计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202221055
60(4): 735−749, 2023

算网融合下时间连续的计算任务卸载机制

郝 昊[1] 杨树杰[2] 张 玮[1]

[1](齐鲁工业大学(山东省科学院)山东省计算中心(国家超级计算济南中心) 济南 250101)

[2](网络与交换技术国家重点实验室(北京邮电大学)北京 100876)

(haoh@sdas.org)

Time-Continuous Computing Task Offloading Mechanism for Computing and Network Convergence

Hao Hao[1], Yang Shujie[2], and Zhang Wei[1]

[1](*Shandong Computer Science Center* (*National Supercomputing Center in Jinan*), *Qilu University of Technology* (*Shandong Academy of Sciences*), *Jinan* 250101)

[2](*State Key Laboratory of Networking and Switching Technology* (*Beijing University of Posts and Telecommunications*), *Beijing*, 100876)

Abstract Computing power network breaks the performance bottleneck of single network point by collaborating the computing resources of cloud servers, edge nodes and devices. Computing power network provides support for intelligent society through computing power. Computing and network convergence has gradually become the trend of new information and communication network technology. Due to heterogeneous computing resources and dynamic network, it is an challenging problem to cooperate the computing resources of cloud, edge and devices to reduce the delay of computing tasks in computing power network. To simplify the problem, existing work often assumes that the system time is discrete and only make computing offloading decisions at the end of time slots. However, this assumption results in decision waiting latency and increases the overall waiting delay of computing tasks. To address this problem, we propose a time-continuous computing task offloading optimization mechanism for computing and network convergence. Under the premise of continues timeline and collaboration of edge nodes, we model the computing offloading problem between cloud, edge and devices with the optimization goal of service experience improvement rate. Besides, we design a computing offloading algorithm based on deep reinforcement learning to improve the utility of computing resources in computing power network. Through extensive simulation experiments, it is demonstrated that the proposed algorithm can effectively reduce the task delay and improve the service experience compared with two baseline algorithms.

Key words computing power network; collaboration of computing resources; task offloading; multi-edge nodes collaboration; temporal continuity

摘 要 算力网络通过网络协同云、边、端计算资源,突破单点算力的性能瓶颈,为智能化社会提供了算力支撑.算网融合逐渐成为新型信息通信网络技术发展的趋势.由于计算资源异构、网络负载动态,如何协同云、边、端计算资源,从而降低计算任务时延是算网融合下极具挑战性的问题之一.为简化问题,现有工作往往假设系统时间是离散的,并且只在时隙结束时进行计算卸载决策.但该假设会引入决策等待时间,增加了计算任务的整体时延.针对上述问题,提出一种算网融合下时间连续的计算任务卸载机制,在保证时

收稿日期: 2022-12-30; **修回日期**: 2023-01-31

基金项目: 山东省自然科学基金项目(ZR2022QF040); 齐鲁工业大学科教产融合试点项目(2022PX083)

This work was supported by the Natural Science Foundation of Shandong Province (ZR2022QF040) and the Pilot Project of Science, Education and Industry of Qilu University of Technology (2022PX083).

通信作者: 张玮(wzhang@sdas.org)

间轴连续和协同多个边缘节点计算资源的前提下,以服务体验提升率为优化目标,对云、边、端间任务卸载问题进行建模,并设计了一种基于深度强化学习的任务卸载方法,从而更高效地利用算力网络计算资源.通过大量的仿真实验证明,与2种基线算法相比所提算法能够有效降低任务时延,提升服务体验.

关键词 算力网络;计算资源协同;任务卸载;多边缘节点协作;时间连续性

中图法分类号 TP393

随着移动终端设备的快速发展,催生出许多计算密集和时延敏感的新型网络服务,它极大地丰富我们生活的同时,也给网络带来了巨大的计算压力.预计到2035年,人均需求算力达到10 000 Gflops,是2018年人均需求算力的21倍[1].但由于工艺的限制和摩尔定律的逐渐失效,传统集约化的数据中心算力增长空间极为有限,将计算任务卸载到远端云服务器中进行集中处理的传统云计算模式将难以满足未来社会发展的需求.作为补充,一种新的计算范式——移动边缘计算(mobile edge computing, MEC)[2]应运而生.与传统云计算相比,移动边缘计算可以在网络边缘节点提供计算能力以支持计算密集型网络服务,从而降低计算任务传输时延和远端云服务器的计算压力.由此,通过由远端云服务器、边缘节点、移动终端设备共同组成的"云—边—端"算力网络(computing power network)架构将逐渐成为未来发展的必然趋势.

算网融合的算力网络可以通过网络协同调度远端云服务器、边缘节点和移动终端的计算资源来为用户提供高质量计算服务.这3种计算模式也各有利弊,其中远端云服务器计算资源充足,但由于传输距离较长、回程链路受限等原因容易导致传输时延过高、网络拥塞等问题;边缘计算通过在边缘节点提供计算能力可以有效缩短传输距离、降低处理时延,但边缘节点计算资源相对有限,难以支持所有的计算任务;本地终端计算则完全没有传输时延,但终端设备本身计算能力十分有限,容易导致计算任务时延过高.因此,如何确定计算任务的卸载位置,协同利用远端云服务器、边缘节点和终端设备的计算资源,是算力网络算网融合下的一个重要的研究方向.

尽管目前研究人员在算网融合下计算卸载方向上进行了大量的尝试和努力,并取得了一系列优异的成果,但由于问题的复杂性,该方向依然面临着诸多挑战.具体来说,本文主要考虑3方面的问题:

1)时间连续性问题.在大部分工作中,为简化问题建模,往往会将时间进行离散化设置,系统仅在每个时隙开始或结束时进行卸载决策.换言之,当某个计算任务到达时,系统不会立刻进行计算卸载决策,而是需要等到时隙的开始或结束时进行统一决策.计算任务的平均决策等待时间为$\Delta/2$,其中Δ为时隙的间隔时长.但时隙的间隔时长往往是不可忽略的,时隙长度与无线信道的相干时间(wireless channel coherence time)有关,在实际系统中通常被设置为10~100 ms[3-4].此外,在很多研究工作中都有一个潜在假设,即计算任务不会跨时隙,所有计算任务需要在一个时隙内完成,这样保证每个时隙可分配的计算资源上限不变,均为设备的总计算能力,从而方便计算能力这一约束条件的建模,但这也就意味着在许多场景中时隙的间隔时长不能太短.因此,若不考虑时间连续性问题,将时间离散化会造成决策等待时延,增加了计算任务的延迟.

2)算法可扩展性问题.在每个时隙中会产生多个计算任务的请求,在时隙开始或结束时,计算决策算法需要确定所有计算任务的卸载状态,因此决策变量往往与计算任务数量相关.特别是一些基于深度强化学习的计算卸载算法,模型构建时输出神经元的数量往往取决于计算任务数量[5].但在实际中,新的计算任务会源源不断产生,输出神经元数量也需要随之改变,这也就意味着之前训练好的深度强化学习模型需要重新训练,造成大量计算资源的浪费.为简化这个问题,很多工作都有一个隐含假设,即计算任务总数不变,不考虑算法的可扩展性.

3)边缘节点的协同问题.现有部分计算卸载方面的工作只考虑了云、边、端间计算资源的纵向协同,却忽略了边缘节点间的横向协作[6],即假设终端设备只能将计算任务卸载到某个特定的边缘节点(如接入的基站),而无法将计算任务卸载到其他边缘节点.但由于边缘节点所在区域的不同,终端设备的数量和任务量也不同,若不考虑边缘节点间的协同,容易导致各个边缘节点间负载不均衡,无法充分利用整体边缘计算资源.因此,边缘节点间的协同是十分必要的,终端设备可以将计算任务卸载到其他边缘节点(即使终端设备与目标边缘节点不直接相连,也可以通过终端设备接入的边缘节点转发至目标节点),从而实现边缘节点间计算资源的协同和计算任务的负载均衡.

针对这 3 个问题，本文提出了一种算网融合下时间连续的计算任务卸载方法，在满足计算任务允许时延阈值条件下，以最大化服务体验提升率为目标，通过深度强化学习方法进行卸载决策. 本文的主要贡献包括 3 个方面：

1）构建了基于时间连续性的计算任务卸载模型. 本文没有将时间轴进行离散化，然后从时隙的角度进行模型构建，而是从计算任务本身出发构建时间连续性的计算卸载模型. 该模型不会在每个时隙结束或开始时才进行计算卸载决策，而是在计算任务到达时立刻进行决策，从而避免了决策等待时延. 此外，模型构建时还考虑了边缘节点间的协同，即终端设备可以选择任意一个边缘节点进行计算卸载.

2）提出了一种基于深度强化学习的计算卸载方法. 首先通过定义系统状态、动作空间、奖励函数等要素将问题模型转化为马尔可夫决策过程，进而通过深度强化学习方法进行求解. 设计时充分考虑了算法的可扩展性，决策变量与计算任务数量无关，仅与边缘节点数量相关，即当边缘节点数量保持不变时，即使计算任务数量发生变化，也不需要修改模型中输出神经元数量，从而提高了算法的可扩展性.

