Representation and Reinforcement Learning for Task Scheduling in Edge Computing

Zhiqing Tang, Weijia Jia, *Fellow, IEEE*, Xiaojie Zhou, Wenmian Yang, and Yongjian You

Abstract—Recently, many deep reinforcement learning (DRL)-based task scheduling algorithms have been widely used in edge computing (EC) to reduce energy consumption. Unlike the existing algorithms considering fixed and fewer edge nodes (servers) and tasks, in this article, a representation model with a DRL based algorithm is proposed to adapt the dynamic change of nodes and tasks and solve the dimensional disaster in DRL caused by a massive scale. Specifically, 1) we apply the representation learning models to describe the different nodes and tasks in EC, i.e., nodes and tasks are mapped to corresponding vector sub-spaces to reduce the dimensions and store the vector space efficiently. 2) With the space after dimensionality reduction, a DRL-based algorithm is employed to learn the vector representations of nodes and tasks and make scheduling decisions. 3) The experiments are conducted with the real-world data set, and the results show that the proposed representation model with DRL-based algorithm outperforms the baselines 18.04 and 9.94 percent on average regarding energy consumption and service level agreement violation (SLAV), respectively.

Index Terms—Edge computing, task scheduling, representation learning, reinforcement learning

1 Introduction

In recent years, with the rapid growth in the number of Internet of things (IoT) devices, cloud computing, which is relatively far apart from these devices, cannot meet applications that have strict requirements for delay or mobility, such as real-time video streaming and real-time object detection [1], [2]. To compensate for these weaknesses in cloud computing, edge computing (EC) [3] paradigm currently can play important roles. In EC, tasks are generated from mobile users and offloaded to edge nodes (edge servers) to be processed [4]. Since nodes are deployed in the edge of networks with limited computation capacity, how to schedule the tasks efficiently has drawn wide attention [5]. Through efficient task scheduling, significant savings in energy consumption and improvement in service level agreement violations (SLAV) can be achieved in EC. To design a better scheduling strategy, the researchers have proposed optimization methods based on greedy [6], dynamic programming [7], linear programming [8], etc., and made some improvements. Recently, with the development of deep learning, deep reinforcement learning (DRL)-based optimization methods have been paid more attention [9], [10]. Compared with traditional optimization algorithms, DRL-based algorithms can get a more efficient scheduling strategy [11], [12]. However, due to the complex EC environments in real-world, DRL-based algorithms still face two major challenges.

First, in real-world EC scenarios, some nodes can be dynamically turned off (or turned on) to save energy (or meet the computation requirements). Moreover, the number of pending and running tasks is also dynamically changed. However, existing DRL-based algorithms cannot handle the dynamic increase or decrease of nodes and tasks efficiently [13]. Therefore, how to effectively represent the dynamic environment to adapt to the dynamic changes of nodes and tasks and further improve the scalability of the algorithm is a major challenge. Second, current DRL algorithms assume the environment with relatively small state sets and action sets, while real EC scenarios have a large number of heterogeneous nodes and tasks. This requires processing a large amount of data. Moreover, each node or task has its properties, e.g., nodes with resource capacities, etc., and tasks with resource request, allocation, etc. Those make it difficult to store all the information directly, and further lead to dimensional disasters. Therefore, how to store the entire environment information effectively, save computing resources, and make fast decisions is another major challenge.

To solve the above challenges, data compression and dimensionality reduction techniques are necessary. By compressing and mapping a high-dimensional state space to a low-dimensional vector space, and at the same time as little as possible affecting the relations between the original data, the state of the environment can be represented effectively and the problem of dimensional disasters can be solved. In this paper, the representation learning is introduced and improved to dynamically represent the EC environment and reduce the state dimension of DRL. In representation

- *Z. Tang and W. Yang are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China, and also with the State Key Lab of IoT for Smart City, University of Macau, SAR Macau 999078, China. E-mail: {domain, sdq11111}@sjtu.edu.cn.*
- *W. Jia is with BNU-UIC Joint AI Resrach Institute, Beijing Normal University & UIC (Zhuhai), Guangdong 519087, China, and also with State Key Lab of IoT for Smart City, University of Macau, SAR Macau 999078, China. E-mail: jiawj@um.edu.mo.*
- *X. Zhou and Y. You are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {szxjzhou, youyongjian}@sjtu.edu.cn.*

Manuscript received 4 Sept. 2019; revised 20 Apr. 2020; accepted 21 Apr. 2020. Date of publication 27 Apr. 2020; date of current version 13 May 2022.
(Corresponding author: Weijia Jia.)
Recommended for acceptance by J. Liu.
Digital Object Identifier no. 10.1109/TBDATA.2020.2990558

2332-7790 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

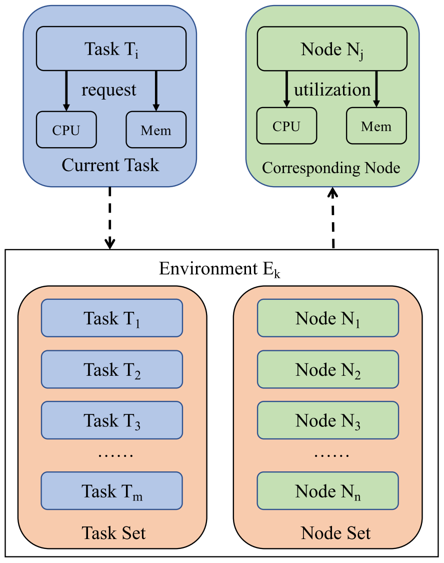


Fig. 1. Two-level relation between tasks and nodes.

learning, the entities and relations are the main components, which are similar as vertices and edges in graphs [14]. Moreover, relations are generally interpreted as translations operating on the low-dimensional representations of the entities (called TransE) [15].

Before employing the representation learning, first the entities and relations in EC are defined with two levels (as shown in Fig. 1, more details see Section 3.2). The tasks and nodes themselves without any properties are denoted as the first-level entities. As for the first-level entities, the relations between tasks and their properties are defined as resource requirements, and the relations between nodes and their properties are defined as resource utilization. Then, the tasks and nodes with corresponding properties, and the environment are defined as the second-level entities, where the environment includes the set of tasks and nodes. The relation between the second-level entities is defined as the scheduling decision, i.e., the relation is extracted to get the corresponding node when a task comes into a specific environment. With representation learning, new arriving tasks and devices in EC and the communications between devices can also be easily defined different entities and relations, which makes the algorithm very scalable. The entities and relations then can be represented with vectors and distances between vectors, respectively.

A novel representation model TransEC is proposed with the entities and relations defined above, which maps the tasks, nodes, and environment of EC to different vector subspace. Specifically, embedding layers are first used to embed the tasks and nodes to different sub-spaces. Then, linear transformations and convolution neural networks (CNNs) [16] are used to extract the potential features of the environment and map the extracted knowledge to another sub-space. Finally, the relations are extracted among second-level entities with our TransEC model.

With the help of TransEC model to reduce the state dimension efficiently, scheduling decisions in large-scale and dynamic EC environment can be solved by DRL. Among different DRL algorithms, the Deep Q-Learning (DQL) algorithm is chosen since it is suitable for fast decision-making [17]. In this paper, a novel TransEC-DQL algorithm is proposed, which can train the TransEC model and make the scheduling decision efficiently. Moreover, the training frequency of the TransEC-DQL algorithm are modified to take not only the state knowledge of the current time slot, which includes the distributions of resource situation of tasks and nodes, but also the state knowledge around this time slot into consideration. And the double Q-learning is adopted to further improving the performance of DQL [18].

To sum up, in this paper, first the entities and relations are defined. Then, a novel TransEC model is proposed to represent the tasks, nodes, environment, and their relations into a vector space. Third, to learn the representations and make task scheduling decisions, TransEC-DQL algorithm is designed and implemented. Our contributions are summarized as follows:

1) We apply the representation learning to describe the dynamic and large-scale nodes and tasks in EC. Moreover, a novel TransEC model is proposed to map nodes and tasks to corresponding vector sub-spaces to reduce the dimensions and store the vector space efficiently.
2) With the space after dimensionality reduction, the TransEC-DQL algorithm is proposed to learn the vector representations of nodes and tasks and make scheduling decisions. Furthermore, the state of the algorithm is modified to a resource level based distribution for better scalability.
3) The experiments are conducted with real-world data set. The experimental results show that the performance of our algorithm is better than baselines 18.04 and 9.94 percent on average regarding energy consumption and SLAV, respectively.

The remainder of this paper is organized as follows. In Section 2, the related work of representation learning and DRL is introduced. Then, the system model and TransEC model are proposed in Sections 3 and 2.1, respectively. Finally, the DQL based task scheduling algorithm and the experimental results are shown in Sections 5 and 6, respectively.

2 Related Work

In this section, the related work of representation learning models and DRL algorithms are introduced in Sections 2.1 and 2.2, respectively.

2.1 Representation Learning Models

Representation learning is usually used in knowledge graph (KG) to represent the entities and relations of a KG into continuous vector spaces to reduce the dimension and simplify the manipulation [19]. Considering a graph, entities with different types and attributes are the vertices in the graph, while different types of relations are the edges of the graph [14]. Usually, they are represented as triple facts $(head\_entity, relation, tail\_entity)$, also denoted as (h, r, t), indicating the relation between two entities [20].

