Energy-efficient Joint Task Assignment and Migration in Data Centers: A Deep Reinforcement Learning Approach

Jiong Lou, Zhiqing Tang, and Weijia Jia, *Fellow, IEEE*

***Abstract*—Energy-efficient task scheduling in data centers is a critical issue and has drawn wide attention. However, the task execution times are mixed and hard to estimate in a real-world data center. It has been conspicuously neglected by existing work that scheduling decisions made at tasks' arrival times are likely to cause energy waste or idle resources over time. To fill in such gaps, in this paper, we jointly consider assignment and migration for mixed duration tasks and devise a novel energy-efficient task scheduling algorithm. Task assignment can improve resource utilization, and migration is required when long-running tasks run in low-load servers. Specifically: 1) We formulate mixed duration task scheduling as a large-scale Markov Decision Process (MDP) problem; 2) To solve such a large-scale MDP problem, we design an efficient Deep Reinforcement Learning (DRL) algorithm to make assignment and migration decisions. To make the DRL algorithm more practical in real scenarios, multiple optimizations are proposed to achieve online training; 3) Experiments with real-world data have shown that our algorithm outperforms the existing baselines 14% on average in terms of energy consumption while keeping the same level of Quality of Service (QoS).**

***Index Terms*—Energy-efficient task scheduling, Deep reinforcement learning, Data center.**

I. Introduction

With the rapid development of cloud computing, more and more tasks can be submitted to data centers to be processed. One of the severe problems in data centers is the tremendous amount of energy consumption, which causes high operating costs and substantial carbon dioxide (CO_2) emissions. A three percent reduction in energy cost for a large company like Google can translate into over a million dollars in cost savings [1]. Statistic results show that much energy is wasted when servers run at a very low load [2]. Consequently, it is indispensable to devise energy-efficient task scheduling algorithms to reduce the energy consumption of data centers.

In the area of energy-efficient task scheduling, researchers focus on the task assignment at the arrival time. When a new task is submitted, the task is assigned to a proper server, and then the server initiates a virtual machine (VM) or a container for it. Task assignment can be considered a kind of classical bin-packing problem [3] with several heuristic algorithms being devised for wide usages, such as Round-Robin [4] and Best Fit [5]. In [6]–[8], Deep Reinforcement Learning (DRL) based algorithms are designed to assign tasks to achieve high performance. Some previous work [3], [9] studies energy-efficient task migration. During the runtime, tasks running in VMs or containers can be dynamically migrated to other servers for consolidation. In this way, more servers can be freed and switched to sleep mode [10].

However, few studies on joint task assignment and migration can achieve better energy efficiency during the entire lifecycle of tasks. On the one hand, since the task duration in data centers often follows a long-tail distribution [11], long-running tasks form a small fraction of the total number of tasks but consume a large number of resources. Only performing the task assignment cannot ensure these long-running tasks' energy efficiency since it can hardly distinguish long-running tasks at their arrival times [12] or assign them to a few servers for consolidation. When the task workload decreases, short-running tasks are finished earlier and leave long-running tasks running on the server in low load, which results in energy wastage. On the other hand, a large number of short-running tasks are running in the data center, and migrating numerous short-running tasks costs from a few seconds to some minutes [13], which significantly affects the performance of these short-running tasks. Due to the significant estimation errors of task runtime [14], it is inherently hard to distinguish long-running tasks from short-running tasks at the arrival time, which makes mixed duration task scheduling more challenging. In short, previous work does not pay enough attention to scheduling tasks with mixed duration during the entire lifecycle.

To fulfill this gap, the mixed duration task scheduling problem is formulated by jointly considering task assignment and migration. The task assignment and migration process is a sequence of correlated scheduling decisions. To reflect the correlations between scheduling decisions at different times, the scheduling problem is modeled as a Markov Decision Process (MDP) problem. The MDP problem has a huge state space considering many simultaneous running tasks in data centers. To deal with such a large-scale MDP problem, we choose DRL, which can handle the huge state space of complicated control problems as the underlying technology. Previous work based on DRL technologies for resource allocation in the data

Corresponding authors: Zhiqing Tang, Weijia Jia.

J. Lou is with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China, and is also with Institute of Artificial Intelligence and Future Networks, Beijing Normal University, Guangdong, 519087, PR China. E-mail: lj1994@sjtu.edu.cn

Z. Tang and W. Jia are with Institute of Artificial Intelligence and Future Networks, Beijing Normal University (BNU Zhuhai), and W. Jia is also with Guangdong Key Lab of AI and Multi-Modal Data Processing, BNU-HKBU United International College Zhuhai, Guangdong, 519087, PR China. E-mails: domain@sjtu.edu.cn, jiawj@bnu.edu.cn

center mainly needs heavy offline training of Deep Neural Network (DNN) [6], [15]–[17], which has the following shortcomings: (1) It requires a large amount of workload traces and elaborately generated state transition profiles to model the mixed duration tasks. (2) The transition probability of container resource demand is quasi-static for a relatively short-term (e.g., one day) [9]. Historical data collected several days ago can hardly reflect the current scenarios' characteristics, so online training is critical.

To satisfy the time-varying resource demand and avoid DNN's heavy offline training, we propose an Online DRL-based Task Scheduling algorithm (ODTS), consisting of the task assignment policy and the task migration policy. The task assignment policy assigns tasks to proper servers at their arrival time. Various optimizations are proposed to make DNN train in an online manner. First, to better evaluate each assignment decision, the reward is formulated as the difference between average power before and after the assignment. Then, to achieve the online training, a weight-sharing structure is used to reduce the number of DNN parameters, and a sort module is applied to recognize different permutations of server state. Besides, efficient exploration is applied to accelerate the convergence speed and generate training data to alleviate the sparse reward problem. Furthermore, ODTS heuristically selects a few tasks already executing for a long time to reduce the action space of simultaneously migrating all tasks. The proposed task migration policy is then applied to choose the proper server for migrating each selected task. Finally, to eliminate the impact of the varied task workload, the reward is reformulated as the integral of CPU utilization divided by the average power consumption, instead of the linear combination of energy consumption and QoS, etc. [6], [17]. Experiments with real-world data have shown that the proposed algorithm outperforms the existing baselines by 14% on average in terms of energy consumption while keeping the same level of Quality of Service (QoS) and achieving online training.

In short, our contributions can be summarized as follows:

1) To energy-efficiently schedule the mixed duration tasks in data centers, we jointly consider task assignment and migration during the entire lifetime of tasks. We formulate the task scheduling problem and further model it as an MDP problem.
2) A DRL-based algorithm is devised to solve the MDP problem. Furthermore, to satisfy the time-varying resource demand, we optimize the DNN structure, reformulate the reward, adopt efficient action selection, and generate training data to achieve online training.
3) We implement ODTS and evaluate the performance. Experiments show that, with nearly the same Quality of Service (QoS) and the same migration times, the proposed algorithm outperforms the existing baseline algorithms by 14% on average in terms of energy consumption.

The remainder of this paper is organized as follows. The related work is reviewed in Section 2. Section 3 describes the DRL, defines the system model, and formulates the task scheduling problem. The proposed DRL-based algorithm for energy-efficient task scheduling is discussed in Section 4. The performance of ODTS is evaluated in Section 5. Finally, the paper is concluded in Section 6.

II. Related work

In this section, we introduce studies on energy-efficient task scheduling and reinforcement learning applications in the computer network.

Energy-efficient task assignment and migration: Studies on energy-efficient task scheduling can be divided into task assignment and task migration. When a new task is submitted, task assignment dispatches it to a proper server to reduce energy consumption and guarantee the desired resource [18], [19]. Task assignment can be considered a kind of classical bin-packing problem [3]. Many heuristics like First Fit (FF), First Fit Decreasing (FFD) [20], Best Fit (BF) [5], and Best Fit Decreasing (BFD) [21], have been proposed to solve it as a bin-packing scheduling problem [22]. Huang et al. [23] design two heuristic schemes to allocate arriving VMs to appropriate physical servers and migrate the VM when the deployment of VMs is far from optimal.

On the other hand, task migration implemented by migrating VMs or containers is a resource-intensive operation in the data center. Jin et al. [24] model the resource demands of VMs as normal distribution and proposed a stochastic bin-packing algorithm. Han et al. [9] formulate dynamic VM management as a large-scale MDP problem and propose an approximate MDP-based dynamic VM management method. Some other researchers predict the resource demands of VMs with historical data [25] or assume prior distribution of resource requirements [26]. In [27], the researchers design a benders decomposition-based task migration algorithm to shift intensive workloads to locations sufficient in renewable energy sources at a coarse scale and adjust the execution time of the workload in response to real-time renewable energy source fluctuations at a fine scale. Moreover, most dynamic approaches are heuristic-based [28], hence lacking sufficient theoretical performance guarantee.

