边缘计算场景下基于强化学习的应用最优部署

马新新，管昕洁，白光伟，糜元根

（南京工业大学 计算机科学与技术学院，江苏 南京 211816）

摘 要：为有效解决城市范围内智能公共交通应用程序的布局问题，制定总代价最小化的应用布局优化策略 MIN-COST，以降低应用程序部署的总代价为目标，同时满足应用程序服务延时要求。通过提出一个基于深度强化学习技术优化公交边缘应用程序部署的一般框架，可以从历史经验中学习到最优化部署方法，相对于一般启发式算法更加快速。将仿真结果与其它部署策略进行比较，验证了所提策略可以在保证服务时延的基础上有效降低应用程序服务总代价。

关键词：移动边缘计算；资源分配；容器技术；应用程序部署；强化学习

中图法分类号：TP391.9 **文献标识号**：A **文章编号**：1000-7024（2021）01-0015-09

doi：10.16208/j.issn1000-7024.2021.01.003

Optimal deployment of applications based on reinforcement learning in edge computing scenarios

MA Xin-xin, GUAN Xin-jie, BAI Guang-wei, MI Yuan-gen

(College of Computer Science and Technology, Nanjing Tech University, Nanjing 211816, China)

Abstract: To solve the layout problem of intelligent public transportation applications in the city effectively, an application layout optimization strategy named MIN-COST with a minimum total cost was developed. This strategy targeted the overall cost of reducing application deployment while meeting application service latency requirements. By proposing a general framework for optimizing the deployment of public transport edge applications based on deep reinforcement learning techniques, it was possible to learn from the historical experience to optimize deployment methods, which was faster than general heuristic algorithms. Comparing its simulation results with that of other deployment strategies, it is verifies that the proposed strategy can effectively reduce the total cost of application services based on the guaranteed service delay.

Key words: mobile edge computing; resource allocation; containers; application deployment; reinforcement learning

0 引 言

现在交通场景中，智能辅助驾驶应用程序不断涌现，此类应用通常对服务时延要求极为苛刻，由于路况瞬息万变，过高服务延迟将会对用户造成很大影响。究其高服务时延的原因主要是通信时延，此外在服务器端的处理时延[1]，以及如今各式各样的终端设备通过核心网络传输海量数据，信道的拥堵、服务器的任务等待造成的服务等待时延，种种方面的原因都提高了通信的时延。

而传统云计算架构将很难满足低延迟应用程序的需求，为缩减网络延迟，移动边缘计算（MEC）作为一种新的网络架构用来解决时延问题。对 MEC 的研究一直是热点，尤其是边缘服务器的选择一直是研究的焦点问题，但现有研究多从节能[2]、高效负载[3]、服务器放置[4]着手，仅少量研究考虑了时延敏感应用程序的使用场景，特别是在快速服务尤为重要的交通应用程序中。

考虑到上述的问题，本文聚焦于城市范围内的智能公共交通场景下的智能应用部署，如自动报站系统、车载安全监控系统等。本文设计了一种最小化应用服务时延的模型与部署策略，并提出一种基于深度强化学习（DQN）的

收稿日期：2019-09-17；**修订日期**：2019-12-02

基金项目：国家自然科学基金项目（61802176、61602235）；江苏省自然科学基金项目（BK20161007）

作者简介：马新新（1994-），男，江苏连云港人，硕士研究生，研究方向为网络优化、边缘计算；管昕洁（1984-），女，江苏南京人，博士，助理教授，研究方向为网络优化、边缘计算、软件定义网络等；白光伟（1961-），男，河北唐山人，博士，教授，博士生导师，CCF 杰出会员，研究方向为网络计算、移动互联网、边缘计算、计算机网络等；糜元根（1962-），男，江苏无锡人，硕士，副教授，硕士生导师，研究方向为人工智能、边缘计算、计算机网络等。E-mail: mxx_cs@njtech.edu.cn

方法求解最优部署方案，能有效满足交通场景下应用对低时延严苛的要求。

1 相关介绍

本文为能最大化减小应用的服务时延，采用了 Docker 容器和微服务框架相组合的新架构，此外，从应用部署的实例化过程着手，尽可能减少来自底层部署的时延。在方案求解使用了近年来火热的深度强化学习方法，相对于一般启发式算法有着更快的求解速度。下文对相关知识做出介绍。

1.1 应用程序实例化过程

用户使用程序服务会通过用户终端将服务请求提交至边缘服务器，服务器收到请求后作出若干部署决策，后对应用程序进行部署。考虑到节能、降低成本等实际情况，运营商将部分服务器设置为低功耗睡眠状态。当应用程序部署所在服务器处于睡眠状态，部署操作将其唤醒并进入正常待机状态，随后相关服务器为新虚拟机搭建相关运行环境。搭建过程包括诸如下载、提取（extracting）系统文件、库文件，在容器中此类文件称为镜像。运行环境搭建完成后应用程序即可正常运行，期间服务端可能会与终端进行数据传递。过程如图 1 所示。