With entity and relation representations, we can further complete KG, extract relations, etc. Most of the currently proposed methods perform the representation tasks based on the observed facts. Given a KG, such a method first represents

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

entities and relations in a continuous vector space and defines a scoring function on each fact to measure its plausibility. Then entity and relation representations can be obtained by maximizing the total plausibility of observed facts [19].

Among different representation learning models, TransE [15] is the most representative translational distance model, which represents both entities and relations as vectors in the same vector space $\mathbb{R}^d$. In TransE, the relation is interpreted as a translation vector r to connect the embedded entities h and t by r with the least error, i.e., $h + r \approx t$ when the fact (h, r, t) holds. The scoring function is defined as the distance between $h + r$ and t and expected to be small if (h, r, t) holds.

Despite its simplicity and efficiency, TransE has flaws in dealing with 1-to-N, N-to-1, and N-to-N relations [14], [21]. To overcome these disadvantages, TransH [21] follows the general idea by introducing relation-specific hyperplanes, i.e., the entity representations h and t are projected onto a hyperplane before calculating the scoring function. TransR [14] shares a very similar idea with TransH, but it introduces relation-specific spaces instead of hyperplanes. Furthermore, TransD [22] simplifies TransR by further decomposing the projection matrix into a product of two vectors. TranSparse [23] is another work that simplifies TransR by enforcing sparseness on the projection matrix.

2.2 Deep Reinforcement Learning Algorithms

Under uncertain and stochastic environments, most of the decision-making problems can be modeled as a Markov Decision Process (MDP) [24] with large-scale and complicated state space and action space. Compared with traditional machine learning methods, reinforcement learning can make decisions efficiently, which is suitable for the area of communications and networking [25]. To solve the problem of the dimensional disaster of large-scale MDP, DRL is adopted [10]. Liu, *et. al.* [13] propose a novel hierarchical framework in the cloud data center to achieve the trade-off between energy consumption and processing delay with DRL technology.

Among different DRL algorithms, the DQL algorithm can find a policy with complicated system models. Generally, the DQL algorithm is used along with some improvements, like particle swarm optimization [26], double Q-learning [18], and dueling network [27], etc. He *et. al.* [28] use DQL to improve the scheduling performance of Vehicular Ad-hoc NETworks. Ferreira *et.al.* [29] show that the DQL can be used for the rate control to achieve multiple objectives in complex communication systems. Tang *et. al.* [30] model the container migration decision problem as a multidimensional MDP, and solve it by DQL. Besides, He *et. al.* [31] propose a DQL framework for energy-efficient resource allocation in green wireless networks.

In this paper, to effectively extract the relations of EC and reduce the dimension, the novel TransEC model and the TransEC-DQL algorithm are proposed and introduced in the next sections.

3 System Model

In this section, the system model is introduced. First, the entities of tasks, nodes, and environment and the relations between entities are described in Section 3.1. Then, the task scheduling problem is formulated in Section 3.2.

3.1 Entities and Relations

The main components of EC are the tasks and nodes, which are also the key components for task scheduling. Based on these components, the entities and relations of EC are defined for representation learning [15].

First, the main components of EC are defined as two types of entities: tasks and nodes. The tasks are generated from users, and sent to nodes to be processed. While the nodes are belonging to EC and provide computation resource. Let $\mathbf{T} = \{T_1, T_2, \ldots, T_m\}$ and $\mathbf{N} = \{N_1, N_2, \ldots, N_n\}$ denote the sets of tasks and nodes, respectively. Generally, the time is divided into time slots according to scheduling period [30]. During time slot t_k, each task T_i has two properties: CPU request and memory request, denoted as $T_i.c_k$, $T_i.m_k$, respectively. In addition, for task T_i, its arrival time and leave time are denoted as $T_i.t_0$ and $T_i.t_1$, respectively. Meanwhile, each node N_j has two properties: CPU capacity $N_j.c_0$ and memory capacity $N_j.m_0$. And $N_j.c_k$ and $N_j.m_k$ are denoted as the CPU and memory usage during t_k, respectively.

After defining the entities of EC, the two-level relations between the entities are defined. For the first-level relation, during time slot t_k, task T_i is requesting a certain amount of CPU resource $T_i.c_k$ and memory resource $T_i.m_k$. The relations are denoted as $request$ between task T_i and the requested resource amount, i.e., $(T_i, request, T_i.c_k)$ and $(T_i, request, T_i.m_k)$ during t_k. Besides, $utilization$ is denoted as the relation between node and its remaining resource capacity, e.g., during t_k, the CPU and memory utilization of node N_j is denoted as $(N_j, utilization, N_j.c_0 - N_j.c_k)$ and $(N_j, utilization, N_j.m_0 - N_j.m_k)$, respectively.

Based on the first-level relation, the incoming task and the corresponding scheduled target node with features are regarded as entities of second-level relation. Besides, another second-level entity, denoted as environment, is defined as the combination of the set of tasks arrived during t_k and the set of the utilization levels of nodes during t_k. The environment during t_k is denoted as $E_k = \{(T_i.c_k, T_i.m_k) | T_i.t_0 = t_k \cap T_i \in \mathbf{T}\} \cup \{(N_j.c_0 - N_j.c_k, N_j.m_0 - N_j.m_k) | N_j \in \mathbf{N}\} \in \mathbf{E}$, where $\mathbf{E}$ is the set of environment. E_k is regarded as the relation between the task and node, i.e., the triplet is (T_i, E_k, N_j). In other words, as denoted in Fig. 1, if one task T_i (entity A) with the environment E_k (relation) is known, then the corresponding node N_j (entity B) of this task can be computed. The first-level relations ($request$ and $utilization$) are also illustrated in Fig. 1.

Generally, only the two most basic components (tasks and nodes) of EC are considered. However, other devices of EC can also be easily considered if needed. As long as the devices have features like computation resources (including CPU, memory, etc.) and have relations between each other, they can be defined as different entities and relations. With the definitions of entities and relations, the problem is formulated in the next subsection.

3.2 Problem Formulation

In many EC scenarios, typically like wireless systems and IoT, there are many heterogeneous devices. In addition to the nodes with powerful computation resources, there are also many nodes with limited computation resources, which are typically battery constrained devices (including Raspberry Pi, small development board, etc.). For these

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

nodes, energy consumption must be considered carefully. Besides, energy consumption can reflect the resource usage of all nodes, thus reflecting the quality of task scheduling. A good scheduling strategy can reduce the energy consumption significantly.Furthermore, the quality of service (QoS) needs to be optimized to meet the requirements of tasks from mobile users, which can be denoted as SLAV [32]. To make better use of resources and improve the QoS of users, we aim to minimize the energy consumption and SLAV of EC with resource constraints. CPU and memory are considered as two main resources. The CPU resource can be overbooked [33]. If the CPU resource of one node is overbooked, there will exist a potential SLAV. Besides, for each time slot t_k, the memory resource allocated should not exceed the memory resource capacity of this node, which can be denoted as $\sum_{T_i.l_k=N_j} T_i.m_k \leq N_j.m_0$, where $T_i.l_k$ is the location of task T_i during t_k.

Energy Consumption. The total energy consumption P is equivalent to the addition of each node's energy consumption. If node N_j is switched off or set in sleep mode, its energy consumption $P_j \approx 0$ [34]. Otherwise, its energy consumption is proportional to resource utilization $U_j(t_k)$ as indicated in [35]

$$P = \int_{k=1}^{|t|} P_k, P_k = \left(\sum\nolimits_{j=1}^{n} \left(P_{idle} + (P_{max} - P_{idle}) \times U_j(t_k)\right)\right),$$

where P_{idle} and P_{max} represent the energy consumption of 0 and 100 percent CPU utilization of N_j, respectively. $|t|$ is the number of time slots. P_k is the energy consumption during t_k. Besides, $U_j(t_k)$ is the resource utilization of N_j during t_k, which is defined as

$$U_j(t_k) = \min\left\{1, \frac{\sum_{1\{T_i.l_k=N_j\}} T_i.c_k}{N_j.c_0}\right\},$$

where $1\{\cdot\}$ is Iverson bracket, which is equivalent to 1 when the condition is satisfied. Otherwise, it is equivalent to 0.

SLAV. SLAV is used to describe the degree of resource shortage of nodes, which is defined as [36]

$$SLAV = \int_{k=1}^{|t|} SLAV_k SLAV_k = \frac{\sum_{i=1}^{m}(T_i.c_k - T_i.a_k)}{\sum_{i=1}^{m} T_i.c_k},$$

where $T_i.a_k$ is the CPU resource allocated to task T_i during t_k.

