Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

UDL: a cloud task scheduling framework based on multiple deep neural networks

Qirui Li[1], Zhiping Peng[1*], Delong Cui[1], Jianpeng Lin[2] and Hao Zhang[3]

Abstract

Cloud task scheduling and resource allocation (TSRA) constitute a core issue in cloud computing. Batch submission is a common user task deployment mode in cloud computing systems. In this mode, it has been a challenge for cloud systems to balance the quality of user service and the revenue of cloud service provider (CSP). To this end, with multi-objective optimization (MOO) of minimizing task latency and energy consumption, we propose a cloud TSRA framework based on deep learning (DL). The system solves the TSRA problems of multiple task queues and virtual machine (VM) clusters by uniting multiple deep neural networks (DNNs) as task scheduler of cloud system. The DNNs are divided into exploration part and exploitation part. At each scheduling time step, the model saves the best outputs of all scheduling policies from each DNN to the experienced sample memory pool (SMP), and periodically selects random training samples from SMP to train each DNN of exploitation part. We designed a united deep learning (UDL) algorithm based on this framework. Experimental results show that the UDL algorithm can effectively solve the MOO problem of TSRA for cloud tasks, and performs better than benchmark algorithms such as heterogeneous distributed deep learning (HDDL) in terms of task scheduling performance.

Keywords Deep neural network, Memory replay, United, Task scheduling, Sample memory pool

Introduction

The information and communications technology (ICT) has taken a huge leap forward in recent years. Cloud computing is one of the core sources of power. Unlike traditional web server platforms, in addition to availability and convenience, cloud computing can provide on-demand services by abstracting CPU, memory, network, platform, and software applications into a computing resource pool. On a cloud platform, once the management mode and policies for resource scheduling, monitoring, and backup are designed, the CSP no longer needs to perform excessive online management. The user can efficiently obtain both on-demand and pay-as-you-go computing resources through a small amount of interaction with the CSP. With the powerful computing and storage capabilities, cloud computing platforms can provide personalized services for different users. According to the different levels of external services provided, cloud computing architecture has several service models, among which Infrastructure as a Service (IaaS) is the most mature and widely used.

Cloud workloads are increasingly heterogeneous such that a single Cloud job may encompass one to several tasks, and tasks belonging to the same job may behave distinctively during their actual execution [1]. Virtual machines(VM), which are important computing resources in data center, have heterogeneous processor architectures and speeds, hardware features, memory and disk capacities [2]. Due to heterogeneity of tasks and resources, variability of service quality, and huge number of users on cloud computing platform,

*Correspondence:
Zhiping Peng
zhipingpeng@gdupt.edu.cn
[1] School of Computer, Guangdong University of Petrochemical Technology, Maoming 525000, China
[2] School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China
[3] Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai 200433, China

the system has had to process large quantities of tasks and data. With increasing number of users and tasks, optimal task scheduling becomes a strenuous process [3]. In this case, to deploy tasks on a VM alone is prone to overloading the VM server, resulting in slow task response and increased risk of a service level agreement (SLA) violation. For this reason, CSPs and users tend to use a service mode with multiple queues and VM clusters. Multiple queues refer to the organization of tasks of different natures or with requirements into multiple queues for submission. A multiple VM cluster refers to a cluster of VMs belonging to a user that can communicate and coordinate with each other, and whose overall performance is substantially improved by control techniques such as intra-cluster load balancing. In this service mode, efficient task scheduling, reasonable resource allocation, and reduced task makespan and energy consumption of VM clusters directly determine the service quality level and operational profit of the cloud platform [4], and TSRA optimization under multiple queues and clusters is a core problem.

Substantial research has addressed the scheduling optimization problem of cloud computing. Heuristic algorithms have been tried, but traditional heuristic algorithms require specific conditions to obtain the optimal solution. Their generality is not strong in the complex, changeable cloud environment, and they can easily fall into a local optimal solution of a MOO problem. Reinforcement learning (RL), as a model-free method with powerful decision-making capability, obtains the optimal solution through continuous trial and error, but it is prone to slow convergence in the case of a large-scale state space. DNNs have powerful perceptual capabilities to effectively cope with large-scale state spaces.

DL has made breakthroughs in natural language processing, games, robot control, and other fields. Scholars have used such models to solve the TSRA problem in complex cloud environments. There are studies using multiple DNNs for the MOO problem of TSRA. They usually treat each DNN equally and use the same samples for training, so each DNN learns about the same knowledge, and the scheduling is prone to falling into local optimum. The UDL we proposed improves this multiple DNNs schudling method. We innovatively divid multiple DNN networks into two parts: exploration and exploitation, and trains only the exploited DNNs to preserve the randomness of the scheduling. The improved method can achieve better scheduling performance than before, and effectively solve the scheduling problem of multiple queues and clusters on cloud platforms.

The major contributions and results of this paper are as follows:

- This work proposes a UDL-based model to solve the MOO problem for TSRA with multi-task queues and multi-VM clusters.
- The proposed UDL model divides multiple DNNs into two parts: exploration and exploitation. The exploration DNNs have strong randomness to explore better scheduling strategies, while the exploitation DNNs are responsible for learning the explored scheduling strategies, thus improving the learning efficiency of the model and ensuring convergence stability.
- We assign an adjustable weight to each of the two optimization objectives, energy consumption and task latency, so that it can dynamically adjust the bias of the system optimization objectives.
- Multiple sets of experiments with different sizes of task queues and VM clusters are taken to validate the performance of the model. The experimental results show that the proposed model outperforms several benchmark algorithms for the MOO problem of cloud TSRA.

The remainder of the paper is organized as follows. The literature review is described in Section "Literature review". The system model framework and its mathematical model are described in Section "System model". The united deep network and its training algorithm are described in Section "United deep network and its training". Section "Simulation experiments and results analysis" provides the simulation experiment results and their analysis, and finally, the paper is concluded in Section "Conclusion and future work".

Literature review

Since cloud computing platforms have powerful computing and storage capabilities, many users begin to replace cloud service with local service and submit tasks to cloud for processing. Because the task scheduling strategy determines both the service quality level and profits of the cloud platform, the optimization of cloud TSRA has always been a research focus.

Research has focused on the TSRA problem of cloud computing [5, 6]. Verma et al. [7] proposed a mixed particle swarm optimization (PSO) algorithm with non-dominance ranking to handle workflow scheduling problems on IaaS clouds. Zuo et al. [8] demonstrated a resource-cost model reflected the relationship between resource cost and user budget. The proposed scheduling method was based on an improved ant colony optimization (ACO) algorithm to achieve MOO of system performance and cost. Alkayal et al. [9] proposed multi-objective PSO algorithm based on a new ranking strategy, with the goal to maximize system throughput and minimize task wait

time during task scheduling to VMs. Duan et al. [10] proposed a VM scheduling method, PreAntPolicy, consisting of a scheduler based on an improved ACO and a predictive model based on fractal mathematics. The predictive model attained more reasonable scheduling by predicting the load trend. A task scheduling algorithm combining the properties of a genetic algorithm (GA) and bacterial foraging (BF) algorithm was proposed by Srichandan et al. [11] to achieve efficient TSRA under the constraints of a guaranteed SLA. However, traditional heuristic algorithms must obtain the optimal solution under certain conditions. Their versatility is not strong in a complex, changeable cloud environment, they easily fall into local optima when solving a MOO problem, and the global optimal solution is not obtained.

Some researchers have used RL methods to solve the above problem. RL is a model-free learning method with powerful decision-making capability and can effectively solve multi-constrained MOO problem. Peng et al. [12, 13] utilized RL to find the optimal scheduling strategy and solve TSRA problem in cloud environments. Cui et al. [14] proposed a task scheduling scheme based on RL, which also applies multi-agent and parallel technology to balance exploration and exploitation in the learning process, and achieves the maximum reduction of task makespan under the constraints of task deadlines and VM resources. Thein et al. [15] achieved high energy efficiency and prevented SLA violations in data centers by an RL-based approach. Aiming at the problem for Software as a Service (SaaS) CSPs of automatically scaling applications to meet customer needs in dynamically changing cloud environment, Wei et al. [16] proposed a RL-based adaptive lease scheme generation algorithm, with adjusting IaaS facility adaptively. Liang et al. [17] modeled the resource allocation problem in Internet of Vehicles (IoV) as a semi-Markov decision process and used RL to solve it. The RL algorithm can get the optimal decision through constant trial and error, but it converges slowly in a large-scale state space. DNNs have strong feature perception capabilities to effectively deal with large-scale state spaces and make up for the shortcomings of RL.

