计算机科学
COMPUTER SCIENCE

云边协同计算中基于强化学习的依赖型任务调度方法

胡晟熙, 宋日荣, 陈星, 陈哲毅

引用本文

胡晟熙, 宋日荣, 陈星, 陈哲毅. 云边协同计算中基于强化学习的依赖型任务调度方法[J]. 计算机科学, 2023, 50(11A): 220900076-8.

HU Shengxi, SONG Rirong, CHEN Xing, CHEN Zheyi. Dependency-aware Task Scheduling in Cloud-Edge Collaborative Computing Based on Reinforcement Learning [J]. Computer Science, 2023, 50(11A): 220900076-8.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220900076

云边协同计算中基于强化学习的依赖型任务调度方法

胡晟熙　宋日荣　陈　星　陈哲毅
福州大学计算机与大数据学院　福州 350108
福建省网络计算与智能信息处理重点实验室　福州 350108
(sxhu0913@qq.com)

摘　要　云边协同计算中,计算资源分散在移动设备、边缘服务器和云服务器。将应用程序中的计算密集型任务从本地卸载到远程设备执行,利用远程资源来扩展本地资源,是解决移动设备资源受限问题的一个有效途径。针对云边协同计算中存在依赖关系的任务调度问题,提出一种基于强化学习的无模型方法。首先,将移动应用程序建模为有向无环图,建立云边协同计算中的任务调度问题模型。其次,将任务调度过程建模为马尔可夫决策过程,即使用Q学习通过与网络环境交互学习合理的调度策略。实验结果表明,所提出的基于Q学习的依赖型任务调度方法在不同场景下均优于所对比的基准算法,有效地减少了应用程序的执行时间。

关键词:云边协同计算;任务调度;依赖型任务;强化学习
中图法分类号　TP338

Dependency-aware Task Scheduling in Cloud-Edge Collaborative Computing Based on Reinforcement Learning

HU Shengxi,SONG Rirong,CHEN Xing and CHEN Zheyi
College of Computer and Data Science,Fuzhou University,Fuzhou 350108,China
Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing,Fuzhou 350108,China

Abstract　In cloud-edge collaborative computing,computing resources are scattered among mobile devices,edge servers and cloud servers,Offloading the computation-intensive tasks from mobile devices to remote servers for execution and thus expand local computing capability by utilizing powerful remote resources,which is an effective way to solve the resource-constrained problem of mobile devices. Aiming at the scheduling decision problem of tasks with dependencies in cloud-edge collaborative computing, this paper proposes a model-free approach based on reinforcement learning. First,this paper models the mobile application as a directed acyclic graph,and builds a task scheduling problem model in cloud-edge collaborative computing. Second,it models the task scheduling process as a Markov decision process,using Q-learning to learn reasonable scheduling decisions by interacting with the network environment. Experimental results show that,the dependency-aware task scheduling based on Q-learning method proposed in this paper outperforms the compared benchmark algorithms in different scenarios,and effectively reduces the execution time of the application.

Keywords　Cloud-edge collaborative computing,Task scheduling,Dependency-aware task,Reinforcement learning

1　引言

随着智能技术的兴起,越来越多的计算密集型应用程序(如自动驾驶[1]、人脸识别[2]、增强现实[3]等)被开发以满足人们的需求。与此同时,这些应用程序的操作平台已从笔记本电脑和智能手机扩展到广泛的移动设备[4],如可穿戴设备、车辆和无人机。尽管这些移动设备具备的功能越来越强大,但是,由于尺寸和重量受到约束,它们在处理能力、内存容量和电池容量等方面仍受到限制[5],大多数移动设备依然无法在短时间内处理不断涌现的各类计算密集型任务。

计算卸载是解决移动设备资源受限问题的一个有效途径,它将应用程序中的计算密集型任务从本地发送到远程设备执行,利用远程资源来扩展本地资源[6]。移动云计算(Mobile Cloud Computing,MCC)的提出,首次将云端计算能力集成到移动网络,使移动用户能够通过移动运营商的核心网访问并使用云计算服务[7];在MCC中,计算卸载把计算密集型任务发送到云服务器,从而使用云服务器的计算和存储资源,然而,云服务器与移动设备之间的距离过远,这导致了高的网络时延。为了解决MCC中时延过高的问题,移动边缘计算(Mobile Edge Computing,MEC)考虑将云计算服务部署在移动设备的附近,即移动网络的边缘[8]。

考虑到云服务器的强大计算能力和边缘服务器更靠近

基金项目:国家自然科学基金项目(62072108);福建省自然科学基金杰青项目(2020J06014)
This work was supported by the National Natural Science Foundation of China(62072108) and Natural Science Foundation of Fujian Province for Distinguished Young Scholars(2020J06014).
通信作者:陈哲毅(z.chen@fzu.edu.cn)

用户端的优势，可以将云计算与边缘计算各自的优势有机结合，形成云边协同计算模式，为各种网络请求提供不同的按需服务[9]。在云边协同计算中，计算资源分散在移动设备、边缘服务器和云服务器，每个计算平台所拥有的资源在体量上又差异明显；对于一个应用程序来说，能够获取到的计算资源常常分散在多个不同的计算平台。应用程序的任务调度方案决定着应用程序的哪些计算任务应该运行在哪个计算平台上，为应用程序提供一个合理的任务调度方案至关重要。

已有的研究工作中[10-19]，启发式算法被广泛应用于解决任务调度决策问题，如蜂群算法、遗传算法、粒子群算法[11-14]等，这些方法能在一定程度上解决依赖型任务调度问题，但这些方法建立在高层抽象模型的基础上，很难用于实际的调度方案评估。本文使用无模型(model-free)方法，通过与网络环境交互，自适应地学习优化调度决策。这种 model-free 方法是基于 Q 学习(Q-learning)实现的。Q-learning 是一种强化学习(Reinforcement Learning，RL)方法[15-19]，它可以在事先不知道任何状态转移概率的情况下解决马尔可夫决策过程。