Our algorithm aims to reduce the cost $\mathcal{C} = P + \beta \times SLAV$ which is consisted of energy consumption P and SLAV $SLAV$, where β is the parameter controlling the weight of $SLAV$. The target is to find the strategy to minimize $\mathcal{C}$. Meanwhile, for each N_j, the resource constraint $\sum_{T_i.l_k=N_j} T_i.m_k \leq N_j.m_0$ should be obeyed. Therefore, the task scheduling problem of EC is defined as follows:

Problem 1:

$$\begin{aligned} &\min \mathcal{C} = P + \beta \times SLAV \\ &s.t. \sum_{T_i.l_k=N_j} T_i.m_k \leq N_j.m_0, \end{aligned} \tag{1}$$

Problem 1 is an advanced bin-packing problem, which is NP-hard and can only be solved heuristically. However, most of the existing heuristic algorithms are unstable which cannot solve the large-scale problem in the real-world environment. In this problem, P and $SLAV$ can be represented as the addition of P_k and $SLAV_k$ during each time slot k. In addition, the first-order transition probability of the users' resource demands is also quasi-static for a long period and non-uniformly distribution by properly choosing the time slot duration [34], which is a sequential decision-making process, and has memoryless property [37]. Therefore, this problem can be modeled as a MDP.

To solve this problem, DQL based algorithms are adopted. Besides, in order to solve the dimensional disaster problem in DQL and improve the robustness of the algorithm, the state needs to be represented in a form independent of the numbers of nodes or tasks. And representation learning models are used to reduce the dimension of the state space of DRL efficiently. The TransEC model and the TransEC-DQL algorithm are introduced in Sections 2.1 and 5, respectively.

4 System Representation

In this section, the representation method based on the definitions of entities and relations is introduced in Section 4.1. Then, the TransEC model for relation extraction is proposed in Section 4.2.

4.1 Representations for Entities and Relations

The tasks and nodes are represented in a vector space with different sub-spaces, and the representation flow is shown in Fig. 2. First, the task T_i with features like CPU $T_i.c$ and memory $T_i.m$ is represented as a vector V_{T_i}, and the environment with task and node distribution is represented as a vector V_{E_k}. Meanwhile, the node N_j is also represented as a vector V_{N_j}. Then, if $V_{T_i} \times V_{E_k} = V_{N_j}$, the node N_j is the selected node. The details are as follows.

During time slot t_k, for task T_i, $T_i.c_k$ and $T_i.m_k$ are represented separately. First, the CPU and memory resource are divided into $X+1$ levels, i.e., the level of $T_i.c_k$ is denoted as $L(T_i.c_k) \in [0, X]$. Then, an embedding layer L_{emb} is used to map the requested CPU value to a Y-dimension vector space denoted as $V_{T_i.c_k}$. The process of the embedding layer L_{emb} is defined as

$$V_{T_i.c_k} = tanh(W_{emb} \times L(T_i.c_k)),$$

where $W_{emb} \in \mathbb{R}^{X \times Y}$ is a matrix, $tanh(\cdot)$ is the activation function, which is widely adopted in representation learning algorithms, like TransE [15], TransR [14], etc., and is denoted as

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

After getting the representation of $T_i.c_k$, the same representation process is used to map $T_i.m_k$ to $V_{T_i.m_k}$. Then we concat these two vectors and input the connected vector to a linear layer L_{fc}, which can be denoted as

$$V_{T_i} = tanh(W_{fc} \times (V_{T_i.c_k} \oplus V_{T_i.m_k}) + b_{fc}),$$

where $W_{fc} \in \mathbb{R}^{2Y \times Y}$ and $b_{fc} \in \mathbb{R}^{2Y}$ are the linear transformation matrix and bias matrix of L_{fc}, respectively. $\oplus$ is

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

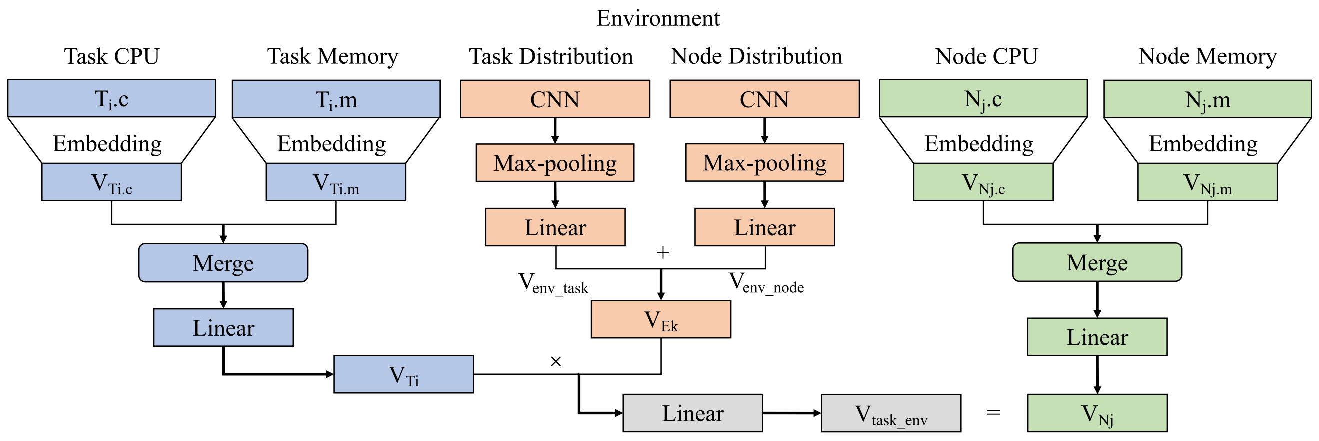


Fig. 2. System representation architecture.

the concatenation operator, e.g., if $V_1 = [1, 2]$, $V_2 = [3, 4]$, then $V_1 \oplus V_2 = [1, 2, 3, 4]$. The representation process of nodes is the same with the representation of tasks but with different weights and bias. The representation of N_j is denoted as V_{N_j}.

Different from the representations of tasks and nodes, the representation of the environment E_k during time slot t_k is more complex, since the task scheduling decisions are made based on the overall knowledge which includes the resource requests, resource availabilities, etc. First, to deal with dynamic number of tasks and nodes with different states, the resource requirements of tasks and the remaining resource capacities of nodes of the environment are divided and transferred to distributions of resource levels. The process is described as follows. 1) Two zero matrices R_{task} and R_{node} are initialized with dimensions of $\mathbb{R}^{X\times X}$, which store the distributions of resource level of task requests and node remaining capacities, respectively. 2) For $R_{task}(i,j)$ in matrix R_{task}, the value is updated during each time slot t_k as follows:

$$R_{task}(i,j) = \sum_{1\{L(T_h.c_k)=i\}\{L(T_h.m_k)=j\}\{T_h.t_0=t_k\}} count(T_h),$$

where $count(T_h)$ is the number of T_h. Similarly, for $R_{node}(i,j)$ in matrix R_{node}, the value $R_{node}(i,j)$ is updated as follows:

$$R_{node}(i,j) = \sum_{1\{L(N_h.c_k)=i\}\{L(N_h.m_k)=j\}} count(N_h).$$

With these distributions, no matter how many nodes and tasks in EC, the same representations of the environment with the same trained model can be easily used for the representation of the global system environment, which provides very good scalability as compared with existing approaches. Then, the distributions are input to a CNN layer L_{cnn} and a max pooling layer L_{pool} to extract the knowledge of the environment. CNN is usually used in image processing to extract the local patterns of images. In this paper, it is used to extract the potential features of the environment in EC. In the CNN layer, a task distribution R_{task} of size $\mathbb{R}^{X\times X}$ is represented as

$$R_{task}^{1:X,1:X} = \begin{bmatrix} R_{task}(1,1) \oplus \cdots \oplus R_{task}(1,X) \\ \cdots \quad \cdots \quad \cdots \\ R_{task}(X,1) \oplus \cdots \oplus R_{task}(X,X) \end{bmatrix}.$$

In general, let $R_{task}^{i:i+j,k:k+l}$ refer to the concatenation of $R_{task}(i,k), \ldots, R_{task}(i,k+l), \ldots,\ R_{task}(i+j,k), \ldots, R_{task}(i+j,k+l)$. Then, the convolution layer involves a filter $W_{cnn} \in \mathbb{R}^{h\times h}$, which is applied to a window of size $h \times h$ to produce a new value. For example, the feature $c_{i,k}$ is generated from a window of $R_{task}^{i:i+h-1,k:k+h-1}$ by

$$c_{i,k} = tanh\Big(W_{cnn} \cdot R_{task}^{i:i+h-1,k:k+h-1} + b\Big),$$

where $b \in \mathbb{R}^h$ is a bias term. Such filter is applied to each possible window of task distribution to produce a feature map

$$\mathbf{c}_{1,X-h+1} = \begin{bmatrix} c_{1,1} & \cdots & c_{1,X-h+1} \\ \vdots & \ddots & \vdots \\ c_{X-h+1,1} & \cdots & c_{X-h+1,X-h+1} \end{bmatrix},$$

where $\mathbf{c}_{1,X-h+1} \in \mathbb{R}^{X-h+1\times X-h+1}$. Next, a max-over-time pooling operation [38] with the max pooling layer L_{pool} is applied over each feature map and the maximum value is taken, e.g., $\hat{\mathbf{c}}_{1,X-h+1} = \max\{\mathbf{c}_{1,X-h+1}\}$, as the feature corresponding to each particular filter. After that, each maximum value $z = [\hat{\mathbf{c}}_{1,1}, \ldots, \hat{\mathbf{c}}_{1,X-h+1}, \ldots, \hat{\mathbf{c}}_{X-h+1,1}, \ldots, \hat{\mathbf{c}}_{X-h+1,X-h+1}]$ is collected and the set z is input to a linear layer $L_{fc_{env}}$ to get the vector of task distribution of environment, which is described as follows:

$$V_{E_k^T} = tanh(W_{fc_{env}} \times z + b_{fc_{env}}),$$

where $W_{fc_{env}} \in \mathbb{R}^{(X+h-1)\times k}$ is the weight of $L_{fc_{env}}$, and $b \in \mathbb{R}^{X+h-1}$ is the bias. Similarly, the vector of the node distribution can be obtained which is denoted as $V_{E_k^N}$.

