Pattern Recognition 121 (2022) 108254

Contents lists available at ScienceDirect

Pattern Recognition

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

Learning to schedule multi-NUMA virtual machines via reinforcement learning

Junjie Sheng[a,1], Yiqiu Hu[b,1], Wenli Zhou[c], Lei Zhu[d], Bo Jin[e], Jun Wang[f], Xiangfeng Wang[g,*]

[a] School of Computer Science and Technology, East China Normal University, Shanghai, China
[b] Software Engineering Institute, East China Normal University, Shanghai, China
[c] Algorithm Innovation Lab, Huawei Cloud Huawei Technologies Co., Ltd. Xi'an, China
[d] Alkaid Lab, Huawei Cloud Huawei Technologies Co., Ltd. Xi'an, China
[e] School of Computer Science and Technology, East China Normal University and Shanghai Research Institute for Intelligent Autonomous Systems, Shanghai, China
[f] School of Computer Science and Technology, East China Normal University, Shanghai, China
[g] School of Computer Science and Technology, East China Normal University and Shanghai Research Institute for Intelligent Autonomous Systems, Shanghai, China

ARTICLE INFO

Article history:
Received 30 March 2021
Revised 10 August 2021
Accepted 11 August 2021
Available online 13 August 2021

Keywords:
Dynamic virtual machine scheduling
Multi-NUMA
Reinforcement learning
Cloud computing

ABSTRACT

With the rapid development of cloud computing, the importance of dynamic virtual machine scheduling is increasing. Existing works formulate the VM scheduling as a bin-packing problem and design greedy methods to solve it. However, cloud service providers widely adopt multi-NUMA architecture servers in recent years, and existing methods do not consider the architecture. This paper formulates the multi-NUMA VM scheduling into a novel structured combinatorial optimization and transforms it into a reinforcement learning problem. We propose a reinforcement learning algorithm called SchedRL with a delta reward scheme and an episodic guided sampling strategy to solve the problem efficiently. Evaluating on a public dataset of Azure under two different scenarios, our SchedRL outperforms FirstFit and BestFit on the fulfill number and allocation rate.

© 2021 Elsevier Ltd. All rights reserved.

1. Introduction

Cloud computing services have become significantly popular in recent years. With the development of cloud computing technologies, infrastructure as a service (IaaS) is welcomed as the most basic service scheme. Leading cloud service providers, e.g., AWS, Azure Alicloud, and HuaweiCloud, usually supply virtual machines (VMs) for users to fulfill their computation requests (CPU, memory, etc.) as shown in Fig. 1. The VMs are generated from a number of servers (cluster). Deciding which server to schedule the VM (VM scheduling) is a challenging problem, although variant scheduling schemes can be employed accordingly. With a better scheduler, the server cluster can process more requests, leading to higher profit for providers. As a result, how to design an efficient scheduler has attracted momentous interests, especially for commercial companies.

In cloud computing, the business model is "pay as you use". The customers will delete the VM as long as they finish their tasks to reduce the cost, resulting in that nearly no prior information can be leveraged to predict future requests. Once a creation/deletion request arrives, the cluster will create/delete a corresponding VM from a server. Existing works formulate the VM scheduling problem as the classical dynamic vector bin packing problem (i.e., DVBPP) [1–3] which has been studied for decades [4]. In the DVBPP formulation, the VM and the physical servers are denoted as the items and bins. The corresponding vector in DVBPP represents computation resources, such as CPU, memory, network, etc. Existing works often adopt greedy algorithms to solve the DVBPP [5–8].

However, these existing methods do not explicitly consider the server architecture, leading to incompatibility with the practical scheduling scenario. Greedy methods score on a single server instead of taking the whole resource pool's state into consideration, which leads to significant challenges to reach global optimality. Recently, cloud service providers adopt multi Non-Uniform Memory Access (NUMA) [9] architecture to provide large size

* Corresponding author.
E-mail addresses: 52194501003@stu.ecnu.edu.cn (J. Sheng), 51184501015@stu.ecnu.edu.cn (Y. Hu), zhouwenli@huawei.com (W. Zhou), zhulei41@huawei.com (L. Zhu), bjin@cs.ecnu.edu.cn (B. Jin), jwang@cs.ecnu.edu.cn (J. Wang), xfwang@cs.ecnu.edu.cn (X. Wang).
[1] Equal contribution.

https://doi.org/10.1016/j.patcog.2021.108254
0031-3203/© 2021 Elsevier Ltd. All rights reserved.

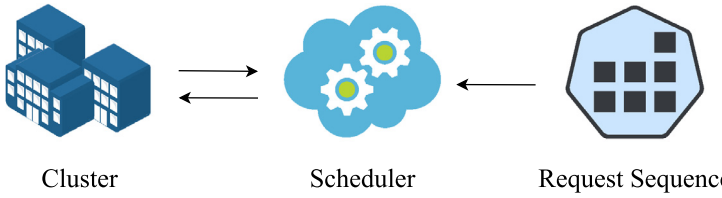
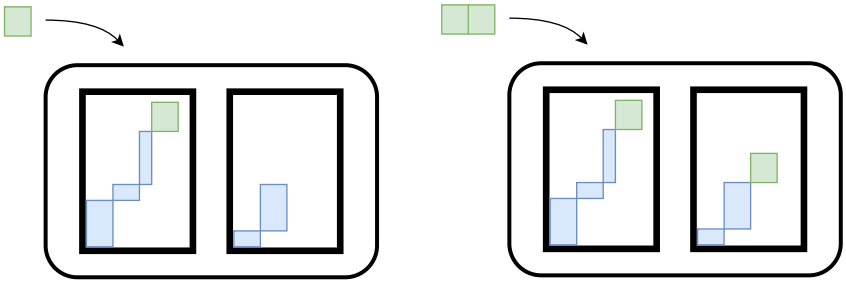


Fig. 1. Virtual machine scheduling procedure. Every time the scheduler receives a request from the request sequence, it chooses a server from the cluster to create or delete a VM.



(a) Small VM scheduling (b) Large VM scheduling

Fig. 2. The scheduling illustration for a two-NUMA server. The outer black boxes in Fig. 2(a) and (b) denote the server. The inner black boxes denote the NUMA units. The green rectangle describes the VM. In Fig. 2(a), the server schedules a small VM on the first NUMA. For a large VM in Fig. 2(b), the server divides it equally and schedules them on both NUMAs, respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

(memory, cpu) VMs. Every server has multiple NUMAs, and the NUMA here is denoted as an entity with several CPU and memory resources. When it comes to a small VM creation request, the service providers create the VM on one of the NUMA of the server. However, when there comes a large VM creation request, one NUMA's resource maybe not enough for the VM, and the VM has to be created across different NUMAs. Take a two-NUMA server as an example (as shown in Fig. 2). The small VM is created on the left NUMA of the server, while the large VM is divided equally and created across the NUMAs of a server as in Rao et al. [10]. In this case, the advanced multi-NUMA architecture could lead to a new combinatorial optimization problem compared with DVBPP and Huawei [11] released related issues to discuss it. The different but challenging allocation mechanism compared with previous DVBPP motivates us to propose new formulations and design more efficient schedulers for multi-NUMA VM scheduling. The data-driven method instead of the greedy type methods could be introduced to handle the challenging allocation mechanism. The data-based method's critical advantage is that it allows the scheduler to handle the online and real-time scheduling. However, the expert data/optimal label of the scheduling is hard to obtain, which prevents us from supervised learning.

To overcome the difficulties mentioned above, we formulate the VM scheduling problem into a reinforcement learning framework, as shown in Fig. 3. The scheduler takes the cluster status and incoming request as the observation. For each observation, it selects an action to decide which server or NUMA to schedule the request. The objective of the scheduler is to maximize the number of created VMs during scheduling. This framework frees us from the expert data and enables online scheduling. However, using the number of created VMs as the reward signal leads us to reinforcement learning with sparse rewards[2], leading to low sample efficiency and inefficient learning [12]. To address the issue about sparse rewards, we propose an efficient RL-based multi-NUMA VM scheduling algorithm called **SchedRL**. Specifically, it devises two techniques to handle the sparse reward issue: 1)delta reward scheme, 2)the episodic guided sampling strategy. The delta reward scheme directly measures the influence of current scheduling and provides a dense reward signal for the scheduler. The episodic guided sampling strategy utilizes the experienced samples to form a guided policy to exploit experienced samples and improve the sample efficiency.

