×4×4 types of configuration with mix of following properties: σ_{T_j}, d_j, T_j^{max} with respect of general configuration. Thus 1920 trajectories in total. The algorithm only gets to train in a new trajectory when it achieves the scheduling convergence in the previous trajectory. After finishing the generalization throughout this training process, we apply the learned scheduling policy model to new environment training.

Workload Setup

For comparison with RL, based on workloads of [56] (see Fig. 6), we set up three kinds of environments with respect to workload: *Light workload:* with $\sigma_{T_j} \in (0.1, 0.3)$, $d_j < 0.3$ m, $T_j^{max} < 5t_0$. *Medium workload:* with $\sigma_{T_j} \in (0.4, 0.5)$, $d_j < 0.5$ m, $T_j^{max} < 10t_0$. *Heavy workload:* with $\sigma_{T_j} \in (0.5, 1)$, $d_j < 0.8$ m, $T_j^{max} < 15t_0$. Each set we sampled 3 × 3 × 3 environments to do comparison (see Table 6).

4.4. Experiments

We aim to investigate three aspects:

1. To assess how much improved scheduling performance robustness the Meta-Learning approach could achieve, we conducted an implementation to compare different RL-based approaches' scheduling performance with NAI and CSP metrics. Before and after integration with the Meta Learning approach, the results will be demonstrated in Section 4.5.1.
2. To assess the deadline guarantee achieved by our novel reward function, we conduct implementations to compare RL variants-based approaches with MRED and MRHD metrics before and after integration with our deadline guarantee reward function; the results will be demonstrated in Section 4.5.2.

Table 7
Scheduling Adaptation Iteration: we compare four fine-tuned DQN, Double-DQN, and CEM-based approaches based on real-world Argo log records and showed that the meta-learning paradigm improves performance stability and speed of adaptation across different workload modes.

Workload	Light		Medium		Heavy	
Indicators / Approaches	NAI	CSP	NAI	CSP	NAI	CSP
DQN	5035 ± 523	90.21 ± 3.76%	8789 ± 982	89.76 ± 3.62%	9265 ± 1210	86.28 ± 9.45%
Meta-DQN	**4522 ± 352**	**92.54 ± 2.65%**	**7832 ± 650**	**91.22 ± 3.21%**	**8266 ± 1522**	**89.35 ± 5.36%**
Double-DQN	5247 ± 335	92.35 ± 3.21%	7056 ± 754	91.25 ± 2.65%	8932 ± 942	89.31 ± 5.50%
Meta-Double-DQN	**3568 ± 632**	**94.53 ± 3.71%**	**6533 ± 1504**	**92.82 ± 6.43%**	**20270 ± 3502**	**90.31 ± 7.50%**
CEM	5537 ± 314	91.76 ± 5.97%	8247 ± 2324	89.65 ± 4.23%	8983 ± 2525	85.35 ± 6.68%
Meta-CEM	**4326 ± 714**	**92.21 ± 3.53%**	**7765 ± 2124**	**91.35 ± 5.65%**	**8306 ± 2231**	**87.73 ± 4.62%**

Table 8
Time-Critical Comparison: We implement fine-tuned DQN, Double-DQN, and CEM-based scheduling approaches twice: the first time is without integrating our proposed time-critical deadline guarantee scheme, and the second time is with the guarantee scheme. As is shown in the table, our proposed time-critical deadline guarantee scheme can effectively decrease the deadline missing rate.

Workload	Light		Medium		Heavy	
Indicators / Approaches	MRED	MRHD	MRED	MRHD	MRED	MRHD
DQN	8.1 ± 1.6%	6.8 ± 2.4%	11.2 ± 2.1%	7.5 ± 1.7%	15.2 ± 3.6%	10.6 ± 3.2%
DQN_Guaranteed	**7.1 ± 2.5%**	**4.2 ± 1.4%**	**9.2 ± 2.3%**	**5.5 ± 2.6%**	**11.7 ± 3.2%**	**8.4 ± 3.7%**
Double-DQN	5.5 ± 2.6%	3.2±0.6%	8.8 ± 3.2%	6.7 ± 2.4%	12.5 ± 3.2%	9.4 ± 2.2%
Double-DQN_Guaranteed	**3.2 ± 0.6%**	**2.1 ± 1.2%**	**5.7 ± 2.6%**	**3.7 ± 1.4%**	**8.7 ± 1.2%**	**6.7 ± 2.2%**
CEM	9.7 ± 3.1%	6.5 ± 3.2%	13.7 ± 5.5%	8.2 ± 4.2%	15.1 ± 3.3%	11.6 ± 4.1%
CEM_Guaranteed	**8.4 ± 2.2%**	**6.1 ± 1.4%**	**10.7 ± 3.5%**	**7.2 ± 3.2%**	**13.3 ± 2.1%**	**10.6 ± 3.3%**