Finally, the vectors are connected and the final vector of the environment is output, which is calculated as follows:

$$V_{E_k} = tanh\Big(V_{E_k^T} \oplus V_{E_k^N}\Big).$$

Now the vector $V_{T_i}(T_i \in \mathbf{T})$ of the task T_i, the vector $V_{N_j}(N_j \in \mathbf{N})$ of the node N_j, and the vector $V_{E_k}(E_k \in \mathbf{E})$ of the

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

environment E_k are constructed. As illustrated in Section 3.1, there are more heterogeneous nodes and devices in the real-world EC environment, so in order to get a general representation of nodes, tasks, and resources, etc., such complicated mathematical notation is necessary. Based on these vectors, the computation of the corresponding node V_{N_j} according to V_{T_i} and V_{E_k} is further introduced in next sub-section.

4.2 Relation Computation

In this subsection, first, the TransE model [15] is briefly introduced, then the relation computation model TransEC is described.

TransE. Given a training set $\mathbf{G}$ of triplets (h, r, t), the basic idea behind TransE is that $h + r \approx t$ when (h, r, t) holds, while $h + r$ should be far away from t otherwise. Following an energy-based framework [39], the energy of a triplet is equal to $d(h + r, t)$. For some dissimilarity measure d, it is measured using either L_1 or the L_2-norm. The scoring function is then defined as the distance between $h + r$ and t, i.e.,

$$d(h + r, t) = -||h + r - t||_{1/2}.$$

The score is expected to be large if (h, r, t) holds. As a result, the loss function of TransE is the margin ranking loss, which is defined as

$$L_{TransE} = \sum_{(h,r,t)\in\mathbf{G}} \sum_{(h',r,t')\in\mathbf{G}'_{(h,r,t)}} [\gamma + d(h + r, t) - d(h' + r, t')]_+,$$

where $[x]_+$ denotes the positive part of x, $\gamma > 0$ is a margin hyperparameter, and the set $\mathbf{G}'_{(h,r,t)}$ is composed of training triplets with either the head or tail replaced by a random entity, but not both at the same time, which is defined as follows:

$$\mathbf{G}'_{(h,r,t)} = \{(h', r, t)|h' \in \mathbb{E}\} \cup \{(h, r, t')|t' \in \mathbb{E}\}.$$

TransEC. Based on TransE, the relation computation model TransEC is proposed. Compared with TransE, the vector space of EC is divided into different sub-spaces. Some changes are made to better apply to EC and the differences between TransEC and TransE are mainly in two aspects: the scoring function and the loss function. The details are described as follows.

Scoring Function. TransE uses $h + r \approx t$ as the scoring function, which means the relation between two entities corresponds to a translation of the vectors. Here in EC it has the same meaning as $V_{T_i} + V_{E_k} \approx V_{N_j}$. However, compared with TransE, the V_{T_i}, V_{E_k} and V_{N_j} are in different sub-spaces of the vector space in EC, as a result, adding up the head entity V_{T_i} and relation V_{E_k} directly has no meaning. To solve this problem, the vector of the task V_{T_i} is mapped to the sub-vector space of environment by changing the scoring function to $V_{T_i} \times V_{E_k} \approx V_{N_j}$. By changing the operator from $+$ to $\times$, relations among tasks, nodes, and environment can be established in different sub-spaces. Therefore, the scoring function is then defined as

$$d\Big(V_{T_i}, V_{E_k}, V_{N_j}\Big) = -\Big|V_{T_i} \times V_{E_k} - V_{N_j}\Big|.$$

Loss Function. there exists lots of ready-made data sets in NLP field, such as Freebase [40], Wikidata[41], etc. Based on these truthful data sets, TransE can use margin ranking loss function to learn the representations of entities and relations fully and can make enough training episodes to make the different entities far enough from each other. However, in EC, the task scheduling decision is obtained real-time from the dynamic environment, and the true labeled solution cannot be generated in polynomial time due to NP-hard problem nature. As a result, there are not enough training triplets to separate the distance among the representation vectors for the entities. To solve this problem, the loss is changed into

$$L_{TransEC} = \sum_{(T_i,N_j)\in(\mathbf{T},\mathbf{N})} \sum_{N'_j\in\mathbf{N}'_j} [\gamma_{TransEC} + d\Big(V_{T_i}, V_E, V_{N_j}\Big) - d\Big(V_{T_i}, V_E, V_{N'_j}\Big)\Big]_+, \tag{2}$$

where $\mathbf{N}'_j$ is the set of all nodes except N_j, since the task can only be assigned to one node, this can make sure the triplets $(V_{T_i}, V_{E_k}, V_{N_j})$ not hold. $\mathbf{N}'_j$ is generated as

$$\mathbf{N}'_j = \{N_i|N_i \neq N_j, N_i \in \mathbf{N}\}.$$

$\gamma_{TransEC}$ is the margin between correct relation and incorrect relation, which is defined as

$$\gamma_{TransEC} = \max(|N_j.c_k - N'_j.c_k|, |N_j.m_k - N'_j.m_k|).$$

With this loss function, enough distance between the pair of positive relation and negative relation can be pulled. After defining the scoring function and loss function, the TransEC model can be learned as shown in Algorithm 1. The input is the training set $\mathbf{G} = \{(T_i, E_k, N_j)\}$, the set of entities $\mathbf{T}$ and $\mathbf{N}$, the set of relations $\mathbf{E}$, and the dimension of the vector space Y. The output is the weights of network with weights $\theta_{TransEC}$ whose architecture is denoted in Fig. 2. First, T_i, N_j, and E_k are initialized with uniform distribution [15] as shown in line 1 - 7. Then, for each episode, a batch is sampled from $\mathbf{G}$, and $\mathbf{O}_{batch}$ is set as an empty set, as shown in line 8 - 9. After that, the negative triplets (T_i, E_k, N'_j) is generated and all triplets are joined to $\mathbf{O}_{batch}$, as shown in line 10 - 13. Finally, $L_{TransEC}$ is calculated by Eq. (2) and the gradient descent is performed to learning the representations of entities and relations.

However, the TransEC model cannot be trained adequately to get well-represented vectors of entities and relations due to lack of enough labeled data. DQL is a method which does not need labeled data set, and can cope with environmental changes with dynamic decision-making. Besides, TransEC model can solve the dimensional disaster of DQL by vector representations. In the next section, a DQL based task scheduling algorithm is introduced, which modifies and combines TransEC and is suitable for learning TransEC and making scheduling decisions in EC.

5 Task Scheduling Algorithm

In this section, reinforcement learning algorithms are adopted to attain task scheduling decisions. Among different reinforcement learning algorithms, DQL [10], combined with

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

Q-learning and Deep Neural Network (DNN), has the advantage in rapid decision-making, which is suitable for EC. Reinforcement learning is based on the settings of state, action, and reward, which are introduced in Section 5.1. With the help of representation learning, the significant amount of state knowledge for DQL is reduced to a lower dimension, the TransEC-DQL algorithm is proposed in Section 5.2.

Algorithm 1. Learning TransEC

```
Input: $\mathbf{G}, \mathbf{T}, \mathbf{N}, \mathbf{E}, Y$
Output: $\theta_{TransEC}$
 1: Initialize $E \leftarrow$ uniform$(-1,1)$ for each $E \in \mathbf{E}$
 2:          $E \leftarrow E/||E||$ for each $E \in \mathbf{E}$
 3:          $T_i \leftarrow$ uniform$(-1,1)$ for each $T_i \in \mathbf{T}$
 4:          $N_j \leftarrow$ uniform$(-1,1)$ for each $N_j \in \mathbf{N}$
 5: for $k \in 1, |t|$ do
 6:   $T_i \leftarrow T_i/||T_i||$ for each $T_i \in \mathbf{T}$
 7:   $N_j \leftarrow N_j/||N_j||$ for each $N_j \in \mathbf{N}$
 8:   $\mathbf{G}_{batch} \leftarrow sample(\mathbf{G}, batch)$
 9:   $\mathbf{O}_{batch} \leftarrow \emptyset$
10:   for $\{(T_i, E, N_j)\} \in \mathbf{G}_{batch}$ do
11:     $N'_j \leftarrow sample(\mathbf{N}'_j)$
12:     $\mathbf{O}_{batch} \leftarrow \mathbf{O}_{batch} \cup \{((T_i, E, N_j), (T_i, E, N'_j))\}$
13:   end for
14:   Calculate $L_{TransEC}$ by Eq. (2)
15:   $\theta_{TransEC} = argmin_{\theta_{TransEC}} L_{TransEC}$
16: end for
17: end
```

5.1 Reinforcement Learning Settings

In reinforcement learning algorithms, for each time slot t_k, the reinforcement learning agent collects system state S_k, and calculates the reward R_{k-1} during last time slot t_{k-1}. Then, the agent selects action A_k according to pre-defined strategy. After performing the action, the system would transit to the new state S_{k+1} in the next time slot. Similarly, the agent calculates reward R_k and chooses new action A_{k+1} according to pre-defined strategy.