Reinforcement learning: Reinforcement learning (RL) [29] is an essential branch of machine learning to solve the problem of making decisions. Encouraged by these successes in applying the DRL to solve complex online control problems [30], [31], many researchers use DRL to solve the scheduling problem in computer networks [32], [33]. Mao et al. [15] first use policy gradient DRL in network resource scheduling, and the proposed algorithm achieves the best result in simulated experiments. Ye et al. [16] make further improvements on [15] by using the imitation learning [34] that has been widely applied in robotics [35] and Self-Driving [36]. There has also been some related work on task scheduling in data centers [6]–[8], [17]. Liu et al. [6] propose a novel hierarchical framework in the data center, which weighs the overall resource allocation and power management issues. The best tradeoff between power consumption and processing delay can be effectively achieved with the DRL.

Moreover, RL can also be applied to make task migration decisions. Tang et al. [17] first model the container migration as a multi-dimensional MDP, which is solved by the

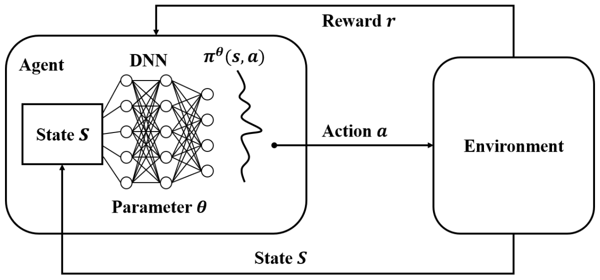


Fig. 1. Reinforcement learning with policy represented via DNN.

deep Q-learning (DQL). They perform various optimizations to improve the efficiency and stability of the system. An adaptive DRL-based framework is proposed to tackle energy consumption issues in cloud task scheduling [7]. In [37], a decentralized reinforcement learning management policy is proposed to optimize the VM migration. Peng et al. [38] propose a novel dynamic service migration scheme based on DRL, aiming to achieve the QoS and migration cost tradeoff. However, these previous studies for RL applications in the computer network require heavy offline DNN training. They need massive workload traces and significant training time, so they cannot be applied to a highly dynamic environment.

III. Background and System Model

In this section, the DRL mentioned above is first reviewed. Next, the system model is described, and the task scheduling problem is formulated. Then, the characteristics of the task scheduling problem are analyzed.

A. Background

As shown in Fig. 1, we consider a general RL setup consisting of an agent interacting with an environment in discrete epochs. Specifically, at each epoch t, the agent observes some state $s(t)$ of the environment, and chooses an action $a(t)$. Following the action, the state of the environment transitions to $s(t{+}1)$ and the agent receives reward $r(t)$. The state transitions and rewards are stochastic and assumed to have the Markov property. The objective is to find a policy $\pi(s)$ mapping a state to an action (deterministic) or a probability distribution over actions (stochastic) to maximize the expected discounted cumulative reward $\mathbb{E}[\sum_{t=t_0}^{\infty}\gamma^t r(t)]$, where t_0 is the current time, $r(\cdot)$ is the reward and $\gamma \in [0,1]$ is the discount factor.

Among all kinds of RL algorithms, Q-learning algorithm [39] has an advantage in fast computation, which is consistent with the requirement of rapid decision-making in data centers. In such an algorithm, the quality of each state-action pair is indicated by Q-value $Q(s(t),a(t))$, which is stored in Q-matrix. $Q(s(t),a(t))$ is equal to the expected discounted cumulative reward defined before. The Q-matrix is initialized to a zero matrix. For discrete-time Markov Decision Process, each $Q(s(t),a(t))$ can be updated online with the learning-rule:

$$\begin{aligned} Q\left(s(t),a(t)\right) \leftarrow\ & (1-\alpha)Q(s(t),a(t)) \\ & + \alpha[r(t) + \gamma \cdot \underset{a(t+1)}{\arg\max}\, Q\left(s(t+1),a(t+1)\right)], \end{aligned} \tag{1}$$

where α is the learning rate.

For most practical problems, it is infeasible to learn all possible combinations of state-action pairs. Thus function approximation technique is commonly used to learn the policy [40]. A function approximator $\pi_\theta(s(t),a(t))$ is parameterized by θ, whose size is much smaller than the number of all possible state-action pairs, where θ represents all parameters of the neural network. Many forms of function approximators can be used to represent the policy. For instance, linear combinations of state/action space features are a popular choice. DeepMind introduced DRL [31] that extends Q-learning to enable end-to-end system control based on high-dimensional sensory inputs. The DRL adopts a DNN called Deep Q-Network (DQN) to derive the correlation between each state-action pair $(s(t),a(t))$ of the system under control and calculate its *value function* $Q\left(s(t),a(t)\right)$. A common-used off-policy algorithm takes the ϵ-greedy policy [41] to derive the action with the highest Q: $\pi(s(t)) = \arg\max_{a(t)} Q\left(s(t),a(t)\right)$ with probability $1-\epsilon$ and choose the other actions randomly with total probability ϵ. In deep Q-learning, the DQN can be trained by minimizing the loss:

$$L\left(\theta^Q\right) = \mathbb{E}\left[\left(y(t) - Q\left(s(t),a(t)|\theta^Q\right)\right)^2\right], \tag{2}$$

where θ^Q is the weight vector of the DQN and $y(t)$ is the target value, which can be estimated by:

$$y(t) = r(t) + \gamma Q\left(s(t+1), \pi\left(s(t+1)|\theta^\pi\right)|\theta^Q\right). \tag{3}$$

However, using a DNN as a function approximator in RL suffers from instability or even divergence. Two practical techniques have been introduced in [31] to improve stability: experience replay and the target network. Unlike traditional RL, a DRL agent updates the DNN with a mini-batch from an experience replay buffer, which stores state transition samples collected during learning. Experience replay can smooth out learning and avoid oscillations or divergence. In order to solve the overestimation problem from max operator $\arg\max_{a(t)} Q\left(s(t),a(t)\right)$, Google DeepMind [31] uses a separate target network that has the same structure as the DQN, to estimate target values $y(t)$ for training the DQN, whose parameters, however, are slowly updated with the DQN weights every $k > 1$ epochs and are held fixed between individual updates.

B. System Model

In this paper, a large-scale data center with M physical servers that offer computing resources is considered. The data center adopts containerization as virtualization technology. The set of physical servers is defined as $\{p_1,...,p_M\}$. The CPU utilization of server p_i at time t is defined as $u_i(t)$. A server can be in active mode for task execution or sleep mode for energy saving. For ease of reference, the key notations used in the paper are summarized in TABLE I.

Task Model: Tasks with mixed duration stochastically arrive over time. Different tasks have diverse requirements for multiple resources, like CPU and memory. When each task arrives, a new container is created to comply with its requirements and run on a physical server. The set of

TABLE I
NOTATIONS

Notation	Description
p_i	i^{th} physical server
$u_i(t)$	CPU utilization of server p_i at time t
c_j	i^{th} container
$C_{j\cdot r}(t)$	CPU requirement of container c_j at time t
$C_{j\cdot a}(t)$	CPU allocation of container c_j at time t
$C_{j\cdot l}(t)$	Allocation server of container c_j at time t
$C'_{j\cdot l}(t)$	Allocation server for container c_j after migration at time t
T_m	The maximum number of migrated tasks in one time slot
$\|C\|$	Total container number
p_{idle}	Power consumption of 0% CPU utilization of a server
p_{max}	Power consumption of 100% CPU utilization of a server
E_i	Energy consumption of server p_i
E_{total}	Total energy consumption
C_{SLAV}	Performance degradation due to SLAV
t_j	Task j's arrival time
$s_a(t)$	System state for task assignment at time t
$s_m(t)$	System state for task migration at time t
$s_c(t)$	Server cluster state at t
$s_i(t)$	State of server p_i at t
s_j^a	Task j's state for task assignment
s_j^m	Task j's state for task migration
A	Action space for task assignment

containers is denoted as $\{c_1, ..., c_N\}$. For container c_j, we set $C_{j\cdot r}(t)$, $C_{j\cdot a}(t)$ and $C_{j\cdot l}(t)$ as the resource requirement, the resource allocation and the container allocation server at time t, respectively. With little information about tasks, obtaining accurate task duration estimates at the arrival time is not trivial. Both the predictor based on the mean task execution time [42] and based on machine learning [43] exhibit significant estimation errors. Therefore, it is assumed that the duration of a task is unknown at its arrival time. Besides, during the running time of a task, the resource requirements of its container are varied.

