计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(12):95−103 [doi: 10.15888/j.cnki.csa.009336]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于改进 TD3 的 MEC 多任务计算卸载①

于　波[1], 毛鑫浩[1,2]

[1](中国科学院 沈阳计算技术研究所, 沈阳 110168)
[2](中国科学院大学, 北京 100049)
通信作者: 毛鑫浩, E-mail: mxh1116@foxmail.com

摘　要: 在多用户多任务场景下, 使用传统的决策算法去对短时间内接踵而来的任务进行计算卸载决策, 已经不能满足用户对决策效率和资源利用率的要求. 因此有研究提出使用深度强化学习算法来进行卸载决策以满足各种场景下的需求, 但是这些算法大多只考虑卸载优先的策略, 这种策略使用户设备 (UE) 被大量闲置. 我们提高了移动边缘计算 (MEC) 服务器和用户设备 (UE) 的资源利用率, 降低计算卸载的错误率, 提出了一种本地优先和改进 TD3 (twin delayed deep deterministic policy gradient) 算法相结合的决策卸载模型, 并设计了仿真实验, 通过实验证明该模型确实可以提高 MEC 服务器和 UE 的资源利用率并降低错误率.

关键词: 移动边缘计算; 计算卸载; 双延迟深度确定性策略梯度 (TD3); 资源分配

引用格式: 于波,毛鑫浩.基于改进 TD3 的 MEC 多任务计算卸载.计算机系统应用,2023,32(12):95–103. http://www.c-s-a.org.cn/1003-3254/9336.html

Multi-task Computation Offloading for MEC Based on Improved TD3

YU Bo[1], MAO Xin-Hao[1,2]

[1](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In multi-user and multi-task scenarios, using traditional decision algorithms to make computation offloading decisions for upcoming tasks in a short period can no longer meet users' requirements for decision-making efficiency and resource utilization. Therefore, some studies have proposed deep reinforcement learning algorithms for offloading decisions to cater to various scenarios. However, most of these algorithms only consider the offloading first strategy, which leaves user equipment (UE) idle. This study improves the resource utilization of mobile edge computing (MEC) servers and UE and reduces the error rate of computation offloading. It proposes a decision offloading model combining local first and improved twin delayed deep deterministic policy gradient (TD3) algorithm and designs a simulation experiment. The experimental results show that the model can indeed improve the resource utilization of MEC servers and UE and reduce the error rate.

Key words: mobile edge computing; computation offloading; twin delayed deep deterministic policy gradient (TD3); resource allocation

随着 5G 无线网络的快速发展, 越来越多要求低时延的计算密集型应用, 如智慧城市、无人驾驶汽车, VR, 云游戏等领域开始加速发展, 网络数据量急剧增加[1]. 这些领域的发展使用户对计算服务和质量有了新的要求. 但是对于移动设备而言, 其本身的计算能力是有限的, 如果提升移动设备的计算能力和电源储量[2,3],

① 收稿时间: 2023-06-12; 修改时间: 2023-07-12; 采用时间: 2023-07-21; csa 在线出版时间: 2023-10-25
CNKI 网络首发时间: 2023-10-26

必然会使移动设备的体积和价格大幅增长, 这不是用户希望的. 因此为了降低用户使用移动设备的成本, 同时让用户能够实时享受高性能高质量的计算服务, 移动边缘计算 (MEC) 被提了出来[4]. MEC 不同于云计算, 云计算是将用户的计算任务放到云端服务器中去运行, 但是云端服务器一般都会距离用户所在位置很远, 这会造成一定的通信延迟, 而且当大量用户同时卸载时, 还会造成远程链路和核心网络的拥塞, 使用户体验变差. 移动边缘计算是将服务器放置在距离基站比较近的位置, 使得用户任务可以就近在网络边缘处被执行, 这会在很大程度上降低通信延迟和能耗并节省带宽. 移动边缘计算的这一特性非常契合 5G 高容量、低延时、低能耗的通信要求, 因此已被视作是 5G 发展的关键使能技术之一.

MEC 是 C-RAN (云无线接入网) 的一种新范式, 它可以通过部署高性能服务器来提高网络边缘的计算能力. MEC 服务器分布在靠近移动用户的地方, 移动用户可以通过无线信道将计算任务卸载到 MEC 服务器. 通过计算卸载, 移动用户可以显著减少应用的经历时延, 提高服务质量. 因此, 计算卸载和计算资源分配作为 MEC 系统的一个关键问题, 引起了人们极大的兴趣. 充分利用 MEC 服务器的计算资源, 合理决策用户任务是否应该卸载到边缘服务器, 以及卸载到哪一个边缘服务器, 提高边缘服务器的资源利用率和服务质量, 将为智慧城市、无人驾驶汽车等行业提供更有稳定, 更高效的基础计算服务, 有助于这些应用领域的快速发展.

MEC 计算卸载的实验很难在现实网络中进行, 那需要花费巨大的成本, 所以 MEC 计算卸载的研究需要使用仿真环境来进行模拟, 但在 MEC 计算卸载目前的研究中, 大多数仿真实验都只考虑用户设备 (UE) 数量和边缘服务器数量, 他们并不考虑一个 UE 可能有多个计算任务, 而且大多数实验计算任务都是固定的, 但在实际环境中, UE 数量是动态变化的, 每个 UE 的计算任务也是动态变化的, 所以只用静态数据测试卸载决策算法的性能是远远不够的, 因此如何设计一个更加精确的仿真实验环境也是需要考虑的问题. 另一方面随着人工智能领域的快速发展, 深度强化学习已经在自然语言处理、计算机视觉、机器人和游戏等领域进行了广泛应用并取得了较好的表现. 因此, 近几年国内外将深度强化学习应用于 MEC 计算卸载的研究也越来越多. 但是这些研究大多都只考虑降低 UE 用户的能源消耗和计算时延[5,6], 这样就会导致 UE 用户的计算资源有可能被大量浪费, 而随着 MEC 边缘服务器计算任务的饱和, 其他急需使用 MEC 边缘服务器的用户将无法正常进行计算卸载. 本研究主要贡献如下.