The system state is based on the environment E_k and the current arrival task T_i. Generally, the tasks are handled according to the arrival time, and if there are more than one task arriving at the same time, they are handled one by one [13]. Thus, let $S_k = \{(E_k, T_i)\} \in \mathbb{S}$ denote the system state of task T_i during time slot t_k, where $\mathbb{S} = \{(E_k, T_i) | E_k \in \mathbf{E}, T_i \in \mathbf{T}\}$ is the set of states.

The target is to select a suitable node to place the task, so the action set during t_k is defined as $A_k \in \mathbb{A} = \{N_1, N_2, \dots N_n\}$, where $\mathbb{A}$ is the set of all possible actions.

We focus on minimizing the total cost over time. Compared with traditional reinforcement learning, a training frequency f is set, i.e., for each f time slots, the reward is calculated. The reward R_k is defined as

$$R_k = P_{k-f} - P_k + K \times (SLAV_{k-f} - SLAV_k).$$

Among all kinds of reinforcement learning algorithms, Q-learning algorithm has an advantage in fast computation, which is consistent with the requirement of rapid decision-making in EC. In such algorithms, the quality of each state-action pair (S_k, A_k) is indicated by Q-value $Q(S_k, A_k)$, which is stored in Q-matrix. The Q-matrix is initialized to a zero matrix, and each element in Q-matrix indicates the $Q(S_k, A_k)$ of corresponding state-action pair.

The large size of $\mathbb{S}$ and $\mathbb{A}$ could result in the large size of Q-matrix. To solve the problem, DQL, which combines Q-learning with DNN, can extract the reward information and store it in deep Q-network [17]. Besides, with the TransEC model, the DNN can be further improved with less computation resource. The TransEC-DQL algorithm is proposed, which combines the TransEC model and DQL algorithm. In the TransEC-DQL algorithm, the Q-matrix (or Q-network) is replaced by the vector space. The system state is first represented as vectors of the environment and task, then the corresponding Q-values is calculated according to different state-action pairs. The details are described in the next subsection.

Algorithm 2. TransEC-DQL

```
Input: $\theta, \theta', D, D'$
Output: $A_k$
 1: Initialize $\theta, \theta', D, D'$
 2: for episode = $1, \kappa$ do
 3:   Observe state $S_0$
 4:   for $k = 0, |t|$ do
 5:     Get task $T_i$
 6:     Generate random number $\phi$
 7:     if $\phi > \epsilon$ then
 8:       Select best action according to Eq. (6)
 9:     else
10:       Select action according to Eq. (7)
11:     end if
12:     Observe $S_{k+1}$
13:     Push $(S, A, S')$ to $D'$
14:     if $t \bmod f = 0$ then
15:       Calculate $R_k$
16:       Get history from $D'$
17:       Update $D$ and clear $D'$
18:       Optimize $\theta$ by Algorithm 3.
19:     end if
20:     Output $A_k$
21:   end for
22: end for
23: end
```

5.2 TransEC-DQL Algorithm

In Q-learning algorithm, each $Q(S_{k-1}, A_{k-1})$ can be updated online with the learning rule

$$\begin{aligned} Q(S_{k-1}, A_{k-1}) \leftarrow (1-\alpha) Q(S_{k-1}, A_{k-1}) \\ + \alpha\big[R_{k-1} + \gamma \times \max_{A_k} Q(S_k, A_k)\big], \end{aligned} \tag{3}$$

where α is the learning rate and γ is the discount parameter.

At the beginning of each time slot, the Q-learning agent first observes the task and environment knowledge from the system as the state. After that, it chooses an action according to ϵ-greedy algorithm, which consists of exploration and exploitation [42]. In ϵ-greedy algorithm, a threshold ϵ is set in advance, and a random number ϕ is generated each time. When $\phi > \epsilon$, the action is selected by exploitation. The agent selects the best action $A_k = argmax_{A_k} Q(S_k, A_k)$ according to the Q-values stored in Q-matrix. Otherwise, the action is

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

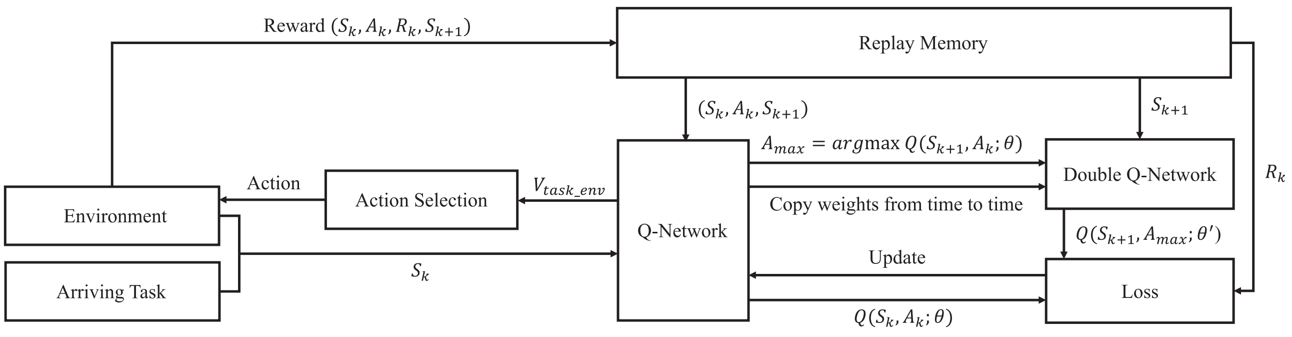


Fig. 3. TransEC-DQL algorithm.

selected by exploration. The agent attempts to get rid of local optimization by selecting a random action.

The environment E_k would be updated according to the action. The agent observing the next state will push the state, the action, and the corresponding reward to the experience replay memory. At the end of each time slot, the agent samples some experience from the experience replay memory and trains the network. Then, it optimizes the network parameters using gradient descent.

To provide more stability when training the network, the target network and policy network are introduced [10]. Target network provides stable $Q(S_k, A_k; \theta')$, where θ' is the weights of the target network. And policy network with weights θ is used for training and parameter update. The weights for the target network are copied from the policy network after every certain training steps. The network architecture of policy network is set the same as the task and environment parts of TransEC model (the layers which output V_{T_i} and V_{E_k} in Fig. 2). The training object of our DQL based algorithm is defined as

$$L(\theta) = E[(y_k - Q(S_k, A_k; \theta))^2],$$

where $Q(S_k, A_k; \theta)$ is the output of policy network, and y_k is defined as

$$y_k = R_k + \gamma \max Q(S_{k+1}, A_{k+1}; \theta'), \tag{4}$$

Furthermore, the max operator $\max Q(S_{k+1}, A_{k+1}; \theta')$ in Eq. (4) uses the same Q-values both to choose and to evaluate an action. This makes it more likely to select overestimated values, which will result in overoptimistic estimates. To solve this problem, two Q-value functions arelearned by assigning experiences randomly to update one of them in Double Q-learning [18]. One function is used to determine the maximizing action, and the other one is used to estimate its Q-value. With these two functions, the action selection is decoupled from the evaluation.

In deep Q-learning algorithm, the target network provides a natural candidate for the second value function. Therefore, the policy network can be used to evaluate the greedy policy, and the target network can be used to estimate its value, as proposed in Double DQN [43]. The target y_k^d is then defined as

$$y_k^d = R_k + \gamma Q(S_{k+1}, arg\ \max_A Q(S_{k+1}, A, \theta); \theta'). \tag{5}$$

The main algorithm flow of TransEC-DQL is shown in Fig. 3. First, the state S_k is obtained from the environment and the arriving task. Then, the Q-network outputs V_{task_env} and an action is selected by the agent. After that, the reward tuple (S_k, A_k, R_k, S_{k+1}) is stored in the replay memory D. Finally, the Q-network and double Q-network output the necessary Q-values for calculating the loss and updating the weights of the network.

The specific TransEC-DQL algorithm is shown in Algorithm 2. First, the weights of policy network and target network are initialized with the replay memory D and temporary memory D'. Replay memory D is used to store the transitions (S_k, A_k, R_k, S_k') for experience replay [17], and temporary memory D' is used to store the temporary transitions consist of (S_k, A_k, S_k').

For each episode, first, an initial state S_0 is observed. Then, for each time slot, the task T_i whose $T_i.t_0 = t_k$ is obtained, and action is selected according to ϵ-greedy algorithm, as shown in lines 5 - 11. The best action selection shown in lines 7 - 9 is according to the output of the target network as follows:

$$A_k = arg\ \max_{A_i}(Q(S_k, A_i; \theta') | A_i \in \mathbf{N}). \tag{6}$$

Otherwise, the action is selected according to the scoring function, which is defined as

$$A_k^d = arg\ \max_{A_i}(d(V_{T_i}, V_{E_k}, V_{A_i}) | A_i \in \mathbf{N}), \tag{7}$$

where V_{A_i} is the vector of node $N_j = A_i$, which is pretrained, e.g., one-hot encoding [44]. By doing so, the node with maximum score is obtained, and further used to train the Q-network.