3）通过大量仿真实验验证了所提算法的有效性. 实验验证了算法的收敛性，并在性能方面与其他 2 种先进算法进行了对比. 实验结果表明，所提算法能够有效收敛，并在任务平均时延、服务提升率等方面明显优于其他 2 种对比方法.

1 相关工作

数据经济的发展将产生海量数据，数据处理需要云、边、端协同的强大算力和广泛覆盖的网络连接. 作为一种在云、边、端之间按需分配和灵活调度计算资源、存储资源以及网络资源的新型信息基础设施，算力网络已经引起了产业界和学术界的广泛关注. 2019 年，中国联通联合多家高校和科研院所发布了第一个算力网络方面的白皮书——《中国联通算力网络白皮书》[7]，其中明确了算力网络的体系框架，即算力必然会从云和端向网络边缘进行扩散，网络将出现云、边、端三级算力架构. 2020 年，华为在《泛在算力：智能社会的基石》[1] 文件中指出，由云—边—端组成的泛在算力架构是技术发展至今的必然选择结果，未来算力必然构建“云—边—端”泛在部署架构，满足社会智能化发展带来的算力需求. 中国移动在 2021 年发布的《中国移动算力网络白皮书》[8] 中也明确指出，以算为中心，构筑云—边—端立体泛在的算力体系. 算力网络标准化工作也在持续进行，2021 年 7 月在国际电信联盟电信标准化部门（Telecommunication Standardization Sector of the International Telecommunications Union, ITU-T）第 13 研究组报告人会议上，通过了由中国电信研究院网络技术研究所雷波牵头的算力网络框架与架构标准（Computing Power Network- framework and Architecture）系列编号 Y. 2501，该标准是首项获得国际标准化组织通过的算力网络标准.

近年来，计算任务卸载作为算力网络算网融合下的一个重点研究内容，引起了学术界的诸多关注. 部分工作[9-12] 研究了仅包含单个边缘节点的计算卸载问题. 文献 [9] 考虑了一个多用户、单边缘服务器和单云服务器的 3 层计算卸载场景，在问题建模时加入了传输链路的带宽分配，将计算卸载和带宽分配的联合优化构建成一个分段凸规划问题，并提出了一种具有较强鲁棒性的优化算法. 文献 [10] 研究了面向移动云计算的计算任务动态卸载问题，将其描述成满足任务依赖要求和时间期限约束的能效成本最小化问题，并提出了一种最优的动态卸载算法，可以根据无线信道状态动态改变计算任务卸载速率. 文献 [11] 着重考虑了计算卸载时的能耗问题，将其建模为随机优化问题，目标是在保证平均等待队列长度的同时最小化任务卸载的能量消耗，通过随机优化方法将原始问题进行了转化，并提出了一种动态节能的计算卸载方法. 文献 [12] 则整体考虑了计算任务时延和能量消耗 2 方面的因素，将计算卸载问题转化为多目标优化问题，并结合并行深度神经网络和深度 Q 学习算法设计了一种新的深度元强化学习方法对问题进行求解.

当系统中包含多个边缘节点时，由于每个边缘节点服务范围内终端设备数量不同、计算任务量不同，单边缘节点卸载模型容易导致部分边缘节点计算压力大，而部分边缘节点相对空闲，即出现边缘节点间负载不均衡的现象，从而影响服务体验. 因此，部分最新研究工作[13-19] 考虑了多边缘节点协作共同完成计算任务. 文献 [13] 提出了一种面向多边缘设备协作的计算卸载优化机制，将优化问题建模成混合整数非线性规划问题，并设计了一种基于偏好的双边匹配算法，在降低任务整体执行时延的同时实现边缘设备间的负载均衡. 文献 [14] 通过计算量、通信成本等方面对应用程序任务进行分类，帮助计算卸载决策，并提出了一种基于贪婪任务图划分的卸

载算法，采用贪婪优化方法使任务通信成本最小化. 文献 [15] 关注于具有统计服务质量保证的计算卸载问题，放松了对计算任务时延阈值的限制，将必须在时限内完成计算任务改变为完成概率高于某个阈值即可，将其表述为具有统计延迟约束的混合整数非线性规划问题，并通过凸优化理论和吉布斯抽样方法对问题进行了求解. 文献 [16] 考虑了计算卸载时任务延迟和无线传输时能耗间的均衡问题，通过对系统流量的预测，提出了一种高效、分布式的多层雾计算系统预测卸载和资源分配方案，只需少量预测信息值就可显著降低任务时延. 文献 [17] 构建了一种基于软件定义的细粒度多接入边缘计算架构，对网络和计算资源进行协同管理，并设计了一种基于深度强化学习的 2 级资源分配策略，从而提供有效的计算卸载服务. 由于终端设备的移动性，终端设备与边缘节点间的连接可能不稳定，这个将会影响卸载决策，甚至导致卸载失败. 文献 [18] 则加入了移动性方面的考虑，提出了一种具有故障恢复的高鲁棒性计算卸载策略，在终端设备移动的情况下也可降低能量消耗和缩短任务完成时间. 文献 [19] 研究了工业物联网中的计算卸载问题，提出了整合迁移成本和计算时延的协同卸载框架，并设计了以最小化系统成本为目标的基于学习的任务协同卸载算法，从而在不需要完整卸载信息的情况下选择最优边缘节点.

然而，当前关于算力网络算网融合下计算任务卸载方面的研究大多没有考虑时间连续性问题，将时间进行离散化处理，导致产生额外的决策等待时延. 而且其中一些基于深度强化学习设计的卸载算法只考虑了固定数量的计算任务，当计算任务数量改变时，往往需要修改神经网络输出神经元数量并重新进行训练，算法的可扩展性较差.

综上所述，本文针对多边缘节点协作的算网融合场景，研究基于时间连续性的计算任务协作卸载问题，提出一种可扩展性强的、基于深度强化学习的计算任务卸载算法，目标是在满足计算任务延迟阈值的条件下，最大化服务体验提升率.

2 系统模型与问题描述

本节首先进行场景描述；然后介绍系统模型，包括终端设备模型和其他节点模型；最后对本文要解决的问题进行数学建模. 本文涉及的主要变量及相关描述如表 1 所示.

Table 1 Variable Notations

表 1 变量符号

变量	描述
N	边缘节点数量
M	终端设备数量
c_k	第 k 个计算任务需要的计算量
u_k	第 k 个计算任务的传输数据量
t_k	第 k 个计算任务的生成时刻
d_k	第 k 个计算任务允许的延迟阈值
m_k	产生第 k 个计算任务的终端设备
x_k	第 k 个计算任务是否本地计算的决策变量
y_k	第 k 个计算任务的卸载目标节点
$q_m(k)$	第 k 个任务产生后终端设备 m 的计算队列
$Q_n(k)$	第 k 个任务产生后边缘节点 n 的计算队列
f_m^{device}	终端设备 m 的计算能力
f_n^{edge}	边缘节点 n 的计算能力
f_m^{cloud}	云服务器为终端设备 m 分配的计算能力

2.1 场景描述

本文针对的是云边端协同的泛在算力网络架构，如图 1 所示，云为远端云服务器，边为边缘节点，端主要包括用户终端设备，各节点间通过无线链路进行数据传输. 终端设备在产生计算任务后，既可以选择在本地进行计算，又可以通过无线链路卸载到边缘节点或远端云服务器进行计算，从而降低自身计算压力，提升服务质量.

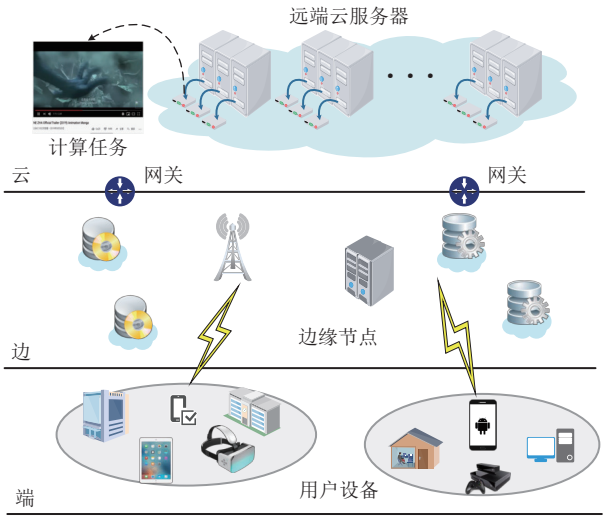

Fig. 1 Illustration of our system architecture

图 1 本文系统架构示意图

假设云边端协同的泛在算力网络中包含 N 个边缘节点、M 个终端设备和 1 个远端云服务中心. 其中每个边缘节点 $n \in \{1, 2, \cdots, N\}$ 都配置有一个服务器，

可提供的计算能力为f_n^{edge}(单位为CPU周期数/s). 每个终端设备$m \in \{1, 2, \cdots, M\}$也具备一定的计算能力，为$f_m^{\text{device}}$(单位为CPU周期数/s). 远端云服务器具有充足的计算能力，会为每个终端设备m分配固定的计算能力f_m^{cloud}. 除了云边端的纵向协同模式外，边缘节点间也可以相互协作，即终端设备并非只能卸载到自身所属的某个边缘节点(如只能卸载到自身接入的边缘基站)，而是可以将计算任务卸载到任意一个边缘节点上.

