移动边缘计算中基于深度强化学习的计算卸载调度方法*

詹文翰[1a,1b]，王 瑾[2]，朱清新[1a]，段翰聪[1b]，叶娅兰[1b]
(1. 电子科技大学 a. 信息与软件工程学院; b. 计算机科学与工程学院，成都 611731; 2. 埃克塞特大学 计算机系，英国 埃克塞特 EX44RN)

摘 要: 针对移动边缘计算中具有依赖关系的任务的卸载决策问题，提出一种基于深度强化学习的任务卸载调度方法，以最小化应用程序的执行时间。任务调度的过程被描述为一个马尔可夫决策过程，其调度策略由所提出的序列到序列深度神经网络表示，并通过近端策略优化(proximal policy optimization)方法进行训练。仿真实验表明，所提出的算法具有良好的收敛能力，并且在不同环境下的表现均优于所对比的六个基线算法，证明了该方法的有效性和可靠性。

关键词: 移动边缘计算; 计算卸载; 任务调度; 深度强化学习
中图分类号: TP391 **文献标志码:** A **文章编号:** 1001-3695(2021)01-048-0241-05
doi:10.19734/j.issn.1001-3695.2019.10.0594

Deep reinforcement learning based offloading scheduling in mobile edge computing

Zhan Wenhan[1a,1b], Wang Jin[2], Zhu Qingxin[1a], Duan Hancong[1b], Ye Yalan[1b]
(1. *a. School of Information & Software Engineering, b. School of Computer Science & Engineering, University of Electronic Science & Technology of China, Chengdu* 611731, *China*; 2. *College of Engineering, Mathematics & Physical Sciences, University of Exeter, Exeter EX44RN, UK*)

Abstract: Aiming at the problem of task offloading with dependency in mobile edge computing, this paper proposed a deep reinforcement learning based offloading scheduling method to minimize the execution time of mobile applications. This method described the process of task scheduling as a Markov decision process. It adopted a sequence to sequence deep neural network to represent the scheduling policy, and then trained the deep neural network with the proximal policy optimization method. Simulation results show that the proposed method has good convergence ability and outperforms six baseline algorithms in different environments, demonstrating the effectiveness and reliability of the proposed method.

Key words: mobile edge computing; computation offloading; task scheduling; deep reinforcement learning

0 引言

随着物联网及移动通信技术的不断发展，移动应用已变得日益多样和复杂，对计算、存储、网络等资源的需求也越来越高。尽管当前的用户设备(user equipment，UE)在硬件性能方面有着较大的提升，但在处理(如虚拟现实、增强现实、模式识别和移动医疗等)资源密集型应用时，仍会显著增加用户的等待时间，降低用户体验。移动边缘计算(mobile edge computing，MEC)的提出为此类问题提供了一个很好的解决方案[1]。与集中式的云计算相比，MEC 将计算和存储资源分布式地部署在靠近用户的网络边缘(如移动基站、无线热点或边缘路由器中)。移动应用中的计算任务可以被就近卸载(offload)执行，从而有效地降低了应用程序的通信开销和网络延迟，同时靠近用户的处理方式也极大地缓解了核心网络和数据中心的处理压力。在 MEC 中，移动应用性能的提升很大程度上依赖于高效的任务卸载决策。由于在卸载决策过程中需要考虑诸多因素(如应用程序时延、网络和带宽限制、任务间的依赖关系等)，对最优卸载方案的求解往往非常具有挑战性。因此，卸载决策问题在近些年获得了相关学者的广泛关注[1,2]。

通常，一个移动应用程序可以被建模为一个有向无环图(directed acyclic graph，DAG)，以实现细粒度的任务卸载调度，并使得任务的并行处理成为可能[3]。但是，除了几种极其简单的情况外，基于 DAG 的任务调度问题本身就是 NP 难的[4,5]。因此许多现有工作主要基于启发式/近似算法对该任务卸载调度问题进行求解。例如，文献[6]通过考虑 UE 和服务器之间负载均衡设计了一种启发式的 DAG 分解调度算法以达到最小化移动应用时延的目的；文献[7]提出了一种启发式算法，对移动应用中的任务调度问题分三个步骤进行求解，旨在规定程序执行时延以内最小化系统能耗；文献[8]考虑了 DAG 中的某些任务必须在 UE 本地执行的情况，以最小化本地执行的任务之间的等待时延。该问题被建模为一个非线性整数规划问题，并采用元启发式算法近似求解。通常，为了得到让人满意的近似解，在对启发式/近似算法进行设计时需要人类专家知识的辅助，这一般会导致所设计的算法缺乏灵活性；若使用更具灵活性的元启发式算法又往往会带来极高的计算成本[9]，无法满足 MEC 场景下的实时性要求。

深度强化学习(deep reinforcement learning，DRL)是强化学习与深度神经网络(deep neural network，DNN)的结合[10~12]。

收稿日期: 2019-10-13; 修回日期: 2019-11-29 基金项目: 国家自然科学基金面上项目(61871096，61976047)；四川省科技厅重点研发项目(2019YFG0122)

作者简介: 詹文翰(1987-)，男，四川宜宾人，实验师，博士，主要研究方向为边缘计算、人工智能(zhanwenhan@uestc.edu.cn)；王瑾(1990-)，男，湖北武汉人，博士研究生，主要研究方向为人工智能、分布式系统；朱清新(1954-)，男，四川人，教授，博导，博士，主要研究方向为生物信息学、信息检索、计算运筹学与最优化；段翰聪(1974-)，男，四川成都人，研究员，博导，博士，主要研究方向为分布式系统、人工智能；叶娅兰(1974-)，女，四川宜宾人，副教授，博士，主要研究方向为智能信息处理.