DL has powerful feature extraction ability, is a popular research topic in artificial intelligence, and is widely applied in image processing and pattern recognition. Some scholars have applied it to resource and task scheduling in cloud platforms. Guo et al. [18] proposed DeepRM_Plus, a cloud resource management scheme based on convolutional neural network (CNN) , which uses imitation learning to reduce the learning time, improve convergence speed, and reduce the average cycle time and weighted turnaround time. Chudasama et al. [19] used DL and queuing theory in an efficient auto-scaling technology with a predictive function to solve the problem that a static threshold method may fail under high dynamic and unpredictable workloads. The elasticity of cloud system resources is enhanced, and the accuracy of SLA violations can be predicted more accurately. Lakhan ea al. [20] devised a deep neural networks energy cost-efficient partitioning and task scheduling algorithm framework to deal with the partitioning and scheduling of IoT applications in terms of resource management for mobile workflow applications in enterprise systems. Rangra et al. [21] proposed a cloud TSRA algorithm based on multi-task CNNs, achieving a balance between makespan and cost. The algorithm was used in tweet task sets and gene workflow task sets, with good results. Lin et al. [22] proposed a multi-intelligent two-stage TSRA framework for collaborative scheduling between cloud task and cloud resource. The task scheduling stage uses a HDDL model to schedule user tasks to data centers. The resource scheduling stage uses a deep Q-network model to deploy VMs to physical servers. The framework globally optimizes scheduling through local optimization in each stage.

RL has powerful decision-making ability, while DL has powerful feature-acquisition ability. Scholars have combined them to form deep RL (DRL), which has made breakthroughs [23] in fields such as natural language processing [24], games [25], robot control [26], and cloud resource scheduling in complex environments, providing a new solution to TSRA problem of cloud computing. Peng et al. [27] proposed a framework for TSRA based on DRL, which synergistically considers the balance of interests between users and CSPs, and can optimize different objectives by adjusting the corresponding optimization weights. Lin et al. [28] made full use of the perception of CNN and the decision-making ability of RL in a TSRA model, abstracting the cloud resources and cloud task in the form of "images" as the input of the CNN, and outputting a scheduling strategy. For large-scale TSRA problems, Bitsakos et al. [29] proposed an elastic resource supply system based on DRL, which could automatically and dynamically allocate computer resources according to users' fluctuating workload demands, and follow the optimal resource management policy. Zhang et al. [30] applied the DQN algorithm to the problem of wireless LAN task offloading to minimize monetary and energy costs of mobile users. Huang et al. [31] combined RL training methods and distributed DL models to solve the problem of task offloading in mobile edge computing, reducing energy consumption and ensuring service quality.

We study effective task scheduling to minimize the overall task completion time and energy consumption of a data center when submitting batch tasks to several computing clusters for execution in a cloud task system.

This is essentially an offline task scheduling approach. We propose a united deep network framework for cloud task scheduling. By combining multiple deep networks (DNNs) as task schedulers, and training them with reference to the trial and memory playback mechanism in DRL, the framework solves the problem of batch task submission in cloud systems.

System model

Model framework

When users obtain personalized cloud computing services, they submit tasks through the network, and obtain virtual resources to meet their needs.

The system model is shown in Fig. 1. The tasks to be executed on cloud are submitted to CSP in batches. The CSP inputs the tasks to a task scheduler consisting of multiple trained DNNs. The task scheduler generates a scheduling strategy according to the status of a submitted task, and uses this to schedule the task to a computing cluster for processing. Before the task is submitted, the CSP must train each DNN network based on the training task set with the goal to minimize energy consumption and task completion time.

The system model has the following key parts: (1) The strategy generation component consists mainly of multiple deep networks, and generates TSRA strategies to minimize task latency and system energy consumption according to user tasks. The structure and quantity of networks can be dynamically adjusted according to need; (2) Energy consumption calculation components determine communication and computational energy consumption; (3) The SLA considers the task completion time, including latency of task communication and computation; (4) The task scheduler is the core component of the system, responsible for scheduling tasks in multiple queues to different computing clusters according to the scheduling strategy. It must guarantee the SLA and minimum system energy consumption.

The large number of cloud users of various types results in a diversity of user loads, whose multiple tasks have different dependencies and priorities, and data transmission between them. Therefore, the task scheduling process must ensure the execution order and dependencies between tasks. In the user load layer, our model decouples dependent user loads into child tasks and distributes them to multiple waiting queues. The model ensures that parent tasks in the waiting queue have priority in data

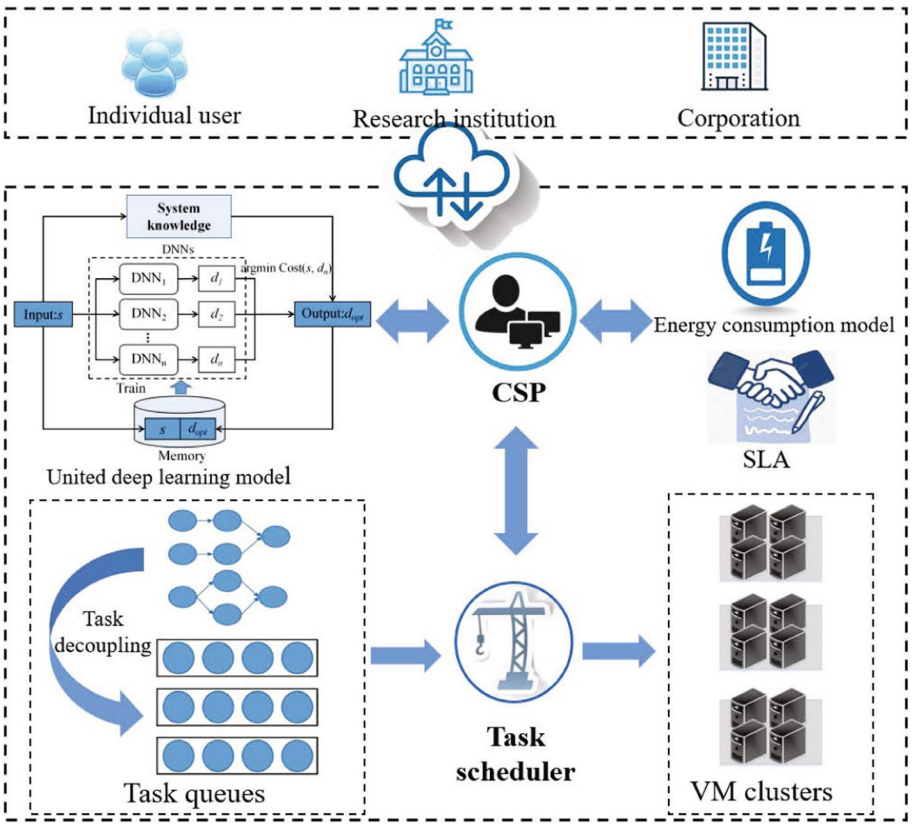

Fig. 1 System model

transmission and execution, and each task in the queue is atomic and can run independently. Each waiting queue has the same storage space, and the number of queues is dynamically adjusted according to needs.

A large number of infrastructure devices form a large-scale data center, which clusters adjacent servers into computing clusters according to geographic locations. The communication between multiple VM clusters is carried out through high-speech optical fiber, so data transmission latency and energy consumption between them can be ignored. However, the bandwidth and distance of users connected to different VM clusters are obviously different. Hence both are important considerations for optimization problems. Moreover, because of differences in hardware, cluster computing ability and computing power are also key factors that affect system scheduling efficiency.