1) 设计并实现了一个考虑 UE 用户数量可变, UE 用户距离边缘服务器距离不定, UE 用户任务数量及大小随机可变以及信号传输过程噪声问题的仿真实验环境.

2) 在 TD3 的基础上提出了一种针对 MEC 计算卸载的改良版 TD3 模型, 用于解决离散动作空间的 MEC 计算卸载问题, 提高 MEC 计算卸载的效率和正确率, 以及边缘设备资源利用率.

3) 在自主设计的仿真实验环境中测试对比了基线 DDPG、TD3 和结合本地优先策略的 TD3 的决策效果并进行了分析总结.

1 相关工作

近年来, 国内外已经有许多关于 MEC 计算卸载的研究. 文献 [7]提出了一种增强型多种群多目标鲸鱼优化算法, 并为了避免种群多样性的损失, 提出了一种新的领导者选择算法来引导种群搜索, 该算法可以很好地提高系统的响应速度和能源效率; 文献 [8]对遗传算法, 差分进化算法, 粒子群优化算法和混合蛙跳算法 4 种算法进行了研究, 这些算法都具有全局搜索能力、鲁棒性强等优点, 可以有效地解决计算卸载问题; 文献 [9]将计算卸载问题建模为了一个混合整数非线性规划问题, 并采用基于 RLT 的分支定界方法进行高效求解, 该算法所选择的卸载方案在延迟时间和能量消耗方面都具有较好的性能. 文献 [10]在 Kernighan Lin 算法的基础上, 提出了一种基于谱图划分的图划分方法. 该方法通过将应用程序分解成多个子任务, 然后将这些子任务分配到不同的设备上进行计算, 提高系统的响应速度和能源效率. 文献 [7–10]所使用的方法都没有考虑网络环境发生变化时决策方案应该如何进行改变, 而使用深度强化学习模型则可以根据网络环境的改变实时训练决策模型来动态适应环境变化, 从而增强模型的适应能力. 文献 [11]提出了一种基于 Q-Learning 的机会式边缘计算中的计算卸载时间优化方法. 该方法通过不断学习, 自适应地调整计算卸载策略, 提高系统的响应速度和用户体验质量. Q-Learning 是一种基于强化学习的优化算法, 可以通过学习和试错来寻找最优策略. 但是 Q-Learning 算法更新速度慢, 且预见能

力不强. 文献 [12] 提出了一种基于深度强化学习的计算卸载和资源分配算法 AHP-DQN 算法. 该算法通过学习决策动作和网络环境, 自动调整资源分配策略, 提高系统的响应速度和能源效率. 文献 [13]提出了一种基于 Dueling DQN 的半在线的计算卸载模型, 通过强化学习来获取未知的环境信息. 该算法在不同的服务器场景下均能实现负载均衡. 文献 [14]提出使用 LSTM (长短期记忆) 算法预测下一时隙资源利用状态, 然后使用一种分布式多智能体深度强化学习算法来解决计算卸载问题, 该算法有效地解决了计算卸载问题. 文献 [15]提出了一种适用于连续动作空间的基于深度强化学习的自适应计算卸载算法, 并考虑了无线信道的多样性和相邻时隙之间的可用带宽, 该算法在随机环境中优于 Dueling DQN 和贪婪策略. 文献 [12–15]使用了基于深度 Q 网络 (DQN) 的强化学习方法来进行计算卸载, 但是 DQN 对训练数据要求很高, 并且容易出现过估计问题.

以上这些研究都是只考虑降低延迟或者能耗, 并且 UE 用户数也是固定的, 每个 UE 也只有一个计算任务, 而本研究主要考虑在蜂窝网络中, 当 UE 数量和 UE 卸载任务不断变化的条件下, 考虑最大化资源利用率, 最小化卸载决策错误率的情况下, 设计实现一个本地资源优先使用的情况下的结合改进 TD3 的卸载决策算法. 实验表明该算法资源利用率比使用基线 DDPG 算法和 TD3 算法进行卸载决策有更高的资源利用率和更低的错误率.

在本研究中我们考虑采用本地优先的策略再结合深度强化学习进行计算卸载决策. 采用这种方法可以优先使用 UE 用户的计算资源, 在 UE 用户计算资源匮乏或者能源不足时, 在进一步考虑将计算任务进行卸载.

2 系统架构和问题建模

2.1 系统架构

本研究将搭建一个 MEC 仿真环境, 该仿真环境中, 边缘服务器的性能各不相同, UE 设备的性能也各不相同, 仿真环境还添加了 UE 到各个边缘服务器的距离, 除此之外还模拟了蜂窝网络的噪声来尽可能增加仿真环境的准确性. 仿真环境的网络模型如图 1 所示. 该仿真环境共有 4 部分组成, 分别是 UE 用户端、蜂窝网络基站、MEC 服务器和总控制器. 其中 UE 用户用 $U=\{1,2,3,\cdots,n\}$ 表示[16], UE 用户每隔一定时间会以一定的概率随机产生一个任务, 单个任务表示为 $T=(u,t,s,p)$, 其中 u 表示 UE 用户, t 表示执行任务所需要花费的时钟周期, s 表示执行该任务所需要耗费的内存资源也就是任务大小, t 和 s 成正相关, p 表示该任务预期所要花费的时间. 用 $M=\{1,2,3,\cdots,m\}$ 表示 MEC 服务器[17], 每一个基站都会有一个 MEC 服务器与之相连. UE 用户会连接最近的基站, 并通过基站将要卸载的任务信息发送给控制器, 再由控制器来决定卸载到哪一个 MEC 服务器.