传统建模主要是通过时隙的增加来体现系统时间的演变，本文为保证时间的连续性，不再将时间建模成离散的多个时隙，因此无法通过时隙的增加来描述系统时间的变化. 但问题优化目标是长期平均性能，问题建模时仍需要系统时间的演变. 为此，本文从计算任务产生顺序的角度进行建模，即计算任务k并不表示某一类或某一个具体的计算任务，而是指系统运行后终端设备产生的第k个计算任务，这样通过计算任务编号的增加来体现系统时间的演进. 例如，对视频A进行解码这一计算任务，终端设备a和终端设备b均产生了该计算任务，但分别对应整个网络系统的第6个计算任务和第9个计算任务，虽然任务相同，由于2个终端设备产生该任务的时间不同，在本文中也将会以第6个计算任务和第9个计算任务来分别描述. 而在其他文献中，计算任务k往往表示某个具体任务，而非任务产生的排序，即对于相同任务而言，即使产生时间不同其表述也相同，如若用计算任务k表示对视频A进行解码这一计算任务，则无论此任务何时产生，其表述都为计算任务k. 可以用一个五元组$(c_k, u_k, t_k, d_k, m_k)$来表示第$k$个计算任务的各方面属性，其中$c_k$为完成该计算任务需要的计算量，$u_k$为该计算任务卸载时需要传输的数据量，$t_k$为生成该计算任务的时刻($t_k$是一个连续值)，$d_k$为该计算任务允许的延迟阈值，$m_k$为生成该计算任务的终端设备.

与文献[20−21]一致，本文假设计算任务是不可分割的. 如图2所示，当用户终端设备产生新计算任务时，首先需要决定是在本地处理还是需要卸载到边缘节点或远端云服务器. 如果终端设备选择本地处理计算任务，将该计算任务加入到本地计算队列中进行等待；否则终端设备需要决定将计算任务卸载到哪个边缘节点或远端云服务器，并将计算任务加入到该节点的计算队列中.

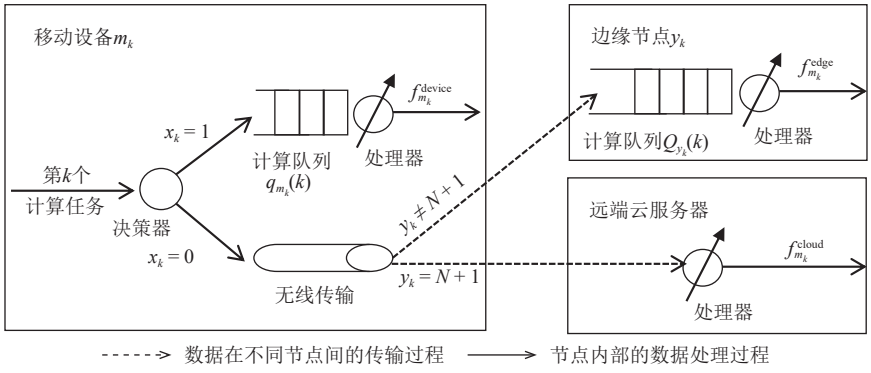

Fig. 2 Process of computing task

图2 计算任务处理过程

用二元变量$x_k \in \{0, 1\}$来表示第k个计算任务的卸载决策. 若x_k=1则表示第k个计算任务会在本地终端设备进行处理；若x_k=0则表示第k个计算任务不会在本地终端设备进行处理，需要进行计算卸载. 若第k个计算任务不进行本地计算，还需确定其计算卸载位置，用多元变量$y_k \in \{1, 2, \cdots, N, N+1\}$表示第$k$个计算任务卸载的位置，若$y_k \in \{1, 2, \cdots, N\}$则表示第$k$个计算任务需要卸载到编号为$y_k$的边缘节点上处理；若$y_k = N+1$则表示第$k$个计算任务需要卸载到远端云服务器上进行处理.

2.2 终端设备模型

计算任务进行本地处理时，需要将该计算任务加入到相应终端设备的本地计算队列中，并按照先进先出(first-in first out, FIFO)的模式进行服务[22]，此时终端设备计算队列模型的更新分为2种情况：生成该计算任务的终端设备计算队列更新和其他终端设备的计算队列更新.

1)对于生成第k个计算任务的终端设备m_k而言，其需要本地计算第k个计算任务，计算队列长度增加第k个任务的计算量，表示为

$$q_{m_k}(k)=[q_{m_k}(k-1)-(t_k-t_{k-1})f_{m_k}^{\text{device}}]^+ +c_k, \tag{1}$$

其中运算$[z]^+=\max\{0,z\}$，t_k-t_{k-1}为第$k-1$个计算任务产生到第k个计算任务产生时的间隔时长，$(t_k-t_{k-1})f_{m_k}^{\text{device}}$则代表在此间隔时长中终端设备$m_k$能完成的计算任务量，$q_{m_k}(k-1)$为第$k-1$个计算任务产生后终端设备$m_k$的计算队列积压量.

2)对于其他终端设备($m\neq m_k$)而言，第k个计算任务进行本地计算不会增加自身计算任务量，则其计算队列表示为

$$q_m(k)=[q_m(k-1)-(t_k-t_{k-1})f_m^{\text{device}}]^+. \tag{2}$$

综上2种情况，若第k个计算任务进行本地计算，终端设备m的计算队列长度表示为

$$q_m(k)=[q_m(k-1)-(t_k-t_{k-1})f_m^{\text{device}}]^+ +L_{\{m==m_k\}}c_k, \tag{3}$$

其中若条件z为真，则$L_{\{z\}}=1$，若条件z为假，则$L_{\{z\}}=0$.

需要说明的是在离散时间建模中大多假设计算任务可以在一个时隙内完成，主要记录每个时隙内到达的计算任务和其需要的计算资源，没有构建计算等待队列，其等待时间计算公式为$T^{\text{l}}(k)=c_k/f_{m_k}^{\text{device}}$，其中$f_{m_k}^{\text{device}}$是终端设备$m$为计算任务$k$分配的计算资源. 等待时间计算公式同样适用于离散时间建模下的边缘节点.

由于计算任务是在本地处理计算，因此不需要进行数据的传输，此时第k个计算任务的时延为

$$T^{\text{l}}(k)=\frac{q_{m_k}(k)}{f_{m_k}^{\text{device}}}, \tag{4}$$

值得一提，本文虽然采取的是先进先出无优先级的服务模式，但这项工作也可以扩展到计算任务具有不同优先级的场景. 具体来说，每个终端设备不再只有一个计算等待队列，而是对每类优先级的计算任务分别建立一个计算等待队列，即计算等待队列数量与优先级的数量相同. 当计算任务在本地计算时，会将该计算任务添加到对应优先级的计算等待队列中. 计算时延时，将不低于本计算任务优先级的等待队列相加即可确定在该计算任务前需要完成的计算量，再除以设备计算能力即可获得任务的计算时延.

若计算任务k需要卸载到边缘节点或远端云服务器，除了对终端设备本地队列进行更新外，还需要计算终端设备m_k通过无线链路将计算任务k传输到相应节点上的传输时间. 考虑终端设备在正交信道上传输的无线网络模型[23]，终端设备到边缘节点或远端云服务器的无线传输链路存在信号衰减. 用$|h_{m,n}|^2$代表从终端设备m到节点n(边缘节点或远端云服务器)的信道增益，则其传输速率为

$$rate_{m,n}=W_{m,n}\text{lb}\left(1+\frac{|h_{m,n}|^2P_m}{\sigma_{m,n}^2}\right), \tag{5}$$

其中$W_{m,n}$为终端设备m到节点n的链路带宽，P_m为终端设备m的传输功率，$\sigma_{m,n}^2$表示噪声功率. 需要说明的是，若终端设备m和节点n间不直接相连，可以通过其他节点进行转发，在模型计算时调整链路带宽也可以模拟估计出其传输速率，而不用对每个转发过程进行单独建模. 若第k个计算任务卸载到编号n的节点上处理，则其传输时延为

$$T_n^{\text{u}}(k)=\frac{u_k}{rate_{m_k,n}}. \tag{6}$$

2.3 其他节点模型

当终端设备将计算任务卸载到边缘节点时，即x_k=0&&$y_k\in\{1,2,\cdots,N\}$，由于边缘节点自身计算能力有限，且需要为多个终端设备提供服务，因此也会形成计算队列的积压，边缘节点计算队列的到达是终端设备卸载到该边缘节点的计算任务，计算队列的减少是该边缘节点根据自身计算能力完成的计算任务量. 与终端设备计算队列相似，边缘节点的计算队列也采用先进先出的服务模式，队列更新也分为目标节点计算队列更新和非目标节点计算队列更新2种情况.

