Software Engineering and Applications 软件工程与应用, 2022, 11(1), 91-100
Published Online February 2022 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2022.111011

Hans汉斯

基于DDPG的MEC视频任务卸载算法

程浩宇

浙江理工大学信息学院，浙江 杭州

收稿日期：2021年12月13日；录用日期：2022年2月3日；发布日期：2022年2月10日

摘 要

近年来移动边缘计算(Mobile Edge Computing, MEC)的出现满足了边缘设备(Edge Device, ED)处理视频数据任务的需求，ED通过将任务卸载到MEC服务器，来缓解自身计算能力不足的缺点。然而MEC系统中网络的时变性和任务生成的动态性，为求解任务卸载问题带来了挑战。本文考虑一个多ED的MEC系统，将MEC系统的任务处理时延作为优化目标。为了最小化任务处理时延，将原问题模型转化为马尔科夫决策过程(Markov Decision Process, MDP)。考虑到任务动态性及网络时变性，本文采用一种基于深度强化学习的深度确定性策略梯度算法(Deep Deterministic Policy Gradient, DDPG)来解决MDP问题。仿真结果表明，该算法能够有效地降低任务处理时延，并优于其他基准卸载策略。

关键词

移动边缘计算，深度强化学习，任务卸载

Video Task Offloading Algorithm Based on DDPG in MEC

Haoyu Cheng

School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou Zhejiang

Received: Dec. 13th, 2021; accepted: Feb. 3rd, 2022; published: Feb. 10th, 2022

Abstract

In recent years, the emergence of Mobile Edge Computing (MEC) meets the needs of Edge Device (ED) to process video data tasks. ED alleviates its lack of computing capacity by offloading tasks to MEC server. However, the time variability in MEC and the dynamics of task generation bring challenges to the task offloading problem. In this paper, a multi-EDs MEC system is considered, and the task processing latency of the MEC system is taken as the optimization objective. In order to mi-

文章引用: 程浩宇. 基于 DDPG 的 MEC 视频任务卸载算法[J]. 软件工程与应用, 2022, 11(1): 91-100.
DOI: 10.12677/sea.2022.111011

nimize the latency of task processing, the original problem is transformed into a Markov Decision Process (MDP). Considering the dynamics of task and time variability of network, this paper exploits the Deep Deterministic Policy Gradient (DDPG) algorithm based on deep reinforcement learning to solve the MDP problem. Simulation results show that this algorithm can effectively reduce the task processing latency and is better than other baseline offloading policies.

Keywords

Mobile Edge Computing, Deep Reinforcement Learning, Task Offloading

1. 引言

移动边缘计算(Mobile Edge Computing, MEC)作为近年一种新型计算架构，有效地解决了云计算中时延和网络负载等问题，使得物联网设备和应用程序的网络响应、实时分析、访问安全性等需求得以满足[1] [2]。随着 MEC 的出现，越来越多的边缘设备通过部署深度学习模型使其能够运行目标检测、视频分析和处理等计算密集型的任务应用[3]。然而该类任务通常也是时延敏感型的，大多的边缘设备和移动设备自身没有足够强大的计算能力来满足这一需求，因此，针对时延敏感型的任务卸载问题近年来得到越来越多的关注与研究。文献[4]中针对时延敏感型且不可分的任务卸载问题，以最小化长期成本为目标，提出了一种基于深度强化学习的无模型算法，该算法能够高效利用边缘节点的处理能力，显著降低系统时延。文献[5]研究了动态 MEC 场景下的计算卸载问题，通过提出的深度强化学习算法优化卸载决策，实现任务完成数最大化的同时能耗也最小化。

在本文中所考虑的视频分析、处理类任务也属于 MEC 中时延敏感型任务，但因为视频数据任务的分析、处理需要多个具体的操作(如视频编解码、转码等)。例如道路交通中车辆检测的视频数据，需要编解码、运行深度学习模型推理来检测车辆道路状况[6]；工业生产中的质量检测也需要对生产线上工业摄像头采集的视频数据加以处理并分析。因此，本文通过对 MEC 场景中视频分析任务的卸载问题建立具体的模型，对时延敏感型任务建立对应的优化目标。为了最小化视频任务的处理时延，我们将原问题转化为马尔科夫决策过程，同时考虑 MEC 系统动态性与任务到达随机性，采用深度确定性策略梯度算法(Deep Deterministic Policy Gradient, DDPG)适应动态环境优化边缘设备的卸载决策。最后，通过仿真验证本文基于深度强化学习的卸载策略能够有效降低系统时延成本，且优于其他基准卸载方案。