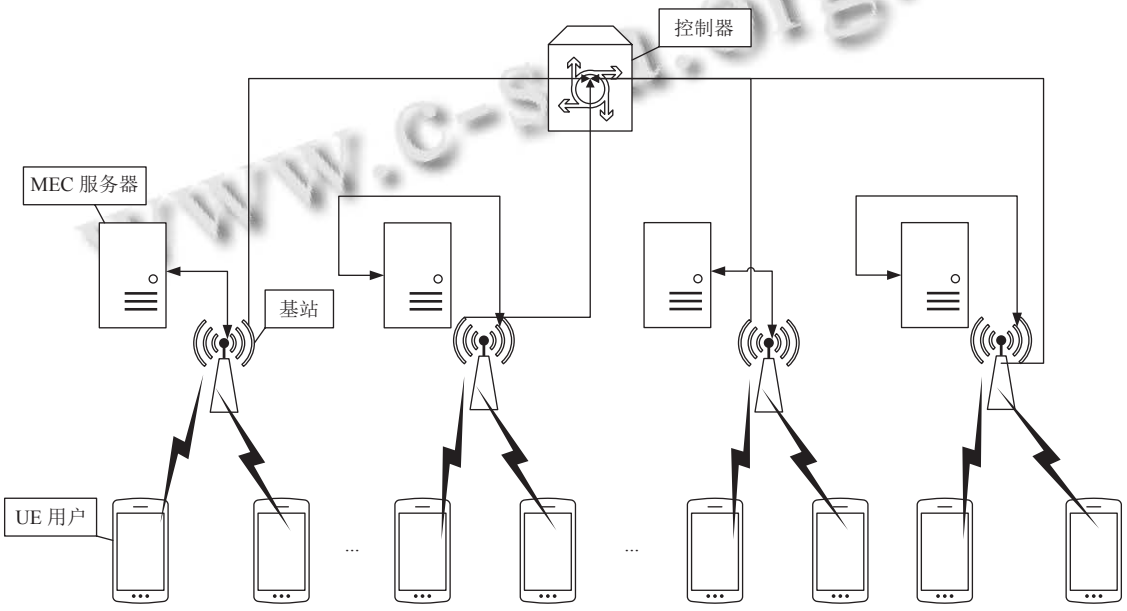

图 1 网络架构图

控制器会维护一个 MEC 服务器列表用 $ML=\{M_1,M_2,M_3,\cdots,M_n\}$ 表示, 该列表中包含每一个 MEC 服务器剩余的内存资源, 所拥有的最大计算频率以及正在执行的任务数量. 每一个 MEC 服务器也会维护一个与之连接的 UE 用户列表 $UL=\{U_1,U_2,U_3,\cdots,U_n\}$, 该列表包含 UE 用户的状态、距离该边缘服务器的距离以及 UE 用户的标识码, MEC 服务器还会根据 UE 列表来为每一个连接的 UE 用户维护一个任务列表 $UTL=\{T_1,T_2,T_3,\cdots,T_n\}$, 任务列表包含该 UE 用户卸载到 MEC 服务器的所有正在执行任务的状态. 除此之外, 该仿真系统在模拟完成 UE 用户产生的任务是, 采用按时间片轮转进行计算的方式去完成任务, 因此任务数目越多, 每个任务所能分到的时间片就越少, 计算所花费的时间也越多. 为了防止任务过多导致卸载的任务超时, MEC 服务器在接收到任务时会先计算完成当前任务所要等待的时间, 如果该时间超过预期则会提前拒绝卸载. 预期所要花费的时间是由 UE 用户来产生并随任务信息一同发送给 MEC 服务器的. 综上所述, 该仿真系统采用的是 3 层架构, 分别是负责卸载决策的控制层, 负责通信和计算的 MEC 服务器层以及产生计算任务的 UE 用户层. UE 产生计算任务后通过基站将任务信息发送给控制器, 控制器使用卸载决策模型得出卸载动作, 再将任务信息转发给要接受卸载的 MEC 服务器, 然后让 MEC 服务器和用户建立连接进行任务卸载.

2.2 传输模型

本实验模拟移动通信中的使用的正交频分多址 (OFDMA) 来进行 UE 和基站之间的通信, 但是我们考虑每一个 UE 用户只和距离自己最近的基站进行直接通信, 因此不考虑使用相同子信道移动用户之间的影响且与同一个基站连接的用户均分基站带宽, 因此用户带宽可以表示为 $W_{im}=\dfrac{W}{n}$ 其中 W 为基站总带宽, n 为连接该基站的 UE 数目. 我们假设每一个 UE 都和距离自己最近的基站之间有一个上行链路, 这个链路的信道增益定义为 h_{im}, h_{im} 是根据路径损失 (L_{dB}), 宏蜂窝天线增益 (A_{dB}) 以及功率损耗 (P_{dB}) 来计算出来的, 具体计算公式如下:

$$L_{dB}=128.1+37.6\times\lg(dist) \tag{1}$$

$$h_{im}=10^{\frac{A_{dB}-L_{dB}-P_{dB}}{10}} \tag{2}$$

其中, $dist$ 是 UE 和基站之间的距离, A_{dB} 采用固定值 15 dBi, P_{dB} 采用固定值 8 dB. 而 UE 用户的信号发射功率定义为 $P=\{p_n|0<p_n<P,n\in N\}$, 其中 P 为最大传输功率, 无线传输的噪声功率定义为 N_0, 噪声功率计算如下:

