Computers, Materials & Continua CMC, vol.64, no.3, pp.1657-1671, 2020

A DRL-Based Container Placement Scheme with Auxiliary Tasks

Ningcheng Yuan[1], Chao Jia[2, *], Jizhao Lu[3], Shaoyong Guo[1], Wencui Li[3], Xuesong Qiu[1] and Lei Shi[4]

Abstract: Container is an emerging virtualization technology and widely adopted in the cloud to provide services because of its lightweight, flexible, isolated and highly portable properties. Cloud services are often instantiated as clusters of interconnected containers. Due to the stochastic service arrival and complicated cloud environment, it is challenging to achieve an optimal container placement (CP) scheme. We propose to leverage Deep Reinforcement Learning (DRL) for solving CP problem, which is able to learn from experience interacting with the environment and does not rely on mathematical model or prior knowledge. However, applying DRL method directly dose not lead to a satisfying result because of sophisticated environment states and huge action spaces. In this paper, we propose UNREAL-CP, a DRL-based method to place container instances on servers while considering end to end delay and resource utilization cost. The proposed method is an actor-critic-based approach, which has advantages in dealing with the huge action space. Moreover, the idea of auxiliary learning is also included in our architecture. We design two auxiliary learning tasks about load balancing to improve algorithm performance. Compared to other DRL methods, extensive simulation results show that UNREAL-CP performs better up to 28.6% in terms of reducing delay and deployment cost with high training efficiency and responding speed.

Keywords: Container placement, deep reinforcement learning, auxiliary learning.

1 Introduction

Container has become a popular operating system (OS) level technology for providing cloud computing services, due to high-performance, scalability, lightweight resource allocation and good isolation. Unlike virtual machines (VMs), containers do not require the entire OS resources. They are able to share the same OS kernel with each other to reduce the provisioning cost and improve resource utilization efficiency. The application development platform like Docker allows containers deployed on the top of any

[1] Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[2] China Electronics Standardization Institute, Beijing, China.

[3] Communication Operation Center, State Grid Henan Electric Power Company Information & Telecommunication Company, Zhengzhou, China.

[4] Institute of Technology Carlow, Carlow, Ireland.

[*] Corresponding Author: Chao Jia. Email: jiachao@cesi.cn.

Received: 21 January 2020; Accepted: 02 April 2020.

infrastructure [Hussein, Mousa and Alqarni (2019)]. Therefore, more and more modern service providers tend to deploy services in the form of containers.

Generally, cloud users require several containers to create a container cluster (CC) to deploy a service chain [Zhou, Li and Wu (2019)]. Container placement (CP) problem is critical because it has a significant impact on QoS and network performance. In this paper, we aim to obtain a placement scheme for containers on servers to improve end to end delay while minimizing the deployment cost, which is an NP-hard combinational optimization problem. Moreover, CCs with different resource requirement arrive randomly, making dynamic network state hard to be captured with a mathematical model.

Facing the above challenges, deep reinforcement learning (DRL)-based algorithm is proved to have great advantages in solving such problems. It does not rely on accurate and solvable mathematical model and able to learn from the experiences generated by interacting with the environment. And after learning, it can directly derive the CC placement strategy with greatest long-term reward based on the current network state.

In this paper, we propose a DRL-based algorithm to solve the CP problem. First, we introduce the Markov decision process (MDP) model to capture network dynamics. Secondly, to deal with the high-dimensional state and solution space, we chose the actor-critic method because it has great advantages in solving dynamic and continuous control problems [Xu, Tang, Meng et al. (2018); Mnih, Badia, Mirza et al. (2016)]. However, we find that direct application of the state-of-art actor-critic method, asynchronous advantage actor-critic (A3C) does not work well for our CP problem. Finally, we consider the impact of resource utilization balancing on network performance. Load balancing prevents any server from overloading and improves service availability. Meanwhile, it also guarantees acceptable server resource utilization, which makes the servers keep fast response time. Therefore, we design two auxiliary tasks related to resource utilization balancing to speed up the training process and improve the performance of A3C. This improvement that using auxiliary tasks to accelerate convergence is called unsupervised reinforcement and auxiliary learning (UNREAL) [Jaderberg, Mnih, Czarnecki et al. (2016)]. The main contributions of this paper are as follows:

- We study the CP problem for reducing the end to end delay and service deployment cost and establish an optimization model for it.
- We introduce DRL based solution to CP problem, in order to adapt network changes. And we show that direct application of A3C does not work well for CP problem.
- We present an improved A3C algorithm called UNREAL-CP to achieve optimal placement scheme of containers with low delay and deployment cost.

The rest of this paper is organized as follows. Existing studies are reviewed in Section 2. The system model and CP problem definition are presented in Section 3. The UNREAL-CP algorithm is introduced in Section 4. Section 5 shows the evaluation results and Section 6 concludes the paper.