Task Scheduler: Fig. 2 illustrates the task scheduling system. Similar to Google Borg [44], there is a centralized task scheduler to collect the information from each server and schedule tasks in the data center. As soon as a new task is submitted, the task scheduler assigns it to one server with enough available resources and creates a container for it. During the runtime of the task, the task scheduler monitors the running status of the data center and periodically sends commands to migrate some containers for energy-efficient allocation. Task migration is used to refer to container migration in the following, usually it is considered as hot migration [17], i.e., the shutdown time is not considered. Once a task is completed, it releases the resources, and the corresponding container is stopped.

For convenience, there is assumed to be sufficient disk and network capacity for all tasks on each server. The scheduling of computational power (CPU time) is considered the main factor determining the energy consumption of the servers, and memory is the constraint. Without loss of generality, the servers in the data center are homogeneous, and the resource capacity of each server is normalized to 1.

Different task scheduling decisions will lead to different CPU utilization and energy consumption, so the centralized task scheduler should be carefully designed for energy conservation. This paper aims to design a task scheduling algorithm to reduce the energy consumed during the entire lifecycle of tasks while maintaining an acceptable QoS. The task scheduling problem is formulated as follows.

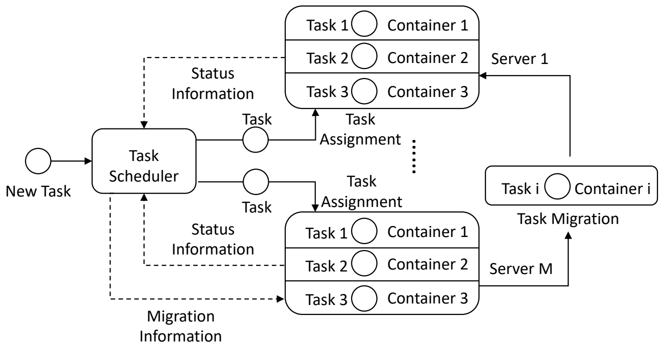


Fig. 2. System architecture. At the arrival time of a new task, the task scheduler assigns it to the target server according to the state of the servers. For the duration of tasks, the task scheduler performs periodical task migration.

C. Problem Formulation and Analysis

The resource utilization and the QoS can benefit from task assignment and migration. Plenty of idle servers can be switched off, and much energy consumption can be reduced through container consolidation. However, an unsuitable migration strategy performed on short-running tasks may be useless, waste network bandwidth, or even increase execution times. Besides, if too many containers are merged into a server, the risk of resource over-allocation will be risen dramatically, leading to performance degradation. Therefore, there exists a trade-off between QoS and energy consumption. Considering QoS and energy consumption, the task scheduling problem can be formulated, as shown below.

QoS: Achieving desirable QoS requirements is extremely important for a data center. The QoS requirements are commonly defined by Service Level Agreement (SLA), which describes such characteristics as throughput or response time. The average SLA violation (SLAV) percentage represents the percentage of average CPU performance that has not been allocated to a container when requested, resulting in performance degradation, e.g., more latency or lower reliability [17], [25], defined as:

$$C_{SLAV} = \frac{\sum_{j=1}^{n} \int \left(C_{j\cdot r}(t) - C_{j\cdot a}(t)\right) dt}{\sum_{j=1}^{n} \int \left(C_{j\cdot r}(t)\right) dt}, \tag{4}$$

Energy Consumption: In general, the total energy consumption E_{total} is equivalent to the addition of each server's energy consumption E_i:

$$E_{total} = \sum_{i=1}^{M} E_i. \tag{5}$$

The linear approximation model, widely adopted in the literature, is used to evaluate the power consumption of servers. If server p_i is switched off or set in sleep mode, its consumption is approximately equal to 0 [6]. Otherwise, its power consumption is proportional to its resource utilization, which was indicated by [45]. For each server, the energy consumption is equal to the integral of power consumption over time:

$$E_i = \int \left(p_{idle} + (p_{max} - p_{idle}) \times u_i(t)\right) dt, \tag{6}$$

where p_{idle} and p_{max} represent the power consumption of 0% and 100% CPU utilization of server p_i , respectively. Besides, $u_i(t)$ is the CPU utilization of p_i at t, defined as:

$$u_i(t) = \min\left\{1, \sum 1[C_{j\cdot l}(t) = p_i] C_{j\cdot r}(t)\right\}, \tag{7}$$

where $1[\cdot]$ is the indicator function, which is equal to 1 when the condition holds, and 0 otherwise. Specially, we set $C_{j\cdot r}\left(t\right)$ and $C_{j\cdot a}\left(t\right)$ as the CPU requirement and the CPU allocation at t, respectively. Recall that the resource capacity of server p_i is normalized to 1. When $u_i(t) > 0$, it means that server p_i is in the active mode at t. In the system model, the CPU allocation of a task is proportional to its CPU requirement, which is defined as:

$$C_{j\cdot a}\left(t\right) = \frac{C_{j\cdot r}\left(t\right)}{\sum 1[C_{j\cdot l}(t) = p_i] C_{j\cdot r}(t)} \times u_i(t), \tag{8}$$

However, too many task migrations result in much data transmission, which consumes a large amount of network bandwidth and increases the task execution time. The maximum number of migrated tasks in one time slot as T_m is set according to [9], i.e.,

$$\sum_{i=1}^{|C|} 1\left[C'_{j\cdot l}\left(t\right) \neq C_{j\cdot l}\left(t\right)\right] \leq T_m, \tag{9}$$

where $C'_{i\cdot l}\left(t\right)$ is defined as the allocation for task c_i after migration at time t.

Problem Definition: The target is to reduce the total cost for the long-term:

Problem 1:

$$\min \quad C = \omega_1 \cdot E_{total} + \omega_2 \cdot C(SLAV) \tag{10}$$

$$\text{s.t.} \quad 0 \leq \sum\nolimits_{C_{j\cdot l}(t) = p_i} C_{j\cdot a}(t) \leq 1, \ i = 1, ..., M, \forall t, \tag{11}$$

where ω_1 and ω_2 are two positive parameters to control the weights of E_{total} and C_{SLAV}, respectively. The constraints are the maximum number of task migrations defined in Eq. (9) and the resource capacity constraints.

Furthermore, the total energy consumption is analyzed:

$$\begin{aligned} E_{total} &= \sum_{i=1}^{M} E_i = \int (\sum_{i=1}^{M} p_{idle} \times 1[u_i(t) > 0]) \ dt \\ &+ \int (\sum_{i=1}^{M} (p_{max} - p_{idle}) \times u_i(t) \times 1[u_i(t) > 0]) \ dt \\ &= \int (\sum_{i=1}^{M} p_{idle} \times 1[u_i(t) > 0]) dt \\ &+ (p_{max} - p_{idle}) \times (1 - C_{SLAV}) \int (\sum_{i=1}^{n} C_{j\cdot r}(t)) \ dt. \end{aligned} \tag{12}$$

From Eq. (12), it can be observed that the E_{total} consists of two terms. The first term $\int(\sum_{i=1}^{M} p_{idle} \times 1[u_i(t) > 0])dt$ is related to the scheduling algorithm and the second term $(p_{max} - p_{idle}) \times (1 - C_{SLAV}) \int(\sum_{i=1}^{n} C_{j\cdot r}(t))dt$ is a constant for a certain set of tasks and the same C_{SLAV}. In order to consume less energy, we must reduce the accumulated active server number $\int(\sum_{i=1}^{M} 1[u_i(t) > 0])dt$ in the premise of the same C_{SLAV}. From this perspective, it is reasonable to use SLAV as a constraint in the case of the linear approximation power model.

Problem Analysis: We explain the necessity of jointly considering task assignment and migration, and the reason why DRL is chosen to solve the task scheduling problem.

Analyzing real-world data from Google cluster [46], we find that only 8% of tasks run for more than 2 hours but use 92.9% resources, which is much more than short-running tasks. More detailed analysis from multiple workloads also shows similar results [11], [47], which means the long-tail distribution of task duration is typical.

Energy-efficient mixed duration task scheduling requires both judicious task assignment and migration. On the one hand, frequent task migration may affect performance for a massive number of short-running tasks. Besides, many times of task migration will produce much migration cost, e.g., bandwidth. As a result, the task assignment at the arrival time should be well designed. On the other hand, only task assignment cannot guarantee their energy efficiency during the entire lifecycle for long-running tasks without prior information on task duration. For example, if Best Fit is performed to assign tasks. When the workload decreases rapidly, the resource utilization of long-running tasks is 58.1%, much lower than short-running tasks' resource utilization, 81.8%.