$$N_0=n_0+10\lg(B) \tag{3}$$

其中, $n_0=-174$ dBm/Hz 为噪声功率谱密度, B 为信道带宽. 最后我们根据式 (1)–式 (3) 结合香农定理和信干噪比 ($SINR$)[18] 来计算每一个移动用户的传输速率 C, 计算过程如下:

$$SINR=\frac{P\times h_{im}}{N_0} \tag{4}$$

$$C=B\times\ln(1+SINR) \tag{5}$$

2.3 计算模型

2.3.1 本地计算模型

定义每一个 UE 的 CPU 计算频率为 F_n^{local} GHz/s, 每一个 UE 用户的计算能力是随机的, 如果 UE 用户 n 的任务在本地执行, 则执行该任务所需要耗费的时延定义为 T_n^{local}:

$$T_n^{\text{local}}=c_i\times\frac{sum}{F_n^{\text{local}}} \tag{6}$$

其中, c_i 为该任务需要消耗的计算资源, sum 为本地当前需要计算的总任务数量. 本地计算所需要花费的能量为 E_n^{local}:

$$E_n^{\text{local}}=\mu(F_n^{\text{local}})^2c_i \tag{7}$$

其中, μ 为有效转化因子, 本研究中有效转化因子设为 $\mu=10^{-27}$. 根据时延公式和能量消耗公式, 本地计算成本定义为:

$$C_n^{\text{local}}=it\times T_n^{\text{local}}+ie\times E_n^{\text{local}} \tag{8}$$

其中, it 和 ie 为计算时延和能量所占的权重[19,20], 权重满足:

$$\{0\leqslant it\leqslant 1,\ 0\leqslant ie\leqslant 1,\ it+ie=1\} \tag{9}$$

2.3.2 MEC 服务器计算模型

MEC 服务器中的任务由 UE 用户卸载得到, 必须考虑任务卸载过程中的传输时延, 由式 (5) 我们可以算出数据的传输速率, 根据传输速率以及任务大小我们可以计算出传输时延 T_n^{up} 为:

$$T_n^{\text{up}}=\frac{S}{C} \tag{10}$$

其中, S 为要卸载的任务大小, C 为根据香农定理计算出来的传输时延. 除了传输时延外还需要考虑传输功率 E_n^{up}:

$$E_n^{\text{up}} = p_n T_n^{\text{up}} \tag{11}$$

MEC 服务器完成任务所要花费计算时间定义为 T_n^{edge}, 所要耗费的能量定义为 E_n^{edge}, 他们的计算方式与本地相同, 因此 MEC 服务器完成某个任务的总成本为:

$$C_n^{\text{edge}} = it\left(T_n^{\text{up}} + T_n^{\text{edge}}\right) + ie\left(E_n^{\text{up}} + E_n^{\text{edge}}\right) \tag{12}$$

2.4 问题建模

为降低所有 UE 用户的所有任务的总计算时延和能耗, 提高 MEC 服务器和 UE 用户移动设备的资源利用率, 以本地计算优先结合控制器集中控制的方式最大化资源利用率, 最小化任务平均计算成本. 系统总资源利用率定义为 RU:

$$RU = \frac{\sum_{i\in m} CR_i^{\text{edge}} + \sum_{i\in m} CR_i^{\text{local}}}{\sum_{i\in m} R_i^{\text{edge}} + \sum_{i\in m} R_i^{\text{local}}} \tag{13}$$

其中, CR_i^{edge} 为 MEC 服务器已经消耗的资源, R_i^{edge} 为 MEC 服务器总资源, CR_i^{local} 为 UE 设备已经消耗的资源, R_i^{local} 为 UE 设备总资源. 所有任务的平均计算成本定义为 AC:

$$AC = \frac{\sum_{i\in n} a_i C_i^{\text{local}} + \sum_{i\in n} (1-a_i) C_i^{\text{edge}}}{n} \tag{14}$$

其中, $\sum_{i\in n} a_i C_i^{\text{local}}$ 为在本地计算的所有任务的计算成本之和, $\sum_{i\in n} (1-a_i) C_i^{\text{edge}}$ 为在 MEC 服务器进行计算的所有任务的计算成本之和, n 为本地和 MEC 服务器中的所有任务数目. 优化目标可以表示为:

$$\begin{cases} \max(RU) \\ \min(AC) \end{cases} \tag{15}$$

为了防止 UE 设备以及该 MEC 服务器过载, 所以在任务卸载过程中应该满足一下约束:

$$\text{s.t.} \begin{cases} C_1 : a_{ij} \in \{0,1\} \\ C_2 : 0 < P_i < P_{i,\max} \\ C_3 : T_i < L_i \\ C_4 : F_i^{\text{local}} < F_{i,\max}^{\text{local}},\ F_j^{\text{edge}} < F_{j,\max}^{\text{edge}} \\ C_5 : \sum_{j=1}^{m} a_{ij} = 1, \forall i \in n \end{cases} \tag{16}$$

其中, C_1 表示卸载决定 0 为本地执行, 1 为卸载执行; C_2 表示 UE 的数据发送功率的约束; C_3 为计算任务的时延约束; C_4 为 UE 设备和 MEC 服务器的算力约束; C_5 为每一个任务只能选择一个 MEC 服务器进行任务卸载约束.

3 改进 TD3 计算卸载模型

TD3 算法又叫做双延迟深度确定性策略梯度, 他是在 DDPG 的基础上提出来的, TD3 的目的是解决 DDPG 已经学习好的 Q 函数显著高估 Q 值导致策略被破坏的问题. TD3 相对于 DDPG 主要引入了 3 个技巧来解决这个问题.