We conduct comparative experiments on two different scenarios based on a public dataset of Azure [13], which shows that SchedRL can outperform widely adopt baselines like FirstFit as well as BestFit on both allocation rate and fulfill number. The overall contributions of this paper can be summarized as follows:

1. For multi-NUMA VMs scheduling, we introduce a new optimization formulation to model the dynamic VM scheduling procedure, which can be considered as a structured combinatorial optimization problem;
2. Instead of directly solving the abstract intricate combinatorial optimization problem, we formulate it into a reinforcement learning framework. An algorithm (SchedRL) using two techniques (delta reward scheme and episodic guided sampling strategy) is proposed to solve it efficiently.
3. The SchedRL is evaluated on a public dataset of Azure under two different scenarios, which shows the efficiency of SchedRL compared with FirstFit and BestFit on both allocation rate and fulfill number.

This paper is organized as follows. In Section 2 we provide a brief introduction to related works in VM scheduling, especially on machine learning algorithms. We provide the background material on Markov decision progress (MDP) and Deep Q Network (DQN) in Section 3. The proposed new formulation and RL-based SchedRL algorithm are presented in Section 4. Extensive experiments are presented in Section 5, and we conclude this paper in Section 6.

[2] Agents interact with environment and obtains a series reward signal. If most of them are non-positive, we call it sparse reward else it is a dense reward scenario.

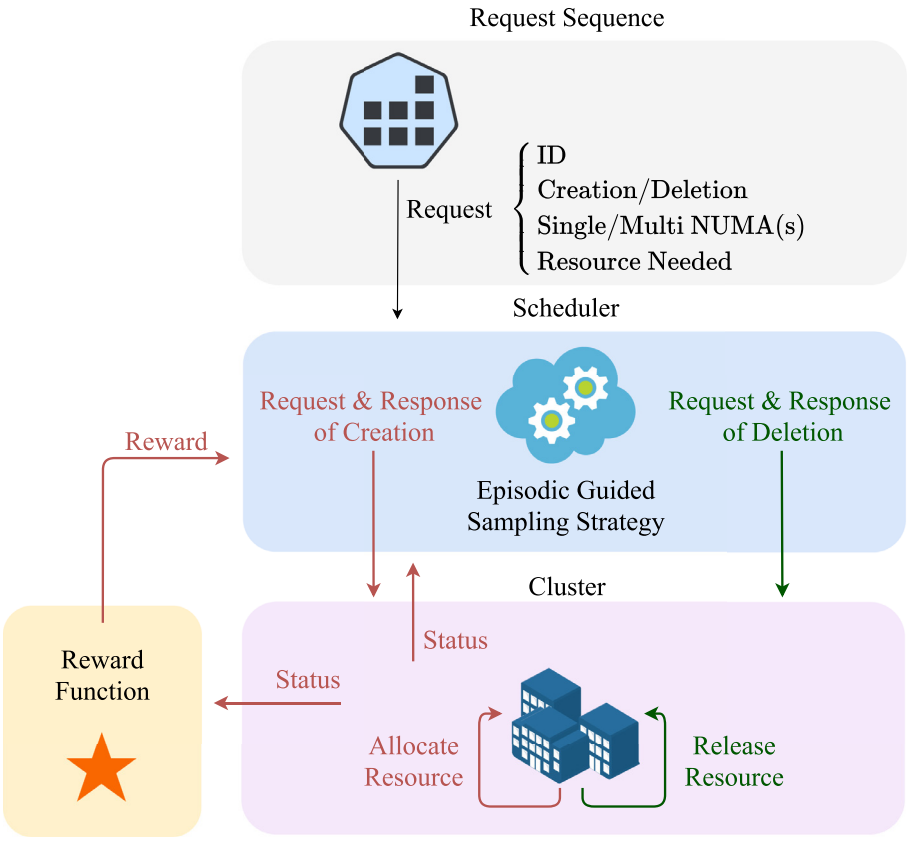


Fig. 3. The flowchart of the SchedRL procedure. At every step, the scheduler receives the cluster status and the incoming request. It uses the episodic guided sampling strategy to select which server or NUMA to schedule the request as a response. After that, the cluster allocates or releases resources based on the response and sends a reward signal to the scheduler. This procedure is iteratively executed until termination.

2. Related work

We focus on designing RL based algorithm for VM scheduling problems. In the following, we will review the related works from three aspects, i.e., dynamic VM scheduling problem, machine learning for combinatorial optimization, and reinforcement learning for scheduling

2.1. Dynamic VM scheduling problem

The classical dynamic VM scheduling problem is typical in traditional VM scheduling applications. Greedy algorithms were usually proposed to solve dynamic VM scheduling problems with competitive ratio analysis. Stolyar [5] assumed that the VM creation requests could be modeled as a Poisson process while the lifetime of VM is modeled to follow a fixed exponential distribution. A greedy algorithm was employed and achieved the asymptotically optimal solution with theoretical solution quality analysis. Li et al. [6] introduced a new goal called MinUsageTime into the formulation, which denotes the total usage time of all the used bins and is gained significant attention. Li et al. [7] developed a hybrid FirstFit algorithm with an improvement on competitive ratio. Azar and Vainstein [14] proposed a method which combined the FirstFit and classify-by-duration strategies. They also mathematically proved that their method could improve the upper bound of competitive ratio. Unfortunately, all these work are greedy based, which can not make use of experience or potential information of the request sequence.

Some recent works solved the dynamic VM scheduling problem by combining VM migration. Li et al. [15] designed linear integer programming via the VM migration and evaluated it in commercial experiments. Ahn et al. [16] introduced a live VM migration method to decrease the conflicts among resources and evaluate NUMA enabled. Rao et al. [17] minimized the system-wide uncore penalty with Bias Random CPU Migration (BRM) algorithm. Jiankang et al. [1] proposed a two-stage heuristic algorithm to create the VM efficiently and migrate to fine-tune. Their method considered both resource utilization of physical servers and networks. Shi et al. [3] defined some VM type patterns in advance, which are made in scheduling and migrating.

2.2. Machine learning for combinatorial optimization

The problem considered in this paper can be formulated as a combinatorial optimization problem, which has raised significant research interest [18]. Solving combinatorial optimization problems end-to-end by machine learning techniques has been popularly discussed in recent years [19–21]. Traveling salesman problem (TSP) attracts the most attention in classical combinatorial optimization (CO) problems. Vinyals et al. [19] creatively designed the pointer network, which is trained in a supervised learning scheme. However, the optimization solutions are usually extremely difficult to obtain, Bello et al. [20], Nazari et al. [21] combined the pointer network with reinforcement learning. The pointer network is employed as the actor in A2C or A3C algorithm [22]. Further, Deudon et al. [23], Kool et al. [24] introduced the attention mechanism with better performance improvement. Hu et al. [25] brought up a new bin packing problem, which involves only one single bin. For a set of items that were going to pack in a bin according to the order, the agent needs to decide how to rotate the item so that the wrapping had the smallest surface area when the packing is done. The authors employed the pointer network as the policy network, which was trained through policy gradient. Duan et al. [26] extended the prior work by combining PPO and supervised learning

to learn the orientations and sequence order. Further, Laterre et al. [27] formulated the BPP as a single-player game and employed neural MCTS to construct the solution. Zhao et al. [28] trained a robotic agent to pick and pack items on a conveyor belt into a single bin properly using the reinforcement learning method under the actor-critic framework. Machine learning techniques can be utilized to configure parameters or offer some extra information for existing operations research tools. Kruber et al. [29] combined supervised learning method with Dantzig-Wolf method on solving MILP. Bonami et al. [30] employed a machine learning model to decide whether linearizing the problem with QP B&B or not. Mahmood et al. [31] produced candidates like GAN [32] and further be refined by a combinatorial optimization algorithm.