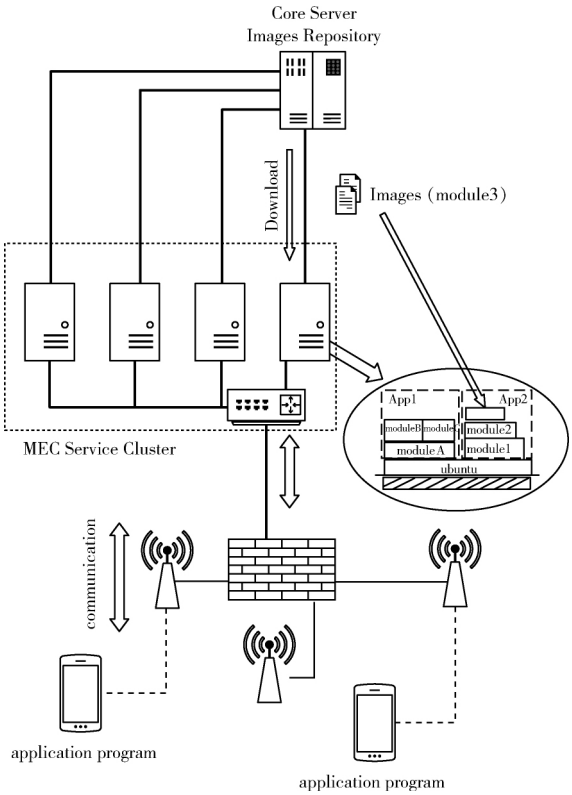

图 1 应用程序实例化过程

1.2 实例化中的镜像重复下载

系统周期进行镜像快照（snapshot）操作会占用高额存储空间，为节省空间，在服务结束后系统将移除服务镜像。而在一个服务器上反复运行同一个程序或同类型程序时，所需的基础镜像几乎相同，每次运行时的重新下载变为额外的下载代价。如多个应用程序均在 ubuntu 上运行，一个 ubuntu 15.10 的镜像文件大约有 51 MB，下载、提取此文件需要数秒时间，期间用户不能进行任何操作。而利用镜像文件重用的方式，仅首个应用程序会有一定下载时延，之后的应用程序则可直接使用已有镜像文件，再下载剩余所需文件即可补足运行环境，此方式避免了重复下载。

1.3 微服务架构的优势

微服务架构提倡将单一应用程序作为多种服务的组合，不同服务功能可以通过不同语言编写、使用不用数据存储技术，并能独立部署。微服务架构的优势众多，一方面微服务组件化的结构可以分散、灵活地部署在不同物理机上，每一个模块都可以独立运行；另一方面，整体应用程序由于其高耦合性，程序全部功能整合于一体，只有将各功能所需环境全部部署完毕才能运行程序。此外某一个功能如果需要更新，则要将整个程序代码重新部署。而在微服务框架下，每个组件仅需下载自身所需的运行环境，相对于整体部署则大大降低了部署代价。例如：一个应用程序此时需要迁移 3 次，整体程序需要将完整代码和环境部署 3 次。而在微服务框架下，只迁移当前所需的服务模块而并不处理其余模块，所部署的代码和环境文件数量远低于 3 次整体程序的总数量。

1.4 微服务应用程序的构成

本文中应用程序由多个服务子任务组成，将其视为一条串行虚拟服务链。如：在图像渲染服务[9]中，将整个工作流程分为顶点处理（vertex process）、裁剪和原始组装（clipping and primitively assembling）、光栅化（rasterizing）、片段处理（fragment processing），各部分服务依次进行，每一阶段服务的处理结果都将作为输入送入下阶段服务中。将程序分解为不同功能服务模块后，各功能模块可运行在不同服务器上，增强了程序部署和维护的灵活性。需要注意的是，服务模块并不全都在服务器上运行，如人脸识别服务中，第一步的拍摄功能在终端设备上完成，捕获图像后才将数据传入服务器进行处理。为支持智能公交系统，本文假设将部署多套应用程序，同时为了提高资源使用效率，所有的应用程序都使用微服务架构。

1.5 代价的定义

为将实际中时延和能耗统一简化计算，本文将其抽象为一系列代价。MEC 提倡将应用程序托管至离用户较近的小型云服务器中，从地理上缩短应用程序终端和服务端之间的通信距离，由此降低所用服务器与终端之间的通信代价（communication cost），并减轻核心网络中的拥塞情况。通信时延在本文内被抽象为通信代价，降低通信代价可视作减小通信时延。另外值得注意的是，随着用户的不断移

动，用户与托管应用程序的服务器距离可能会不断变化，由此产生的通信代价变化可能导致服务器切换等情况，此情况下会产生高额的迁移代价[10]（migration cost），另外应用程序迁移至新服务器上可能产生额外的部署代价（deployment cost）。同时在服务器运行时也包括多种代价：从选中指定服务器开始，包括正常运行时的基线能耗（运行代价），也包括诸如：唤醒休眠服务器、下载镜像文件等部署代价。

1.6 深度强化学习（DQN）

实际环境中并不能获得完整的状态转移矩阵，根据bellman方程（公式）将值函数（回报）分解为两个部分：

（1）立即可得的回报；

（2）之前回报的折扣值。

可以将此等式看作状态值函数和行为值函数自身以及相互之间的递推关系

$$V(s) = max_a(R(s,a) + \gamma V(s')) \tag{1}$$

Bellman方程是强化学习的理论基础，状态值函数和行为值函数的递推关系也正是获得未来信息的一种方式。然而在实际应用中，状态以及动作的维度往往很大，仅仅用传统的数据结构难以存储如此巨大的信息，而深度神经网络的出现也让强化学习的方法再次受到关注。在强化学习的基础上，深度神经网络的增加使得此学习方法获得以下优势：

（1）神经网络在高度结构化的特征提取方面表现优异，使得强化学习可以直接从高维原始数据学习控制；

（2）传统强化学习的Q-table有着规模限制，很多问题在现实世界里无法遍历全部状态，而神经网络可以根据先前经验直接生成所需价值函数。

综合以上问题、方法，本文提出了一种综合考虑了部署代价、通信代价以及迁移代价的新型应用布局策略MIN-COST。布局模型基于微服务架构和容器技术，将应用程序以灵活的模块化方式部署在边缘服务器中，同时利用容器技术的轻量化部署方式满足低时延需求的交通应用程序，而达到降低总代价、减少用户等待时延目的。在使用基于DQN的方法后可以快速获得全局最优布局方案，减少了使用一般算法的计算用时并且不受限于问题规模的扩张。

2 最优应用部署策略（MIN-COST）

为最大化降低边缘计算场景下的智能交通应用从部署到服务的整体时延，我们提出了一种基于微服务框架的应用程序并应用于Docker容器的最优应用部署策略（MIN-COST），下文将对此策略做详细介绍。

2.1 建模目标

本文将智能公交应用程序布局问题建模为最小化总代价的优化问题。优化目标是最小化由部署代价、通信代价、迁移代价组合成的总布局代价，具体体现在：①尽可能减少活动物理机器的数量；②提高镜像文件重用；③缩短应用程序副本与用户之间的通信距离；以及，④减少由用户移动引起的服务迁移次数。