2 Related work

Service deployment algorithms often focus on minimizing the total hosting cost (such as resource requirements, power consumption, etc.) while trying to maximize total revenue (such as throughput, network utility, etc.). One related problem is the VNF placement problem. Song et al. [Song, Zhang, Yu et al. (2017)] consider the tradeoff between computing resource cost and communication resource cost. In Gu et al. [Gu, Chen, Jin et al. (2018)], the VNF placement problem is formulated into a mixed-integer linear programming problem and solved by relaxation-based algorithm to deal with the computational complexity. Cloud container cluster provisioning belongs to the category of virtual network embedding. Zhou et al. [Zhou, Li and Wu (2019)] design an online method to address the optimal placement of containers with maximal value of all served clusters. Lv et al. [Lv, Zhang, Li et al. (2019)] study the container allocation problem in real industrial environment while considering the balance between communication cost and resource utilization in large-scale data centers, and propose two algorithms to solve the container placement problem and container reassignment problem respectively. Zhang et al. [Zhang, Chen, Dong et al. (2019)] propose an improved genetic algorithm to search a placement scheme with optimal energy consumption. These existing works mainly use mathematical methods such as integer linear programming (ILP) and integer nonlinear programming (INLP) model to abstract the problem, and provide mathematical method like primal-dual algorithm or heuristic algorithm [Zhou, Li and Wu (2019); Piet, Bart and Pieter (2018); Quang, Singh, Bradai et al. (2018)] to solve it.

However, these existing approaches can only work under the assumption that arriving services are predictable or known a priori. Due to the randomness arrival of service requests, network state and network flow are time-varying. The deployment methods mentioned below have limitations to adapt to these network changes. DRL-based algorithm has great advantages in solving such problems because it interacts with the environment and learns from experience [Huang, Yuan, Qiao et al. (2018); Li (2017); Wei, Wang, Guo et al. (2019)]. In Zhao et al. [Zhao, Liang, Niyato et al. (2018)], double deep Q-learning (DDQN) approach is introduced to obtain optimal resource allocation in heterogeneous networks. However, DDQN only works for the problem with a low-dimensional discrete action space. Xu et al. [Xu, Tang, Meng et al. (2018)] propose to leverage deep deterministic policy gradient for model-free control in network, but find that direct application of DRL-based solution does not lead to satisfying performance. Therefore, an effective DRL-based method is needed to capture network dynamics and deal with high-dimensional action space.

3 Problem definition

3.1 Network model

The cloud network is modeled as an undirected graph $G=\{Z,E\}$. Z denotes the set of servers where computing resources are hosted. There are multiple types of resources, like CPU, Memory, RAM and disk. Let R denote the set of resource types. For each server $z \in Z$, let C_{zr} denote the capacity of resource $r \in R$. Let E be the set of links, and $e(z_i, z_j) \in E$ denote the link that connects servers z_i and z_j, with bandwidth resource $B(z_i, z_j)$.

3.2 Service model

Considering a set of discrete time slots $\{0,1,2,...,t,...,T-1\}$, the arriving service set is H, and service $h \in H$ arrives at time t_s, requiring a CC. Let V_h be the set of containers in service h's CC. A container $v \in V_h$ needs x_{vr}^h units of resource r. Upon each service arriving, the service provider determines whether to accept it and the placement scheme of service h's CC. $\alpha_h \in \{0,1\}$ indicates whether service h is accepted ($\alpha_h = 1$) or not ($\alpha_h = 0$). We define the placement decision variables $\beta_{vz}^h \in \{0,1\}, \forall v \in V_h, \forall z \in Z$, where $\beta_{vz}^h = 1$ if container v is placed on server z and 0 otherwise. If all the containers of a service are successfully deployed, the service request is satisfied. Fig. 1 shows a placement scheme for the CC of service 1. Let $y^h(v_i, v_j)$ denote the bandwidth requirement for packet transmission from v_i to v_j in service h's CC.

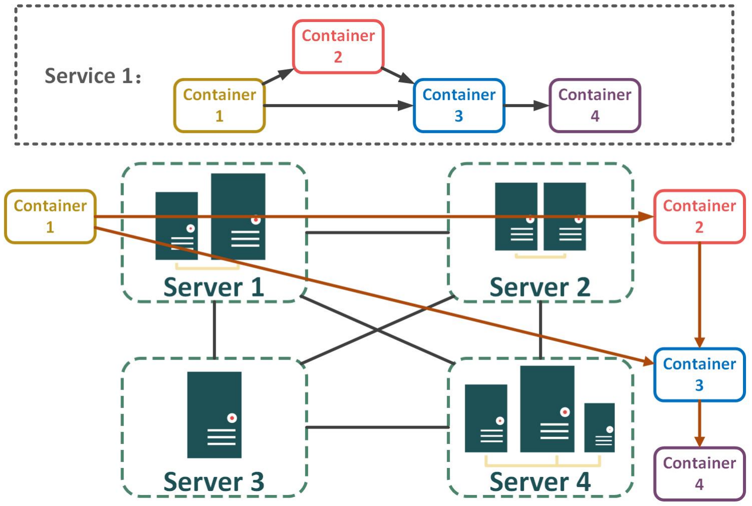


Figure 1: An example of container placement

3.3 Objective

3.3.1 Average delay

If h is accepted, the processing rate at v is μ_v^h, thence the average packet processing time at v in h's CC is $p_v^h = 1/\mu_v^h$. We have the total processing delay of h as:

$$p_h = \sum_{v \in V_h} p_v^h . \tag{1}$$

The average transmission time the packets experience from v_i to v_j is $l^h(v_i, v_j)$. Let $Z(v)$ denote the server where container v is placed. If v_i and v_j are hosted on the same server ($Z(v_i) = Z(v_j)$), the transmission process is negligible, i.e., $l^h(v_i, v_j) = 0$. Therefore, the total transmission delay of h is:

$$l_h = \sum_{v_i, v_j \in V_h, v_i \neq v_j} l^h(v_i, v_j) . \tag{2}$$

In conclusion, the average packet delay of accepted service h is $d_h = p_h + l_h$.

If h is rejected, the delay of h can be defined as its maximum tolerable delay $d_h = d_{tol}$.

3.3.2 Resource cost

For container v in h, each unit of resource r it uses requires the cost of f_{vr}^h. Similarly, cost per unit bandwidth is $f^h(v_i, v_j)$. Therefore, the total cost of satisfied service h is:

$$f_h = \sum_{v\in V, r\in R, z\in Z} f_{vr}^h x_{vr}^h \beta_{vz}^h + \sum_{v_i, v_j\in V_h, z\in Z} f^h(v_i, v_j) y^h(v_i, v_j) \beta_{v_i z}^h \beta_{v_j z}^h . \tag{3}$$

Notations are summarized in Tab. 1.

Table 1: Summary of notations

Z	set of servers
R	set of computational resource types
E	set of links
C_{zr}	capacity of resource r at server z
$B(z_i, z_j)$	bandwidth of link between servers z_i and z_j
H	set of services
V_h	set of containers in service h's CC
x_{vr}^h	units of resource r that v in service s's CC needs
$y^h(v_i, v_j)$	bandwidth requirement from v_i to v_j
α_h	$\alpha_h = 1$ if service h is accepted, and 0 otherwise
β_{vz}^h	$\beta_{vz}^h = 1$ if container v is placed on server z and 0 otherwise
μ_v^h	the processing rate at v
$l^h(v_i, v_j)$	average transmission time packets experience from v_i to v_j
d_h	total delay of service h
$Z(v)$	the server where container v is placed
f_{vr}^h	cost per unit of resource r
$f^h(v_i, v_j)$	cost per unit of bandwidth

3.3.3 Optimization problem

Based on the above definitions, our objective is to minimize the total cost of all deployed CCs and the total delay. We transfer the multiple objectives into a single scalar and define the objective to be minimized as a weighted sum of cost and delay:

$$\min[\omega_f \sum_{h\in H} f_h + \omega_d \sum_{h\in H} d_h] . \tag{4}$$

Subject to:

$$\sum_{z\in Z}\beta_{vz}^{h}\le 1,\forall h\in H,\forall v\in V, \tag{4a}$$

$$\alpha_i=\sum_{z\in Z}\beta_{vz}^{h},\forall h\in H,\forall v\in V, \tag{4b}$$

$$\sum_{h\in H}\sum_{v\in V_s}x_{vr}^{h}\beta_{vz}^{h}\le C_{zr},\forall r\in R,\forall z\in Z, \tag{4c}$$

$$\sum_{h\in H}\sum_{v_i,v_j\in V_s}y^{h}(v_i,v_j)\beta_{v_i z(v_i)}^{h}\beta_{v_j z(v_j)}^{h}\le B(z(v_i),z(v_j)),\forall e(z(v_i),z(v_j))\in E. \tag{4d}$$

Constraint (4a) guarantees that a container can be deployed in at most one server. Constraint (4b) indicates that only when all the containers of a service's CC are successfully deployed, the service request is satisfied. Constraints (4c) and (4d) ensure that each CC's resource occupation cannot exceed network resources.