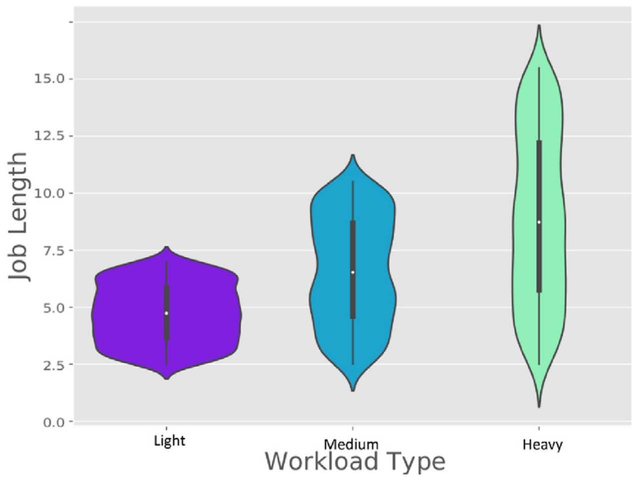


Fig. 6. Workload distributions.

3. To assess MLR-TC-DRLS' performance and robustness, we implement the comparisons between our MLR-TC-DRLS with other RL-based approaches; the results will be demonstrated in Section 4.5.3.

4.5. Experimental results

4.5.1. Robustness validation

As is shown in Table 7, we integrate our Meta-Learning scheme with the other four fine-tuned RL and three advanced variants DQN, Double-DQN, and CEM. We perform the implementations with Euro-Argo data sets and synthetic data sets with light, medium, and heavy workloads. The training iteration needed is shown in Table 7; the shorter ones are in bold font. As shown in Table 8, compared with the training before integration of the Meta-Learning scheme, the one integrated with the Meta-Learning scheme needs around 1000 times less iteration to converge and with around 2.0%–5.0% better performance when reaching the new convergence after workload change.

This part of the experiment validates the improvement of robustness brought by Meta-Learning. According to the results, Meta-Learning could improve the robustness of each RL scheduling method by reducing around 1000 times iterations, answering the first question: how much improvement in scheduling performance robustness could the Meta-Learning approach achieve? By improving robustness, we could offer around 90% of scheduling performance before dynamics but no deadline guarantee from each of them. Next step, we aim to improve the deadline guarantee of the scheduling.

4.5.2. Deadline guarantee validation

As shown in Table 8, for deadline guarantee, we integrate other three fine-tuned advanced RL variants based on scheduling methods with our deadline guarantee scheme: DQN, Double-DQN, and CEM. We perform comparison implementations based on the synthetic data and Euro-Argo data. We compare the scheduling performance before and after the integration with our deadline guarantee scheme. Finally, we increase the same proportion of workload for each method in the same environment. Then, we calculate the average of two deadline missing rates assessing the performance deviation due to integration. As shown in Table 8, all values in bold font are the best ones compared with others in the same row; all three RL methods' scheduling performance is improved, with a lower deadline missing rate, after integration with our proposed deadline guarantee scheme.

This part of the experiment is to validate our proposed deadline guarantee scheme. According to the results, our deadline guarantee scheme could lower the missing deadline rate by 1.0%–4.2%, answering the first question: how is the deadline guarantee achieved by integrating our novel reward function?

4.5.3. MLR-TC-DRLS validation

After validating our proposed robustness and deadline guarantee schemes for scheduling, we integrate them into complete MLR-TC-DRLS and validate its deadline guarantee and robustness. As shown in Table 9, we increase the same proportion of workload for each method in the same environment then we calculate the average of two deadline missing rates. As shown in Table 9, all values in bold font are the best ones compared with others in the same row. MLR-TC-DRLS steadily guarantees task

Table 9
Time-Critical Task Scheduling Performance Comparison: We conduct comprehensive implementation regarding time-critical tasks with MLR-TC-DRLS, and four fine-tuned RL-based approaches to assess their scheduling performance in terms of deadline guarantee. The results demonstrate that MLR-TC-DRLS outperforms the other RL-based scheduling methods by scheduling more percentage of tasks without violating their deadlines.