Algorithm 3. Training of TransEC-DQL

```
Input: $D$
Output: $\theta$
1: Sample $D_k \subset D$
2: for $(S_{k-1}^{(j)}, A_{k-1}^{(j)}, R_{k-1}^{(j)}, S_k^{(j)})$ in $D_k$ do
3:     Calculate $Q(S_k^{(j)}, A_k^{(j)}; \theta')$
4:     Calculate $y_k^d$ by Eq. (5)
5: end for
6: $\theta = arg\min_\theta L_{TransEC}$
7: From time to time reset $\theta' = \theta$
8: Return $\theta$
9: end
```

After that, the next state S_{k+1} is observed, and the transitions (S_k, A_k, S_k') is pushed to the temporary memory D', as

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

shown in lines 12 - 13. In lines 14 - 19, the policy network is trained from time to time, e.g., at the end of every three time slot, and if the current time slot is the training time slot, the reward R_k is calculated, and the history is read from D'. Then, the transitions from D' combined with the same R_k are added to D. By using the same R_k, the agent can take the before and after time slots into consideration. The temporary knowledge in D' is cleared. After that, we do the experience replay and optimize the policy network by Algorithm 3. Finally, the action A_k is output.

In Algorithm 3, first, some transitions are sampled from replay memory D. Then, for each sampled transition, the Q-value $Q(S_k^{(j)}, A_k^{(j)}; \theta')$ and y_k are calculated. Finally, the gradient descent for the policy network is computed once every certain training steps, and the target network is copied from the policy network.

5.3 Convergence and Computational Complexity Analysis

The convergence of the TransEC-DQL algorithm is analyzed as follows. It has been proven that Q-learning will gradually converge to the optimal policy under stationary MDP system and sufficiently small learning rate [45]. Besides, the learning rule given in Eq. (3) converges to the optimal Q-function as long as $\sum_k \alpha_k(S, A) = \infty$ and $\sum_k \alpha_k^2(S, A) \leq \infty$ for all $(S, A) \in \mathbb{S} \times \mathbb{A}$ [46], where $\alpha_k(S, A)$ is the learning rate at T_k. Hence, the DQL-based task scheduling algorithms will converge to the optimal policy when (1) the system evolves as a stationary memoryless MDP, (2) the learning rate is sufficiently small and (3) the DNN is sufficiently accurate to return the action with optimal $Q(S_k, A_k)$ estimate. In this paper, the MDP characteristics have been analyzed in Section 3.2. Besides, the learning rate is sufficiently small $(0 \leq \alpha_k \leq 1)$ in our problem, and the convergence of DQL has been illustrated in [17]. In short, the TransEC-DQL algorithm is convergent, and the experimental results will demonstrate the effectiveness of the algorithm.

For computational complexity, as illustrated in Section 3.1, there are m tasks and n nodes. Let l denote the batch size. The main process of the system consists of two parts: the learning process of TransEC model and the decision-making process of TransEC-DQL algorithm. For the learning process of TransEC in Algorithm 1, the time complexity is $O(m + n + l)$ at time slot t_k. Besides, for Algorithms 2 and 3, the time complexity of action selection, state observation, and reward calculation in Algorithm 2 is $O(n)$, $O(n + m)$ and $O(n + m)$, respectively. And the time complexity for training of TransEC-DQL in Algorithm 3 is $O(l)$. In short, the time complexity of Algorithm 2 is $O(m + n + l)$. To sum up, the algorithms have polynomial-time complexity.

6 Experiments

In this section, the algorithms are evaluated through the following two key points:

1) The influences of different hyper-parameters and parameters settings.
2) The effectiveness of our TransEC model compared against traditional DQL algorithm.

The experimental settings are introduced in Section 6.1, and then the experimental results are reported and analyzed in Section 6.2.

6.1 Experimental Settings

Data Pre-Processing. The data set used in our experiment is chosen from the Google cluster trace [47], [48]. With proper pre-processing, this data trace can be used in virtualized cloud, EC, etc. [49]. The raw trace data contains statistics of about 12.5k servers during 29 days in May 2011, and the size of it is larger than 40 GB. In Google cluster, work arrives at a cell, which is a set of servers, in the form of jobs, and a job is comprised of one or several tasks. In total, six kinds of data tables are provided in this cluster trace, which are the job events, task events, machine events, machine attributes,[1] task constraints, and task usage.

Since the data trace is enormous, first, the entire data set is sampled and the features are extracted. The tables of task events and task usage are used to extract the arriving task information, and the table of machine attributes is used to extract the machine information which is used as the CPU and memory capacity of nodes. Since there are some conflicts in the tables of task events and task usage, these two tables need to be merged together, and some null or illegal values are deleted. Then, the CPU and memory capacities of nodes are extracted from the merged table. As for task information, the start time, end time, and duration need to be obtained. The first and last time slot of one task is considered as the start time and end time of this task, respectively. The durations of tasks can be calculated based on the start time and the end time.

After pre-processing, about 1×10^5 tasks and 100 nodes are extracted from Google cluster data. 50 percent of the data is used as a training set, 20 percent is used as a validation set, and the remaining 30 percent is used as a test set.

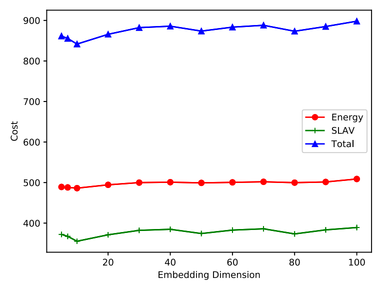
Moreover, since the raw data of resource requirements and capacities are floating numbers, it cannot be converted in vector representations directly. According to Section 4.1, all values are divided into $X + 1$ levels between their maximum value and minimum value, and each value can be graded into one level by rounding.

Baselines. To demonstrate the effectiveness of our models, the following methods are adopted as baselines for performance comparison:

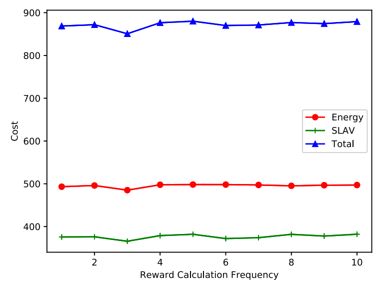
1) *Greedy*: This is a popular algorithm which always chooses the best action during the current time slot according to the minimum total cost.
2) *PABFD*: The Power Aware Best Fit Decreasing (PABFD) algorithm [50] is an improvement of the BFD algorithm [51], it sorts all the nodes in the decreasing order of their current CPU utilization and allocates each task to a node that provides the least increase of the energy consumption caused by the allocation.
3) *TDQL*: It is a Traditional DQL (TDQL) method which combines Q-learning with a three-layer DNN.

1. The word 'node' is used in this paper, 'machine events' and 'machine attributes' are the names of tables in Google cluster trace.

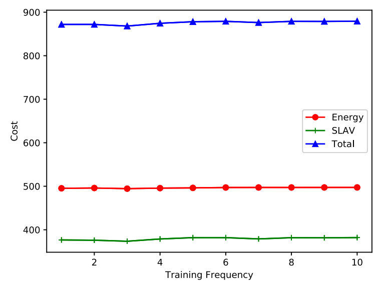
Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.



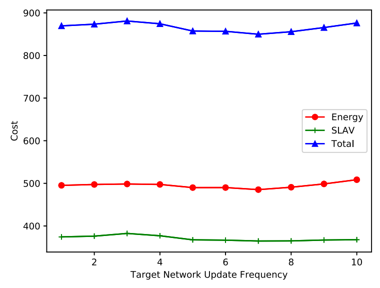
(a) Embedding Dimension



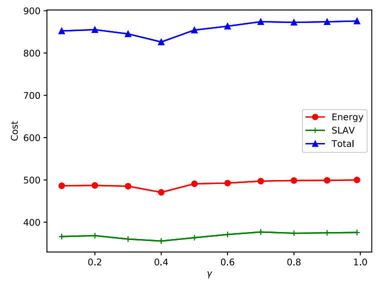
(b) Reward Calculation Frequency



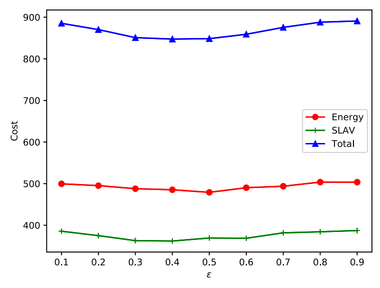
(c) Training Frequency



(d) Target Network Update Frequency



(e) γ



(f) ϵ

Fig. 4. Hyper-parameter selection.

4) *TransEC-TDQL*: It is a TDQL method which combines Q-learning with a TransEC model (without double Q-learning).

For TDQL algorithm, the amount of parameters used to represent the environment is increased as the number of nodes and tasks increases (while the TransEC-DQL algorithm is learned by representation and the amount of parameters required is certain). In real-world EC scenarios, TDQL algorithm cannot be executed since the number of nodes and tasks are in millions. As a result, to be able to compare with traditional methods, in the experiment the representation of environment is also used in TDQL algorithm.