2.2 相关参数

假设一个由 n 个MEC计算集群覆盖的城市，这些集群为城市的公交系统提供计算服务，其中每个集群中有若干物理机，用一个 N_i 表示某一个具体的物理机。简单起见，认为全部物理机有相同的计算能力，但每个物理机拥有计算资源各不相同，即每台物理机处理任务速度相同但支持运行任务数量不尽相同。所设公交系统中，有 k 个不同的应用程序（$task$），进一步假设每个应用程序 $A_k \in task$ 可以被视作一条虚拟微服务链 S_k，虚拟微服务链 $S_k = \{\alpha_1, \alpha_2, \alpha_3 \cdots\}$ 由多个功能模块组成，每个微服务模块 $\alpha_d \in S_k$，$d \in N$ 可以具有多个副本，这些副本在不同的物理机上通过容器部署并以并行的方式执行不相交的子任务（不同的功能模块任务）。这里每个微服务被表示为元组 $(\tau_\alpha, L_{\alpha,l}, R_{m,\alpha})$，其中，$\tau_\alpha$ 指定微服务 α 的最大允许服务时延。$L_{\alpha,l}$ 表示用于部署微服务 α 所需支持镜像集。$R_{m,\alpha}$ 表示微服务 α 总计算量。本文涉及其它参数由表1列出。

表1 相关参数

符号	定义
i,j	某一物理机序号，有 $i \in PM$ 或 $j \in PM$
m	某一具体用户，有 $m \in user$
t	时刻的集合，有 $t=\{0,1,\cdots,T\}, T \in N$
α,β	某一微服务的序号，有如：$task=\{\alpha,\beta,\cdots\}$
τ_α	微服务 α 的最大允许服务时延
l	某一被使用的镜像序号
ε_l	下载 l 镜像文件的代价
Δ	计算量化成代价的单位，为常量
S_0	虚拟微服务链的首个微服务
S_d	虚拟微服务链的最后一个微服务
C_b	物理机运行状态必须的基线代价
C_a	物理机从休眠状态切换至运行状态的代价
$R_{m,\alpha}$	用户 m 的微服务 α 所需的计算量
R_i	物理机 i 上的最大可用计算资源
$d(i,j)$	物理机 i、j 之间的距离（视为 rtt）
$u(i,m(t))$	在 t 时刻物理机 i 和用户 m 的通信距离
$f(\alpha,\beta)$	微服务 α 与 β 间的通信量
$m(t)$	用户 m 在 t 时刻的位置
$L\alpha$	微服务 α 所需要支持的镜像 l

2.3 相关变量

模型中有以下3种变量：$x(i,t)$ 用来判断物理机 i 在 t

时刻状态，如果物理机 i 在此刻为工作状态则 $x(i,t)=1$，否则为 0；$y(i,t,m,\alpha)$ 用来判断在 t 时刻，用户 m 应用程序中的微服务 α 是否映射到物理机 i 上，如果已被映射则 $y(i,t,m,\alpha)=1$，否则为 0；$L(i,t,l)$ 判断 t 时刻物理机 i 上是否有镜像 l，如果存在则 $L(i,t,l)=1$，否则为 0。通过此 3 个变量，可以获得在某一个时刻的全部物理机工作状态，即一台正常运行的物理机上已下载的支持镜像文件和正在运行的应用程序模块。

2.4 模型分析

将整体优化模型表述为式（2）

$$COST = C_{op} + C_{com} + C_{mig} \tag{2}$$

总代价由 3 个部分构成，分别为部署代价、通信代价和迁移代价。

部署代价（C_{op}）：

部署代价被描述为一个新建容器在物理机上所造成的一系列能耗，其中包括了以下各种能耗代价：

（1）下载所需环境的能耗，此过程包括下载、提取镜像文件所用时间，以及产生的额外能耗，此代价用 ε_l 表示；

（2）唤醒已睡眠物理机的能耗，此过程包括休眠物理机启动用时，以及休眠转化为正常运行的能耗，此代价用 C_a 表示；

（3）物理机的基线能耗，即物理机正常运行时所必要的能耗，此代价用 C_b 表示；

（4）物理机运算时的额外能耗，由于处理用户应用程序服务，物理机需要提供额外的计算资源，也需提供额外的计算能耗，此代价用 $R_{m,\alpha} * \Delta$ 表示。

部署代价由式（3）表示出

$$C_{op} = \sum_{t=0}^{T}\sum_{i}\sum_{m}\sum_{\alpha} x(i,t)\times C_b + y(i,t,m,\alpha)\times R_{m,\alpha}\times\Delta + \sum_{t=0}^{T}\sum_{l}[(L(i,t,l)-L(i,(t-1),l)\times\varepsilon_l] + C_a \tag{3}$$

式（3）可分为 4 部分代价的累加：第一部分为服务器正常运行时的基线代价；第二部分为服务器处理用户应用程序服务的计算产生的额外代价；第三部分为下载所需镜像的下载代价；第四部分为处于休眠服务器被唤醒的启动代价。

通信代价（C_{com}）：迁移能耗被描述为应用程序不同微服务间的必要信息传输，体现在串行服务间上一层的输出传到下一层并作为输入再次进行计算的过程。同时还考虑到虚拟微服务链中首尾服务和用户的通信。通信代价由式（4）表示出

$$C_{com} = \sum_{t}\sum_{i,j}\sum_{m}\sum_{\alpha,\beta} y(i,t,m,\alpha)\times y(j,t,m,\beta)\times d(i,j)\times f(\alpha,\beta) + \sum_{t}\sum_{i,j}\sum_{m}\sum_{\alpha,\beta} y(i,t,m,S_0)\times u(i,m(t))\times f(m,S_0) + \sum_{t}\sum_{i,j}\sum_{m}\sum_{\alpha,\beta} y(i,t,m,S_d)\times u(i,m(t))\times f(m,S_d) \tag{4}$$

式（4）中由 3 部分累加所得，第一部分为同一应用程序在两个服务器之间的数据迁移的通信代价；第二部分为用户终端上传应用程序数据的通信代价；第三部分为用户终端接受应用程序数据的通信代价。