4 Algorithm description

4.1 A3C algorithm

In a standard DRL setup, there is an agent interacting with environment. And the optimal problem is modeled as a Markov process (MDP) which is typically expressed as $\{s,a,r,p\}$, where s is state space, a is action space, r is instantaneous reward, and p is state transition probability. At each epoch i, agent observes the state s_i, executes the action a_i and get the next state s_{i+1} and instantaneous reward r_i. Using r_i, the agent updates an n-step return which is defined as the discounted sum of rewards:

$$G_{i:i+n}=\sum_{k=1}^{n}\gamma^{k}r_{i+k}\ , \tag{5}$$

where $\gamma\in[0,1]$ is a discount factor. This process continues until agent reaches a terminal state and restarts after it. The agent's objective is to find a mapping from state to action, which is called policy $\pi(s)$, to maximize the expected return $G_{i:\infty}=\sum_{k=0}^{\infty}\gamma^{k}r_{i+k}$.

The action-value function $Q^{\pi}(s,a)=E[G_{i:\infty}\mid s_i=s,a_i=a,\pi]$ is the expected return following action a from state s. In the seminal work, value-based model-free reinforcement learning methods (like Q-learning) approximate the action-value function using $Q(s,a;\theta)$ with parameters θ. The parameters θ are learned by minimizing a mean-squared error, for example optimizing a loss function in n-step Q-learning:

$$L_Q=E[(G_{i:i+n}+\gamma^{n}\max_{a'}Q(s',a';\theta_{i-1})-Q(s,a;\theta))^2], \tag{6}$$

where s' is the next state after s.

Compared to value-based methods, policy gradient methods parameterize the policy as $\pi(a\mid s;\theta)$, update parameters θ and adjust the policy to maximize the expected reward by performing gradient ascent on $E[G_{i:\infty}]$.

However, value-based DRL methods can only work for problems with a low-dimensional action space, since it needs to find the action that maximizes the action-value function, which requires an iteration process to solve a non-linear programming problem at every epoch. Policy-based methods can handle a large action space, but can only update parameters after each episode is completed, resulting in a very slow learning speed.

Since the total number of CC placement decision is $|Z|^{|H|\times|V_h|}$, our optimization problem is difficult to be solved by value or policy-only methods. So, we consider tackling it by an actor-critic approach, asynchronous advantage actor-critic algorithm, which uses a parameterized actor network to generate actions, so it can handle high dimension action space; at the same time, critic's value function estimation supports actor to update the gradient.

The parameterized actor function is implemented by CNN and expressed as $\pi(s_i \mid \theta^\pi)$. To measure the quality of the existing policy $\pi(s_i)$, value function $V(s_i) = E_{\pi(s)}[G_{i:\infty}]$ is defined to indicate the expected discounted return for following policy π from current state s_i. In actor-critic method, value function is estimated by parameterized critic: $V(s_i \mid \theta^V)$. The parameters of the actor and critic are learned by iteratively minimizing a sequence of loss functions, where actor's and critic's loss functions respectively defined as:

$$L_{actor} = E[\log \pi(s_i \mid \theta^\pi)(G_i - V(s_i \mid \theta^V))], \tag{7}$$

$$L_{critic} = E[(G_i - V(s_i \mid \theta^V))^2]. \tag{8}$$

To make the process of propagating rewards to relevant state much more efficient, G_i is updated toward the n-step return which defined as $\sum_{k=0}^{n-1} \gamma^k r_{i+k} + \gamma^n V(s_{i+n} \mid \theta^V)$. This results in a single reward r directly affecting the values of n preceding states.

In conclusion, the loss function of A3C is defined as

$$L_{A3C} = L_{actor} + L_{critic} - E[\beta H(\pi(s_i \mid \theta^\pi))], \tag{9}$$

where H is the entropy and hyperparameter β controls the strength of the entropy regularization term, which is used to prevent policy convergence too early.

To eliminate the correlation between samples, A3C algorithm relies on asynchronous actor-learners and accumulated updates for improving training stability. Compared with previous solutions that use experience replay to reduce time correlation, the asynchronous RL Framework saves storage space. Besides stabilizing learning, we obtain a reduction in training time, which is roughly linear in the number of parallel actor-learners.

In order to utilize the A3C algorithm, we model our placement problem as an MDP and design the state space, action space and reward function. The design of state, action and reward is important to the success of DRL methods. A valuable design should capture network states and key compositions CC placement without useless information.

State space: The sate consists of two components: network resource status and service requests. $s = \{s_{net}, s_{ser}\}$, where $s_{net} = \{C_{zr}, B(z_i, z_j) \mid z, z_i, z_j \in Z, r \in R\}$ and $s_{ser} = \{x_{vr}^h, y^h(v_i, v_j) \mid v, v_i, v_j \in V_h, h \in H, r \in R\}$.

Action space: The action is defined as a placement scheme for all services' CC, i.e., the solution to the optimization problem. The action vector $a=\{\alpha_h,\beta_{vz}^h\}$, where $v,\in V_h, h\in H, z\in Z$.

