文章编号:1007-5321(2019)02-0025-06 **DOI**:10.13190/j.jbupt.2018-078

基于机器学习的 MEC 随机任务迁移算法

孟 浩[1], 霍 如[1], 郭倩影[1], 黄 韬[1,2], 刘韵洁[1,2]

(1.北京工业大学 北京未来网络科技高精尖创新中心, 北京 100124;
2.北京邮电大学 网络与交换技术国家重点实验室, 北京 100876)

摘要: 针对移动边缘计算(MEC),提出了一种基于机器学习的随机任务迁移算法,通过将任务划分为可迁移组件和不可迁移组件,结合改进的 Q 学习和深度学习算法生成随机任务最优迁移策略,以最小化移动设备能耗与时延的加权和. 仿真结果表明,该算法的时延与能耗加权和与移动设备本地执行算法相比节约了 38.1%.

关 键 词: 移动边缘计算; 随机任务迁移; 机器学习; 时延; 移动设备能耗

中图分类号: TN929.53 **文献标志码**: A

Machine Learning-Based Stochastic Task Offloading Algorithm in Mobile-Edge Computing

MENG Hao[1], HUO Ru[1], GUO Qian-ying[1], HUANG Tao[1,2], LIU Yun-jie[1,2]

(1. Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology, Beijing 100124, China;
2. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: For mobile-edge computing (MEC), a machine learning-based stochastic task offloading algorithm was proposed. By dividing the task into offloadable components and unoffloadable components, the improved Q learning and deep learning algorithm were used to generate the optimal offloading strategy of stochastic task, which minimized the weighted sum of energy consumption and time delay of the mobile devices. The simulation results show that the proposed algorithm saves the weighted sum of energy consumption and time delay by 38.1%, compared to the local execution algorithm.

Key words: mobile-edge computing; stochastic task offloading; machine learning; delay; mobile device's energy consumption

移动设备随机任务的迁移算法在 MEC(mobile-edge computing)系统中扮演着重要的作用,决定着移动设备的计算效率和计算性能[1]. 近年来一些研究将一个确定的任务划分为一组有数据依赖关系的计算组件,通过一些优化策略将部分计算组件迁移到 MEC 服务器,加快移动设备任务的执行,同时节约能耗[2]. 然而,确定任务的优化策略不适用于真实多任务环境下任务随机到达时最优迁移策略生成的问题. 另一些研究考虑多个独立的随机任务调度迁移策略的优化,通过将到达但是尚未执行的任务

收稿日期: 2018-04-30
基金项目: 北京市科技新星计划项目(Z151100000315078); 国家科技重大专项项目(2018ZX03001019-003); 国家高技术研究发展计划(863 计划)项目(2015AA015702)
作者简介: 孟 浩(1991—), 男, 硕士生.
通信作者: 霍 如(1988—), 女, 讲师, E-mail:huoru@bjut.edu.cn.

暂存入缓存队列,利用马尔可夫决策过程(MDP, markov decision process)理论、lyapunov 优化技术或者博弈论理论等对系统建模,生成任务迁移策略来最小化移动设备能耗和执行时延[3-4]. 但是,在真实场景中,部分任务由于需要与用户交互或者访问本地 I/O 设备等原因无法迁移到 MEC 服务器,导致其执行效率无法达到预期. 笔者提出了一种基于机器学习的随机任务迁移算法,通过将确定任务划分为 N 个与系统调用无关的可迁移组件和 2 个需要进行系统调用的不可迁移组件,利用深度学习算法和强化学习中 Q 学习算法可以在真实环境下高效生成随机任务最优迁移策略. 该算法充分利用机器学习算法的学习能力和泛化能力,随着处理任务数量的增多,通过持续的训练神经网络,不断提升其产生策略的精度.