1) 采用截断的双 Q 学习, TD3 会同时学习两个 Q 函数, 两个 Q 函数都使用一个目标, 两个 Q 函数中较小的值会被作为 Q-target.

2) 采用延迟的策略更新, TD3 没有采用同步训练动作网络和评价网络的方式, 而是以较低的频率更新动作网络, 以较高的频率更新评价网络.

3) 采用目标策略平滑, TD3 在目标动作中加入噪声, 来使策略更难利用 Q 函数的误差.

本研究对 TD3 的改进主要涉及以下 3 个方面.

1) 由于 TD3 是用于连续动作空间的强化学习算法, 而本研究将计算卸载问题抽象为离散空间中的决策问题, 所以需要对 TD3 进行改进, 使其能够用于离散空间的决策问题, 因此本研究在 TD3 中加入了 *Softmax* 方法用来使 TD3 的连续动作转化为离散动作进行决策.

2) 本研究根据 MEC 计算卸载的目的, 定义了新的状态空间和奖励函数, 该状态空间主要考虑 MEC 服务器的剩余资源, 而奖励函数则分为奖励和惩罚两个部分, 这样可以使模型能够向着更高效的决策方向进行学习.

3) 将本地优先策略和 TD3 算法相结合, 以此来最大化总体资源利用率, 使更多的用户卸载可以得到满足.

本节将对卸载决策模型的设计和改进 TD3 与本地优先的决策算法进行详细介绍.

3.1 卸载决策模型

TD3 训练过程可以转化为 MDP 模型, 其中 MDP 模型中的状态主要设计 3 个: 1) 环境状态 s; 2) 动作 a; 3) 奖励 r. TD3 根据环境的状态 $s(t)$ 来采取动作 a, 动作 a 作用于环境后得到奖励 r 和下一个环境的状态 $s(t+1)$. 本研究中环境状态, 动作和奖励的定义如下.

1) 状态: 状态主要考虑当前 MEC 服务器的剩余资源, 单个 MEC 服务器的剩余资源计算如下:

$$SR_i^{\text{edge}} = \left\{SM_i^{\text{edge}}, SF_i^{\text{edge}}\right\} \tag{17}$$

其中, SM_i^{edge} 为 MEC 服务器剩余的内存, SF_i^{edge} 为 MEC 服务器每个任务所能分到的最大计算频率. 状态空间 $s(t)$ 由所有 MEC 服务器在 t 时刻的剩余资源组成:

$$s(t) = \left[SR_1^{\text{edge}}(t), \cdots, SR_n^{\text{edge}}(t)\right] \tag{18}$$

2) 动作: 动作主要是决策将任务卸载到哪一个 MEC 服务器中, 由于本研究采用本地优先的策略, 所以当任务请求卸载时表明本地资源或能量已经不能够支持计算任务, 因此只能选择卸载到某个 MEC 服务器中. 考虑卸载到所有 MEC 服务器的概率之和为一, 那么动作空间为:

$$a(t) = \left[a_1^{\text{edge}}(t), \cdots, a_n^{\text{edge}}(t)\right], \sum_{i=1}^{n} a_i^{\text{edge}} = 1 \tag{19}$$

3) 奖励: 考虑当智能体卸载的任务超时, 或者所选择的 MEC 服务器剩余资源不足时为决策失误, 这时应该为惩罚, 而当任务卸载成功时, 我们应最大化资源利用率, 最小化平均计算成本, 所以奖励函数定义为:

$$r(t) = \begin{cases} \mu\dfrac{RU+1}{AC+1}, & T_i \leqslant L_i \\ -\mu\dfrac{RU+1}{AC+1}, & T_i > L_i \end{cases} \tag{20}$$

其中, RU 为资源利用率, AC 为平均计算成本, μ 为奖励因子.

3.2 改进 TD3 算法

TD3 适用于连续动作空间的决策问题, 但对于 MEC 计算卸载而言, 选择一个 MEC 服务器进行卸载并不是连续动作空间问题, 而是离散动作空间, 为了解决这个问题需要改进 TD3 的动作选择机制, 采用 $Softmax(a)$ 激活函数来将 TD3 得出的动作归一化为一个概率分布向量, 然后按照概率进行随机采样来获得所要选择的 MEC 服务器.

$$Softmax(a) = \frac{\exp(a_i)}{\sum_{j=1}^{n}\exp\left(a_j\right)} \tag{21}$$

加入 $Softmax(a)$ 后的 TD3 算法架构如图 2 所示. TD3 算法流程大致如算法 1 所示.

