



# A DRL-based online VM scheduler for cost optimization in cloud brokers

Xingjia Li<sup>1</sup> · Li Pan<sup>1</sup> · Shijun Liu<sup>1</sup>

Received: 24 October 2022 / Revised: 1 December 2022 / Accepted: 29 January 2023 /  
Published online: 14 March 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

## Abstract

The virtual machine (VM) scheduling problem in cloud brokers that support cloud bursting is fraught with uncertainty due to the on-demand nature of Infrastructure as a Service (IaaS) VMs. Until a VM request is received, the scheduler does not know in advance when it will arrive or what configurations it demands. Even when a VM request is received, the scheduler does not know when the VM's lifecycle expires. Existing studies begin to use deep reinforcement learning (DRL) to solve such scheduling problems. However, they do not address how to guarantee the QoS of user requests. In this paper, we investigate a cost optimization problem for online VM scheduling in cloud brokers for cloud bursting to minimize the cost spent on public clouds while satisfying specified QoS restrictions. We propose DeepBS, a DRL-based online VM scheduler in a cloud broker which learns from experience to adaptively improve scheduling strategies in environments with non-smooth and uncertain user requests. We evaluate the performance of DeepBS under two request arrival patterns which are respectively based on Google and Alibaba cluster traces, and the experiments show that DeepBS has a significant advantage over other benchmark algorithms in terms of cost optimization.

**Keywords** Cloud computing · Deep reinforcement learning · Online scheduling · Cloud brokering · Hybrid cloud · Cloud bursting

---

✉ Li Pan  
panli@sdu.edu.cn

✉ Shijun Liu  
lsj@sdu.edu.cn

Xingjia Li  
xingjia.li@mail.sdu.edu.cn

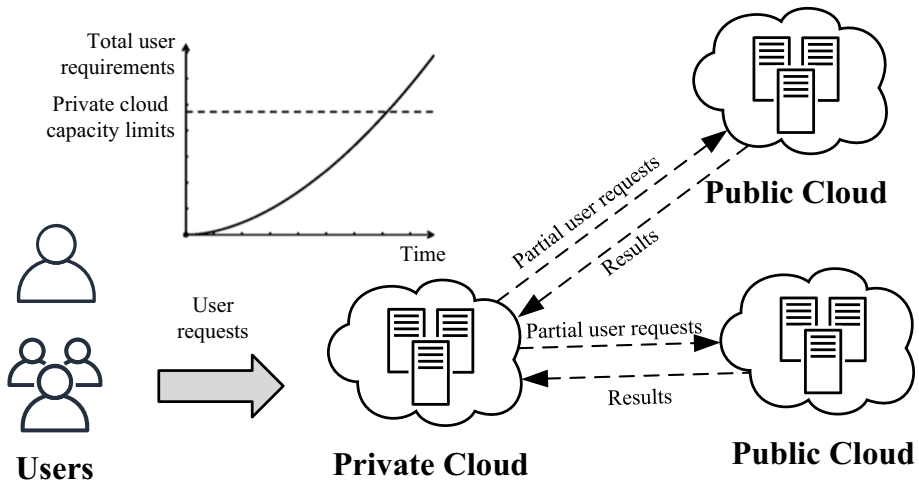
<sup>1</sup> School of Software, Shandong University, Jinan, China

## 1 Introduction

As virtualization and networking technologies advance, cloud computing has established an irreplaceable paradigm that has altered the way of designing and purchasing computing resources and IT services. Cloud computing possesses on-demand and elastic features, enabling the information business to access low-cost computing resources, therefore transforming it into the century's most inventive area, and enabling industrial and scientific computing to access inexpensive on-demand services. Cloud computing's quick growth has resulted in a more established and mature market and technology, leading to improved services for consumers. However, the proliferation of cloud providers has introduced new hazards for customers. While the majority of cloud providers currently offer similar services, they do not all guarantee cloud interoperability and portability. As consumers become more reliant on a single cloud provider, it becomes more difficult for them to switch cloud vendors, a phenomenon known as vendor lock-in [1–3]. As a result, multi-cloud arises as a model for concurrently utilizing services supplied by different cloud providers. Utilizing services based on multiple clouds, consumers receive a more efficient and robust service, while cloud providers benefit from increased user adoption of cloud services [4]. Cloud brokers [5] play a significant role in the cloud ecosystem by providing consumers more appealing cloud services from multiple cloud providers. By utilizing a cloud broker, consumers can avoid being overly reliant on a single cloud provider and instead obtain a cloud service that is more favorable in terms of cost and service quality. According to available data, the worldwide cloud service brokerage market is worth USD 6993.19 million in 2020 and is predicted to reach USD 7679.41 million in 2021, before increasing at a 10.15 percent compound annual growth rate to reach USD 12.49135 billion by 2026 [6].

Global pandemic of COVID-19 has driven a surge in demand for cloud computing. Because of the proliferation of home offices, internet businesses, such as online conferencing, have seen increased adoption. This rise in demand has left some firms unable to meet all of their computing needs entirely through their own private cloud, prompting several to shift some of their computing needs to the public cloud. According to the State of the Cloud Report Survey provided by Flexera [7], 92% of the 750 companies participating in the study are using multiple or hybrid clouds. *Cloud bursting* is a typical hybrid cloud paradigm that mixes on-premises and cloud computing resources to provide services to users. As shown in Figure 1, when the total user requirements surge, it is difficult for the private cloud to handle all the user requests. Faced with these challenge, the cloud bursting architecture accommodates all user requests by transferring some of them from the private cloud to the public cloud. Typical cloud bursting application scenarios include 3D rendering, data analysis, and seasonal online marketing campaigns. These scenarios are characterized by brief overflows of traffic or sudden demands on computing resources that are difficult for local computing resources to cope with. Cloud brokers can help users manage resources, both local computing resources as well as multi-cloud resources. Therefore, cloud brokers are well-suited to assisting customers who already have local data centers in achieving a hybrid cloud environment, and cloud bursting enabled by cloud brokers has been studied [8].

Infrastructure as a Service (IaaS) is the most popular cloud computing service model, providing users with virtual machines called instances as services under some billing models. On-demand instances with a pay-per-use billing model are one of the important categories. It is vital for cloud brokers that provide IaaS to fulfill the on-demand nature of services. This means that, until a request arrives, cloud brokers are unaware of its arrival

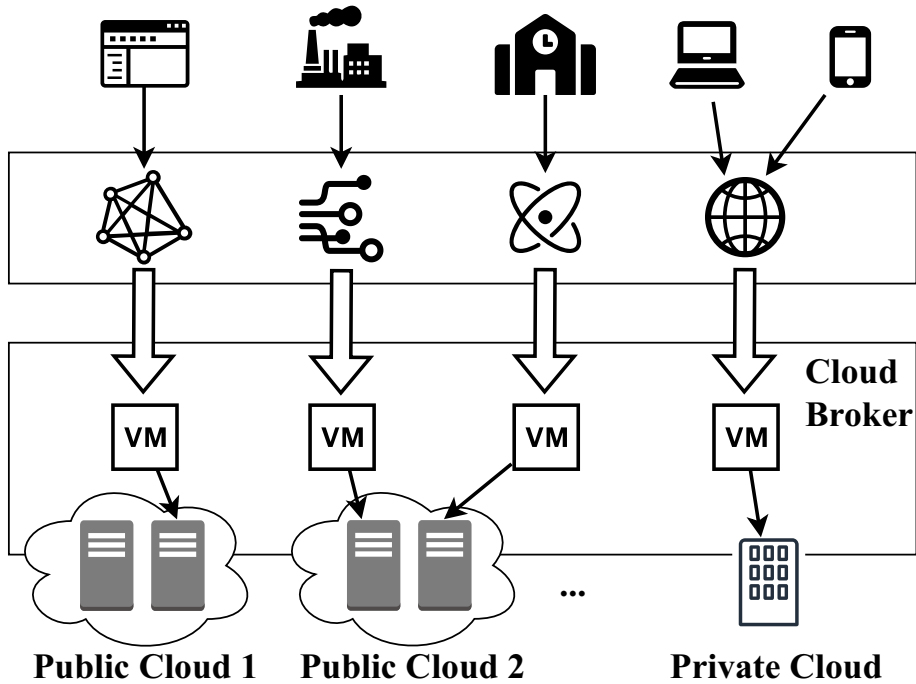


**Figure 1** An overview of cloud bursting for handling surges in user requests

time or its exact configuration requirements. Moreover, there is no way to know or accurately predict the exact moment of departure before this request leaves. On the other hand, cloud brokers considering cloud bursting need to select among numerous servers in private and public clouds, and should minimize the public cloud cost while dealing with request uncertainty. Therefore, cloud brokers need an appropriate online scheduling strategy to accommodate current cloud bursts. Figure 2 gives a scenario where cloud bursting is implemented using a cloud broker, where requests from different organizations such as data analysis service providers, factories, and research institutions are transferred to the public cloud via cloud bursting. Different types of jobs are scheduled as virtual machines in private clouds and multiple public clouds.

Existing scheduling problems in cloud computing focus on the use of heuristics and meta-heuristics to obtain near-optimal scheduling solutions in an acceptable running time for large-scale jobs and clusters. For dynamic environments, such as cloud brokers with uncertain user requests, meta-heuristics suffer from rapid diversity loss and premature convergence [9], and thus there is no guarantee that meta-heuristics will always achieve superior performance. In contrast to meta-heuristics, machine learning methods can uncover hidden relationships between data in order to generate scheduling decisions that are difficult to detect using conventional optimization techniques or heuristics based on common sense [10]. Reinforcement learning (RL), a machine learning paradigm characterized by trial and error and delayed gains, has achieved abundant and effective results in recent years on sequential decision-making problems in various fields of engineering and academia. Deep reinforcement learning (DRL) is a combination of reinforcement learning and deep learning, which overcomes the shortcomings of table-based reinforcement learning and empowers reinforcement learning with the ability to model complex systems and adapt to complex environments.