1 系统模型

考虑单用户 MEC 系统中移动设备随机多任务场景下的任务调度问题,单用户 MEC 系统包含 1 个移动设备和 1 个 MEC 服务器,如图 1 所示. 其中移动设备运行多个计算密集和时延敏感的独立应用,由 MEC 服务器辅助执行计算任务. MEC 服务器部署在移动接入网络侧,通过无线信道与移动设备通信. 移动设备包含 2 个逻辑执行单元:本地中央处理器(CPU, central processing unit)和传输单元,设备可以通过一定的策略将应用的一部分计算任务置于本地执行,一部分发送到 MEC 服务器执行,加快任务执行速度的同时节约移动设备的电池电量.

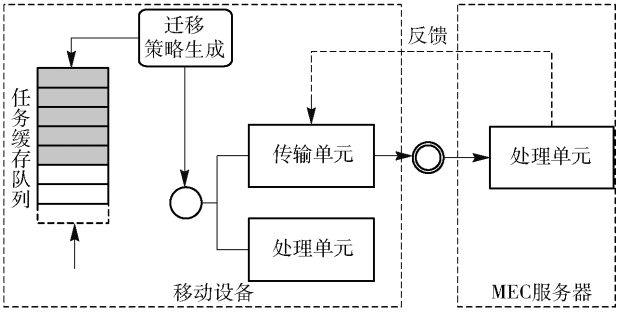

图 1 单用户 MEC 系统模型

1.1 任务模型

假设移动设备有 M 个独立的计算任务在缓存队列中等待处理,将这个队列用符号表示为 $Q=\{T_1,T_2,\cdots,T_i,\cdots,T_M\}$,其中每一个计算任务分为 2 个不可迁移组件和 N 个可迁移组件,表示为 $T_i=\{C_0,C_1,\cdots,C_j,\cdots,C_N,C_{N+1};u_j\}$,其中:$T_i$ 为第 i 个计算任务,C_0 与 C_{N+1} 为不可迁移组件,$C_1\sim C_N$ 为可迁移组件,u_j 为该任务执行 1 bit 数据需要的 CPU 转数[4]. 不可迁移组件主要由于任务开始执行和执行结束后需要调度系统驱动或显示设备等原因必须在设备端执行,如果有多个不可迁移组件,可将其合并为 2 个部分. 对于可迁移组件,设定其组件数为 N,便于对任务建模. 如果一个任务可迁移组件数小于 N,可以将多余组件数据置 0;如果一个任务可迁移组件数大于 N,可合并相关组件使组件数等于 N. 任务的第 j 个组件表示为 $C_j=\{n_j,w_j\}$,其中:n_j 为该组件的总输入数据量;$w_j=\{0,1\}$ 为任务执行指示器,$w_j=0$ 表示任务在移动设备 CPU 执行,$w_j=1$ 表示任务迁移到 MEC 服务器执行. 对于一个确定的计算任务,将任务分为 $N+2$ 个组件的同时必须考虑各个组件之间的数据相关性,为便于建模,构建确定任务组件的数据依赖关系为平行依赖关系[5],如图 2 所示,即 2 个不可迁移组件在移动端执行,其他可迁移组件选择性地在移动端执行或迁移到 MEC 服务器端执行,在之后的研究中会扩展到一般依赖关系的任务建模.

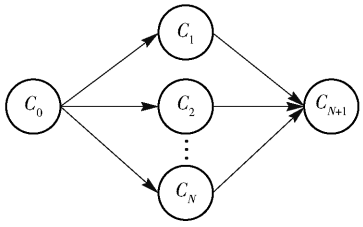

图 2 平行依赖关系

1.2 计算模型

假设移动设备配备单核 CPU 和一个数据传输天线,在执行一个任务时,CPU 执行频率为 f_{loc},并且其功率为 P_{loc}. 移动设备传输单元一次传输一个任务组件,其无线发射功率为 P_{tx},根据香农公式,信道最大数据传输速率[3]可以表示为

$$R(p_{\mathrm{tx}})=\omega\mathrm{lb}\left(1+\frac{g_0\,(L_0/L)^{\theta}P_{\mathrm{tx}}}{N_0\omega}\right) \tag{1}$$

其中:ω 为系统信道有效带宽,L_0 为参考距离,g_0 为参考距离损失常量,θ 为路径衰减指数,L 为移动设备与 MEC 服务器的距离,N_0 为信道高斯噪声功率.