其中 $u(i,m(t))$ 是控制用户应用程序放置位置的关键参数，将每一时刻用户于全部物理机的通信距离都记录，实验时此记录作为输入数据进入模型计算。

迁移代价（C_{mig}）：迁移能耗被描述为容器信息在当前物理机中复制到其它物理机上所用的能耗，这部分代价主要与物理机之间通信距离有关，在同一集群的物理机之间迁移会比跨集群的物理机之间迁移代价要小很多[11]。迁移代价由式（5）表示出

$$C_{mig} = \sum_{t=1}^{T}\sum_{i,j}\sum_{m}\sum_{\alpha} x(i,(t-1),m,\alpha)\times y(j,t,m,\alpha)\times d(i,j) \tag{5}$$

式（5）中只有两个变量 $x(i,(t-1),m,\alpha)$ 和 $y(j,t,m,\alpha)$ 均赋值为 1 时才有迁移代价，即在（$t-1$）时刻用户 m 的微服务 α 运行在物理机 i 上，在下一个时刻 t 此微服务 α 运行在物理机 j 上，则说明微服务进行了迁移。

优化目标和约束条件：将系统模型结果最小化获得最终优化目标。优化目标由式（6）表示

$$min\{C_{op} + C_{com} + C_{mig}\} \tag{6}$$

此外，优化模型包括若干约束条件（s. t.），具体如下：

约束 1：保证已放置微服务模块的物理机是开启的，即在时刻 t 时，当物理机 i 已经成功运行微服务 α 时，物理机 i 必定是处于工作状态。

s. t. 1

$$x(i,t) \geqslant y(i,t,m,\alpha), \forall i,t,m,\alpha$$

约束 2：保证镜像文件的正确表示，变量 $L(i,t,l)$ 需始终大于等于右边公式，即在 t 时刻，当物理机已经成功运行微服务 α 时，必有镜像文件 l 存在。

s. t. 2

$$L(i,t,l) \geqslant y(i,t,m,\alpha)\times L_{\alpha,l}, \forall i,t,m,\alpha$$

约束 3：保证每一个物理机上全部微服务模块所需资源量不会超过此物理机的最大资源量。本文中物理机资源设置为内存、CPU。

s. t. 3

$$\sum_{m}\sum_{\alpha} y(i,t,m,\alpha)\times R_{m,\alpha} \leqslant R_i, \forall i,t$$

约束 4：保证每个微服务模块的唯一性，即一个时刻内某一个微服务模块只能存在于一个物理机上。为简易化模型，本文中我们假定每个微服务模块为最小运行单位。

s. t. 4

$$\sum_{i} y(i,t,m,\alpha) = 1, \forall t,m,\alpha$$

约束 5～约束 7：全部变量均为 0-1 变量，即赋值为 1 时，依次表示为微服务存在，处于工作状态、镜像文件存

在，反之不存在

s. t. 5

$$y(i,t,m,\alpha) \in \{0,1\}$$

s. t. 6

$$x(i,t) \in \{0,1\}$$

s. t. 7

$$L(i,t,l) \in \{0,1\}$$

约束 8：用户应用程序的全部服务模块运行时延不能高于阈值，需要严格遵守 QoS（quality of service）内时延要求。

s. t. 8

$$\sum_{task} \tau_\alpha \leqslant \tau_{QoS}$$

本文根据以上优化公式和约束条件，将其作为优化模型，并设计算法实现 MIN-COST 策略的目标。

3 基于深度强化学习的求解方法

服务器迁移问题是典型的组合优化问题，而此类问题往往都是 NP-hard 问题，使用传统求解器如 CPLEX 只能求解一定规模的问题，而随着问题规模指数增长，求解器的功能则变得乏力。启发式算法用于求解 NP-hard 问题最为广泛，但由于缺乏理论支撑，算法设计需要大量的专业知识和试错。本文使用了一种基于深度强化学习（DQN）的方法来求解组合优化问题，一方面强化学习算法不受限于问题规模，另一方面避免了启发式算法繁琐的算法设计，让计算机自主学习问题的信息并解答。

3.1 学习模型设置

此节我们定义了学习模型的必要参数。在 2.2 小节中已经定义了有 n 个边缘计算集群，每个计算集群中的物理机记为 N_i，以及一段时间 $t=\{0,1,\cdots,T\}, T\in N$。$m(t)$ 定义为用户位置。小节 2.3 中，$y(i,t,m,\alpha)$ 定义为用户应用部署在物理机 N_i 上。

强化学习模型中的元素定义如下：

Agent：集群中的控制器作为强化学习中的 agent，由于控制器具备和全部物理机通信的功能，并且掌握各物理机的状态和部署情况，本文中由控制服务器做出任务的部署和迁移决策。

States：在不同时刻 $t\in T$ 的状态表示为 S_t，由于路径可以根据历史数据获得，本文将用户当前坐标作为输入状态，用户位置的移动会导致与各服务器的相关代价改变。states 定义为 $S_t = m(t)$。

Actions：在本文中，控制器的决策域的大小与全部边缘服务器数量成正比，当前应用所在服务器需要与全部服务器一一对应，具体表示为：$A_t \in Act = \{A^{i,j}\}, i,j \in PM$，其中 i、j 分别表示为当前服务器和下一个迁移服务器，如果不迁移则有 $j=i$。值得注意的是，本文依据 POMDP 的思想缩小了决策域，对于用户距离过大的服务器从决策中排除，简化了实验过程。

Reward：控制器根据动作和状态获得当前时刻 t 的奖励。文中我们设置一个初始奖励 R，需要注意的是，由于每台物理机的当前状况并不相同，需要根据式（3）先行扣除部分奖励。上文所提出的通信式（4）、迁移代价式（5）在奖励中均为惩罚，也就是所用代价越大，获得的奖励越小，所以最终奖励设置为

$$R_t(S_t,A_t) = R - C_{op} - C_{com} - C_{mig} \tag{7}$$

上式根据 action 的情况写成如下公式

$$R_t(S_t,A_t) = \begin{cases} R - C_{op} - C_{com} - C_{mig}, & if\ A_t = A^{i,j} \\ R - C_{op} - C_{com}, & if\ A_t = A^{i,i} \end{cases} \tag{8}$$