Due to the enormous potential demonstrated by DRL, approaches based on DRL have been applied to cloud resource management and scheduling issues [11, 12]. Several works [10, 13, 14] have proposed DRL-based approaches for offline VM scheduling and placement policies. Sheng et al. [15] investigate online VM scheduling in NUMA architectures



**Figure 2** A scene where cloud bursting is implemented using a cloud broker

to optimize the amount of requests that can be accommodated from the cloud provider's perspective. However, existing research on using reinforcement learning for cloud resource scheduling does not directly address the scheduling of requests with explicit QoS restrictions. Due to the nature of reinforcement learning in trial and error, agents are prone to take actions that violate the requested QoS requirements.

Considering the challenges of online decision making and guaranteeing QoS of user requests faced by the VM scheduling problem in cloud broker architectures for cloud bursting, in this paper we propose a DRL-based online VM scheduler called DeepBS that takes cloud bursting into account from the perspective of cloud brokers. When confronted with dynamic user requests that have unknown arrival time, configuration requirements, and departure time, DeepBS can make immediate decisions and adapt to the changing environment to minimize the cost of computing resources while ensuring that the QoS requirements of user requests are satisfied. We evaluate the performance of our proposed algorithm under two request arrival patterns which are respectively based on Google cluster trace and Alibaba cluster trace, and the experiments show that DeepBS has a significant advantage over other benchmark algorithms in terms of cost optimization. Our main contributions are as follows:

- Facing the increasing demand for computing resources, a cloud broker architecture for cloud bursting is proposed for unified management of cloud resources in private clouds and multiple public clouds. As a result, computing resources in the form of hybrid clouds are viewed as a unified whole.

- The online VM scheduling problem is modeled in the cloud broker architecture for cloud bursting with the goal of minimizing computing resource cost while guaranteeing QoS. This problem is quite challenging because user requests have uncertainty in arrival time, configuration requirements, and departure time.
- To address the problem of online VM scheduling in the cloud broker architecture used for cloud bursting, we propose a DRL based scheduler called DeepBS for online scheduling of VM requests with unknown arrival and departure time as well as heterogeneous configuration requirements in a multi-cloud environment, with the goal of minimizing computing resource cost and guaranteeing QoS of user requests.
- We evaluate the performance of our proposed algorithm under two request arrival patterns based on Google cluster trace and Alibaba cluster trace and the latest pricing of Amazon EC2. The experimental analysis shows that DeepBS has a significant advantage in terms of cost optimization. Specifically, DeepBS has a 27.84% performance advantage in cost optimization over the best performing benchmark algorithm in both real cluster-based tracking request arrival modes, and still maintains optimal performance in simulation scenarios with more extreme request arrival patterns.

The rest of the paper is structured as follows. Section 2 takes a look at related work. Section 3 describes the system model and problem formulation. Section 4 presents a method for modelling the cloud broker VM scheduling problem with the objective of minimizing cost as an RL problem. Section 5 details the DRL-based DeepBS. Experiments and performance analysis are performed in Section 6. Finally, Section 7 concludes this paper. A side-by-side comparison of related works is shown in Table 1.

## 2 Related work

Through virtualization technology, cloud providers make physical machines (also known as host, server, or PM) available to consumers in a multi-tenant basis. Multiple virtual machines can be deployed concurrently on the same physical hardware [22]. From the user's perspective, virtual machines delivered in the form of instances appear to be limitless, elastic, and on-demand. Appropriate resource management is required for the provider to reduce resource waste and maximize cost savings [23, 24]. One of the most challenging resource allocation problems in cloud computing is the VM scheduling problem, which involves determining which physical machine will host the VM request based on current VM requirement and server resource consumption conditions [25].

In this section, the literature on VM scheduling is reviewed. Related research is divided into three categories: bin packing based methods, meta-heuristic based methods, and reinforcement learning based methods.

### 2.1 Bin packing based methods

The resource allocation problem in cloud environments is often modelled as a variant of the classical bin packing problem. A number of greedy heuristic strategies are proposed based on competitive ratio analysis and worst-case performance guarantees. Online strategies are widely studied and applied in practice as they give intuition-based and performance-guaranteed results.

Li et al. [16, 17] analyze the competitive ratios of First Fit, Best Fit and Any Fit, which are commonly used in dynamic bin packing problem, and propose a hybrid First Fit

**Table 1** A side-by-side comparison of related works

Ref.	Year	Method	Performance metrics	Dataset	Online or Offline
[16, 17]	2016	Hybrid First Fit	Number of bins	-	Online
[18]	2019	Hybrid algorithm	Number of open bins	-	Online
[19]	2021	SCA, ALO	Resource utilization	PlanetLab	Offline
[20]	2021	MBO	Power consumption	Simulation	Offline
[11]	2016	REINFORCE	Resource utilization	PlanetLab	Offline
[13]	2018	Q-learning	Average job slowdown	Simulation	Offline
[14]	2020	Q-learning	Power consumption SLA	-	Offline
[12]	2020	DQN	Energy consumption	Simulation	Offline
[10]	2021	Rainbow DQN	Performance loss SLA	-	Offline
[21]	2021	PPO	Makespan Load balance	Simulation	Offline
[15]	2022	D2QN	QoE to power ratio	Simulation	Offline
DeepBS	2022	PPO with QoS Guarantor	Consumed power	Simulation	Online
			Cost Energy consumption	Azure	Online
			Test fulfill number	Google Alibaba Simulation	Online
			Cost Penalty for violating QoS		Online

algorithm with improved competitive ratio compared with First Fit to solve the MinTotal dynamic packing problem for on-demand cloud resource allocation. Azar and Vainstein et al. [18] investigates a variant of the online packing problem called Clairvoyant Dynamic Bin Packing, in which the time of departure is known when items arrive. An  $O(\sqrt{\log \mu})$  – *competitive* online algorithm is proposed.

## 2.2 Heuristic and meta-heuristic based methods

Due to the large solution space and high computational complexity of service deployment problems, it is impossible to achieve an exact solution under these conditions, even given detailed information about the request and specific information about the cloud resources [26]. Meta-heuristics are significantly effective in solving large computational and complex problems. This class of algorithms is not problem-specific but a general framework that gives near-optimal or suboptimal solutions to NP-complete problems in an acceptable running time by efficiently exploring the search space. As a result, meta-heuristic algorithms have been widely studied as a service deployment problem with a large solution space and difficulty in finding exact solutions [27].

Gharehpasha et al. [19] propose a method that combines the Sine Cosine Algorithm (SCA) and the Antlion optimizer (ALO) as discrete multi-objective chaotic functions for optimal allocation of VMs. The first objective of the algorithm is to minimize power consumption in the cloud data center by balancing the number of active PMs and the second objective is to reduce resource wastage by using an optimal allocation of VMs on the PMs in the cloud data center. Ghetas [20] proposes a new approach to virtual machine layout based on the monarch butterfly optimization (MBO) algorithm, which aims to maximize packing efficiency and reduce the number of active physical servers. The aim is to improve the energy efficiency and resource utilization of data centers and thus reduce overall cloud management costs.

Although meta-heuristics can search for near-optimal solutions in a large solution space, they are difficult to use for online problems due to the unknowable nature of online scheduling problems with respect to future information.

## 2.3 Reinforcement learning based methods

The resource management problem encounters the following difficulties in both genuine cloud providers and cloud brokers. To begin, the systems are massive and heterogeneous, as seen by the vast number and variety of servers or instances that cloud providers and cloud brokers must manage. Additionally, there is uncertainty regarding the arrival time of user requests, their configuration requirements, and their departure time. Finally, in the face of unpredictable user requirements, scheduling algorithms must make real-time decisions, and it is impossible to make the assumption that all requests are known before decisions are made [28]. DRL is a combination of RL and DL to provide a solution allowing for high-dimensional problems [29]. The goal of DRL is to learn optimal behaviour by interacting with the environment so that the system can actively learn optimal policies without a prior knowledge and can solve complex problems in environments with uncertainty, such as resource management problems in cloud environments.

The DeepRM proposed by Mao et al. [11] is the most representative work on DRL-based resource management solutions. Their proposed approach allows DeepRM-based systems to learn how to schedule job resources themselves instead of manual-based heuristics. Rolik et al. [13] propose a Q-learning based approach to cloud data center resource management

for dynamic virtual machine placement in an online manner to optimize data center power consumption, user service level agreement compliance and resource utilization. Long et al. [14] propose a reinforcement learning-based virtual machine placement strategy (RLVMP) for solving energy saving problems in cloud data centers. This strategy uses performance loss as part of the reward function in reinforcement learning to avoid performance degradation due to energy saving. The deep Q-learning task scheduling (DQTS) proposed by Tong et al. [12] is used to solve the directed acyclic graph (DAG) task processing problem in a cloud environment, with the optimization objectives of load balancing and makespan minimizing. Caviglione et al. [10] propose a multi-objective management strategy based on DRL with the objectives of (i) minimizing the effects of hardware/software outages, (ii) minimizing co-location interference, and (iii) minimizing power consumption. Zhao and Buyya et al. [21] propose a DRL-based task scheduling method in hybrid clouds considering the cyclical fluctuations of clean energy sources as exemplified by solar energy. The method aims at minimizing energy cost and public cloud cost, and achieves multi-objective DRL scheduling by constructing reward function based on a weighting method. Sheng et al. [15] propose a DRL-based approach called SchedRL to solve the VM scheduling problem in multi-NUMA architectures. They model the VM scheduling problem as a bin packing problem. An evaluation with Azure's public dataset shows that SchedRL can accommodate more requests than Best Fit and First Fit with the same amount of computational resources.