当第k个计算任务从终端设备卸载到边缘节点时，若该边缘节点为卸载的目标节点，即$n=y_k$，则其需要将第k个计算任务加入到自身的计算队列中，计算队列表示为

$$Q_n(k)=[Q_n(k-1)-(t_k-t_{k-1})f_n^{\text{edge}}]^+ +c_k, \tag{7}$$

其中$(t_k-t_{k-1})f_n^{\text{edge}}$为在时间间隔内边缘节点能够完成的计算量. 若该边缘节点不是目标节点，即$n\neq y_k$，则不需要将第k个计算任务加入到自身队列，计算队列更新为

$$Q_n(k)=[Q_n(k-1)-(t_k-t_{k-1})f_n^{\text{edge}}]^+. \tag{8}$$

综上，第k个计算任务进行卸载后，边缘节点n的计算队列长度更新为

$$Q_n(k)=[Q_n(k-1)-(t_k-t_{k-1})f_n^{\text{edge}}]^+ +L_{\{n==y_k\}}c_k, \tag{9}$$

此时，第k个计算任务的计算时延为

$$T^{\text{e}}(k)=\frac{Q_{y_k}(k)}{f_{y_k}^{\text{edge}}}. \tag{10}$$

除了边缘节点外，终端设备还可以将计算任务卸载到远端云服务器，即x_k=0&&$y_k=N+1$，与边缘节点不同，远端云服务器具有充足的计算资源，可以为每个终端设备提供固定的计算能力[24]，从而快速完成计算任务. 因此，若第k个计算任务卸载到远端云服务器中，则不需要考虑云服务器中计算队列的积

压，即计算队列长度为 0，计算时延为处理计算任务本身的时间：

$$T^{\mathrm{c}}(k)=\frac{c_k}{f_{m_k}^{\mathrm{cloud}}}, \tag{11}$$

其中 $f_{m_k}^{\mathrm{cloud}}$ 为远端云服务器为终端设备 m_k 提供的计算资源量.

2.4 云边端计算卸载问题建模

在云边端协同的计算卸载模式中，计算任务可以在终端设备、任意边缘节点或远端云服务器上执行，其服务时延为

$$T(k)=x_kT^{\mathrm{l}}(k)+(1-x_k)[T_{y_k}^{\mathrm{u}}(k)+T^{\mathrm{o}}(k)], \tag{12}$$

其中 $T_{y_k}^{\mathrm{u}}(k)$ 为将计算任务卸载到其他节点 y_k 的传输时延，$T^{\mathrm{o}}(k)$ 为在边缘节点或远端云服务器上的计算时延. 由 2 部分相加组成，当 $y_k\neq N+1$ 时，说明计算任务在边缘节点上执行；当 $y_k=N+1$ 时，说明计算任务在远端云服务器上执行. 因此 $T^{o}(k)$ 定义为

$$T^{\mathrm{o}}(k)=L_{\{y_k\neq N+1\}}T^{\mathrm{e}}(k)+(1-L_{\{y_k\neq N+1\}})T^{\mathrm{c}}(k). \tag{13}$$

计算任务协同卸载可以协同利用云边端的计算资源完成计算任务，而在云计算模式中只能通过远端云服务器来完成相关计算任务. 为了直观体现云边端协同卸载带来的优势，以云计算模式下服务时延作为对比，定义了服务体验提升率 R_k 评价指标：

$$R_k=\left(1-\frac{T(k)}{T^{\mathrm{c}}(k)+\dfrac{u_k}{r_{m_k,N+1}}}\right)\times 100\%, \tag{14}$$

其中 $\dfrac{u_k}{r_{m_k,N+1}}$ 为将计算任务传输到远端云服务器的传输时延，$T^{\mathrm{c}}(k)+\dfrac{u_k}{r_{m_k,N+1}}$ 为该任务通过云计算模式完成所需要的服务时延.

服务体验提升率是一个相对性指标，服务体验提升率越高说明与云计算模式相比，云边端协同的计算卸载模式能够减少的时延越多，服务体验越好. 与直接优化服务时延相比，优化服务体验提升率能更好反映各个计算任务的服务体验，降低某些极端计算任务对整体优化目标的影响. 因此，将平均服务体验提升率定为云边端计算卸载问题的优化目标，建立优化问题模型为：

$$\max_{X,Y}\ \lim_{K\to\infty}\frac{1}{K}\sum_{k=1}^{K}R_k, \tag{15}$$

$$\text{s.t.}\quad T(k)\leqslant d_k, \tag{15.1}$$

$$\text{式(3)}-\text{(6)},\ \text{(9)}-\text{(11)}, \tag{15.2}$$

$$x_k\in\{0,1\}, \tag{15.3}$$

$$y_k\in\{1,2,\cdots,N,N+1\}, \tag{15.4}$$

其中 $X=\{x_1,x_2,\cdots,x_k,\cdots x_K\}$ 为是否进行本地计算的决策变量，$Y=\{y_1,y_2,\cdots,y_k,\cdots y_K\}$ 为计算卸载的目标选择变量，约束条件(15.1)保证计算任务能够在计算任务允许的延迟阈值内完成，约束条件(15.2)为终端设备等待队列、边缘节点等待队列、云服务排队状态等相关约束，约束条件(15.3)和(15.4)是对计算卸载变量取值的约束.

与离散时间建模相比，对计算卸载问题进行连续时间建模的优势主要有 2 个方面：1)避免决策等待时延. 在离散时间建模中，时间轴被分成了离散的时隙，并只在时隙结束时进行卸载决策，即当计算任务到达时并不会立刻进行决策而是需要等待时隙的结束，但时隙的间隔时长往往是不可忽略的[3-4]，导致较长的决策等待时间；在连续时间建模中，计算任务到达后可以立刻进行卸载决策，不需要等到时隙结束时进行统一决策，从而避免了决策等待时延，降低了计算任务的整体等待时延. 2)提升算法模型的可扩展性. 在离散时间建模中，由于一个时隙内会有多个计算任务请求，卸载决策时需要确定所有任务的状态，因此决策模型与计算任务数量相关，即计算任务数量变化时，决策模型也需要随之修改；在连续时间建模中，计算任务到达后会立刻对其进行卸载决策，决策模型每次只需要确定一个计算任务的卸载状态，与计算任务数量无关，因此可以适应不同计算任务数量的场景，从而提升了算法模型的可扩展性.

云边端计算卸载问题模型的目标是对长时间平均服务体验提升率的优化，该类问题若通过动态规划等传统方式进行求解就往往需要获取与用户请求相关的完整状态转移概率，即获取所有时间内的用户请求后才会做出决策，因此传统方法需要预测未来信息. 然而，由于网络系统的动态特性，这种预测是十分困难的. 而深度强化学习通过数据采样的方式代替状态转移概率，可以实现无模型学习，即只需要当前状态，而不用预测未来信息，是解决这类问题的有效方法.

3 问题的马尔可夫决策过程

通过深度强化学习解决的问题往往可以描述为马尔可夫决策过程，本节将云边端计算卸载问题描述为马尔可夫决策过程，从而便于后续问题的求解.

由于连续时间建模下没有定义时隙，因此将原来时隙变化而导致状态转移修改为计算任务编号增加而导致状态转移，即每个新计算任务为一个状态节点而非每个时隙为一个状态节点. 具体思路为：在

新计算任务产生时,终端设备会观察系统状态(例如,任务大小、计算队列信息),然后基于系统状态为该任务选择卸载动作(即本地计算卸载到边缘节点、卸载到云服务器等),该状态和动作将产生相应奖励(例如,任务时延减少、超出任务延迟阈值受到惩罚).每个终端设备的目标是通过优化从状态映射到动作的策略来最大化系统长期奖励.

3.1 系统状态

当第 k 个计算任务产生时,产生该任务的终端设备 m_k 当前的负载状态是影响卸载决策的重要因素之一.用开始处理第 k 个计算任务前需要的等待时延表示终端设备的负载状态,其定义为

$$t^d(k)=\frac{[q_{m_k}(k-1)-(t_k-t_{k-1})f_{m_k}^{\text{device}}]^+}{f_{m_k}^{\text{device}}}, \tag{16}$$

其中式(16)的分子为第 k 个计算任务产生时终端设备 m_k 积压的计算量,分母为终端设备的计算能力.

除终端设备 m_k 的负载状态外,边缘节点的当前计算负载状态也是影响卸载决策的重要因素,与终端设备相似,用边缘节点开始处理第 k 个计算任务前需要的等待时延表示边缘节点的当前负载状态:

$$T_n^{\text{d}}(k)=\frac{[Q_n(k-1)-(t_k-t_{k-1})f_n^{\text{edge}}]^+}{f_n^{\text{edge}}}, \tag{17}$$

除节点当前负载信息外,还需要考虑任务本身的一些属性,具体来说,当第 k 个计算任务产生时终端设备 m_k 观察到的系统状态定义为

$$s_k=\left(t^{\text{d}}(k), f_{m_k}^{\text{device}}, \boldsymbol{T}^{\text{d}}(k), \boldsymbol{T}^{\text{u}}(k), \boldsymbol{f}^{\text{edge}}, f_{m_k}^{\text{cloud}}, c_k, d_k\right), \tag{18}$$

其中 $\boldsymbol{T}^{\text{d}}(k)$ 为所有边缘节点等待时延组成的向量:

$$\boldsymbol{T}^{\text{d}}(k)=\left(T_1^{\text{d}}(k), \cdots, T_n^{\text{d}}(k), \cdots, T_N^{\text{d}}(k)\right). \tag{19}$$

$\boldsymbol{T}^{\text{u}}(k)$ 为计算任务 k 传输到所有其他节点(边缘节点或远端云服务器)需要的传输时延:

$$\boldsymbol{T}^{\text{u}}(k)=(T_1^{\text{u}}(k), \cdots, T_n^{\text{u}}(k), \cdots, T_N^{\text{u}}(k), T_{N+1}^{\text{u}}(k)), \tag{20}$$

$\boldsymbol{f}^{\text{edge}}$ 为所有边缘节点计算能力组成的向量:

$$\boldsymbol{f}^{\text{edge}}=\left(f_1^{\text{edge}}, \cdots, f_n^{\text{edge}}, \cdots, f_N^{\text{edge}}\right). \tag{21}$$

系统状态组成要素中,$t^{\text{d}}(k)$, $f_{m_k}^{\text{device}}$, $\boldsymbol{f}^{\text{edge}}$, $f_{m_k}^{\text{cloud}}$, c_k 这几个要素是终端设备已知或固定的,因此主要需获取 $\boldsymbol{T}^{\text{d}}(k)$, $\boldsymbol{T}^{\text{u}}(k)$,特别是终端设备位置发生改变后,则其传输时延需要根据当前链路状态重新计算.