通过不断与环境进行交互，其能够自动学习不同状态下应该采取的最优动作(即策略)，以最大化所获奖励；同时，DNN 强大的表示能力可以充分拟合最优策略，能很好地适应复杂环境。基于以上特点，DRL 被广泛认为是一种解决复杂环境下决策问题的有效方法，并在多个学科领域受到大量关注[13,14]。近来，基于 DRL 进行 MEC 任务卸载决策的相关研究也已经开始出现，例如，文献[15]提出了一种基于深度 Q 学习(deep Q-learning，DQN)的算法来解决多用户共享网络边缘计算资源的卸载决策问题；文献[16]为具有能量收集能力的 UE 设计了基于 DQN 的卸载方案，以选择最优卸载基站和卸载能耗；文献[17]使用 DRL 来解决车辆在 MEC 环境中的任务卸载调度问题，以最小化其长期卸载成本。本文对典型 MEC 场景下移动应用中的任务卸载调度问题进行了研究，并提出了一种基于 DRL 的 DAG 调度算法，旨在最小化移动应用的执行时延。利用 DRL 对该问题进行求解能够带来诸多好处：a) 对于复杂的网络边缘环境，DRL 对最优调度策略的学习是免模型的(model-free)，该过程无须对环境建模，更无须专家知识，体现了强大的灵活性；b) DNN 优秀的表示能力和泛化能力能够很好地支持 DAG 调度问题的巨大状态空间；c) DRL 以优化长期卸载奖励为目标，这种优化方式将明显优于大多数基于贪心的启发式 DAG 调度算法，更可能求得(或接近)全局最优解。

本文首先对 DAG 中任务的优先级进行计算并根据该优先级将 DAG 转换为任务序列。通过对状态、动作和奖励的分别定义，该任务序列的调度过程被描述为一个马尔可夫决策过程(Markov decision process，MDP)。基于该决策过程的特殊形式，设计了一个序列到序列深度神经网络用于拟合该 MDP 的最优调度策略。为了有效地将 DAG 输入神经网络，设计了一种嵌入(embedding)方法，将 DAG 转换为序列数据。最后，采用近端策略优化(proximal policy optimization，PPO)[18]对该策略网络进行训练。

1 系统模型

1.1 系统架构

一个典型的 MEC 系统架构如图 1 所示。MEC 服务器被部署到网络边缘，为移动用户提供低延迟的就近计算服务。其为每个用户分配专门的软/硬件服务资源，并基于虚拟化技术进行资源隔离(如虚拟机、Cloudlet 或 Cloud-Clone 等[1,2])以保证服务质量和用户隐私。在用户侧，移动应用中的计算任务可以直接在 UE 的 CPU 上本地执行，也可以经数据传输单元(data transmission unit，DTU)发送到 MEC 服务器中(交由与用户对应的服务实例)进行远程卸载执行。卸载调度模块对 UE 中的所有任务进行调度决策，决定任务的执行方式(本地执行或卸载执行)和调度顺序。

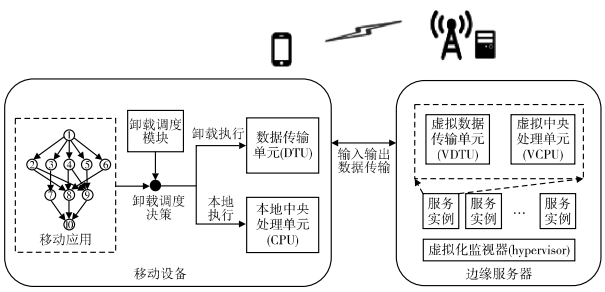

图1 MEC任务卸载架构

Fig.1 MEC task offloading architecture

通常，一个移动应用程序由具有依赖关系的多个计算任务构成(如机器视觉应用程序包含 10 ~ 30 个任务组件[3])，从而可以实现细粒度的任务卸载。将移动应用程序建模为一个 DAG，表示为 $G=(V,E)$。每个顶点 $v_i \in V$ 代表应用程序中的一个任务，每条有向边 $e(v_i,v_j) \in E$ 代表任务 v_j 的执行依赖于任务 v_i。在 $e(v_i,v_j) \in E$ 中，称 v_i 为 v_j 的前驱，v_j 为 v_i 的后继。一个任务只有在其所有前驱任务完成之后才能开始执行。在 DAG 中，没有任何前驱的任务被称为入口任务，没有任何后继的任务被称为出口任务，一个应用程序允许同时具有多个入口任务和出口任务。对每一个任务 $v_i \in V$，分别定义其执行所需的输入数据量(bit)、计算所需的 CPU 周期数和执行完毕对应的输出数据量(bit)为 D_i^{in}、C_i 和 D_i^{out}，它们的值可以通过执行程序分析器来获取[19]，代表了执行该任务所需的传输和计算成本。

如果任务 $v_i \in V$ 被调度为卸载执行，则 v_i 的执行将分为任务发送、边缘执行以及结果回传三个阶段。在任务发送阶段，输入数据 D_i^{in} 将由 DTU 上传到 MEC 服务器(对应的服务实例中)。设 UE 到 MEC 服务器的上行链路速率为 R^{ul} bps，则 v_i 的数据上传所需时间 T_i^{ul} 可以表示为 $T_i^{ul}=D_i^{in}/R^{ul}$。在边缘执行阶段，C_i 个 CPU 周期将在 MEC 服务实例上执行，将 MEC 服务器为该用户分配的 CPU 时钟频率(即对应服务实例的虚拟 CPU(virtual CPU，VCPU)时钟频率)表示为 F^s，则 v_i 在该阶段所需时间为 $T_i^s=C_i/F^s$。结果回传阶段与任务发送阶段类似，回传时间可以表示为 $T_i^{dl}=D_i^{out}/R^{dl}$，其中，$R^{dl}$ 为 MEC 服务器到 UE 的下行链路速率(也即对应服务实例的虚拟数据传输单元(virtual DTU，VDTU)的传输速率)。如果 $v_i \in V$ 被调度为本地执行，则不需要对任务数据进行上传或下载，其在本地的执行时间 T_i^l 可以表示为 $T_i^l=C_i/F^l$，其中，F^l 为 UE 上 CPU 的时钟频率。

在该系统架构中，所有处理单元(包括 UE 上的 CPU 和 DTU，以及服务实例中的 VCPU 和 VDTU)在同一时刻只能执行或发送一个任务，在处理过程中不允许抢占发生。与传统的异构环境 DAG 调度相比，该边缘计算场景下的 DAG 调度主要有以下三个特点：a) 每个节点有其自身的通信代价(任务输入与输出)，DAG 中的边仅指代任务之间的依赖关系；b) 由于网络边缘的带宽限制，UE 与服务实例之间的数据传输速率固定，当有多个任务需要传输时，必须进行排队；c) 退出节点的输出结果必须回传到 UE 才算调度结束。