本文提出一种云边协同计算中基于 RL 的智能调度方法，通过与网络环境交互，自适应地学习优化调度决策。主要贡献如下：

(1)首先，将应用程序建模为有向无环图(Directed Acyclic Graph，DAG)，其中子任务由顶点表示，子任务之间的依赖关系由边表示。进一步对云边协同计算中的依赖型任务调度问题进行形式化定义。

(2)其次，对 DAG 中任务的优先级进行计算排序，基于该任务优先级调度队列，每个子任务的执行位置被依次决定。同时，使用马尔可夫决策过程对任务调度过程进行建模，并对状态空间、动作空间、状态转换函数和奖励函数进行定义。

(3)最后，设计了充分且合理的仿真环境，进行了大量的数值仿真实验来验证所提出方法的有效性。实验结果表明，所提出的基于 Q-learning 的依赖型任务调度(Dependency-aware task Scheduling based on Q-learning，DSQL)方法在不同环境下的表现均优于所对比的基准算法。

本文第 2 节回顾了相关工作；第 3 节对云边协同计算中依赖型任务调度问题进行形式化定义；第 4 节详细介绍了 DSQL 方法；第 5 节进行了数值仿真实验验证与分析；最后总结全文并展望未来。

2 相关工作

由于移动设备的存储空间、电池寿命和计算能力有限，为了提高移动应用程序的性能，移动应用程序的计算密集型任务可以从移动设备卸载到云或边缘服务器上执行。本节回顾和分析了云边缘环境中卸载问题的相关工作。

通常，存在两种计算任务卸载模型[20]：一种称为二进制卸载，另一种称为部分卸载。对于二进制卸载，应用程序中的任务作为一个整体在移动设备上本地执行或卸载到 MEC 服务器[21]。至于部分卸载，应用程序中的任务可以任意分为两部分，一部分在移动设备上执行，另一部分卸载至远程服务器执行[22]。然而，在实际中，一个移动应用程序通常有多个任务，它们之间的依赖性不能忽略，因为一些任务只有收到其他任务的处理结果才能执行。在这方面，DAG[23]被用来模拟移动应用程序中不同任务之间复杂的相互依赖关系。

由于 DAG 中子任务的相互耦合约束，任务调度策略设计变得非常具有挑战性。针对该问题，Topcuouglu 等[10]提出了优化 DAG 执行时间的异构最早完成时间(Heterogeneous Earliest Finish Time，HEFT)算法。HEFT 算法在调度决策过程中，根据子任务的最早完成时间获得当前的最优解，但是当子任务的非线性约束影响复杂性较高时，将导致 HEFT 算法的性能随之下降，难以在全局层次上获得一个优质解。Liang 等[11]应用了人工蜂群算法，通过随机生成初始种群，采用了两种常见的局部搜索方法，即交换法和插入法，并且提出了一种改进的插入机制实现应用程序执行时间的最小化。Cheng 等[12]提出了一种由可穿戴设备、移动设备和远程云组成的新型三层架构，设计了一种基于遗传算法(Genetic Algorithm，GA)的计算卸载智能调度方法，在给定时延约束下增加了可穿戴设备的任务吞吐量。Xie 等[13]针对混合云边缘环境中工作流应用的总完成时间和云和边缘服务器的经济成本，提出了一种定向非局部收敛的粒子群优化(Particle Swarm Optimization，PSO)算法，通过更新速度和位置使整个粒子群的搜索更有方向性，并能够达到更好的近似最优解。Lin 等[14]提出一种基于遗传算法操作的自适应离散粒子群优化算法，该算法继承了 PSO 算法易实现且收敛迅速等优势，同时结合了 GA 来弥补 PSO 算法容易陷入局部最优解的缺陷，获得了更加优质的调度方案。然而，上述工作建立在高层抽象模型的基础上，并假设每个计算平台上的任务执行成本是已知的。因此，很难应用这些启发式算法来处理实际中云边缘环境中的调度决策问题。

为了应对上述挑战，RL 应运而生。在没有任何先验知识的情况下，基于 RL 的方法能够通过与网络环境的充分交互来学习最优的调度决策，具有强大的灵活性。例如，Min 等[15]提出了一个基于 RL 的物联网设备计算卸载框架，用于在不了解 MEC 模型和本地计算和能耗模型的情况下实现最优卸载策略。Chen 等[16]研究了超密集切片 RAN 中的计算卸载问题，提出了一种基于深度 Q 网络的策略计算卸载算法，在不知道网络动力学先验知识的情况下学习最优策略。Huang 等[17]研究了采用二进制卸载策略的无线供电 MEC 网络，并提出了一种基于 RL 的在线卸载框架，该算法从过去的卸载经验中学习，通过 RL 改进生成的卸载行为，实现了与现有基准测试方法类似的接近最优的性能。然而，上述工作均采用二进制卸载，将应用程序中的任务视为一个整体，没有很好地考虑任务之间的依赖关系。考虑到任务的依赖性，Lin 等[18]针对车联网场景下的工作流调度问题，以特定截止日期为约束目标提出了一种基于 RL 的调度决策算法；同时，为了在学习过程中平衡 Q 学习的探索性和开发性，应用了模拟退火算法加快了 Q 学习的收敛时间。Tong 等[19]针对云环境下的工作流应用调度问题，提出了一种基于 RL 的优先级排序算法，使用 Q 学习算法在任务排序阶段对子任务的优先级进行决策。与 HEFT 算法所得的子任务优先级相比，该算法能够得到一个更好表现的子任务优先级序列。然而，上述工作只考虑了云或边缘环境中的调度决策问题，没有很好地研究分层云边缘架构来进一步提高任务调度性能。

与上述工作不同，本文将移动应用程序建模为 DAG，

DAG 由许多具有特定依赖性的任务组成。与现有的基于模型的方法相比，在没有任何先验知识的情况下，基于 RL 的 Q 学习方法在不同环境下的表现均优于所对比的基准算法。此外，在仿真环境中，考虑了云边缘的分层架构，为各种网络请求提供不同的按需服务。