3.2 动 作

当第 k 个计算任务产生后,终端设备 m_k 需要对该任务进行卸载决策:①是否在本地处理计算任务,即确定 $x_k=0$ 或 $x_k=1$;②若需要进行计算任务卸载,则进一步确定其卸载的位置,卸载到哪个边缘节点或远端云服务器,即确定 y_k 的取值.因此,当第 k 个计算任务产生后,终端设备 m_k 采取的动作可以表示为

$$a_k=(x_k, y_k). \tag{22}$$

当 $x_k=1$ 即计算任务在本地执行时,y_k 的取值就无关紧要;只有当 $x_k=0$ 即计算任务需要进行卸载时,y_k 的取值才决定计算卸载的位置.因此,问题动作空间的大小为 $N+2$.在其他基于时间离散的计算卸载策略中,每个时刻需要为所有计算任务进行卸载决策,其动作空间的大小往往与计算任务数量成指数关系,增加了模型训练的难度,而且计算任务的数量往往是动态变化的,这就导致当任务数量改变时,需要重新训练决策模型,可扩展性差.而该模型的优势在于其动作空间的大小只与边缘节点的数量成线性关系,与服务数量无关.因此,本文模型的训练成本更低,也更具有可扩展性.

3.3 奖励函数

在本文构建的云边端计算卸载问题中优化目标为最大化平均服务体验提升率,因此如果一个动作能带来更高的服务体验提升率,其获得的奖励值也应越大.此外,每个计算任务都有其允许的延迟阈值,若超出这个阈值,则应受到一定的惩罚.基于此基本准则,奖励函数定义为

$$r_k=\begin{cases} R_k, \text{满足约束条件(15.1)}, \\ Pu, \text{不满足约束条件(15.1)}. \end{cases} \tag{23}$$

4 基于深度强化学习的计算卸载算法

本节提出了一种基于深度强化学习的计算卸载算法,可以让每个终端设备分布式的进行卸载决策.该算法基于深度 Q 学习(deep Q learning, DQN)模型方法[25],可以在不获取先验知识的前提下,通过数据采样的方式对优化问题进行求解.而且,该方法通过神经网络来预测获得 Q 值(状态-动作对的预期长期回报),每个输出神经元对应一种动作的 Q 值,可以有效解决状态空间大、动作离散的优化问题.本节将分别介绍构建的神经网络模型和基于 DQN 的计算卸载算法.

4.1 神经网络模型

神经网络的目标是建立状态-动作对到 Q 值的映射,即通过学习训练获得每个状态-动作对的 Q 值.本文采用 Double DQN(DDQN)训练模式提升模型训练的效率,图 3 为构建的神经网络模型示意图,具体来说,系统状态信息通过输入层传递到神经网络中,此后通过 2 个全连接层(full connected layer, FC)学习从状态-动作对到 Q 值的映射.

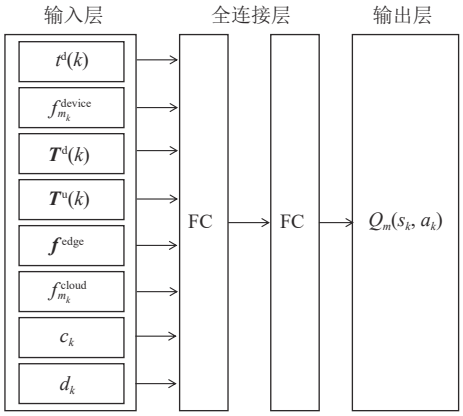

Fig. 3 Illustration of neural network model

图 3 神经网络模型示意图

1)输入层. 该层负责将系统状态作为输入传递到神经网络中, 输入层神经元数量与系统状态的变量参数相关, 如式(18)所示, 系统状态包括本地计算延迟、传输时延等 7 方面的参量, 因此输入神经元的数量为$1+1+N+(N+1)+N+1+1=3N+5$.

2)全连接层. 2 个 FC 层负责学习从状态到对应动作 Q 值的映射关系, 每个神经元层包含一组具有线性整流函数(rectified linear unit, ReLU)的神经元, 这些神经元与前一层和后一层的神经元相连.

3)输出层. 每个输出神经元对应于一个动作的 Q 值, 动作空间的大小为 $N+2$, 因此输出神经元的数量也为 $N+2$.

DDQN 模型训练方式为: 为避免过估计问题(overestimate)而导致模型训练稳定性差, 本文采用 DDQN 的学习方式. 有 2 个具有相同结构的神经网络模型, 一个称为主网络, 另一个称为目标网络, 其中主网络是我们需要训练学习的模型, 目标网络用来生成 Q 值的预测值, 参数更新公式为

$$a_{\mathrm{m}} = \arg\max_{a} Q_{\mathrm{m}}(s_{k+1}, a), \tag{24}$$

$$Q_{\mathrm{m}}(s_k, a_k) = Q_{\mathrm{m}}(s_k, a_k) + \eta(r_k + \gamma Q_{\mathrm{t}}(s_{k+1}, a_{\mathrm{m}}) - Q_{\mathrm{m}}(s_k, a_k)), \tag{25}$$

其中$Q_{\mathrm{m}}(s, a)$代表主网络的输出 Q 值, $Q_{\mathrm{t}}(s, a)$代表目标网络的输出 Q 值, η 为学习速率, γ为折扣因子. 模型采用经验回放的方式进行学习, 即将探索形成的数据组进行随机打乱, 使神经网络可以重复多次训练学习, 从而打乱样本间的关联性, 提升样本利用效率.

4.2 基于深度强化学习的求解算法

本文提出的基于深度强化学习的计算卸载方法最终将运行在终端设备上, 帮助终端设备进行计算卸载决策. 但在模型的训练期间, 需要消耗大量的计算资源, 因此, 采用边缘节点辅助的方式进行训练. 在模型训练期间终端设备 m 可以让与自身具有最大传输能力的边缘节点n_m进行辅助训练, 然后边缘节点n_m将训练好的模型参数传输给终端设备 m, 终端设备 m 则可以应用训练好的模型直接进行计算卸载决策.

算法 1 和算法 2 分别描述了在终端设备 m 和边缘节点n_m上执行的深度强化学习算法, 其主要思想是将终端设备给出的经验信息(即状态、动作、奖励和下一个动作)传输给对应的边缘节点, 边缘节点利用经验信息来训练神经网络, 得到模型参数, 再将模型参数传回终端设备, 从而实现计算卸载决策.

终端设备主要负责信息的收集, 在生成第 1 个计算任务时, 对终端设备 m 上状态信息进行初始化:

$$s_1 = (t^{\mathrm{d}}(L), f_{m_1}^{\mathrm{device}}, \boldsymbol{T}^{\mathrm{d}}(L), \boldsymbol{T}^{\mathrm{u}}(L), \boldsymbol{f}^{\mathrm{edge}}, f_{m_1}^{\mathrm{cloud}}, c_1, d_1), \tag{26}$$

其中$t^{\mathrm{d}}(L)=0$, $\boldsymbol{T}^{\mathrm{d}}(L)=0$, 表示在进行第 1 个计算任务时终端设备和边缘节点均无计算任务的积压. 终端设备 m 上运行的具体步骤总结如算法 1 所示.

算法 1. 终端设备执行的算法.

输入: 神经网络参数、系统状态信息;

输出: 经验信息四元组.

