Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

MRLCC: an adaptive cloud task scheduling method based on meta reinforcement learning

Xi Xiu[1], Jialun Li[1], Yujie Long[1] and Weigang Wu[1*]

Abstract

Task scheduling is a complex problem in cloud computing, and attracts many researchers' interests. Recently, many deep reinforcement learning (DRL)-based methods have been proposed to learn the scheduling policy through interacting with the environment. However, most DRL methods focus on a specific environment, which may lead to a weak adaptability to new environments because they have low sample efficiency and require full retraining to learn updated policies for new environments. To overcome the weakness and reduce the time consumption of adapting to new environment, we propose a task scheduling method based on meta reinforcement learning called MRLCC. Through comparing MRLCC and baseline algorithms on the performance of shortening makespan in different environments, we can find that MRLCC is able to adapt to different environments quickly and has a high sample efficiency. Besides, the experimental results demonstrate that MRLCC can maintain a high utilization rate over all baseline algorithms after a few steps of gradient update.

Keywords Meta reinforcement learning, Deep reinforcement learning, Task scheduling, Resource management

Introduction

Cloud computing has become a dominating paradigm for large scale information systems [1]. Cloud data centers consist of physical and virtual infrastructure resources which include server, network system and different resources. Cloud computing becomes an admired technology around the world because it offers a huge amount of storage and resource to different companies and organizations which can access these resources through proper management, rule and security. Some of the main characteristics of cloud computing are virtualization, large network access, automatic system and scalability [2]. In recent years, the development of cloud computing is remarkable. It has been used in several different fields, such as finance, health care, industrial manufacturing [3]. Many applications of cloud computing require the rapid increase of computing resources to satisfy the requirements of the clients. An easy solution for the problem is increasing the supply of resources. However, the economic cost is so large that it is not practical. Other solutions are proposed, such as improving the strategies of tasks scheduling to use resources as much as possible [4], executing the online and off-line tasks simultaneously to utilize the spare resources [5] and applying load balancing approaches to improve the utilization rate [1].

Many heuristic algorithms have been proposed to solve the problems mentioned above. For example, first fit [6], sample packing strategies [7], fair scheduling [8] and so on. There are also many complex meta-heuristic algorithms, like genetic algorithm [9], ant colony algorithm [10]. The performance of the above heuristic algorithms depends on not only the patterns of the resource demands, but also manual tests and adjustment. This means that it cannot adapt quickly if the environment changes.

However, the fast development in machine learning, particularly in reinforcement learning combining with deep neural network, offers new opportunities to tackle

*Correspondence:
Weigang Wu
wuweig@mail.sysu.edu.cn
[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China

the problem. For example, DeepRM [11] uses deep reinforcement learning to schedule tasks by representing the states as images, GoSu [12] applies graph convolution network to task scheduling, and Peng et al. [13] solved the problem of task scheduling using Q-learning. The training process of reinforcement learning depends on the large amount of data collected through interacting with the environment. Also, if the environment changes or some unexpected perturbations happen, the trained model may fail because of the weak adaptability. Therefore, it has low sample efficiency and needs full retraining to learn an updated policy for the new environment, and this process is time-consuming.

Meta learning is a promising method to address the aforementioned issues by taking advantage of the previous experiences across a range of learning tasks to significantly accelerate learning efficiency of new tasks [14]. For the reinforcement learning problem, meta reinforcement learning (MRL) [15] aims to learn policies from new environments within a small number of interactions with the environment by adjusting the previous meta model. The learning process of meta reinforcement learning consists of two "loops" of learning, the "outer loop" uses the experience over many tasks to gradually adjust the parameters of meta policy. The "inner loop" adapts fast to specific tasks through a small number of gradient updates.

There has been many researches related to meta reinforcement learning. For example, Pong et al. [16] proposed a hybrid offline meta-RL algorithm, which uses offline data to train an adaptive policy, and can adapt to a variety of new tasks at meta-test process. Dynamic-PMPO-CMA [17] integrates meta-learning with dynamic-PPO-CMA to train robots to learn multi-task policy. Meta-MAPPO [18] applies meta reinforcement learning to routing problem of packet networks to optimize the network performances under fixed and time-varying traffic demands. Kim et al. [19] proposed a novel meta-multiagent policy gradient theorem that directly accounts for the non-stationary policy dynamics inherent to multiagent learning settings based on meta reinforcement learning. However, there are few researches related to meta-RL of task scheduling in cloud. The most mentioned methods are applied in path planning of robots. So we combined meta learning with scheduling problems in this paper. And the meta learning process has large computing cost, we use the first-order approximation to reduce the cost.

The advantages of using meta reinforcement learning can be summarized as follows: First, the learning process of meta reinforcement learning is faster than reinforcement learning. Second, the training data can be reused in the process of training, which means that it is not necessary to get as much data as the reinforcement learning. Finally, the adaptability of meta reinforcement learning is better than reinforcement learning, and it can adapt to changes in the environment.

In this paper, we apply the meta reinforcement learning on task scheduling in cloud computing, and we proposed the MRL-based method (MRLCC). To evaluate the performance of MRLCC under dynamic environment, we construct different task scheduling scenarios with unique features. Through comparing the performance of MRLCC and baseline algorithms, MRLCC guarantees the shortest makespan and highest utilization rate of the servers among all algorithms. Furthermore, we verify that MRLCC can adapt to new scenarios more quickly than heuristic algorithms and is better than a DRL algorithm.

The rest of this paper is organized as follows. The related work is reviewed in Section "Related work". Background knowledge about RL and MRL is presented in Section "Background". The detail of MRLCC is elaborated in Section "Design". Section "Performance evaluation" is the performance evaluation of MRLCC and other algorithms. Finally, Section "Conclusion and future work" concludes the paper.

Related work

The task scheduling problem has attracted many researchers' interests. Based on the approaches used in task scheduling, there are two main kinds of the methods. One is the conventional approaches including heuristic methods and meta-heuristic algorithms. The other kind is DRL-based methods.