1.2 任务调度

对 DAG 中的每一个任务 $v_i \in V$，分别采用 FT_i^{ul}、FT_i^s、FT_i^{dl} 和 FT_i^l 表示其任务发送、边缘执行、结果回传以及本地执行的完成时刻。若 v_i 被调度为本地执行，则 FT_i^{ul}、FT_i^s 和 FT_i^{dl} 的值没有意义，令 $FT_i^{ul}=FT_i^s=FT_i^{dl}=0$；同样地，若 v_i 被调度为卸载执行，则令 $FT_i^l=0$。

1.2.1 本地调度

在开始调度某任务 $v_i \in V$ 之前，必须确保其所有前驱都已经被调度。假设 v_i 被调度为本地执行，则该调度的就绪时刻(即可以进行该调度的最早时刻)被定义为

$$RT_i^l=\max_{v_j \in pred(v_i)}\max\{FT_j^l,FT_j^{dl}\} \tag{1}$$

其中：$pred(v_i)$ 代表 v_i 的所有前驱。可以看出，RT_i^l 是 v_i 的所有前驱均已完成执行并且它们的输出结果对 v_i 可用的最早时刻。若 v_i 的前驱 v_j 在本地执行，则其输出结果可用的时刻与其在本地执行完成的时刻相同，即 $\max\{FT_j^l,FT_j^{dl}\}=FT_j^l$；若 v_j 被调度为远程执行，其输出结果可用的时刻则为其结果回传完成的时刻，即 $\max\{FT_j^l,FT_j^{dl}\}=FT_j^{dl}$。

需要注意的是，由于任务在被本地 CPU 执行之前可能需要排队，v_i 真正开始执行的时刻 ST_i^l 并不一定等于其就绪时刻 RT_i^l，可得 $ST_i^l \geqslant RT_i^l$。其本地执行的完成时刻为 $FT_i^l=ST_i^l+T_i^l$。

1.2.2 卸载调度

假设 v_i 被调度为远程卸载执行，根据任务远程执行的三个阶段，分别定义 v_i 在这三个阶段的就绪时刻和开始时刻。发送任务 v_i 的就绪时刻被定义为

$$RT_i^{ul} = \max_{v_j \in pred(v_i)} \max\{FT_j^l, FT_j^{ul}\} \tag{2}$$

此时，v_i 的所有前驱，或者已经在 UE 本地执行完成，或者已经被发送到 MEC 服务实例上（发送 v_i 的输入数据并不要求其远程卸载的前驱执行完成）。与本地调度类似，开始发送 v_i 的时刻 ST_i^{ul} 还与等待 DTU 发送的任务的排队情况相关。可以得到 $ST_i^{ul} \geqslant RT_i^{ul}$。发送 v_i 的结束时刻为 $FT_i^{ul} = ST_i^{ul} + T_i^{ul}$。

任务 v_i 在 MEC 服务实例上执行的就绪时刻 RT_i^s 可表示为

$$RT_i^s = \max\{FT_i^{ul}, \max_{v_j \in pred(v_i)} FT_j^s\} \tag{3}$$

该就绪时刻 RT_i^s 依赖于以下两个事件：v_i 的输入数据已上传完成；v_i 的所有被远程卸载的前驱任务均已执行完成。值得注意的是，式(3)并没有对 v_i 在本地执行的前驱作约束（若 v_i 的前驱 v_j 在本地执行，则 $FT_j^s = 0$）。这是由于该约束已包含在式(2)中，允许上传 v_i 的输入数据即表明其在本地执行的前驱已全部执行完成；同理，v_i 开始执行的时刻依赖于 VCPU 上任务的排队情况，可得 $ST_i^s \geqslant RT_i^s$，同时 $FT_i^s = ST_i^s + T_i^s$。

当任务 v_i 执行完成时（时刻 FT_i^s），v_i 便进入回传就绪状态，即 $RT_i^{dl} = FT_i^s$；同理，v_i 开始进行数据回传的时刻 $ST_i^{dl} \geqslant RT_i^{dl}$，并且 $FT_i^{dl} = ST_i^{dl} + T_i^{dl}$。

1.3　问题表述

UE 上整个应用程序的执行完成时刻为其所有出口任务均已完成（并完成数据回传）的时刻，可通过式(4)得到

$$T^{total}(G) = \max_{v_i \in exit(G)} \max\{FT_i^l, FT_i^{dl}\} \tag{4}$$

其中：$exit(G)$ 代表 G 中所有出口任务的集合。由于 MEC 的应用场景主要针对延迟敏感的移动应用，为了最大化用户体验，需要找到针对不同的 G 的最优任务调度方案，以最小化 $T^{total}(G)$。在任务调度的过程中，需要确定每一个任务的执行方式（本地执行或卸载执行）和调度顺序（当同时存在多个就绪任务时）。值得注意的是，除了 UE 上的任务调度（执行方式和调度顺序），服务实例上的任务调度（已就绪任务的调度顺序）也会对全局调度的结果产生影响。本文规定服务实例上任务的调度顺序与 UE 中任务调度模块确定的任务调度顺序一致。

2　算法描述

2.1　调度优先级

与大多数基于 DAG 的调度算法相同[20]，该算法首先计算待调度 DAG 中每个任务的优先级，并基于该优先级按顺序对任务进行卸载调度决策。

调度优先级的计算方式与 HEFT 算法[21]类似。对任一 DAG（表示为 $G=(V,E)$），每个任务 $v_i \in V$ 的计算代价被定义为 $w_i = T_i^{ul} + T_i^s + T_i^{dl}$，即通过卸载执行该任务使用的最短时间（无排队）。基于 w_i，任务 v_i 的调度优先级 $P(v_i)$ 被递归地定义为

$$P(v_i) = \max_{v_j \in succ(v_i)} P(v_j) + w_i \tag{5}$$

其中：$succ(v_i)$ 代表 v_i 的所有后继。若 v_i 为退出任务，则

$$P(v_i) = w_i, v_i \in exit(G) \tag{6}$$

通过从退出任务开始遍历整个 DAG，能够递归地计算出所有任务的调度优先级。$P(v_i)$ 可以被认为是从任务 v_i 到退出任务间的关键路径长度。通过对 G 上的所有任务基于调度优先级降序排列，得到其调度任务序列，表示为

$$Q^G = (v'_1, v'_2, \cdots, v'_{|V|}) \tag{7}$$

其中：$|V|$ 为 G 中节点的个数。对 G 中任务的调度从 Q^G 的第一个元素 v'_1 开始，依次进行调度决策，直到所有任务均被执行完成。值得注意的是，Q^G 同时是 G 的一个拓扑排序，按此顺序进行任务调度确保了任务之间原有的依赖关系。