Most of the existing work, whether it is the VM placement problem, the container placement problem, or the task scheduling problem, has been done from the cloud provider's perspective, with energy as the optimization goal. This does not apply to resource optimization problems in cloud brokers, which provide IaaS services and higher level services to users based on the IaaS services provided by the cloud provider, where the cloud broker does not directly manage the physical machines and the cost is not the energy overhead of the physical machines, but the cost of the IaaS services. This paper presents a DRL-based approach to address the problem of online VM scheduling in the cloud broker architecture used for cloud bursting. To the best of our knowledge, this paper is the first to use a DRL-based approach to address the cost optimization problem of resource management from the perspective of a cloud broker. Since RL agents need to learn by trial and error, it is challenging to use RL to solve the scheduling problem for cloud services while ensuring QoS. In contrast to other work using RL to solve the scheduling problem in cloud environments while ignoring QoS, we propose a method to guide RL agents to learn policies that can avoid QoS violations.

### 3 System model and problem formulation

In this section, the cloud broker architecture for cloud bursting discussed in this paper is first presented, with a detailed description of the problem scenario. Afterwards, the cost minimization problem in this scenario is formulated.

#### 3.1 The cloud broker architecture for cloud bursting

The existing deployment models of cloud computing can be divided into the following three types [30]: *public* clouds, where the cloud infrastructure is made available for public development use; *private* clouds, in which the cloud infrastructure is dedicated to a single organization; *hybrid* clouds, in which the cloud infrastructure is a hybrid of several different cloud infrastructures. Cloud brokers serve as intermediaries between cloud providers



and users, providing multi-cloud based services to help users avoid vendor lock-in, i.e., addressing the lack of support for cloud interoperability and portability in cloud environments. Different from the traditional cloud broker model, the cloud broker model proposed in this paper not only provides services based on multiple cloud providers but also provides support for local data centers in private clouds, i.e., a hybrid cloud model based on a cloud broker.

The cloud broker architecture for cloud bursting is as shown in Figure 3. The cloud broker acts as a middleman between multiple users and multiple clouds, the extent of which is marked with a solid black line. After a user sends VM requests to a cloud broker, the cloud broker makes an immediate decision using the scheduler DeepBS. As different public clouds provide different user interfaces, specific adapters need to be implemented for different cloud providers, which are managed by Server Manager, enabling a uniform way of operating and monitoring servers in different clouds. DeepBS obtains an observed state for decision making by looking at the current user requests and the information provided by Server Manager on servers' usage state, which reflects the current state of the resources managed and requests received by the cloud broker. After feature extraction of the state by

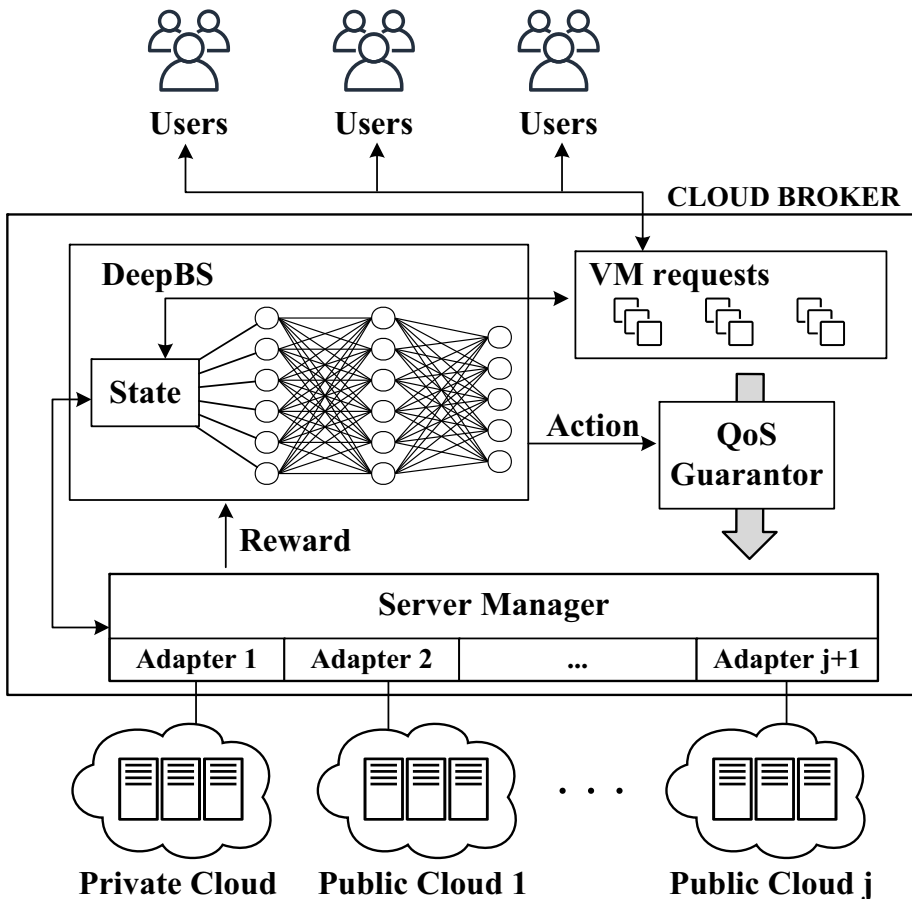


Figure 3 An overview of cloud broker system model based on DeepBS

a deep neural network, DeepBS gives an action which indicates the scheduling result of the VM request. As the DRL based approach learns through trial and error and exploration, DeepBS is subject to illegitimate scheduling at the start of training and in the case of fluctuating user requests. To ensure QoS requirements, a QoS Guarantor is introduced to ensure that requests are placed legally and to provide a penalty as feedback when DeepBS makes illegal decisions. Considering the demands of cloud bursting, DeepBS needs to perform VM scheduling in an online manner. Offline VM scheduling is scheduled against a batch of requests that have already arrived. In this way, information about all jobs can be obtained before the scheduling decision is made, thus enabling optimal or near-optimal decisions to be made. Unlike offline VM scheduling, online VM scheduling requires scheduling decisions to be made as soon as the requests arrive, which means that the scheduler can only make decisions based on available information.

This architecture considers user requests in the form of VMs. However, this architecture can be easily extended to include user requests in the form of containers and user requests in the form of tasks executed via containers. Container technology, as represented by Docker [31], is widely used as a lightweight virtualization technology with simple and efficient features compared to traditional virtual machines [32, 33]. For any service with on-demand characteristics, cloud bursting and multi-cloud capabilities can be obtained through this architecture, with adaptive cost savings through DeepBS. Based on lightweight virtualization technology, this architecture can also be extended to fog and edge computing nodes. Edge and fog computing, as extensions to cloud computing, offer end users lower latency compared to cloud computing. Existing edge computing nodes typically have similar prices to cloud computing but with fewer computing resources [34]. From a cost optimization perspective, edge computing nodes are not a necessary option, although they can be considered for this architecture. For latency-sensitive tasks edge nodes are valuable to consider, but it is beyond the scope of this paper and future work in this direction is discussed in Section 7.

### 3.2 Problem formulation

The objective of the VM scheduling problem in cloud brokers for cloud bursting presented in this paper is to minimize the total cost of computing resources, which includes: the cost of private clouds and the cost of an additional on-demand approach to public clouds. However, in order to meet user QoS demands and the on-demand characteristics, there are several requirements and limitations for the scheduler in this architecture. Firstly, the scheduler should ensure that the sum of all VM requests placed on the same server at any time does not exceed the capacity of that server. Secondly, the scheduler in the cloud broker needs to make real-time decisions on the placement of user VM requests, i.e., for a given user VM request, the cloud broker needs to immediately decide whether to place it on a server in a local private cloud or on a server in a public cloud. Finally, it is important to note that the knowledge of the cloud broker's scheduler is limited, as the scheduler only knows the configuration requirements of currently arriving VM requests and the usage of currently available computing resources, while it is difficult for the scheduler to obtain a prior knowledge of future arrivals and when current VMs will be released.

For each VM request, the cloud broker can place the request on a private cloud server, or on a server of one of the available cloud providers. In this paper, we define the set of VM requests from users as  $\mathcal{R} = \{r_i | 1 \leq i \leq R\}$ , where  $R = |\mathcal{R}|$  is the total number of user

**Table 2** Summary of main notations

Notation	Definition
$r_i$	the $i$ th arriving user VM request
$r_i^{para}$	the configuration requirement for the $para$ for the $i$ th arriving user VM request, $para \in \{CPU, RAM\}$
$r_i^{time}$	the time point at which the $i$ th arriving user VM request arrives/ends, $time \in \{arr, end\}$
$\mathcal{R}$	the set of all VM requests from users
$c_j$	the public cloud with index $j$
$\mathcal{M}$	the available VM types considered by the cloud broker
$M_p^i$	the VM types indexed as $i$ in the private cloud
$M_{c_j}^i$	the VM type indexed as $i$ in the public cloud $c_j$
$x_{r_j}$	the scheduling decision for request $r_i$
$b_{x,y}^{para}$	the capability of the $para$ of VM type $M_x^y$ , $para \in \{CPU, RAM\}$
$cost_{x,y}$	the price per unit of time of the VM type $M_x^y$
$Cost_{broker}(\mathcal{R})$	the total cost incurred by the cloud broker due to request set $\mathcal{R}$

requests. A user request is modelled as a tuple  $r_i = \langle r_i^{CPU}, r_i^{RAM}, r_i^{arr}, r_i^{end} \rangle$ , where  $r_i^{CPU}$  represents the request’s demand for CPU,  $r_i^{RAM}$  represents the demand for memory,  $r_i^{arr}$  is the arrival time of the request, and  $r_i^{end}$  is the end time of the request. Note that when scheduling online, the values of  $r_i^{CPU}$  and  $r_i^{RAM}$  are only known when the request arrives, while  $r_i^{arr}$  and  $r_i^{end}$  are hidden information to the scheduler because they are only known when the user makes a *create* and *destroy* request and requires an immediate response. Let  $\mathcal{M} = \{M_p^1, \dots, M_{c_j}^T\}$  be all the available VM types considered by the cloud broker, and  $T = |\mathcal{M}|$  be the number of all available VM types. For all  $M_x^y$ ,  $x$  represents the cloud which the server belongs to (the private cloud ( $p$ ), the  $i$ th of the  $j$  public clouds ( $c_i$ )), and  $y$  represents the index of the VM type in the cloud. Thus, for each container request  $r_i$ , the corresponding scheduling decision can be defined as:

$$x_{r_j} = M_x^y. \tag{1}$$

For an available VM  $m_x^y$ , its configuration information is modelled as the tuple  $\langle b_{x,y}^{CPU}, b_{x,y}^{RAM}, cost_{x,y} \rangle$ , where  $b_{x,y}^{CPU}$  is the total CPU capacity of  $m_x^y$ ,  $b_{x,y}^{RAM}$  is the total RAM capacity of  $m_x^y$ , and  $cost_{x,y}$  is the price per unit time of the VM type  $M_x^y$ .  $|M_x^y|$  is the number of VMs enabled for this category in this cloud provider.