2.3. Reinforcement learning for scheduling

Reinforcement Learning is emerging in many areas [33–37]. Specifically for scheduling problems, RL has been tentatively used to solve scheduling problems. These works target to schedule the workloads for large-scale distributed computing cluster like Spark. Mao et al. [38] developed a DeepRM system, which can pack tasks with multiple resource demands by policy gradient method. Li and Hu [39] proposed reinforcement learning based DeepJS to minimize the makespan of a job set also by policy gradient method. Mao et al. [40] built a job scheduler named DECIMA for distribute computation cluster, while the actor-critic method can be employed with input-dependent baselines. Bao et al. [41] proposed a machine learning task placement model together with an auxiliary revenue prediction model Harmony to place predicted benefits for unseen tasks. Zhang et al. [42] solved CPU-GPU heterogeneous computing scheduling problem with deep Q-learning according to the running state and task characteristics of the cluster environment. Our work differs them from two perspectives: 1) virtual machine versus workload; 2) NUMA structure. Existing RL for scheduling methods schedules the workload, which assumes the knowledge of how long the jobs last. However, it is hard to access and predict how long a user will use the VM for the VM scheduling. This requires the learning methods to capture the duration of VMs exist implicitly. Moreover, our work considers the multi-NUMA structure of servers, making the scheduling strategy more complex (dynamic action space).

The above literature reviews include the current research status of dynamic VM scheduling problems (to illustrate the current status and deficiencies of existing research in the modeling and solving methods of VM scheduling problems), the machine learning for combinatorial optimization (to show the efficiency of machine learning for CO) and the specific reinforcement learning for scheduling. However, how to use the reinforcement learning technique to solve the (multi-NUMA) VM scheduling problem is still unexplored.

3. Preliminaries

3.1. Markov decision process

We first model the dynamic virtual machine scheduling problem under the Markov decision process (MDP) framework. Markov decision process can be defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ with $\mathcal{S}$ denotes the state space; $\mathcal{A}$ denotes the action space; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denotes the transition function, which specifies the probability of next state $s' \in \mathcal{S}$ for given current state $s \in \mathcal{S}$ and $a \in \mathcal{A}$; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function; γ denotes the discount factor for calculating the cumulative return. The policy $\pi : \mathcal{S} \to \mathcal{A}$ for the agent need to be learned from the MDP system. The agent executes an action according to the observed current state s and policy π at each step. Then the environment is changed based on $\mathcal{T}$. The agent can obtain a reward signal and execute a new action according to the new state until the process is done. With the trajectory sampled, we can calculate the cumulative return G_t for each time step t by:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1}$$

where r_{t+k+1} denotes the reward obtained at step $t + k + 1$. In order to learn the optimal policy π^*, the goal is to maximize the expected cumulative return.

3.2. Deep Q-Learning

In the learning procedure, the policy is evaluated based on the state-action value function $Q_\pi(s, a)$, i.e.,

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}\Big[G_t \;\Big|\; s_t = s, a_t = a\Big] \\ &= \mathbb{E}\Big[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\Big|\; s_t = s, a_t = a\Big] \\ &= \mathbb{E}\Big[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) \;\Big|\; s_t = s, a_t = a\Big]. \end{aligned} \tag{2}$$

By maximizing the state-action value function over all policies, we can obtain the optimal state-action value function $Q^*(s, a)$, i.e.,

$$Q^*(s,a) = \max_\pi Q_\pi(s,a). \tag{3}$$

Based on the obtained $Q^*(s, a)$, we can directly calculate the optimal policy π^* by

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{4}$$

The update of Q function needs to be iteratively calculated in order to approximate the optimal $Q^*(s, a)$, while the scheme is called Q-learning. The deep Q-learning algorithm(DQN) employs a deep neural network to play the role of Q function, which can be denoted as the Q-network $Q_\theta(s, a)$ with θ being the parameters. With the trajectory experiences $\{\langle s, a, r, s' \rangle\}$ which is collected from the interactions between the agent and environment, the $Q_\theta(s, a)$ can be updated iteratively by

$$\min_\theta \mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}\Big[\|r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)\|^2\Big]. \tag{5}$$

where D is a replay buffer that stores the experience of the agent. Actually, the $Q_\theta(s, a)$ is unstable because of the constantly updating and problem (5) is difficult to solve. To address this problem, a replica of $Q_\theta(s, a)$ is introduced which is called target Q network and denoted as $Q_{\theta'}(s, a)$. Problem (5) can be modified to be

$$\min_\theta \mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}\Big[\|r + \gamma \max_{a'} Q_{\theta'}(s', a') - Q_\theta(s, a)\|^2\Big]. \tag{6}$$

During the updating the $Q_\theta(s, a)$, $Q_{\theta'}(s, a)$ is copied from $Q_\theta(s, a)$ every fixed number steps. A double deep Q-network is designed to improve the state-action value overestimation problem [43], i.e., a predict network $\hat{Q}(s, a)$ is further introduced to the model

$$\min_\theta \mathcal{L}(\theta) = \mathbb{E}\Big[\|r + \gamma Q_\theta(s', \arg\max_{a'} \hat{Q}(s', a') - Q_\theta(s, a)\|^2\Big]. \tag{7}$$

As a result in double deep Q-learning method, the chosen action might be different with the action obtained by $Q_\theta(s, a)$.

4. The SchedRL algorithm

This section aims to cast the virtual machine scheduling problem into the reinforcement learning framework and describe the SchedRL. We first describe the prototypical problem and give the combinatorial optimization formulation of multi-NUMA VM scheduling in Section 4.1. Then the Sections 4.2, 4.3 and 4.4 present the details of reinforcement learning environment. Finally, the algorithmic architecture, sampling, and training procedure are given in Sections 4.5 and 4.6.

4.1. Prototypical problem

Suppose the cloud service provider has N servers, and each server has M NUMA. At the time t, it receives a request to allocate or release some resources. The overall requests can be categorized as creation requests and deletion requests. For creation requests, the provider needs to generate a VM on a server to fulfill the request. The multi-NUMA architecture allows allocation based on the VM size: A small size VM needs to be generated on a single NUMA of a server while a large one needs to be divided equally and created on the two NUMAs of a server. As for deletion requests, the provider releases the corresponding request resources from the specified server. The scheduler's key responsibility is to select the server or the NUMA to handle the creation requests, and the main goal is to maximize the possible serving requests for a cluster with finite servers.

At the beginning, we denote the whole cluster as $s : N \times M \times d$. It has N servers, each server has M NUMAs and the ith server's jth NUMA's remaining computation resource is $s(i, j, \cdot) \in \mathbb{R}^d$ (d denotes the resource feature dimension). This cluster needs to handle a sequence of requests, i.e., $\text{req} \in \mathbb{R}^{T\times(3+d)}$. T denotes the number of requests which need to be served, and the tth column $\text{req}(t, \cdot) \in \mathbb{R}^{3+d}$ denotes the request at time t. The first 3 dimensions of the request vector $\text{req}(t, \cdot)$ denote the VM ID, the type of request(creation or deletion) and the type of VM (single or multi NUMAs) respectively. In details, $\text{req}(t, 1) \in \mathbb{R}$ denotes the VM ID; $\text{req}(t, 2) \in \{0, 1\}$ denotes the type of request, where 0 and 1 mean creation request and deletion request respectively; $\text{req}(t, 3) \in \{0, 1\}$ denotes the type of VM, 0 and 1 mean single NUMA VM and multi NUMAs VM respectively. The other d dimensions of $\text{req}(t, [4 : 3 + d]) \in \mathbb{R}^d$ denotes the resources it required. At time t, the scheduler needs to handle the request $\text{req}(t, \cdot)$, while we introduce a corresponding matrix $\text{res} \in \mathbb{R}^{T\times 2}$ to model the response correspondence. $\text{res}(t, 1) \in \{1, \cdots, N\}$ and $\text{res}(t, 2) \in \{1, \cdots, M + 1\}$ denote the server number and the NUMA number of the request's VM at time t. To emphasize, $\text{res}(t, 2) = M + 1$ means that the VM of the request is divided equally to be created on all the NUMAs of the specific server[3]. Furthermore, in order to describe whether a request $\text{req}(t, \cdot)$ is responded successfully or not, we introduce a vector $w \in \mathbb{R}^T$ with $w(t) \in \{0, 1\}$. $w(t) = 0$ denotes that the request is not responded to and otherwise is responded. Although w has close relationship with res, but we still need to calculate the optimal solution of w.