3 系统模型与问题定义

本节将阐述云边协同环境中的任务调度决策问题。通常，计算密集型移动应用程序由具有依赖性的不同任务组成。例如，一个真实世界的人脸识别应用程序由依赖型任务组成，如拼接、检测或特征合并[24]。可以在终端设备执行任务，或将任务卸载到边缘服务器或云服务器执行。然而，不合理的任务调度方案可能会导致应用的响应时间过长。一方面，由于终端设备的计算能力较弱，在终端设备执行计算密集型任务可能会带来过高的任务执行延迟。另一方面，如果两个任务之间的数据流量很大，将这两个任务卸载到不同的计算节点可能会造成严重的传输延迟。因此，需要为应用程序提供一个合理的任务调度方案以缩短应用程序的响应时间。

3.1 网络模型

本文考虑如图 1 所示的架构，它由一个移动设备（MD）、一个边缘服务器（ES）和一个云服务器（CS）组成。用 $V=\{MD,ES,CS\}$ 表示计算节点的集合，每个计算节点的计算能力用 $f_k(k\in V)$ 表示。通常，由于移动设备的尺寸和重量受到约束，终端设备的计算能力最弱，边缘服务器的计算能力比终端设备强，但比云服务器弱。不同计算节点之间的数据传输速率用 $v_{k,l}(k,l\in V)$ 表示，由于边缘服务器被部署在靠近终端设备的位置，而云服务器和终端设备的位置较远，因此终端设备与边缘服务器之间的数据传输速率快于云服务器与终端设备、边缘服务器之间的数据传输速率。

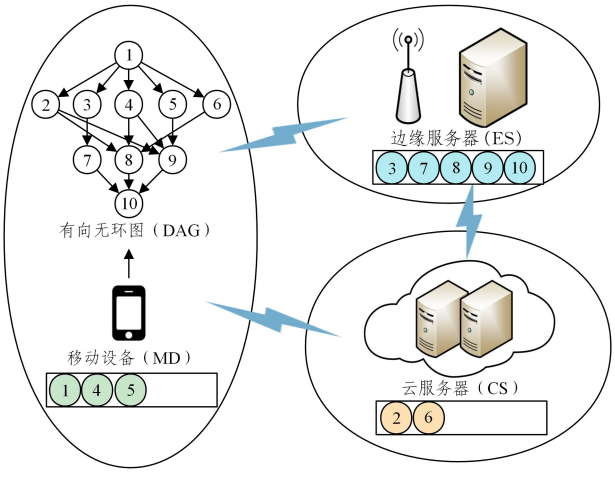

图 1 云边协同计算中的任务调度

Fig. 1 Task scheduling in cloud-edge collaborative computing

3.2 任务模型

如图 2 所示，一个应用程序可由一个有向无环图 $G=(N,E)$ 表示，其中 $N=\{1,2,\cdots,n\}$ 表示子任务集合，n 为子任务个数，每个子任务的计算量用 $c_i(i\in N)$ 表示；$E=\{e_{i,j}\mid i,j\in N,i<j\}$ 表示子任务间的依赖有向边集，对于一条 $e_{i,j}\in E$ 的有向边，子任务 i 是子任务 j 的直接前驱任务，子任务 j 是子任务 i 的直接后继任务。此外，每条 $e_{i,j}\in E$ 的有向边与权重 $d_{i,j}$ 相关联，$d_{i,j}$ 表示从子任务 i 传输到子任务 j 的数据量。本文用 $pre(i)$ 和 $suc(i)$ 来表示子任务 i 的直接前驱任务集合和直接后继任务集合，一个子任务只有接收到它所有前驱任务的处理结果后才能开始执行。例如，子任务的直接前驱任务集合为 $pre(9)=\{2,4,5\}$，因此子任务 9 在开始执行之前，必须接收到子任务 2,4,5 的处理结果。

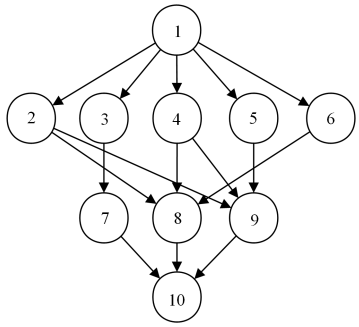

图 2 包含 10 个任务的 DAG 模型

Fig. 2 Example of DAG with 10 tasks

3.3 优化目标

子任务可以在本地处理，或者卸载至远程服务器（边缘服务器、云服务器）执行。因此，本文定义了一个二进制变量 x_{ik} 来表示任务的调度方案，$x_{ik}=1$ 表示将子任务 i 分配给计算节点 k，反之 $x_{ik}=0$。由于每个子任务只能分配给网络中的一个计算节点，因此有以下定义：

$$\sum_{k\in V}x_{ik}=1,\forall i\in N \tag{1}$$

此外，任一子任务 $j\in N$ 需满足两个条件才可以开始执行。首先，分配的计算节点可用，即当前没有其他子任务在该计算节点上执行。子任务 j 所分配的计算节点可用时间 T_j^a 应满足以下约束：

$$T_j^a\geqslant\sum_{k\in V}x_{jk}x_{ik}t_i^e,\forall i\in pre(j),\forall j\in N \tag{2}$$

其中，t_i^e 为子任务 i 的完成时间。

接着，子任务 j 应该准备就绪，即它已经接收到所有直接前驱子任务的处理结果。子任务 j 就绪时间 T_j^r 定义为：

$$T_j^r=\max_{i\in pre(j)}\left\{t_i^e+\sum_{(k,l)\in V}\frac{e_{i,j}x_{ik}x_{jl}}{v_{k,l}}\right\},\forall j\in N \tag{3}$$

如果子任务 j 和它的一个直接前驱子任务 $i\in pre(j)$ 分别被分配给不同的计算节点 k 和 l，则需要考虑通信延迟 $e_{i,j}/v_{k,l}$。否则，它们之间的数据传递可以通过共享内存实现，而没有通信延迟。在这种情况下，约束右边的第二项将为零。