在表 2 中列出了强化学习中的全部参数。

表 2 相关参数

符号	定义
N_i	某一物理机序号，有 $i\in PM$
m	某一具体用户，有 $m\in user$
t	时刻的集合，有 $t=\{0,1,\cdots,T\}, T\in N$
S_t	在时刻 t 时的状态
A_t	在时刻 t 时的动作
Act	动作的集合
$A^{i,j}$	$Station$：应用程序从服务器 i 迁移到服务器 j
$R_t(S_t,A_t)$	在 t 时刻控制器获得的奖励
$m(t)$	用户 m 在 t 时刻的位置
$\Phi_t(S_t)$	在 t 时刻控制器从状态 S_t 到一个动作的映射
Φ_t^*	在 t 时刻从状态 S_t 到一个动作的最优映射
Q_t	Q 函数
$Q_t(S_t,A_t;\theta)$	在 DQN 中带参数 θ 的 Q 函数
D	$Replay$ 的存储容量
π	控制器的决策
Π	控制器的决策集合
γ	折扣率
α	学习率（区别表 1 中 α）
ε	探索与利用的权衡参数

3.2 基于 DQN 的应用部署算法

控制器在 t 的全部时刻选择合适的策略来最大化总的奖励值。控制器关于时间 t 序列的决策定义如下

$$\pi = \Phi_t(S_t) \quad \forall\, t = \{0,1,\cdots,T\}, T \in N \tag{9}$$

在决策 π 中，有 $\pi\in\Pi$，当在 t 时刻选择了策略 π，则有 S_t^π。如果在 t 时刻选择的是最优映射，则有 $\pi^* = \Phi_t^*$，获得总奖励策略集合的表示如下

$$\max E_s^\pi = \sum_{t=1}^{T} \gamma R_t(S_t^\pi, A_t) \quad \pi \in \Pi, T \in N \tag{10}$$

折扣率 γ 一定程度上保留了历史经验中的状态效用的最大值，当 γ 越大控制器越会重视以往经验（far-sighted），越小则就只重视眼前的奖励（myopic），有 $\gamma \in (0,1)$。DQN 算法使用 bellman 等式递归并更新自身参数

$$Q_t^*(S_t,A_t) = E_{s_{t+1}}[R_t(S_t,A_t)+\gamma \max_{A_{t+1}} Q_t(S_{t+1},A_{t+1}) \mid S_t,A_t] \quad (11)$$

Q 函数的最优值 $Q_t{}^*(S_t, A_t)$ 即为选择的最优策略，在一般强化学习中 Q 函数值由列表保存，一般称为 Q-table，最优值 Q^* 代表在状态 S_t 下可以获得最多回报的 action。Q 函数表示为

$$\Phi_t^* = arg \max_{A_t \in ACT} Q_t^*(S_t,A_t) \quad (12)$$

在 DQN 中，获得的近似 Q 函数用 $Q_t(S_t,A_t;\theta)$ 表示，θ 是权重参数。由于 DQN 中 Q 值直接由神经网络生成，替代了传统 Q-table 的保存方法，神经网络中每个神经元的 weight 和 bias 参数都影响着 Q 值的变化。同传统强化学习一样，最优策略表示为

$$\Phi_t^* = arg \max_{A_t \in ACT} Q_t^*(S_t,A_t;\theta) \quad (13)$$

在 DQN 算法中目标值（Q 值）为

$$Q_{target} = R_t(S_t,A_t)+\gamma \max_{A_{t+1}} Q_t(S_{t+1},A_{t+1};\theta_i^-) \quad (14)$$

θ_i^- 表示上一次迭代的参数。

为使网络能最终收敛，需要计算两个网络的差距（MSE）来找到参数更新的方向

$$L_i = Q_{target} - Q_{eval} \quad (15)$$

更为详细的损失函数表示如下

$$L_i(\theta_i) = E_{S_t,A_t,R_t,S_{t+1}}[(Q_{target} - Q_t(S_t,A_t;\theta_i))^2] \quad (16)$$

微分上式获得损失函数的梯度

$$\nabla_{\theta_i} L_i(\theta_i) = E_{S_t,A_t,R_t,S_{t+1}}[(Q_{target} - Q_t(S_t,A_t;\theta_i))\nabla_{\theta_i} Q_t(S_t,A_t;\theta_i)] \quad (17)$$

$\nabla_{\theta_i} Q_t(S_t,A_t;\theta_i)$ 指导式（16）在可行方向上减小，并通过反向传播更新 $Loss$ 函数权重，使得 $Loss$ 函数达到可能的最小值，最后更新相关参数完成此次训练。反向传播如下式

$$\theta_{i+1} = \theta_{i+1} + \alpha \nabla_{\theta_i} L_i(\theta_i) \quad (18)$$

α 为学习效率，且有 $\alpha \in (0,1)$。

此外 DQN 中使用了 experience replay 的方法，即保存之前的学习经历在更新算法时随机抽取再次学习，这种方法打乱了学习经验的相关性，克服了经验数据非平稳分布的问题。

不同于 Q-learning 算法，DQN 增加的 fixed Q-targets 方法打破数据关联性，稳定了学习过程。DQN 使用两个结构相同的神经网络，分别为 target 和 evaluation。evaluation 网络随训练步数实时更新而 target 网络则是隔步用 evaluation 网络直接替换。式（15）、式（16）正是计算两个网络之差，称之为 TD-error。