2.2　MDP 构造

针对任一 DAG（表示为 $G=(V,E)$），设其对应的已排序任务队列为 Q^G。对 Q^G 中的任务进行逐个调度决策的过程可以被建模为一个 MDP，写做 $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{P}_0, \mathcal{R}, \gamma)$。式中元素从左到右分别表示问题的状态空间、动作空间、状态转移矩阵、初始状态概率分布、奖励函数和折扣因子。

2.2.1　状态空间

将系统状态定义为正在被调度的 DAG 与该 DAG 上任务调度情况的结合。系统状态空间可以表示为

$$S = \{s \mid s = (G, k, \boldsymbol{A}^k)\} \tag{8}$$

其中：G 代表当前正在调度的 DAG（由 G 可直接获得 Q^G）；k 代表 Q^G 中当前已调度任务的个数（即前 k 个任务）；$\boldsymbol{A}^k$ 为具有 k 个元素的序列，描述对 Q^G 的前 k 个任务的调度情况。$\boldsymbol{A}^k$ 的定义为

$$\boldsymbol{A}^k = (a_1, a_2, \cdots, a_k) \tag{9}$$

其中：a_n 指对 Q^G 中第 n 个任务的调度决策。具体地，$a_n = 1$ 代表卸载执行，反之，代表本地执行。

2.2.2　动作空间

设当前状态为 $s_i = (G, k, \boldsymbol{A}^k)$。若 $k < |V|$（即 s_i 并非终止状态），需要对 Q^G 中第 $k+1$ 个任务进行调度决策。将动作空间定义为 $\mathcal{A} = \{0, 1\}$，基于当前状态 s_i，执行动作 $a_i = 1$ 表示对当前需要调度的任务（即第 $k+1$ 个任务）进行卸载；执行动作 $a_i = 0$ 则代表本地执行当前任务。值得注意的是，基于对状态空间与动作空间定义，从初始状态 s_0 开始，每执行一个动作（经过一次状态转换）就调度一个任务。因此，对任一状态 $s_i = (G, k, \boldsymbol{A}^k)$，状态转换的次数 i 与已调度任务的数量 k 恒相等，即 $i = k$。在后文中，为简化符号，将状态 $s_i = (G, k, \boldsymbol{A}^k)$ 写做 $s_i = (G, \boldsymbol{A}^i)$；同时，$\boldsymbol{A}^i$ 所描述的任务调度情况实质上就是从 s_0 开始到当前状态的历史动作序列，本文采用同一符号（即 a）表示。

2.2.3　奖励函数

已知任一状态 $s_i = (G, \boldsymbol{A}^i)$，可得已调度的任务序列为 Q^G 的前 i 项，将该已调度的任务序列表示为 $Q_{1:i}^G$。由于 Q^G 为 $G = (V, E)$ 的一个拓扑排序，基于其任一已调度的任务序列（即 $Q_{1:i}^G$）可构造 G 的一个已调度子图，表示为 $G_{1:i} = (V', E')$。其中，$V' \subseteq V$ 为所有已调度任务的集合（即序列 $Q_{1:i}^G$ 中的所有任务）；$E' \subseteq E$ 为所有连接已调度任务的边的集合。可容易看出，$G_{1:|V|} = (V, E) = G$；而 $G_{1:0} = (\varnothing, \varnothing)$ 代表图为空。为最小化 G 的调度时间，将奖励函数 $r_i = \mathcal{R}(s_i, a_i)$ 定义为

$$r_i = \mathcal{R}(s_i, a_i) = T^{total}(G_{1:i}) - T^{total}(G_{1:i+1}) \tag{10}$$

即在状态 s_i 下执行动作 a_i 以前与执行动作 a_i 以后所对应的已调度子图的执行时间之差。

2.2.4　MDP 过程

将卸载调度模块的调度策略定义为一个条件概率函数，表示为 $\pi(a_i \mid s_i)$。策略函数 π 基于当前状态 s_i，给出选择不同动作（本地执行或卸载执行）的概率。从初始状态 s_0 开始，根据 $\pi(a_i \mid s_i)$，卸载调度模块每执行一个动作，系统即进入一个新的状态并得到一个奖励，直到 Q^G 中的最后一个任务（即第 $|V|$ 个任务）执行完成。整个任务调度过程可以表示为

$$(s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_{|V|}) \tag{11}$$

其中：$s_{|V|}$ 即为终止状态，表示所有任务调度已经完成。该过程的带折扣累计奖励为

$$R = \sum_{i=0}^{|V|} \gamma^i r_i = \sum_{i=0}^{|V|} \gamma^i (T^{total}(G_{1:i}) - T^{total}(G_{1:i+1})) \tag{12}$$

若折扣因子 $\gamma = 1$，可得

$$R = T^{total}(G_{1:0}) - T^{total}(G_{1:|V|}) = -T^{total}(G) \tag{13}$$

该式表明，对任一 G，最大化其任务调度过程的累计奖励与最小化其任务调度时间是一致的。

若该 MDP 的状态转移矩阵$\mathcal{P}$已知，则可以通过 Bellman 方程递归地表示出该 MDP 的值函数[10]。之后，再通过值迭代或者策略迭代可求出最优调度策略 $\pi(a_i|s_i)$[11]。但是，由于被调度的 DAG 的多样性，该问题的状态空间无穷大，不可能预先求得该状态转移矩阵$\mathcal{P}$，所以本文采用 DRL 为卸载调度模块寻找最优策略。