综合考虑上述两个条件，子任务 j 的开始时间定义为：

$$ST(j)=\max\{T_j^a,T_j^r\},\forall j\in N \tag{4}$$

进一步，子任务 j 的结束时间定义为：

$$FT(j)=ST(j)+\sum_{k\in V}\frac{x_{jk}c_i}{f_k},\forall j\in N \tag{5}$$

值得注意的是，DAG 中的任务是并行执行的，因此，应用程序的延迟将等于完成任务依赖链中的任务所花费的最长时间。用 $D_{1:t}$ 表示在第 t 个时间步成功完成的所有子任务集合。进一步，应用程序在第 t 个时间步的累积执行延迟 $T_{1:t}$ 可以定义为：

$$T_{1:t}=\max\{FT(j)\},\forall j\in D_{1:t} \tag{6}$$

如图 3 所示，假定在第 4 个时间步完成的所有子任务集合 $D_{1:t}=\{1,3,4,6\}$。因此，应用程序在第 4 个时间步的累计

执行延迟 $T_{1,t}=\max\{FT(1),FT(3),FT(4),FT(6)\}$。

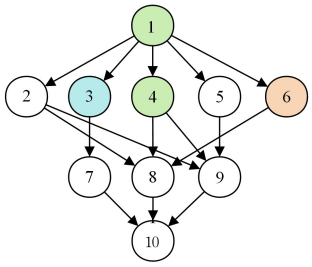

图 3 第 4 个时间步时任务的执行情况

Fig. 3 Execution of task at the 4th time step

一个应用程序被认为是完成的，当且仅当其所有 n 个子任务都成功完成。当一个应用程序的所有 n 个任务都成功完成时，$D_{1,n}=\{1,2,\cdots,n\}$。因此，应用程序的总执行延迟 $T_{1,n}$ 可以通过以下公式计算：

$$T_{1,n}=\max\{FT(j)\},\forall j\in D_{1,n} \tag{7}$$

不同的任务调度方案可能会导致 $T_{1,n}$ 的取值不同。为了在云边协同环境中达到一个较好的计算卸载效果，本文的优化目标是尽可能地最小化 $T_{1,n}$。因此，目标函数定义为：

$$\text{Minimize } T_{1,n} \tag{8}$$

然而，由于移动边缘环境的高动态性和任务依赖图的非线性约束，在运行时环境中准确地评估不同卸载方案下应用程序的响应时间是十分困难的。传统基于启发式的方法建立在高层抽象模型的基础上，很难用于实际的卸载方案评估。为了解决这一问题，本文使用一种基于 RL 的方法求解卸载方案。基于 RL 的方法将复杂的系统视为一个黑匣子，并与之交互以学习最优策略，而无需对系统动力学建模。

4 方法描述

4.1 调度优先级

按优先级排序子任务至关重要。与大多数依赖型任务调度工作相同[25-27]，本文首先对每个子任务的优先级进行设置，并相应地根据优先级降序进行调度决策。

如前所述，在本文中应用程序由有向无环图 $G=(N,E)$ 表示，每条 $e_{i,j}\in E$ 的有向边与权重 $d_{i,j}$ 相关联，$d_{i,j}$ 表示从子任务 i 传输到子任务 j 的数据量。因此，有向边 $e_{i,j}\in E$ 的平均通信成本定义为：

$$cost_1=\sum_{(k,l)\in V}\frac{e_{i,j}}{v_{k,l}}/3 \tag{9}$$

此外，子任务 $i\in N$ 的平均计算成本定义为：

$$cost_2=\sum_{k\in V}\frac{c_i}{f_k}/3 \tag{10}$$

基于子任务的平均通信成本和平均计算成本[27]，子任务 i 的调度优先级 $rank(i)$ 被递归地定义为：

$$rank(i)=\max_{j\in suc(i)}(cost_1+rank(j))+cost_2 \tag{11}$$

其中，$suc(i)$ 表示子任务 i 的直接后继子任务集合。对于一个没有后继子任务的结束任务，其优先级为：

$$rank(i)=cost_2 \tag{12}$$

通过式(11)和式(12)对 DAG 中每个子任务的调度优先级进行计算并排序，根据子任务的优先级排序之后，得到其任务调度队列，表示为：

$$Que=(q_1,q_2,\cdots,q_n) \tag{13}$$

基于该任务调度队列，每个子任务的执行位置被依次决定。值得注意的是，为了应用程序在本地移动设备处启动，本文规定第一个子任务必须在本地移动设备执行[9]。

4.2 马尔可夫决策过程

在任务调度决策过程中，移动设备代理根据系统状态选择子任务的调度操作，执行调度操作后移动设备代理根据环境的反馈收到一个对应的奖励值和一个新的系统状态。因此，可以使用马尔可夫决策过程(Markov Decision Process，MDP)对任务调度过程进行建模。更具体地说，MDP 可以定义为 4 元组$\langle S,A,T,R\rangle$，其中 S 是状态空间，A 是动作空间，T 是状态转换函数，R 是奖励函数[28]。Q-learning 作为一种无模型且广泛使用的 RL 算法，将环境视为“黑盒”，在没有任何先验知识的情况下，通过与环境进行交互学习出一个合理的任务调度方案。

基于问题定义(在第 3 节中描述)，在所提出的任务调度问题中，相应的状态空间 S、动作空间 A、状态转换函数 T 和奖励函数 R 定义如下。

状态空间：状态空间定义为 S，其中 $s_t\in S$ 代表了在第 t 个时间步的状态。为了更好地表示子任务的调度方案，s_t 被定义为 $s_t=(a_{1,t},a_{2,t},\cdots,a_{n,t})$，$s_t$ 描述了在第 t 个时间步时任务的调度情况。具体来说，其中 $a_{i,t}\in\{0,1,2,3\},\forall i\in N$ 表示了在第 t 个时间步应用程序中第 i 个子任务的执行情况，$a_{i,t}=1,2,3$ 分别表示在第 t 个时间步时第 i 个子任务在移动设备、边缘服务器、云服务器执行。特别地，$a_{i,t}=0$ 表示在第 t 个时间步时第 i 个子任务还未执行。如图 3 所示，假设所得的任务调度队列 $Que=(1,3,4,6,2,7,5,8,9,10)$，在第 4 个时间步时，对应的 $s_t=s_4=(1,0,2,1,0,3,0,0,0,0)$。