Mathematical modeling

Cloud system task scheduling involves the scheduling of atomic tasks in multiple queues to multiple clusters. Assume that the number of computing clusters is K, which is expressed as $\{Clu_1, Clu_2, \ldots, Clu_k\}$. The number of task queues waiting to be scheduled is N, which is expressed as $\{Q_1, Q_2, \ldots, Q_n\}$. The number of tasks contained in each queue is M, which is expressed as $\{T_1, T_2, \ldots, T_m\}$. Therefore, the total number of tasks is $M * N$. Task T_{nm} denotes task m in queue n. The attributes of task T_{nm} are expressed as a binary tuple, $(r_{nm}^{cpu}, r_{nm}^{data})$, where r_{nm}^{cpu} denotes the number of CPU cycles required by T_{nm}, and r_{nm}^{data} denotes the amount of data required to be transferred by T_{nm}. r_{nm}^{data} is a random variable that obeys a uniform distribution, $r_{nm}^{data} \sim (r_{min}^{data}, r_{max}^{data})$, where the maximum and minimum amounts of task data are respectively expressed by r_{min}^{data} and r_{max}^{data}. In addition, we assume that the CPU cycles required for each task are linearly related to the amount of data in the task [32],

$$r_{nm}^{cpu} = \mu \times r_{nm}^{data}, \tag{1}$$

where μ is the computation-to-data ratio (CDR), whose value depends on the type of task.

The attributes of cluster Clu_k are represented by the triplet $(CP_k, P_k^{comm}, P_k^{comp})$, where CP_k is the computing power of the cluster, i.e., the number of cycles of the CPU; P_k^{comm} is the communication power consumption of the cluster; and P_k^{comp} is the computing power consumption of the cluster. The allocation of task T_m in queue Q_n to cluster Clu_k for processing is expressed by action $a_{nmk} \in \{0, 1\}, 1 \le n \le N, 1 \le m \le M, 1 \le k \le K$, specified as:

$$a_{nmk} = \begin{cases} 1, \text{ if } \mathrm{T_{nm}} \text{ is asigned to } \mathrm{Clu_k} \\ 0, \text{ otherwise} \end{cases} \tag{2}$$

The communication bandwidth between the queue and cluster is expressed as $\{BW_{12}, \ldots, BW_{nk}\}$, and BW_{nk} is the bandwidth allocated between queue Q_n and cluster Clu_k.

We consider two key factors of the scheduling process: task latency and energy consumption. Below, we formally define the communication and calculation models involved in task scheduling.

(1) Communication model

According to the definition of a_{nmk}, in scheduling time slot t, the number of tasks allocated to cluster Clu_k from queue Q_n is:

$$A_{nk} = \Sigma_{1 \le m \le M} a_{nmk}. \tag{3}$$

The communication model includes the transmission time and energy consumption required to transfer the task data. When multiple tasks in the same queue are scheduled to the same cluster at the same time, we use the principle of equal distribution to allocate the bandwidth to these tasks. Therefore, if task T_{nm} is allocated to cluster Clu_k, the bandwidth it could occupy is:

$$R_{nm}^{bw} = \frac{BW_{nk}}{A_{nk}}. \tag{4}$$

The communication latency T_{nm}^{comm} is the time consumed to upload the task data to the server, specified as:

$$TD_{nm}^{comm} = \frac{r_{mn}^{data}}{R_{nm}^{bw}}. \tag{5}$$

The communication energy consumption is the energy consumed during task transmission, specified as:

$$EC_{nm}^{comm} = P_k^{comm} \times TD_{nm}^{comm}. \tag{6}$$

Therefore, the communication energy consumption of all tasks in queue Q_n is:

$$EC_n^{comn} = \Sigma_{m=1}^{M} EC_{nm}^{comm}. \tag{7}$$

(2) Computational model

The computational model includes the computational latency and energy consumption of the tasks. We also use the principle of equal distribution, by which the computational power of a cluster is divided equally among all tasks scheduled to it. Similarly, the number of tasks scheduled to cluster Clu_k is:

$$B_k = \Sigma_{n=1}^{N} \Sigma_{m=1}^{M} a_{nmk}. \tag{8}$$

Therefore, each task receives computational power as:

$$R_{nm}^{cpu} = \frac{CP_k}{B_k}. \tag{9}$$

Computational latency is the time consumed by a task to complete the computation, specified as:

$$TD_{nm}^{comp} = \frac{r_{nm}^{cpu}}{R_{nm}^{cpu}}. \tag{10}$$

Computational energy consumption is the energy consumed by a task during computation, specified as:

$$EC_{nm}^{comp} = P_k^{comp} \times TD_{nm}^{comp}. \tag{11}$$

The computational energy consumption of all tasks in queue Q_n is

$$EC_n^{comp} = \Sigma_{m=1}^{M} EC_{nm}^{comp}. \tag{12}$$

(3) Optimization objectives

At some scheduling time slot t, the scheduled tasks are executed in parallel in the cluster, so that the total time latency required for the batch of tasks is:

$$TD = \max_{1 \le n \le N, 1 \le m \le M} (TD_{nm}^{comm} + TD_{nm}^{comp}). \tag{13}$$

However, the total energy consumption in executing the batch of tasks is the sum of the energy consumption of each task, i.e.,

$$EC = \Sigma_{n=1}^{N} (EC_n^{comm} + EC_n^{comp}). \tag{14}$$

The optimization objectives of the research problem in this section is to minimize the task latency and energy consumption, which is a MOO problem. We assign a weight factor to each objective to characterize its bias in the total optimization objective. If the scheduling strategy adopted by task set s is d, then the payoff function of the system is defined as follows:

$$Cost(s, d) = \lambda \times TD + (1 - \lambda) \times EC, \tag{15}$$

where $\lambda \in [0, 1]$ is the weight of task latency in the total optimization objective, and $(1 - \lambda)$ is the optimization weight of energy consumption. The larger the value of λ, the greater the weight of time latency in the total optimization objective and the smaller the energy consumption. If $\lambda = 0$, then the optimization objective only considers the energy consumption factor, and if $\lambda = 1$, then the optimization objective only considers the time latency factor.

The objective of the system is to obtain the optimal scheduling strategy, i.e., to minimize the task latency and energy consumption. Let D denote all scheduling strategies for task set s. Then the system optimization objective can be expressed as:

$$\begin{aligned} &\min_{d \in D} Cost(s, d) \\ &s.t. \\ &C1: \ \Sigma_{m=1}^{M} R_{nm}^{bw} = BW_{nk} \\ &C2: \ \Sigma_{m=1}^{M} R_{nm}^{cpu} = CP_k \\ &C3: \ 1 \le n \le N, 1 \le k \le k. \end{aligned} \tag{16}$$

The scheduling of cloud tasks is an NP-complete problem that has never been fully solved [22]. In the case of the multi-queue multi-cluster (MQMC) scheduling model studied in this section, there are as many as K^{M*N} possibilities for scheduling, which is an exponential level of problem space. When the problem size scales up, the traditional exact and approximate methods will require huge computational effort and time. In recent years, DL has broken the barriers of traditional methods in many fields and made remarkable breakthroughs, with its powerful learning ability. Therefore, scholars are trying to solve combinatorial optimization problems by DL [33]. We next study the use of DL to solve the above cloud task scheduling problem.

United deep network and its training

United deep network

A DNN is a neural network (NN) composed of many hidden layers. A UDL model unites multiple DNNs as the fitting function. In each DNN of UDL model, the number of network layers is the same, and the number of hidden layer nodes is different, but the overall scale of network parameters is comparable, as shown in Fig. 2. Similar to the experience replay mechanism of DRL, the model stores the samples generated by itself in SMP for use as a public training sample set. When the number of samples reaches the predetermined threshold, small batches of samples in SMP are randomly choice periodically for each DNN training. In this way, it improves both the agent's ability to explore the optimal strategy.

During the model's training, the input of X DNNs is the state s_t, which is expressed as $\{r_{11}^{cpu}, r_{11}^{data}, r_{12}^{cpu}, r_{12}^{data}, \ldots, r_{nm}^{cpu}, r_{nm}^{data}\}$, consisting of multiple task attributes in multiple queues. Since each DNN works independently, they would output different action decisions, which are expressed as $(d_t^1, d_t^2, \ldots, d_t^X)$. In scheduling time slot t, s_t is used as input, and the output action decision $d_t^x, 1 \le x \le X$, of each DNN can be expressed as:

$$f_{\theta_t^x} : s_t \to d_t^x, \tag{17}$$

where $f_{\theta_t^x}$ is a function denoting the x-th DNN network parameter.