Conventional approaches

There are many works being done to enhance the conventional approaches. Pradhan et al. [20] proposed a modified round robin resource allocation algorithm to satisfy customer demands by reducing the waiting time. DGLB [21] reduces the energy consuming in data centers by designing energy-aware and geographical load balancing schemes for data-center networks. Under this comprehensive approach, workload and power balancing schemes are designed across the network, novel smart-grid features such as energy storage units are incorporated to cope with renewables, and incentive pricing mechanisms are adopted in the design. Ghobaei-Arani et al. [22] proposed a linear programming approach to web service composition problem, which is called 'LP-WSC', to select the most efficient service per request in a geographically distributed cloud environment for improving the quality-of-service criteria. Based on the concept of the control monitor-analyze-plan-execute (MAPE) loop, Ghobaei-Arani et al. [23] proposed an autonomic resource provisioning approach. Megh [24] models the problem of energy and performance-efficient

resource management as a Markov decision process, it uses a novel dimensionality reduction scheme to project the combinatorially explosive state-action space to a polynomial dimensional space with a sparse basis. Inspired by particle swarm algorithm, Kumar et al. [25] presented PSO-COGENT algorithm that not only optimizes execution cost and time but also reduces the energy consumption of cloud data centers. The APSO-VI algorithm is used to provide nonlinear ideal average velocity to control the search process to avoid the prematurity and divergence problems of PSO. Inspired by the mechanism of biological evolution, Jin et al. [26] took genetic algorithms as a mechanism based on optimal reservation selection to optimize the dispatch probability. It performs well on reducing response time and optimizing the energy consumption in the cloud system. Medara et al. [27] used a nature-inspired meta-heuristic approach called WWO through shutting down unemployed hosts to maintain a balance between performance and energy consumption in a cloud environment. WWO can efficiently search near-optimal solutions in multi-dimensional optimization problems. The weakness of these methods is that they rely heavily on expert knowledge or mathematical models.

DRL-based methods

The DRL-based methods have been used in cloud computing widely. QEEC [28] divides the scheduling process into two phases, the first phase implemented M/M/S to construct queueing model, the second phase uses a Q-learning based scheduler to assign tasks to virtual machines. This method can minimize task response time and maximize each server's CPU utilization. RLTS [29] works on reducing time consumption on task scheduling by using deep Q network. The reward function is related to the makespan after finishing action a at state s and transitioning to the next state s'. Besides, Yan et al. [30], Cheng et al. [31] and Cheng et al. [32] used deep Q-learning network to achieve high quality of service (QoS) by setting the reward function related to cost, response time and execution time of the job. Wei et al. [33] created a scheduler to make appropriate decisions to assign jobs without any prior knowledge using deep Q learning. Huang et al. [34] combined the adversarial imitation learning and deep Q learning together for cloud job scheduling, imitation learning provides an expert policy to guide the agent to find a near-optimal scheduling action. MADRL [35] is a multiagent deep reinforcement learning scheme, it uses actor-critic method to significantly reduce the computation delay and improve the channel access success rate in mobile-edge computing. Guo et al. [36] proposed DeepRM_Plus based on DeepRM, the method used a convolutional neural network to capture the resource management model and utilized imitation learning to accelerate convergence in the training process. Liu et al. [37] focused on saving the power consumption and energy usage, so they comprised a LSTM based workload predictor and a model-free RL based power manager in the local tier of the framework to predict workloads which can achieve the best trade-off between latency and power consumption. Moreover, Xu et al. [38] applied a Deep Neural Network (DNN) to approximate the action-value function, and formally formulate the resource allocation problem as a convex optimization problem. The simulation results show it can achieve significant power savings. However, the training process of DRL relies on a huge amount of data and the training process is time-consuming. Besides, when facing with new scenario, the trained policy needs to retrain to adapt to the new environment and the previous policy may be lost during the training process, which leads to low sample efficiency.

Background

Reinforcement learning

The purpose of RL [39] is to train an agent, which will get a maximize cumulative reward through interacting with the environment. A learning task is modeled as a Markov decision process (MDP) [40] as shown in Fig. 1, which is defined by a tuple $(S, A, P, P_0, R, \gamma)$. The agent could collect state information (s) from the environment, where s belongs to state space S, then the agent follows the state-transition probabilities matrix (P) to select an action a belonging to an action set (A). P_0 is the initial state distribution, which will be used at the first step. After each step, the agent will receive a reward as r_t at time t, which is calculated by the reward function (R). The cumulative reward is calculated by adding the product of each-step reward and the discount factor ($\gamma \in [0, 1]$). During the process of RL, we can sample a trajectory according a policy $\pi(a \mid s)$, where $a \in A$ and $s \in S$. The trajectory is represented as $\tau_\pi = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$.

When the policy $\pi(a \mid s)$ is parameterized by neural network parameters θ as the Fig. 2, the state value function of a state s_t is calculated by

$$v_\pi(s_t) = E_{\tau \sim p_\tau(\tau|\theta)}\left[\sum_{k=t} \gamma^{k-t} r_k\right], \tag{1}$$

where $P_\tau(\tau \mid \theta)$ is the probability distribution of sampled trajectories based on $\pi(a \mid s; \theta)$. In this way, the goal of RL is to find optimal parameters to maximize the expected total rewards $J = \sum_{s_0 \sim P_0} v_\pi(s_0)$.

Meta reinforcement learning

The goal of meta learning is to enable an agent to quickly acquire a policy for a new test task using only a small amount of experience in the test setting.