Ultimately, the goal of the VM scheduling problem in a cloud broker is to find the optimal set of placement decisions for all VM requests so that the cloud broker spends the least amount of money. In this paper, a hybrid cloud scenario is considered, where the servers used to host VMs can come from both private clouds and multiple cloud providers. The cost considered therefore consists of two components, the local server cost and the cost paid to the cloud provider. However, the cost of a private cloud  $\sum_i^{|M_p|} cost_{p,i}$  can usually be considered as a fixed overhead compared to the cost of a pay-as-you-go public cloud, so the

cost minimization problem in a cloud broker architecture for cloud bursting is modelled as follows:

$$\min \quad \text{Cost}_{broker}(\mathcal{R}) = \sum_j^{|C|} \sum_i^{|M_{c_j}|} |M_{c_j}^i| \text{cost}_{c_j,i} \tag{2}$$

$$s.t. \quad \sum_i^{x_{r_i} = M_x^y, t \in [r_i^{arr}, r_i^{end}]} r_i^{CPU} \leq b_{x,y}^{CPU}, \forall t \in T, \forall M_x^y \in \mathcal{M} \tag{3}$$

$$\sum_i^{x_{r_i} = M_x^y, t \in [r_i^{arr}, r_i^{end}]} r_i^{MEM} \leq b_{x,y}^{MEM}, \forall t \in T, \forall M_x^y \in \mathcal{M} \tag{4}$$

$$x_{r_j} \in \mathcal{M}, \forall r_j \in \mathcal{R} \tag{5}$$

where  $\text{Cost}_{broker}(\mathcal{R})$  represents the total cost of the cloud broker facing the request set  $\mathcal{R}$  in this VM scheduling problem, with the constraint (3) limiting the total CPU demand of VMs placed on the same server at any one time to no more than the server’s CPU limit. Similarly, the constraint (4) limits the total memory requirements of VMs placed on the same server at any one time to the memory limit of that server. Finally, the constraint (5) defines a restriction on the deployment scheme that should be given for all requests. All symbols are summarized in Table 2.

### 4 DRL for Cost Minimization

In this section, we propose a model for solving the cloud broker VM scheduling problem with DRL to minimize costs.

DRL combines reinforcement learning with deep learning that can solve high-dimensional problems to become a general framework for solving decision-making problems with high-dimensional state space [29]. DRL learns by trial and error, with agents selecting actions by obtaining observations of the state from the environment. The environment will give a reward based on the impressions generated by the action given by the agent. The goal of RL or DRL is to maximize cumulative reward, i.e., to maximize  $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ , where  $\gamma \in [0, 1]$  is a discount factor, expressing the discount of future rewards to current impacts. Markov decision process (MDP) is defined as  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$ , where  $\mathcal{S}$  is the observation space,  $\mathcal{A}$  is the action space,  $\mathcal{R}$  is the set of rewards,  $\mathcal{P}$  is the distribution describing the MDP dynamics, and  $x^0$  is the initial state of the observation [35]. DRL is balanced between exploration and exploitation to obtain better decision strategies in MDP.

Considering the above VM scheduling problem, it is a non-trivial task to schedule user VM requests with uncertainty in an online way. Existing meta-heuristic algorithms are difficult to adapt to the online environment. We propose a DRL algorithm based cloud broker for making online VM placement decisions, DeepBS, and one of the main challenges in using DRL to solve a particular problem is how to model the problem as an MDP. We define the main concept of the DRL as follows:

- State space  $\mathbb{S}$ : In DeepBS, the state space is designed to express the current state of the system, including: the current level of idle servers available in multiple clouds, and the current VM requests that need to be processed. For each of the server resource

categories to be considered within the scenario, e.g. CPU, memory size, etc., the observation of servers includes the number of resources available in each category of server currently available in the cloud broker's computing resources. The state representation of computing resources in a cloud broker can be denoted as

$$s_{\mathcal{M}} := [(s_{p,1}^{CPU}, s_{p,1}^{RAM}), \dots, (s_{c_j,x}^{CPU}, s_{c_j,x}^{RAM})] \in \mathbb{R}^{2(j+1)}. \quad (6)$$

For a VM request, the information about this request is represented by its demand for each of the server resource categories. The state representation of VM requests in a cloud broker can be denoted as

$$s_{\mathcal{R}} := [(r_i^{CPU}, r_i^{RAM}), \dots, (r_{i+k-1}^{CPU}, r_{i+k-1}^{RAM})] \in \mathbb{R}^{2k}. \quad (7)$$

Therefore, at time  $t$ , the state of DeepBS is represented as

$$s_t = [s_{\mathcal{M}}, s_{\mathcal{R}}] \in \mathbb{R}^{2(j+k+1)}. \quad (8)$$

- Action space  $\mathbb{A}$ : The action space represents the optional scheduling result for DeepBS. For each action, DeepBS chooses to make a scheduling decision for the first VM request at the head of the current queue. The optional action spaces include: actively dropping the request, placing the request on a server in a private cloud, and placing the request on a server in a public cloud, i.e.,

$$a_t = \begin{cases} -1 \\ M_p^t, t \in T \\ M_{c_j}^t, j \in |C|, t \in T. \end{cases} \quad (9)$$

In reinforcement learning, the execution of each action means the end of the current time step and the acquisition of the next state and a reward signal. It is important to distinguish between time step in reinforcement learning and real-world time. In DeepBS scheduling is not done with a fixed time period as the decision cycle, but performed in an event-driven manner. When a VM request arrives, DeepBS needs to make an immediate response and decision. Consider the case where requests arrive centrally or where a single user makes multiple VM requests at the same time, and place the requests that arrive at the same time in a queue to make decisions in turn. For requests arriving at the same time DeepBS needs to acquire multiple states and perform multiple actions in a short period of time in succession. However, as it takes some time for the VM to start and the image to be transferred, it is possible that the creation of the VM requested in the previous request is not completed by the time a new decision needs to be made. To solve this problem, DeepBS takes into account the already reserved computational resources of the VM being created when acquiring a new state.

- Reward  $\mathbb{R}$ : Reward signals are designed to motivate the agent to learn policies that are effective in this environment, and the design of rewards in DeepBS is based on the definition of the cloud broker VM scheduling optimization problem. The cost used for the private cloud is considered fixed in the objective function and the variable lies in the additional on-demand public cloud cost. For this reason, the reward is designed to signal a negative value of the on-demand cost as a penalty for the additional cost. Its accumulation is the total amount of public cloud cost. Therefore, to guide the agent to

learn the least costly scheduling policy, we set its reward to the negative of the public cloud cost incurred in the last time slot, i.e.,

$$r_t = - \sum_{c_j \in C} \sum_{t \in T} |M_{c_j}^t| \text{cost}_{c_j,t}, \quad (10)$$

where  $|M_{c_j}^t|$  is the number of VMs of type  $t$  used in public cloud provider  $c_j$  in the previous time slot and  $\text{cost}_{c_j,t}$  is the price of one time slot of VMs of type  $t$  used in public cloud provider  $c_j$ .

At this point, we have given a mapping of the cost minimization problem in a cloud broker to a form of MDP that can be solved using DRL. However, there is one more challenge that needs to be addressed. As mentioned earlier, reinforcement learning methods learn through trial and error in the environment. Therefore, it is common for a reinforcement learning agent to take to an illegitimate action. This illegitimacy refers to the fact that the action sampled by the agent may not meet the QoS requirements of the user. For most work applying reinforcement learning to the cloud resource scheduling problem all actions are set to be legal, i.e. ignoring the QoS requirements of users that are common in the cloud. This is possible in some scenarios, such as solving workflow scheduling problems where the agent takes an ill-advised action that causes the workflow to take too long to execute, and the poor results from that action are reflected in the reward. However, in the cost minimization problem in a cloud broker, if the agent takes an illegal action, such as allocating a smaller resource to a user's request than the user requirements, or not allocating any resources to the request, then the result in the reward will be a lower public cloud cost, or even no public cloud cost at all. In other words, the agent can profit from the QoS violations, which will result in the reinforcement learning agent to learn a policy that we do not expect.

---

```

1: Initialize: Current VM request to be scheduled  $r_j$ , penalty  $p$ 
2:  $a_t := \text{DeepBS.step}(s_t)$ ;
3: cluster state := ServerManager.getCurrentClusterState();
4: allocated resource := getResource(cluster state, a_t);
5: if allocated resources do not meet the QoS required by  $r_j$  then
6:   return 0;
7: else
8:   Rent additional public cloud VM for  $r_j$ ;
9:   return penalty  $p$ ;
10: end if

```

---

**Algorithm 1** QoS Guarantor.

For this problem, we propose the QoS Guarantor, which is used to ensure that each scheduling policy satisfies the request requirements specified by the user and to guide the reinforcement learning agent so that it does not fall into a local optimum solution generated by a wrong scheduling decision. The QoS Guarantor is shown in Algorithm 1. When the QoS Guarantor determines that the action currently taken by the reinforcement learning agent is not satisfying the QoS requirements of the corresponding request, the QoS Guarantor processes the request separately and gives a penalty  $p$ . The penalty  $p$  far exceeds the public cloud cost incurred by the request. In this way, the reinforcement learning agent is

informed that the action it has taken is wrong. As a result, the reinforcement learning agent will learn the least costly VM scheduling policy in the space of actions that satisfy QoS.