In Eq. (17), d_t^x is denoted as $d_t^x = \{a_{111}, a_{121}, \ldots, a_{nmk}\}$, where a_{nmk} is defined in Eq. (2). If $a_{nmk} = 1$, then T_m in Q_n is scheduled into Clu_k. The Eq. (15) is then used to calculate the cost value of each action decision. The action

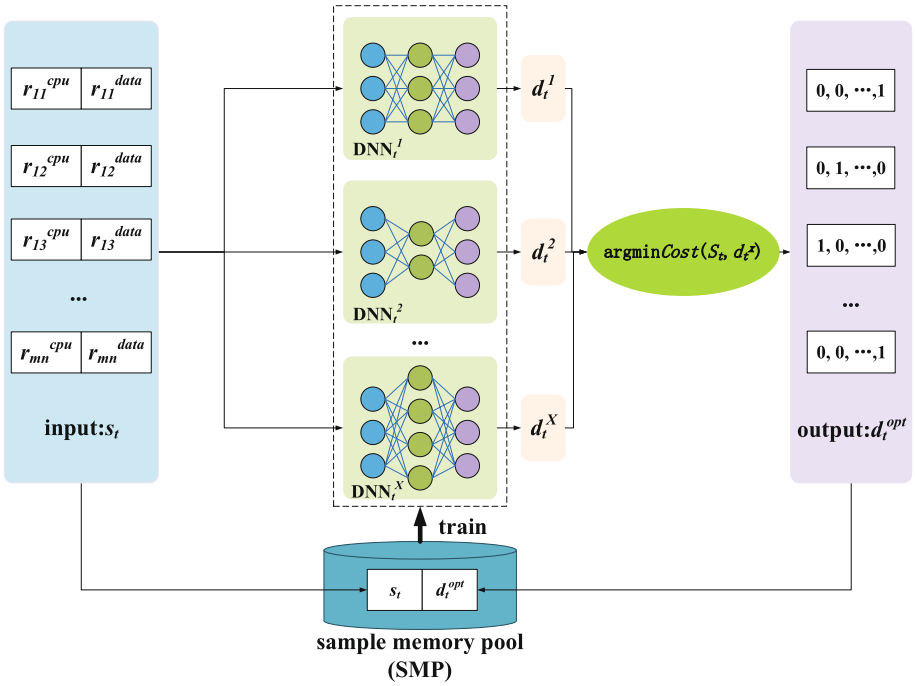

Fig. 2 United DNN model

decision that obtains the smallest *Cost* is selected as best for the group of tasks:

$$d_t^{opt} = \arg\min_{x \in X} Cost(s_t, d_t^x), \tag{18}$$

where s_t is the current task set state and d_t^{opt} is the best decision action.

In a scheduling time slot, (s_t, d_t^{opt}) is stored as sample in SMP. The model would randomly select *miniBatch* samples for training as the number of samples reaches a predetermined threshold. The training process uses a gradient descent algorithm to minimize the cross-entropy loss to optimize the parameter values θ_t^x of each DNN.

$$L(\theta_t^x) = -d_t^T \log f_{\theta^x}(s_t) - (1 - d_t)^T \log(1 - f_{\theta^x}(s_t)). \tag{19}$$

The above is a common training method, that is, each DNN uses the same training samples for learning at the same time. It is necessary for each DNN to train to learn the optimal scheduling strategy. In this way, all DNNs can learn the scheduling experience, and the convergence speed of the UDL model is faster. The HDDL algorithm proposed in [22] uses this training method. However, this method is easy to get stuck at locally optimal value. This is because when all tasks in the training set are executed for the first time, the current optimal strategy generated by the UDL model is put into SMP. According to the way that all DNNs learn at the same time, after the first episode of training, all DNNs have learned the current optimal strategy. In the next episode of training, facing the same training set, all DNNs will output the corresponding scheduling strategies according to the learned experience. Since these DNNs learn the same experience, their output strategies are basically the same when faced with the same task. Therefore, through such training, the DNNs are difficult to be improved.

The fundamental reason for this situation is that these DNNs have all learned the same strategies, that is, all DNNs are exploited, and there is no possibility of further exploring other strategies. Therefore, we made corresponding improvement to the training method. We divided DNNs into two parts: one is responsible for exploration, and the other is responsible for exploitation, as shown in Fig. 3.

In Fig. 3, the multiple DNNs have been divided into two parts exploitation and exploration. Because of the diversity of tasks and the heterogeneity of resources, the number and size of the two partial DNNs are not strictly defined and need to be determined according to the training effect of different task sets. However, no matter how they are divided, only DNNs in the part of exploitation train, and DNNs in the part of exploration do not train, so that the model retains the possibility of random scheduling strategies. Through the combination of exploration and exploitation, the UDL model can go out of the local optimum and move towards the global optimum.

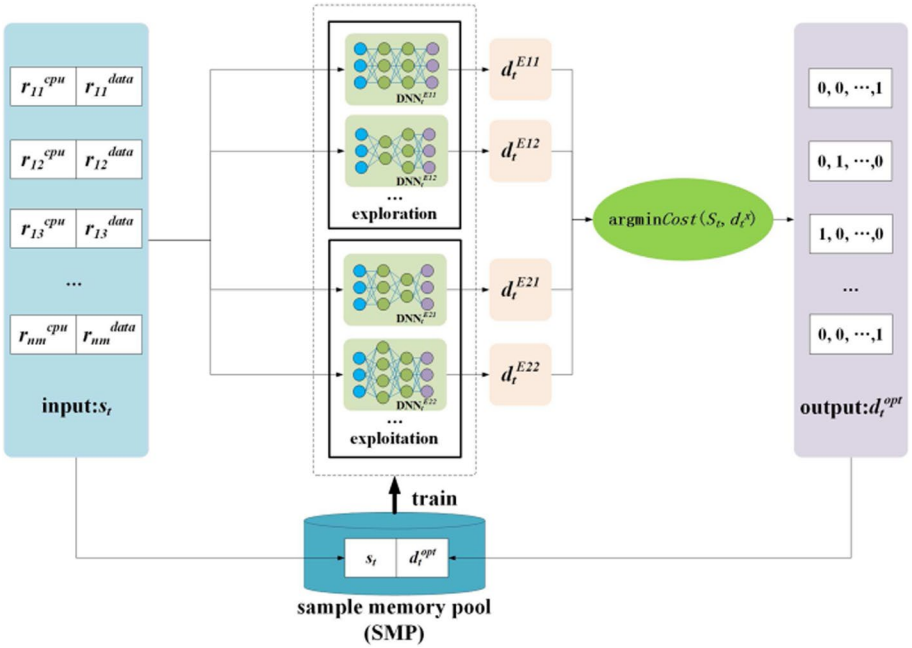

Fig. 3 United DNN model with division of exploitation and exploration

Training of united deep network

After building the DNN network, SMP is generated according to the given task training set by the experience playback mechanism. Then a batch of samples is randomly selected from SMP to train the DNNs in the part of exploitation. We call the learning model based on multiple DNNs a UDL model, and a TSRA algorithm based on this is called a UDL algorithm. The pseudo-code of the UDL model training process is shown as Algorithm 1.

Require: all task requirements in task ready queues.
Ensure: task scheduling decisions d^x.
Initialize each DNN with different random wights θ^x.
Initialize SMP with capacity M.
for $t = 1, 2, \ldots, T$ **do**
  Input the same s_t to each DNN.
  Generate d_t^x from the DNNs with Eq. (17).
  Compute the Cost of d_t^x with Eq. (15)
  Select the optimal decision d_t^{opt} with Eq. (18).
  Store (s_t, d_t^{opt}) into SMP.
  if number of samples exceeds the threshold and t is the training interval **then**
    Randomly choice a batch training samples S from SMP.
    for each sample in S **do**
      Define loss function as Eq. (19)
      Update parameters of DNNs in exploitation part using stochastic gradient descent.
    end for
  end if
end for

Algorithm 1 UDL model training algorithmWhen these DNNs of exploitation part are trained, they are packaged into an executable scheduler or package to be deployed in a realistic scheduling environment. When a new batch of tasks arrives, the new tasks are put in the trained model and the model will output the corresponding scheduling policy. The specific scheduling algorithm is shown in Algorithm 2.