The problem mentioned before is an advanced bin-packing problem that is NP-hard. It can be solved optimally, but it would require a long time. Moreover, most of the existing heuristic algorithms are unstable for dynamically arrived tasks in a real data center, which cannot handle large-scale problems that largely slow down decision-making. In this problem, assume that $C(\tau)$ is the total cost until T_τ. Based on Eq. (4) and Eq. (10), $C(\tau)$ obeys first-order Markov Process as:

$$C(\tau) = 1[\tau > 0] \times C(\tau - 1) + \omega_1 \sum_{i=1}^{M} p_i(\tau) + \omega_2 C_{SLAV}(\tau). \tag{13}$$

Therefore, it can be formulated as an MDP problem and is suitable to adopt RL algorithms in solving this problem.

However, in data centers with massive physical servers, task scheduling often exhibits high dimensions in state and action spaces, prohibiting the use of traditional RL techniques. This is because the convergence speed of traditional RL techniques is in general proportional to the number of state-action pairs [29]. Therefore, we adopt the emerging DRL technique, which has the potential to handle large state space of complicated control problems, to solve the task scheduling problem, as presented in the next section.

Algorithm 1 ODTS

```
1: Initialize $\mathbf{D}$, $\mathbf{Q}$, $\hat{\mathbf{Q}}$ for task assignment.
2: Initialize $\mathbf{D_m}$, $\mathbf{Q_m}$, $\hat{\mathbf{Q}}_\mathbf{m}$ for task migration.
3: $j = 0$
4: $k = 0$
5: for $event = e_0$ to $e_N$ do
6:     if $event = 0$ then
7:         /* DRL-based Task Assignment */
8:         $j = j + 1$
9:     else
10:        /* DRL-based Task Migration */
11:        $k = k + 1$
12:    end if
13: end for
14: end
```

IV. ODTS: DRL-based Joint Task Assignment and Migration Algorithm

In this section, the proposed DRL-based task scheduling algorithm ODTS is introduced, as shown in Algorithm 1. ODTS consists of two parts, the task assignment and the task migration. For task assignment, the replay memory, the Q-network, and the target Q-network are D, Q, and $\hat{Q}$, respectively. For task migration, the replay memory, the Q-network, and the target Q-network are D_m, Q_m and $\hat{Q}_m$, respectively. There are two types of events in ODTS: task assignment event and task migration event. The task assignment event and the task migration event are denoted as 0 and 1, respectively. For task assignment, the decision epoch is defined as the arrival time of each task. At each decision epoch, the task scheduler allocates a newly arriving task to one of the physical servers according to the DRL-based task assignment policy. Then, the task is loaded into a new container, which shares the physical resources with other containers. For task migration, the decision-making is periodical. A slotted scheduling framework is adopted for task migration, where time is divided into a sequence of time slots of the same duration. Each time slot is the epoch of task migration. The task scheduler makes the task migration decision according to the state information at the beginning of each epoch. The epoch count is separate for task assignment and migration.

A. DRL-based Task Assignment

In this section, we present how the task scheduler assigns tasks to suitable servers at the arrival time. Function 2 illustrates task assignment for one epoch. The reinforcement learning settings are introduced in Section IV-A1. Multiple optimizations to improve the efficiency and robustness of action selection are described in Section IV-A2.

1) Reinforcement Learning Settings: The environment of the DRL-based task assignment is a set of servers in the data center. The state space, action space, and reward are defined as follows:

State Space: The system state $s_a(t_j)$ at task j's arrival time t_j is defined as the union of the server cluster state $s_c(t_j)$ and the task j's state s^a_j, i.e., $s_a(t_j) = [s_c(t_j), s^a_j]$. The state of a single server p_i at time t is defined as $s_i(t)$. In this paper, since the duration of a task is unavailable at the arrival time, the longest runtime of the tasks still running on the server p_i is used to represent the state better and denoted as $d^i(t)$:

$$d^i(t) = \max_{C_{j'\cdot l}(t)=p_i} (j' - t_j). \tag{14}$$

$s_i(t)$ is the combination of resource utilization $u_i(t)$ and the longest runtime $d^i(t)$:

$$s_i(t) = [u_i(t), d^i(t)]. \tag{15}$$

The longest runtime $d^i(t)$ of the tasks on the server p_i gives the basic information of the tasks on the server, for long-running tasks remaining on servers often result in low resource utilization. Therefore, the system state $s_a(t_j)$ can be represented by combining resource utilization $u_i(t_j)$, longest runtime $d^i(t)$ and resource requirement $C_{j\cdot r}$, as follows:

$$\begin{aligned} s_a(t_j) =& [s_c(t_j), s^a_j] = [s_1(t_j), ..., s_M(t_j), s^a_j] \\ =& [u_1(t_j), d^1(t_j), ..., u_M(t_j), d^M(t_j), C_{j\cdot r}]. \end{aligned} \tag{16}$$

The state space consists of all possible states and thus has a high dimension.

Action Space: In order to reduce the action space, the continuous-time and event-driven decision framework [6] is adopted, in which each decision epoch coincides with the arrival time of a new container request. In this way, the action at each decision epoch is simply the index of the target server for task assignment, which ensures that the available actions are enumerable at each epoch. As a result, the action space of the DRL-based task assignment is defined as follows:

$$A = \{1, 2, \dots, M\}. \tag{17}$$

It can be observed that the action space has the same size as the total number of servers.

Reward: In previous work, researchers primarily used a linear combination of energy consumption and QoS to represent the reward r_j to balance efficiency and performance. However, the training process can be unstable with such a form of reward. It is hard to trade off energy consumption and QoS by DQL, a model-free technique. Besides, according to Eq. (12), SLAV has a linear relationship with energy consumption. Therefore, we use the SLAV as a constraint and focus on reducing the total energy consumption.

The reward function $r(t_j)$ is defined as:

$$r(t_j) = \frac{E_{total}(t_{j+1})}{t_{j+1} - t_j} - \frac{E_{total}(t_j)}{t_j - t_{j-1}}, \tag{18}$$

where t_j and t_{j+1} are the arrival time of the task j and task $j+1$, respectively. $E_{total}(t_j)$ is the energy consumption during the epoch between t_j and t_{j-1}. Specially, $\frac{E_{total}(t_1)}{t_1 - t_0}$ is set to 0. The reward is measured by reducing the average energy consumption at the current epoch by the average energy consumption of the last epoch. The arrival time of each task is dynamic, which means the varied length of each epoch. Compared with the total energy consumption, the energy consumption averaged by epoch length can lead to lower variance. Besides, the difference between average energy consumption better represents the effect of scheduling the arrival tasks on this epoch. Such a reward helps Q-network

Function 2 DRL-based Task Assignment

```
1: Generate random number $\vartheta$.
2: $A = \emptyset$
3: for $a = 1$ to $M$ do
4:     if $a$ complies with Eq. (19) and Eq. (20) then
5:         $A = A \cup \{a\}$
6:     end if
7: end for
8: if $\vartheta > \epsilon$ then
9:     Select $a(t_j) = \arg\max_{a(t_j)} Q(s_a(t_j), a(t_j)|\theta)$ from $A$.
10: else
11:     Select an efficient action $a(t_j)$ from $A$ with least estimated instantaneous power in Eq. (21).
12: end if
13: Calculate $r(t_j)$ by Eq. (18).
14: Store transition $(s_a(t_j), a(t_j), r(t_j), s_a(t_{j+1}))$ into $\mathbf{D}$.
15: Generate random number $\vartheta$.
16: if $\vartheta > \epsilon_g$ then
17:     Set $a(g)$ as a server in sleep mode.
18:     Store transition $(s_a(t_j), a(g), r_g, s_a(t_{j+1}))$ into $\mathbf{D}$.
19: end if
20: Perform action $a(t)$.
21: Select random samples $c(s_a(t_k), a(t_k), r(t_k), s_a(t_{k+1}))$ from $\mathbf{D}$.
22: The weights of $\mathbf{Q}$ then are optimized.
23: if $j\%P = 0$ then
24:     Reset $\hat{\mathbf{Q}} = \mathbf{Q}$
25: end if
26: end
```

train in an online manner. Next, we introduce how the task scheduler selects a proper action according to the current state.