## 5 DRL-based online VM scheduler

In this section, we present DeepBS, a cost-optimized hybrid cloud VM scheduling algorithm for cloud brokers oriented to cloud bursting based on proximal policy optimization (PPO). PPO is an improvement of trust region policy optimization (TRPO) [36] with some advantages, but is simpler to implement and more general with better empirical sample complexity and runtime complexity. Specifically, the proposed algorithm uses the actor-critic RL framework and constrains the update size by trust region [37] to reduce the policy method variance. DeepBS uses two neural networks to construct actor and critic, learning both the actor's policy function  $\pi(\cdot|s)$  and the critic state value function  $V^\pi(s)$ . DeepBS selects the scheduling decision action  $a$  from the action space by observing the current system state  $s$ , and optimizes the DeepBS scheduling policy after obtaining the reward signal  $r$  given by the environment.

In our algorithm a combination of value-based DRL algorithm and policy-based DRL algorithm is used, and they use different networks respectively. Among them, the value-based DRL algorithm determines the value function by using a neural network as function approximator. The policy-based DRL algorithm learns the probabilities of the parameterized policy output actions directly instead of storing Q-values. Therefore, it has the ability to solve continuous action spaces and high-dimensional action spaces compared to the value-based reinforcement learning algorithm. The policy-based reinforcement learning algorithm performs the following updating each time:

$$\theta = \theta + \alpha \nabla_\theta J(\theta) \quad (11)$$

for performance objective  $J(\theta)$  and learning rate  $\alpha \in \mathbb{R}^+$ , and the goal of reinforcement learning is to maximize  $J(\theta)$ . To achieve this, according to the Policy Gradient Theorem [38], our optimization objective and its gradient are denoted as

$$J(\theta) = v_{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim \pi_\theta}(R(\tau)) = \int_{\tau} p(\tau|\theta) R(\tau), \quad (12)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t'=0}^{\infty} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \sum_{t'=0}^{\infty} \gamma^{t-t'} R_t \right]. \quad (13)$$

In order to avoid large variance in the estimation of the gradient, the benchmark function is introduced as an estimate of the state  $S_i$  value, and the gradient of the optimization objective after the introduction of the benchmark function  $b(S_i)$  can be expressed as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t'=0}^{\infty} \nabla_\theta \log \pi_\theta(A_{t'}|S_{t'}) \left( \sum_{t'=0}^{\infty} \gamma^{t-t'} R_t - b(S_{t'}) \right) \right]. \quad (14)$$

The actor-critic RL framework is used in DeepBS to estimate the Q-value function by using critic’s state value function as a benchmark function and using the idea of bootstrapping, so the advantage function  $A^{\pi_\theta}(s, a)$  as well as the gradient  $\nabla_\theta J(\theta)$  is

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V_\psi^{\pi_\theta}(s), \tag{15}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t'=0}^{\infty} \nabla_\theta \log \pi_\theta(A_{t'} | S_{t'}) A^{\pi_\theta}(S_t, A_t) \right]. \tag{16}$$

However, for DeepBS, the action space is an alternative scheduling scheme, which generates a very large action space when considering larger clusters and multiple cloud providers, and creates more challenges for optimization of reinforcement learning problems. One of the problems with the traditional method is that the training is unstable, because an inappropriate step size can lead to too slow or too much variance in the training. To address this challenge, a policy gradient algorithm called TRPO is proposed, which reduces the policy method variance by using the advantage function and an idea based on a trust region to better handle the step size. The optimization objective is to maximize the loss function

$$\mathcal{L}_{\pi_\theta}(\pi'_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t'=0}^{\infty} \frac{\pi_\theta(A_t | S_t)}{\pi_{\theta_{old}}(A_t | S_t)} A^{\pi_\theta}(S_t, A_t) \right] \tag{17}$$

while constraining the old and the new policy being not too far away, which is achieved by using Kullback-Leibler divergence. The optimization problem of TRPO can be expressed as

$$\begin{aligned} \max_{\pi'_\theta} \quad & \mathcal{L}_{\pi_\theta}(\pi'_\theta) \\ \text{s.t.} \quad & \mathbb{E}_{s \sim \rho_{\pi_\theta}} [KL(\pi_{\theta_{old}}, \pi_\theta)] \leq \delta. \end{aligned} \tag{18}$$

The ability to apply reinforcement learning algorithm to complex, high-dimensional problems is empowered by solving this form of optimization problem, and original experiments show that this approach has robust performance in many high-dimensional complex scenarios. This policy gradient method has superior problem solving ability. However, the solution of its optimization problem requires the calculation of  $H^{-1}g$  with a complexity of  $O(N^3)$ , and in practice the conjugate gradient method is commonly used to approximate this value, which still requires the execution of the conjugate gradient method several times at each parameter update, which makes the method computationally complex and complicated to implement.

DeepBS considers the real-time nature of the scheduling task and adopts a PPO-based approach to limit the changes in policy updates. Compared to TRPO, PPO has some advantages of TRPO and has a simple format and high computational efficiency to be used for large-scale problems, and multiple updates for small batches of data also improve the efficiency of data utilization. It has the same motivation as TRPO, i.e., to take the maximum possible improvement in the policy parameters without causing a performance crash. One approach is to directly clip the objective function used for the policy gradient, resulting in a more conservative update. The clipping-based objective is as follows:

$$\mathcal{L}^{PPO-Clip}(\pi'_\theta) = \mathbb{E}_{\pi_\theta} [\min(r_t(\theta)A^{\pi_\theta}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A^{\pi_\theta})], \tag{20}$$

where  $r_t(\theta) = \frac{\pi_\theta(A_t | S_t)}{\pi_{\theta_{old}}(A_t | S_t)}$  is the ratio of the probability under the new and old policies. Another approach is to use the constraint (18) in the optimization problem (19) as a penalty term for the optimization objective, i.e., to optimize the regularized version of the problem. The KL-penalized objective is as follows:

$$\mathcal{L}^{PPO-Penalty}(\pi'_\theta) = \mathbb{E}_{\pi_\theta} [r_t(\theta)A^{\pi_\theta} - \beta KL(\pi_{\theta_{old}}, \pi_\theta)] \tag{21}$$



---

```

1: Initialization: policy parameters  $\theta_0$ , value function parameters  $\phi_0$ 
2: for each training epoch do
3:   Empty trajectory buffer  $\mathcal{D} = \{\}$ ;
4:   while a user VM request  $r_i$  arrives do
5:     Observe  $s_{\mathcal{M}}$  from Server Manager;
6:     Observe  $s_{\mathcal{R}}$  from the request queue;
7:      $s_{r_j} := [s_{\mathcal{M}}, s_{\mathcal{R}}]$ ;
8:     Policy network gives action  $a_{r_j}$  based on  $s_{r_j}$ ;
9:     Initialization  $r_{r_j}$ ;
10:     $r_{r_j} += \text{QoS Guarantor}(a_{r_j})$ ;
11:    if Reaching a new billing cycle then
12:      Server Manager gives  $c_t$  based on public cloud cost;
13:       $r_{r_j} += c_t$ ;
14:    end if
15:    Add  $s_{r_j}, a_{r_j}, r_{r_j}$  to  $\mathcal{D}$ ;
16:  end while
17:  for for  $i = \mathcal{D}.length$  to 1 do
18:     $r_t(\theta') = \pi_{\theta}(A_t|S_t)/\pi_{\theta_{old}}(A_t|S_t)$ 
19:     $\theta = \arg \max_{\theta} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T \min(r_t(\theta)A^{\pi_{old}}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A^{\pi_{old}} - \beta KL(\pi_{\theta_{old}}, \pi_{\theta}))$ ;
20:     $\phi = \arg \max_{\phi} \frac{1}{|\mathcal{D}|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^T (V_{\phi}(S_t) - \hat{G}_t)^2$ ;
21:  end for
22: end for

```

---

**Algorithm 2** DeepBS: a DRL-based online VM scheduler for cloud brokers.

DeepBS uses a hybrid approach combining *PPO-Clip* and *PPO-Penalty*, i.e., we use both a clipping-based approach and a KL-penalized approach, thus reducing the complexity of the computation while ensuring the stability of updates from  $\pi_{\theta}$  to  $\pi'_{\theta}$ . The optimization objectives of DeepBS is:

$$\mathcal{L}(\pi'_{\theta}) = \mathbb{E}_{\pi_{\theta}}[\min(r_t(\theta)A^{\pi_{\theta}}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A^{\pi_{\theta}}) - \beta KL(\pi_{\theta_{old}}, \pi_{\theta})] \quad (22)$$

Finally the whole process of the algorithm is summarized in Algorithm 2. To begin with, DeepBS initializes the parameters of the Actor and Critic neural networks separately. As previously stated, DeepBS is event-driven. It makes a decision as soon as a user VM request arrives. The state of the system including the state of computing resources and the state of the VM requests is obtained from the Server Manager and the request queue respectively. The policy network then takes an action. Considering DeepBS learns through trial and error so that there is a possibility of giving illegal actions, a QoS Guarantor is required to determine the action's legality. For legal actions, the scheduling will be performed according to the action. For illegal actions, that is, actions that violate QoS, the QoS Guarantor notifies the Server Manager to use a separate server to accommodate the request and give DeepBS a large penalty. Server Manager periodically gives a reward that reflects the current cloud broker cost in the public cloud in accordance with the public cloud billing method. For

each request, the state, action and reward are saved and used to periodically update network parameters to optimize DeepBS.