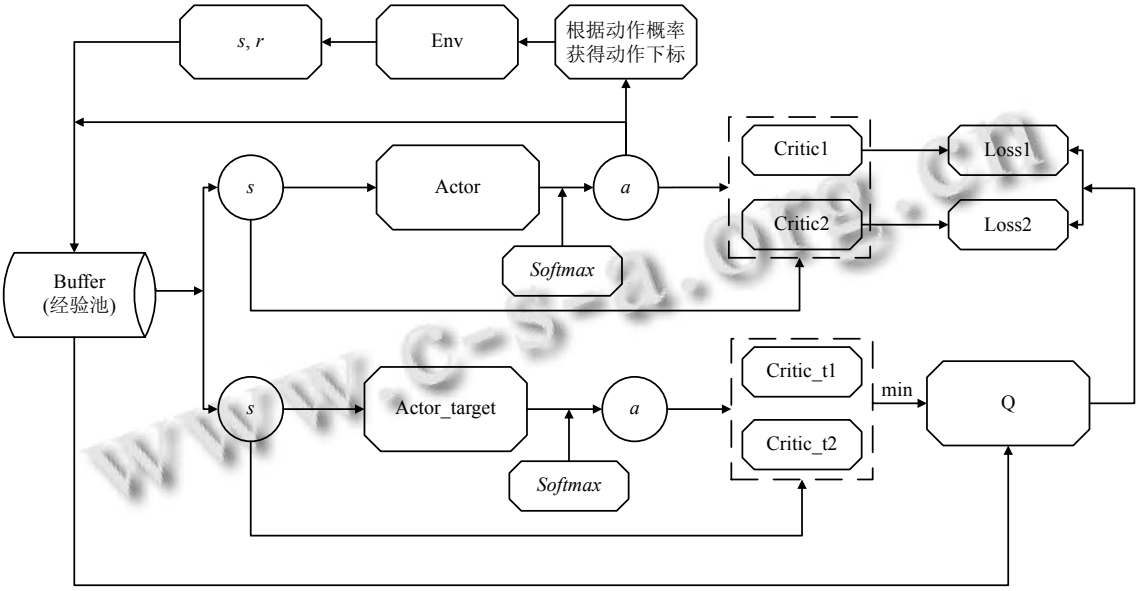

图 2 TD3 算法架构图

首先进行初始化, 初始化网络结构和经验池后, 开始填充经验池, 经验池填满之前通过使用 actor 网络获得随机的动作进行决策并将环境状态、动作、奖励和下一个状态存入经验池, 经验池填充结束后, 每次执行动作就从经验池中进行随机采样, 然后利用样本进行学习, 更新 actor 和 critic 网络, 在使用软更新策略更新目标网络.

算法 1. 改进 TD3

1) 初始化 actor 和 critic 网络
2) 初始化 actor_target 和 critic_target 网络

3) 初始化经验池 Buffer
4) for *episode* = 1, *M* do
5) 初始化环境 Env
6) for *t* = 1, *T* do
7) 从 actor 网络中获取带有噪声的动作 $a(t)$, 在使用 $Softmax(a(t))$ 获得概率分布
8) 根据动作概率进行采样将连续动作转化为离散动作 $\_a(t)$
9) 执行动作 $\_a(t)$, 得到奖励 $r(t)$ 和下一个状态 $s(t+1)$
10) 将 $(s(t), a(t), r(t), s(t+1))$ 存放进 Buffer
11) if Buffer 中样本数目大于 batch size then
12) 从 Buffer 中随机采样 batch size 个样本进行训练
13) 更新 critic 网络和 actor 网络
14) 更新 critic_target 网络和 actor_target 网络
15) end if
16) end for
17) end for

4 实验结果

4.1 实验设置

仿真实验使用的环境参数为 Python 3.8, PyTorch 2.0.0+cu118, CUDA 12.1 和 Windows 10. 显卡为 GTX1050Ti, 4 GB 显存和 24 GB 内存. 仿真环境设有 4 个 MEC 服务器和可变个数的 UE 设备. 改进的 TD3 算法中的 Actor 网络和 Critic 网络均为 4 层全连接神经网络, Actor 的两个隐藏层神经元个数分别为 512 和 256, Critic 的两个隐藏层神经元个数为 256 和 128. Actor 输出层使用 *Softmax* 进行激活, 其他激活函数均使用 ReLU 激活函数. 梯度下降优化算法为 Adam 优化器. Actor 和 Critic网络的学习率分别为 0.000 3 和 0.001. 目标网络采用软更新, 更新速率 tau=0.005. 经验回放池大小设置为 10 000. 折扣因子 γ=0.99. 仿真环境中 UE 设备和 MEC 设备等的仿真参数设置如表 1 所示.

表 1 仿真环境参数设置

参数	数值
信道噪声功率谱密度n_0	−174 dBm/Hz
信道带宽W	6 MHz
有效转换因子	10^{-27}
传输功率P_t	1W
计算功率P_c	1W
MEC计算能力F^{edge}	[2.5, 3.2] GHz
UE计算能力F^{local}	[1, 2] GHz
MEC内存大小inter_edge	[5000, 6000] kb
UE内存大小inter_local	[3000, 4000] kb
任务大小taskSize	[100, 1000] kb
任务需要的计算周期数taskTime	[10^8, 10^9] Hz

本实验对比了一下 3 种卸载策略, 分别是 3 种由非本地优先的卸载策略 DDPG 算法、TD3 算法和结合本地优先的 TD3 卸载算法.

4.2 实验结果对比分析

仿真实验环境中 UE 设备随着时间动态生成计算任务, 通过这些任务来寻来你卸载决策模型, 本实验主要对比了随着任务数量不断增多, 模型决策的错误率和平均资源利用率的变化情况.

1) 本研究工作了记录了 3 组不同卸载频率下的错误率变化情况, 每组实验都执行了 40 万次计算卸载, 且每进行 20 次卸载决策记录一次决策错误率. 第 1 组实验控制任务卸载频率为 20 个/s, 在该频率下 3 个模型的训练结果如图 3 所示. 根据图 3 可以看出, 该频率下由于卸载任务比较缓慢, 3 个模型决策错误率相差不大, 但是结合本地优先的 TD3 算法比 DDPG 算法决策错误率, 下降了 1.5%. 第 2 组实验我们将卸载频率上升至 100 个/s, 实验结果如图 4 所示, 此时 3 个决策模型的错误率都开始增大, 其中 DDPG 算法增大了 13.5%, TD3 增大了 11%, 而结合本地优先的 TD3 算法只增大了 6%, 在该频率下结合本地优先的 TD3 算法决策错误率比 DDPG 降低了 9%. 紧接着本实验又将卸载频率上升到 200 个/s, 实验结果如图 5 所示, 3 个决策模型错误率又有了提升, 但在该频率下结合本地优先的 TD3 算法的决策错误率要比 DDPG 算法降低 11%. 图 6 所示是对比了 3 个模型在 3 组实验中的最终决策错误率, 由图 6 可以看出, 随着任务卸载频率的上升, 3 个模型的决策错误率都增加了, 但是结合本地优先的 TD3 算法能够表现出更好的决策效果.