2. 系统模型

本文研究的 MEC 系统，由一个 MEC 服务器(MEC Server, MECS)和多个边缘设备(Edge Device, ED)组成。这里将 $\mathcal{N}=\{1,2,\cdots,N\}$ 定义为 Eds 的集合，ED_n 表示为边缘设备集合 $\mathcal{N}$ 中第 n 个 ED，本文中假定所有 ED 的硬件配置是相同的，即计算能力相同。

2.1. 任务模型

本文将视频处理任务具体化为视频数据的目标检测任务，即设备端运行深度学习(Deep Learning, DL)模型检测视频中每一帧的目标物体并对其分类。在该 MEC 系统中，各 ED 每隔 Δ 秒到达一个视频任务的

处理需求。首先，该视频任务的原视频格式转码为 RGB 格式，然后将 RGB 格式的帧数据输入到 DL 模型中进行目标检测，最后保存并分析 DL 模型推理结果，以完成整个视频任务。我们将系统时间定义为一个时隙集合 $\mathcal{T}=\{1,2,\cdots,T\}$，将 ED_n 在 t 时刻($t\in\mathcal{T}$)到达的视频任务用二元组 $\left(B_n(t),D_n(t)\right)$ 表示。其中 $B_n(t)$ 表示视频任务数据大小，单位为(bit)；$D_n(t)$ 表示其视频任务的时长，单位为(秒)。

本文采用对每个视频任务细粒度划分的方式，将单个视频任务切分为多个时长相等的视频块。每个视频块的时长由 d 表示，单位为(秒)，一个视频任务 $\left(B_n(t),D_n(t)\right)$ 切分后的视频块的数量为 $D_n(t)/d$。这里为了简化问题模型，本文假定每个视频块的数据大小是相等的，即 $B_n(t)\cdot d/D_n(t)$ bits。ED_n 在 t 时刻的视频任务被切分为 $D_n(t)/d$ 个视频块，那么 MEC 系统的卸载决策就转换为决定每个视频块是否卸载到 MECS 上，或其在本地设备处理。$\lambda_n(t)$ 表示 ED_n 对 t 时刻到达的视频任务的卸载决策，这里 $\lambda_n(t)\in[0,1]$。因此，对于任务 $\left(B_n(t),D_n(t)\right)$，卸载到 MECS 处理的视频块数量 $M_n^{off}(t)$ 表示如下：

$$M_n^{off}(t)=\left\lceil \lambda_n(t)\frac{D_n(t)}{d}\right\rceil \tag{1}$$

$\lceil\cdot\rceil$ 项表示卸载到 MECS 的视频块数量必须为整数，因此根据式(1)，剩下在本地设备 ED_n 处理的视频块数量 $M_n^{loc}(t)=D_n(t)/d-M_n^{off}(t)$。每个视频块完成转码操作所需的 CPU 周期数为 C_0，完成 DL 模型推理所需的 CPU 周期数为 C_2。

2.2. 时延模型

2.2.1. 本地计算模型

在本文的 MEC 系统中，每个视频块须进入其所属 ED 的处理队列中排队处理。本地计算模型中，每个 ED 包含两个处理队列，分别为转码队列和 DL 模型推理队列(下文简称推理队列)，其在 t 时刻起始时长度分别为 $Q_n^{e,0}(t)$ 和 $Q_n^{e,1}(t)$。对于 ED_n 在 t 时刻视频任务中将在本地处理 $M_n^{loc}(t)$ 个视频块的第 i 个视频块，我们将 $L_{n,i}^{e,tc}$ 定义为该视频块完成转码的预计时延，其计算方式如下：

$$L_{n,i}^{e,tc}=\frac{Q_n^{e,0}(t)}{f_c^e}C_0+\frac{i}{f_c^e}C_0 \tag{2}$$

上式(2)中 $i\in\left[0,M_n^{loc}(t)\right]$，$f_c^e$ 表示各 ED 的 CPU 频率，单位为(周期/秒)。以第 i 个视频块为例，在完成转码后的 RGB 帧数据进入推理队列排队等待推理，因此其完成推理的预计时延表示为

$$L_{n,i}^{e}=\max\left(\frac{Q_n^{e,1}(t)}{f_g^e}C_1,L_{n,i}^{l,tc}\right)+\frac{C_1}{f_g^e} \tag{3}$$