算法 1 表示了本文所提出的基于 DQN 的应用程序优化部署算法。算法流程如下：

算法 1：基于 DQN 的最优服务器选择

输入：折扣率 γ，学习率 α，探索与利用的权衡参数 ε，Replay 的存储容量 D，更新步数 c。

输出：最优服务器位置

(1)使用存储容量 D 初始化 Replay memory 的容量。

(2)使用参数 θ 初始化评估网络。

(3)使用参数 $\theta^- = \theta$ 初始化目标网络。

(4) 初始化全部服务器信息(资源量、坐标、库文件数量等)。

(5)$t \leftarrow 1$，$m(1) \leftarrow$ random value，$i \leftarrow$ random PM

(6)设置 $S_1 = m(1)$

(7) 将全部服务器以到用户距离为优先级排序，选择前 10 个服务器作为备选服务器。

(8)**while** $t \leqslant T$：

(9) 生成随机数值 $\eta \in [0,1]$

(10) **if** $\eta < \varepsilon$：

(11) 随机选择一个动作 A_t；

(12) **else**：

(13) 选择 $A_t = \Phi_t^* = arg \max_{A_t \in ACT} Q_t^*(S_t,A_t;\theta)$，$Q$ 是神经网络生成的最优值。

(14) 判断 $x(i,(t-1),m,\alpha)$，$y(j,t,m,\alpha)$ 是否同值：

(15) 分别根据式(8)计算 $R_t(S_t,A_t)$

(16) $i \leftarrow j, S_{t+} \leftarrow m(t+1)$

(17) 将此次经验(S_t，A_t，R_t，S_{t+1})存储在 Replay memory 中

(18) 运行算法 2 更新估值网络

(19) **if** $t \ \% \ c = 0$：

(20) 设置 $\theta^- = \theta$

(21) $t \leftarrow t+1$

(22) **end while**

算法 2：更新估值网络

(1) 采用 minibatch 的方式从 Replay memory 中随机读取先前经验(S_t，A_t，R_t，S_{t+1})，$k \in T$

(2) **if** $t+1$ 是最后一个循环：

(3) 赋值 $Q_{target}^t = R_t$

(4) **else**

(5) 赋值 $Q_{target}^t = R_t(S_t,A_t)+\gamma \max_{A_{t+1}} Q_t(S_{t+1},A_{t+1};\theta_t^-)$

(6) **end if**

(7) 使用 MSE 式（16）、式（17）计算下降梯度，根据式（18）更新估值网络权重 θ

至此一个完整的 DQN 构建完成，通过输入数据集可以训练出能自动选择最优部署位置的决策模型，具体实验在

下节给出。

4 实验仿真

为测试所提的 MIN-COST 策略的有效性，本文与 SABFD（space aware best fit decreasing）[12]、ITEM（ITerative expansion move）[13]和 DF（distance first）3 种策略进行对比实验，SABFD 策略主要关注了服务节能问题，强调尽可能将服务放置在资源充足的服务器上。ITEM 将多个优化部分组合成一个模型并求解最小代价。DF 强调将应用程序服务放置在最靠近用户的边缘服务器上，是一种被广泛使用的部署策略。实验先按照已有研究[13]的实验数据范围生成一个小型的仿真场景，考察小型场景与实际场景的相似度。并针对不同实验场景：不断增加资源量、不断增加用户人数、不断增加服务时间分别测试上述几种部署策略，将 MIN-COST 策略中镜像重用特征与其它策略进行对比，最后给出了使用 DQN 在不同超参数的收敛速度。

4.1 实验设置

实验参照文献［14］将场景设置在 4＊4 的格子中，在此区域中有若干边缘服务器，用户在区域中随机行走（保留其路径）。实验中假定共有 10 种不同的基础镜像文件，不同的微服务模块需要种类不同、数量不一的镜像文件，此需求数值均为随机生成。物理机待机/工作能耗代价、通讯相关固定参数、时延阈值等均沿用文献［15］中实验参数设置。其它参数和用户、物理机状态参数均使用随机数据，数据控制在文献［15］实验数据大致相同的范围内。

剩余参数设置[16,17]如下：reward 中最大回报 $R=90$，学习率 $\alpha=0.01$，折扣率 $\gamma=0.9$，探索与利用的权衡参数 $\varepsilon=0.1$，$Replay\ memory=500$，批大小 $batch\_size=32$，总时间 $t=300$。实验使用了两层、每层 30 个神经元的神经网络[18]，每层的 $weight$ 使用范围为［0，0.3］的随机数值，$bias$ 初始值为 0.1。

为证实所提的 MIN-COST 策略有效性，将本策略与 SABFD、ITEM 和 DF 这 3 种策略在相同实验环境和参数下进行对比实验。实验将 3 种参数变化对策略的影响进行测试，实验安排如下：

（1）为测试不断增加计算资源量情况下 MIN-COST 策略自适应调整状态的效果，实验设置如下：控制时间段、用户数量改变物理机数量，物理机数量控制在［80，240］内；

（2）实际中可能有不断增多的相同应用程序请求的情况，比如使用相同应用程序的用户增多，为测试 MIN-COST 策略在此情况下的自适应调整效果，实验设置如下：控制时间段、物理机数量改变用户数量，用户数量控制在［1，50］内；

（3）为测试用户请求更长服务时间 MIN-COST 策略的调整情况，实验中将用户使用服务时间不断延长，记录该策略部署的变化。实验设置如下：控制物理机数量、用户数量改变总时间，时间长度控制在［180，480］内。

实验结果均使用相同数据在不同的策略下进行计算获得。随后对比了不同策略镜像文件的下载次数，以及相关超参数对算法收敛的影响。

4.2 结果分析与讨论

4.2.1 计算资源量对策略的影响

改变物理机数量的实验结果如图 2 所示，所提出的 MIN-COST 策略相比其它 3 种有较低的代价。由于不断增加新的物理机，出现资源量充足或初始拥有多种镜像文件的物理机概率会提高，所以有可能随着物理机的增多而使得代价降低，实验结果也证实了此猜测。而 DF 策略只考虑距离用户最近的物理机，若新增物理机距离都不及已有物理机的距离，该策略将忽视新增物理机，则无论新物理机的资源量和其它条件更优也都不会对代价产生有利的影响。反而若新增物理机距离近但资源不足或镜像文件不足，将会产生迁移和下载代价，总的代价反而会因此上升。SABFD 策略同理，该策略只考虑寻找资源最充足的物理机，忽视关键的用户与物理机间的通信代价，也很有可能会造成总代价的大幅上升。ITEM 策略相比于上两种策略有较好的提升，但缺少镜像重用的考虑，代价仍高于 MIN-COST 策略。