Reward function: When agent performs the action according to the current state and accomplish CC placement, it will get instant reward. For the goal of DRL is to maximize cumulative reward, we use the difference between the objective function at current state and that of next state: $r=(\omega_f\sum_{h\in H}f_h+\omega_d\sum_{h\in H}d_h)-(\omega_f\sum_{h\in H}f_h'+\omega_d\sum_{h\in H}d_h')$.

Although A3C algorithm performs well in many tasks as a state-of-art DRL method, our experimental results show that applying A3C directly to CC placement does not bring satisfactory performance. Its convergence speed is intolerable when the solution space is large. According to the analysis, we speculate that it is due to the following reasons:

- In the general A3C training process, the agent only considers maximizing cumulative reward to achieve the best policy, and doesn't clearly know how to explore.
- Although A3C uses an asynchronous method to reduce the correlation between samples, it cancels the replay buffer, which can increase the efficiency and stability of learning.

In response to such problems, we propose an improved A3C algorithm for CC placement problem.

4.2 UNREAL-CP

In the general RL training process, the agent only considers maximizing cumulative reward to achieve the best policy, a simple random noise based exploration method does not work well for our CP problem. But the environment may also contain other available training information. Firstly, to address the problem of low exploration efficiency, we introduce the idea of auxiliary learning to guide the exploration direction of the agent in the base task. The agent is trained to maximize the reward of multiple tasks that face the same goal. This method does not require additional supervision and is an unsupervised learning method, called unsupervised reinforcement and auxiliary learning (UNREAL). In addition to auxiliary tasks, we also use the replay buffer to improve the accuracy of value function and the efficiency of auxiliary tasks

Given a set of auxiliary control tasks C, we define an auxiliary control task $c\in C$ by reward function $r^{(c)}$, with the agent's policy $\pi^{(c)}$ for it. The overall objective is to maximize total performance across all tasks: $\arg\max_{\theta}E\{G\mid\pi\}+\lambda_c\sum_{c\in C}E\{G^{(c)}\mid\pi^{(c)}\}$. Where $G^{(c)}=\sum_{i=0}^{\infty}\gamma^i r_i^{(c)}$ is the discount return of auxiliary task c, and θ is the Common parameter of $\pi^{(c)}$ and π. By sharing parameters, the agent improves the performance of the policy π by optimizing the performance on the auxiliary task. For the auxiliary task c, we sample minibatch of transitions from replay buffer and use the Q-learning to optimize the action-value function $Q(s,a)=E[G_i\mid s_i=s,a_i=a]$. For each control task c, we optimize an *n*-step Q-learning loss:

$$L_Q^{(c)} = E[(\sum_{k=0}^{n-1} \gamma^k r_{i+k}^{(c)} + \gamma^n \max_{a'} Q^{(c)}(s', a', \theta_{i-1}) - Q^{(c)}(s, a, \theta_i))^2]. \tag{10}$$

In our problem, the balance of server resource utilization has a great impact on network performance. First of all, load balancing can prevent any one server from being overloaded or crashed, thereby increasing service availability. Second, when resource utilization is high, servers usually generate exponential response time. Load balancing ensures servers' acceptable resource utilization, resulting in a shorter response time. Third, under a balanced workload, no server becomes a bottleneck, which improves the overall network throughput.

Based on the significance of load balancing to network performance, we design the following auxiliary tasks:

4.2.1 Resource utilization control

If the resource utilization in one server is significantly higher than that in others, then it will become a bottleneck in the service and seriously reduce the overall network performance. So, we want the variance of resource utilization for all servers to be as small as possible.

The variance of resource utilization for type-r resource of all servers is given by

$$\sum_{z \in Z} \frac{(U_{zr} - \overline{U_r})^2}{|Z|}, \tag{11}$$

where U_{zr} is the utilization of r on server z and given by

$$U_{zr} = \frac{\sum_{h \in H, v \in V_h} x_{vr}^h \beta_{vz}^h}{C_{zr}}, \tag{12}$$

$\overline{U_r}$ is r's mean utilization of all servers and $|Z|$ is the number of servers. The total resource utilization variance is the sum of that for all resource types, which is expressed as:

$$Var(U) = \sum_{r \in R} \sum_{z \in Z} \frac{(U_{zr} - \overline{U_r})^2}{|Z|}. \tag{13}$$

The objective of this auxiliary task is to minimize the total variance of total resource utilization (i.e., $\min Var(U)$), so we refer to it as resource utilization control. The reward function of this task is defined as

$$r^{(RUC)} = Var(U) - Var'(U), \tag{14}$$

where $Var'(U)$ is the total utilization variance of next state.