Workload	Light		Medium		Heavy	
Indicators / Approaches	MRED	MRHD	MRED	MRHD	MRED	MRHD
MLR-TC-DRLS	**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 ± 2.4%	5.5±3.6%	12.2 ± 5.3%	8.2 ± 3.3%	22.9 ± 6.1%	11.4 ± 5.2%
RL2	15.2 ± 4.3%	7.3 ± 3.1%	21.7 ± 10.3%	12.3 ± 5.5%	33.2 ± 7.9%	17.4 ± 11.2%
RL3	13.3 ± 3.1%	10.5 ± 3.1%	14.7 ± 5.5%	10.6 ± 4.6%	15.3 ± 2.5%	12.5 ± 3.3%
RL4	13.5 ± 5.3%	8.2 ± 2.1%	14.1 ± 6.4%	7.4±3.3%	16.3 ± 5.2%	10.2 ± 4.3%

Table 10
Scheduling Adaptation Speed Comparison: We conduct comprehensive implementation regarding adaptation speed with MLR-TC-DRLS and four fine-tuned RL-based approaches. We collect the measurements after changing the environment to assess the adaptation speed of each approach. According to the figures in the table, MLR-TC-DRLS' are all in bold, which outperform the others in adaptation speed and newly converged scheduling performance after influence under different workload.

Indicators / Workload Modes / Approaches	Light			Medium			Heavy		
	NAI	CSP	CW1,0000	NAI	CSP	CW1,0000	NAI	CSP	CW1,0000
MLR-TC-DRLS	**1210 ± 300**	**96.13 ± 3.24%**	✓	**2677 ± 657**	**95.22 ± 2.82%**	✓	**4366 ± 1211**	**91.23 ± 5.15%**	✓
RL1	10423 ± 2145	91.45 ± 5.21%	X	22754 ± 1203	75.31 ± 11.46%	X	8422 ± 2102	67.62 ± 14.71%	X
RL2	13473 ± 1133	85.36 ± 5.17%	X	17621 ± 3125	76.65.45 ± 4.81%	X	10125 ± 1014	82.45 ± 6.33%	X
RL3	6623 ± 765	92.12 ± 3.36%	X	12500 ± 1423	88.9 ± 7.53%	X	15032 ± 600	77.8 ± 5.76%	X
RL4	5477 ± 675	87.25 ± 7.56%	✓	9644 ± 1325	73.23 ± 9.77%	–	9642 ± 1742	85.67.45 ± 5.21%	–

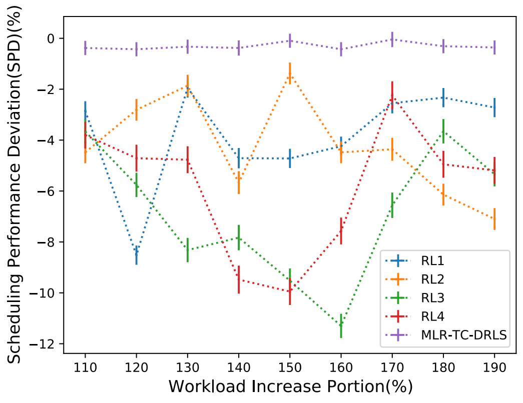
execution time within the deadline in each environment, which is better than the other four fine-tuned RL-based methods. Among tasks scheduled by MLR-TC-DRLS, only 3.0 ± 1.5% executed tasks violate hard execution slowdown variable 5. In contrast, among the tasks scheduled by RL-based methods, at least 10% of the tasks missed the expected deadline. With the increase in workload, for heavy tasks, as shown in Table 9, MLR-TC-DRLS offers the tasks deadline guarantees among different workloads. As shown in Table 10, we compare MLR-TC-DRLS's scheduling performance adaptation speed against four fine-tuned RL-based approaches. MLR-TC-DRLS takes fewer times to reach a new convergence and better scheduling performance. Above all, MLR-TC-DRLS works better than the other four fine-tuned RL in terms of task deadline guarantee and adaptation speed. We also compare the scheduling performance deviation right after dynamics to assess how robust each approach is. The task trajectories are sampled from the Euro-Argo log data and synthetic data. As shown in Fig. 7(a): with the same proportional workload increase, MLR-TC-DRLS' performance deviation remains within 50%, at the lowest point is even within 20%. While the fine-tuned RL-based methods experience the scheduling performance deviation ratio from −200% to even beyond −800% with the exact proportional change of workload. As shown in Fig. 7(a) the adaptation speed of MLR-TC-DRLS is more than 5 to 10 times faster than RL-based approaches on average after each time workload increase, demonstrating better robustness against dynamics.

As shown in Fig. 7(a): with the same proportional workload increase, MLR-TC-DRLS' scheduling performance deviation remains within −50%, at the lowest point even within −20%. In comparison, the fine-tuned RL-based methods experience the scheduling performance deviation ratio from −50% to even −250%. As shown in Fig. 7(c), the adaptation speed of MLR-TC-DRLS is more than five times faster than RL variants-based approaches on average after some proportional workload increase, demonstrating better robustness against dynamics. In the meantime, MLR-TC-DRLS stably guarantees the tasks' deadline across different environments, which beats the other three fine-tuned RL-based approaches. More specifically, only 3 ± 1.5% tasks scheduled by MLR-TC-DRLS violate hard execution deadline variable 5. In contrast, at least 7%–11.7% of other tasks scheduled by RL-based methods violate the expected deadline. As shown in Table 9, MLR-TC-DRLS converges with less time and with better-converged scheduling performance than RL-based methods. Thus MLR-TC-DRLS works better than the other three fine-tuned RL variants-based methods regarding deadline guarantee and adaptation speed.