Require: a new batch of tasks.
Ensure: task scheduling decisions d^{opt}.
Generate the s_t with the new batch of tasks.
Input the same s_t to each DNN of exploitation part.
Generate d_t^x from the DNNs with Eq. (17).
Select and output the optimal decision d_t^{opt} with Eq. (18).

Algorithm 2 UDL model scheduling algorithmSimulation experiments and results analysis

Experimental design and parameters

We designed a two-part simulation experiment to verify the effectiveness and performance of the proposed model. The first part verifies the convergence of the UDL model with different queue numbers and clusters. The second part compares the task scheduling performance of the proposed algorithm to that of benchmark algorithms, including random, round robin (RR), multi-objective particle swarm (PSO), deep Q network(DQN) and heterogeneous distributed deep learning (HDDL) [22]. Random algorithm schedules tasks to the clusters randomly. RR algorithm schedules tasks to the ordered clusters in turn. PSO algorithm is a random search

algorithm based on group collaboration, which is one type of swarm intelligence (SI). DQN is classical deep reinforcement learning algorithm. HDDL algorithm also uses multiple DNNs as scheduler, but doesn't divide them into two parts of exploration and exploitation, and uses all of them as exploitation.

In the simulation experiment, the number of tasks in the queue was set to 4, the minimum value of task data r_{min}^{data} was 100, and the maximum value of task data r_{max}^{data} was 500. There are four types of task settings, and the ratio between the required CPU cycles and the amount of data μ(CDR) is shown in Table 1 [32]. During the experiment, the generated task types were randomly obtained from Table 1 with the same probability. The learning rate was set to 0.01, the training interval to 10, the sample batch to 128, and the SMP size to 1024.

In the simulation experiment, a total of 12 clusters were designed for selection, with configurations as shown in Table 2.

Eight heterogeneous DNN networks were designed as decision generators, each with one input layer, three hidden layers, and one output layer. The number of neurons in each layer is shown in Table 3. We divided these eight DNNs into exploration part and exploitation part. The exploration part includes odd-numbered DNNs, and the exploitation part includes even-numbered DNNs.

The simulation experiment platform was developed based on the Python language and a TensorFlow framework, running on a Windows 10 OS, with an Intel core i7-8550U dual-core CPU at 1.80 GHz and 16 GB memory.

Network model verification experiment

We experimentally verified the convergence of the model with different numbers of queues and clusters.

(1) Convergence under different numbers of queues and a fixed number of clusters

The convergence of the UDL algorithm was examined when the number of clusters (CN) was 5, λ was 0.9, and the number of queues (QN) was set to 4, 6, 8, and 10. The experimental results are shown in Fig. 4.

From Fig. 4, it is obvious that the proposed UDL algorithm basically reached a state of convergence in all cases. In the first round of training (the first 1000 iterations), the algorithm converges the fastest and then gradually slows down. After about 60 to 80 rounds, the convergence state is basically reached. Moreover, the experimental results shows that the *Cost* of the UDL algorithm increased with the number of queues for a given number of clusters. This is mainly because, as the number of queues increases, the number of tasks to be executed increases, the competition for resources becomes more intense, the computational and bandwidth resources available to each task decrease accordingly, and the computation and communication times increase, with a corresponding increase in overall latency and energy consumption.

Tasks in the dataset were randomly generated by the task generator. The task computation and data volume satisfied Eq. (1). A total of 1000 sets of training tasks and 100 sets of testing tasks were generated.

(2) Convergence under different numbers of clusters and the same number of queues

The convergence of the UDL algorithm was examined when the number of queues (CN) was 5, λ was 0.9, and the number of clusters (QN) was set to 3, 6, 9, and 12. The experimental results are shown in Fig. 5.

Table 1 Task Types and its CDR

Workload	CDR
gzip ASCII compress	330
x264 VBR encode	1300
x264 CBR encode	1900
html2text wikipedia.org	2100

Table 2 Main parameters of clusters

No.	Computation Ability(cycles/s)	Bandwidth (MB/s)	Computation Power(w)	Communication Power(w)
1	1.5×10^{15}	250/8	1.0×10^{5}	0.20
2	2.5×10^{15}	250/8	2.5×10^{5}	0.20
3	3.5×10^{15}	500/8	4.0×10^{5}	0.40
4	5.0×10^{15}	500/8	6.0×10^{5}	0.40
5	6.0×10^{15}	750/8	7.0×10^{5}	0.50
6	7.0×10^{15}	750/8	8.0×10^{5}	0.50
7	6.5×10^{15}	800/8	7.5×10^{5}	0.60
8	7.2×10^{15}	800/8	8.6×10^{5}	0.60
9	6.8×10^{15}	850/8	7.8×10^{5}	0.65
10	7.5×10^{15}	850/8	8.8×10^{5}	0.65
11	8.0×10^{15}	900/8	9.0×10^{5}	0.80
12	10.0×10^{15}	900/8	10.5×10^{5}	0.80

Table 3 Main parameters of DNNs

No.	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Output Layer
1	$N \times M \times 2$	150	30	10	$N \times M \times K$
2	$N \times M \times 2$	160	40	10	$N \times M \times K$
3	$N \times M \times 2$	150	50	10	$N \times M \times K$
4	$N \times M \times 2$	170	40	10	$N \times M \times K$
5	$N \times M \times 2$	180	30	10	$N \times M \times K$
6	$N \times M \times 2$	190	50	10	$N \times M \times K$
7	$N \times M \times 2$	100	40	10	$N \times M \times K$
8	$N \times M \times 2$	200	40	10	$N \times M \times K$

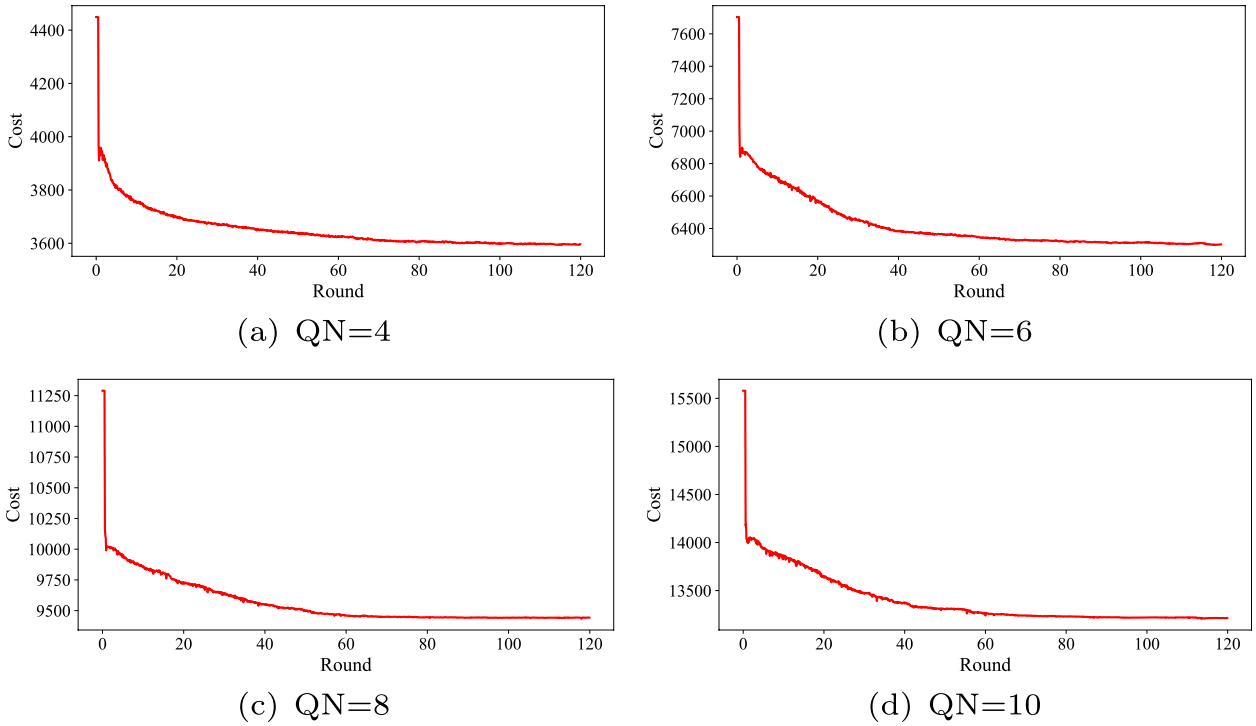

Fig. 4 Convergence of UDL algorithm under different numbers of queues and a fixed number of clusters

From Fig. 5, it is obvious that the proposed UDL algorithm reaches a state of convergence in all cases, too. The experimental results show that with the number of queues fixed, *Cost* as obtained by the UDL algorithm decreases as the number of clusters increases. This is because, with a fixed number of tasks, as the number of clusters increases, the computational and bandwidth resources available to each task increase accordingly, and the computation and communication time decrease, with a corresponding decrease in overall latency and energy consumption.