4.2.2 Available resource balance control

The amount of different types of available resources in each server also needs to be balanced. If there are no RAM resources available, the remaining CPU resources in the server are useless for coming service requests. For two kinds of resources r_i and r_j, let

$\delta(r_i,r_j)$ denote the expected ratio between r_i and r_j. The available resource balance index for resources r_i and r_j is defined as:

$$\sum_{z\in Z}\max\{0, A_{zr_i}-A_{zr_j}\times\delta(r_i,r_j)\}, \tag{15}$$

where $A_{zr}=C_{zr}-\sum_{h\in H,v\in V_h} x_{vr}^h\beta_{vz}^h$ is the available resource r on server z. This index can reflect whether different resources are used according to the expected ratio. The total available resource balance index is the sum of the index of all possible pairs of different resources:

$$Bal(A)=\sum_{\forall r_i,r_j\in R,r_i\neq r_j}\sum_{z\in Z}\max\{0, A_{zr_i}-A_{zr_j}\times\delta(r_i,r_j)\}. \tag{16}$$

Finally, the reward function of available resource balance control task is defined as:

$$r^{(RBC)}=Bal(A)-Bal'(A), \tag{17}$$

where $Bal'(A)$ is the total available resource balance index of next state.

Based on the above definitions, the UNREAL algorithm for CC placement problem aims to maximize total performance across all these auxiliary tasks, and optimize a single combined loss function with respect to the joint parameters θ. The loss function combines the A3C loss L_{A3C} together with auxiliary resource utilization control loss L_{RUC} and available resource balance control loss L_{RBC}:

$$L_{UNREAL}=L_{A3C}+\lambda_{RUC}L_{RUC}+\lambda_{RBC}L_{RBC}. \tag{18}$$

In summary, Our UNREAL-CP algorithm is expressed as algorithm 1:

Algorithm.1 UNREAL-CP

```
Input: set of servers $Z$, set of services $H$
Output: CC placement scheme $\alpha_h$, $\beta_{vz}^h$
Initialize: $C_{zr}$, $B(z_i,z_j)$ and other parameters
repeat
    for each actor-critic thread:
        Reset gradients: $d\theta^\pi \leftarrow 0$, $d\theta^V \leftarrow 0$
        Synchronize thread-specific parameters $\theta^{\pi'}=\theta^\pi$ and $\theta^{V'}=\theta^V$
        $t_{start}=t$
        Get state $s_t$
        repeat
            Get $a_t$ according to policy $\pi(s_t\mid\theta^\pi)$
            Execute action $a_t$, receive reward $r_t$, $r_t^{(c)}$ and new state $s_{t+1}$
            Store transition $\{s_t,a_t,(r_t,r_t^{(c)}),s_{t+1}\}$ to replay buffer
            $t\leftarrow t+1$
        until terminal $s_t$ or $t - t_{start} == t_{max}$
        $G = \begin{cases} 0, \text{ for terminal } s_t \\ V(s_t \mid \theta^{V'}), \text{ for non-terminal } s_t \end{cases}$
        for $i \in \{t-1, ..., t_{start}\}$ do
            $G \leftarrow r_i + \gamma G$
            Accumulate gradients w. r. t. $\theta^{\pi'}$ and $\theta^{V'}$
        end for
        Perform asynchronous update of $\theta^{\pi}$ using $d\theta^{\pi}$ and of $\theta^{V}$ using $d\theta^{V}$
    end for
    for auxiliary Q-learning thread:
        Reset gradients: $d\theta^{c} \leftarrow 0$
        Synchronize thread-specific parameters $\theta^{c'} = \theta^{V}$
        Sample minibatch of transitions $\{s_i, a_i, (r_i, r_i^{(c)}), s_{i+1}\}$ from replay buffer
        $G^{(c)} = \begin{cases} 0, \text{ for terminal } s_i \\ \max Q(s_i, a_i; \theta^{c'}), \text{ for non-terminal } s_i \end{cases}$
        $G^{(c)} \leftarrow \sum_{k=0}^{n-1} \gamma^k r_{i+k}^{(c)} + \gamma^n G^{(c)}$
        Perform update of $\theta^{V}$ using $d\theta^{V} \leftarrow \frac{\partial (G^{(c)} - Q(s_i, a_i; \theta^{c'}))^2}{\partial \theta^{c'}}$
    end for
    $T \leftarrow T + 1$
until $T > T_{max}$
```

5 Performance evaluation

In order to evaluate the performance of the proposed algorithm, we used the python language to perform experiments on a regular desktop with an Intel Core 2.6 Ghz CPU with 8 GB memory. The CNN network in the architecture is implemented by TensorFlow.

In the experiment, the default number of servers is 10. The types of resource include CPU, Memory, SSD, network I/O and so on. The amount of each kind of resource on each node is between 50 and 100, and the link bandwidth between all nodes is 100-300. We assume that each CC contains 1-10 containers and each container consumes three types of resources. The resource consumption is set according to the task's demand in the real-world trace. The traffic volume between containers is set in the range of [5,15].

We compare our algorithm with original A3C and auxiliary learning algorithms that only include resource utilization control (RUC) or resource balance control (RBC), and for fair comparison, all settings (such as state, action, reward, and CNN parameters) are consistent.