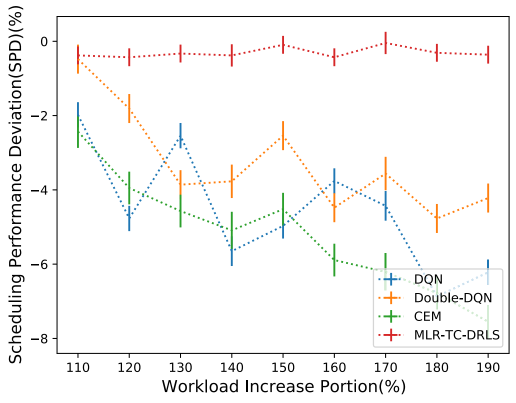
As to the comparison with advanced RL variants, shown in Fig. 7(b): with the same proportional workload increase, MLR-TC-DRLS' performance deviation remains within −50%, even within −20% at the lowest point. On the other hand, the fine-tuned RL variants-based approaches experience the scheduling performance deviation ratio from −200% to even more than −600%. From this, the robustness of MLR-TC-DRLS outperforms DQN, Double-DQN, and CEM-based approaches. As shown in Fig. 7(d) the adaptation speed of MLR-TC-DRLS is more than 5 to 10 times faster than advanced RL variants-based methods averagely after the same proportional workload increase, demonstrating better robustness against dynamics.

5. Discussion

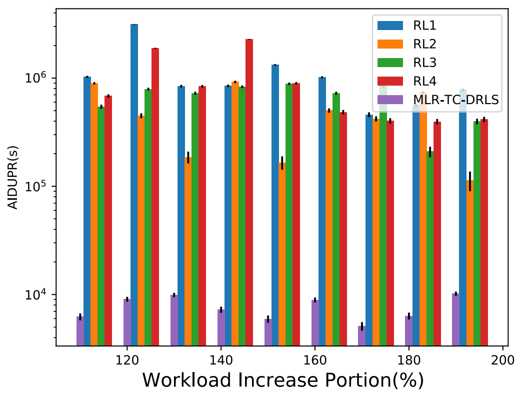
As shown in Section 4.5, MLR-TC-DRLS improves scheduling robustness and deadline guarantee. MLR-TC-DRLS outperforms the four conventional RL approaches and three advanced RL variants in scheduling performance robustness and adaptation speed to changing environments. MLR-TC-DRLS' scheduling performance deviation and adaptation speed change and the workload increase: both start firstly with 10%–30% increase then experience 30%–50% decrease finally increase again. While other fine-tuned RL-based methods do not show accordingly change in the pattern as workload changes in terms of their scheduling performance stability and adaptation speed. Moreover, MLR-TC-DRLS can keep the scheduling performance deviation within 30% when workload increase is within 50%. The deviation only experiences a more noticeable decrease when workload increases more than 50% but is still lower than fine-tuned RL methods, which are more than 200%. The advanced RL variants-based approaches: DQN, Double-DQN, and CEM, shows a decrease in scheduling stability and adaptation speed with the increase in workload. We are currently working on diversifying the resources to schedule in more complex service scenarios.



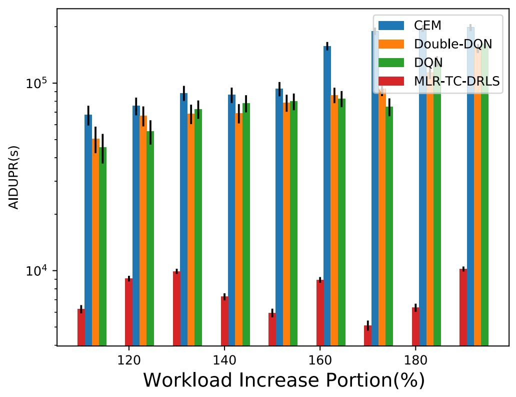
(a) *Comparison of performance deviation right after workload changes between MLR-TC-DRLS and other 4 fine-tuned RL-based approaches: "Workload Increase Portion" shows how much the change of the workload is, while "Scheduling Performance Deviation(SPD)" indicates how much the performance change right after the corresponding workload changes*



(b) *Comparison of performance deviation right after workload changes between MLR-TC-DRLS and other fine-tuned DQN, Double-DQN and CEM-based approaches: "Workload Increase Portion" shows how much the change of the workload is, while "Scheduling Performance Deviation(SPD)" indicates how much the performance change right after the corresponding workload changes*