假设 MEC 服务器的 CPU 频率为 f_{ser},数据处理和传输均需排队,计算资源充足,可同时处理多个任

务组件而无需排队，不需要考虑 MEC 服务器上的等待时延，并且假定计算结果很小，可以忽略反馈时延. 因此，任务 T_i 的可迁移组件完成时延 $t(m_t)$ 可以表示为

$$t(m_t)=\max\begin{cases}\sum_{j=1}^{N}\dfrac{(1-w_j)n_ju_j}{f_{\text{loc}}}, & \text{LCU 为 0}\\ \sum_{j=1}^{N}w_j\left(\dfrac{n_ju_j}{f_{\text{ser}}}+\dfrac{n_j}{R(p_{\text{tx}})}\right), & \text{LCU 为 1}\end{cases}\tag{2}$$

其中：m_t 表示任务可迁移组件当前所处的状态；逻辑计算单元(LCU, logic computing unit)为 0 表示移动设备计算单元，其时延为移动设备计算单元的计算时延；LCU 为 1 表示 MEC 服务器计算单元，其时延为任务组件传输时延与 MEC 服务器计算时延之和. 在处理任务 T_i 时，2 个 LCU 并行计算，故取时延较大者作为 S_t 状态下任务编号为 1 – N 组件的整体计算时延 $t(m_t)$. 任务 T_i 整体计算时延表示为

$$t(S_t)=\frac{n_0u_0}{f_{\text{loc}}}+t(m_t)+\frac{n_{N+1}u_{N+1}}{f_{\text{loc}}}\tag{3}$$

其中 S_t 表示任务 T_i 所有组件当前所处状态.

相应地，移动设备执行单个任务的能耗表示为

$$e(m_t)=\text{sum}\begin{cases}\sum_{j=1}^{N}\dfrac{(1-w_j)n_ju_jp_{\text{loc}}}{f_{\text{loc}}}, & \text{LCU 为 0}\\ \sum_{j=1}^{N}\dfrac{w_jn_jp_{\text{tx}}}{R(P_{\text{tx}})}, & \text{LCU 为 1}\end{cases}\tag{4}$$

其中：LCU 为 0 表示移动设备计算单元，其能耗为移动设备计算单元执行时的能耗；LCU 为 1 表示 MEC 服务器计算单元，笔者仅关注移动设备自身的能耗，故其能耗仅包含移动设备的传输能耗. 单个任务的整体能耗为两部分能耗之和，所以任务 T_i 的整体能耗可表示为

$$e(S_t)=\frac{n_0u_0p_{\text{loc}}}{f_{\text{loc}}}+e(m_t)+\frac{n_{N+1}u_{N+1}p_{\text{loc}}}{f_{\text{loc}}}\tag{5}$$

2　随机任务迁移算法

2.1　Q 学习算法

Q 学习是一种基于随机动态过程的不依赖模型的强化学习方法[6]. 在 Q 学习中，一个智能体通过 MDP 与环境进行交互，并从环境中获得延迟奖励来学习最优的控制策略[7]. 在智能体与环境交互的过程中，智能体需要观察到环境在时刻 t 的状态 S_t，选择一个动作(随机或者通过 Q 表选择)执行，获得下一个时刻状态 S_{t+1} 和该动作执行的即时奖励 r_t，通过式(6)更新 Q 表，并且将 S_{t+1} 设置为当前状态 S_t，不断循环这个过程直到 Q 表可以被近似为 Q^* 函数，Q 值的更新公式为

$$Q_{t+1}(s_t,a_t)=Q_t(S_t,a_t)+\eta[r_t+\gamma\max Q_t(S_{t+1},a_{t+1})-Q_t(S_t,a_t)]\tag{6}$$

其中：η 为学习率，控制 Q 值更新的速度；γ 为折扣因子，表示学习系统的远视程度. 当 Q 表收敛时，$Q_t(S_t,a_t)$ 的值等于在状态 S_t 执行 a_t 所获得的奖励，即 $r_t+\gamma\max Q_t(S_{t+1},a_{t+1})$. 因此，一旦 Q^* 的值被学习到，最优策略也随之确定.