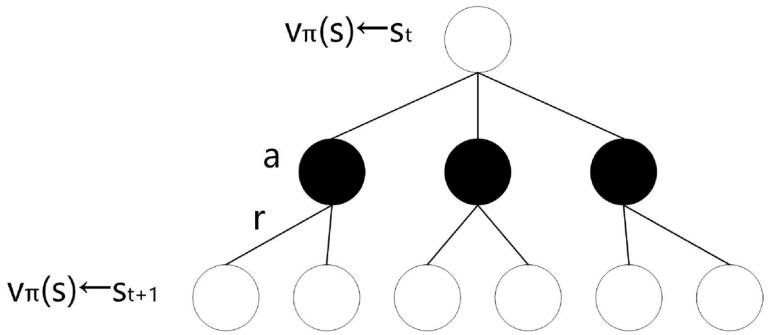


Fig. 1 The markov decision process

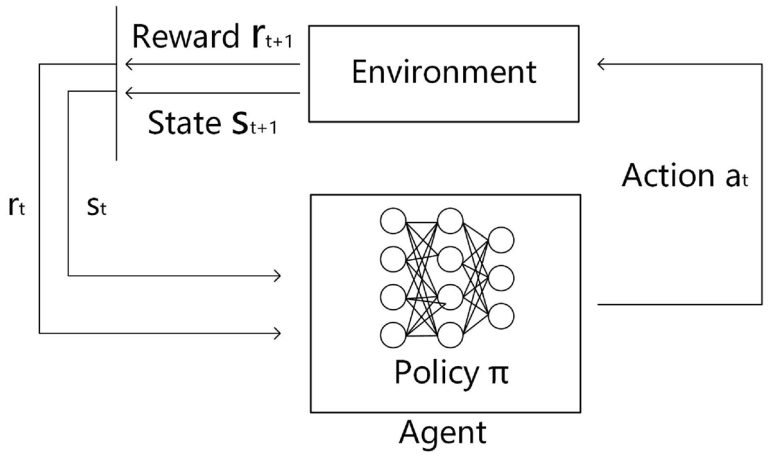


Fig. 2 The deep reinforcement learning

Specifically, the task here is not as same as the definition in cloud computing. As for the meta reinforcement learning, each learning task corresponds to a different MDP with different reward functions in the learning process, but the different learning tasks have the approximate states and action spaces. To accomplish this goal, the model is trained during a meta learning phase on a set of tasks, then the trained model can quickly adapt to new test tasks using only a small number of examples or trials. Meta learning has been used in a variety of machine learning tasks, such as classification, regression, and reinforcement learning [41, 42]. There are lots of methods, which present different ideas in different aspects of MRL. For example, MAML [15] presented a formulation of model-agnostic meta learning, which will be introduced later.

We take policy-based reinforcement learning as an example to simply introduce the process of meta reinforcement learning. For each task $\mathcal{T}_i$, the objective function is denoted as $J_{\mathcal{T}_i}(\theta)$. During the process of training for each task $\mathcal{T}_i$, the parameters are updated by

$$\theta_i \leftarrow \theta + \alpha \nabla_\theta J_{\mathcal{T}_i}(\theta). \tag{2}$$

α is the learning rate for the task $\mathcal{T}_i$, θ is the network parameters of meta policy. Each task uses θ as initial network parameters to train reinforcement learning model, so we can get θ_1, θ_2, ..., θ_i. The learning process for the meta learning, follows the equation

$$\theta \leftarrow \theta + \beta E_{\mathcal{T}_i \sim \rho(\mathcal{T})}\left[\nabla_\theta J_{\mathcal{T}_i}(\theta_i)\right]. \tag{3}$$

where β is the learning rate for meta learning. The training process of meta reinforcement learning is shown in Fig. 3.

Design

Task scheduling scenario

The structure of the scheduling system can be decomposed into three parts [43], job pool, resource cluster, and the scheduler. The job pool is used to cache different tasks from different types of users. The tasks are temporarily stored in the task queue until the servers have

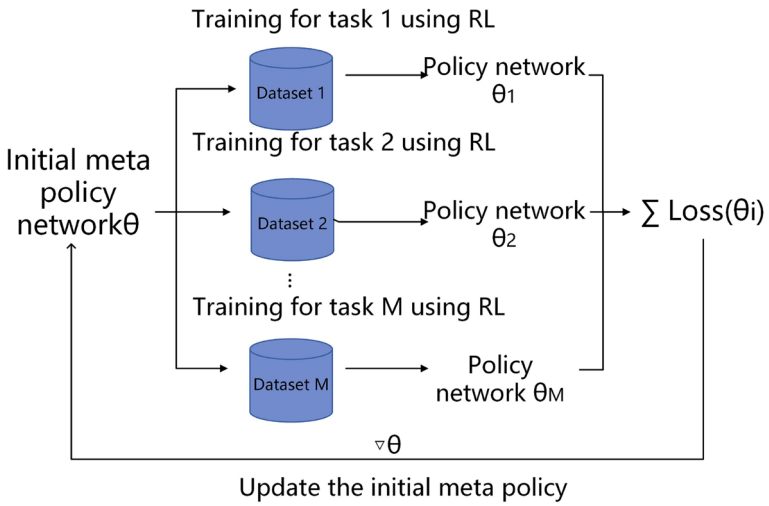


Fig. 3 Training process of meta reinforcement learning

enough resource to finish the tasks. The task is denoted by required resources *res* and execution time *t*. We assume that the resource demands and execution time of each task are known before scheduling. And we also assume that the problems like breaking down of servers and information loss don't happen in the platform. The server cluster is deemed as the resource cluster, which is responsible for executing the tasks in the job pool reasonably using remaining resources, usually the resources include CPU and memory. In this paper, we mainly consider the task scheduling problem about CPU. The scheduler is an important part of the cloud platform, it consists of multiple components, such as the job monitor, resource monitor, scheduling policy model and SLA. SLA is a mutually agreed agreement defined between service providers and users to guarantee the performance and availability of the service under cost. The state information of tasks and servers is collected by job monitor and resource monitor. The scheduling policy model uses the collected information to make scheduling decisions. The scheduling policy is modeled by a neural network, which takes the state information as input and outputs the scheduling decision executed by the scheduler. The architecture of the system is shown in Fig. 4.

The systems architecture is constructed based on the DRL-based cloud platform, and the difference is that the