End-to-end average packet delay, total deployment cost and the weighted sum of the delay and cost are set as indicators to measure the performance of each algorithm under different amounts of containers. In addition, we compare the performance of A3C and UNREAL-CP in terms of learning efficiency and cumulative reward.

From Fig. 2, we can see that compared to the other three algorithms, UNREAL-CP reduces deployment cost significantly. For example, when each CC contains 5 containers, UNREAL-CP reduces the deployment cost by 28.1%, 39.2% and 51.6% respectively compared to A3C with RUC, RBC and original A3C.

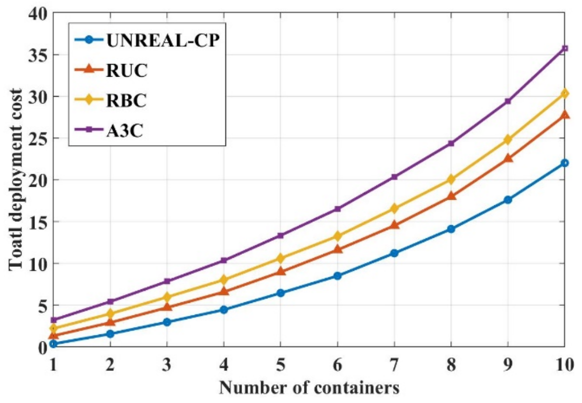


Figure 2: Total deployment cost under different algorithms

Fig. 3 shows the average end-to-end delay under different number of containers. We can see that when the number of containers is small, the delays under the placement strategies obtained by the four algorithms are similar. However, the gap between different algorithms becomes larger when the number of containers increases. This is because the UNREAL-CP algorithm considers the resource balance. When the number of containers in service increases, some overloaded servers will become the bottleneck of the network, affecting the traffic transmission, while balancing network load can effectively avoid such problems.

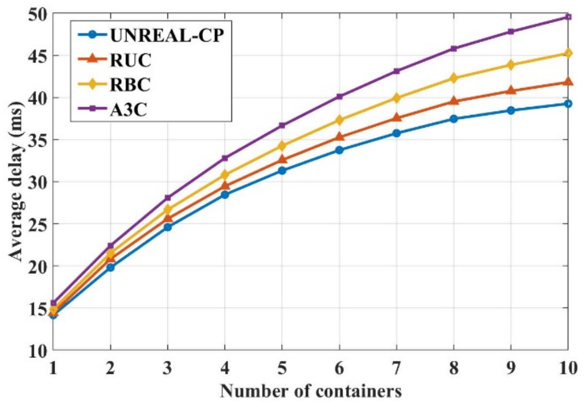


Figure 3: Average delay under different algorithms

Because UNREAL-CP performs better in reducing placement costs and average latency, it finally achieves an average reduction about the sum of cost and delay by 11.9%, 19.1% and 28.6% respectively, which is shown in Fig. 4.

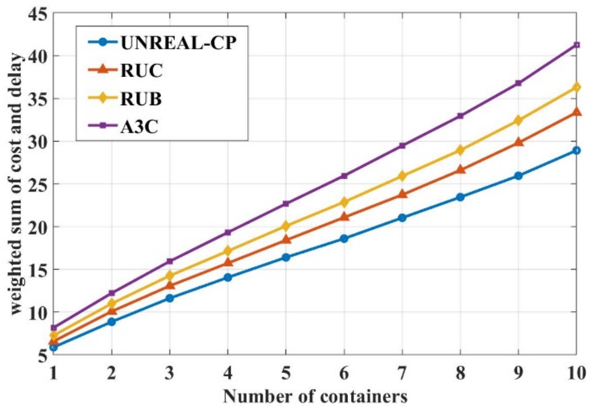
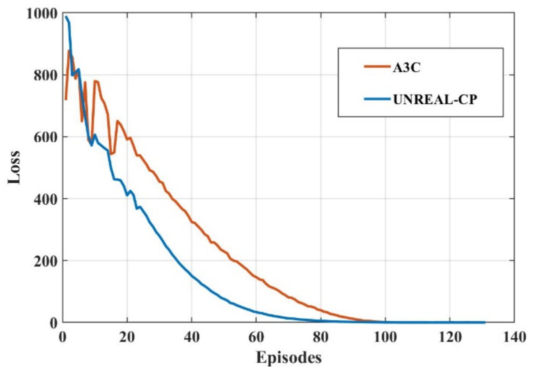
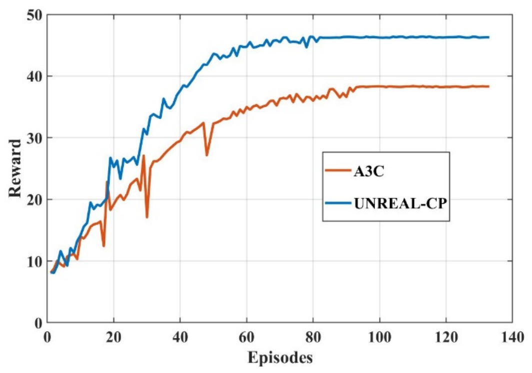


Figure 4: Weighted sum of cost and delay under different algorithms

From Fig. 5, we can observe that UNREAL-CP has better convergence performance. UNREAL-CP reaches a better placement solution with higher reward within 60 episodes, while A3C uses more time and only obtains a local optimal solution. It means that the auxiliary tasks we designed significantly improves the learning efficiency of the agent in container placement problem.