动作空间：为了将子任务卸载到合适的计算节点上，将动作空间定义为 $A=\{1,2,3\}$，其中一个潜在的动作 $a_t\in A$ 表示为待决策的任务(即在任务调度队列中的第 $t+1$ 个任务)确定执行位置。具体来说，$a_t\in\{1,2,3\}$ 分别表示在移动设备执行当前待决策任务、将当前待决策任务卸载到边缘服务器执行、将当前待决策任务卸载到云服务器执行。

状态转换函数：状态转移函数被定义为 $T(s_t,a_t)$，其返回是在第 t 个时间步的状态 s_t 下执行动作 a_t 之后的下一个状态。例如，表 1 描述了一个完整的任务调度过程，如在第 3 个时间步的状态 s_3 下执行了动作 $a_3=3$，为当前待决策的任务(即在任务调度队列中的第 4 个任务，在此处为子任务 6)卸载到云服务器执行。因此，$s_4=T(s_3,a_3)=(1,0,2,1,0,3,0,0,0,0)$。

奖励函数：为了引导 RL 代理最小化应用程序的响应时间，奖励函数定义为：

$$R(s_t,a_t)=T_{1,t}-T_{1,t+1} \tag{14}$$

其中，$R(s_t,a_t)$ 表示在第 t 个时间步的状态 s_t 下执行动作 a_t 后 RL 代理所得到奖励，其中 $T_{1,t}$ 对应了应用程序在当前时间步，即第 t 个时间步的累积执行延迟；$T_{1,t+1}$ 表示执行动作 a_t 后，即在第 $t+1$ 个时间步的应用累积执行延迟。RL 代理将学习最优的任务调度决策方案，以获得最大的累积奖励，这相当于应用程序的最小执行时间。如表 1 所列，在第 3 个时间步将待决策子任务(即子任务 6)卸载到云服务器执行的奖励 $r_3=R(s_3,a_3)=T_{1,t}-T_{1,t+1}=T_{1,3}-T_{1,4}=0.276-0.276=0$。

表 1　任务调度过程的一个例子

Table 1　Example of a task scheduling process

t	s_t	a_t	$T_{1,t}$/s
1	1,0,0,0,0,0,0,0,0,0	2	0.069
2	1,0,2,0,0,0,0,0,0,0	1	0.276
3	1,0,2,1,0,0,0,0,0,0	3	0.276
4	1,0,2,1,0,3,0,0,0,0	3	0.276
5	1,3,2,1,0,3,0,0,0,0	2	0.276
6	1,3,2,1,0,3,2,0,0,0	1	0.390
7	1,3,2,1,1,3,2,0,0,0	2	0.425
8	1,3,2,1,1,3,2,2,0,0	2	0.461
9	1,3,2,1,1,3,2,2,2,2	2	0.505
10	1,3,2,1,1,3,2,2,2,2	—	0.531

4.3　基于 Q-learning 的依赖型任务调度算法

Q-learning 是由 Watkins 提出的一种 model-free 的强化学习算法[29]。Q-learning 拥有一个用于储存 Q 值的 Q 表，Q 表常被认为是智能体的大脑，能够对智能体的决策行为进行指导。具体来说，Q 表的横向表示动作，纵向表示状态，$Q(s,a)$表示在状态 s 执行动作 a 的 Q 值。在学习过程中，RL 代理通过 ε-greedy 策略[29]，根据当前时间步 t 的状态 s_t 选择动作 a_t。接下来，RL 代理接收即时奖励 $r_t=R(s_t,a_t)$，并且将状态更新为 s_{t+1}。Q-learning 算法的流程图如图 4 所示。

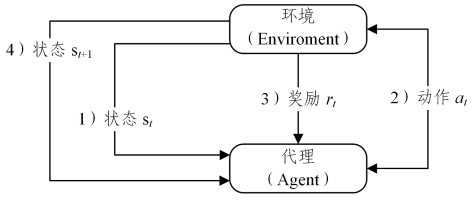

图 4　Q-learning 算法的流程

Fig. 4　Process of Q-learning algorithm

相应地，Q-learning 的更新方程定义如下：

$$Q(s_t,a_t)=Q(s_t,a_t)+\alpha(r_t+\beta\max_a(s_{t+1},a_{t+1})-Q(s_t,a_t)) \tag{15}$$

其中，$\alpha(0\leqslant\alpha\leqslant1)$表示学习效率，$\beta(0\leqslant\beta\leqslant1)$表示折扣因子，$\max_a(s_{t+1},a_{t+1})$是在下一个状态 s_{t+1} 中选取 Q 值最大的状态-动作对。

基于 Q-learning 的依赖型任务调度算法流程具体如下：

Step 1　初始化 Q 表及相关参数，根据 4.1 节的相关公式计算所有子任务的调度优先级，得到任务调度队列。

Step 2　在每个训练轮次开始时，在本地移动设备启动应用程序，初始化当前状态为(1,0,0,0,0,0,0,0,0,0)。

Step 3　在训练过程中，基于任务调度队列，每个子任务的执行位置被依次决定。ε-greedy 策略兼具了探索与利用，它以 ε 的概率从所有的调度方案中随机选择一个，以 1−ε 的概率选择 Q 值最大的调度方案。RL 代理执行动作 a_t 并根据式(14)得到对应奖励，根据式(15)更新 Q 表中的 Q 值。同时，由状态转换函数更新当前状态 $s_{t+1}=T(s_t,a_t)$。

Step 4　算法收敛后输出 $T_{1,n}$ 和对应的任务调度方案。

算法 1　DSQL

输入：G=(N,E)