Simulation experiment for algorithm comparison

We verified the optimization performance of the UDL algorithm for different numbers of queues and clusters, using the random, RR, PSO, DQN and HDDL algorithms as benchmarks.

(1) Performance comparison with different numbers of queues and a fixed number of clusters

We compared each algorithm for different numbers of queues with a fixed number of clusters. In the experiments, the number of clusters was fixed at 5, λ was set to 0.9, and the number of queues increased from 3 to 12. The experimental results are shown in Fig. 6.

From Fig. 6, it can be seen that as the number of task queues increases, the system load increases, and the return values of all algorithm models show an upward trend. The growth rates of the return values of the RR and random algorithms are relatively fast, while those of PSO, and UDL are relatively slow. When the number of task queues is relatively small, the costs of PSO, DQN, HDDL and UDL are closer. However, when the number of task queues is 5 or more, the optimization effect of the UDL algorithm is better than that of the heuristic algorithm PSO, the reinforcement learning algorithm DQN and the similar algorithm HDDL. In this experiment, the number of computing clusters is fixed. When the number of task queues increases, the competition between queues for limited computational resources becomes more intense. In this case, the UDL algorithm shows better task scheduling performance than the PSO, DQN and HDDL algorithm.

(2) Performance comparison with different numbers of clusters and a fixed number of queues

We compared each algorithm for different numbers of clusters and a fixed number of queues. The number of task queues was fixed at 10, λ was set to 0.9, and the number of clusters increased from 3 to 12. The experimental results are shown in Fig. 7.

The experimental results in Fig. 7 show that as the number of clusters increases, and the costs of all algorithms decrease. That is because the available resources of the system increase as the number of clusters increases. Similar to the previous experiment, with a fixed number

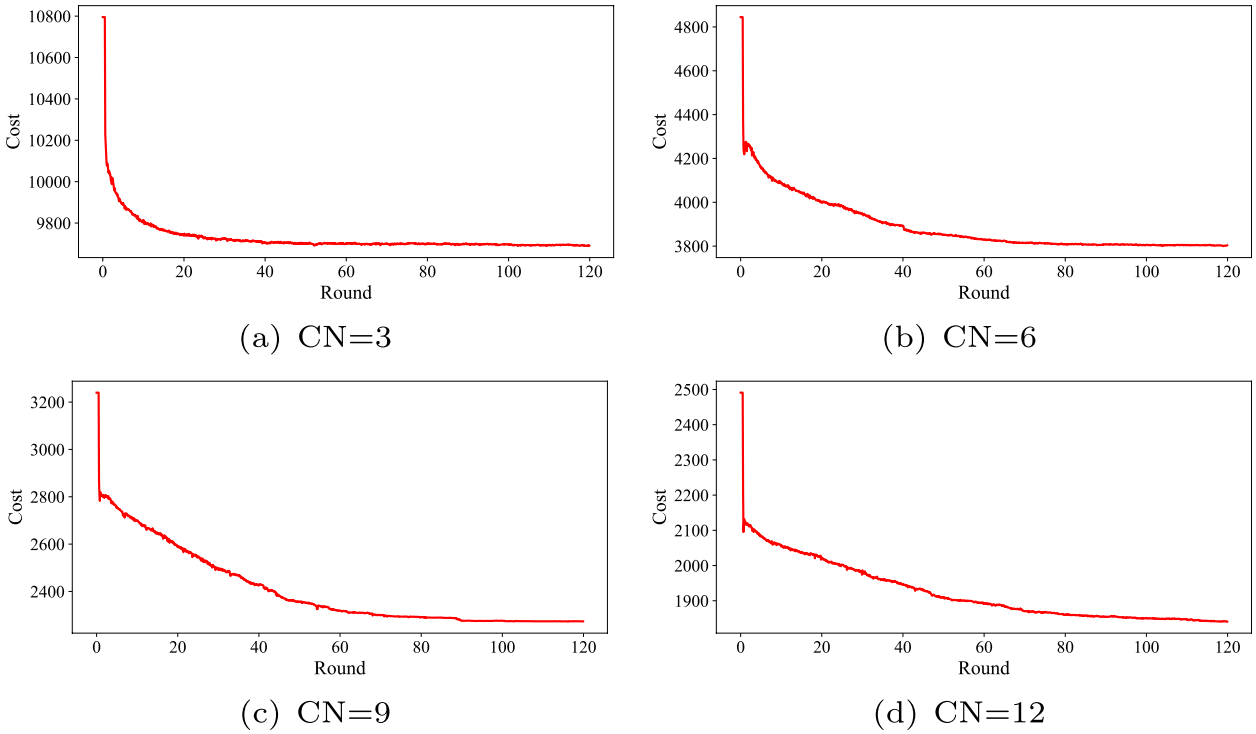

Fig. 5 Convergence of UDL algorithm under different numbers of clusters and a fixed number of queues

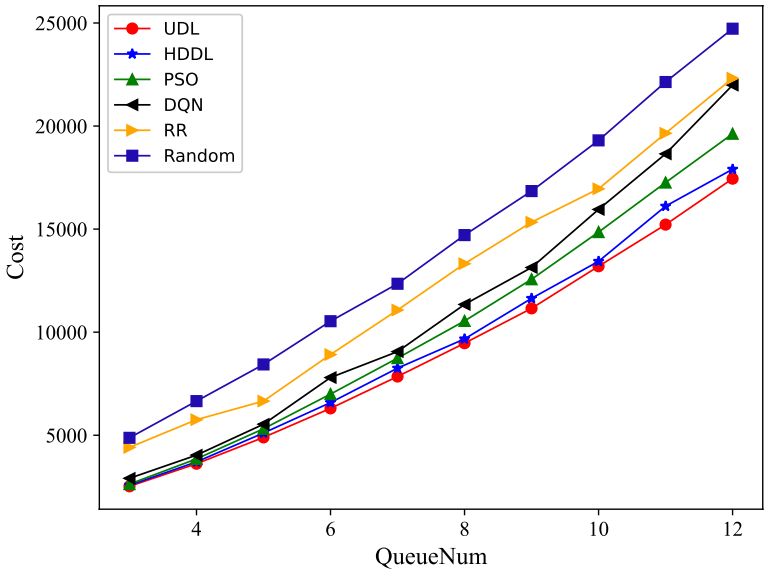

Fig. 6 Algorithm performance comparison under different numbers of queues and a fixed number of clusters

of task queues, when the number of clusters is small, the tasks compete more fiercely for computational resources, and the performance of the UDL and HDDL algorithm is better than that of algorithms such as PSO and DQN. However, when the number of clusters exceeds 8, the computational resources are relatively sufficient, and UDL, HDDL, DQN and PSO algorithms show basically comparable performance. But in most cases, the optimization performance of UDL algorithm is better than the benchmark algorithms.

It can be observed from Figs. 6 and 7 that the DQN model can achieve similar optimization results to UDL and HDDL for small queue and cluster numbers, but as the number of queues and clusters increases, the exploration space size of the scheduling problem grows exponentially, so it becomes increasingly difficult for DQN to explore to the optimal or suboptimal scheduling policy, and the optimization effect decreases significantly.

(3) Scheduling time comparison with PSO

Because RR and random algorithm do not involve complex computations, their decision times are very fast. The UDL, HDDL, DQN are machine learning algorithms, and their decision times are about the same. We compared the time required for the UDL and PSO to make task scheduling decisions in the same environment. We ran the test set 100 times and averaged the results, with results as shown in Table 4. The operating environment in the table is characterized by numbers of queues and clusters, denoted by *QnCm*, which means that there are n queues and m clusters.