(c) *Comparison of adaptation time spent on retraining after workload changes between MLR-TC-DRLS and other 4 fine-tuned RL-based approaches: "Workload Increase Portion" shows how much the change of the workload is, as defined shows how much spent in the adaptation process after the corresponding workload changes*



(d) *Comparison of adaptation time spent on retraining after workload changes between MLR-TC-DRLS and other fine-tuned DQN, Double-DQN and CEM-based approaches: "Workload Increase Portion" shows how much the change of the workload is, as defined shows how much spent in the adaptation process after the corresponding workload changes*

Fig. 7. Comprehensive comparisons in terms of scheduling performance deviation and adaptation speed between MLR-TC-DRLS and typical RL-based approaches.

MLR-TC-DRLS's overall scheduling performance deviation remains stable within −30% to −50%, which is much more stable than fine-tuned RL-based approaches, from −200% to −1000%. We plan to investigate the optimization approach to reduce and stabilize the instant scheduling deviation right after the dynamic influences for future work. Furthermore, we are also working on the online optimization of our proposed approach. We are also profiling reward function switches with different Reinforcement Learning approaches, considering an extension of our approach to be a framework, which is easy to automate integration with different RL-based scheduling approaches and platforms. Finally, we are also investigating more heterogeneous resource management regarding the increasing resource diversity. Another direction of future work is to minimize the time between arrival time and the start of execution time. Currently, we are guiding the scheduler to schedule tasks to wait longer time earlier. The next step would be further shortening the time period between arrival time and starting time of the execution. By reducing this period, we could achieve lower expected deadline missing rate and lower hard deadline missing rate as well.

6. Conclusion

This work presents MLR-TC-DRLS, a task scheduling approach that offers time-critical tasks deadline-guarantee at a cloud computing platform while improving scheduling performance robustness. We firstly optimize and upgrade RL-based scheduling approach with a Meta-Learning framework to improve the robustness and adaptation speed of the scheduling policy model against dynamics in the environment. Then we propose a novel deadline-guaranteed reward system for time-critical tasks. After validation of each approach, we optimize them together to propose MLR-TC-DRLS. Experimental results show that MLR-TC-DRLS can satisfy the deadline guarantee, outperforming fine-tuned basic RL methods and advanced RL variants. Furthermore, our proposed MLR-TC-DRLS can adapt to new environments taking 200%–500% less time than the fine-tuned RL and its variant-based scheduling approaches, achieving better scheduling performance robustness while offering deadline guarantees. Moreover, our proposed approach can also integrate with different RL-based scheduling algorithms to improve their scheduling performance robustness.

CRediT authorship contribution statement

Hongyun Liu: Conceptualization, Methodology, Software, Writing – original draft. **Peng Chen:** Conceptualization, Methodology, Software, Writing, Data curation, Writing – original draft. **Xue Ouyang:** Methodology, Software, Data curation, Writing. **Hui Gao:** Methodology, Software, Data curation, Writing. **Bing Yan:** Methodology, Software, Data curation, Writing. **Paola Grosso:** Supervision, Writing – review & editing. **Zhiming Zhao:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Zhiming Zhao reports financial support was provided by Horizon 2020 European Innovation Council Fast Track to Innovation. Zhiming Zhao reports financial support was provided by BlueCloud. Zhiming Zhao reports financial support was provided by LifeWatch ERIC. Hongyun Liu reports financial support was provided by China Scholarship Council. Xue Ouyang reports financial support was provided by National Natural Science Foundation of China No. 62002364. Hui Gao reports financial support was provided by the Natural Science Foundation of Shaanxi No. 2022JQ-651. Peng Chen reports financial support was provided by International Science and Technology Cooperation Program of Sichuan No. 2020YFG0326. Peng Chen reports financial support was provided by Talent Program of Xihua University No. Z202047. Zhiming Zhao reports financial support was provided by H2020 ARTICONF No. 825134. Zhiming Zhao reports financial support was provided by ENVRI- FAIR project No. 824068.

Data availability

Data will be made available on request.

Acknowledgments

This work is funded by the European Union's Horizon 2020 projects: ARTICONF (Grant No. 825134), ENVRI-FAIR project (Grant No. 824068), BLUECLOUD (Grant No. 862409), LifeWatch ERIC, also funded by: the National Natural Science Foundation of China (Grant No. 62002364), the Natural Science Foundation of Shaanxi, China (Grant No. 2022JQ-651), Science and Technology Program of Sichuan Province, China (Grant No. 2020YFG0326), and Talent Program of Xihua University, China (Grant No. Z202047).