输出：$T_{1,n}$ 和对应的任务调度方案

1. 初始化 Q 表中的 Q 值为 0
2. 初始化参数
3. 计算所有子任务的调度优先级，得到任务调度队列 Que(q_1,…,q_n)
4. for each episode e do
5.　初始化当前状态为 s_t→(1,0,0,0,0,0,0,0,0,0)
6.　for each step t do
7.　　使用 ε-greedy 策略根据当前状态 s_t 从 Q 表中选择动作：a_t→select_action(s_t,Q_Table)
8.　　RL 代理执行动作 a_t，并得到对应的奖励值 $r_t=R(s_t,a_t)$
9.　　更新 Q 表中的 Q 值
10.　　更新当前状态 s_t→s_{t+1}
11.　end for
12. end for

5　仿真实验与分析

5.1　仿真设置

首先对三层体系结构中的计算节点的计算能力和计算节点之间的传输速率进行设置。考虑到不同计算节点的差异性，设置了 5 种不同的场景，如表 2 所列。

表 2　5 种不同的场景

Table 2　Five different scenarios

No.	F/GHz			R/(MB/s)		
	f_{MD}	f_{ES}	f_{RC}	$v_{MD,ES}$	$v_{MD,RC}$	$v_{ES,RC}$
1	2.4	4.5	6.4	8.0	2.5	3.1
2	2.2	4.3	6.2	6.5	2.3	3.4
3	2.0	4.0	6.0	6.0	2.0	3.0
4	1.8	3.8	5.4	5.5	1.8	2.6
5	1.6	3.6	5.6	5.0	1.6	2.8

其次，为了模拟应用程序(即 DAG)的多样性，参考文献[30-32]的任务图结构，分别构造了任务规模 $n=\{10,15,20,25\}$的 4 种 DAGs，所构造的任务图结构如图 5 所示。对于每个应用程序 $G=(N,E)$，每个子任务的计算量 $c_i(i\in N)$服从[50,500] megacycles 内的均匀分布。而对于每条 $e_{i,j}\in E$ 的有向边，子任务 i 传输到子任务 j 的数据传输量服从[0,1 000]kB 内的均匀分布。

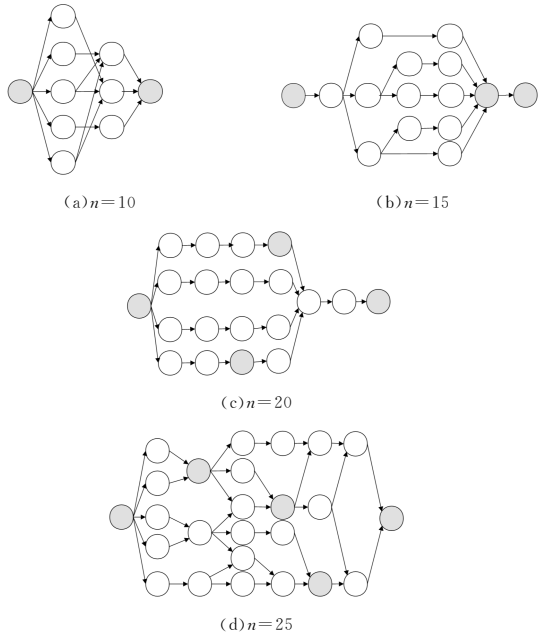

图 5　不同结构的 DAGs

Fig. 5　Different structures of DAGs

最后，对 Q-learning 中的学习率 α、折扣因子 β 和贪婪系数 ε 进行设置。其中，学习效率 α 表示每次训练的经验被学习的程度，本文将其设置为 0.7；折扣因子 β 表示对未来奖励

的重视程度，为了权衡 RL 代理收到的当前奖励反馈和未来的奖励反馈，本文将其设置为 0.95；贪婪系数 ε 是指对状态空间的探索程度，一开始将贪婪系数设置为 0，这样可以有更大的概率依靠随机来决定动作，而随着时间不断推进，贪婪系数将逐渐变大至 1，并保持到算法结束，这样可以有更大的概率依靠 Q 值大小来决定动作，在后期有利于算法的收敛。

5.2 实验结果与分析

在本小节中，为了验证所提出 RL 算法的有效性，将 RL 与以下 3 个基准算法进行对比。

(1)基于 PSO-GA[14] 的调度：PSO-GA 算法是在传统 PSO 算法的基础上，通过引入了 GA 的交叉和变异算子对原算法的粒子更新策略进行改进。该算法继承了 PSO 算法易实现且收敛迅速等优势，同时结合了 GA 具有良好全局搜索能力的特点，相对于传统的 PSO 算法，能够获得更加优质的解。基于文献[14]，PSO-GA 的参数设置如下：种群大小为 200，最大迭代次数为 1000，认知因子 c_1 的初始值和最终值分别为 0.9 和 0.2，认知因子 c_2 的初始值和最终值分别为 0.9 和 0.4，惯性权重因子 w 的最大值和最小值分别为 0.9 和 0.4。

(2)基于 HEFT 的调度(HEFT-based)：HEFT[10] 是一种经典的启发式调度算法，它根据子任务的最早完成时间进行调度决策，即在每一步选择当前最优的调度决策。