From Table 4, it can be seen that the decision time of the UDL algorithm is much less than that of the PSO algorithm in the same environment. As the queue and cluster sizes increase, the decision times of both algorithms increase accordingly, but much more so for PSO than for UDL.

Weight factor change experiment

In Eq. (15), λ is the weight of time latency in the total optimization objective, and $(1-\lambda)$ is the weight of energy consumption. We examined how the time latency, energy consumption, and total optimization objective changed as λ varied from 0.1 to 0.9. The number of queues was set to 10, and the number of clusters to 6. The experimental results are shown in Fig. 8.

Figure 8(a) shows the variation of the total task latency as the weight factor λ varies between $0.1 \sim 0.9$. It can be seen that the total task latency decreases as λ increases. This is because our goal is to minimize task latency, and the increase of λ means that the time latency weighs more in the total optimization objective; hence the algorithm is biased toward the optimization of time latency. Figure 8(b) shows how the energy consumption varies with λ. Since λ is the time latency weight and $1-\lambda$ is the energy consumption weight, to increase λ will reduce the energy consumption weight; hence the algorithm will pay more attention to optimizing time latency, and energy consumption will increase accordingly. Figure 8(c) shows how the total optimization objective follows the change in λ. We can find that our proposed UDL algorithm optimizes better than the benchmark algorithms, no matter the case. That is, UDL can improve system performance regardless of whether the optimization objective is biased toward time latency

Table 4 Decision time comparison between UDL and PSO(Unit:s)

Number	Running Environment	UDL	PSO
1	*Q3C3*	0.3054	8.1511
2	*Q5C5*	0.3703	13.3691
3	*Q8C8*	0.4828	21.8820
4	*Q10C10*	0.5724	28.1806
5	*Q12C12*	0.6770	34.9222

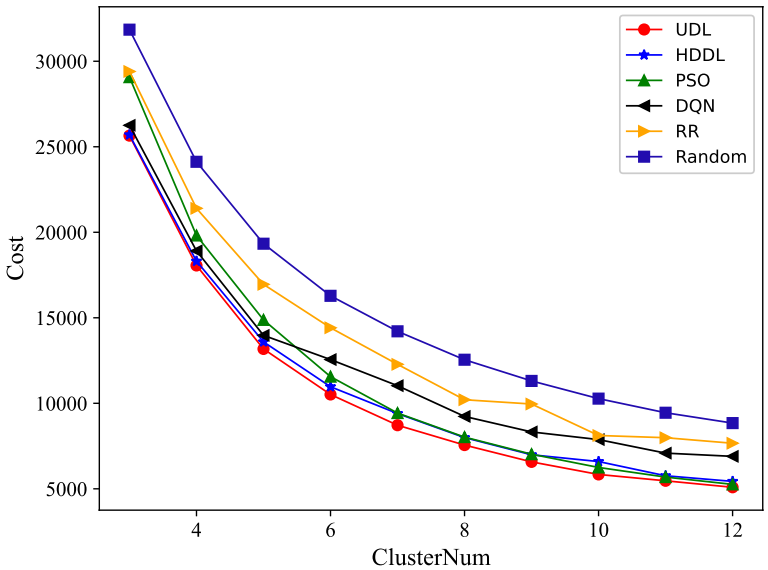

Fig. 7 Algorithm performance comparison with different numbers of clusters and a fixed number of queues

or energy consumption. When λ takes values from 0.1 to 0.9, the cost value of UDL are reduced by 1.08%, 1.24%, 1.33%, 2.10%, 1.03%, 1.06%, 2.06%, 1.15% and 2.35%, respectively, compared to the HDDL. Although the reduction is not very large, considering the relatively large value of the cost and the large amount of tasks in data center, the overall cost savings are still significant.

The weight factor in the valuation function is dynamically adjustable according to the actual demand. And with different weight settings, the proposed UDL can all find a better scheduling policy compared to other benchmarks. This also indicates that UDL has better explorability.

Conclusion and future work

We proposed a cloud TSRA framework model based on UDL to solve the MOO problem of MQMC scheduling in cloud computing. The framework effectively improves algorithm performance by joining multiple DNNs model and employing a DRL experience replay mechanism to train the scheduling model. In order to further improve the performance of the framework, we improved the multiple DNNs model and divided the DNNs into two parts: exploration and exploitation. The part of exploration is responsible for exploring unknown scheduling strategies and the part exploitation is responsible for exploiting the known scheduling strategies. This improved method not only speeds up the convergence of the algorithm, but also avoids falling into a local optimal solution. Experimental results show that the UDL algorithm is better adapted than the random, RR, PSO and HDDL algorithms to MOO problems, especially in the case of intense resource competition. The proposed UDL algorithm is essentially an offline task scheduling algorithm, which is suitable for batch submission task scheduling methods.

There are many online task scheduling scenarios in an actual cloud environment. The characteristics of online task scheduling are different from those of offline. Online tasks are more variable and complicated. Our next direction of study will be how to implement online task scheduling using DRL.

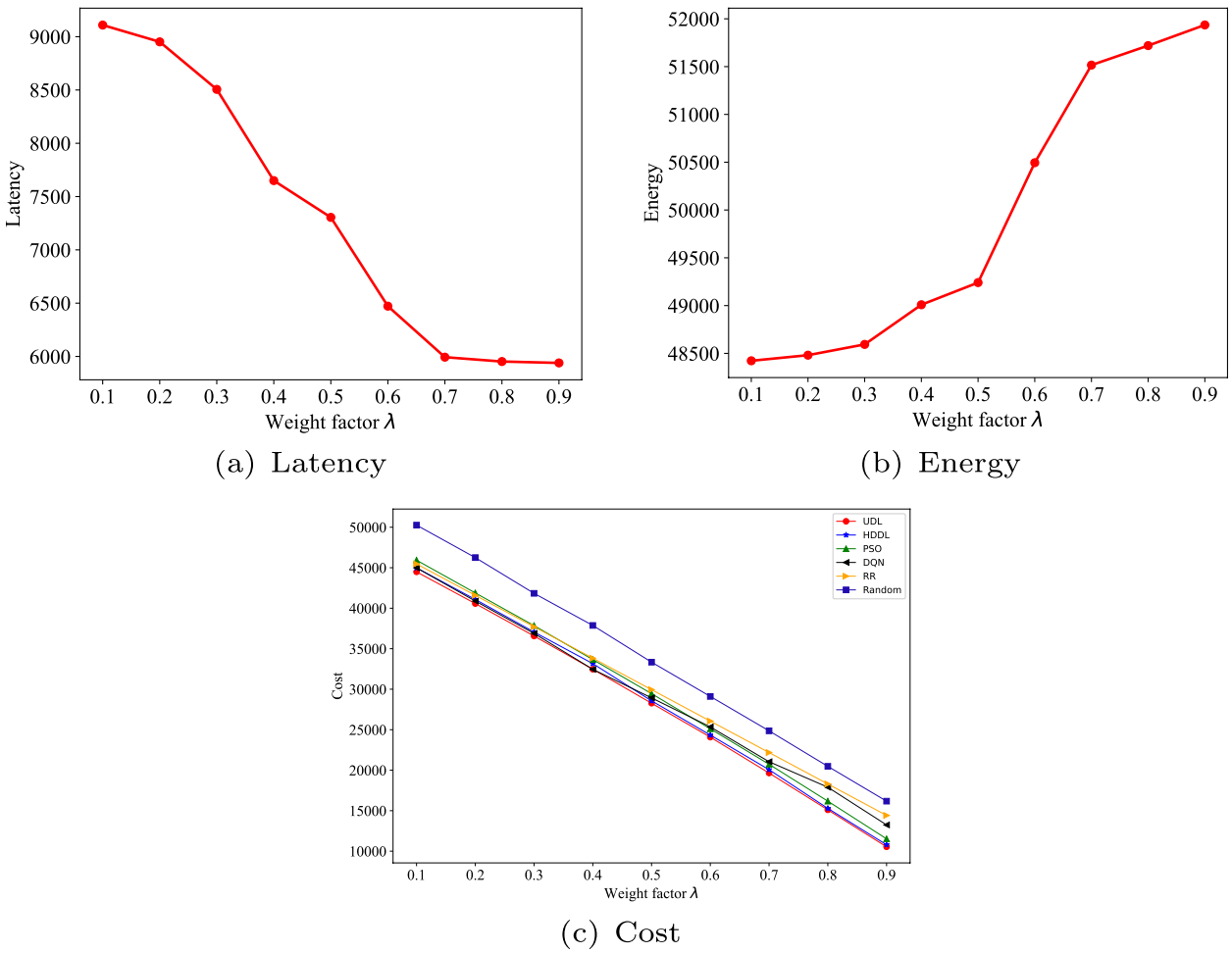

Fig. 8 Algorithm performance comparison with different λ

Acknowledgements
The authors thank the reviewers for their insightful comments and suggestions to improve the quality of the paper. Zhiping Peng is corresponding author.