2) Action Selection: Each action needs to satisfy two constraints in lines 3 - 7 of Function 2. The first is the resource constraint defined as follows:

$$u_i(t_j) + C_{j \cdot a}(t_i) \leq 1 \quad j = 1, 2, ..., m, \tag{19}$$

where task c_j is tried to be executed on the server p_i. Resource constraint means the resource utilization in a server cannot exceed the resource capacity. The second one is the SLAV constraint for CPU utilization:

$$\begin{cases} \frac{\sum_{i=1}^{j}(C_{i\cdot r}(t) - C_{i\cdot a}(t))}{\sum_{i=1}^{j} C_{i\cdot r}(t)} \leq \upsilon, & \frac{\sum_{i=1}^{j-1}(C_{i\cdot r}(t) - C_{i\cdot a}(t))}{\sum_{i=1}^{j-1} C_{i\cdot r}(t)} \leq \upsilon, \\ u_j(t_j) + C_{j\cdot a}(t_j) \leq 1, & \frac{\sum_{i=1}^{j-1}(C_{i\cdot r}(t) - C_{i\cdot a}(t))}{\sum_{i=1}^{j-1} C_{i\cdot r}(t)} > \upsilon, \end{cases} \tag{20}$$

where $\upsilon \in [0, 1]$ is the SLAV constraint parameter that controls the instantaneous SLAV after each task assignment. If the former instantaneous SLAV is higher than υ, then the CPU utilization of the selected server cannot exceed the CPU capacity after the task assignment, so the SLAV will decrease gradually. Otherwise, the new instantaneous SLAV should still be less than υ.

In lines 8 - 12 of Function 2, the action is selected based on ϵ-greedy policy, including exploration and exploitation. A threshold ϵ is set in advance, and a random number ϑ is generated for each action selection. The details of action selection are described below:

- When $\vartheta \leq \epsilon$, the action is selected by exploration.
- Otherwise, the best action $a(t) = \arg\max_{a(t)} Q(s(t), a(t), \theta)$ is selected by exploitation according to the Q-values calculated by the Q-network.

Exploration: In exploration, the agent of the general deep Q-learning algorithm randomly selects an action. However, random assignment for tasks may result in higher energy consumption. For example, assigning tasks to servers in sleep mode will make more servers run in low load, causing energy waste. Besides, the task scheduler cannot learn efficient actions from such random exploration, which slows down the DQN's training process. Therefore, to speed up the training and achieve online training, more efficient exploration is adopted: the task scheduler assigns each task to the server with the least estimated instantaneous power consumption P'_e after the assignment, which is defined as:

$$P'_e = \sum_{i=1}^{M} (p_{idle} + (p_{max} - p_{idle}) \times u'_i(t)), \tag{21}$$

where $u'_i(t)$ is the estimated CPU utilization of p_i at time t:

$$u'_i(t) = \begin{cases} \min\{1, \sum 1[C_{k\cdot l}(t) = p_i]C_{k\cdot a}(t) + C_{j\cdot r}(t)\}, \\ \qquad\qquad\qquad\qquad\qquad C_{j\cdot l}(t) = p_i, \\ \sum 1[C_{k\cdot l}(t) = p_i]C_{k\cdot a}(t), \qquad C_{j\cdot l}(t) \neq p_i. \end{cases} \tag{22}$$

Exploitation: In exploitation, for system state $s(t)$, action with the highest Q-value, $a(t) = \arg\max_{a(t)} Q(s(t), a(t), \theta)$, is selected by the task scheduler, where $Q(s(t), a(t), \theta)$ is produced by the DQN. Most studies on DRL-based algorithms in the communication network include offline DNN construction, which needs massive workload traces for a long time and elaborately generated state transition profiles. Besides, offline DNN training often costs a lot of time. Unlike previous studies, the DQN designed in this paper is fully trained online. With high-dimensional state space, a conventional feed-forward neural network consists of too many parameters, making it hard to initialize the parameters properly. Such a neural network is unstable at the beginning and needs much training data unavailable for online learning. As a result, it is infeasible to directly use a conventional feed-forward neural network to output Q.

To address this issue, we harness the power of weight sharing for DQN construction, with the basic procedure shown in Fig. 3. The neural network can be divided into two parts. The first part is to extract a lower-dimensional high-level representation of the global state of the system. In this part, $s_c(t_j)$ is sorted based on CPU utilization at first. Therefore, the DQN can recognize different permutations of each server's state. Then Global State Encoder is designed to extract the global information from the union of sorted system state and task resource requests. The extracted global information is the same for each action. For the second part, the output network takes each server's state and the global information as input features to estimate each action's Q value. The shared

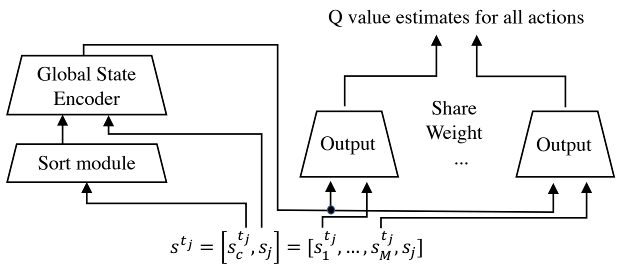

Q value estimates for all actions
Global State Encoder
Sort module
Output
Share Weight ...
Output
$s^{t_j} = [s_c^{t_j}, s_j] = [s_1^{t_j}, ..., s_M^{t_j}, s_j]$

Fig. 3. The structure of the DQN for task assignment.

weight among all M output networks with the same structure is adopted to reduce the number of required parameters and training samples.

Online Training: The DRL-based task assignment is updated online. Updating on the DQN with online transition suffers reward sparse in the scenario of task scheduling. Selecting a sleep server that essentially increases instantaneous power consumption is not energy-efficient scheduling, which is avoided by the proposed efficient exploration. Most of the scheduling decisions are scheduling tasks to servers in active mode. In this case, the reward for such energy-inefficient action will be sparse, and there is an insufficient number of such transitions to train the DQN. Thus, the training process will become unstable, and the task scheduler may not properly estimate $Q(s, a)$ value. The Q-value of selecting sleep servers can be even more significant than others. To solve this issue, the transition generation is performed in lines 15 - 19. Specifically, as the system runs, transitions that randomly select servers in the sleep mode with low reward r_g are generated and stored in the replay memory. Then, the task scheduler performs value updating on the DQN by minimizing the loss in Eq. (2). In lines 23 - 25, it updates the target DQN finally.

B. DRL-based Task Migration

In this section, the DRL-based task migration is presented. The task scheduler collates state information of the data center at the beginning of each epoch and selects suitable servers for task migration.

1) Task Selection: In previous DRL-based algorithms for resource allocation like [17] and [48], the agent selects actions that comprise the aggregation of all tasks' locations at each epoch. However, such action selection has some issues. Firstly, the action space is too large for the task scheduler to find the optimal action. For each task, the action number is the number of servers M, which means that for N tasks, it has M^N actions in total. Moreover, DRL framework typically requires a relatively low-dimensional action space [30] because, in each decision epoch, the DRL agent needs to enumerate all possible actions for the current state and perform inference using DQN to derive the optimal $Q(s, a)$ value estimate. Secondly, task migration must satisfy the resource and SLAV constraints, which is difficult for all actions simultaneously. Finally, based on the long-tail distribution of the task duration in the data center, the migration of a massive number of short-running tasks is a waste of bandwidth and leads to longer task delays.

To address these issues, ODTS selects migration actions for a subset of tasks sequentially in lines 3 - 24 of Function 3. The majority of the tasks in general has a short duration. The task scheduler selects a small set of tasks before selecting migration actions. At the beginning of each epoch, the task scheduler selects the task with the longest runtime from each physical server and sorts them in decreasing order based on the runtime (lines 3 - 4). Then, the task scheduler sequentially selects a reallocation action for each selected task and updates the estimated system state. After making reallocation actions for at most T_m tasks, the task scheduler performs all the actions in order (line 30).

2) Reinforcement Learning Settings: The state space, action space, and reward are defined as follows:

State Space: In the DRL-based task migration, the state at each epoch of task migration t_k is defined as $s_m(t_k)$, similar with the state defined in Section IV-A. The estimated state of the former server, which contains the selected task, should be reduced by the resource allocation of the task:

$$s_{out}(t_j) = u_{out}(t_j) - C_{j\cdot r}, \tag{23}$$

where $s_{out}(t_j)$ is the estimated server state after migrating the container c_j out. Besides, the task state includes the resource requirements and the task runtime in epoch k:

$$d_j(t_k) = t_k - tj, \tag{24}$$

which is useful state information for task migration. Therefore, the system state $s_m(t_k)$ of the DRL-based task migration can be represented as follows:

$$\begin{aligned} s_m(t_k) =& [s(t_k), s_j^m] = [s_1(t_k), ..., s_M(t_k), s_j^m] \\ =& [u_1(t_k), d^1(t_k), ..., u_M(t_j), d^M(t_k), C_{j\cdot r}, d_j(t_k)]. \end{aligned} \tag{25}$$

After each action selection, the estimated state of the server that container c_j is migrated into should be updated:

$$s_{in}(t_j) = u_{in}(t_j) + C_{j\cdot r}. \tag{26}$$

Action Space: The action space is also defined as the index of the server, which is the same as Eq. (17). It can be observed that for each selected task, the action space is reduced to M.