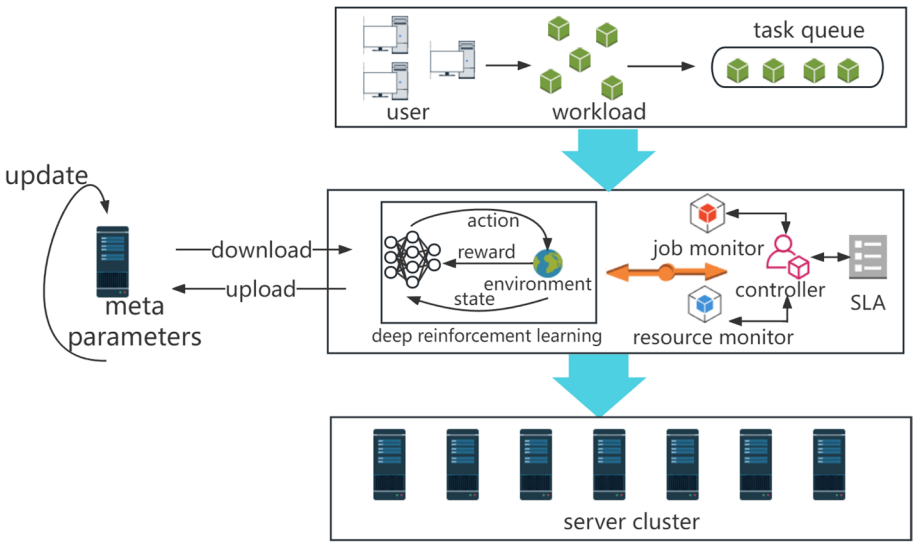


Fig. 4 The structure of the scheduling system

network parameters θ have to be downloaded from the server. Then the agent applies the parameters to train a policy to schedule jobs, and parameters θ are updated following the reinforcement learning method in the cloud. After finishing the update process in different scenarios, the meta parameters θ have to be updated using the new parameters in those scenarios. The new parameters in different scenarios are uploaded to the server where meta parameters are stored, and the meta parameters θ are updated using the uploaded parameters to accomplish the outer loop of meta learning as the Eq. (2).

Model

In this section, we detail the state, action, reward of reinforcement learning in the training process of MRLCC.

States: The state of the cluster is defined as S^t at time t, the task and server are recorded as j, m respectively. We express the environment state of the system as a one-dimensional vector, which includes the resource usage of the server at the moment t as m^u and the server's resource capacity as m^c. For task j, the resource demand is represented as j^r, execution time of task j is j^e, and the task's waiting time is j^w. The resource usage of the server and the resource demand are represented as the occupancy rate of the resource. So the value of the two characteristics ranges from 0 to 1. For a cluster including N servers, the state S^t can be defined as $S^t = \left[S_1^m, S_2^m, ...S_N^m, S^j\right] = [m_1^u, m_1^c, ...m_N^u, m_N^c, j^r, j^e, j^w]$.

Actions: The job monitor pays attention to the job pool, when the task in the queue is ready, the scheduler is activated to make decisions using the reinforcement learning algorithm. The scheduling action of the scheduler can be expressed as a finite number of discrete numbers. For example, if the scheduling action is i, it means that the task should be scheduled to server i. Also, we consider that the policy should take all situations in account, especially in some cases, the scheduler may take no action that means the scheduling policy thinks it is better to scheduling nothing. Then the task is put back to the queue to wait next time step. Therefore, we use action *None* to represent the action to delay the job to next time step. The state of a task is either scheduled to a server, or assigned to wait next step. The space is defined as $A = \{a \mid a \in \{None, 1, 2, \ldots, N\}\}$, where N is the Nth servers.

Rewards: The scheduling goal is to improve the overall resource utilization of all machines, besides we expect the scheduler could fast finish the scheduling process. So the reward function consider the resource utilization U and the amount of scheduled tasks *Num* at the time t. The resource utilization U^t at time t is represented by the average of all servers' resource utilization, which is shown as:

$$U^t = \frac{\sum_{i=1}^{N} U_t^i}{N}, \tag{4}$$

where U_t^i is the ith server's resource utilization and N is the number of servers.

However, the amount of scheduled tasks is not used directly as a part of the reward function, we combine the time t with it to be a part of the reward. This part is similar to the concept of handling capacity, which is the number of scheduled tasks per unit time.

$$Through^t = \frac{Num^t}{t}. \tag{5}$$

In order to avoid the scheduler prefers to schedule the short tasks, we use *cost* to be a part of the reward function to stimulate the scheduler to schedule the long tasks if the requests could be satisfied. The cost of the task is shown as:

$$Cost^j = Execution^j \times Require^j, \tag{6}$$

where $Execution^j$ and $Require^j$ are the execution time and required resources of task j respectively. So, the reward function could be expressed as:

$$r(t) = \lambda U^t + \mu Through^t + \nu Cost^j, \tag{7}$$

we have tested many combinations of the coefficient, then λ, μ, ν are set to 0.4, 0.4, 0.2. Besides, when the action is *None*, ν is set to 0 and λ, μ are both 0.5 to make U and *Through* have same weight.

Training algorithm of MRLCC

The process of MRLCC consists of two parts, as we mentioned before, the one part is to train a policy for a specific scenario which is the "inner loop", the another is to improve the meta policy, this is the "outer loop". For the specific scenario training, we apply policy gradient method as the training method of reinforcement learning algorithm. Policy gradient method is one of the major reinforcement learning methods, compared with value-based algorithm like Q-learning [44], it works on the policy directly through sampling trajectories from the environment iteratively [45]. The policy is parameterized by parameters θ, it is a mapping between the environment state and the action, and the state vectors are input of the policy network. It trains a probability distribution through policy sampling and enhances the probability of selecting actions that lead to high returns. In other words, it directly strengthens the probability of selecting good behaviors and weakens the probability of selecting bad behaviors via rewards. θ represents the parameters of meta policy, it is initialized randomly. For each scenario i, the initial parameters of the policy network θ^i is copied from the meta policy, this

means that $\theta^i = \theta$ at the beginning of training, which is the step 5 in Algorithm 1. The reward function has been introduced in the last part, for a trajectory τ sampled using the policy parameters θ, the cumulative return is expressed as

$$R_\theta = \sum_{t=0}^{T} \gamma^{T-t} r_t. \tag{8}$$

The expected total reward J mentioned in *reinforcement learning* is recorded as $\bar{R_\theta}$, the expectation of the collected N trajectories. Then the gradient of the expectation $\bar{R_\theta}$ is expressed as

$$\begin{aligned} \nabla \bar{R_\theta} &= \sum_{\tau} R(\tau) \nabla p_\theta(\tau) \\ &= \sum_{\tau} R(\tau) p_\theta(\tau) \nabla \log(p_\theta(\tau)) \\ &\approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log(p_\theta(\tau^n)) \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} R(\tau^n) \nabla \log(p_\theta)(a_t^n \mid s_t^n). \end{aligned} \tag{9}$$

As we can see, the gradient is related to the probability of the choice, we use the gradient to finish the process of gradient ascent, as the Eq. (2). After the training process, the policy is updated to be suitable for the specific scenario i and the parameters of the policy is updated as $\theta_i^{'}$.