Before giving the overall formulation, we will discuss three kinds of constraints that the decision variables res and w need to be satisfied, i.e., the spatial constraint, the consistent constraint, and the temporal constraint. The spatial constraint means that at any time $t \le T$, none of the NUMA has a negative amount of resources. The computational resource $s(i, j, \cdot)$ is non-negative after allocating the VMs and adding the resources from the released VMs at any time t. The resources of the created VMs and deleted VMs are summarized as $a(t, i, j)$ and $b(t, i, j)$ respectively, i.e.,

$$\begin{cases} a(t,i,j) = & \sum_{k=1}^{t} \big(\mathbb{1}_{\text{req}(k,2)=1,\text{req}(k,3)=0,\text{res}(k,\cdot)=(i,j)} \\ & \quad + \frac{1}{M} \mathbb{1}_{\text{req}(k,2)=1,\text{req}(k,3)=1,\text{res}(k,\cdot)=(i,M+1)} \big) \\ & \quad \cdot \text{req}(k, [4 : d + 3]), \\ b(t,i,j) = & \sum_{k=1}^{t} \big(\mathbb{1}_{\text{req}(k,2)=0,\text{req}(k,3)=0,\text{res}(k)=(i,j)} \\ & \quad + \frac{1}{M} \mathbb{1}_{\text{req}(k,2)=0,\text{req}(k,3)=1,\text{res}(k)=(i,M+1)} \big) \\ & \quad \cdot \text{req}(k, [4 : d + 3]). \end{cases} \tag{8}$$

The spatial constraint can be formulated as

$$\text{res} \in \mathcal{C}_s, \quad \mathcal{C}_s := \left\{ \text{res} \;\middle|\; \begin{array}{ll} s(i,j,\cdot) & -a(t,i,j) + b(t,i,j) \succeq 0, \\ & \forall\, 0 \le i \le N, \forall\, 0 \le j \le M, \forall\, t \le T \end{array} \right\}. \tag{9}$$

The consistent constraint requires that the deleted and created location of the same VM should be the same, i.e.,

$$\text{res} \in \mathcal{C}_c, \;\; \mathcal{C}_c := \Big\{ \text{res} \;\Big|\; \text{res}(t,\cdot) = \text{res}(t',\cdot), \;\; \text{if } \text{req}(t,1) = \text{req}(t',1), \;\; \forall\, t, t' \le T \Big\}. \tag{10}$$

The temporal constraint describes the online characteristic of the problem: once a request is not responded to (no server has enough resource to create VM for the request), the environment is terminated, and no requests will be handled after that. Finally, we can formulate the whole procedure as a constrained optimization problem as follows:

$$\begin{aligned} \max_{\text{res},w} \quad & \|w\|_0, \\ \text{s.t.} \quad & \text{res} \in \mathcal{C}_s \cap \mathcal{C}_c, \\ & w(1) = \begin{cases} 0, & \text{if } \text{res}(1) = (0,0), \\ 1, & \text{otherwise}, \end{cases} \\ & w(t) = \begin{cases} 0, & \text{if } \text{res}(t) = (0,0) \\ & \quad \text{or } w(t-1) = 0, \\ 1, & \text{otherwise}, \end{cases} \quad \text{for } 2 \le t \le T. \end{aligned} \tag{11}$$

Directly solving the constrained optimization problem is significantly difficult. As a result, we solve it by utilizing the machine learning technique and transform it into an RL framework. In the following subsections, we will properly define the observation space, action space, transition design, and reward design respectively of the MDP system. All the notations are listed in Table 1.

Table 1
Notations of problem setting.

Notation	Explanation
f_t	The creation request's resources information at time t.
f_t^d	The sequence of delete requests' resources information after f_t.
Seqs_t	The request sequence(f_t, f_t^d, f_{t+1}).
$o_t^{i,j}$	The ith server's jth NUMA's resources information at time t.
o_t	The cluster's resources status at time t.
s_t	The state of the scheduler's environment which contains o_t and f_t.
a_t	The action of the scheduler at time t.
m	The action mask function for valid action choosing $s_t \times \mathcal{A} \to \mathcal{A}$.
Γ	The transition function of the environment($s_t \times \text{Seqs}_t \times a_t \to s_{t+1}$).
r	The reward function of the environment($s_t \times a_t \to \mathcal{R}$).
c	The threshold to spanning across two NUMAs.

4.2. Observation space

For the virtual machine scheduling problem, the scheduler is able to access both the current request information $f_t \in \mathbb{R}^2$ and the overall cluster's current status o_t (the status of all the N physical servers). The status of each physical server includes M NUMA, and each NUMA is composed of the remaining computation resources. Further in this paper, only two key factors, i.e., the remain CPU and memory, are taken into consideration as the resource features. These two factors (CPU and memory) are both represented by continuous scalars. For a cluster with N physical servers, its status representation can be denoted as $o_t := [(o_t^{1,1}, \ldots, o_t^{1,M}), \ldots, (o_t^{N,1}, \ldots, o_t^{N,M})] \in \mathbb{R}^{NM\times 2}$ with $o_t^{i,j} \in \mathbb{R}^2$ denoting the remaining CPU and memory of the ith server jth NUMA.

All the requests can be divided into two categories: creation requests and delete requests. The request space is set to be $\mathcal{F}$. The creation request aims to apply the needed CPUs and memories to allocate on physical servers, which is denoted as $f_t : (\text{c}_t, \text{m}_t) \in \mathcal{F}$. The delete request aims to release the corresponding CPUs and memories. It should be noted that the scheduler need not take action for those delete requests. As a result, the state representation

[3] In practice, a multi-NUMA request may only need part of NUMAs in the server. We here simplify the problem by only allowing all NUMAs allocation.

of the scheduler only contains the creation requests. The observation of the scheduler at time t then can be set to the combination $[f_t, o_t] \in \mathbb{R}^{(NM+1)\times 2}$.

4.3. Action space and transition design

The scheduler needs to select which physical server or NUMA index to create VM for the current request. However, for different types of requests, the behaviors of the scheduler are different. For instance, the scheduler selects a specific NUMA to respond to single-NUMA request cases, while it only needs to select the server to create VM for multi-NUMA requests. The action space (discrete) for the scheduler is set to be $\{1, \ldots, NM\}$ to represent both the server and NUMA indexes. In the training procedure, the corresponding actions are sampled from valid actions with an available mask($m : s_t \times \mathcal{A} \to \mathcal{A}$), which indicates that the NUMA with enough CPUs and memories will handle the current states.

When the environment received a valid action a_t at current state $s_t : [f_t, o_t]$, the scheduler will transit to the next state $s_{t+1} : [f_{t+1}, o_{t+1}]$ based on a transition function Γ. We assume that two functions are employed to establish the transition, i.e., the creation handler $\mathbf{h}^c$ and the delete handler $\mathbf{h}^d$, which will simulate the allocate and release process, respectively. The Γ firstly handle the creation request

$$\hat{o}_{t+1} = \mathbf{h}^c(o_t, f_t, a_t),$$

then further handle all the delete requests until a creation request

$$o_{t+1} = \mathbf{h}^d(\hat{o}_{t+1}, f_t^d),$$

while the next state is established at $s_{t+1} = (o_{t+1}, f_{t+1})$.