Reward: In the data center, the number of tasks varies with time. With no information about the task duration and future tasks' arrival times, the number and workload of tasks have high variability. It is biased to use the differential value of average power consumption as the reward, since the power consumption of all servers tightly relates to the total resource requirement. The target is to finish a set of tasks with less energy consumption. The reward $r(t_k)$ is formulated as the integral of CPU utilization divided by the energy consumption, which is defined as below:

$$r(t_k) = \frac{\int(\sum C_{j\cdot r})dt}{E_{total}(t_k)}, \tag{27}$$

where $E_{total}(t_k)$ is also the energy consumption during the epoch k after the migration. The proposed reward means that ODTS aims to minimize the energy consumption per

Function 3 DRL-based Task Migration

```
1: Observe $s_m(t_k)$.
2: $A(t_k) = \emptyset$.
3: Set $C$ as the list of each physical server's longest running
   containers.
4: Sort $C$ in decreasing order based on runtime.
5: for $m = 1$ to $T_m$ do
6:     Select the $m^{th}$ container in the sorted list.
7:     Update $s_m(t_k)$ by Eq. (23).
8:     Generate random number $\vartheta_m$.
9:     $A = \emptyset$
10:    for $a = 1$ to $M$ do
11:        if $a$ complies with Eq. (19) and Eq. (20) then
12:            $A = A \cup \{a\}$
13:        end if
14:    end for
15:    if $\vartheta_m > \epsilon_m$ then
16:        Select $a(t_k) = \arg\max_{a(t_k)} Q_m(s_m(t_k), a(t_k)|\theta)$ from
           $A$.
17:    else
18:        Select an efficient action $a(t_k)$ from $A$ with least
           estimated instantaneous power in Eq. (21).
19:    end if
20:    Calculate $r(t_k)$ by Eq. (18).
21:    Store transition $(s_m(t_k), a(t_k), r(t_k), s_m(t_{k+1}))$ into
       $\mathbf{D_m}$.
22:    $A(t_k) = A(t_k) \cup \{a(t_k)\}$
23:    Update $s_m(t_k)$ by Eq. (26).
24: end for
25: Select random samples $c(s(t_l), a(t_l), r(t_l), s(t_{l+1}))$ from
    $\mathbf{D_m}$.
26: The weights of $\mathbf{Q_m}$ then are optimized.
27: if $k\%P_m = 0$ then
28:     Reset $\hat{\mathbf{Q}}_\mathbf{m} = \mathbf{Q_m}$
29: end if
30: Perform action set $A(t_k)$.
31: end
```

workload, which means more energy-efficient. Since the performance of the system is decided by all actions selected in the same epoch, the same reward is set for them.

3) Action Selection: As mentioned above, in the DRL-based task migration function, the task scheduler firstly sorts running tasks by their runtime decreasingly. Then, the task scheduler selects the migration action for each selected task according to ϵ-greedy policy.

Threshold ϵ_m is defined, and the task scheduler generates ϑ_m for ϵ-greedy policy, which is described in Section IV-A. In exploration, instead of random action selection, the task scheduler selects the server with the least estimated instantaneous power consumption as defined in Eq. (21). In exploitation, the task scheduler selects action $a(t) = \arg\max_{a(t)} Q_m(s(t), a(t), \theta)$ for system state $s(t)$. $Q_m(s(t), a(t), \theta)$ is calculated by the DQN, which has the same structure with Fig. 3.

Online Training: The training of the DRL-based task

TABLE II
THE POWER CONSUMPTION AT DIFFERENT CPU UTILIZATION IN WATTS

CPU Utilization(%)	**0%**	**10%**	**20%**	**30%**	**40%**	**50%**
HP ProLiant G4	86	89.4	92.6	96	99.5	102
CPU Utilization(%)	**60%**	**70%**	**80%**	**90%**	**100%**	
HP ProLiant G4	106	108	112	114	117	

migration algorithm is performed online. At each migration epoch k, for each selected container ordered decreasingly, the task scheduler uses ϵ-greedy policy for action selection and performs inference to derive the $Q(s_m(t_k), a)$ value of each state-action pair $(s_m(t_k), a)$. Then, for each action selection, the task scheduler stores all transitions in line 21. After that, the task scheduler also performs value updating by minimizing the loss in lines 25 - 26 and updates the target DQN after every P_m epochs in lines 27 - 29.

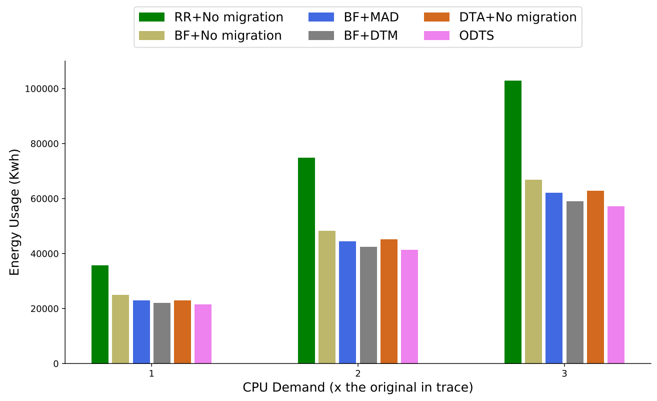
C. Convergence and Computational Complexity Analysis

Watkins et al. [39] proved that the Q-learning technique would gradually converge to the optimal policy under a stationary MDP environment and sufficiently small learning rate. Hence, the proposed DRL-based task scheduling algorithm will converge to the optimal policy when (1) the environment evolves as a stationary, memoryless MDP, (2) the learning rate is sufficiently small, and (3) the DNN is sufficiently accurate to return the action associated with the optimal $Q(s, a)$ estimate. In this paper, the MDP characteristics have been analyzed in Section III-C. Besides, the learning rate is sufficiently small ($0 \leq \alpha \leq 1$) in our problem, and the convergence of deep Q-learning has been illustrated in [30]. Simulation results demonstrate the effectiveness of ODTS in realistic data center environments.

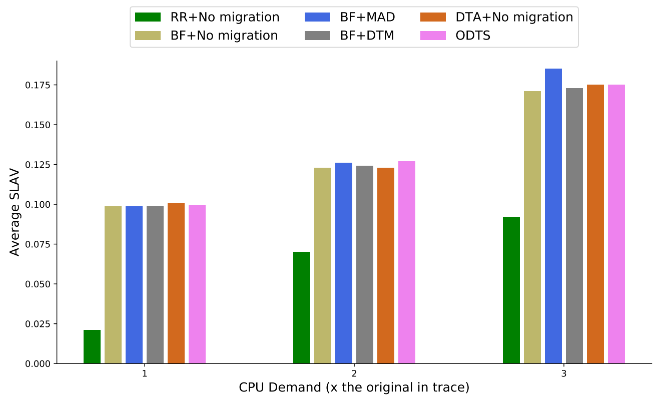
ODTS exhibits low online computational complexity. The computational complexity of the DRL-based task assignment function is proportional to M at each decision epoch. As illustrated in Section III-B, there are M servers, T_m maximum migration number for one migration epoch. The DRL-based task migration function selects target servers for T_m containers, so the computational complexity is proportional to $M \times T_m$ at each migration epoch. The computation overhead of scheduling decisions is relatively insignificant for the data center.

V. PERFORMANCE EVALUATION

In this section, multiple experiments are conducted to evaluate ODTS. Firstly, the experimental setup is described in Section V-A. Then, from perspectives of energy consumption and QoS, ODTS is compared with multiple baselines in Section V-B. Next, ODTS is compared with the non-optimized DRL-based task scheduling algorithm to prove the effectiveness of our optimization in Section V-C. Finally, ODTS is evaluated by investigating the optimal between energy consumption and QoS in Section V-D.



(a) Energy Usage for different times of CPU demand.



(b) Average SLAV for different times of CPU demand.

Fig. 4. Comparison among ODTS, DTA, DTM, MAD, RR and BF with different CPU demand and task number = 2700000.

A. Experimental Setups

In the following experiments, for simplicity and without loss of generality, physical servers in a data center are homogeneous. The peak power of each server is $P_{max} = 117W$, and the idle power consumption is $P_{idle} = 86W$ [48]. Moreover, the power model of each server is shown in TABLE II. To simplify the simulation, the task migration time and the transition time of server mode are set to 0. The number of machines in the data center M is set to 600. The epoch for migration is set to 30 minutes, and the maximum number of task migrations in one epoch is set to 2% of the current number of tasks. It is noted that the proposed algorithm can be easily applied to the case with more servers as well. Moreover, the task scheduling problem in the heterogeneous server data centers can be solved by adding available resources to the system state.