Next, we will introduce the "outer loop" of MRLCC. We apply the training process of policy gradient on all scenarios, and we can get several different policy networks [$\theta_1^{'}$, $\theta_2^{'}$, ..., $\theta_M^{'}$], they are used to update the meta policy parameters. Then, we conduct the gradient ascent to maximize the expectation of all scenarios, the expectation is expressed as follows.

$$E_{\mathcal{T}_i \sim \rho(\mathcal{T})} J_{\mathcal{T}_i}\left(\theta_i^{'}\right). \tag{10}$$

It aims to get a policy which can adapt fast to perform well among all scenarios, but it is unexpected to overfit. However, the gradient of the expectation is hard to calculate, because the computation cost is too large to implement. So we use the first-order approximation to replace the second-order derivatives [46], the process can be simplified as

$$g = \frac{1}{M} \sum_{i=1}^{M} \left[\left(\theta_i^{'} - \theta\right)/\alpha\right]. \tag{11}$$

g is the simplified gradient, M is the number of training scenarios, and α is the learning rate for the process of gradient ascent in the "inner loop". The overall design of the algorithm is presented in Algorithm 1.

We first set a batch of learning scenarios for the training of the specific environment. After finishing the training, we update the meta-policy parameters θ by using gradient ascent mentioned in the Eq. (3). β is the learning rate of the meta-policy updating process, and the process of getting the gradient is a little different from the definition of the goal of meta policy.

Input: Datesets of several training scenarios
1: Randomly initialize the parameters of meta policy,θ
2: **for** iteration $k \in [1, K]$ **do**
3: Sample M scenarios $[\mathcal{T}_1, \mathcal{T}_2, ...\mathcal{T}_M]$ from the training datasets
4: **for** each scenario $\mathcal{T}_i$ **do**
5: Initial θ_i using θ
6: Sample trajectories set $D = [\tau_1, \tau_2, ...]$ from $\mathcal{T}_i$ using sample policy θ_i
7: Compute the policy network parameters $\theta_i^{'} \leftarrow \theta_i + \alpha \nabla_\theta J_{\mathcal{T}_i}(\theta_i)$ with D
8: **end for**
9: Update meta policy $\theta \leftarrow \theta + \beta g$ via Eq. (11).
10: **end for**

Algorithm 1 Meta Reinforcement Learning of MRLCC

Performance evaluation

This section presents the experimental results of the proposed method. First, the hyperparameters of the algorithm and simulation environment are introduced. Next, we evaluate the performance of the algorithm by comparing it with the policy gradient applied on all scenarios and several heuristic algorithms.

Experimental settings

The algorithm is implemented via Pytorch 1.12.1. The policy network is set as a three-layer fully connected neural network, with 256, 128, 64 units at each layer. The activation function and optimization method are Tanh and Adam respectively. And the last activation function is Softmax. The learning rate α, β for the training of "inner loop" and "outer loop" are both set as $3{\times}10^{-4}$, and the discount factor for "inner loop" training is set as 0.99. In total, we summarize the hyperparameter setting in Table 1.

We design two experiments to evaluate the performance on different scenarios. The first experiment simulates the scenarios where the scheduler faces a new scenario. The new scenario is different from the scenarios used in the training process. Then in the second experiment, we change the resource capacity of the servers to show the influence of the resource capacity on the performance of different algorithms. We use K-means to classify the data of online service, and the purpose of this action is to find the different patterns behind the utilization of resource. The clustered online data is used to simulate different scenarios where different kinds of online

Table 1 The neural network and training hyperparameters

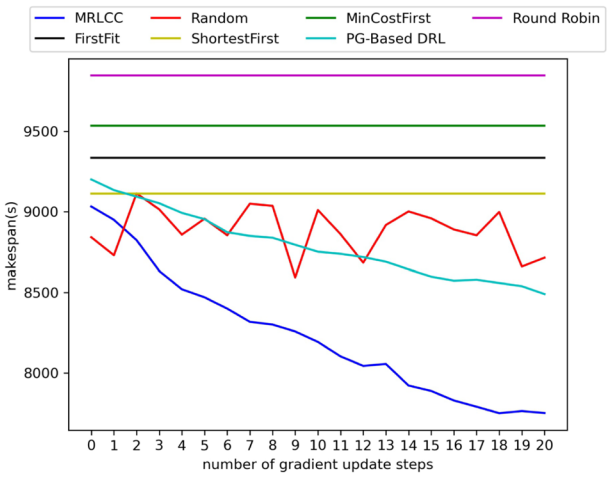
Hyperparameter	Value
lay1	256
lay2	128
lay3	64
activation function1,2	Tanh
activation function3	Softmax
optimization method	Adam
learning rate α	3×10^{-4}
learning rate β	3×10^{-4}
discount factor γ	0.99

services are deployed. Besides, the data of offline service is also clustered to get different types of task. Then we combine different types of online service and offline service to generate different scheduling scenarios for the process of training the model. We compare the algorithm with six baseline algorithms:

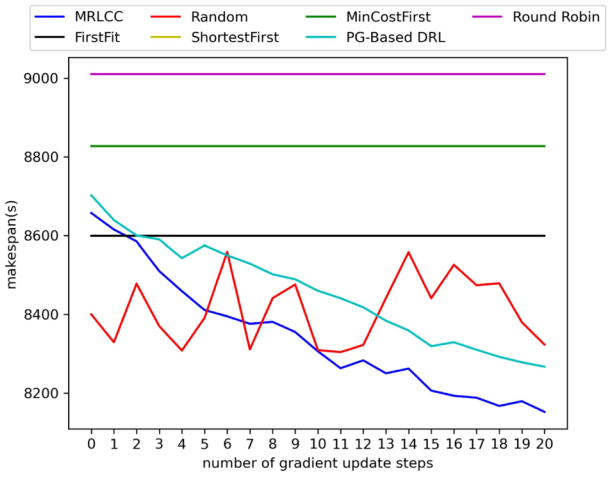
First Fit: The scheduler will schedule the task to the first server which meets the requirement encountered in the traversal.
Random: The scheduler will randomly choose a feasible scheduling action from all choices.
Shortest First: The scheduler will preferentially schedule the tasks whose execution time is shortest.
Mincost First: This algorithm will take the product of task's execution time and resource requirement as the criterion of prioritizing.
Round Robin: The servers are ordered as a cyclic manner, and a mark is allocated to the first server. The task is scheduled to the server with the mark, when scheduling is finished, the mark is passed to next server according the cyclic order.
Policy Gradient-Based (PG-based) DRL: We use policy gradient method to pre-train one policy using all training data, and then use the parameters of the policy network as the initial parameters to update on the testing data. The parameters are updated according to Eq. (2).