4.4. Reward design

The goal of the scheduler is to handle as many as possible creation requests. As a result, the most simple reward setting is to obtain $+1$ for every successful creation. This simple case is denoted as **identical reward**, i.e., $r = +1$. However, the identical reward setting treats all the creation equally, which will make the learning process more difficult. A novel reward scheme called δ**-reward** is introduced to measure the allocation's influence on the whole resource pool. The scores before and after each allocation can be calculated, and the absolute difference value is denoted as δ. Moreover, only the allocated server's score will change. This leads us to efficient reward calculation. The scheduler's objective can then be translated to select actions to achieve minimal accumulative cluster influence. We design the corresponding reward function as

$$r := \frac{C - \delta}{C},$$

with a large enough normalization scalar C.

4.5. Algorithmic architecture

This section describes the network architecture of our SchedRL. We adopt deep Q-network with dueling network architecture [44] to represent the scheduler's policy. The Q-network takes cluster status and request information as input. Then it flattens and concatenates them as a vector. The fully connected neural networks are employed to establish the Q-network with parameters as θ. This Q-network outputs the current state value $V(s;\theta)$ and the action's advantage value[4] $A(s,a;\theta)$. The $Q(s,a)$ value can be calculated as

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|A|}\sum_{a'} A\big(s,a';\theta\big)\right). \tag{12}$$

[4] The value that obtained by taking action a over $V(s)$.

4.6. Sampling and training procedure

This section states the sampling and training procedure of SchedRL. In each scheduling step, the scheduler receives the state information s_t and needs to select an action. A popular way to explore in RL is ϵ-greedy exploration which adopt a random action at a probability. However the random exploration is often highly inefficient and may fail in some cases, one sound way to gain more efficient exploration is incorporating expert policy to guide the exploration [12]. As shown in (13), the scheduler will have $\epsilon \in (0,1)$ probability to select the greedy action.

$$a_t = \begin{cases} \arg\max_{a' \in m(s_t,\mathcal{A})} Q(s_t, a'), & \text{with probability } \epsilon, \\ \text{expert}(s_t), & \text{with probability } (1-\epsilon)*\alpha, \\ \text{rand } (a' \in m(s_t,\mathcal{A})), & \text{with probability } (1-\epsilon)*(1-\alpha). \end{cases} \tag{13}$$

With probability $(1-\epsilon)*\alpha$, the scheduler uses the "expert" policy to select the action and with the remaining probability to randomly select a valid action, where $\alpha \in (0,1)$ denotes the expert ratio. The environment transits to the next state and return the new state information s_{t+1}, reward r_{t+1} based on the transition function Γ. Moreover, it also sends a terminal signal to the scheduler(i.e., $d_{t+1} = True$ if the $s_t + 1$ is a termination state else $d_{t+1} = False$). The scheduler stores the transition $(s_t, a_t, r_t, s_{t+1}, d_{t+1})$ into the experience replay memory and keeps interacting with the environment until the terminated state.

However, there is no expert policy for multi-NUMA VM scheduling. SchedRL devises the episodic guided policy to approximate the expert policy. Inspired from episodic memory control [45], we can maintain the "best" state-action-reward buffer $\mathcal{B}$. $\mathcal{B}$ stores the highest reward-to-go for every state ($\mathcal{B}(s_t)[0]$) and the corresponding action ($\mathcal{B}(s_t)[1]$). Suppose the agent interacts with the environment, collects a trajectory $\tau = (s_0, a_0, r_0, \ldots, s_k, a_k, r_k)$ and the s_k is the termination state. Then the buffer $\mathcal{B}$ updates as follows:

$$\mathcal{B}(s_t) = \begin{cases} (\sum_{t'=t}^{k} r_{t'}, a_t), & \text{if } \sum_{t'=t}^{k} r_{t'} \geq \mathcal{B}(s_t)[0], \\ \mathcal{B}(s_t), & \text{else.} \end{cases} \tag{14}$$

The **episodic guided sampling** then can be defined as $a_t = \mathcal{B}(s_t)[1]$. We utilize it as the expert policy to gain better sample efficiency.

In the training procedure, the scheduler sample a batch of transitions from the experience replay memory and the loss function is defined as

$$\mathcal{L}(\theta) := \mathbb{E}_{(s,a,r,s',d)} \times \left[\left(r + \gamma(1-d)Q(s, \arg\max_{a' \in m(s,\mathcal{A})} Q(s,a';\theta);\theta^-) - Q(s,a;\theta)\right)^2\right], \tag{15}$$

with θ^- be the parameters of the target Q-network. The parameters θ of the Q-network are updated through gradient descent scheme, while the soft update technique is further employed as follows.

$$\theta^- = (1-\beta)\theta^- + \beta\theta, \tag{16}$$

where β denotes the soft update ratio. Finally the full procedure can be summarized in the following flow Fig. 4 and Algorithm 1.

5. Experiments

This section provides the empirical study of SchedRL. It first provides the simulator details and describes two practical scenarios which are adopted to evaluate scheduling algorithms. Then the

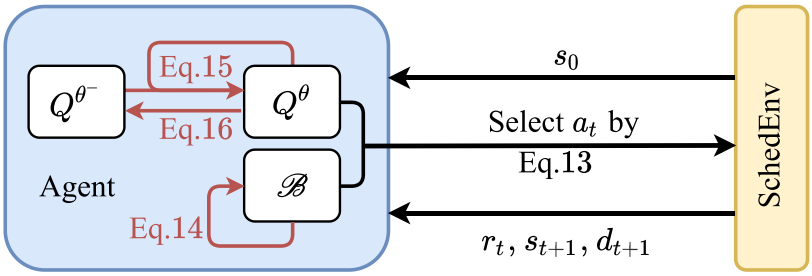


Fig. 4. The algorithm flow of our SchedRL. The black lines denote the interaction during the execution stage. The red lines denote the update process during training stage. The algorithmic description is shown at Algorithm 1. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Algorithm 1 SchedRL: the RL-based multi-NUMA VM scheduling algorithm.

Initialization: Q-network parameter: θ, target Q-network parameter: θ^-, best state-action-r buffer $\mathcal{B}$, discount factor γ, maximum epochs K;
for Episode $= 1, \cdots, K$ **do**
  Sample a request Seqs from dataset and send s_0 to the scheduler;
  for $t = 0, \cdots, T$ and $d_t \neq False$ **do**
    Use episodic guided sampling based on (13) to obtain an action sampling a_t;
    The environment transfers to next state and send r_t, s_{t+1}, d_{t+1} to the scheduler;
    Push the transition into replay buffer $s_t, a_t, r_t, s_{t+1}, d_{t+1}$ and update best state-action-r buffer $\mathcal{B}$ follows (14);
    Sampling a batch of transitions and update Q-network by optimize (15);
    Soft update the target Q-network following (16).
  end for
end for

baselines, measurements, and hyperparameters are stated in the following subsection. Finally, we take the empirical experiments and provide the comparisons and analysis of our SchedRL.

5.1. Simulator details

The simulator is mainly built based on Azure's scheduling dataset [13]. In a practical scheduling environment, two scenarios are often considered: creation-only and general (contains both creation and deletion). The creation-only scenario is common in a dedicated cloud (the resources are provided for enterprises and governments for a long time and barely happen a delete request). And the general scenario happens for a cloud that provides resources to individual users (the delete request can happen frequently).

For both scenarios, the simulator takes the 5 servers as a cluster, each server has 2 NUMAs, and each NUMA has 50 CPU and 160G memory. The VMs that may happen in the cluster have 5 different sizes, and the threshold for spanning is $c = 32$. For the creation-only scenario, the request sequences are sampled from the VMs distribution. For the general scenario, we analyze the lifetime distribution of different VMs in the original dataset. Then we generate a lifetime for each creation request based on the distribution. Note that it is only used to generate datasets, while the scheduler cannot access this information.