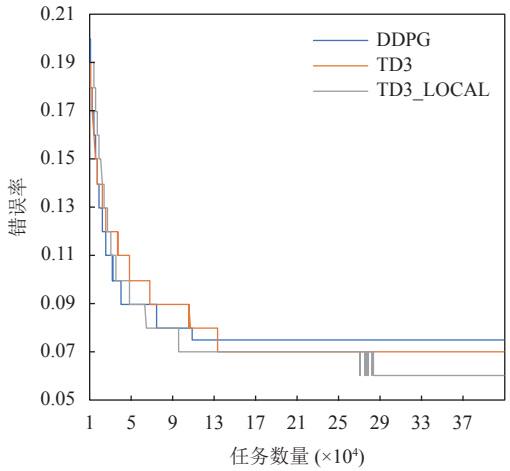

图 3 卸载频率 20 个/s 下的决策错误率

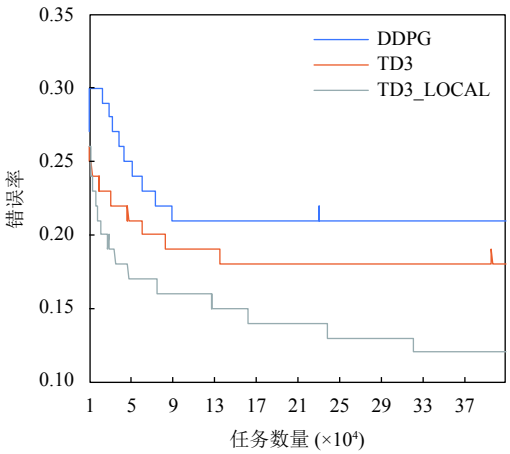

图 4 卸载频率 100 个/s 下的决策错误率

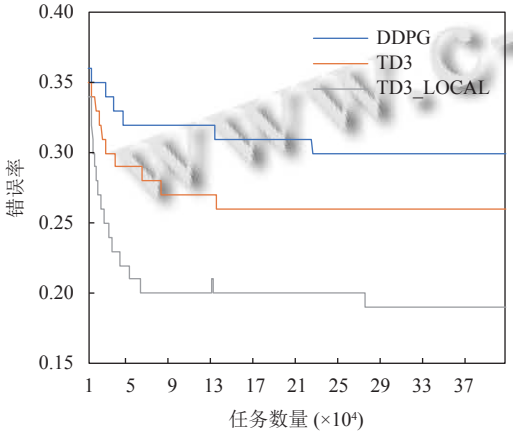

图 5 卸载频率 200 个/s 下的决策错误率

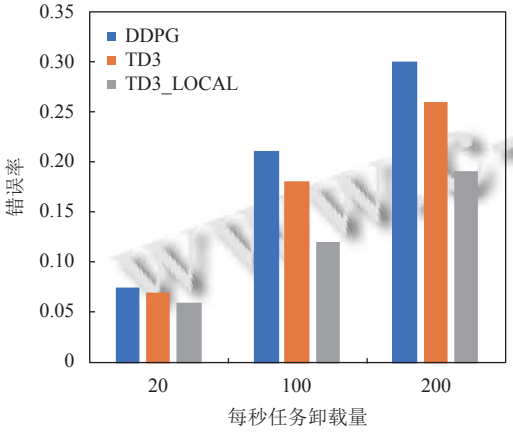

图 6 不同频率决策错误率

2) 图 7 展示在任务卸载频率为 100 个/s 的情况下随着任务数目增多 UE 设备和 MEC 服务器的平均资源利用率的变化情况, 由图信息可以看出随着任务数量的不断增加, 平均资源利用率趋于稳定, 但考虑到不论是 MEC 服务器还是 UE 设备都存在资源阈值, 因为资源利用率过高会导致设备内存资源不够, 反而会降低运行效率, 因此设备资源利用率无法达到百分之百. 但是根据图 7 中统计结果可以看出, 随着任务数量增加最终结合本地优先的 TD3 卸载策略有一个更高的资源利用. 根据实验得出结合本地优先的 TD3 卸载策略平均资源利用率可以达到 0.85 左右, 而只使用 TD3 和 DDPG 进行卸载决策平均资源利用率只能达到 0.78 和 0.73 左右. 因此使用结合本地优先的 TD3 卸载策略能够达到一个更高的资源利用率和更低的决策错误率.

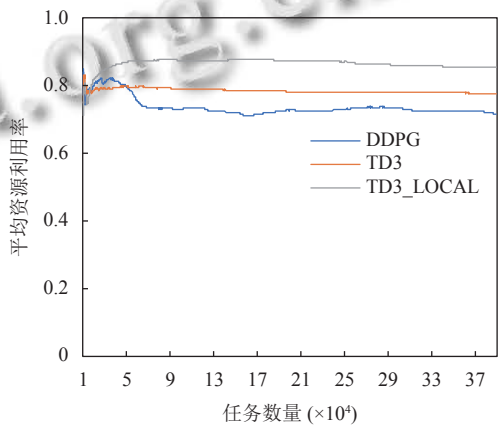

图 7 设备资源平均利用率

5 结论

本文主要研究了在多用户多 MEC 服务器且每个用户有多个任务的场景下计算任务卸载决策的问题. 实验考虑了蜂窝网络的延迟、噪声、能耗、UE 用户位置、MEC 服务器位置等问题来设计了一个仿真实验环境, 并使用该仿真环境以最大化资源利用率、最小化决策错误率为目标, 在 TD3 的基础上通过添加 *Softmax* 和本地优先策略来进行卸载决策仿真实验. 仿真实验表明, 结合了本地优先策略的 TD3 决策模型可以有效地降低决策错误率, 提高资源利用率. 但是本实验的不足之处是没有考虑用户任务的重要程度, 而是采用时间片轮转的方式执行任务, 这可能无法满足一些用户的对服务质量的要求, 在未来的实验中计划进一步考虑用户任务的执行顺序问题.