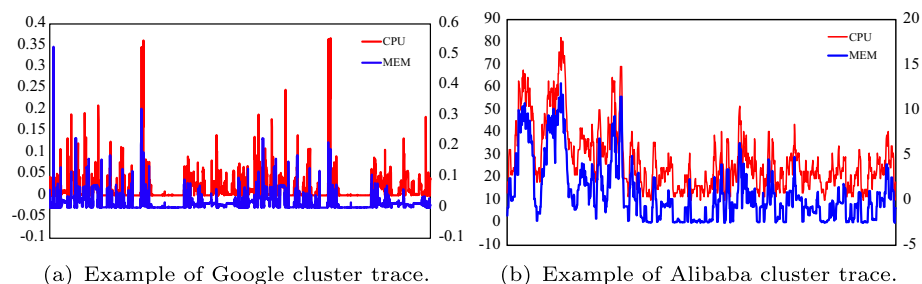
## 6 Performance analysis

In this section, we evaluate the DeepBS proposed in this paper. The details of the experimental setting and benchmark algorithms are first described, followed by a comparison and analysis of the performance of DeepBS.

### 6.1 Experiment settings

We implemented DeepBS using Pytorch 1.10 [39] and evaluated it using two real traffic trace datasets. Cloud brokers providing on-demand services are faced with several aspects of uncertainty regarding user requests. This uncertainty is in terms of the arrival time of the request, the demand for the request, and the end time of the request. Different workloads have a significant impact on the performance of the scheduling policy. For this reason, two different workloads based on real cluster traces are used in our experiment as simulations of user requests. The first workload used is based on Google cluster trace data [40]. Specifically, we set the arrival patterns of the requests in the experiment and the request sizes based on several days of tracing of resource-intensive workloads in the Borg clusters published by Google. In addition, to consider the scheduler to be generalizable and effective in other workload scenarios, another workload used in the experiments comes from the Alibaba cluster trace dataset [41]. This dataset records the tracking of requests for an Alibaba production cluster, where each machine in the cluster is required to handle long-running online services and batch workloads. Figure 4 shows examples of partial requests for two different workloads, with real-world user requests reflecting more pronounced fluctuations. For the two different workloads, the scheduler only knows the demand for the request when it arrives and the scheduler does not know when the request will end until the request actually leaves.

We developed a real cluster tracking-driven cloud broker environment based on OpenAI Gym [42] to train and evaluate the DRL-based cloud broker VM placement scheduler proposed in this paper. Heterogeneous cloud computing resources are considered in the Gym-based cloud broker environment we built. In a cloud broker architecture that considers cloud bursting, the computing resources consist of a private cloud and multiple public clouds. In the experiments, multiple public cloud instance categories are considered, as



**Figure 4** Examples of partial VM requests for each of the two types of workloads used in our experiments

exemplified by the instance type t3 in EC2 in AWS [43], whose configuration and pricing are shown in Table 3.

We evaluate the performance of DeepBS using three heuristics and two DRL-based algorithms as comparisons. Among them, several models of DRL are selected from methods commonly used in the cloud computing field to solve scheduling problems. To make them applicable to the optimization problems considered in this paper, these models use the same state, action, and reward as DeepBS. Next, we briefly describe how these algorithms are implemented.

- Dominant Resource Fairness (DRF) [44]: The default scheduling policy in YARN [45] and Mesos [46] is a fairness-based scheduling procedure that achieves maximum-minimum fairness of the dominant resource when work arrives and when work is completed.
- The Best-Fit (BF) [47]: The BF algorithm selects the server with the smallest remaining capacity that can satisfy the request.
- Tetris [48]: It prioritizes resource allocation to requests with short duration or low resource consumption and pack requests to the server to minimize resource fragmentation.
- Asynchronous Advantage Actor-Critic (A3C) [49]: A3C is an asynchronous variant of Actor-Critic that uses asynchronous gradient descent to optimize deep neural network controllers.
- Dueling Double DQN (D3QN): D3QN combines the advantages of Dueling DQN [50] and Double DQN [51]. It is an improved algorithm for DQN [52] with faster convergence and more stable convergence.

The DRL algorithms described above all use a standard implementation of RLlib [53] based on PyTorch 1.10 [39], and the hyper-parameters are set as follows: the learning rate is set to  $5e-5$ , the discount factor  $\gamma$  is 0.94, batch size is 200, layers are [256, 256], optimizer is Adam, and the rest of the hyper-parameters are the default values in RLlib.

## 6.2 Performance study

In this subsection, the performance of DeepBS under different workloads is discussed. First the convergence results of DeepBS with different parameter settings are presented, then the performance of the algorithm is analyzed under two real cluster trace and finally an analysis of the performance of the algorithm under more extreme request arrival patterns is given.

**Table 3** Amazon EC2 on-demand pricing [43]

Instance name	On-Demand hourly rate	vCPU	Memory
t3.nano	0.0052 USD	2	0.5GiB
t3.micro	0.0104 USD	2	1GiB
t3.small	0.0208 USD	2	2GiB
t3.medium	0.0416 USD	2	4GiB
t3.large	0.0832 USD	2	8GiB
t3.xlarge	0.1664 USD	4	16GiB
t3.2xlarge	0.3328 USD	8	32GiB

### 6.2.1 Convergence results with different parameter settings

We train our model based on the request arrival pattern generated by Google cluster tracking data and first evaluate the effect of three basic parameters on the convergence rate and results, i.e. learning rate, discount factor and train batch size. The results are shown in Figure 5.

First, we investigate the effect of different learning rates, and the results are shown in Figure 5(a). In this investigation, we set the other parameters to their default values. Larger learning rates lead to larger episode rewards more quickly. However, high learning rates are not always beneficial for training, setting the learning rate to 0.01 or 5e-4 resulted in the agent acquiring fewer rewards and converging to a local optimum, we investigate the discount factor, which represents how the agent evaluates future rewards. A too large discount factor means that the agent is considering too long term rewards even beyond the influence of the current action, and a too large discount factor also risks having too large Q-value. In this investigation, DeepBS sets the gamma to 0.9 to obtain the best performance in terms of convergence and convergence results. Finally, we investigate the effect of train batch size on training, which represents the amount of data used per update when updating using the optimizer. Larger batch sizes require more epochs to converge to the minimum validation loss and generalise less well to the test data. For this reason, we chose a relatively small train batch size = 2000 for the training of the model.

Figure 6 records the number of times during training that the agent produces a QoS violation and is given a penalty term by the QoS Guarantor. As training progresses, the metric converges after 100 thousand episodes, which is not completely cleared due to the fact that the RL agent still maintains action exploration during training, with a certain probability

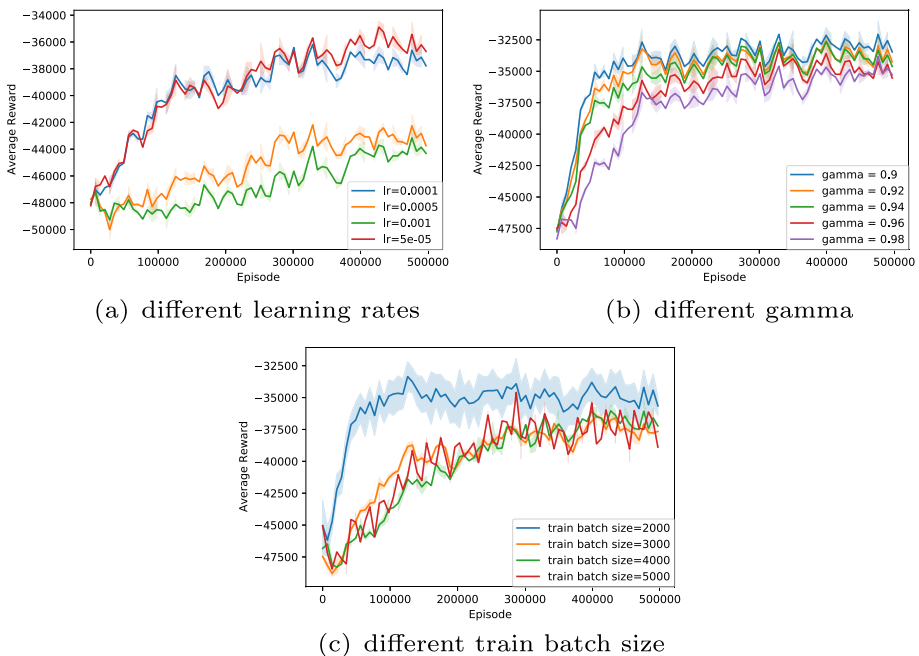
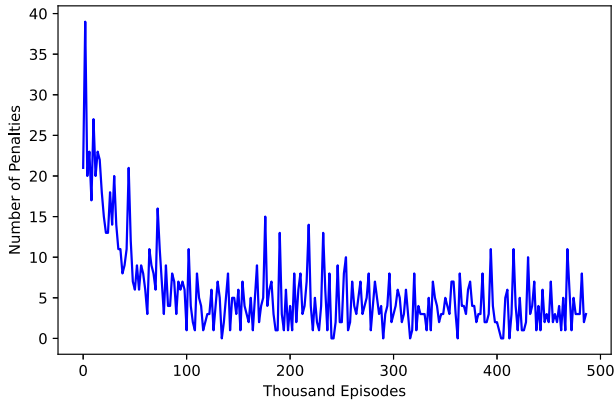


Figure 5 Moving average episode reward with respect to different parameter settings



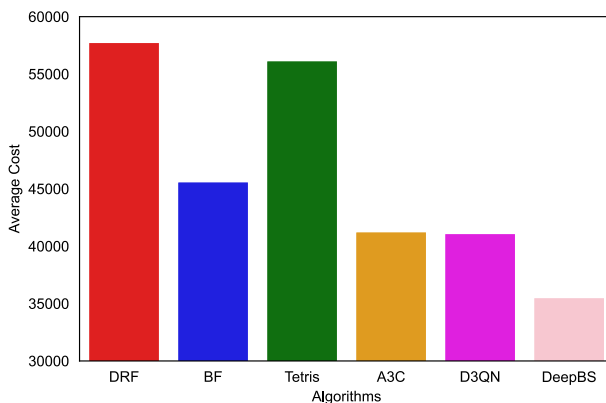
**Figure 6** Number of penalties given by the QoS guarantor during training

that the current policy considered non-optimal to make a random choice among the actions. This demonstrates the effectiveness of the QoS Guarantor in guiding the DRL agent to learn to avoid actions that violate the user’s requested QoS.