Table 2
Hyperparamaters for SchedRL in both scenarios.

learning rate	$5e^{-3}$
discount factor	0.85
batch size	2048
Activation function	Relu
ϵ_{start}	1.0
ϵ_{end}	0.01
expert ratio α	0.6
memory capacity	1e6
optimizer	Adam
Hidden Layers	MLP(128, 256, 256)

5.2. Baselines, measurements and hyperparameters

For the baselines choosing, we first select two traditional and popular adopted algorithms in the realistic environment: FirstFit and BestFit as baselines. Some RL methods are also included in baselines to show the influence of our proposed reward setting and expert sampling strategy. The baselines are summarized as follows:

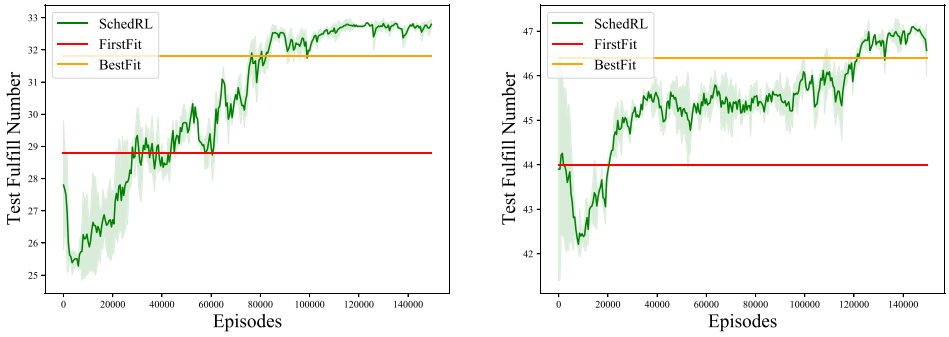
- FirstFit [46] (FF): Select the server with the smallest index number available.
- BestFit [46] (BF): Select the server with the highest allocation rate. The BestFit is modified as the largest one of multiple NUMAs when the request needs to be compatible with the multi-NUMA architecture.
- SchedRL-alpha: SchedRL with different expert ratio;
- SchedRL w/o Delta: SchedRL with identical reward settings.

This paper takes the following factor as the measurements:

- Fulfill Number: The number of creation requests that the cluster handled.
- CPU/Memory Allocation Rate: The rate that one server's allocated CPU/memory divides its total CPU/memory resource after termination.

The fulfill number is a direct measurement that indicates how many creation requests a scheduler can handle. The allocation rate describes the termination status and can be taken to measure how better allocation status a scheduler can lead the cluster.

To enable reproducibility, we provide the hyperparameters of SchedRL in Table 2. The ϵ_{start} and ϵ_{end} are the exploration ratio at start and end, respectively. The discount factor, learning rate, and expert rate are tuned for every algorithm by grid search, and others are kept the same.



(a) Creation-only scenario (b) General scenario

Fig. 5. The test performance(fulfill number) comparison with Anyfit algorithms in different scenarios.

Table 3
Performance comparisons with different measurements in the creation-only scenario, The bold denotes the best result in each column, and the ± shows the variance.

Method	Fulfill Number	Cluster CPU Allocation Rate	Cluster Memory Allocation Rate
SchedRL	**33.30 ± 1.79**	**51.40% ± 10.35%**	**82.94% ± 15.74%**
FirstFit	28.80 ± 7.29	46.36% ± 14.52%	74.88% ± 21.70%
BestFit	31.80 ± 4.62	49.72% ± 13.02%	80.26% ± 19.20%
SchedRL w/o Delta	33.10 ± 1.58	51.04% ± 9.70%	82.39% ± 14.63%

Table 4
Performance comparisons with different measurements in the general scenario, The bold denotes the best result in each column, and the ± shows the variance.

Method	Fulfill Number	Cluster CPU Allocation Rate	Cluster Memory Allocation Rate
SchedRL	**47.80 ± 3.71**	**50.52% ± 5.65%**	**81.46% ± 8.43%**
FirstFit	44.00 ± 2.49	44.16% ± 12.30%	70.65% ± 16.33%
BestFit	46.40 ± 3.17	48.00% ± 11.16%	77.00% ± 15.98%
SchedRL w/o Delta	45.40 ± 4.78	44.96% ± 12.56%	71.95% ± 17.56%
SchedRL-0.0alpha	45.20 ± 2.04	46.08% ± 10.70%	73.83% ± 14.23%
SchedRL-0.3alpha	45.80 ± 2.44	47.04% ± 6.35%	75.41% ± 10.17%
SchedRL-0.9alpha	46.40 ± 2.33	48.12% ± 5.59%	77.33% ± 9.89%

(a) FirstFit(step = 116) (b) BestFit(step = 116) (c) SchedRL(step = 116)

(d) BestFit(step = 122) (e) SchedRL(step = 122) (f) SchedRL(step = 127)

Fig. 6. The cluster status at different steps in the general scenario. The red pillar and yellow pillar are allocated resources on NUMA 1 and 2. The grey pillar denotes the next placed request. Figure (a), (b), and (c) show the status of FirstFit, BestFit, and SchedRL at step 116 at which the FirstFit terminated. Figure (d), (e) show the status of BestFit and SchedRL at step 122 at which the Best terminated. Figure (f) shows the terminated status of SchedRL, which is at step 127. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.3. Experiments results and analysis

5.3.1. Performance comparison

We first train SchedRL for 150000 epochs in the creation-only scenario with 3 different seeds. Fig. 5(a) shows the scheduler performance regarding the fulfill number for every 500 episodes. If not explicitly mentioned, the solid line and shadow area in all the learning curves denote the mean and variance, respectively. It can be seen that SchedRL outperforms both FirstFit and BestFit regarding the fulfill number. To better evaluate SchedRL and traditional baselines' performance, we statistics the converged performance regarding different measurements at Table 3. SchedRL achieves a 33.3 fulfill number while FirstFit and Bestfit achieve 28.8 and 31.8 fulfill number. This indicates that SchedRL can handle more VM creation requests than the traditional baselines. Moreover, the lowest variance on the fulfill number states that the SchedRL has stabler performance on different sequences than the FirstFit and BestFit. With the ability to handle more creation requests, the SchedRL achieves higher CPU and memory allocation rate than the others.

For the general scenario, we take the same training procedure as in the creation-only scenario. As shown in Fig. 5(b), the SchedRL achieves a 47 fulfill number and outperforms the FirstFit and BestFit. The Table 4 compares the SchedRL with FisrtFit and BestFit with different measurements. The SchedRL can lead the server to a higher allocation rate than the baselines. To understand the reason that SchedRL outperforms others, we sample a sequence in the environment and visualize the cluster status of different methods in Fig. 6. The FirstFit terminated at step 116 due to resources are placed over all the servers, and the left space in none of the servers can handle the next request. BestFit and SchedRL hold resource spaces on the server 4 and 3 respectively and can still handle the following requests. At step 122, the BestFit can not handle the following requests while SchedRL holds some resource space at server 3. This leads SchedRL to handle more requests and achieve higher allocation rates, as shown in Fig. 6(f).

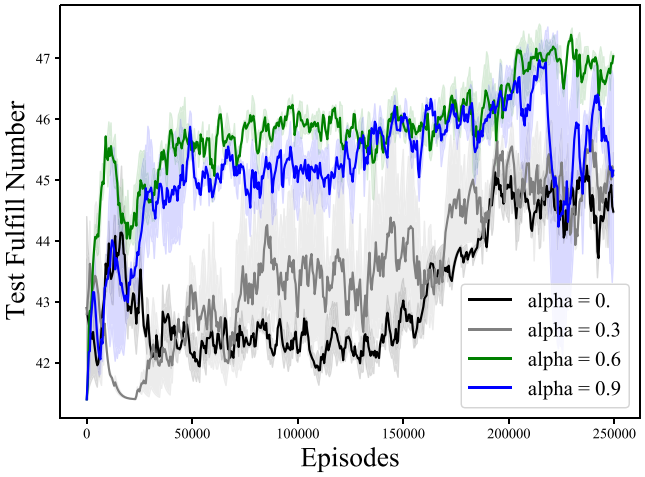


Fig. 7. The test performance(fulfill number) comparison with different expert ratios.