式(3)中 $\max(\cdot)$ 项代表该视频块必须在其完成转码，并且推理队列中剩余任务完成推理后，才能开始推理。那么对于 t 时刻，ED_n 处理完全部任务的预期时延可表示为最后一个视频块在本地处理完成的时延：

$$L_n^e(t)=L_{n,M_n^{loc}(t)}^e \tag{4}$$

需要注意的是，推理结果通常为数据量较小的文本文件，因此结果的收集时延可以忽略不计。

2.2.2. 卸载计算模型

当 ED_n 对视频任务 $\left(B_n(t),D_n(t)\right)$ 中 $M_n^{off}(t)$ 个视频块卸载到 MECS 处理时，其整个过程由三部分组成。

1) 转码：与本地处理模型一样的是，视频块的原数据格式需要转码为 RGB 格式。该过程所需时延与上一小节中本地计算模型 $L_{n,i}^{e,tc}$ 的计算方式一致，这里不重复给出。

2) 传输：当一个视频块转码完成后，其进入 ED_n 的传输队列 $Q_n^{trans}(t)$，视频块 RGB 帧数据通过无线信道传输至 MECS。对于 t 时刻 ED 与 MECS 间的传输速率，我们定义为

$$v(t)=W(t)\log_2\left(1+\frac{P_v h}{\sigma^2}\right) \tag{5}$$

这里 $W(t)$ 表示 t 时刻的上行信道带宽，P_v 和 h 分别表示 ED 的传输功率和无线信道增益，σ^2 代表高斯白噪声功率。本研究假定在一个时间间隔内传输速率是保持不变的。对于第 i 个视频块，它开始传输前的等待时延

$$L_{n,i}^{wait}=\max\left(\frac{Q_n^{e,0}(t)\cdot C_0}{f_c^e},\frac{B_n(t)\cdot d\cdot Q_n^{trans}(t)}{D_n(t)\cdot v(t)}\right) \tag{6}$$

式(6)中 $\max(\cdot)$ 项代表第 i 个视频块的转码及传输必须在其转码队列和传输队列的剩余任务均完成后才能开始。因此，第 i 个视频块到达 MECS 的时间可表示为：

$$L_{n,i}^{arri}=L_{n,i}^{wait}+\max\left(\frac{B_n(t)\cdot d}{D_n(t)\cdot v(t)},\frac{C_0}{f_l^e}\right) \tag{7}$$

上式中 $\max(\cdot)$ 项代表了第 i 个视频块从开始转码到传输完成所消耗时延的近似表示，这样做的原因是 MEC 系统中，转码和传输采用并行的方式进行，转码时产生的固定大小的数据包不断向 MEC 服务器传输。同时，当网络条件较好时，转码的同时 ED 以数据包的形式不断传输视频块数据，转码消耗的时延和视频块数据传输完成时延近似相等；当网络条件较差时，数据传输的时延远大于转码操作的时延，$\max(\cdot)$ 项近似等于其传输时间，即

$$\max\left(\frac{B_n(t)\cdot d}{D_n(t)\cdot v(t)},\frac{C_0}{f_l^e}\right)\approx\frac{B_n(t)\cdot d}{D_n(t)\cdot v(t)} \tag{8}$$

3) 推理：视频块到达 MECS 后，视频块 RGB 帧数据进入 MEC 服务器的推理队列，这里用 $Q^{inf}(t)$ 表示。那么，ED_n 的第 i 个视频块输入 DL 模型前，所需的排队时间为 $Q^{inf}(t)\cdot C_1/f^s$，这里 f^s 表示 MECS 的 CPU 频率。则第 i 个视频块完成推理的预期时延可表示为

$$L_{n,i}^s=\frac{Q^{inf}(t)\cdot C_1}{f^s}+\frac{i}{f^s}C_1 \tag{9}$$

对于 t 时刻，ED_n 卸载 $M_n^{off}(t)$ 个视频块任务到 MECS，MECS 处理完成的预计时延 $L_n^s(t)$，即为最后一个视频块在 MECS 推理完成的时延：

$$L_n^s(t)=L_{n,M_n^{off}(t)}^s \tag{10}$$

综上，t 时刻到达 ED_n 的视频任务 $(B_n(t),D_n(t))$ 全部完成的预期时延为

$$L_n(t)=\max\left(L_n^e(t),L_n^s(t)\right) \tag{11}$$

因此对于 t 时刻整个 MEC 系统完成全部任务的预期时延，即全部 ED 的视频任务最后一个视频块完成的时间如下：

$$L(t)=\max\left(L_1(t),L_2(t),\cdots,L_N(t)\right) \tag{12}$$

2.2.3. 优化问题

在本文中，我们将 N 个 ED 和一个 MECS 作为一个完整的 MEC 系统，将这 N 个 ED 的任务卸载问

题形式化为一个优化问题。本文的优化目标是通过求解最优卸载决策 $\boldsymbol{\lambda}(t)=\left[\lambda_1(t),\lambda_2(t),\cdots,\lambda_N(t)\right]$ 使整个 MEC 系统的时延成本最小化，该问题表示如下：