Evaluation Methods. The experiments are conducted following these steps: 1) To compare the performance of different hyper-parameters and select the proper hyper-parameters, experiments with different β, number of nodes are conducted, and total reward, energy consumption, and SLAV are observed. 2) To compare the effectiveness of the proposed models against TDQL algorithm, the memory usage of GPU is recorded when running the different models.

Parameter Settings. The dimension of vector space Y is set to 10. The reward calculation frequency and training frequency of reinforcement learning is set to 3. The ϵ is set to 0.4 and changed over time as [52]

$$\epsilon \leftarrow \epsilon_1 + (\epsilon_0 - \epsilon_1) \times e^{-step/200},$$

where $\epsilon_0 = 0.4$, $\epsilon_1 = 0.05$, and the *step* is training step. The target network is updated every 7 episodes. The discount parameter γ is set to 0.4. The selections of these parameters are based on our experiments as shown in Fig. 4, and are described in the next subsection.

Besides, the X which denotes the number of resource levels is set to 100. The number of CNN layers is 1 and the number of linear layers is also 1. The sizes of filters of the CNNs for task distribution and node distribution are set to $(3, 5)$ and $(3, 3)$, respectively. And the filter numbers of all CNNs are set to 1. The kernel size of max-pooling layer is set to (3,3). Finally, the energy model of each node are shown in Table 1 [32].

6.2 Experimental Results

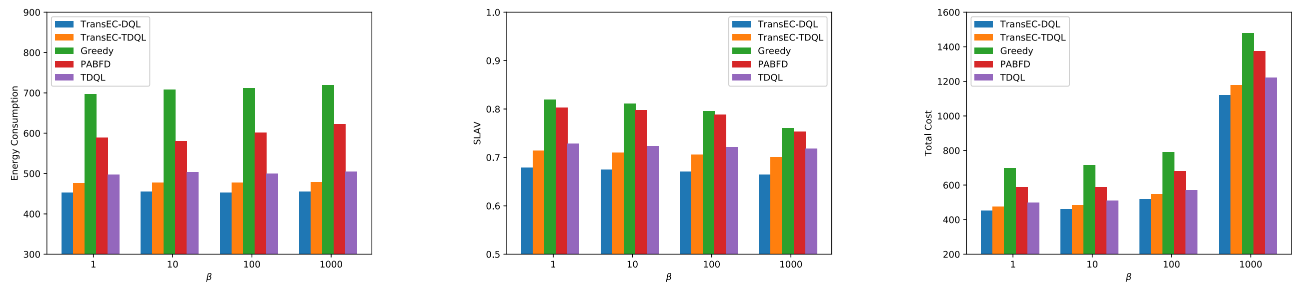
The experimental results are introduced in this section. First, the results of hyper-parameter selection are described. Then the performance with different parameters and the advantages against TDQL are shown.

Hyper-Parameter Selection. The hyper-parameter selections are based on the validation set, and the results are shown in Fig. 4. In Fig. 4, for better comparison, β is set to 500. First, as shown in Fig. 4a, the cost of different embedding dimension of vector space Y is shown. The Y is set to 10 since the cost is minimal when $Y = 10$. This is because larger dimensions lead to inadequate training, while smaller dimensions cause overfitting of the network. Second, for reward calculation frequency shown in Fig. 4b, the total cost is minimal when frequency equals to 3. The reason is that a larger frequency will cause reward to consider too many subsequent actions which reduces the impact of the previous action, while a smaller frequency cannot fully consider the after-effect of the current action.

TABLE 1
Energy Consumption at Different CPU Utilization (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	

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.



(a) Energy Consumption (b) SLAV (c) Total Cost

Fig. 5. Performance with different β.

Third, from Fig. 4c, the performance of different training frequencies is shown. The total cost is minimal when training frequency equals 3. The results show that the proper training frequency can take enough and sufficient knowledge of other time slots for decision-making of task scheduling and improve the cost. Then, the performance of different target network update frequencies is shown in Fig. 4d. With proper update frequency which equals to 7, the target network can provide a stable target Q value while effectively reduce the cost.

Finally, the performance of different discount parameter γ and ϵ is shown in Figs. 4e and 4f, respectively. γ is used to measure the impact of the future Q-value when updating the Q-value. By choosing the appropriate γ, we can well consider the impact of current and future actions. Besides, ϵ is used to determine the ratio of the selection of the best action and random action. By setting a reasonable ϵ, a good trade-off can be made between exploration and exploitation.

Moreover, as shown in Fig. 4, the hyper-parameters have little effect on our experimental results, so our TransEC model and TransEC-DQL algorithm have a high robustness.

Performance With Different Parameters. The parameter β and node number are evaluated as follows.

Performance With Different β. Experiments with different β in Problem 1 are conducted, the performances of the energy consumption, SLAV, and total cost of TransEC-DQL, TransEC-TDQL, Greedy, PABFD, and TDQL are shown in Figs. 5a, 5b, and 5c, respectively. From Fig. 5a, it is obvious that the energy consumption of TransEC-DQL is less than the TransEC-TDQL, Greedy, PABFD, and TDQL algorithms of all different β values. Besides, Fig. 5b shows that the SLAV of four algorithms is ordered as TransEC-DQL $<$ TransEC-TDQL $<$ TDQL $<$ PABFD $<$ Greedy. And the performance of SLAV of TransEC-DQL is much less than Greedy and PABFD algorithms. The results show that the proposed TransEC-DQL improve the SLAV significantly. In addition, in Fig. 5c, the total cost consists of energy consumption and SLAV with different β is shown. It can be concluded that the proposed TransEC-DQL algorithm is better than the other three algorithms.

Performance With Different Node Numbers. Experiments with different node numbers are conducted, the performances of the energy consumption, SLAV, and total cost are shown in Figs. 6a, 6b, and 6c, respectively. Fig. 6a shows that the energy consumption of TransEC-DQL is less than that of TransEC-TDQL, Greedy, PABFD, and TDQL algorithms. The reason is that more consideration of energy consumption is taken in the proposed algorithm. The SLAV performance of four different algorithms is shown in Fig. 6b, which is ordered as TransEC-DQL $<$ TransEC-TDQL $<$ TDQL $<$ PABFD $<$ Greedy. Besides, as shown in Fig. 6c, the total cost of TransEC-DQL is less than PABFD, Greedy, and TDQL algorithms. In short, the TransEC-DQL algorithm is better than three baseline algorithms.

To sum up, our proposed algorithms outperform the baselines 18.04 and 9.94 percent on average regarding energy consumption and SLAV, respectively.

Advantage against TDQL. To show the advantage of the proposed TransEC model against DNN, we compare the computation resource consumption when running our algorithm against TDQL algorithm. The comparison results are shown in Table 2. The values are the average of 5 experimental results of TransEC-DQL, TransEC-TDQL (without the optimization of double Q-learning), and TDQL algorithms, respectively. From Table 2, it is evident that the proposed TransEC-DQL algorithm needs less computation resource.

Beside, the time consumptions for decision-making and training of different algorithms are shown in Table 3, where TE-DQL and TE-TDQL are short for TransEC-DQL and TransEC-TDQL, respectively. From Table 3, it can be concluded that although the time consumption for decision-making of TransEC-DQL is longer than PABFD and Greedy algorithms, it is still short enough for EC scenarios. The training time is longer than TransEC-TDQL, because double Q-learning needs to find the action with the maximum Q-value by outputting all Q-values in the Q-network first, and then calculate the Q-value for the max action again in the target network. However, it is still acceptable for training in EC scenarios.

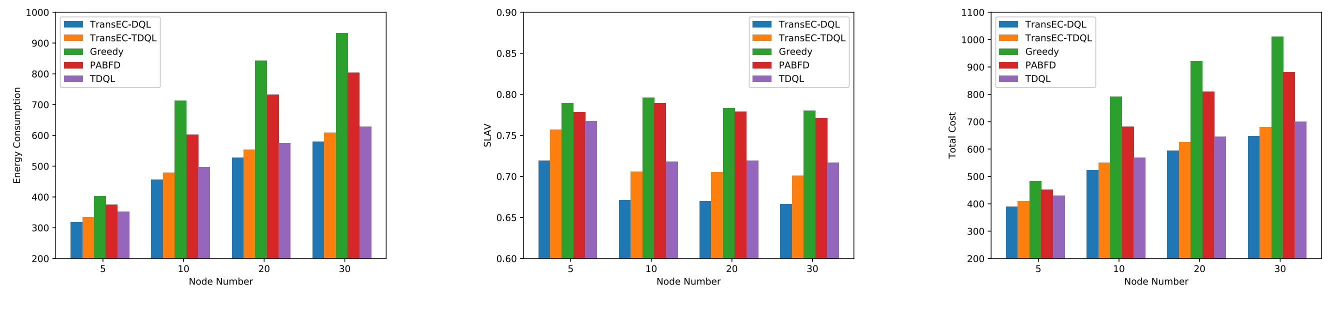
TABLE 2
GPU Memory Usage Comparison (MB)

TransEC-DQL	TransEC-TDQL	TDQL
673	713	983

TABLE 3
Time Consumption for Decision-Making and Training (s)

	TE-DQL	TE-TDQL	TDQL	PABFD	Greedy
Decision	0.0127	0.0132	0.0066	0.0002	0.00005
Training	0.4038	0.3015	0.3326	-	-

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.



(a) Energy Consumption (b) SLAV (c) Total Cost

Fig. 6. Performance with different node number.