References

[1] W. Chen, D. McDuff, DeepMag: Source-specific change magnification using gradient ascent, ACM Trans. Graph. 40 (1) (2020) 1–14.
[2] S. Singh, I. Chana, A survey on resource scheduling in cloud computing: Issues and challenges, J. Grid Comput. 14 (2) (2016) 217–264.
[3] T. Goethals, F. DeTurck, B. Volckaert, Extending kubernetes clusters to low-resource edge devices using virtual kubelets, IEEE Trans. Cloud Comput. (2020).
[4] J. Tang, W.P. Tay, T.Q. Quek, Cross-layer resource allocation with elastic service scaling in cloud radio access network, IEEE Trans. Wireless Commun. 14 (9) (2015) 5068–5081.
[5] L. Leite, C. Rocha, F. Kon, D. Milojicic, P. Meirelles, A survey of DevOps concepts and challenges, ACM Comput. Surv. 52 (6) (2019) http://dx.doi.org/10.1145/3359981.
[6] R.S. Olson, N. Bartley, R.J. Urbanowicz, J.H. Moore, Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 485–492.
[7] L. Wei, J. Cai, C.H. Foh, B. He, QoS-aware resource allocation for video transcoding in clouds, IEEE Trans. Circuits Syst. Video Technol. 27 (1) (2016) 49–61.
[8] J. Wei, X. Chen, J. Wang, X. Hu, J. Ma, Enabling (end-to-end) encrypted cloud emails with practical forward secrecy, IEEE Trans. Dependable Secure Comput. (2021).
[9] J. Mao, C.G. Cassandras, Q. Zhao, Optimal dynamic voltage scaling in energy-limited nonpreemptive systems with real-time constraints, IEEE Trans. Mob. Comput. 6 (6) (2007) 678–688.
[10] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, Multi-tiered on-demand resource scheduling for VM-based data center, in: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE, 2009, pp. 148–155.
[11] J.W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, Joint VM placement and routing for data center traffic engineering, in: 2012 Proceedings IEEE INFOCOM, IEEE, 2012, pp. 2876–2880.
[12] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, M. Steinder, Docker containers across multiple clouds and data centers, in: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing, UCC, IEEE, 2015, pp. 368–371.
[13] A. Ahmed, G. Pierre, Docker container deployment in fog computing infrastructures, in: 2018 IEEE International Conference on Edge Computing, EDGE, IEEE, 2018, pp. 1–8.
[14] V. Marx, The big challenges of big data, Nature 498 (7453) (2013) 255–260.
[15] S. Li, S. Dragicevic, F.A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, B. Jiang, J. Haworth, A. Stein, et al., Geospatial big data handling theory and methods: A review and research challenges, ISPRS J. Photogramm. Remote Sens. 115 (2016) 119–133.
[16] Z. Zhao, P. Martin, C. de Laat, A. Jones, I. Taylor, A. Hardisty, M. Atkinson, K. Jeffery, A. Zuiderwijk-van Eijk, Y. Yin, et al., Time Critical Requirements and Technical Considerations for Advanced Support Environments for Data-Intensive Research, Zenodo, 2016.
[17] P. Chen, H. Liu, R. Xin, T. Carval, J. Zhao, Y. Xia, Z. Zhao, Effectively detecting operational anomalies in large-scale IoT data infrastructures by using a gan-based predictive model, Comput. J. 65 (11) (2022) 2909–2925.
[18] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-aware datacenter tcp (d2tcp), ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 115–126.
[19] D. Bega, M. Gramaglia, A. Banchs, V. Sciancalepore, X. Costa-Pérez, A machine learning approach to 5G infrastructure market optimization, IEEE Trans. Mob. Comput. 19 (3) (2019) 498–512.
[20] J. Ru, 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, IEEE, 2013, pp. 78–87.
[21] L. George, 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, IEEE, 1997, pp. 441–448.
[22] C. Wang, S. Zhang, Z. Qian, M. Xiao, J. Wu, B. Ye, S. Lu, Joint server assignment and resource management for edge-based MAR system, IEEE/ACM Trans. Netw. 28 (5) (2020) 2378–2391.
[23] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica, Dominant resource fairness: Fair allocation of multiple resource types, in: 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 11, 2011.
[24] X. Tang, Reliability-aware cost-efficient scientific workflows scheduling strategy on multi-cloud systems, IEEE Trans. Cloud Comput. (2021).
[25] H. Tian, Y. Zheng, W. Wang, Characterizing and synthesizing task dependencies of data-parallel jobs in alibaba cloud, in: Proceedings of the ACM Symposium on Cloud Computing, 2019, pp. 139–151.
[26] S. Huang, X. Huang, N. Ansari, Budget-aware video crowdsourcing at the cloud-enhanced mobile edge, IEEE Trans. Netw. Serv. Manag. (2021).
[27] L. Niu, D. Zhu, Fixed-priority scheduling for reliable and energy-aware (m, k)-deadlines enforcement with standby-sparing, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2021).
[28] Y. Zhang, J. Yao, H. Guan, Intelligent cloud resource management with deep reinforcement learning, IEEE Cloud Comput. 4 (6) (2017) 60–69.
[29] W. Zhang, T.G. Dietterich, Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling, J. Artif. Intell. Res. 1 (2000) 1–38.
[30] X. Zhou, K. Wang, W. Jia, 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, IEEE, 2017, pp. 1–6.
[31] Q. Zhang, M. Lin, L.T. Yang, Z. Chen, P. Li, Energy-efficient scheduling for real-time systems based on deep Q-learning model, IEEE Trans. Sustain. Comput. 4 (1) (2017) 132–141.