$$\begin{aligned}&\arg\max_{\boldsymbol{\lambda}(t)} L(t)\\&\text{s.t. } \mathrm{C1}: 0\le\lambda_n(t)\le 1,\forall n\in\mathcal{N},\forall t\in\mathcal{T}\\&\qquad \mathrm{C2}: Q^{\inf}(t)+\sum_{n=1}^{N} M_n^{off}(t)\le Q_{\max},\forall t\in\mathcal{T}\end{aligned}\tag{13}$$

约束条件 C1 表示每个 ED 在时刻 t 的卸载率必须在 0 和 1 之间。C2 表示 MEC 服务器端的队列长度不能超过其阈值 $Q_{\max}$，即 MECS 的缓存容量是存在上限的，该阈值需根据 MECS 硬件配置情况设定，本文 $Q_{\max}$ 取值将在实验部分给出。本文讨论的用于视频任务处理的 MEC 系统中，视频任务的处理时延作为重要的性能指标，因此将时延作为问题模型的优化目标。需要注意的是，该优化问题为 NP 难问题并且 MEC 系统的网络时变性也给问题的求解带来了挑战，因此本文考虑采用深度强化学习算法来解决该问题。

3. 基于 DDPG 的任务卸载方案

3.1. 马尔科夫决策过程

为求解上述优化问题，本文采用深度确定性策略梯度(Deep Deterministic Policy Gradient, DDPG)算法。DDPG 算法是一种基于连续动作空间的无模型深度强化学习(Deep Reinforcement Learning, DRL)算法，与其他强化学习算法一样，其智能体(Agent)与所在环境不断进行交互，每次交互根据当前状态信息(State)采取动作(Action)，然后从环境中获取奖励(Reward)。智能体通过不断地学习、更新模型权重，寻找最优的 Action 以获取更大的 Reward。强化学习的求解首先需要建立马尔科夫决策过程(Markov Decision Processing, MDP)，本文所研究的问题中状态空间、动作空间和奖励函数表示如下：

1) 状态：我们将 $S(t)$定义为 t 时刻 MEC 系统环境的状态向量，其由当前时刻的上行带宽 $W(t)$、全部 ED 的视频任务信息、全部 ED 的队列信息及 MEC 服务器的队列信息组成。状态向量 $S(t)$的具体表示如下：

$$\boldsymbol{S}(t)=\left[W(t),\boldsymbol{B}(t),\boldsymbol{D}(t),\boldsymbol{Q}^{e,0}(t),\boldsymbol{Q}^{e,1}(t),Q^{inf}(t)\right]\tag{14}$$

其中，表示任务信息的向量 $\boldsymbol{B}(t)$ 和 $\boldsymbol{D}(t)$ 为

$$\boldsymbol{B}(t)=\left[B_1(t),B_2(t),\cdots,B_N(t)\right]$$

$$\boldsymbol{D}(t)=\left[D_1(t),D_2(t),\cdots,D_N(t)\right]$$

$\boldsymbol{Q}^{e,0}(t)$ 和 $\boldsymbol{Q}^{e,1}(t)$ 表示各 ED 在 t 时刻的队列状态，具体为

$$\boldsymbol{Q}^{e,0}(t)=\left[Q_1^{e,0}(t),Q_2^{e,0}(t),\cdots,Q_N^{e,0}(t)\right]$$

$$\boldsymbol{Q}^{e,1}(t)=\left[Q_1^{e,1}(t),Q_2^{e,1}(t),\cdots,Q_N^{e,1}(t)\right]$$

2) 动作：本文的动作空间由所有 ED 的卸载决策组成，即卸载率 $\lambda_n(t),\forall n\in\mathcal{N}$。Agent 在 t 时刻执行的动作用动作向量 $\boldsymbol{A}(t)$ 表示如下：

$$\boldsymbol{A}(t)=\boldsymbol{\lambda}(t)=\left[\lambda_1(t),\lambda_2(t),\cdots,\lambda_N(t)\right]\tag{15}$$