different expert ratio($\alpha = 0., 0.3, 0.6, 0.9$) for 250000 episodes[5] in general scenario. As shown in the Fig. 7, SchedRL achieves the best converged result when $\alpha = 0.6$. When the $\alpha = 0$, the SchedRL becomes double DQN. As the α increases to 0.6, the SchedRL achieves higher converge result. However, if the expert ratio is too large($\alpha = 0.9$), the scheduler achieves worse performance than the $\alpha = 0.6$ after convergence. This is because the large ratio can impede exploration which may let agent stuck on the sub-optimal solution (i.e. in Fig. 7 after 210000 episodes). The performance of different expert ratios regarding different measurements is summarized in Table 4. The SchedRL-0.3alpha, SchedRL-0.9alpha, and SchedRL($\alpha = 0.6$) achieve better performance regarding fulfill number, cluster CPU, and memory allocation rate than the Sched-0.0alpha. This in-

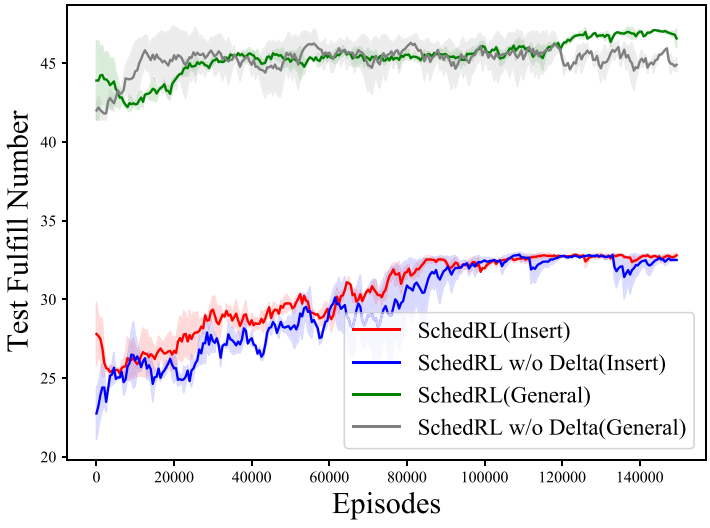


Fig. 8. The test performance(fulfill number) comparison with different reward settings.

5.3.2. Ablation study for sampling strategy

The expert ratio provides a trade-off between exploration and exploitation off. With a large expert ratio, the scheduler tends to exploit more, while the small ratio tends to explore more. To investigate the influence of the expert ratio, we train SchedRL with

[5] In some expert ratios, the 150000 episodes are not enough to converge, thus we extend it 250000 episodes.

dicates the episodic guided sampling does benefit the learning process. Moreover, the different converged results state that choosing an appropriate expert ratio is essential for training also.

5.3.3. Reward studies

To investigate the influence of different reward settings, we compare SchedRL with SchedRL w/o Delta. The results in both scenarios are presented in Fig. 8. In the creation-only scenario, the SchedRL achieves similar converged results with identical rewards while has a minor variance. The SchedRL achieves both higher reward and low variance in the general scenario compared with SchedRL w/o Delta. The measurements of SchedRL w/o Delta are provided at Tables 3 and 4. It can be concluded that the delta reward helps significantly in the general scenario, while the identical reward can achieve similar performance in the creation-only scenario.

6. Conclusion

In this paper, we give the formal constrained optimization formulation for the multi-NUMA VMs scheduling problem. To solve the problem efficiently, we further formulate it into a reinforcement learning framework and propose the SchedRL algorithm to solve it approximately. The SchedRL introduces a delta reward function and an episodic guided sampling strategy into deep Q-learning. Conducting experiments on a public dataset, our SchedRL outperforms the FirstFit and BestFit on both fulfill number and allocation rate.

Our work still has some limitations, and we would like to point them out and discuss some future directions. 1) scalability: When the number of servers increases, both the state space and action space increase exponentially, bringing difficulty to learning. A cloud service provider often needs to schedule requests among many servers in the practical scenario, thus making the learning process scalable is an important future direction. 2) generalization: The requests are generated by several users, and the request distribution is changing over time, and our method does not take that into consideration. How to generalize the scheduling policy to changing distribution is closely related to meta-learning and robust learning. We leave it to the future work.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by National Key Research and Development Program of China (No. 2020AAA0107400), NSFC (No. 12071145), a grant form Huawei Cloud Huawei Technologies Co., Ltd., Shanghai Municipal Science and Technology Major Project (No. 2021SHZDZX0102), STCSM (No. 18DZ2270700 and 20511101100), the Open Research Projects of Zhejiang Lab (NO.2021KE0AB03) and a grant from Shenzhen Institute of Artificial Intelligence and Robotics for Society.

References

[1] D. Jiankang, W. Hongbo, L. Yangyang, C. Shiduan, Virtual machine scheduling for improving energy efficiency in IaaS cloud, China Commun. 11 (3) (2014) 1–12, doi:10.1109/cc.2014.6825253.
[2] A. Wolke, B. Tsend-Ayush, C. Pfeiffer, M. Bichler, More than bin packing: dynamic resource allocation strategies in cloud data centers, Inf. Syst. 52 (2015) 83–95, doi:10.1016/j.is.2015.03.003.
[3] J. Shi, J. Luo, F. Dong, J. Jin, J. Shen, Fast multi-resource allocation with patterns in large scale cloud data center, J. Comput. Sci. 26 (2018) 389–401, doi:10.1016/j.jocs.2017.05.005.
[4] J. E. G. Coffman, M.R. Garey, D.S. Johnson, Dynamic bin packing, SIAM J. Comput. 12 (2) (1983) 227–258, doi:10.1137/0212014.
[5] A.L. Stolyar, An infinite server system with general packing constraints, Oper. Res. 61 (5) (2013) 1200–1217, doi:10.1287/opre.2013.1184.
[6] Y. Li, X. Tang, W. Cai, On dynamic bin packing for resource allocation in the cloud, in: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2014, doi:10.1145/2612669.2612675.
[7] Y. Li, X. Tang, W. Cai, Dynamic bin packing for on-demand cloud resource allocation, IEEE Trans. Parallel Distrib. Syst. 27 (1) (2016) 157–170, doi:10.1109/tpds.2015.2393868.
[8] X. Tang, Y. Li, R. Ren, W. Cai, On first fit bin packing for online cloud server allocation, 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016, doi:10.1109/ipdps.2016.42.
[9] C. Lameter, NUMA (Non-uniform memory access): an overview, Queue 11 (7) (2013) 40–51, doi:10.1145/2508834.2513149.
[10] J. Rao, K. Wang, X. Zhou, C. Xu, Optimizing virtual machine scheduling in NUMA multicore systems, in: 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), 2013, pp. 306–317.
[11] W. Zhou, Huawei cloud algorithm challenge, in: Mathematics Promotes Enterprise Innovation and Development Forum, Chinese Society of Industrial and Applied Mathematics, 2020, p. 1.
[12] A. Khadke, A. Agarwal, A.M. Kabir, D. Schwab, Exploration with expert policy advice, 2018.
[13] O. Hadary, L. Marshall, I. Menache, A. Pan, E.E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, et al., Protean:{VM} allocation service at scale, in: Symposium on Operating Systems Design and Implementation, 2020, pp. 845–861.
[14] Y. Azar, D. Vainstein, Tight bounds for clairvoyant dynamic bin packing, ACM Trans. Parallel Comput. 6 (3) (2019) 1–21, doi:10.1145/3364214.
[15] W. Li, J. Tordsson, E. Elmroth, Modeling for dynamic cloud scheduling via migration of virtual machines, in: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, IEEE, 2011, doi:10.1109/cloudcom.2011.31.
[16] J. Ahn, C. Kim, J. Han, Y.-R. Choi, J. Huh, Dynamic virtual machine scheduling in clouds for architectural shared resources, in: Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing, in: HotCloud'12, USENIX Association, USA, 2012, p. 19.
[17] J. Rao, K. Wang, X. Zhou, C.-Z. Xu, Optimizing virtual machine scheduling in NUMA multicore systems, in: International Symposium on High Performance Computer Architecture, 2013, pp. 306–317.
[18] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, Eur. J. Oper. Res. 290 (2) (2021) 405–421, doi:10.1016/j.ejor.2020.07.063.
[19] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, in: Advances in Neural Information Processing Systems, vol. 28, 2015, pp. 2692–2700.
[20] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, Arxiv 1 (2016) 1–15.
[21] M. Nazari, A. Oroojlooy, L. Snyder, M. Takac, Reinforcement learning for solving the vehicle routing problem, in: Advances in Neural Information Processing Systems, vol. 31, 2018, pp. 9861–9871.
[22] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937.
[23] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, L.-M. Rousseau, Learning heuristics for the TSP by policy gradient, in: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, 2018, pp. 170–181, doi:10.1007/978-3-319-93031-2_12.
[24] W. Kool, H. van Hoof, M. Welling, Attention, learn to solve routing problems!, in: 7th International Conference on Learning Representations, OpenReview.net, 2019, pp. 1–25.
[25] H. Hu, X. Zhang, X. Yan, L. Wang, Y. Xu, Solving a new 3D bin packing problem with deep reinforcement learning method, Arxiv 1 (2017) 1–7.
[26] L. Duan, H. Hu, Y. Qian, Y. Gong, X. Zhang, J. Wei, Y. Xu, A multi-task selected learning approach for solving 3D flexible bin packing problem, in: International Conference on Autonomous Agents and Multi-Agent Systems, 2019, pp. 1386–1394.
[27] A. Laterre, Y. Fu, M.K. Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T.S. Dahl, A. Kerkeni, K. Beguir, Ranked reward: enabling self-play reinforcement learning for combinatorial optimization, Arxiv 1 (2018) 1–11.
[28] H. Zhao, Q. She, C. Zhu, Y. Yang, K. Xu, Online 3D bin packing with constrained deep reinforcement learning, Arxiv 1 (2020) 1–16.
[29] M. Kruber, M.E. Lbbecke, A. Parmentier, Learning when to use a decomposition, in: Integration of AI and OR Techniques in Constraint Programming, Springer International Publishing, 2017, pp. 202–210, doi:10.1007/978-3-319-59776-8_16.
[30] P. Bonami, A. Lodi, G. Zarpellon, Learning a classification of mixed-integer quadratic programming problems, in: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer International Publishing, 2018, pp. 595–604, doi:10.1007/978-3-319-93031-2_43.
[31] R. Mahmood, A. Babier, A. McNiven, A. Diamant, T.C. Chan, Automated treatment planning in radiation therapy using generative adversarial networks, in: Machine Learning for Healthcare Conference, 2018, pp. 484–499.
[32] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, A.A. Bharath, Generative adversarial networks: an overview, IEEE Signal Process. Mag. 35 (1) (2018) 53–65.