参考文献

1 Wang D, Song B, Liu YJ, *et al*. Secure and reliable computation offloading in blockchain-assisted cyber-physical IoT systems. Digital Communications and Networks, 2022,

8(5): 625–635. [doi: 10.1016/j.dcan.2022.05.025]
2 Alshahrani A, Elgendy IA, Muthanna A, *et al*. Efficient multi-player computation offloading for VR edge-cloud computing systems. Applied Sciences, 2020, 10(16): 5515. [doi: 10.3390/app10165515]
3 Elgendy IA, Zhang WZ, Tian YC, *et al*. Resource allocation and computation offloading with data security for mobile edge computing. Future Generation Computer Systems, 2019, 100: 531–541. [doi: 10.1016/j.future.2019.05.037]
4 Li CL, Zhang Y, Luo YL. DQN-enabled content caching and quantum ant colony-based computation offloading in MEC. Applied Soft Computing, 2023, 133: 109900. [doi: 10.1016/j.asoc.2022.109900]
5 Jo S, Kim U, Kim J, *et al*. Deep reinforcement learning-based joint optimization of computation offloading and resource allocation in F-RAN. IET Communications, 2023, 17(5): 549–564. [doi: 10.1049/cmu2.12562]
6 Du TY, Li CL, Luo YL. Latency-aware computation offloading and DQN-based resource allocation approaches in SDN-enabled MEC. Ad Hoc Networks, 2022, 135: 102950. [doi: 10.1016/j.adhoc.2022.102950]
7 Yang B, Pang Z, Wang SL, *et al*. A coupling optimization method of production scheduling and computation offloading for intelligent workshops with cloud-edge-terminal architecture. Journal of Manufacturing Systems, 2022, 65: 421–438. [doi: 10.1016/j.jmsy.2022.10.002]
8 Mehta S, Kaur P. Efficient computation offloading in mobile cloud computing with nature-inspired algorithms. International Journal of Computational Intelligence and Applications, 2019, 18(4): 1950023. [doi: 10.1142/S1469026819500238]
9 Luo J, Deng XH, Zhang HG, *et al*. QoE-driven computation offloading for edge computing. Journal of Systems Architecture, 2019, 97: 34–39. [doi: 10.1016/j.sysarc.2019.01.019]
10 Mathur RP, Sharma M. Graph-based application partitioning approach for computational offloading in mobile cloud computing. Recent Advances in Computer Science and Communications, 2021, 14(1): 92–99. [doi: 10.2174/2213275912666190716114033]
11 Yang GS, Hou L, Cheng H, *et al*. Computation offloading time optimisation via Q-Learning in opportunistic edge computing. IET Communications, 2020, 14(21): 3898–3906. [doi: 10.1049/iet-com.2020.0765]
12 Chen GQ, Cai Q, Fu XB, *et al*. Research on algorithms of computing offloading and resource allocation based on DQN. Journal of Physics: Conference Series, 2021, 1748(3): 032047. [doi: 10.1088/1742-6596/1748/3/032047]
13 Song SN, Fang ZY, Zhang ZY, *et al*. Semi-online computational offloading by dueling deep-Q network for user behavior prediction. IEEE Access, 2020, 8: 118192–118204. [doi: 10.1109/ACCESS.2020.3004861]
14 Xu SL, Guo CL. Computation offloading in a cognitive vehicular networks with vehicular cloud computing and remote cloud computing. Sensors, 2020, 20(23): 6820. [doi: 10.3390/s20236820]
15 Ke HC, Wang J, Deng LY, *et al*. Deep reinforcement learning-based adaptive computation offloading for MEC in heterogeneous vehicular networks. IEEE Transactions on Vehicular Technology, 2020, 69(7): 7916–7929. [doi: 10.1109/TVT.2020.2993849]
16 Chen J, Gao Q, Wu Q, *et al*. A computation offloading scheme based on FFA and GA for time and energy consumption. Proceedings of the 10th International Conference on Computer Engineering and Networks. Singapore: Springer, 2021. 1500–1506.
17 Zhang XJ, Wu WG, Zhao ZH, *et al*. RMDDQN-learning: Computation offloading algorithm based on dynamic adaptive multi-objective reinforcement learning in Internet of vehicles. IEEE Transactions on Vehicular Technology, 2023. [doi: 10.1109/TVT.2023.3270967]
18 Sellami B, Hakiri A, Yahia SB, *et al*. Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled IoT network. Computer Networks, 2022, 210: 108957. [doi: 10.1016/j.comnet.2022.108957]
19 Li J, Gao H, Lv TJ, *et al*. Deep reinforcement learning based computation offloading and resource allocation for MEC. Proceedings of the 2018 IEEE Wireless Communications and Networking Conference. Barcelona: IEEE, 2018. 1–6.
20 Li C, Xia JJ, Liu FG, *et al*. Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach. IEEE Transactions on Vehicular Technology, 2021, 70(3): 2922–2927. [doi: 10.1109/TVT.2021.3058995]

(校对责编: 牛欣悦)