3) 奖励函数：在本文研究的 MEC 系统中，其优化目标是最小化系统时延成本。同时强化学习的目的是最大化奖励值，因此我们将奖励函数 $R(t)$ 定义为系统时延的负相关函数，为

$$R(t)=-L(t)\tag{16}$$

3.2. DDPG 算法

DDPG 是一种基于行为–批评(Actor-Critic, AC)框架的无模型 DRL 算法，它能够在高维状态空间及高维连续动作空间求解最优策略[7]。考虑到 MEC 环境中状态信息的高维度和连续性的特点，状态向量中每一时刻的$W(t)$的时变性以及到达任务的数据量$B(t)$与时长$D(t)$的随机性，本文决定利用 DDPG 算法，通过优化最优卸载决策来求解系统时延最小化问题。

DDPG 算法的网络模型采用 Actor 和 Critic 两个模块，每个模块包含一个当前网络和一个目标网络。Actor 当前网络根据 t 时刻的状态向量$S(t)$，输出并执行基于状态$S(t)$的动作$A(t)$，与环境交互后获得奖励$R(t)$且状态变为$S(t+1)$。将状态转移信息四元组$(S(t),A(t),R(t),S(t+1))$放入经验回放池中，每次从经验回访池中随机采样一定数量的样本通过梯度下降方式更新 Critic 当前网络和 Actor 当前网络。Actor 和 Critic 各自的目标网络用于输出当前网络的目标值，其参数的更新采用延迟更新加软更新的方式，即 Actor、Critic 当前网络的参数每更新固定步长再更新一次目标网络。针对本文中 MEC 视频任务卸载系统的 DDPG 算法伪代码如下：

算法 Video Task Offloading Algorithm based on DDPG

输入迭代次数、时间步长 T，上行带宽$W(t)$，所有 ED 视频任务向量$[\boldsymbol{B}(t),\boldsymbol{D}(t)]$，所有 ED 队列状态$[\boldsymbol{Q}^{e,0}(t),\boldsymbol{Q}^{e,1}(t)]$，MECS 队列状态$Q^{inf}(t)$

输出 ED 的卸载决策动作向量

随机初始化 Actor 网络权重θ和 Critic 网络权重 w、经验回放池大小、软更新系数τ、学习率α、衰减因子γ、探索噪声ϵ

1) For episode=1, 3000：
2) 获取初始状态向量$\boldsymbol{S}(1)$
3) For t=1, T:
4) Actor 网络根据状态$\boldsymbol{S}(t)$输出动作$\boldsymbol{A}(t)=\pi_\theta(\boldsymbol{S}(t))+\epsilon$
5) 执行动作$\boldsymbol{A}(t)$，获取奖励$R(t)$和下一时刻状态$\boldsymbol{S}(t+1)$
6) 将四元组$(\boldsymbol{S}(t),\boldsymbol{A}(t),R(t),\boldsymbol{S}(t+1))$存入经验回放池
7) 从经验回放池随机采样 m 个样本$(\boldsymbol{S}_j(t),\boldsymbol{A}_j(t),R_j(t),\boldsymbol{S}_j(t+1)),j=1,2,\cdots,m$，计算目标 Q 值y_j：

$$y_j=\begin{cases}R_j(t), & \boldsymbol{S}_j(t+1)\text{是终止状态}\\ R_j(t)+\gamma Q'(\boldsymbol{S}(t+1),\pi_{\theta'}(\boldsymbol{S}(t+1)),w'), & \text{否则}\end{cases}$$

8) 通过均方差损失函数$\frac{1}{m}\sum_{j=1}^{m}\left(y_j-Q(\boldsymbol{S}_j(t),\boldsymbol{A}_j(t),w)\right)^2$，更新 Critic 网权重 w
9) 通过损失函数$J(\theta)=-\frac{1}{m}\sum_{j=1}^{m}Q(\boldsymbol{S}_j(t),\boldsymbol{A}_j(t),\theta)$更新 Actor 网络权重$\theta$
10) If t%延迟更新步长=1：
11) 更新 Actor 目标网络，$\theta'\leftarrow\tau\theta+(1-\tau)\theta'$
12) 更新 Critic 目标网络，$w'\leftarrow\tau w+(1-\tau)w'$
13) If $\boldsymbol{S}(t+1)$为终止状态：
14) Endfor

4. 仿真结果

4.1. 参数设置

本文实验均在 Ubuntu 18.04 LTS 系统下基于 Python 3.7 编程实现，深度强化学习算法基于 Py Torch 1.7.0 框架编程实现。实验中将 NVIDIA Jetson NanoB01 作为 MEC 系统中的边缘设备，同时 MEC 服务器搭载 IntelCorei7-7700 的 CPU 及 NVIDIAGe Force GTX 2080 (8 GB)的 GPU。MEC 系统中涉及的其他参数如表 1 所示。