Simulation environment

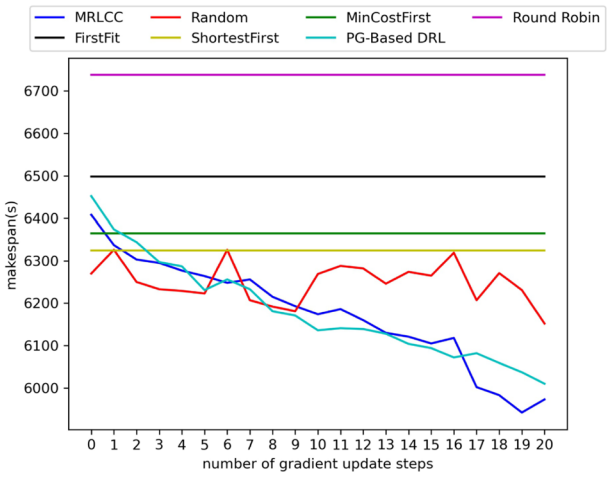
We develop a simulative cloud environment as the model in Section "Background" which consists of 10 servers. The CPU cores of each server are 10. Besides, we consider the servers not only just deal with offline tasks, but also support online service. The online service provides the user with constant connection to the Internet or other facilities, which occupies the resources



(a) scenario 1



(b) scenario 2



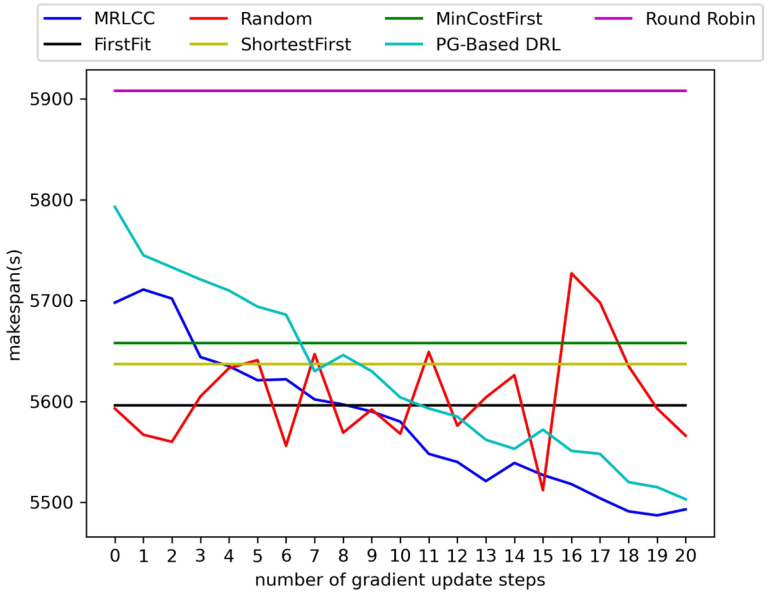
(c) scenario 3

Fig. 5 Evaluation results with new scenarios

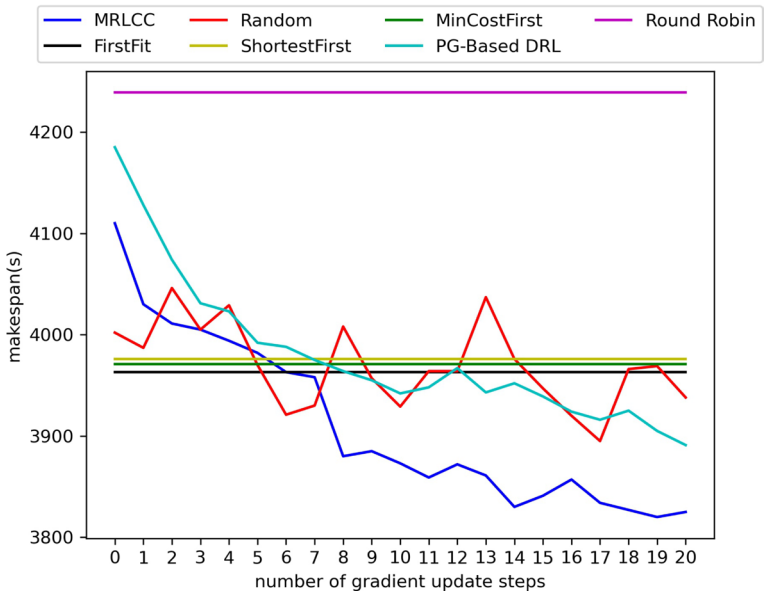
of servers in an irregular pattern. But the online service occupies the resources of the servers continuously with low utilization rate, usually the average is under 20%, which results in wasting of computing resource. So the experiment simulates the scheduling environment to solve the task scheduling problem in a scenario mixing online service with offline service. So that, we have to consider the influence of online service when the offline tasks are scheduled.

Result analysis

In the first experiment, we generate the scenarios for training using the online service data and offline service data from Azure [47] and Tencent respectively. The data of online service is clustered to 5 kinds based on the fluctuation of the utilization rate, and the tasks of offline service are also divided into 5 types based on the characteristics of the tasks. The amplitude of fluctuation represents different patterns of online service, and



(a) number of CPU is 12



(b) number of CPU is 15

Fig. 6 Evaluation results with different number of CPU

the minimum amplitude is about 2% and the maximal amplitude is about 20%. The offline tasks are clustered according to the length of execution time and number of required CPU, the execution time ranges from a few seconds to tens of seconds and number of CPU varies from 0.1 to about 4.0. Then the 5 kinds of online service and 5 kinds of offline service are combined randomly, so we can get 25 scenarios to simulate the different application preferences. We pick up 20 sets as training data sets and we randomly pick 3 sets from the left 5 sets as the testing data sets to evaluate the performance of MRLCC. So that, during training of the algorithm, we set the meta batch sizes M as 20. Then in the training process of the "inner loop", we sample 20 trajectories for a scenario to train the policy model for the specific tasks. After training, we evaluate MRLCC and the PG-based DRL method by running up to 20 policy gradient updates, the number of sample trajectory is also set as 20. The task number of the scheduling process is set as 2000, we evaluate the performance through comparing the makespan of finishing the scheduling. Figure 5 shows the performance of MRLCC and baseline algorithm for the 3 testing scenarios.