(a) Loss function



(b)Reward

Figure 5: Learning performance of two DRL algorithms

6 Conclusion

In this paper, we present UNREAL-CP, a DRL approach for placing container clusters in cloud, taking deployment cost and average E2E delay into consideration. A3C algorithm architecture is used because it makes decisions under the guidance of both actor and critic's DNNs, and has great advantages in solving dynamic and continuous control problems. Since network load balancing has a significant impact on reducing network latency, we propose two auxiliary tasks, resource utilization control and available resource balance control to improve the convergence performance of A3C. The results of extensive experiments show that the proposed UNREAL-CP algorithm can effectively reduce the deployment cost and E2E delay up to 28.6%. Compared with original A3C, the convergence speed of our algorithm is also improved.

Acknowledgement: Many people have offered me valuable help in my thesis writing, including my classmates, my friends and my parents. They provided me with valuable guidance and suggestions in every stage of the writing of this paper. And their encouragement has sustained me through depression and frustration.

Funding Statement: This work is supported by the National Natural Science Foundation of China (61702048) and the Public Support Platform Construction of Industrial Internet platform.

Conflicts of Interest: We have no conflicts of interest to report regarding the present study.

References

Gu, L.; Chen, X.; Jin, H.; Lu, F. (2018): VNF deployment and flow scheduling in geo-distributed data centers. *IEEE International Conference on Communications*, pp. 1-6.

Huang, X.; Yuan, T.; Qiao, G.; Ren, Y. (2018): Deep reinforcement learning for multimedia traffic control in software defined networking. *IEEE Network*, vol. 32, no. 6, pp. 35-41.

Hussein, M. K.; Mousa, M. H.; Alqarni, M. A. A. (2019): Placement architecture for a container as a service (CaaS) in a cloud environment. *Journal of Cloud Computing*, vol. 8, no. 1, pp.7.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z. et al. (2016): Reinforcement learning with unsupervised auxiliary tasks. arXiv:1611.05397.

Li, Y. X. (2017): Deep reinforcement learning: an overview. Xiv:1701.07274.

Lv, L.; Zhang, Y. C.; Li, Y. S.; Xu, K.; Wang, D. et al. (2019): Communication-aware container placement and reassignment in large-scale internet data centers. *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 540-555.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P. et al. (2016): Asynchronous methods for deep reinforcement learning. arXiv:1602.01783v2.

Piet, S.; Bart, D.; Pieter, S. (2018): Docker layer placement for on-demand provisioning of services on edge clouds. *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1161-1174.

Quang, P. T. A.; Singh, K. D.; Bradai, A.; Benslimane, A. (2018): QAAV: quality of service-aware adaptive allocation of virtual network functions in wireless network. *IEEE International Conference on Communications*, pp. 1-6.

Song, X.; Zhang, X.; Yu, S.; Jiao, S.; Xu, Z. (2017): Resource-efficient virtual network function placement in operator networks. *IEEE Global Communications Conference*, pp. 1-7.

Wei, Y. F.; Wang, Z. Y.; Guo, D.; Yu, F. (2019): Deep Q-learning based computation offloading strategy for mobile edge computing, *Computers, Materials & Continua*, vol. 59, no. 1, pp. 89-104.

Xu, Z. Y.; Tang, J.; Meng, J. S.; Zhang, W. Y.; Wang, Y. Z. et al. (2018): Experience-driven networking: a deep reinforcement learning based approach. *IEEE INFOCOM-IEEE Conference on Computer Communications*, pp. 1871-1879.

Zhang, R.; Chen, Y.; Dong, B.; Tian F.; Zheng, Q. (2019): A genetic algorithm-based energy-efficient container placement strategy in CaaS. *IEEE Access*, vol. 7, pp. 121360-121373.

Zhao, N.; Liang, Y.; Niyato, D.; Pei, Y.; Jiang, Y. (2018): Deep reinforcement learning for user association and resource allocation in heterogeneous networks. *IEEE Global Communications Conference*, pp. 1-6.

Zhou, R.; Li, Z.; Wu, C. (2019): An efficient online placement scheme for cloud container clusters. *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1046-1058.