Table 1. Experimental parameters in MEC system
表 1. MEC 系统实验参数

参数	值
系统时间步长 T	50
单位视频块时长 d/Sec	2
任务数据量 $B_n(t)$ /Mb， $\forall n \in N, \forall t \in T$	2.4~14.4
任务视频时长 $D_n(t)$ /Sec， $\forall n \in N, \forall t \in T$	2~12
单位视频块转码所需资源 C_0 /Cycle	5.5×10^{10}
单位视频块推理所需资源 C_1 /Cycle	1.3×10^{10}
边缘设备 CPU 计算能力 f_c^e /GHz	1.09
边缘设备 GPU 计算能力 f_g^e /MHz	900
MEC 服务器 GPU 计算能力 f^{inf} /MHz	14,000
MEC 服务器队列长度阈值 Q_{max}	80

仿真实验中网络带宽数据来自真实世界测得的 500 组网络动态序列。本文中 DDPG 算法的训练迭代轮数设定为 3000，其他超参数设置为：Actor 和 Critic 网络的学习率 $\alpha = 10^{-4}$，更新参数采用 Adam 优化器；经验回放池的大小为 10,000，每次参数更新选取的批样本大小为 64，衰减因子 $\gamma = 0.99$，目标网络软更新的程度为 $\tau = 0.005$，探索噪声 $\epsilon = 0.1$。本次实验中 Actor 网络和 Critic 网络均由两层含有 128 个神经元的隐藏层及一层输出维度为 N 的输出层组成。

4.2. 实验结果分析

为了验证 DRL 算法的收敛性，首先我们从 500 组网络序列中随机抽样 450 组序列用于算法训练，剩余 50 组序列用于算法评估。如图 1 所示，我们测得了不同学习率 α 下的 DDPG 算法的收敛过程，ED 数量设定为 5，其他参数参照表 1。从图中可以看出，在算法训练初期，智能体仍在探索学习阶段，奖励值普遍较低且振荡。在训练中后期，算法的奖励值趋于稳定，且不难看出，对于学习率 $\alpha = 10^{-3}$ 的奖励曲线总是优于其他两个学习率奖励曲线。因此本文后续仿真实验采用基于学习率为 10^{-3} 的 DDPG 算法。

同时，DDPG 算法中的其他超参数也通过实验进行了收敛性比较，如图 2 所示，绘制了 3000 次训练迭代后不同批样本大小(Batch size)下的算法收敛情况。当批样本大小较小时，如 4 和 8，算法无法从历史经验中有效学习，导致奖励值一直无法提高及收敛。对于批样本大小为 16 时，随着迭代次数的增加，奖励值虽有增大趋势，但仍然存在较明显的波动。当批样本大小增大到 32 和 64 时，DRL 算法可以从中学

习到有效信息，最后达到收敛。因此本文后续仿真实验中批样本大小设定为 64。

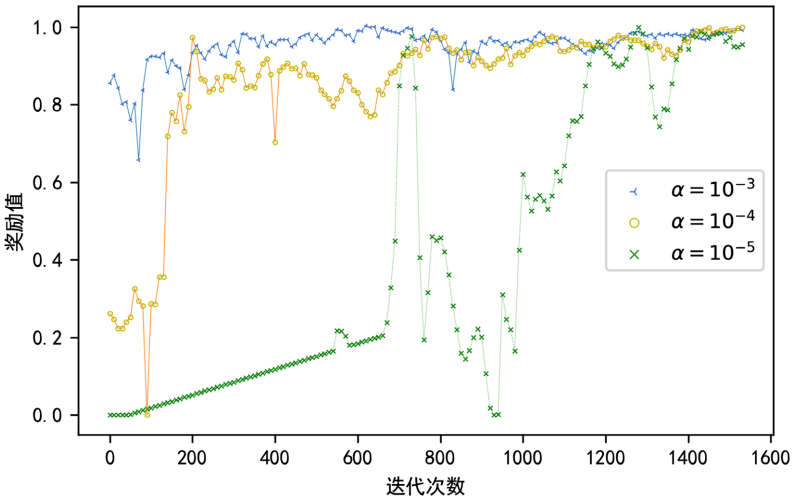

Figure 1. The influence of different learning rates on convergence of DDPG
图 1. 不同学习率对 DDPG 算法收敛性的影响