[33] W. Sihang, W. Jiapeng, M. Weihong, J. Lianwen, Precise detection of Chinese characters in historical documents with deep reinforcement learning, Pattern Recognit. 107 (2020) 107503, doi:10.1016/j.patcog.2020.107503.
[34] M. Sun, J. Xiao, E.G. Lim, Y. Xie, J. Feng, Adaptive ROI generation for video object segmentation using reinforcement learning, Pattern Recognit. 106 (2020) 107465, doi:10.1016/j.patcog.2020.107465.
[35] Y. Hua, X. Wang, B. Jin, W. Li, J. Yan, X. He, H. Zha, Hyper-meta reinforcement learning with sparse reward, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2021.
[36] M. Xu, Y. Song, J. Wang, M. Qiao, L. Huo, Z. Wang, Predicting head movement in panoramic video: a deep reinforcement learning approach, IEEE Trans. Pattern Anal. Mach. Intell. 41 (11) (2019) 2693–2708, doi:10.1109/TPAMI.2018.2858783.
[37] X. Dong, J. Shen, W. Wang, L. Shao, H. Ling, F. Porikli, Dynamical hyperparameter optimization via deep reinforcement learning in tracking, IEEE Trans. Pattern Anal. Mach. Intell. 43 (5) (2021) 1515–1529, doi:10.1109/TPAMI.2019.2956703.
[38] 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, ACM, 2016, doi:10.1145/3005745.3005750.
[39] F. Li, B. Hu, DeepJS: job scheduling based on deep reinforcement learning in cloud data center, in: Proceedings of the 2019 4th International Conference on Big Data and Computing - ICBDC 2019, ACM Press, 2019, doi:10.1145/3335484.3335513.
[40] H. Mao, M. Schwarzkopf, S.B. Venkatakrishnan, Z. Meng, M. Alizadeh, Learning scheduling algorithms for data processing clusters, in: Proceedings of the ACM Special Interest Group on Data Communication, ACM, 2019, doi:10.1145/3341302.3342080.
[41] Y. Bao, Y. Peng, C. Wu, Deep learning-based job placement in distributed machine learning clusters, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, IEEE, 2019, doi:10.1109/infocom.2019.8737460.
[42] H. Zhang, X. Geng, H. Ma, Learning-driven interference-aware workload parallelization for streaming applications in heterogeneous cluster, IEEE Trans. Parallel Distrib. Syst. 32 (1) (2021) 1–15, doi:10.1109/tpds.2020.3008725.
[43] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: D. Schuurmans, M.P. Wellman (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12–17, 2016, Phoenix, Arizona, USA, AAAI Press, 2016, pp. 2094–2100.
[44] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, N. de Freitas, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning, vol. 48, 2016, pp. 1995–2003.
[45] Z. Lin, T. Zhao, G. Yang, L. Zhang, Episodic memory deep q-networks, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, 2018, doi:10.24963/ijcai.2018/337.
[46] C. Bays, A comparison of next-fit, first-fit, and best-fit, Commun. ACM 20 (3) (1977) 191–192, doi:10.1145/359436.359453.

Junjie Sheng received the B.E. degree in Software Engineering from East China Normal University, Shanghai, in 2019. He is currently pursuing the PhD degree in East China Normal University. His research interests include multiagent reinforcement Learning and graph neural network.

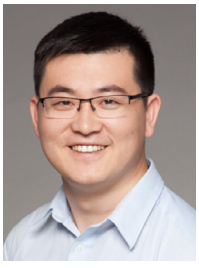
Yiqiu Hu received the B.S. degree in Mathematics from Southwest Jiaotong University School, Chengdu, Sichuam, in 2018. She is currently pursuing the M.S. degree in East China Normal University. Her research interests include federated learning and reinforcement learning.

Wenli Zhou received his Ph.D. degree in Applied Mathematics from Nankai University in 2010. He once worked in IBM Research and Microsoft Bing Search. He is currently a principal engineer in Algorithm Innovation Lab, Huawei Cloud, Huawei Technologies Co., Ltd. His expertise includes combinatorial optimization and information retrieval. He is working on AI for System related topics like virtual machine placement and cloud resource management.

Lei Zhu received his Ph.D. degree in School of Computer Science and Technology from Northwestern Polytechnical University, China, in 2017. His research interests include scheduling and fault tolerance for high-performance computing. Currently, he is a chief engineer of Alkaid Lab-Huawei Cloud, Huawei Technologies Co., Ltd. He is working on virtual machine placementcapacity planning and cloud resource management.

Bo Jin received the B.S. degree in electronic engineering and the Ph.D. degree in control theory and control engineering from Shanghai Jiao Tong University, China, in 2004 and 2014, respectively. He is currently an Associate Professor with the School of Computer Science and Technology and Technology and the MOE Key Laboratory for Advanced Theory and Application in Statistics and Data Science, East China Normal University. His major research interests include machine learning, reinforcement learning and medical diagnosis modeling.

Jun Wang received the PhD degree from Columbia University, New York, in 2011. Currently, he is a professor of computer science with East China Normal University, Shanghai, China. His research interests include machine learning, information retrieval, and data mining.



Xiangfeng Wang received the B.S. degree in mathematics and applied mathematics and the Ph.D.degree in computational mathematics from Nanjing University, Nanjing, China, in 2009 and 2014, respectively. He is currently an Associate Professor with the School of Computer Science and Technology and the MOE Key Laboratory for Advanced Theory and Application in Statistics and Data Science, East China Normal University, Shanghai, China, and the Shanghai Research Institute for Intelligent Autonomous Systems, Shanghai. His research interests lie in the areas of distributed optimization, multi-agent reinforcement learning and trustworty machine learning.