Authors' contributions
All authors contributed to the study conception and design. Material preparation and data collection were performed by Delong Cui and Hao Zhang. Investigation, formal analysis and validationwere performed by Zhiping Peng and Jianpeng Lin. The first draft of the manuscript was written by Qirui Li and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding
This work was sponsored by Guangdong basic and applied basic research foundation(2022A1515012022, 2021A1515012252, 2020A1515010727); Maoming Science and Technology Project(mmkj2020008); Projects of PhDs' Start-up Research of GDUPT (XJ2022000301).

Availability of data and materials
Data will be made available upon reasonable request to the corresponding author.

Declarations

Ethics approval and consent to participate
This material is the authors' own original work, which has not been previously published elsewhere. The paper is not currently being considered for publication elsewhere.

Competing interests
The authors declare no competing interests.

Received: 15 February 2023 Accepted: 20 July 2023
Published online: 28 July 2023

References

1. Panneerselvam J, Liu L, Antonopoulos N (2020) An approach to optimise resource provision with energy-awareness in datacentres by combating task heterogeneity. IEEE Trans Emerg Top Comput 8(3):762–780
2. Zhang Q, Zhani MF, Boutaba R, Hellerstein JL (2014) Dynamic heterogeneity-aware resource provisioning in the cloud. IEEE Trans Cloud Comput 2(1):14–28
3. Mansouri N, Javidi MM (2020) Cost-based job scheduling strategy in cloud computing environments. Distrib Parallel Databases 38(2):365–400
4. Madni SHH, Latiff MSAA, Coulibaly Y, Abdulhamid SM (2016) Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. Clust Comput 20:2489–2533
5. Mathew T, Sekaran KC, Jose J (2014) Study and analysis of various task scheduling algorithms in the cloud computing environment. 2014 International Conference on Advances in Computing, Communications and Informatics(ICACCI), Pune, India, pp 658-664
6. Patil N, Aeloor D (2017) A review - different scheduling algorithms in cloud computing environment. 2017 11th International Conference on Intelligent Systems and Control(ISCO), Coimbatore, India, pp 182-185
7. Verma A, Kaushal S (2017) A hybrid multi-objective Particle Swarm Optimization for scientific workflow scheduling. Parallel Comput 62:1–19
8. Zuo L, Shu L, Dong S, Zhu C, Hara T (2015) A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. IEEE Access 3:2687–2699
9. Alkayal ES, Jennings NR, Abulkhair MF (2016) Efficient task scheduling multi-objective particle swarm optimization in cloud computing. 2016 IEEE 41st Conference on Local Computer Networks Workshops(LCN Workshops), Dubai, United Arab Emirates, pp 17-24
10. Duan H, Chen C, Min G, Wu Y (2017) Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems. Futur Gener Comput Syst 74:142–150
11. Srichandan S, Kumar TA, Bibhudatta S (2018) Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm. Futur Comput Inf J 3(2):210–23
12. Peng Z, Cui D, Zuo J, Li Q, Xu B (2015) Random task scheduling scheme based on reinforcement learning in cloud computing. Clust Comput 18:1595–1607
13. Peng Z, Cui D, Zuo J, Lin W (2015) Research on cloud computing resources provisioning based on reinforcement learning. Math Probl Eng 2015:1–12
14. Cui D, Peng Z, Xiong J, Xu B, Lin W (2020) A reinforcement learning-based mixed job scheduler scheme for Grid or IaaS cloud. IEEE Trans Cloud Comput 4:1030–1039
15. Thein T, Myo MM, Parvin S, Gawanmeh A (2020) Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers. J King Saud Univ - Comput Inf Sci 32(10):1319–1578
16. Wei Y, Daniel K, Liu S, Li P, WU L, Meng X, (2019) A reinforcement learning based auto-scaling approach for SaaS providers in dynamic cloud environment. Math Probl Eng 2019:1–11
17. Liang H, Zhang X, Hong X, Zhang Z, Li M, Hu G, Hou F (2021) Reinforcement learning enabled dynamic resource allocation in the internet of vehicles. IEEE Trans Ind Inform 17(7):4957–4967
18. Guo W, Tian W, Ye Y, Xu L, Wu K (2021) Cloud resource scheduling with deep reinforcement learning and imitation learning. IEEE Internet Things J 8(5):3576–3586
19. Chudasama V, Bhavsar M (2020) A dynamic prediction for elastic resource allocation in hybrid cloud environment. Scalable Comput: Pract Experience 21(4 SI):661-672
20. Lakhan A, Mastoi Q, Elhoseny M, Memon MS, Mohammed MA (2022) Deep neural network-based application partitioning and scheduling for hospitals and medical enterprises using IoT assisted mobile fog cloud. Enterp Inf Syst 16(7):1883122
21. Rangra A, Sehgal VK, Shukla S (2019)A novel approach of cloud based scheduling using deep-learning approach in E-Commerce domain. Int J Inf Syst Model Des 10(3 SI):59-75
22. Lin J, Cui D, Peng Z, Li Q, He J (2020) A two-stage framework for the multi-user multi-data center job scheduling and resource allocation. IEEE Access 8:197863–197874
23. Liu Q, Zhai J, Zhang Z, Zhong S, Zhou Q, Zhang P, Xu J (2018) A survey on deep reinforcement learning. Jisuanji Xuebao/Chin J Comput 41(1):1–27
24. Sharma AR, Kaushik P (2017) Literature survey of statistical, deep and reinforcement learning in natural language processing. 2017 International Conference on Computing, Communication and Automation (ICCCA), Greater Noida, India, pp 350-354
25. Mnih V, Kavukcuoglu K, Silver D, Veness J (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–33
26. Phaniteja S, Dewangan P, Guhan P, Sarkar A, Krishna KM (2017) A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. 2017 IEEE International Conference on Robotics and Biomimetics(ROBIO), Macau, Macao, pp 1818-1823
27. Peng Z, Lin J, Cui D, Li Q, He J (2020) A multi-objective trade-off framework for cloud resource scheduling based on the Deep Q-network algorithm. Clust Comput 23(4):2753–2767
28. Lin J, Peng Z, Cui D (2018) Deep reinforcement learning for multi-resource cloud job scheduling. 2018 25th International Conference on Neural Information Processing, Siem Reap, Cambodia, pp 289-302
29. Bitsakos C, Konstantinou I, Koziris N (2018) DERP: a deep reinforcement learning cloud system for elastic resource provisioning. 2018 IEEE International Conference on Cloud Computing Technology and Science(CloudCom), Nicosia, Cyprus, 21-29
30. Zhang C, Liu Z, Gu B, Yamori K, Tanaka Y (2018) A deep reinforcement learning based approach for cost- and energy-aware multi-flow mobile data offloading. IEICE Trans Commun E101.B:1625-1634
31. Liang H, Feng L, Zhang L, Qian Y (2019) Multi-server multi-user multi-task computation offloading for mobile edge computing networks. Sensors 19(6):1446
32. Li Q, Peng Z, Cui D, Lin J, He J (2022) Two-stage selection of distributed data centers based on deep reinforcement learning. Clust Comput 25:2699–2714
33. Li K, Zhang T, Wang R, Qin W, He Hi, Huang H (2021) Research reviews of combinatorial optimization methods based on deep reinforcement learning. Acta Automatica Sinica 47(11):2521–2537

Publisher's Note