In this section, the real data center workload traces are collected from Google cluster-usage [46], which provides the server cluster usage data over a month-long period in May 2011. Since the CPU and memory-related information of tasks are extracted in the cluster trace, and the types of tasks in the cluster in recent years are basically two types of computation-intensive tasks and delay-sensitive tasks. So the cluster trace is very representative. Even data from 2011 is still widely used in recent years [49], [50]. The extracted task traces include task arrival time (absolute time value), task duration, and resource requirements of each task, which include CPU and memory (normalized by the resource of one server). All the extracted tasks are with a duration from a few seconds to a few days, ordered increasingly based on their arrival time. In order to simulate the workload on a data center with 600 servers, the traces are split into 10 segments, and each segment contains about 2700000 tasks, corresponding to the workload for a 600-machine data center in three weeks.

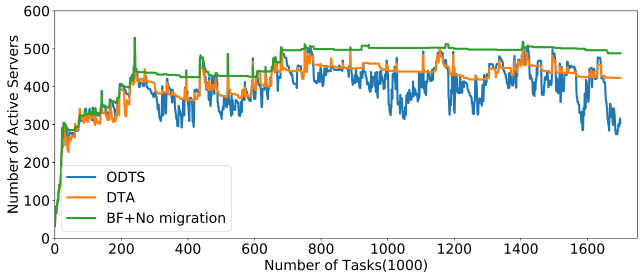
We set the parameters according to the settings in [17], [6] and [15], but without much tuning, which also shows the effectiveness of the proposed optimizations. In DQN construction, a fully-connected hidden layer with 500 hyperbolic tangent function (Tanh) neurons is used to build Global State Encoder. The output network contains two fully-connected layers with 30 and 10 Tanhs and a fully-connected linear layer with a single output for each valid action in the server index. The Q-learning discount factor γ is set to 0.9 in simulations. In DQN updating, the widely-used optimizer RMSprop [51] is applied to increase the training process. For the learning rate and exploration threshold, α, ϵ and ϵ_m are set to 0.01, 0.05 and 0.05, respectively. Update intervals P and p_m are 600 and 9.

In this work, different from previous DRL-based resource allocation methods [17] [6], ODTS does not perform offline training, which means it is more feasible in a practical data center.

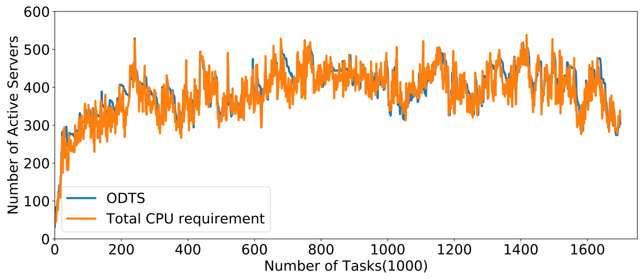
B. Comparison with Baselines

In this subsection, the performance of ODTS is compared against the following baselines.

1) DRL-based Task Assignment only (DTA). It is an independent component of ODTS.
2) DRL-based Task Migration only (DTM). It is also an independent component of ODTS.
3) Best Fit (BF). For task scheduling, the runtime of tasks that is required by sophisticated task scheduling approaches is unavailable in advance. BF assigns a task to a server with the least available resource among the servers which can accommodate the task.
4) Median Absolute Deviation (MAD) [52]. MAD is the baseline for task migration. MAD is a heuristic VM migration algorithm based on the detection and resource allocation of under-utilization and over-utilization of the servers. In the MAD algorithm, there are two thresholds: th_{over} and th_{over}. Some of the containers on over-utilized servers, whose utilization is higher than th_{over}, are migrated to other servers to decrease the number of over-utilized servers. All the containers on under-utilized servers, whose utilization is lower than th_{under}, are migrated to other servers to increase the number of empty servers. The MAD algorithm predicted the th_{over} and th_{under} by measuring the statistical dispersion of each task.



(a) Number of active servers.



(b) Compare active server number of ODTS with the total CPU requirements.

Fig. 5. The number of active servers varies during the experiment.

5) Round-Robin (RR). RR is a baseline for choosing all servers equally in some rational order.

Fig. 4 shows the experimental results when the CPU demand of tasks is set to 1, 2, and 3 times the original in the traces. Fig. 4(a) and Fig. 4(b) demonstrate the average SLAV and the energy usage, respectively. Firstly, all algorithms except for RR have similar SLAV, which means they nearly achieve the same QoS. RR schedules tasks evenly to each server, so it activates more servers and has the highest QoS, which results in the highest energy consumption and lowest SLAV. Compared with BF, DTA saves 7% of energy. The improvement shows that, even with limited information about tasks (i.e., without task duration), better task allocation still achieves higher long-term resource utilization. Similarly, with the same baseline assignment algorithm and the same maximum number of task migrations, DTM consumes 6% energy less than MAD, which proves the effectiveness of task migration based on DRL. ODTS has the best energy efficiency and saves 14% of energy compared with the BF baseline. Besides, with DTA, DTM saves 3% energy, whose improvement is less than with the BF. It may be because that DTA already achieves a good allocation of the tasks.

The number of active servers varies over time, as shown in Fig. 5. To make Fig. 5(a) clearer, we select two representative baselines for comparison. According to Eq. (12), with the same SLAV, the data center should activate fewer servers to reduce energy consumption. ODTS activates the fewest servers during all simulation time, which shows the robustness of our task scheduling algorithm. Moreover, without migration, DTA still leaves a part of servers in low resource utilization, which can be avoided by applying DTM. In Fig. 5(b), scheduled by ODTS, the number of active servers is tightly close to the total CPU requirement. It means that ODTS almost achieves the optimal allocation over all simulation time. Besides, due to the definition of the SLAV in Eq. (4), the sum of the containers' CPU requirements in one server at some time points can exceed the CPU capacity of the server.

TABLE III
EXPERIMENTAL RESULTS OF DIFFERENT TASK ASSIGNMENT ALGORITHMS.

Task Assignment Algorithms	Energy (KWh)	SLAV
RR	35731.9	**0.021**
BF	24888.1	0.100
BDTA	30702.7	0.092
DTA	**22918.2**	0.101

We also record the running time of ODTS. Given the simple and shared neural network structure, we execute ODTS on the CPU, which saves the data communication time between the CPU and the GPU. The average running time of making an assignment decision of ODTS is shorter than 3 ms. It takes about 40 ms to make a migration decision while the migration is infrequent (i.e., the epoch is set to 30 minutes). Compared with the task duration, the running time of ODTS is affordable.

C. Evaluation of Algorithm Optimization

In this subsection, to evaluate the improvements applied in the proposed algorithm, four task assignment algorithms are compared: RR, BF, Basic DRL-based task assignment algorithm (BDTA), and DTA. The difference between BDTA and DTA is that the multiple optimizations are removed from the proposed DRL algorithm, including sort module, transition generation, and efficient exploration. The experiments are repeated ten times to calculate the average performance of each algorithm.

The CPU demand of tasks is set to the original in the traces, and the experimental results are shown in Table III. RR still has the lowest SLAV for its even allocation of tasks, and the other three algorithms have similar SLAV. From the perspective of energy consumption, the usage of BDTA is more than BF and DTA. More specifically, the energy usage of BDTA among experiments has a significant variance, which means that the training process is not stable without our improvements. The reason for the unstable training is that without sufficient offline training, the DQN largely depends on the quality of training samples and the structure of DQN. However, inefficient exploration and sparse reward for selecting servers in sleep mode make the low quality of training samples. Additionally, without the sort module, the DQN cannot recognize different permutations of each server's state, which means that it needs more training data to achieve similar performance.

D. Trade-off between Energy Consumption and QoS

In this section, an experiment is designed to explore the SLAV and energy consumption trade-off curve for the proposed algorithm and two baselines, as shown in Fig. 6.