[32] H. Hasselt, Double Q-learning, Adv. Neural Inf. Process. Syst. 23 (2010) 2613–2621.
[33] S. Mannor, D. Peleg, R. Rubinstein, The cross entropy method for classification, in: Proceedings of the 22nd International Conference on Machine Learning, 2005, pp. 561–568.
[34] Y. Wu, E. Dobriban, S. Davidson, DeltaGrad: Rapid retraining of machine learning models, in: International Conference on Machine Learning, PMLR, 2020, pp. 10355–10366.
[35] J. Yao, Q. Lu, H.-A. Jacobsen, H. Guan, Robust multi-resource allocation with demand uncertainties in cloud scheduler, in: 2017 IEEE 36th Symposium on Reliable Distributed Systems, SRDS, IEEE, 2017, pp. 34–43.
[36] P. Singh, A. Kaur, P. Gupta, S.S. Gill, K. Jyoti, RHAS: robust hybrid auto-scaling for web applications in cloud computing, Cluster Comput. (2020) 1–21.
[37] S. Guo, M. Chen, K. Liu, X. Liao, B. Xiao, Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing, IEEE Trans. Mob. Comput. (2020).
[38] S. Mireslami, L. Rakai, M. Wang, B.H. Far, Dynamic cloud resource allocation considering demand uncertainty, IEEE Trans. Cloud Comput. (2019).
[39] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.
[40] R.B. Slaoui, W.R. Clements, J.N. Foerster, S. Toth, Robust domain randomization for reinforcement learning, 2019.
[41] D. Li, Y. Yang, Y.-Z. Song, T. Hospedales, Learning to generalize: Meta-learning for domain generalization, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, No. 1, 2018.
[42] A. Karthick, E. Ramaraj, R.G. Subramanian, An efficient multi queue job scheduling for cloud computing, in: 2014 World Congress on Computing and Communication Technologies, IEEE, 2014, pp. 164–166.
[43] T. Qiu, K. Zheng, M. Han, C.P. Chen, M. Xu, A data-emergency-aware scheduling scheme for Internet of Things in smart cities, IEEE Trans. Ind. Inform. 14 (5) (2017) 2042–2051.
[44] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, A. Celesti, A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing, IEEE Trans. Ind. Inform. 15 (7) (2019) 4225–4234.
[45] A. Spachis, J. King, Job-shop scheduling heuristics with local neighbourhood search, Int. J. Prod. Res. 17 (6) (1979) 507–526.
[46] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling, IEEE Trans. Cybern. (2020).
[47] Z.-G. Chen, K.-J. Du, Z.-H. Zhan, 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, IEEE, 2015, pp. 708–714.
[48] X. Lu, 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, IEEE, 2011, pp. 296–300.
[49] S. Pandey, L. Wu, S.M. Guru, 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, IEEE, 2010, pp. 400–407.
[50] M.Z. Hasan, H. Al-Rizzo, Task scheduling in internet of things cloud environment using a robust particle swarm optimization, Concurr. Comput.: Pract. Exper. 32 (2) (2020) e5442.
[51] S. Sahoo, B. Sahoo, A.K. Turuk, A learning automata-based scheduling for deadline sensitive task in the cloud, IEEE Trans. Serv. Comput. (2019).
[52] A. Asghari, M.K. Sohrabi, 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 Comput. 24 (21) (2020) 16177–16199.
[53] X. Zuo, G. Zhang, W. Tan, Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid iaas cloud, IEEE Trans. Autom. Sci. Eng. 11 (2) (2013) 564–573.
[54] X. Zhang, C. Wu, Z. Huang, Z. Li, Occupation-oblivious pricing of cloud jobs via online learning, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 2456–2464.
[55] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.
[56] H. Mao, M. Alizadeh, I. Menache, S. Kandula, Resource management with deep reinforcement learning, in: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 50–56.
[57] Y. Hu, C. de Laat, Z. Zhao, Learning workflow scheduling on multi-resource clusters, in: 2019 IEEE International Conference on Networking, Architecture and Storage, NAS, IEEE, 2019, pp. 1–8.
[58] L.-C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of dag scheduling heuristics, in: Grid Computing, Springer, 2008, pp. 73–84.
[59] S. Goren, I. Sabuncuoglu, Robustness and stability measures for scheduling: single-machine environment, IIE Trans. 40 (1) (2008) 66–83.
[60] F. Ghezail, H. Pierreval, S. Hajri-Gabouj, Analysis of robustness in proactive scheduling: A graphical approach, Comput. Ind. Eng. 58 (2) (2010) 193–198.
[61] W. Kuang, L. Brown, Z. Wang, Transfer learning-based co-run scheduling for heterogeneous datacenters, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 29, No. 1, 2015.
[62] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, 2014, arXiv preprint arXiv:1412.6572.
[63] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, 2017, arXiv preprint arXiv:1706.06083.
[64] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 IEEE Symposium on Security and Privacy, Sp, IEEE, 2017, pp. 39–57.
[65] T.B. Brown, D. Mané, A. Roy, M. Abadi, J. Gilmer, Adversarial patch, 2017, arXiv preprint arXiv:1712.09665.
[66] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30, No. 1, 2016.
[67] S. Gu, T. Lillicrap, I. Sutskever, S. Levine, Continuous deep q-learning with model-based acceleration, in: International Conference on Machine Learning, PMLR, 2016, pp. 2829–2838.
[68] T.M. Moerland, J. Broekens, C.M. Jonker, A framework for reinforcement learning and planning, 2020, arXiv preprint arXiv:2006.15009.
[69] J. Wang, W. Bao, X. Zhu, L.T. Yang, Y. Xiang, FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds, IEEE Trans. Comput. 64 (9) (2014) 2545–2558.
[70] R. Khorsand, F. Safi-Esfahani, N. Nematbakhsh, M. Mohsenzade, ATSDS: adaptive two-stage deadline-constrained workflow scheduling considering run-time circumstances in cloud computing environments, J. Supercomput. 73 (6) (2017) 2430–2455.