2.3 神经网络架构

本文采用深度神经网络对策略函数 $\pi(a_i|s_i)$ 进行拟合。这里将通过神经网络拟合的调度策略称为策略网络，表示为 $\pi_\theta(a_i|s_i)$，其中 θ 为神经网络的参数。神经网络的输入为任一状态 $s_i=(G,\boldsymbol{A}^i)$。由于无法将 G 直接输入神经网络，这里将其表示为一个任务嵌入的序列，写做 $\boldsymbol{T}^G=(t_1,t_2,\cdots,t_{|V|})$。该序列中，任务嵌入的排序与 $\boldsymbol{Q}^G$ 一致。每一个任务嵌入由以下三部分构成：a) 一个包含任务上传时间 T_i^{ul}、边缘执行时间 T_i^e、数据回传时间 T_i^{dl} 以及本地执行时间 T_i^l 的向量；b) 一个包含该任务所有前驱的索引向量；c) 一个包含该任务所有后继的索引向量。保存前驱/后继索引的两个向量被设置为定长 p。若任务的前驱/后继的数量小于 p，则使用 -1 填充；若大于 p，则在嵌入中忽略超出的前驱/后继。如此，任一输入状态 $s_i=(G,\boldsymbol{A}^i)$ 被转换为一个任务嵌入序列 $\boldsymbol{T}^G$ 和一个历史动作序列 $\boldsymbol{A}^i$。

神经网络的输出为基于当前状态的可执行动作的概率分布。由于前一步调度决策的动作 a_i 需要作为后一步任务状态 s_{i+1}的一部分（即 $\boldsymbol{A}^{i+1}$的最后一项），本文采用图 2 所示的序列到序列（sequence to sequence，S2S）神经网络架构。该 S2S 网络由编码器（encoder）和解码器（decoder）两部分组成，两部分均由多层循环神经网络（recurrent neural network，RNN）实现。编码器依次接收任务嵌入序列 $\boldsymbol{T}^G$，并输出其最终的隐藏层作为 DAG 的特征；解码器利用编码器的输出初始化其自身的隐藏层，其逐步按顺序输入历史调度动作 $\boldsymbol{A}^i$，并逐步输出策略函数与值函数的结果，以确定下一步动作 a_i，直到到达终止状态 $s_{|V|}$。在该网络架构中，策略函数与值函数共享除输出层外的所有网络结构和参数（策略函数采用 softmax 输出层；而值函数采用全连接输出层）。这是考虑到对估计值函数有帮助的特征也有助于进行动作选择，反之亦然。该网络架构还引入了注意力机制（attention mechanism）[22]，以便更好地找到 DAG 结构中的关键特征。

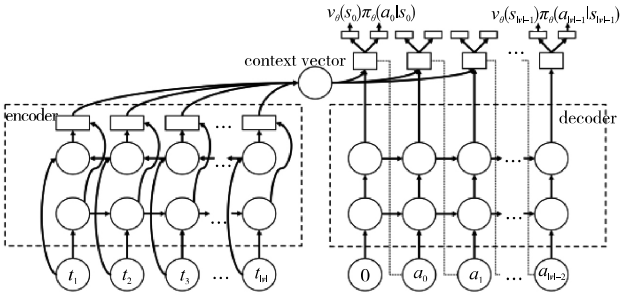

图2 S2S神经网络架构

Fig.2 S2S neural network architecture

2.4 训练方法

为获得最优的任务调度策略 π_θ，DRL 的训练目标可以写为

$$\max_\theta L(\theta)=\max_\theta E\left[\pi_\theta^*(\boldsymbol{A}^{|V|}|G)\sum_{i=0}^{|V|-1}\mathcal{R}(s_i,a_i)\right] \tag{14}$$

其中：

$$\pi_\theta^*(\boldsymbol{A}^{|V|}|G)=\prod_{i=0}^{|V|-1}\pi_\theta(a_i|G,\boldsymbol{A}^i)=\prod_{i=0}^{|V|-1}\pi_\theta(a_i|s_i) \tag{15}$$

可以看出 $\pi_\theta^*(\boldsymbol{A}^{|V|}|G)$ 为单步卸载策略 $\pi_\theta(a_i|G,\boldsymbol{A}^i)$ 的累计连乘，代表基于图 G 的不同调度决策序列的概率分布。训练的最终目的是最大化不同调度决策序列下所获奖励的期望。采用近端策略优化（PPO）对策略网络进行训练。作为一种基于策略的 DRL 算法，PPO 具有很好的稳定性和可靠性，有助于训练的有效收敛[18]。

由于被调度的 DAG 的多样性，该问题的状态空间无穷大，不可能对所有 DAG 进行探索。算法在部署后持续对系统中已经调度过的 DAG 进行收集，以构造训练 DAG 集，并在该训练 DAG 集上进行调度策略训练。基于深度神经网络优秀的泛化能力，训练所得的任务调度策略也可以很好地对训练集之外的 DAG 进行调度。可以认为，深度神经网络通过训练很好地学习到了 DAG 调度的通用模式，这一点可以在后续实验中得到验证。为帮助训练更好地收敛，在训练过程中对单步所获奖励值进行了缩放。通过将单步所获奖励值除以调度过程中所获的最大奖励值，将单步奖励值保持在 [0,1]。

3 仿真实验及分析

3.1 实验设计

实验将对本文基于 DRL 的计算卸载调度方法（DRL-based offloading scheduling method，DRLOSM）进行验证。考虑一个典型的 MEC 任务卸载调度场景：UE 上的 CPU 时钟频率为 $F^l=1\times10^9$ 周期/s，MEC 服务实例的 VCPU 时钟频率默认为 $F^s=8\times10^9$ 周期/s，假设 UE 到 MEC 服务器的上下行链路传输速率相等，默认为 8 Mbps，即 $R^{ul}=R^{dl}=8$。