① 初始化神经网络参数、系统状态信息;

② while 模型训练 do

③ if 设备 m 生成新的第 k 个计算任务 then

④ 向边缘节点n_m发送神经网络参数请求;

⑤ 接收神经网络参数;

⑥ 接收边缘节点当前的状态信息;

⑦ 根据 ε 贪心算法选择动作a_k;

⑧ 观察动作选择后的下一个状态s_{k+1};

⑨ 计算在状态s_k下执行动作a_k的奖励r_k;

⑩ 将四元组(s_k, a_k, r_k, s_{k+1})发送至对应的边缘节点n_m

⑪ end if

⑫ end while

在终端设备 m 产生新任务时, 会向其对应的边缘节点发送参数更新请求, 从而更新自身神经网络模型和其他状态信息, 但过于频繁的参数更新也会加重终端设备的传输压力, 因此这里有一个潜在问题: 如何根据传输链路质量、模型收敛速度来确定参数更新频率, 换言之, 算法 1 中的步骤④~⑥并不一定每次执行. 为实现模型的快速收敛, 本文实验中终端设备每次产生新任务时均会发送参数更新请求. 根据接收到的神经网络模型参数$\boldsymbol{\theta}_m$, 通过 ε 贪心算法选择动作, 即:

$$a_k = \begin{cases} \arg\max\limits_{a} Q_{\mathrm{m}}(s_{k+1}, a; \boldsymbol{\theta}_m), & 1-\varepsilon\text{的概率}, \\ \text{从动作空间中随机选择}\varepsilon\text{的概率}. \end{cases} \tag{27}$$

边缘节点负责神经网络模型的训练, 模型训练

时包括 2 个神经网络：主网络 Net_m和目标网络 Net'_m，Net_m和 Net'_m具有相同的神经网络结构，但其参数不同，分别表示为$\boldsymbol{\theta}_m$和$\boldsymbol{\theta}'_m$，其中终端设备接收到的神经网络参数即为主网络参数$\boldsymbol{\theta}_m$. Net_m用于动作的选择，Net'_m则用来计算目标 Q 值.

边缘节点神经网络模型训练过程如算法 2 所示.

算法 2. 边缘节点执行的模型训练算法.

输入：经验信息四元组，更新频率 L；

输出：神经网络模型参数.

① 用随机化方式初始化 Net_m的参数$\boldsymbol{\theta}_m$；
② 用随机化方式初始化 Net'_m的参数$\boldsymbol{\theta}'_m$；
③ 初始化经验池，且初始化变量 $count$=0；
④ while 模型训练 do
⑤ if 接收到终端设备m的参数请求 then
⑥ 发送主网络参数$\boldsymbol{\theta}_m$给设备 m；
⑦ end if
⑧ if 接收到经验信息四元组 then
⑨ 将经验信息四元组存放到经验池中；
⑩ 从经验池中随机选取一批样本；
⑪ for 每一个样本 do
⑫ 根据式(28)计算目标 Q 值；
⑬ end for
⑭ 根据式(29)计算平均损失函数；
⑮ 根据平均损失函数通过梯度下降的方式更新主网络参数$\boldsymbol{\theta}_m$；
⑯ $count$:= $count$ +1；
⑰ if 变量 $count$ 是 L 的倍数 then
⑱ $\boldsymbol{\theta}'_m$:= $\boldsymbol{\theta}_m$；
⑲ end if
⑳ end if
㉑ end while

在初始化 Net_m和 Net'_m以及经验池后，边缘节点开始等待终端设备 m 的请求消息. 如果接收到的是参数请求消息，边缘节点则将 Net_m当前的参数向量发送给终端设备 m；如果接收到的是一个经验信息四元组，则将其存储到经验池中. 算法 2 中的行⑩~⑲为模型训练，首先从经验池中随机选取一批样本，然后根据样本信息计算目标 Q 值：

$$Q(i)=r_i+\gamma Q_t(s(i+1),\arg\max_a Q_m(s(i+1),a;\boldsymbol{\theta}_m);\boldsymbol{\theta}'_m), \tag{28}$$

其中 i 代表一批样本中的第 i 个样本. 模型优化的目标是减小 Net_m输出的 Q 值与目标 Q 值间的差距，即通过迭代优化算法(如梯度下降等)的方式最小化损失函数

$$Loss=\frac{1}{P}\sum_{i=1}^{P}(Q_m(s(i),a(i);\boldsymbol{\theta}_m)-Q(i))^2, \tag{29}$$

其中 P 代表一批样本中样本的数量. 让 $count$ 代表模型训练的次数，即每当模型训练 L 次后，就需要用主网络参数$\boldsymbol{\theta}_m$更新目标网络参数$\boldsymbol{\theta}'_m$，如算法 2 中行⑰~⑲所示.

4.3 算法分析

由于用户在一段时间内兴趣偏好不会发生改变，神经网络模型并不需要实时训练更新，可以采用离线训练的方式进行更新，如模型可以每 2 个周更新训练一次，而其他时间只是使用训练好的模型进行卸载决策，而不进行训练. 因此，主要分析终端设备每次计算卸载决策时的算法复杂度，而非模型训练时的复杂度. 终端设备根据神经网络输出结果进行卸载决策，在此过程中只涉及到神经网络正向传播的过程，以乘积数量表示其计算复杂度，其中输入层神经元数量为$3N+5$，全连接层神经元数量也与 N 相关，因此输入层到第 1 个全连接层的计算复杂度为 $O(3N\times N)=O(N^2)$；同理，全连接层间计算复杂度、全连接层到输出层的计算复杂度均为$O(N^2)$. 综上，所提算法的计算复杂度为$O(N^2)$.

关于收敛性，正如许多已有的著作(如文献 [26])所提到的，深度强化学习算法的收敛性保证仍然是一个有待解决的问题. 尽管强化学习算法的收敛性可以被证明，但深度强化学习算法需要使用神经网络进行函数逼近(例如深度 Q 学习算法中 Q 值的逼近)，在这种情况下收敛性可能不再得到保证. 在第 5 节通过实验的方式对算法收敛性进行了评估.

5 实验仿真

本节通过仿真实验对提出的算法进行性能分析. 首先验证了算法的收敛性，其次从移动终端数量、边缘节点数量、时间阈值要求等多个方面对算法性能进行评估.

5.1 实验设置

考虑一个 70 个终端设备、8 个边缘节点和 1 个远端云服务器的网络场景. 其中终端设备的计算能力取值区间为 2~3 GHz，边缘节点的计算能力取值区间为 30~40 GHz，终端设备到边缘节点的传输速率取值区间为 10~15 Mbps，终端设备到远端云服务器传输速率的取值区间为 2.5~3.5 Mbps，计算任务传输数据量的取值区间为 1~2 Mb，每个计算任务需要的计算量取值区间为 0.5~1.5 Gigacycles，实验具体参数设

置如表 2 所示. 设置神经网络训练时批量大小为 16, 学习速率η=0.01, 折扣因子为γ=0.9, 并采用随机梯度下降(stochastic gradient descent, SGD)优化器进行模型的训练优化.

Table 2 Parameter Settings

表 2 参数设置

参数	取值
边缘节点数量	8
终端设备数量	70
计算任务的计算量/Gigacycles	[0.5,1.5]
计算任务的传输数量/Mbits	[1,2]
终端设备到边缘节点的传输速率/Mbps	[10,15]
终端设备到云服务器的传输速率/Mbps	[2.5,3.5]
计算任务允许的延迟阈值/s	0.6
终端设备的计算能力/GHz	[2,3]
边缘节点的计算能力/GHz	[30,40]
云服务器分配给终端设备的计算资源/GHz	[10,15]
神经网络批量大小	16
神经网络学习速率η	0.01
神经网络折扣因子γ	0.9

为了评估算法性能, 本文将所提算法与其他 2 种计算卸载算法进行对比.

1)NOCO(non-cooperative algorithm). 利用文献 [9] 所提出的计算卸载策略, 每个终端设备在将计算任务卸载到边缘节点时, 只能选择本地关联的特定边缘节点, 不能卸载到其他边缘节点, 而且边缘节点间不会进行任务卸载.

2)COOF(cooperative-offloading algorithm). 根据文献 [19] 提出的基于学习的计算卸载策略, 边缘节点间可以相互协作卸载, 在不需要完全卸载信息的情况下选择最优边缘节点. 但算法中将时间进行离散化会引入计算决策等待时延.

3) TCCO(time-continuous cooperative offloading algorithm). 本文提出的算法, 考虑了时间的连续性问题, 基于深度强化学习设计了多边缘节点协作的计算卸载算法.

5.2 算法收敛性

本文通过离线方式对神经网络进行训练, 并在不同神经网络超参数下对算法的收敛性进行了评估.

图 4 显示了算法在不同学习速率(learning rate)η下的收敛情况, 其中学习速率即每次迭代时向损失函数最小值移动的步长. 如图 4 所示, 学习速率不同, 模型的学习效果与收敛速度也各不相同. 当学习速率η=0.1 时, 模型会迅速逼近极值(当训练次数在 300 次左右), 但由于学习速率过大, 模型在最后收敛状态时也产生较大的波动, 不利于模型的收敛; 当学习速率η=0.001 时, 学习速率过小, 模型前期训练时收敛速度较慢, 训练次数达到 1 000 次左右时模型才逐渐收敛, 学习速率过小也不利于模型的收敛; 当学习速率η=0.01 时, 模型既可以较快速度的收敛, 而且收敛状态也比较稳定. 因此, 设置实验仿真中模型的学习速率η=0.01.