### 6.2.2 Performance analysis of cluster trace-based requests

After completing the training of the model, we verify the performance of DeepBS under different request arrival patterns based on real cluster tracking data. Figure 7 shows the performance of DeepBS compared to other benchmark algorithms for the request arrival pattern based on Google cluster tracking, with DeepBS having the best performance in terms of cost for the same request arrival pattern and total request volume. Compared to the next best reinforcement learning strategy, DeepBS has a 13.57% cost optimization. Compared to the optimal heuristic, DeepBS has a 22.11% cost improvement.

We also use the Alibaba cluster trace dataset as a model for request arrivals that can reflect the fluctuations in the distribution of requests in the real world, and it is clear that the requests of users in the real dataset are not uniform and do not have specific rules. As



**Figure 7** Performance of different algorithms for experiments with Google cluster trace

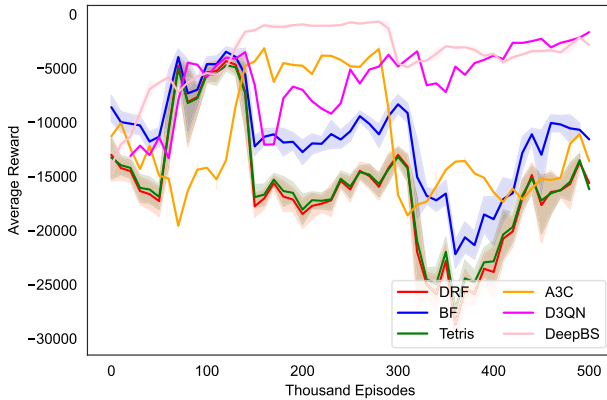


Figure 8 Training trajectories of different algorithms in Alibaba cluster trace

shown in Figure 8, several benchmark heuristics fluctuate considerably in this type of user request, due to significant fluctuations in the arrival patterns of the requests. Although real user requests have more significant fluctuations, the DeepBS we propose still has robust performance through online learning. DeepBS has a stable performance relative to heuristics when user requests fluctuate. Figure 9 shows the performance of DeepBS compared to other benchmark algorithms at real user requests. The DeepBS algorithm significantly outperforms other DRL algorithms as well as heuristic algorithms in terms of cost. DeepBS has a 27.84% performance advantage over the best performing benchmark algorithm in terms of cost optimization in both real cluster-based tracking request arrival modes.

### 6.2.3 Performance analysis based on extreme requests

The above experiments analyze the performance of DeepBS in two request arrival patterns based on real cluster data tracking and demonstrate the effectiveness of our proposed algorithm in cost optimization. In order to consider the scalability and robustness of the algorithm, we evaluate the performance of the algorithm in more extreme cases.

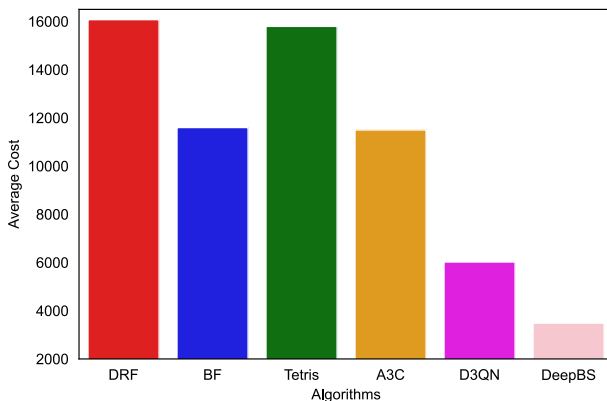
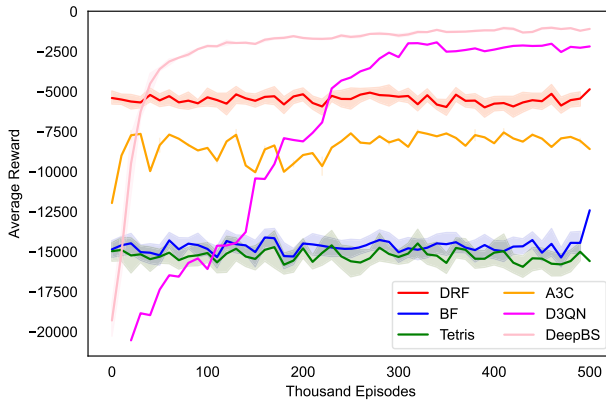


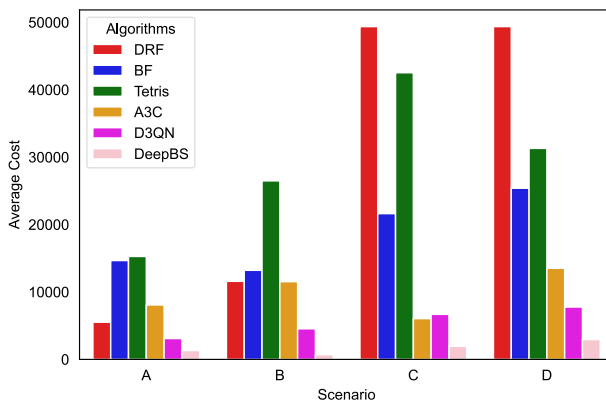
Figure 9 Performance of different algorithms for experiments with Alibaba cluster trace



**Figure 10** Training trajectories of different algorithms in simulation experiments based on Poisson distribution

We set up four scenarios with more extreme arrival patterns than the requests based on real cluster traces, where scenario A has the same request arrival pattern as the Alibaba cluster trace, but each request has a longer duration. The job arrival patterns generated based on Poisson distribution, although simulating the arrival pattern, request duration and request demand size of Alibaba cluster trace, are however more stable compared to real data and the training trajectory of each algorithm is shown in Figure 10. As requests continue to arrive, heuristics have relatively fluctuating but stable performance, with fluctuations reflecting the uncertainty of user request arrival, demand and departure. DRL-based approaches learn through trial and error, ultimately achieving a better performance than heuristics. DeepBS can converge faster and has better performance than other DRL methods.

In scenarios B, C and D, we consider scenarios with more request arrivals and larger clusters. While scenario B has a larger cluster, the request arrival density for scenarios C and D is set to twice the Alibaba cluster trace data to analyze the performance of the scheduler in a more intensive request arrival environment. The performance of heuristic algorithms decreases significantly with the complexity of the scene and the number of requests. On the



**Figure 11** Performance of different algorithms for Poisson distribution-based simulation experiments in a variety of different settings

other hand, DeepBS can converge faster than other DRL algorithms in different scenarios. Figure 11 shows the performance of simulation experiments based on Poisson distribution in different scenarios, from which we can see that DeepBS has a significant performance advantage.

## 7 Conclusion and future work

In this paper, we propose DeepBS, an online VM scheduler in a cloud broker for cloud bursts by learning from experience to adaptively improve scheduling strategies in environments with non-smooth and uncertain user requests. Our proposed DeepBS also solves the problem of reinforcement learning algorithms getting stuck in locally optimal solutions that produce QoS violations by means of QoS Guarantor. In comparison to classical scheduling strategies, DeepBS is capable of solving online scheduling problems more effectively by adaptively learning and adjusting scheduling policies in response to workload fluctuations. We evaluate the performance of our proposed algorithm under two request arrival patterns based on Google cluster trace and Alibaba cluster trace. The experimental analysis shows that DeepBS has a significant advantage in cost optimization over several heuristics and several other reinforcement learning algorithms, and still maintains optimal performance in simulation scenarios with more extreme request arrival patterns.

In this paper, we consider the cloud broker VM scheduling problem to minimize public cloud costs while avoiding QoS violations. However, there are some problems that are not considered in this paper, and some specific cloud services often require consideration of other objectives, such as latency. For end users, edge computing has lower latency compared to cloud computing, but there are limitations in terms of computing resources. Considering both cost-optimized and latency-optimized scheduling issues will be a direction for future work.

**Authors' contributions** Xingjia Li wrote the main manuscript text, prepared all figures and tables and provided the methodology. Li Pan and Shijun Liu provided writing-review and editing and provided funding support.

**Funding** The authors would like to acknowledge the support provided by the Innovation Method Project of Ministry of Science & Technology, China under Grant 2020IM020100, the Key Research & Development Program of Shandong Province (2020CXGC010102), and project ZR2020LZH011 supported by Shandong Provincial Natural Science Foundation.

**Availability of data and materials** All of the materials including figures is owned by the authors and no permissions are required. The initial dataset is from the public dataset of website <https://github.com/google/cluster-data/> and <https://github.com/alibaba/clusterdata/>.

## Declarations

**Ethical approval** This declaration is not applicable.

**Competing interests** I declare that all authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and discussion reported in this paper.