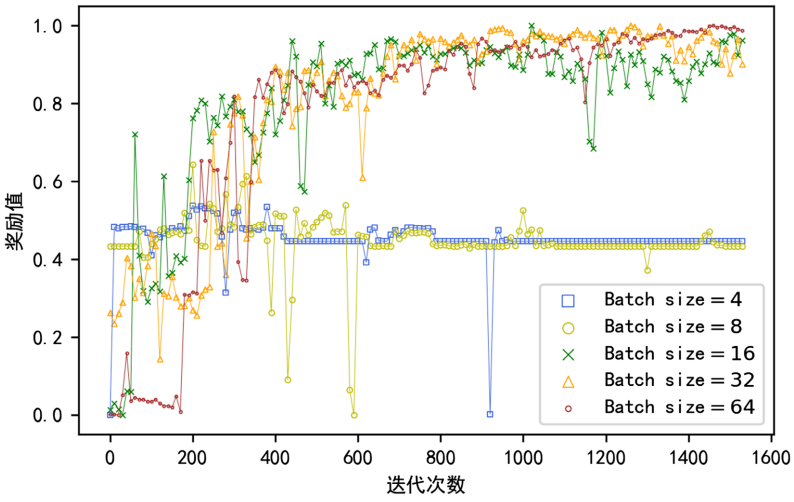

Figure 2. The influence of different batch sizes on convergence of DDPG
图 2. 训练不同批样本大小对 DDPG 算法收敛性的影响

为了验证 DDPG 算法在 MEC 视频任务卸载系统下的性能表现，我们将其与其他基准卸载策略进行了性能对比。所选的基准策略分别为：

1) 全本地处理(AllLocal)：在每一时刻 t，各 ED 到达的新任务均在当前 ED 设备本地处理，即 $\lambda_n(t)=0,\forall n\in\mathcal{N}$。

2) 全卸载处理(AllOffload)：在每一时刻 t，所有新任务执行卸载策略，即 $\lambda_n(t)=1,\forall n\in\mathcal{N}$。

3) 随机卸载(RandomPolicy)：对 t 时刻到达的新任务，各 ED 的卸载率为一个 0 到 1 的随机数，因此 $0\le\lambda_n(t)\le 1,\forall n\in\mathcal{N}$。

图 3 为任务到达时间间隔 Δ 下各卸载策略的系统时延对比(设备数量 $N=5$)。当 Δ 较小时，意味着视频任务到达的更加频繁，使得设备的处理压力更大。如图 3 所示，较小的 Δ 往往带来更高的系统时延成本，反之设备的处理压力不大的情况下，系统时延成本较低。更重要的是，从图 3 中不难看出，本文采

用的基于 DRL 的 DDPG 算法总能优于其他卸载策略，从而实现较低的系统时延成本，同时也说明该算法可以适应不同的任务到达间隔。

图 4 为不同边缘设备数量下的 MEC 系统时延成本对比(任务到达间隔 $\Delta = 20$)。ED 数量的增加直接导致任务数量的增加，当多个 ED 的视频任务都决定卸载到 MECS 时，使得 MECS 的处理压力加剧。如图 4 所示，整体上可以看出随着 ED 数量的增加，系统时延成本呈现增大趋势。全本地策略的时延成本相对全卸载策略变化较小，是因为所有 ED 选择在本地执行任务，ED 数量增加并不会加剧 MECS 的处理压力。ED 的增加使得随机卸载策略相较于前两者策略的优势愈发明显，由于 ED 随机性地卸载一定数量的任务到 MECS，从而减轻了本地设备的压力，同时 MECS 也不会总是超载。最后，从图 4 可以看出，本文采用的 DDPG 算法通过在环境中不断的学习最优策略函数，联合优化所有 ED 的卸载决策，使得其系统时延能够优于其他卸载策略。

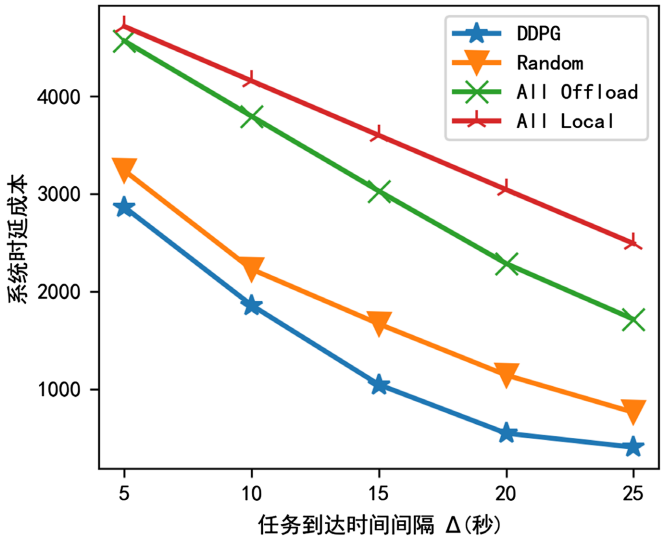

Figure 3. The system latency cost at different task arrival time slots
图 3. 不同任务到达时间间隔下的系统时延成本