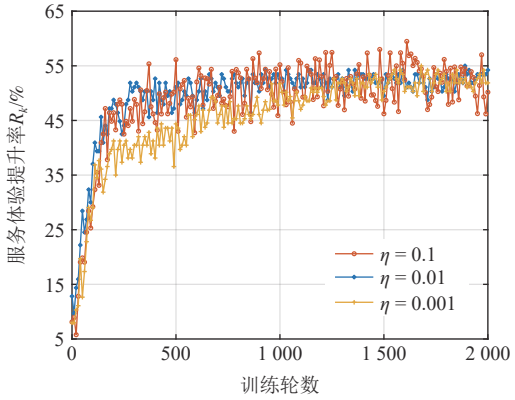

Fig. 4 Convergence of the proposed algorithm under different learning rates

图 4 所提算法在不同学习速率下的收敛性

图 5 显示了算法在不同批量大小(batch size)下的收敛情况, 其中批量大小即模型每一轮训练时选中的经验池中经验信息四元组的数量. 如图 5 所示, 当批量大小为 4 时, 模型收敛速度较慢, 需要经过 600 次左右训练才能收敛; 当批量大小从 4 增加到 16 时, 模型收敛速度明显加快, 在 400 次左右达到收敛; 当批量大小从 16 增加到 64 时, 模型收敛速度没有明显的增加. 而每一轮训练时间和计算量与批量大小成正相关, 即批量越大, 每一轮需要训练的经验信息

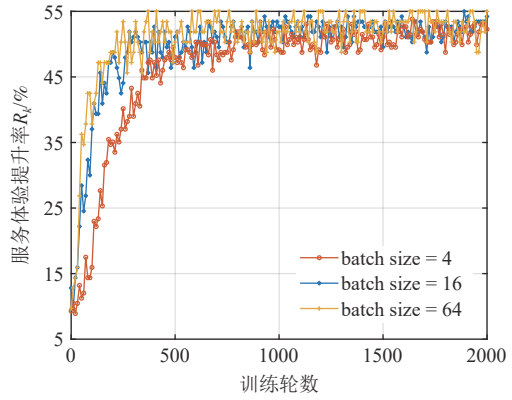

Fig. 5 Convergence of the proposed algorithm under different batch sizes

图 5 算法在不同批量大小下的收敛性

四元组的数量越多,计算量越大,训练时间也就越长.因此,在实验中选择批量大小为16,即可以减少每一轮的训练时间,又不会明显降低算法收敛速度.

5.3 算法性能对比

本节在不同实验参数下,通过与其他算法进行对比,对所提算法的服务提升率R_k进行分析.

本文对3种算法从多个性能指标进行了对比,结果如图6所示.4种性能指标的定义为:1)平均时延为所有计算任务的平均等待时延,即$\frac{1}{K}\sum_{k=1}^{K}T(k)$,其中$K$为计算任务总数;2)平均减少时延:与云计算相比,3种算法能够平均减少的任务时延,即$T(k)-(T^{c}(k)+u_k/r_{m_k,N+1})$;3)服务体验提升率$R_k$见式(14);4)任务超时率即未能在允许的延迟阈值内完成的计算任务比例.图6表明,由于没有决策等待时延,所提算法TCCO在平均减少时延和服务体验提升率方面明显优于其他2种算法,在任务超时率和平均时延方面也明显低于其他2种算法,算法整体性能优于其他2种算法.算法COOF由于考虑了边缘节点间的协作,能够有效利用边缘资源,从而各方面性能也都优于没有节点间协作的NOCO算法.

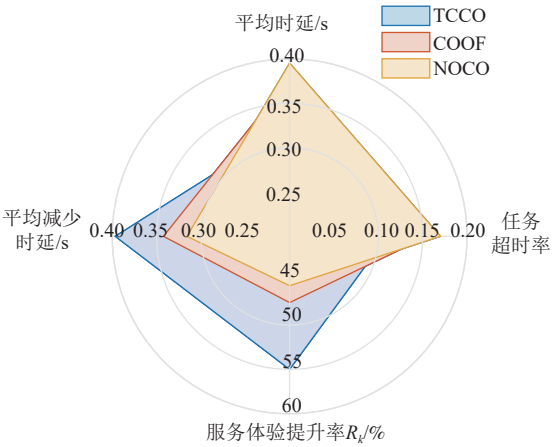

Fig. 6 Performance comparison of three algorithms

图6 3种算法的性能对比

图7显示了3种算法在不同计算任务允许的延迟阈值下的性能对比.整体而言,TCCO要优于其他2种对比方法,这是因为在TCCO中并没有将实验时间进行离散化,当计算任务产生时,TCCO会立刻进行计算卸载决策,没有等待时延;而其他2种算法将时间进行离散,需要等到每个时刻结束或开始时才进行卸载决策,且在本实验中,每个时刻的间隔设置为0.1 s,因此TCCO在性能方面有较大提升.NOCO性能表现最差,这是因为NOCO并没有考虑各个边缘节点间的协同,终端设备只能将计算任务卸载到某个特定的边缘节点,从而导致算法性能较低.有趣的是,随着计算任务允许的延迟阈值增加,服务提升率反而增加.这是由于在实验时没有引入超过阈值就将计算任务进行丢弃的机制,即在不同阈值情况下,计算任务的总量是相同的,而随着延迟阈值的增加,约束条件放宽,算法就具有更多的调度方案,整体性能上也就会有所提升,但该提升会逐渐趋于平缓.

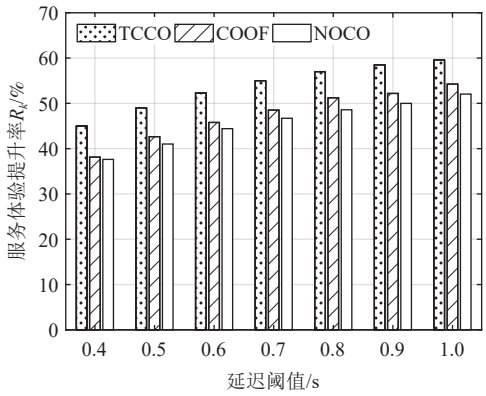

Fig. 7 Three agorithms performance under different delay thresholds

图7 3种算法在不同延迟阈值下的性能

图8描述了3种算法在边缘节点数量不变,不同终端设备数量下的性能对比.终端设备数量从30逐渐增加到130,随着终端设备的增加,3种算法的服务提升率均逐渐减小.原因在于边缘节点数量保持不变,即边缘节点的计算能力没有增加,而终端设备数量的增加,虽然增加了端点处的计算能力,但也意味着计算任务数量的增加,此时受限于有限的边缘计算能力,更多的计算任务将卸载到远端云服务器,服务提升率也就逐渐降低.

本文测试了3种算法在终端设备数量不变,不同边缘节点数量情况下的性能变化,如图9所示.与终

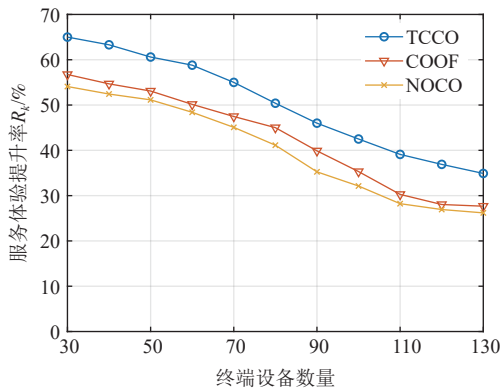

Fig. 8 Three algorithms performance under different number of devices

图8 3种算法在不同终端设备数量下的性能

端设备增加情况相反，随着边缘节点数量增加(边缘节点数量从 3 增加到 12)，3 种算法的服务提升率均有所提升. 原因在于终端设备数量不变，即计算任务量没有发生变化，而边缘节点数量增加使边缘处的计算能力增强，可以在边缘节点处理的计算任务数量也随之增多，卸载到远端云服务器的计算任务数量减少，服务提升率也就随之增加. 但由于边缘计算本身也会有传输时延和计算时延，因此服务提升率并不会随着边缘节点数量增加而一直增加，而是逐渐趋于平缓.

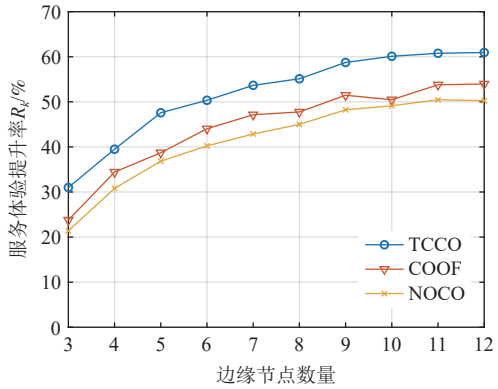

Fig. 9 Three algorithms performance under different number of edge nodes

图 9 3 种算法在不同边缘节点数量下的性能

此外，本文还对 3 种算法在终端设备、边缘节点、远端云服务器上的计算任务数量比例进行了分析，如图 10 所示. 3 种算法中，TCCO 在边缘节点中计算任务数量比例最高，COOF 次之，这 2 种算法在边缘节点完成计算任务数量比例明显高于远端云服务器，这是因为这 2 种算法均实现了边缘节点间的协同，对于边缘计算资源利用更加充分. 而 NOCO 由于没有考虑边缘节点间的协同，边缘资源利用率较低，相对而言更容易将计算任务卸载到远端云服务器，因此在远端云服务器执行的计算任务比例相对较高. 3 种算法中终端设备完成的计算任务比例均很低，原因在于终端设备上计算资源有限，本地计算时延开销大于进行计算卸载时带来的传输时延开销，因此大部分计算任务都将卸载到远端云服务器或边缘节点执行.