## References

- Hsu, P.-F., Ray, S., Li-Hsieh, Y.-Y.: Examining cloud computing adoption intention, pricing mechanism, and deployment model. *Int. J. Inf. Manag.* **34**(4), 474–488 (2014). <https://doi.org/10.1016/j.ijinfomgt.2014.04.006>
- Singh, B., Dhawan, S., Arora, A., Patail, A.: A view of cloud computing. *Int. J. Comput. Technol.* **4**(2b1), 50–58 (2013)
- Leavitt, N.: Is cloud computing really ready for prime time? *Computer* **42**(1), 15–20 (2009). <https://doi.org/10.1109/CMC.2009.20>
- Lucas-Simarro, J.L., Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Scheduling strategies for optimal service deployment across multiple clouds. *Futur. Gener. Comput. Syst.* **29**(6), 1431–1441 (2013). <https://doi.org/10.1016/j.future.2012.01.007>. Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems
- Li, X., Pan, L., Liu, S.: A survey of resource provisioning problem in cloud brokers. *J. Netw. Comput. Appl.* **203**, 103384 (2022). <https://doi.org/10.1016/j.jnca.2022.103384>
- Cloud Services Brokerage Market Research Report by Service Type, by Platform, by Deployment Model, by Region - Global Forecast to 2026 - Cumulative Impact of COVID-19. <https://www.researchandmarkets.com/reports/4989790/cloud-services-brokerage-market-research-report#rela0-4985613>. Accessed 29 Nov 2022
- State of the Cloud Report Survey. <https://info.flexera.com/CM-REPORT-State-of-the-Cloud>. Accessed 29 Nov 2022
- Nair, S.K., Porwal, S., Dimitrakos, T., Ferrer, A.J., Tordsson, J., Sharif, T., Sheridan, C., Rajarajan, M., Khan, A.U.: Towards secure cloud bursting, brokerage and aggregation. In: 2010 Eighth IEEE European Conference on Web Services, pp. 189–196. <https://doi.org/10.1109/ECOWS.2010.33> (2010)
- Song, F., Xing, H., Wang, X., Luo, S., Dai, P., Li, K.: Offloading dependent tasks in multi-access edge computing: a multi-objective reinforcement learning approach. *Futur. Gener. Comput. Syst.* **128**, 333–348 (2022). <https://doi.org/10.1016/j.future.2021.10.013>
- Caviglione, L., Gaggero, M., Paolucci, M., Ronco, R.: 19. *Soft. Comput.* **25**, 12569–12588 (2021). <https://doi.org/10.1007/s00500-020-05462-x>
- Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks. HotNets '16, pp. 50–56. Association for Computing Machinery. <https://doi.org/10.1145/3005745.3005750> (2016)
- Tong, Z., Chen, H., Deng, X., Li, K., Li, K.: A scheduling scheme in the cloud computing environment using deep q-learning. *Inf. Sci.* **512**, 1170–1191 (2020). <https://doi.org/10.1016/j.ins.2019.10.035>
- Rolik, O., Zharikov, E., Koval, A., Telenyk, S.: Dynamic management of data center resources using reinforcement learning. In: 2018 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), pp. 237–244. <https://doi.org/10.1109/TCSET.2018.8336194> (2018)
- Long, S., Li, Z., Xing, Y., Tian, S., Li, D., Yu, R.: A reinforcement learning-based virtual machine placement strategy in cloud data centers. In: 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 223–230. <https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00028> (2020)
- Sheng, J., Hu, Y., Zhou, W., Zhu, L., Jin, B., Wang, J., Wang, X.: Learning to schedule multi- numa virtual machines via reinforcement learning. *Pattern Recognit.* **121**, 108254 (2022). <https://doi.org/10.1016/j.patcog.2021.108254>
- Li, Y., Tang, X., Cai, W.: On dynamic bin packing for resource allocation in the cloud. In: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures. SPAA '14, pp. 2–11. Association for Computing Machinery. <https://doi.org/10.1145/2612669.2612675> (2014)
- Li, Y., Tang, X., Cai, W.: Dynamic bin packing for on-demand cloud resource allocation. *IEEE Trans. Parallel Distrib. Syst.* **27**(1), 157–170 (2016). <https://doi.org/10.1109/TPDS.2015.2393868>
- Azar, Y., Vainstein, D.: Tight bounds for clairvoyant dynamic bin packing. *ACM Trans. Parallel Comput.* **6**(3). <https://doi.org/10.1145/336> (2019)
- Gharehpasha, S., Masdari, M.: A discrete chaotic multi-objective sca-alo optimization algorithm for an optimal virtual machine placement in cloud data center. *J. Ambient. Intell. Humanized Comput.* **12**(10), 9323–9339 (2021). <https://doi.org/10.1007/s12652-020-02645-0>
- Ghetas, M.: A multi-objective monarch butterfly algorithm for virtual machine placement in cloud computing. *Neural Comput. & Applic.* **33**(17), 11011–11025 (2021). <https://doi.org/10.1007/s00521-020-05559-2>

21. Zhao, J., Rodríguez, M.A., Buyya, R.: A deep reinforcement learning approach to resource management in hybrid clouds harnessing renewable energy and task scheduling. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), pp. 240–249. <https://doi.org/10.1109/CLOUD53861.2021.00037> (2021)
22. Silva Filho, M.C., Monteiro, C.C., Inácio, P.R.M., Freire, M.M.: Approaches for optimizing virtual machine placement and migration in cloud environments: a survey. *J. Parallel Distrib. Comput.* **111**, 222–250 (2018). <https://doi.org/10.1016/j.jpdc.2017.08.010>
23. Moghe, U., Lakkadwala, P., Mishra, D.K.: Cloud computing: Survey of different utilization techniques. In: 2012 CSI Sixth International Conference on Software Engineering (CONSEG), pp. 1–4. <https://doi.org/10.1109/CONSEG.2012.6349524> (2012)
24. Cong, P., Xu, G., Wei, T., Li, K.: A survey of profit optimization techniques for cloud providers. *ACM Comput. Surv.* **53**(2), 26 (2020). <https://doi.org/10.1145/3376917>
25. Ranjana, R., Raja, J.: A survey on power aware virtual machine placement strategies in a cloud data center. In: 2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE), pp. 747–752. <https://doi.org/10.1109/ICGCE.2013.6823533> (2013)
26. Zhang, J., Huang, H., Wang, X.: Resource provision algorithms in cloud computing: a survey. *J. Netw. Comput. Appl.* **64**, 23–42 (2016). <https://doi.org/10.1016/j.jnca.2015.12.018>
27. Kumar, M., Sharma, S.C., Goel, A., Singh, S.P.: A comprehensive survey for scheduling techniques in cloud computing. *J. Netw. Comput. Appl.* **143**, 1–33 (2019). <https://doi.org/10.1016/j.jnca.2019.06.006>
28. Zhou, G., Tian, W., Buyya, R.: Deep Reinforcement Learning-based Methods for Resource Scheduling in Cloud Computing: A Review and Future Directions. *arXiv* (2021). <https://doi.org/10.48550/ARXIV.2105.04086>. <https://arxiv.org/abs/2105.04086>
29. Plaat, A.: *Deep Reinforcement Learning*. Springer (2022). <https://doi.org/10.1007/978-981-19-0638-1>
30. Mell, P., Grance, T.: *The NIST Definition of Cloud Computing*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD (2011). <https://doi.org/10.6028/NIST.SP.800-145>
31. Empowering App Development for Developers Docker. <https://www.docker.com/>. Accessed 29 Nov 2022
32. Bhimani, J., Yang, Z., Mi, N., Yang, J., Xu, Q., Awasthi, M., Pandurangan, R., Balakrishnan, V.: Docker container scheduler for i/o intensive applications running on nvme ssds. *IEEE Trans. Multi-Scale Comput. Syst.* **4**(3), 313–326 (2018). <https://doi.org/10.1109/TMSCS.2018.2801281>
33. Ramos, F., Viegas, E., Santín, A., Horchulhack, P., dos Santos, R.R., Espindola, A.: A machine learning model for detection of docker-based app overbooking on kubernetes. In: ICC 2021 - IEEE International Conference on Communications, pp. 1–6. (2021). <https://doi.org/10.1109/ICC42927.2021.9500259>
34. AWS Wavelength brings AWS compute and storage capabilities to 5G networks to build, deploy, and scale ultra-low latency applications on edge devices. <https://aws.amazon.com/wavelength/>. Accessed 29 Nov 2022
35. Szepesvári, C.: Algorithms for reinforcement learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **4**( 1), 1–103 (2010)
36. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust Region Policy Optimization. In: *International Conference on Machine Learning*, pp. 1889–1897. <https://arxiv.org/abs/1502.05477> (2015)
37. Sun, W., Yuan, Y.-x.: *Optimization theory and methods*, vol. 1 (2006)
38. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS’99, pp. 1057–1063. MIT Press. <https://dl.acm.org/doi/10.5555/3009657.3009806> (1999)
39. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada*, pp. 8024–8035. <https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf> (2019)
40. Borg cluster traces from Google. <https://github.com/google/cluster-data/>. Accessed 29 Nov 2022
41. Alibaba cluster trace program. <https://github.com/alibaba/clusterdata/>. Accessed 29 Nov 2022
42. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W. Openai gym. *CoRR* abs/1606.01540 (2016). <https://arxiv.org/abs/1606.01540>
43. Amazon EC2 On-Demand Pricing. <https://aws.amazon.com/ec2/pricing/on-demand/>. Accessed 29 Nov 2022

44. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: Fair allocation of multiple resource types. In: 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11). USENIX Association. <https://www.usenix.org/conference/nsdi11/dominant-resource-fairness-fair-allocation-multiple-resource-types> (2011)
45. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., Balde-schwieler, E.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing. SOCC '13. Association for Computing Machinery. <https://doi.org/10.1145/2523616.2523633> (2013)
46. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R., Shenker, S., Stoica, I.: Mesos: A platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation. NSDI'11, pp. 295–308. USENIX Association. <https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center> (2011)
47. Bays, C.: A comparison of next-fit, first-fit, and best-fit. *Commun. ACM* **20**(3), 191–192 (1977). <https://doi.org/10.1145/359436.359453>
48. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. In: Proceedings of the 2014 ACM Conference on SIGCOMM. SIGCOMM '14, pp. 455–466. Association for Computing Machinery. <https://doi.org/10.1145/2619239.2626334> (2014)
49. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. *CoRR abs/1602.01783* (2016). <https://arxiv.org/abs/1602.01783>
50. Wang, Z., de Freitas, N., Lanctot, M.: Dueling network architectures for deep reinforcement learning. *CoRR abs/1511.06581* (2015). <https://arxiv.org/abs/1511.06581>
51. van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. *CoRR abs/1509.06461* (2015). <https://arxiv.org/abs/1509.06461>
52. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.A.: Playing atari with deep reinforcement learning. *CoRR abs/1312.5602* (2013). <https://arxiv.org/abs/1312.5602>
53. RLlib: Industry-Grade Reinforcement Learning. <https://docs.ray.io/en/latest/rllib/index.html>. Accessed 29 Nov 2022

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.