Through the Fig. 5, we can point out that MRLCC could obtain the lowest makespan after 20 gradient update steps. In the scenario 1, after 2 steps of gradient update, MRLCC could consistently perform best than all baseline algorithms. Besides, the PG-based DRL algorithm cannot adapt to new scenarios as fast as MRLCC in scenario 1 and scenario 2, which indicates that MRLCC has better ability of adaptation. Although the performance of PG-based DRL and MRLCC is similar in scenario 3, after 16 steps of gradient update, MRLCC can perform better than PG-based DRL. Among all algorithms, Round Robin has the worst performance in shortening makespan. Because the workloads are used repetitively in each step of gradient update, heuristic-methods cannot change as the deep reinforcement learning due to the specific policy. Especially, in scenario 2, *Shortest First* and *First Fit* have the same makespan, so there are six lines in the Fig. 5(b). The black line represents the performance of *Shortest First* and *First Fit*.

The second experiment aims to show the influence of resource capacity on the performance of different algorithms. We change the resource capacity of the servers to 12 and 15 respectively, and we apply the trained model to the rest two scenarios. The performance of MRLCC and other algorithms is shown in Fig. 6. Although MRLCC and PG-based DRL don't perform well at the beginning of the testing process, after a few gradient updates, MRLCC outperforms all algorithms and gets the lowest makespan. Moreover, MRLCC shows a better ability of adaptation than PG-based DRL. It is obvious that the makespan of MRLCC is shorter than PG-based DRL from the beginning of the test process.

In addition, to prove that MRLCC also improves the utilization rate of the servers, we summarize the average utilization rate of all algorithms on different testing datasets. The average utilization rate is the average utilization rate of all servers during the scheduling process. Compared to the baseline algorithms, MRLCC always gets the highest utilization rate of all testing datasets, which means that MRLCC not only can shorten the makespan of the task scheduling, but also improves the utilization rate of the servers clusters (Table 2).

Conclusion and future work

This paper proposes a MRL-based approach MRLCC, which can be applied to solve the problem of task scheduling in cloud. The learning ability of MRLCC is better than the five baseline algorithms, proving that MRLCC can adapt to new scenarios quickly within a few number of gradient updates. The comparison of latency and average utilization rate among MRLCC and other algorithm demonstrates MRLCC can get a better performance on the two aspects.

In future, we will consider to increase the features of the tasks in resource requirement, while CPU is the only

Table 2 The utilization of MRLCC and baseline algorithms in average utilization

Dataset	MRLCC	PG-based DRL	Heuristic Algorithm				
	20 steps	20 steps	FirstFit	Random	Shortest	Mincost	Round Robin
Scenario 1	0.7973	0.7891	0.7752	0.7820	0.7793	0.7692	0.7483
Scenario 2	0.8031	0.7997	0.7955	0.7981	0.7945	0.7925	0.7732
Scenario 3	0.8234	0.8175	0.8031	0.8093	0.8035	0.7972	0.7634
CPU=12	0.7719	0.7682	0.7632	0.7592	0.7521	0.7491	0.7094
CPU=15	0.8479	0.8456	0.8415	0.8439	0.8394	0.8408	0.8103

factor which is taken into account in this paper. Moreover, the training algorithm of MRLCC can be improved in several aspects. For example, we can apply other methods to accelerate the process of convergence, and MRLCC can be modified to adapt the DAG task scheduling, which is a common situation of cloud computing.

Authors' contributions
Xi Xiu wrote the main manuscript text. Weigang Wu provide guidence of this paper. All authors reviewed the manuscript. The author(s) read and approved the final manuscript.

Funding
This research is partially supported by Guangdong Provincial Natural Science Foundation of China (2018B030312002), and National Natural Science Foundation of China (U1801266, U1811461).

Availability of data and materials
The data set of Azure used in this paper is publically available at Github, and the links have been included in the paper. The data set of Tencent is not public.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 18 October 2022 Accepted: 11 April 2023
Published online: 10 May 2023