(3)基于规则的调度(Rule-based)：基于规则的调度首先选择最佳 DAG 切分点，并将剩余的子任务从终端设备卸载到云服务器执行[33]。

为了公平起见，以上基准算法都使用 4.1 节的相关公式计算所有子任务的调度优先级并排序。

如图 6 所示，本文所提出的 DTQL 在不同任务规模下的平均表现优于基于 PSO-GA 的调度方法 3.12%。

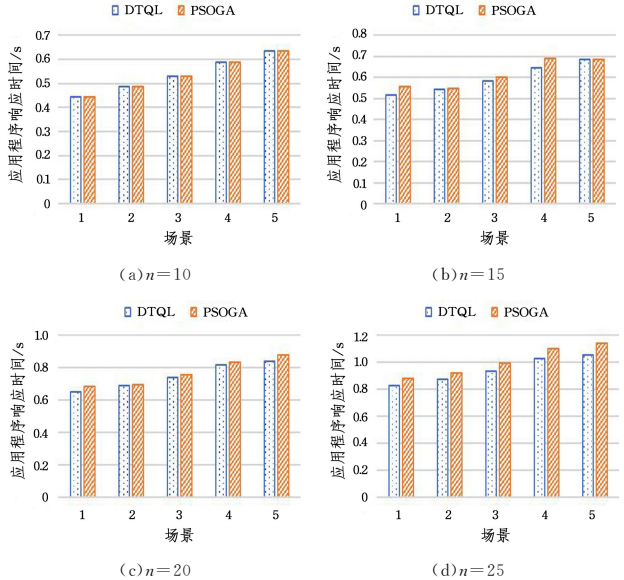

图 6 与 PSO-GA 方法的性能比较

Fig. 6 Performance comparison between the proposed method and PSO-GA method

值得注意的是，在任务规模较小(如 $n=10$)时，DTQL 与基于 PSO-GA 的调度方法取得了相同的结果，这是因为在任务规模较小时，解空间也较小，基于 PSO-GA 的调度方法能够通过自身的交叉变异操作得到最优解。而在庞大的解空间的场景下(如 $n=25$)，基于 PSO-GA 的调度方法易陷入局部最优情况。此外，基于 PSO-GA 的调度方法建立在高层抽象模型的基础上，难以用于实际的卸载方案评估。而 DTQL 通过与环境交互学习最优策略，无需对系统动力学建模。

如图 7 所示，本文所提出的 DTQL 在不同任务规模下的平均表现优于基于 HEFT 的调度方法 16.07%。

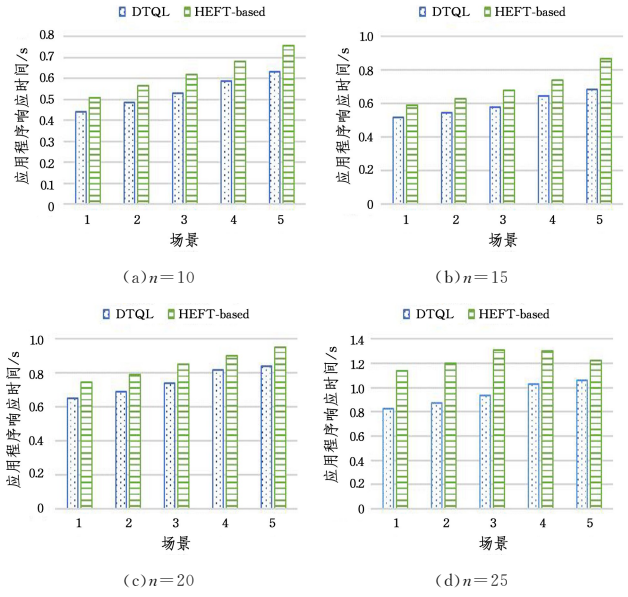

图 7 与 HEFT-based 方法的性能比较

Fig. 7 Performance comparison between the proposed method and HEFT-based method

基于 HEFT 的调度方法根据子任务的最早完成时间进行调度，难以获得一个全局最优的调度方案。特别地，当任务图较为复杂(如 $n=25$)时，受到子任务之间复杂的非线性约束的影响，基于 HEFT 的调度方法表现不佳。

如图 8 所示，本文所提出的 DTQL 在不同任务规模下的平均表现优于基于规则的调度方法 20.19%。

基于规则的调度方法涉及由专家设置的切分规则，但这

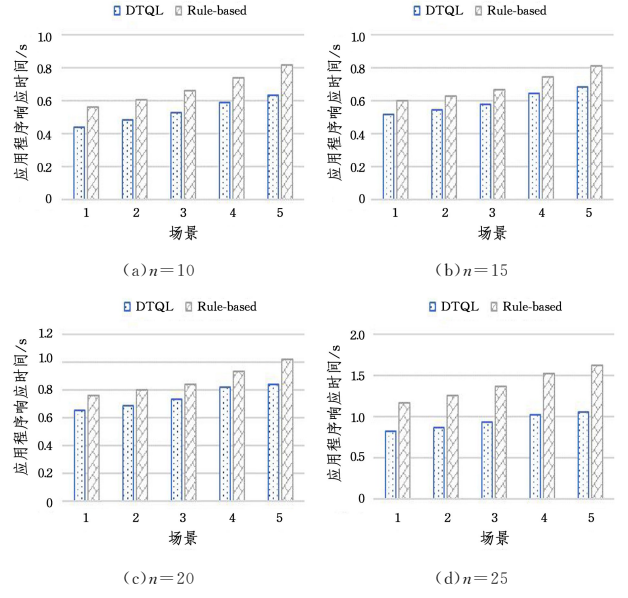

图 8 与 Rule-based 方法的性能比较

Fig. 8 Performance comparison between the proposed method and Rule-based method

与基于 HEFT 的调度方法类似，当任务图较为复杂（如 $n=25$）时，受到子任务之间复杂的非线性约束的影响，DTQL 的表现明显优于基于规则的调度方法。

结束语 本文针对云边协同计算中存在依赖关系的任务调度问题，提出一种基于强化学习的无模型方法。首先，本文将移动应用程序建模为有向无环图，建立云边协同计算中的任务调度问题模型。接下来，使用 Q 学习，通过与网络环境交互，自适应地学习优化调度决策。最后，通过与 PSO-GA，HEFT-based，Rule-based 这 3 种基准调度方法进行比较，验证了 DTQL 能够有效减少应用程序的执行时间。但与此同时，本文在任务选择阶段根据优先级降序进行调度决策，这可能会导致无法获得全局最优解；此外，本文主要关注的是应用程序的执行时间，未考虑能耗、成本等因素对依赖型任务调度的影响。因此，在未来的工作中将重点探索利用 RL 实现任务选择以及任务的多目标优化问题。

参 考 文 献

[1] WANG J，LIU J，KATO N. Networking and communications in autonomous driving：A survey [J]. IEEE Communications Surveys & Tutorials，2018，21(2)：1243-1274.

[2] HASSABALLAH M，ALY S. Face recognition：challenges，achievements and future directions [J]. IET Computer Vision，2015，9(4)：614-626.