为模拟应用程序（即 DAG）的多样性，文献[23]所采用的 DAG 生成器被实现，以随机构造训练和测试时使用的 DAG。使用以下参数对 DAG 进行随机构造：a) 任务规模（n），所构造的 DAG 中的任务数量；b) 图的宽度（fat），在相同的任务规模下，较小的 fat 值可构造较“瘦高”的 DAG，fat 较大时会导致较“矮胖”的 DAG 生成，实验中该参数的值随机地从集合{0.1，0.3，0.5，0.7，0.9}中选取；c) 图的密度（density），该参数用以描述 DAG 中两个连续任务层之间依赖关系的数量，为生成较为稠密的依赖关系，实验中设置该参数大于 0.5，其取值集合为{0.5，0.6，0.7，0.8，0.9}；d) 通信/计算比（communication to computation ratio，CCR），该参数描述所生成的 DAG 的平均通信成本与平均计算成本的比率，低 CCR 意味着应用程序为计算密集型，反之则代表其为通信密集型，由于大多数适合卸载的移动应用为计算密集型，在实验中设置该参数小于 0.5，取值集合为{0.3，0.4，0.5}。

对生成的 DAG 中的每个任务 v_i，其输入输出数据大小被设置为相同（即 $D_i^{in}=D_i^{out}$），取值在 [5，50] KB 随机选择。基于传输数据量和 CCR 可直接求得各任务执行所需的 CPU 周期数。实验中，任务规模 n 在 10～50 取值，步长为 5。每种任务规模下，生成 1 000 个用于训练的 DAG 和 100 个用于测试的 DAG，即共有 9 000 个训练 DAG 和 900 个测试 DAG。值得注意的是，所有的测试 DAG 均未出现在训练 DAG 的集合中，以验证深度神经网络能否学习到 DAG 调度的通用模式。

实验使用 TensorFlow[24] 对提出的神经网络进行实现和训练。编码器被设置为具有 256 个隐藏神经元的两层双向长短期记忆（LSTM）单元；解码器被设置为具有 256 个隐藏神经元的两层 LSTM。此外，编码器和解码器都使用了层规范化（layer normalization，LN）[25] 以提升训练效率。MDP 的折扣因子 γ 被设置为 0.99；任务嵌入中的前驱/后继向量长度 p 被设置为 12。选择 Adam 作为训练过程中的优化器，学习率被设置为 10^{-4}。

3.2 实验结果与分析

首先对算法的收敛能力进行验证。基于 9 000 个训练 DAG 对提出的 DRLOSM 进行训练，并在训练过程中记录训练结果。在每个训练周期（epoch）结束时，将一批测试 DAG 输入到当前训练的 S2S 策略网络中，得到该批测试 DAG 的卸载调度决策。随后，基于该决策在仿真环境中对该批 DAG 进行模拟调度，并记录该次测试所获得的平均奖励。如图 3 所示，平

均奖励在训练前期急剧增加，在 500 个 epoch 后开始趋于稳定。该实验验证了 DRLOSM 良好的收敛能力。

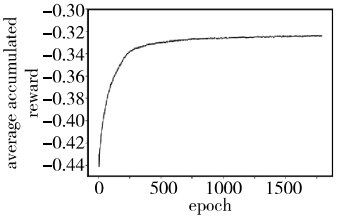

图3　DRLOSM训练曲线
Fig. 3　Training curve of DRLOSM

随后将 DRLOSM 与以下六个基线算法进行对比：

a) 穷举法（exhaustive search，ES）。该方法尝试所有可能的调度方式并记录最优调度结果，即 DAG 的最短执行时间。由于指数级的复杂度，该方法仅适用于任务规模较小的 DAG。

b) 本地执行（local execution，LE）。这是任务卸载概念出现之前移动计算主要的任务执行方式[3]。由于所有任务均在 UE 本地执行，导致很高的 DAG 调度时延。

c) 卸载执行（offloading execution，OE）。DAG 中的所有任务均被调度为卸载执行。这是移动云计算和 MEC 提出时应用程序中任务的理想执行方式[3]。

d) 随机调度（random scheduling，RS）。随机决定 DAG 中任务的调度方式（不违背原任务依赖关系），RS 是任务调度的通用基线算法[17]，一个调度算法的性能应至少高于该算法。

e) 循环调度（round-robin，RR）。该算法是分布式系统中任务调度的经典算法之一[24]，其采用循环的方式将拓扑排序之后的任务依次分配到不同的执行地点（本地或服务实例）。

f) 基于 HEFT 的调度（HEFT-based）。将 DRLOSM 与经典启发式调度算法 HEFT[21] 进行对比，由于两个算法所处理的 DAG 在结构上略有不同（见 1.1 节），对标准 HEFT 进行改进以适应该 MEC 场景。与标准 HEFT 相同，HEFT-based 先基于权重对任务进行排序，之后根据最早完成时间对任务进行调度。

（a）对不同任务规模下各卸载算法的平均 DAG 调度时延进行比较，实验结果如表 1 所示。ES 所对应的结果为最优解，但由于该算法指数级的复杂度，本实验只能给出 $n \leqslant 20$ 时的结果；由于任务无法并行，在当前传输速率（$R^{ul} = R^{dl} = 8$）和处理能力（$F^s = 8 \times 10^9$）下，LE 和 OE 的调度时延均很高。可以看出，RR 的平均表现严格优于 RS，而 HEFT-based 的平均表现又严格优于 RR，HEFT-based 优秀的性能是其被广泛研究和使用的原因。而提出的 DRLOSM 在每一个任务规模上的平均表现均优于 HEFT-based 且非常接近最优解。值得注意的是，对各算法的测试是在所生成的 900 个测试 DAG 上进行的，即 DRLOSM 在训练过程中并未见过这些样本。从实验结果可以看出，DRLOSM 很好地学习到了 DAG 调度的通用模式。

表 1　各卸载算法的平均 DAG 调度时延
Tab. 1　Average DAG scheduling delay with different algorithms　/ms