7 Conclusion and Discussions

In this paper, we are the first to propose the combination of representation algorithms and reinforcement learning algorithms to solve the task scheduling in EC. First, the entities and relations of EC are defined, and the task scheduling problem is formulated. Then, the embeddings of the entities in EC are defined, and the relation computation model is proposed. Third, to train the embedding vectors and make the task scheduling decisions, a DQL based learning algorithm has been proposed. The experimental results show that the performance of our algorithm outperforms the three baseline algorithms which can represent the state of the art technology.

This paper is our first work towards the combination of representation learning and reinforcement learning for task scheduling in EC. We explore the possibility of combining them, and the experiment proved that our exploration is effective.

However, there are still many remaining research topics for this area. For example, considering the advancement of the network structure in the field of natural language processing in recent years, the CNN used in this paper is relatively simple, and the capacity of the model and the ability to extract represented relations are limited. Therefore, we may consider to use a more advanced structure, like PCNN [53], self-attention [54], etc., to replace the CNN.

Acknowledgments

This work was supported by Chinese National Research Fund (NSFC) Key Project No. 61532013 and No. 61872239; also partially funded by The Science and Technology Development Fund, Macau SAR, FDCT.0007/2018/A1, FDCT.0060/2019/A1, and by University of Macau (File no. MYRG2018-00237-FST, CPG2020-00015-IOTSC, and SRG2018-00111-FST).

References

[1] G. Ananthanarayanan *et al.*, “Real-time video analytics: The killer app for edge computing,” *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[2] L. Liu, H. Li, and M. Gruteser, “Edge assisted real-time object detection for mobile augmented reality,” in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Thirdquarter 2017.

[5] A. I. Orhean, F. Pop, and I. Raicu, “New scheduling approach using reinforcement learning for heterogeneous distributed systems,” *J. Parallel Distrib. Comput.*, vol. 117, pp. 292–302, 2018.

[6] X. Ma, S. Zhang, W. Li, P. Zhang, C. Lin, and X. Shen, “Cost-efficient workload scheduling in cloud assisted mobile edge computing,” in *Proc. IEEE/ACM 25th Int. Symp. Quality Service*, 2017, pp. 1–10.

[7] L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, “Joint computation offloading and multi-user scheduling using approximate dynamic programming in NB-IoT edge computing system,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5345–5362, Jun. 2019.

[8] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, “Offloading in mobile edge computing: Task allocation and computational frequency scaling,” *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[9] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[10] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, 2015, Art. no. 529.

[11] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proc. 15th ACM Workshop Hot Topics Netw.*, 2016, pp. 50–56.

[12] L. T. Tan and R. Q. Hu, “Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10 190–10 203, Nov. 2018.

[13] N. Liu *et al.*, “A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning,” in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[14] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, “Learning entity and relation embeddings for knowledge graph completion,” in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, vol. 15, pp. 2181–2187.

[15] A. Bordes, N. Usunier, A. Garcia-Duran , J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 2787–2795.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[17] V. Mnih *et al.*., “Playing atari with deep reinforcement learning,” 2013, *arXiv:1312.5602*.

[18] H. V. Hasselt, “Double Q-learning,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[19] Q. Wang, Z. Mao, B. Wang, and L. Guo, “Knowledge graph embedding: A survey of approaches and applications,” *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017.

[20] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, “Representation learning of knowledge graphs with entity descriptions,” in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2659–2665.

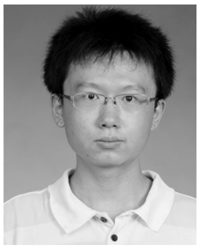
[21] Z. Wang, J. Zhang, J. Feng, and Z. Chen, “Knowledge graph embedding by translating on hyperplanes,” in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, vol. 14, pp. 1112–1119.

[22] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, “Knowledge graph embedding via dynamic mapping matrix,” in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics and the 7th Int. Joint Conf. Natural Lang. Process.*, 2015, vol. 1, pp. 687–696.

[23] G. Ji, K. Liu, S. He, and J. Zhao, “Knowledge graph completion with adaptive sparse transfer matrix,” in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 985–991.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

[24] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.
[25] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," 2018, *arXiv: 1810.07862*.
[26] L. T. Tan, R. Q. Hu, and L. Hanzo, "Twin-timescale artificial intelligence aided mobility-aware edge caching and computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3086–3099, Apr. 2019.
[27] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt , M. Lanctot, and N. De Freitas , "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*.
[28] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.
[29] P. V. R. Ferreira *et al.*, "Multi-objective reinforcement learning for cognitive satellite communications using deep neural network ensembles," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 5, pp. 1030–1041, May 2018.
[30] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 712–725, Sep./Oct. 2019.
[31] Y. He, Z. Zhang, and Y. Zhang, "A big data deep reinforcement learning approach to next generation green wireless networks," in *Proc. IEEE Global Commun. Conf.*, 2017, pp. 1–6.
[32] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service*, 2017, pp. 1–6.
[33] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, no. SI, pp. 239–254, 2002.
[34] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. Lau, "Dynamic virtual machine management via approximate Markov decision process," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[35] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," 2010, *arXiv:1006.0308*.
[36] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2014, pp. 500–507.
[37] Y. Zhai, Y. Wang, I. You, J. Yuan, Y. Ren, and X. Shan, "A DHT and MDP-based mobility management scheme for large-scale mobile Internet," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2011, pp. 379–384.
[38] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, no. Aug, pp. 2493–2537, 2011.
[39] A. Bordes *et al.*, "Learning structured embeddings of knowledge bases," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011, vol. 6, pp. 301–306.
[40] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1247–1250.
[41] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledgebase," *Commun. ACM*, vol. 57, no. 10, pp. 78–85, 2014.
[42] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax," in *Proc. Annu. Conf. Artif. Intell.*, 2011, pp. 335–346.
[43] H. Van Hasselt , A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[44] P. Rodríguez, M. A. Bautista, J. Gonzalez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image Vis. Comput.*, vol. 75, pp. 21–31, 2018.
[45] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.
[46] F. S. Melo, "Convergence of Q-learning: A simple proof," Inst. Syst. Robot., Lisboa, Portugal, pp. 1–4, 2001.
[47] J. Wilkes, "More Google cluster data," *Google research blog*, Nov. 2011, posted at [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.htm l
[48] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," *ACM Symp. Cloud Comput.*, San Jose, CA, USA, 2012. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/CloudComputing/googletrace-socc2012.pdf
[49] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, Secondquarter 2014.
[50] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.
[51] M. Yue, "A simple proof of the inequality FFD (L)$\leq$ 11/9 OPT (L) + 1,$\forall$ L for the FFD bin-packing algorithm," *Acta Math. Appl. Sin.*, vol. 7, no. 4, pp. 321–331, 1991.
[52] S. Devlin, L. Yliniemi, D. Kudenko, and K. Tumer, "Potential-based difference rewards for multiagent reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2014, pp. 165–172.
[53] D. Zeng, K. Liu, Y. Chen, and J. Zhao, "Distant supervision for relation extraction via piecewise convolutional neural networks," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2015, pp. 1753–1762.
[54] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, vol. 1, pp. 2124–2133.



Zhiqing Tang received the BS degree from the School of Communication and Information Engineering, University of Electronic Science and Technology of China, China, in 2015 and is currently working toward the PhD degree with the 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 (Fellow, IEEE) received the BSc and MSc degrees from Center South University, China, in 1982 and 1984, respectively. and the master of applied Science and PhD degrees from the Polytechnic Faculty of Mons, Belgium, in 1992 and 1993, respectively, all in computer science. He is currently a chair professor, deputy director of State Kay Laboratory of Internet of Things for Smart City, University of Macau, Zhuhai. He has been a Zhiyuan chair prof at Shanghai Jiaotong University, China. In 1993-1995, he joined German National Research Center for Information Science (GMD) in Bonn (St. Augustine) as a research fellow. From 1995-2013, he worked in City University of Hong Kong as a professor. His contributions have been reconganized as optimal network routing and deployment; vertex cover; anycast and QoS routing, and sensors networking; knowledge relation extractions; NLP and edge computing. He has more than 500 publications in the prestige international journals/conferences and research books and book chapters. He has received best product awards from the Internatonal Science&Technology Expos (Shenzhen), in 2011/2012 and 1st- Prize of Scientific Research Awards from Ministry of Education of China, in 2017 (list 2). He has served as area editor for various prestige international journals, chair and PC member/keynote speaker for many top international conferences. He is the distinguished member of CCF.

Xiaojie Zhou received the BS degree from the School of Data and Computer Science, Sun Yat-sen University, China, in 2016 and is currently toward the master's degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include edge computing, resource scheduling, and reinforcement learning.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.

Wenmian Yang received the BS degree from the Department of Electronic Information and Electrical Engineering, Dalian University of Technology, China, in 2015 and is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He is also a research assistant at University of Macau, Zhuhai. His current research interests include machine learning and natural language processing.

Yongjian You received the BS degree from the College of Science, Nanjing University of Posts and Telecommunications, China, in 2017 and is currently working toward the master's degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His current research interests include machine learning and natural language processing.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on May 13,2022 at 17:42:36 UTC from IEEE Xplore. Restrictions apply.