Different SLAV constraint parameter υ is set to control the energy consumption. For ODTS and two baselines, the curve is nearly linear, which indicates that these algorithms are

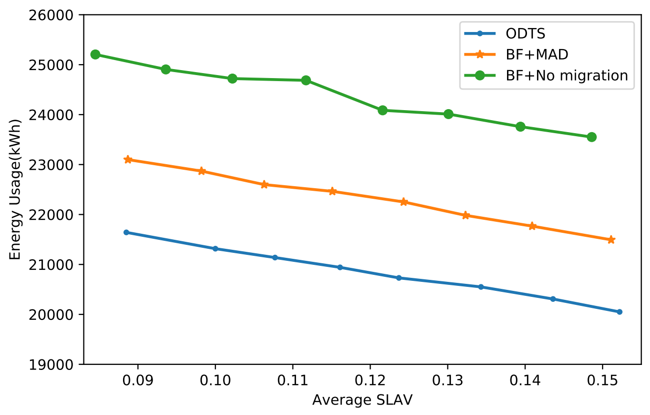


Fig. 6. Trade-off curves between SLAV and energy consumption of baselines and the proposed task scheduling algorithm with original CPU demand and task number = 2700000.

stable under different SLAV constraint parameters. Moreover, it shows that the training of ODTS is stable and controllable.

Furthermore, compared with the two baselines, ODTS achieves substantial improvement for all average SLAV. It can be observed that the three curves have close slopes. According to Eq. (12), it proves that the proposed algorithm better schedules tasks to reduce the number of active servers.

VI. Conclusion

In this paper, an online DRL-based algorithm for energy-efficient task scheduling is proposed. We first introduce the DRL technique and model the task scheduling system, whose optimization target consists of energy consumption and QoS. Then, to cope with the challenge of saving energy during the entire lifecycle of tasks, the DRL-based task scheduling algorithm consists of both task assignment and task migration. To achieve efficient online training, we optimize the DNN structure, reformulate the reward, adopt efficient action selection, and generate training data. Experiments with real-world data trace have shown that our algorithm substantially reduces energy consumption as compared with the existing baselines.

Acknowledgments

This work was supported in part by the Guangdong Key Lab of AI and Multi-modal Data Processing, United International College (UIC), Zhuhai under Grant 2020KSYS007 sponsored by Guangdong Provincial Department of Education; in part by the Chinese National Research Fund (NSFC) under Grants 62272050, 61872239; in part by Institute of Artificial Intelligence and Future Networks (BNU-Zhuhai) and Engineering Center of AI and Future Education, Guangdong Provincial Department of Science and Technology, China; Zhuhai Science-Tech Innovation Bureau under Grants ZH22017001210119PWC and 28712217900001, and in part by the Interdisciplinary Intelligence SuperComputer Center of Beijing Normal University (Zhuhai).

References

[1] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 123–134.

[2] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 510–519.

[3] T. C. Ferreto, M. A. Netto, R. N. Calheiros, and C. A. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1027–1034, 2011.

[4] C.-C. Lin, P. Liu, and J.-J. Wu, "Energy-aware virtual machine dynamic provision and scheduling for cloud computing," in *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 2011, pp. 736–737.

[5] D. G. d. Lago, E. R. Madeira, and L. F. Bittencourt, "Power-aware virtual machine scheduling on clouds using active cooling control and dvfs," in *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 2011, p. 2.

[6] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 372–382.

[7] K. X. Kang, D. Ding, H. M. Xie, Q. Yin, and J. Zeng, "Adaptive drl-based task scheduling for energy-efficient cloud computing," *IEEE Transactions on Network and Service Management*, 2021.

[8] Y. Jiang, M. Kodialam, T. Lakshman, S. Mukherjee, and L. Tassiulas, "Resource allocation in data centers using fast reinforcement learning algorithms," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4576–4588, 2021.

[9] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. Lau, "Dynamic virtual machine management via approximate markov decision process," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[10] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application performance management in virtualized server environments," in *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*. IEEE, 2006, pp. 373–381.

[11] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, 2015, pp. 499–510.

[12] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1180–1194, 2019.

[13] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 11.

[14] Z. Han, H. Tan, S. H.-C. Jiang, X. Fu, W. Cao, and F. C. Lau, "Scheduling placement-sensitive bsp jobs with inaccurate execution time estimation," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1053–1062.

[15] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.

[16] Y. Ye, X. Ren, J. Wang, L. Xu, W. Guo, W. Huang, and W. Tian, "A new approach for resource scheduling with deep reinforcement learning," *arXiv preprint arXiv:1806.08122*, 2018.

[17] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, 2018.

[18] L. Abualigah and M. Alkhrabsheh, "Amended hybrid multi-verse optimizer with genetic algorithm for solving task scheduling problem in cloud computing," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 740–765, 2022.

[19] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, p. 920, 2022.

[20] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 2007, pp. 119–128.

[21] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Release-time aware vm placement," in *2014 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2014, pp. 122–126.
[22] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of network and computer applications*, vol. 52, pp. 11–25, 2015.
[23] Y. Huang, H. Xu, H. Gao, X. Ma, and W. Hussain, "Ssur: An approach to optimizing virtual machine allocation strategy based on user requirements for cloud data center," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 2, pp. 670–681, 2021.
[24] H. Jin, D. Pan, J. Xu, and N. Pissinou, "Efficient vm placement with multiple deterministic and stochastic resources in data centers," in *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 2505–2510.
[25] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2014, pp. 500–507.
[26] F. P. Tso, K. Oikonomou, E. Kavvadia, and D. P. Pezaros, "Scalable traffic-aware virtual machine management for cloud data centers," in *2014 IEEE 34th International Conference on Distributed Computing Systems*. IEEE, 2014, pp. 238–247.
[27] T. Yang, H. Jiang, Y. Hou, and Y. Geng, "Carbon management of multi-datacenter based on spatio-temporal task migration," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021.
[28] M. Zakarya, "Energy, performance and cost efficient datacenters: A survey," *Renewable and Sustainable Energy Reviews*, vol. 94, pp. 363–385, 2018.
[29] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.
[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[32] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 197–210.
[33] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 191–205.
[34] J. Ho, J. Gupta, and S. Ermon, "Model-free imitation learning with policy optimization," in *International Conference on Machine Learning*, 2016, pp. 2760–2769.
[35] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
[36] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.
[37] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Scalable virtual machine migration using reinforcement learning," *Journal of Grid Computing*, vol. 20, no. 2, pp. 1–26, 2022.
[38] Y. Peng, L. Liu, Y. Zhou, J. Shi, and J. Li, "Deep reinforcement learning-based dynamic service migration in vehicular networks," in *2019 IEEE Global communications conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.
[39] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[40] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.
[41] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in neural information processing systems*, 1996, pp. 1038–1044.
[42] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao, "Efficient queue management for cluster scheduling," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 36.
[43] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger, "3sigma: distribution-based cluster scheduling for runtime uncertainty," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 2.
[44] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.
[45] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," *arXiv preprint arXiv:1006.0308*, 2010.
[46] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, pp. 1–14, 2011.
[47] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 265–278.
[48] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 2017, pp. 1–6.
[49] L. Versluis, R. Mathá, S. Talluri, T. Hegeman, R. Prodan, E. Deelman, and A. Iosup, "The workflow trace archive: Open-access data from public and private computing infrastructures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2170–2184, 2020.
[50] Z. Tang, F. Zhang, X. Zhou, W. Jia, and W. Zhao, "Pricing model for dynamic resource overbooking in edge computing," *IEEE Transactions on Cloud Computing*, 2022.
[51] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.
[52] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

Jiong Lou received the B.S. degree from Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2016 and is currently a Ph.D. candidate in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include edge computing, resource allocation, and reinforcement learning.

Zhiqing Tang received the B.S. degree from School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and is currently a Ph.D. candidate in Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include edge computing, resource allocation, and reinforcement learning.

Weijia Jia is currently a Chair Professor, Director of BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (Zhuhai) and VP for Research of BNU-HKBU United International College (UIC) and has been the Zhiyuan Chair Professor of Shanghai Jiao Tong University, China. He was the Chair Professor and the Deputy Director of State Kay Laboratory of Internet of Things for Smart City at the University of Macau. He received BSc/MSc from Center South University, China in 82/84 and Master of Applied Sci./PhD from Polytechnic Faculty of Mons, Belgium in 92/93, respectively, all in computer science. For 93-95, he joined German National Research Center for Information Science (GMD) in Bonn (St. Augustine) as a research fellow. From 95-13, he worked in City University of Hong Kong as a professor. His contributions have been recognized as optimal network routing and deployment; anycast and QoS routing, sensors networking, AI (knowledge relation extractions; NLP etc.) and edge computing. He has over 600 publications in the prestige international journalsconferences and research books and book chapters. He has received the best product awards from the International Science & Tech. Expo (Shenzhen) in 20112012 and the 1st Prize of Scientific Research Awards from the Ministry of Education of China in 2017 (list 2). He has served as area editor for various prestige international journals, chair and PC memberskeynote speaker for many top international conferences. He is the Fellow of IEEE and the Distinguished Member of CCF.