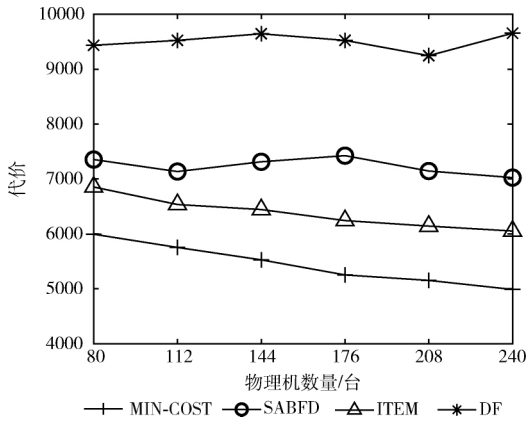

图 2 计算资源量对策略的影响

4.2.2 应用程序请求数量对策略的影响

随着用户的增多，物理机的资源量可能无法满足新用户的请求，用户服务便会被分配至次优的服务器上运行，所以在图 3 中，实验的 4 种策略的总代价都随着用户的增加而增加。MIN-COST 策略中由于镜像的重用，总代价仍低于其它策略。实验假设共有 10 种镜像文件，不同的微服务模块需要不同的镜像文件。由于用户放置完微服务模块后的镜像文件是保留在物理机上的，当下一个或多个用户将微服务模块放置此物理机上时，有很大的概率会重用之

前的镜像文件，随着用户的增多，某一个资源量很足的物理机上甚至可能存在全部 10 种镜像文件，新来的用户微服务模块可以直接运行，避免了镜像文件重复下载的过程。其它的策略并没有考虑到此问题，所以相比 MIN-COST 策略有更高的代价。由于 50 个用户的服务总代价远大于单个用户的总代价，导致图中 4 种策略看似在相同的起点，实际上在单用户时 4 种策略有明显的代价差异。

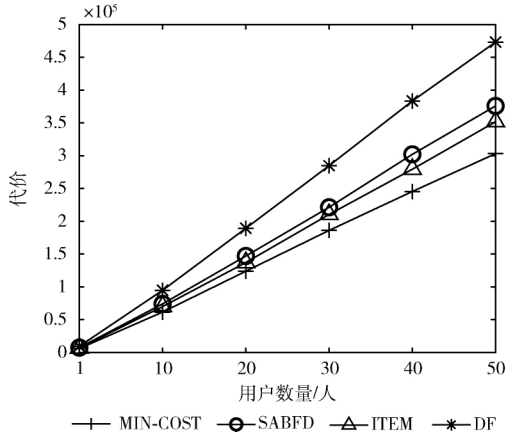

图 3 用户数量对策略的影响

4.2.3 服务时间对策略的影响

随着用户使用微服务模块时间增加，全部的策略代价都有所增加，如图 4 所示，MIN-COST 策略代价总代价仍小于其余 3 种策略。值得注意的是，DF 策略在 360 至 420 的区间有下降趋势，由于用户的移动轨迹是随机生成的，当用户的位置很靠近某个物理机，且此物理机恰好资源足够，有较多的镜像文件时，DF 策略的代价可能会有所减小；当用户的位置很靠近一个资源量最充足的物理机时，SABFD 策略由于降低了用户与此物理机的通信距离，也会产生较小的代价。然而这两种策略的性能具有很强的偶然性，不适合直接作为应用程序部署的优化策略。

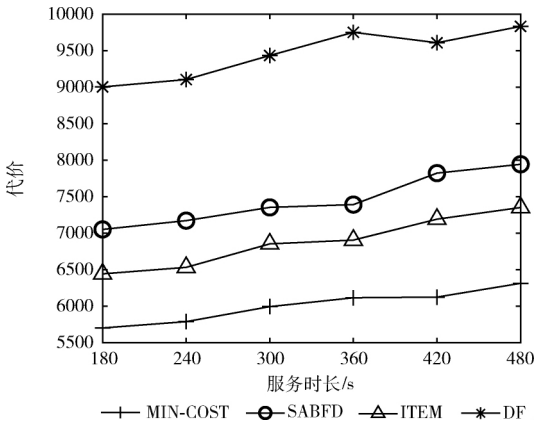

图 4 服务时间对策略的影响

4.2.4 镜像文件下载次数对比

为测试所提的 MIN-COST 策略在 3 组实验情况下的表现，我们记录了每组实验中的镜像下载数量，介于篇幅限制只展示其中一组实验结果，图中 L1、L2…L10 为假设的库文件。在实验中（$pm=80$，$user=10$，$time=300$）统计了全部 10 个镜像文件生成的次数，实验结果如图 5 所示。可见 MIN-COST 策略对镜像文件的重用起到了很好效果，相对于 SABFD、ITEM、DF 策略的下载次数分别最多可以降低 76.7%、73.7%、80.4%，其它策略在这一方面表现的并不突出，特别是在 DF 策略下，大部分时间微服务模块所需要的镜像都要重新下载。

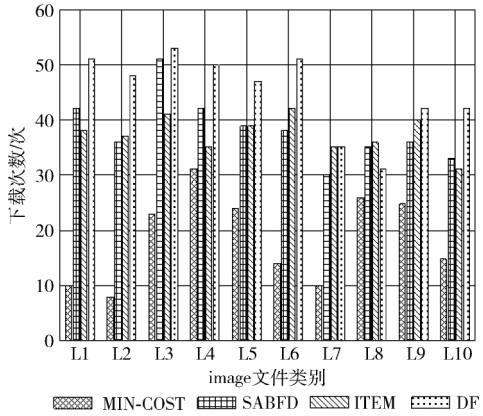

图 5 镜像文件下载次数

4.2.5 超参数与收敛速度

图 6 实验测试了探索与利用的权衡参数 e 的改变对算法收敛的影响，由图可以看出，当探索与利用处于平衡状态 $e=0.5$ 时处于较高的代价，且很长时间都难以收敛。而在利用大于探索的设置里则能较好地实现收敛。