[3] KIM K，BILLINGHURST M，BRUDER G，et al. Revisiting trends in augmented reality research：A review of the 2nd decade of ISMAR(2008-2017) [J]. IEEE Transactions on Visualization and Computer Graphics，2018，24(11)：2947-29629.

[4] HUANG J，ZHOU Y，NING Z，et al. Wireless power transfer and energy harvesting：Current status and future prospects [J]. IEEE Wireless Communications，2019，26(4)：163-169.

[5] CONG P，ZHOU J，LI L，et al. A survey of hierarchical energy optimization for mobile edge computing：A perspective from end devices to the cloud [J]. ACM Computing Surveys(CSUR)，2020，53(2)：1-44.

[6] SHAKARAMI A，GHOBAEI-ARANI M，MASDARI M，et al. A survey on the computation offloading approaches in mobile edge/cloud computing environment：a stochastic-based perspective [J]. Journal of Grid Computing，2020，18(4)：639-671.

[7] CUI Y，SONG J，MIAO C C，et al. Mobile cloud computing research progress and trends [J]. Chinese Journal of Computers，2017，40(2)：273-295.

[8] DONG S Q，LI H L，QU Y B，et al. Survey of research oncomputation unloading strategy in mobile edge computing[J]. Computer Science，2019，46(11)：32-40.

[9] CHEN X，ZHANG J，LIN B，et al. Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments [J]. IEEE Transactions on Parallel and Distributed Systems，2021，33(3)：683-697.

[10] TOPCUOGLU H，HARIRI S，WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems，2002，13(3)：260-274.

[11] LIANG Y C，CHEN A H L，NIEN Y H. Artificial bee colony for workflow scheduling [C]// 2014 IEEE Congress on Evolutionary Computation(CEC). IEEE，2014：558-564.

[12] CHENG Z，LI P，WANG J，et al. Just-in-time code offloading for wearable computing [J]. IEEE Transactions on Emerging Topics in Computing，2015，3(1)：74-83.

[13] XIE Y，ZHU Y，WANG Y，et al. A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment [J]. Future Generation Computer Systems，2019，97：361-378.

[14] LIN B，GUO W Z，CHEN G L. Scheduling strategy for science workflow with deadline constraint on multi-cloud [J]. Journal on Communications，2018，39(1)：56-69.

[15] MIN M，XIAO L，CHEN Y，et al. Learning-based computation offloading for IoT devices with energy harvesting [J]. IEEE Transactions on Vehicular Technology，2019，68(2)：1930-1941.

[16] CHEN X，ZHANG H，WU C，et al. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning [J]. IEEE Internet of Things Journal，2018，6(3)：4005-4018.

[17] HUANG L，BI S，ZHANG Y J A. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks [J]. IEEE Transactions on Mobile Computing，2019，19(11)：2581-2593.

[18] LIN KAI，LU YU，CHEN X，et al. Workflow Scheduling Strategy for Reasoning Task of Autonomous Driving [J]. Journal of Chinese Computer Systems，2021，42(3)：632-639.

[19] TONG Z，DENG X M，CHEN H J，et al. Multi-objective task scheduling algorithm based on reinforcement learning in cloud environment [J]. Journal of Chinese Computer Systems，2020，41(2)：285-290.

[20] MAO Y，YOU C，ZHANG J，et al. A survey on mobile edge computing：The communication perspective [J]. IEEE Communications Surveys & Tutorials，2017，19(4)：2322-2358.

[21] MIETTINEN A P，NURMINEN J K. Energy efficiency of mobile clients in cloud computing [C]// 2nd USENIX Workshop on Hot Topics in Cloud Computing(HotCloud 10). 2010.

[22] MAHMOODI S E，UMA R N，SUBBALAKSHMI K P. Optimal joint scheduling and cloud offloading for mobile applications [J]. IEEE Transactions on Cloud Computing，2016，7(2)：301-313.

[23] JIA M，CAO J，YANG L. Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing [C]// 2014 IEEE Conference on Computer Communications Workshops(INFOCOM WKSHPS). IEEE，2014：352-357.

[24] RA M R，SHETH A，MUMMERT L，et al. Odessa：enabling interactive perception applications on mobile devices [C]// Proceedings of the 9th International Conference on Mobile Systems，Applications，and Services. 2011：43-56.

[25] LIN X，WANG Y，XIE Q，et al. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment [J]. IEEE Transactions on Services Computing，2014，8(2)：175-186.

[26] LIU Y，WANG S，ZHAO Q，et al. Dependency-aware task scheduling in vehicular edge computing [J]. IEEE Internet of Things Journal，2020，7(6)：4961-4971.

[27] SHU C, ZHAO Z, HAN Y, et al. Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach [J]. IEEE Internet of Things Journal, 2019, 7(3): 1678-1689.

[28] ChEN X, HU J, CHEN Z, et al. AReinforcement Learning-Empowered Feedback Control System for Industrial Internet of Things [J]. IEEE Transactions on Industrial Informatics, 2021, 18(4): 2724-2733.

[29] WATKINS C J C H, DAYAN P. Q-learning [J]. Machine learning, 1992, 8(3): 279-292.

[30] ZHANG W, WEN Y. Energy-efficient task execution for application as a general topology in mobile cloud computing [J]. IEEE Transactions on cloud Computing, 2015, 6(3): 708-719.

[31] XIE G, CHEN Y, LIU Y, et al. Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems [J]. IEEE Transactions on Industrial Informatics, 2016, 13(4): 1629-1640.

[32] WU Q, ISHIKAWA F, ZHU Q, et al. Deadline-constrained cost optimization approaches for workflow scheduling in clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(12): 3401-3412.

[33] KANG Y, HAUSWALD J, GAO C, et al. Neurosurgeon: Collaborative intelligence sbetween the cloud and mobile edge [J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 615-629.

HU Shengxi, born in 1998, postgraduate, is a member of China Computer Federation. His main research interests include edge computing and task sche-duling.

CHEN Zheyi, born in 1991, Ph. D. His main research interests include cloud-edge computing, resource optimization, deep learning, and reinforcement learning.