Hongyun Liu is currently a Ph.D. candidate in the Multiscale Networked Systems (MNS) research group. He is also a junior lecturer at Graduate School Informatics, University of Amsterdam. He received the B.Sc. degree in Automation and M.Sc. in Navigation, Guidance and Control both from Northwestern Polytechnical University, Xi'an, China, in 2013 and 2017. His research interests include resource management, cloud computing, applied machine learning.

Peng Chen is currently a professor with School of Computer and Software Engineering, Xihua University, China and also a visiting scholar in the Multiscale Networked Systems (MNS) research group, University of Amsterdam, the Netherlands. He received the B.E. degree in computer science from University of Electronic Science and Technology of China, Chengdu, China in 2001, the M.Sc. degree in computer software and theory from Peking University, Beijing, China in 2004 and Ph.D. degree in computer science from Sichuan University, Chengdu, China in 2017. His research interests include machine learning and service computing.

Xue Ouyang received her B.Sc. and M.Sc. degrees from the National University of Defense Technology (NUDT), China, in network engineering and software engineering, respectively. She received her Ph.D. degree in the Distributed Systems and Services group in the School of Computing, University of Leeds, UK, in 2018. She is currently a Lecturer with the School of Electronic Science, NUDT. Her research interests include Cloud and Edge computing, intelligent scheduling, distributed storage, big spatial data analytics and blockchain.

Hui Gao was born in Shaanxi China. He received the B.S. in automatic control from Northwest Polytechnic University, in 2012 and the Ph.D. degree in control theory and control engineering from Chonqing University, in 2018. He is currently an associate professor in College of Electrical and Control Engineering, Shaanxi University of Science and Technology, Xi'an, China.His research interest includes the adaptive control, neural network system, multi-agent system and so on.

Bing Yan received the B.Sc. degree in automation and the M.Sc. degree in control theory and control engineering from the School of Automation at Northwestern Polytechnical University in 2012 and 2015, respectively. She is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering, University of Adelaide. Her research interests include flight control, formation control, and multi-agent systems.

Paola Grosso is currently an Associate Professor with the University of Amsterdam where she leads the Multiscale Networked Systems research group (mns-research.nl). Her work focuses on the creation of sustainable and secure e-infrastructures, and relying on the provisioning and design of programmable networks. She has an extensive list of publications on the topic and contributes to several national and international projects.

Zhiming Zhao is currently an assistant professor in the Multiscale Networked Systems (MNS) research group, University of Amsterdam, the Netherlands. He leads a team on "Quality Critical Distributed Computing" in the System and Networking Lab (SNE). His research focuses on innovative programming and control models.