节点数	ES	LE	OE	RS	RR	HEFT-based	DRLOSM
10	476.5	723.1	610.5	612.7	605.3	514.6	489.8
15	643.7	1053.4	870.0	862.0	832.6	719.7	660.7
20	826.0	1394.4	1160.0	1080.2	1068.0	925.5	852.5
25	N/A	1796.1	1428.9	1370.2	1313.2	1145.6	1017.6
30	N/A	2154.7	1736.5	1648.8	1591.9	1399.0	1236.5
35	N/A	2463.7	1973.6	1958.0	1907.8	1665.5	1468.4
40	N/A	2910.5	2414.8	2192.1	2114.0	1864.5	1679.8
45	N/A	3182.1	2480.6	2271.8	2187.6	1955.4	1678.7
50	N/A	3663.0	3118.1	2725.3	2572.8	2287.7	2082.4

（b）在不同的环境下对 DRLOSM 的调度策略进行重新训练，并评估不同环境对各算法性能的影响。为使得最优解在实验中均可用，将 DAG 节点的规模限制为 15（由于穷举法的时间复杂度太高）。图 4 和 5 分别显示了各算法在不同网络传输速度下（R^{ul} 和 R^{dl}）和不同服务实例处理能力下（F^s）的平均 DAG 调度时延（其他参数均为默认值）。可以看出，随着网络传输速度以及服务实例处理能力的提升，除了完全不进行任务卸载的 LE，所有算法的调度时间都在下降。在网络传输速度提升较大的情况下，OE 的性能甚至在某些情况超过了 HEFT-based。在所有情况下，DRLOSM 的表现均与最优解最为接近，体现了该算法优秀的性能和对不同环境的适应性。

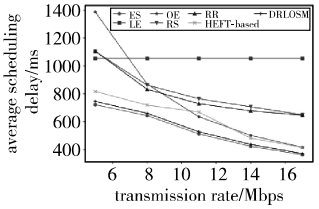

图4　不同网络传输速率下的平均DAG调度时延
Fig.4　Average DAG scheduling delay under different transmission rate

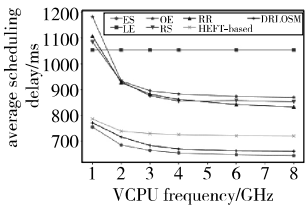

图5　不同服务实例VCPU频率下的平均DAG调度时延
Fig.5　Average DAG scheduling delay under different VCPU frequency

4　结束语

本文将 DRL 应用于 MEC 场景下移动应用的任务卸载调度问题，旨在最小化移动应用的执行时延。该任务卸载调度的过程首先被描述为一个 MDP；随后设计了一个 S2S 的 DNN 用于对最优调度策略进行拟合，并采用 PPO 对该策略网络进行训练；最后，通过与六个基线算法的对比证明了采用 DRL 方法解决 MEC 场景中复杂调度问题的有效性和可靠性。

参考文献：

[1] Abbas N, Zhang Yan, Taherkordi A, *et al*. Mobile edge computing: a survey [J]. IEEE Internet of Things Journal, 2018, 5(1): 450–465.

[2] Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading [J]. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1628–1656.

[3] Ra M R, Sheth A, Mummert L, *et al*. Odessa: enabling interactive perception applications on mobile devices [C] //Proc of the 9th International Conference on Mobile Systems, Applications, and Services. New York: ACM Press, 2011: 43–56.

[4] Ullman J D. NP-complete scheduling problems [J]. Journal of Computer and System Sciences, 1975, 10(3): 384–393.

[5] 李金忠，夏洁武，曾劲涛，等. 网格工作流调度算法研究综述 [J]. 计算机应用研究，2009，26(8)：2816–2820. (Li Jinzhong, Xia Jiewu, Zeng Jintao, *et al*. Survey on grid workflow scheduling algorithm [J]. Application Research of Computers, 2009, 26(8): 2816–2820.)

[6] Mahmoodi S E, Uma R N, Subbalakshmi K P. Optimal joint scheduling and cloud offloading for mobile applications [J]. IEEE Trans on Cloud Computing, 2016, 7(2): 301–313.

[7] Lin Xue, Wang Yanzhi, Xie Qing, *et al*. Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment [J]. IEEE Trans on Services Computing, 2015, 8(2): 175–186.

[8] Cheng Zixue, Li Peng, Wang Junbo, *et al*. Just-in-time code offloading for wearable computing [J]. IEEE Trans on Emerging Topics in Computing, 2015, 3(1): 74–83.

[9] Tsai C W, Rodrigues J J P C. Metaheuristic scheduling for cloud: a survey [J]. IEEE Systems Journal, 2013, 8(1): 279–291.

[10] Li Yuxi. Deep reinforcement learning: an overview [EB/OL]. (2018–11–26). https://arxiv.org/pdf/1701.07274.pdf.

[11] Sutton R S, Barto A G. Reinforcement learning: an introduction [M]. 2nd ed. Cambridge, MA: MIT Press, 2014.　（下转第 263 页）

WSN-Random 实现的网络生命周期相比,场景 WSN-Grid 的网络生命周期更高。这是因为在 WSN-Grid 网络场景的每个网格单元的交叉点处预先确定了潜在位置,使得潜在位置的分布均匀,而且传感器节点之间的负载分布也很容易达到均匀状态。上述实验结果显示,本文方案在 WSN-Grid 场景中的性能更好。

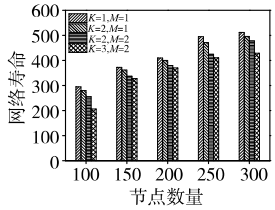

图7　WSN-Random场景在网络寿命方面的性能比较

Fig.7　Performance comparison of WSN-Random scenario in terms of network life

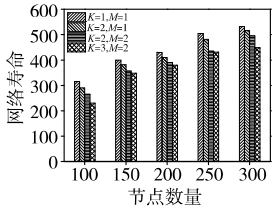

图8　WSN-Grid场景在网络寿命方面的性能比较

Fig.8　Performance comparison of WSN-Grid scenario in terms of network life

4　结束语

本文研究了无线传感器的 K-覆盖和 M-连通适配位置节点布置问题,提出了一种基于 BBO 的传感器节点部署方案。该方案首先为栖息地的表示提供一种有效的编码方案,然后结合 BBO 的迁移和突变算子构建一个多目标函数,通过优化目标函数来选择用于定位传感器节点的最优位置。通过讨论不同 K 和 M 组合下,本文方案寻找合适位置的近似最优数的结果来评估算法的性能。实验结果表明,相对于其他算法,本文方法具有显著的优越性。未来将进一步优化本文方法,增加网络寿命。

参考文献:

[1] Özdağ R, Canayaz M. A new dynamic deployment approach based on whale optimization algorithm in the optimization of coverage rates of wireless sensor networks [J]. European Journal of Technique, 2017,7(2):119-130.