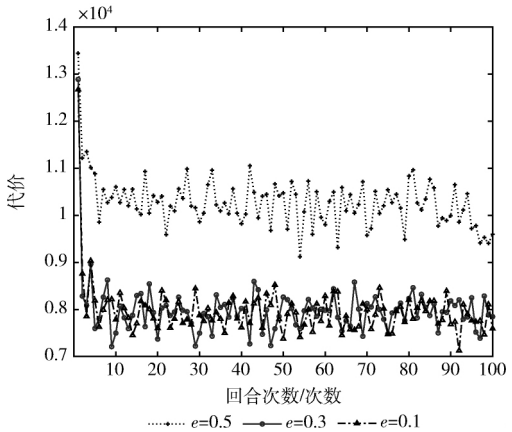

图 6 不同权衡参数 e 的影响

图 7 测试了不同学习率的影响，学习率在 0.1 时收敛速度略低于剩下两种，由于本实验规模并不大，所以此项参数的改变效果并不显著。

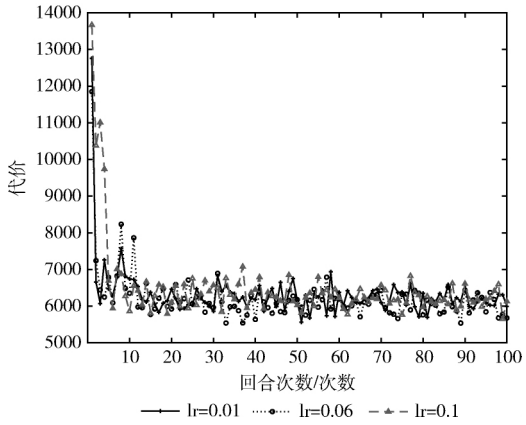

图 7　不同学习率 α 的影响

5　结束语

本文对移动边缘计算场景下的智能交通应用优化部署进行了详细的说明，采用了轻量的应用架构并制定了最优策略，且通过有效的方法将其实现。使用最小化代价的模型表述了该优化问题，并详尽地分析了该模型，使用了深度强化学习的方法求解了该问题，与 SABFD、ITEM、DF 这 3 种部署策略在不同参数环境下对比实验后，验证了该模型的有效性。在未来的研究中，将以此优化策略为基础，将对不确定的用户路径进行研究而不仅限于路径固定的公交场景，同时还将考虑在不同时刻下交通流量对部署情况的影响。

参考文献：

[1] Chen Xi，Huang Chuanhe，Fan Xiying，et al. LDMAC：A propagation delay-aware MAC scheme for long-distance UAV networks [J]. Computer Networks，2018，144：40-52.

[2] Liu Y，Sun X，Wei W，et al. Enhancing energy-efficient and QoS dynamic virtual machine consolidation method in cloud environment [J]. IEEE Access，2018，6：31224-31235.

[3] Jia M，Cao J，Liang W. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks [J]. IEEE Transactions on Cloud Computing，2017，5 (4)：725-737.

[4] Jia Mike，Liang Weifa，Xu Zichuan，et al. Cloudlet load balancing in wireless metropolitan area networks [C] //Proc of the 35th Annual IEEE International Conference on Computer Communications，2016：1-9.

[5] Fazio M，Celesti A，Ranjan R，et al. Open issues in scheduling microservices in the cloud [J]. IEEE Cloud Computing，2016，3 (5)：81-88.

[6] Yin L，Luo J，Luo H. Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing [J]. IEEE Transactions on Industrial Informatics，2018，14 (10)：4712-4721.

[7] Ksentini A，Taleb T，Chen M. A Markov decision process-based service migration procedure for follow me cloud [C] //Proc of IEEE International Conference on Communications，2014：1350-1354.

[8] Liu J，Mao Y，Zhang J，et al. Delay-optimal computation task scheduling for mobile-edge computing systems [C] //Proc of IEEE International Symposium on Information Theory，2016：1451-1455.

[9] Zhou R，Li Z，Wu C. Scheduling frameworks for cloud container services [J]. IEEE/ACM Transactions on Networking，2018，26 (1)：436-450.

[10] Mathieu Bouet，Vania Conan. Geo-partitioning of MEC resources [C] //Proc of the Workshop on Mobile Edge Communications，2017：43-48.

[11] Noghani K A，Kassler A，Gopannan P S. EVPN/SDN assisted live VM migration between geo-distributed data centers [C] //Proc of the 4th IEEE Conference on Network Softwarization and Workshops，2018：105-113.

[12] Wang H，Tianfield H. Energy-aware dynamic virtual machine consolidation for cloud datacenters [J]. IEEE Access，2018，6：15259-15273.

[13] Wang L，Jiao L，He T，et al. Service entity placement for social virtual reality applications in edge computing [C] //Proc of IEEE Conference on Computer Communications，2018：468-476.

[14] Cheng Zhang，Bo Gu，Zhi Liu，et al. A reinforcement learning approach for cost-and energy-aware mobile data offloading [C] //Proc of the 18TH Asia-Pacific Network Operations and Management Symposium，2016：1-6.

[15] Guan X，Wan X，Choi B Y，et al. Application oriented dynamic resource allocation for data centers using docker containers [J]. IEEE Communications Letters，2017，21 (3)：504-507.

[16] Gao Z，Jiao Q，Xiao K，et al. Deep reinforcement learning based service migration strategy for edge computing [C] //Proc of IEEE International Conference on Service-Oriented System Engineering，2019：116-1165.

[17] Cao G，Lu Z，Wen X，et al. AIF：An artificial intelligence framework for smart wireless network management [J]. IEEE Communications Letters，2018，22 (2)：400-403.

[18] Wang J，Zhao L，Liu J，et al. Smart resource allocation for mobile edge computing：A deep reinforcement learning approach [J/OL]. IEEE Transactions on Emerging Topics in Computing，2019. [2019-11-21]. http://doi.org/10.1109/TETC.2019.2902661.