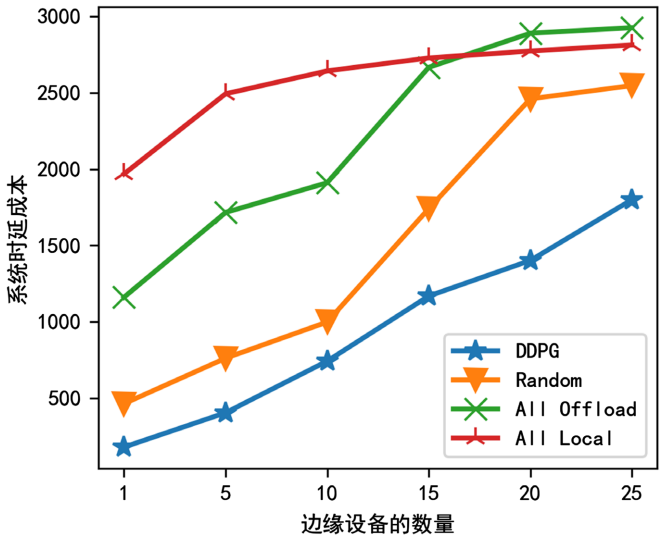

Figure 4. The influence of number of EDs on the system latency cost
图 4. 边缘设备数量对系统时延成本的影响

5. 结论

本文针对 MEC 场景中视频数据任务的卸载问题建立数学模型，建立最小化时延成本的优化目标，并将原问题转化为 MDP 以通过深度强化学习算法求解，通过不断优化边缘设备的卸载决策，最终仿真实验证明了该方法较其他基准卸载策略能够有效地减少视频任务处理时延，降低 MEC 系统成本。但本文侧重于时延敏感型的任务卸载问题，未来的工作中考虑将系统的能耗加入到优化目标中，联合优化 MEC 系统的时延与能耗。除此之外，多 ED 场景下的博弈关系值得研究，即采用多智能体的方法求解多 ED 任务卸载问题。

参考文献

[1] Vhora, F. and Gandhi, J. (2020) A Comprehensive Survey on Mobile Edge Computing: Challenges, Tools, Applications. 2020 4*th International Conference on Computing Methodologies and Communication* (*ICCMC*), Erode, 11-13 March 2020, 49-55. https://doi.org/10.1109/ICCMC48092.2020.ICCMC-0009

[2] Lin, H., Zeadally, S., Chen, Z., *et al*. (2020) A Survey on Computation Offloading Modeling for Edge Computing. *Journal of Network and Computer Applications*, **169**, 102781. https://doi.org/10.1016/j.jnca.2020.102781

[3] Zhu, Z., Han, G., Jia, G., *et al*. (2020) Modified Densenet for Automatic Fabric Defect Detection with Edge Computing for Minimizing Latency. *IEEE Internet of Things Journal*, **7**, 9623-9636. https://doi.org/10.1109/JIOT.2020.2983050

[4] Tang, M. and Wong, V.W.S. (2020) Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems. *IEEE Transactions on Mobile Computing*, 1. https://doi.org/10.1109/TMC.2020.3036871

[5] Ale, L., Zhang, N., Fang, X., *et al.* (2021) Delay-Aware and Energy-Efficient Computation Offloading in Mobile Edge Computing Using Deep Reinforcement Learning. *IEEE Transactions on Cognitive Communications and Networking*, **7**, 881-892. https://doi.org/10.1109/TCCN.2021.3066619

[6] Ranft, B. and Stiller, C. (2016) The Role of Machine Vision for Intelligent Vehicles. *IEEE Transactions on Intelligent Vehicles*, **1**, 8-19. https://doi.org/10.1109/TIV.2016.2551553

[7] Lillicrap, T.P., Hunt, J.J., Pritzel, A., *et al*. (2015) Continuous Control with Deep Reinforcement Learning. arXiv preprint arXiv:1509.02971