References

1. Ullah A, Nawi NM, Ouhame S (2022) Recent advancement in VM task allocation system for cloud computing: review from 2015 to 2021. Artif Intell Rev 55:1–45
2. Ferrer AJ, Marquès JM, Jorba J (2019) Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing. ACM Comput Surv (CSUR) 51(6):1–36
3. Nazir R, Ahmed Z, Ahmad Z, Shaikh N, Laghari A, Kumar K (2020) Cloud computing applications: a review. EAI Endorsed Trans Cloud Syst 6(17):e5
4. Vinothina V, Rajagopal S et al (2022) Review on mapping of tasks to resources in cloud computing. Int J Cloud Appl Comput (IJCAC) 12(1):1–17
5. Zheng B, Pan L, Liu S (2021) Market-oriented online bi-objective service scheduling for pleasingly parallel jobs with variable resources in cloud environments. J Syst Softw 176:110934
6. Song W, Xiao Z, Chen Q, Luo H (2014) Adaptive resource provisioning for the cloud using online bin packing. IEEE Trans Comput 63(11):2647–2660. https://doi.org/10.1109/TC.2013.148
7. Grandl R, Ananthanarayanan G, Kandula S, Rao S, Akella A (2014) Multi-resource packing for cluster schedulers. ACM SIGCOMM Comput Commun Rev 44(4):455–466
8. Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: Fair allocation of multiple resource types. In: Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11). USENIX Association, Boston
9. Xie Y, Sheng Y, Qiu M, Gui F (2022) An adaptive decoding biased random key genetic algorithm for cloud workflow scheduling. Eng Appl Artif Intell 112(104):879
10. Ajmal MS, Iqbal Z, Khan FZ, Ahmad M, Ahmad I, Gupta BB (2021) Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers. Comput Electr Eng 95(107):419
11. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM workshop on hot topics in networks. Association for Computing Machinery, New York, pp 50–56
12. Lee H, Cho S, Jang Y, Lee J, Woo H (2021) A global dag task scheduler using deep reinforcement learning and graph convolution network. IEEE Access 9:158548–158561
13. Peng Z, Cui D, Zuo J, Li Q, Xu B, Lin W (2015) Random task scheduling scheme based on reinforcement learning in cloud computing. Clust Comput 18(4):1595–1607
14. Sohn S, Woo H, Choi J, Lee H (2020) Meta reinforcement learning with autonomous inference of subtask dependencies. arXiv preprint arXiv:2001.00248
15. Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the International conference on machine learning, PMLR, pp 1126–1135
16. Pong VH, Nair AV, Smith LM, Huang C, Levine S (2022) Offline meta-reinforcement learning with online self-supervision. In: International Conference on Machine Learning, PMLR, pp 17811–17829
17. Wen S, Wen Z, Zhang D, Zhang H, Wang T (2021) A multi-robot path-planning algorithm for autonomous navigation using meta-reinforcement learning based on transfer learning. Appl Soft Comput 110:107605
18. Chen L, Hu B, Guan ZH, Zhao L, Shen X (2021) Multiagent meta-reinforcement learning for adaptive multipath routing optimization. IEEE Trans Neural Netw Learn Syst 33(10):5374–5386
19. Kim DK, Liu M, Riemer MD, Sun C, Abdulhai M, Habibi G, Lopez-Cot S, Tesauro G, How J (2021) A policy gradient algorithm for learning to learn in multiagent reinforcement learning. In: International Conference on Machine Learning, PMLR, pp 5541–5550
20. Pradhan P, Behera PK, Ray B (2016) Modified round robin algorithm for resource allocation in cloud computing. Procedia Comput Sci 85:878–890
21. Chen T, Marques AG, Giannakis GB (2016) Dglb: Distributed stochastic geographical load balancing over cloud networks. IEEE Trans Parallel Distrib Syst 28(7):1866–1880
22. Ghobaei-Arani M, Souri A (2019) Lp-wsc: a linear programming approach for web service composition in geographically distributed cloud environments. J Supercomput 75(5):2603–2628
23. Ghobaei-Arani M, Jabbehdari S, Pourmina MA (2016) An autonomic approach for resource provisioning of cloud services. Clust Comput 19:1017–1036
24. Basu D, Wang X, Hong Y, Chen H, Bressan S (2019) Learn-as-you-go with megh: Efficient live migration of virtual machines. IEEE Trans Parallel Distrib Syst 30(8):1786–1801. https://doi.org/10.1109/TPDS.2019.2893648
25. Kumar M, Sharma SC (2018) Pso-cogent: Cost and energy efficient scheduling in cloud environment with deadline constraint. Sustain Comput Inform Syst 19:147–164
26. Jin HZ, Yang L, Hao O (2015) Scheduling strategy based on genetic algorithm for cloud computer energy optimization. In: Proceedings of the 2015 IEEE International Conference on Communication Problem-Solving (ICCP), IEEE, pp 516–519
27. Medara R, Singh RS et al (2021) Energy-aware workflow task scheduling in clouds with virtual machine consolidation using discrete water wave optimization. Simul Model Pract Theory 110:102323
28. Ding D, Fan X, Zhao Y, Kang K, Yin Q, Zeng J (2020) Q-learning based dynamic task scheduling for energy-efficient cloud computing. Futur Gener Comput Syst 108:361–371
29. Dong T, Xue F, Xiao C, Li J (2020) Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurr Comput Pract Experience 32(11):5654
30. Yan J, Huang Y, Gupta A, Gupta A, Liu C, Li J, Cheng L (2022) Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach. Comput Electr Eng 99:107688
31. Cheng F, Huang Y, Tanpure B, Sawalani P, Cheng L, Liu C (2022a) Cost-aware job scheduling for cloud instances using deep reinforcement learning. Clust Comput 25:1–13
32. Cheng L, Kalapgar A, Jain A, Wang Y, Qin Y, Li Y, Liu C (2022) Cost-aware real-time job scheduling for hybrid cloud using deep reinforcement learning. Neural Comput & Applic 34(21):18579–18593
33. Wei Y, Pan L, Liu S, Wu L, Meng X (2018) Drl-scheduling: An intelligent qos-aware job scheduling framework for applications in clouds. IEEE Access 6:55112–55125. https://doi.org/10.1109/ACCESS.2018.2872674

34. Huang Y, Cheng L, Xue L, Liu C, Li Y, Li J, Ward T (2021) Deep adversarial imitation reinforcement learning for qos-aware cloud job scheduling. IEEE Syst J 16(3):4232–4242
35. Cao Z, Zhou P, Li R, Huang S, Wu D (2020) Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. IEEE Internet Things J 7(7):6201–6213
36. Guo W, Tian W, Ye Y, Xu L, Wu K (2020) Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet Things J 8(5):3576–3586
37. Liu N, Li Z, Xu J, Xu Z, Lin S, Qiu Q, Tang J, Wang Y (2017) A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: Proceedings of the IEEE 37th international conference on distributed computing systems (ICDCS), IEEE, pp 372–382
38. Xu Z, Wang Y, Tang J, Wang J, Gursoy MC (2017) A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans. In: 2017 IEEE International Conference on Communications (ICC), IEEE, pp 1–6
39. Li Y (2017) Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274
40. Sigaud O, Buffet O (2013) Markov decision processes in artificial intelligence. John Wiley & Sons
41. Fakoor R, Chaudhari P, Soatto S, Smola AJ (2019) Meta-q-learning. arXiv preprint arXiv:1910.00125
42. Huang L, Zhang L, Yang S, Qian LP, Wu Y (2020) Meta-learning based dynamic computation task offloading for mobile edge computing networks. IEEE Commun Lett 25(5):1568–1572
43. Lin J, Peng Z, Cui D (2018) Deep reinforcement learning for multi-resource cloud job scheduling. In: Proceedings of the International conference on neural information processing, Springer, pp 289–302
44. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3):279–292
45. Ye Y, Ren X, Wang J, Xu L, Guo W, Huang W, Tian W (2018) A new approach for resource scheduling with deep reinforcement learning. arXiv preprint arXiv:1806.08122
46. Nichol A, Achiam J, Schulman J (2018) On first-order meta-learning algorithms. arXiv preprint arXiv:1803.02999
47. Romero F, Chaudhry GI, Goiri Í, Gopa P, Batum P, Yadwadkar NJ, et al (2021) Faa$T: A transparent auto-scaling cache for serverless applications. Proceedings of the ACM Symposium on Cloud Computing. Association for Computing Machinery, New York, 122–137

Publisher's Note