[2] Randhawa S, Jain S. MLBC: multi-objective load balancing clustering technique in wireless sensor networks [J]. Applied Soft Computing,2019,74(1):66-89.

[3] Vatankhah A, Babaie S. An optimized bidding-based coverage improvement algorithm for hybrid wireless sensor networks [J]. Computers & Electrical Engineering,2018,65(4):1-17.

[4] Deif D S, Gadallah Y. An ant colony optimization approach for the deployment of reliable wireless sensor networks [J]. IEEE Access, 2017,5:10744-10756.

[5] Moh'd Alia O, Al-Ajouri A. Maximizing wireless sensor network coverage with minimum cost using harmony search algorithm [J]. IEEE Sensors Journal,2017,17(3):882-896.

[6] El K Y, Tahiri A, Abtoy A, *et al*. A hybrid algorithm for optimal wireless sensor network deployment with the minimum number of sensor nodes [J]. Algorithms,2017,10(3):80-98.

[7] Cao Bin, Zhao Jianwei, Lyu Zhihan, *et al*. Deployment optimization for 3D industrial wireless sensor networks based on particle swarm optimizers with distributed parallelism [J]. Journal of Network and Computer Applications,2018,103(1):225-238.

[8] Naik C, Shetty D P. A novel meta-heuristic differential evolution algorithm for optimal target coverage in wireless sensor networks [C]//Proc of the 9th International Conference on Innovations in Bio-Inspired Computing and Applications. Berlin: Springer,2018:83-92.

[9] Harizan S, Kuila P. Coverage and connectivity aware energy efficient scheduling in target based wireless sensor networks: an improved genetic algorithm based approach [J]. Wireless Networks,2019,25(4):1995-2011.

[10] 郭超,杨宇轩,胡荣磊,等. 基于粒子群算法的 WSN 覆盖优化 [J]. 计算机应用研究,2020,37(4):1170-1173. (Guo Chao, Yang Yuxuan, Hu Ronglei, *et al*. WSN coverage optimization based on particle swarm optimization [J]. Application Research of Computers, 2020,37(4):1170-1173.)

[11] Hanh N T, Binh H T T, Hoai N X, *et al*. An efficient genetic algorithm for maximizing area coverage in wireless sensor networks [J]. Information Sciences,2019,488(2):58-75.

[12] 张景昱,刘京菊,叶春明. 基于区域分割和 Voronoi 图的区域覆盖算法 [J]. 计算机应用研究,2020,37(10):3116-3120. (Zhang Jingyu, Liu Jingju, Ye Chunming. Area coverage algorithm based on region segmentation and Voronoi diagram [J]. Application Research of Computers,2020,37(10):3116-3120.)

[13] Yue Yinggao, Li Cao, Luo Zhongqiang. Hybrid artificial bee colony algorithm for improving the coverage and connectivity of wireless sensor networks [J]. Wireless Personal Communications, 2019,5(1):1-14.

(上接第 245 页)

[12] 陈学松,杨宜民. 强化学习研究综述 [J]. 计算机应用研究,2010,27(8):2834-2838. (Chen Xuesong, Yang Yimin. Reinforcement learning: survey of recent work [J]. Application Research of Computers,2010,27(8):2834-2838.)

[13] Silver D, Hubert T, Schrittwieser J, *et al*. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play [J]. Science,2018,362(6419):1140-1144.

[14] Mnih V, Kavukcuoglu K, Silver D, *et al*. Human-level control through deep reinforcement learning [J]. Nature,2015,518(7540):529-533.

[15] Li Ji, Gao Hui, Lyu Tiejun, *et al*. Deep reinforcement learning based computation offloading and resource allocation for MEC [C]//Proc of IEEE Wireless Communications and Networking Conference. Piscataway,NJ: IEEE Press,2018:1-6.

[16] Chen Xianfu, Zhang Honggang, Wu C, *et al*. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning [J]. IEEE Internet of Things Journal,2019,6(3):4005-4018.

[17] Zhan Wenhan, Luo Chunbo, Wang Jin, *et al*. Deep reinforcement learning-based computation offloading in vehicular edge computing [C]//Proc of IEEE Global Communications Conference. Piscataway, NJ: IEEE Press,2019:1-6.

[18] Schulman J, Wolski F, Dhariwal P, *et al*. Proximal policy optimization algorithms [EB/OL]. (2017-08-28). https://arxiv.org/pdf/1707.06347.pdf.

[19] Lyu Xinchen, Tian Hui, Sengul C, *et al*. Multiuser joint task offloading and resource optimization in proximate clouds [J]. IEEE Trans on Vehicular Technology,2017,66(4):3435-3447.

[20] Wu Fuhui, Wu Qingbo, Tan Yusong. Workflow scheduling in cloud: a survey [J]. Journal of Supercomputing,2015,71(9):3373-3418.

[21] Topcuoglu H, Hariri S, Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel and Distributed System,2002,13(3):260-274.

[22] Luong M T, Pham H, Manning C D. Effective approaches to attention-based neural machine translation [EB/OL]. (2015-09-20). https://arxiv.org/pdf/1508.04025.pdf.

[23] Arabnejad H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table [J]. IEEE Trans on Parallel and Distributed Systems,2013,25(3):682-694.

[24] Abadi M, Barham P, Chen Jianmin, *et al*. TensorFlow: a system for large-scale machine learning [C]//Proc of the 12th USENIX Symposium on Operating Systems Design and Implementation. Berkeley,CA: USENIX Association,2016:265-283.

[25] Ba J L, Kiros J R, Hinton G E. Layer normalization [EB/OL]. (2016-07-21). https://arxiv.org/pdf/1607.06450.pdf.

[26] Tanenbaum A S, Van Steen M. Distributed systems: principles and paradigms [M]. Upper Saddle River,NJ: Prentice Hall,2007.