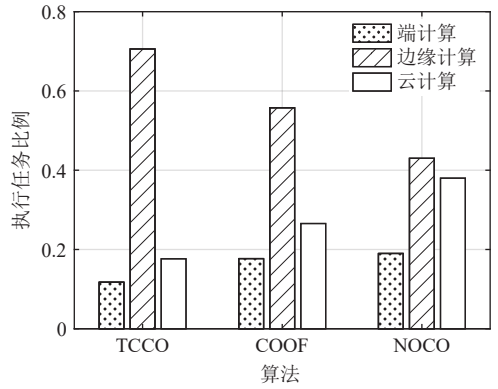

Fig. 10 Executed task ratios of cloud, edge and devices

图 10 在云、边、端上执行的任务比例

6 结 论

针对目前算力网络中计算卸载优化时算法可扩展性差、将时间轴离散化、只对单个边缘节点优化等诸多挑战，本文研究了面向时间连续的算力网络计算任务卸载问题，从计算任务本身而非时隙的角度进行模型构建，并通过系统状态、动作空间、奖励函数等要素定义了问题的马尔可夫决策过程. 提出了一种基于深度强化学习的计算卸载算法以确定计算任务在云、边、端上的执行位置. 仿真实验表明，所提 TCCO 算法在服务平均时延、服务提升率等方面比现有卸载算法具有更好的性能.

由于能耗是某些场景(如物联网)中重要影响因素，因此在未来工作中将在本文基础上加入能量消耗的影响. 同时考虑设备的安全性和隐私性问题，将更多的影响因素加入到算法中，以适应各种不同的网络场景和需求. 此外，用户移动性会导致传输路径和链路带宽发生变化，未来也将对此进行深入研究和详细建模，使得算法能够更好适应移动网络场景.

作者贡献声明：郝昊提出了算法思路并撰写论文；杨树杰设计了实验方案并负责完成实验；张玮提出指导意见并修改论文.

参 考 文 献

[1] HUAWEI TECHNOLOGIES CO.LTD.. Ubiquitous computing power: The cornerstone of intelligent society [EB/OL]. [2022-12-22]. http://www-file.huawei.com/media/corporate/pdf/publicolicy/ubiquitous_computing_power_the_cornerstone_intelligent_society_cn.pdf (in Chinese)
(华为技术有限公司. 泛在算力：智能社会的基石[EB/OL]. [2022-12-22]. http://www-file.huawei.com/media/corporate/pdf/publicolicy/ubiquitous_computing_power_the_cornerstone_intelligent_society_cn.pdf)

[2] Mao Yuyi, You Changsheng, Zhang Jun, et al. A survey on mobile edge computing: The communication perspective[J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322−2358

[3] Zheng Jianchao, Cai Yueming, Wu Yuan, et al. Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach[J]. IEEE Transactions on Mobile Computing, 2019, 18(4): 771−786

[4] Samanta A, Tang Jianhua. Dyme: Dynamic microservice scheduling in edge computing enabled IoT[J]. IEEE Internet of Things Journal, 2020, 7(7): 6164−6174

[5] Zhan Yufeng, Guo Song, Li Peng, et al. A deep reinforcement learning based offloading game in edge computing[J]. IEEE Transactions on Computers, 2020, 6(69): 883−893

[6] Hao Hao, Xu Changqiao, Zhong Lujie, et al. A multi-update deep reinforcement learning algorithm for edge computing service Offloading[C] //Proc of the 28th ACM Int Conf on Multimedia. New York: ACM, 2020: 1−9

[7] China Unicom. Computing power network whitepaper[R/OL]. [2022-12-22]. http://www.bomeimedia.com/China-unicom/white_paper/20191101-06.pdf (in Chinese)
(中国联通. 中国联通算力网络白皮书[R/OL] [2022-12-22]. http://www.bomeimedia.com/China-unicom/white_paper/20191101-06.pdf)

[8] China Mobile. Computing force network whitepaper, [R/OL]. [2022-12-22]. http://www.doc88.com/p-33073048121698.html (in Chinese)
(中国移动通信集团有限公司. 中国移动算力网络白皮书[R/OL]. [2022-12-22]. http://www.doc88.com/p-33073048121698.html)

[9] Guo Kai, Yang Mingcong, Zhang Yongbing, et al. Joint computation offloading and bandwidth assignment in cloud-assisted edge computing[J]. IEEE Transactions on Cloud Computing, 2022, 10(1): 451−460

[10] Guo Songtao, Liu Jiadi, Yang Yuanyuan, et al. Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing[J]. IEEE Transactions on Mobile Computing, 2019, 18(2): 319−333

[11] Chen Ying, Zhang Ning, Zhang Yongchao, et al. Energy efficient dynamic offloading in mobile edge computing for Internet of things[J]. IEEE Transactions on Cloud Computing, 2021, 9(3): 1050−1060

[12] Qu Guanjin, Wu Huaming, Li Ruidong, et al. DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing[J]. IEEE Transactions on Network and Service Management, 2021, 18(3): 3448−3459

[13] Zhang Qiuping, Sun Sheng, Liu Min, et al. Online joint optimization mechanism of task offloading and service caching for multi-edge device collaboration[J]. Journal of Computer Research and Development, 2021, 58(6): 1318−1339 (in Chinese)
(张秋平, 孙胜, 刘敏, 等. 面向多边缘设备协作的任务卸载和服务缓存在线联合优化机制[J]. 计算机研究与发展, 2021, 58(6): 1318−1339)

[14] Naouri A, Wu Hangxing, Nouri A N, et al. A novel framework for mobile-edge computing by optimizing task offloading[J]. IEEE Internet of Things Journal, 2021, 8(16): 13065−13076

[15] Li Qing, Wang Shangguang, Zhou Ao, et al. QoS driven task offloading with statistical guarantee in mobile edge computing[J]. IEEE Transactions on Mobile Computing, 2022, 21(1): 278−290

[16] Gao Xin, Huang Xi, Bian Simeng, et al. PORA: Predictive offloading and resource allocation in dynamic fog computing systems[J]. IEEE Internet of Things Journal, 2020, 7(1): 72−87

[17] Wang Lu, Zhang Jianhao, Wang Ting, et al. A fine-grained multi-access edge computing architecture for cloud-network integration[J]. Journal of Computer Research and Development, 2021, 58(6): 1275−1290 (in Chinese)
(王璐, 张健浩, 王廷, 等. 面向云网融合的细粒度多接入边缘计算架构[J]. 计算机研究与发展, 2021, 58(6): 1275−1290)

[18] Chen Menggang, Guo Songtao, Liu Kai, et al. Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing[J]. IEEE Transactions on Mobile Computing, 2021, 20(5): 2025−2040

[19] Dai Xingxia, Xiao Zhu, Jiang Hongbo, et al. Task co-offloading for D2D-assisted mobile edge computing in industrial Internet of things[J]. IEEE Transactions on Industrial Informatics, 2023, 19(1): 480−490

[20] Liu Zening, Li Kai, Wu Liantao, et al. CATS: Cost aware task scheduling in multi-tier computing networks[J]. Journal of Computer Research and Development, 2020, 57(9): 1810−1822 (in Chinese)
(刘泽宁, 李凯, 吴连涛, 等. 多层次算力网络中代价感知任务调度算法[J]. 计算机研究与发展, 2020, 57(9): 1810−1822)

[21] Hao Hao, Xu Changqiao, Yang Shujie, et al. Multicast-aware optimization for resource allocation with edge computing and caching[J]. Journal of Network and Computer Applications, 2021, 193(1): 1−13

[22] Tang Ming, Wong V W. Deep reinforcement learning for task offloading in mobile edge computing systems[J]. IEEE Transactions on Mobile Computing, 2022, 21(6): 1985−1997

[23] Huang Liang, Bi Suzhi, Zhang Yingjun. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks[J]. IEEE Transactions on Mobile Computing, 2020, 19(11): 2581−2593

[24] Tan Zhiyuan, Yu F R, Li Xi, et al. Virtual resource allocation for heterogeneous services in full duplex-enabled SCNs with mobile edge computing and caching[J]. IEEE Transactions on Vehicular Technology, 2018, 67(2): 1794−1808

[25] Hado H, Arthur G, David S. Deep reinforcement learning with double Q-learning[J]. arXiv: 1509.06461, 2015

[26] Hado H, Doron Y, Strub F, et al. Deep reinforcement learning and the deadly triad[J]. 2018, arXiv: 1812.02648v

Hao Hao, born in 1993. PhD, lecturer. Member of CCF. His main research interests include edge computing, content caching, and reinforcement learning. (haoh@sdas.org)

郝 昊, 1993 年生. 博士, 讲师. CCF 会员. 主要研究方向为边缘计算、内容缓存和强化学习.

Yang Shujie, born in 1989. PhD, master supervisor. Member of CCF. His main research interests include computing power network, edge computing, and reinforcement learning. (sjyang@bupt.edu.cn)

杨树杰, 1989 年生. 博士, 硕士生导师. CCF 会员. 主要研究方向为算力网络、边缘计算、强化学习.

Zhang Wei, born in 1983. PhD, professor. Member of CCF. His main research interests include fusion of computing and network, computing power network, and edge computing.

张　玮, 1983 年生. 博士, 教授. CCF 会员. 主要研究方向为算网融合, 算力网络和边缘计算.