在本文中，确定任务的 MDP 模型表示为

1) 状态空间

定义确定任务 T_i 在 t 时刻到达时的状态空间 S_t 包括 MEC 系统 2 个逻辑计算单元当前的忙闲状态 S_c 和任务的 N 个可迁移组件的分配状态 S_i，即任务迁移策略的一个可行解. 状态 S_c 中 c 的取值集合为 {0, 1}，分别表示移动设备计算单元和移动设备传输单元与 MEC 服务器组成的逻辑计算单元的编号，故 $S_c=\{s_0,s_1\}$，其中 $S_c=\{0,1\}$ 表示当前编号为 c 的逻辑计算单元的状态为{空闲, 忙碌}；用 w_j 表示任务 T_i 第 j 个组件逻辑计算单元的分配状态，所以 $S_i=\{w_1,w_2,\cdots,w_N\}$. 因此，状态空间可以用 S_c 和 S_i 表示为

$$S_t=[S_c,S_i]=[s_0,s_1,w_1,w_2,\cdots,w_N]\tag{7}$$

任务 T_i 的状态转移就是在策略解空间上的搜索.

2) 动作空间

定义动作空间 A 由任务 T_i 的 N 个可迁移组件 C_j 的执行位置表示，对于状态 S_t，动作空间中动作数为 N，第 j 个动作($1\leqslant j\leqslant N$)表示把第 j 个组件迁移到下一个逻辑处理单元中. 因此，动作空间可以表示为

$$A=\{a\mid a\in\{1,2,\cdots,N\}\}\tag{8}$$

3) 即时回报

考虑到任务 T_i 的学习过程就是在解空间上的搜索过程，搜索的目的是找到任务分配策略的最优解，任务在状态 S_t 时的处理时延为 $t(S_t)$，能耗为 $e(S_t)$，由于时延与能耗的计算结果可能不在同一量级，使用函数 $\text{sig}(x)=\dfrac{1}{1+e^x}$ 分别对时延与能耗进行

放缩,所以定义即时回报为

$$R(S_t,a_t)=\mathrm{sig}(t(S_{t+1}))+\mu\mathrm{sig}(e(S_{t+1}))-\mathrm{sig}(t(S_t))-\mu\mathrm{sig}(e(S_t)) \quad (9)$$

其中:μ 为权重,即在状态 S_t 下采取动作 a_t 所获得的即时回报,为下一个状态的时延与能耗加权和与当前状态时延能耗加权和的差. 当状态 S_{t+1} 的时延和能耗比状态 S_t 大时,$R(S_t,a_t)$ 的值为负数,即在状态 S_t 下采取动作 a_t 使状态变差,给予该动作负向惩罚;当状态 S_{t+1} 的时延和能耗比状态 S_t 小时,$R(S_t,a_t)$ 的值为正数,即在状态 S_t 下采取动作 a_t 使状态变优,给予该动作正向奖励.

4) 值函数 $Q(S_t,a_t)$ 的更新公式由式(6)确定.

5) 终止条件:当处于 S_t 状态时,采取 N 个动作时转移到状态 S_{t+1} 时所获得的即时回报均为负,即当前状态为一个局部最优解时当前情节终止.

根据以上描述,Q 学习算法包括以下步骤.

步骤 1 初始化 $Q(S_t,a_t)$,设置情节数 $n=0$ 和情节设定值 N 和贪婪策略上限值 epi;

步骤 2 随机初始化状态 S,并使其满足组件 C_j 同一时间只分配给一个逻辑处理器的原则,步骤数 $\mathrm{step}=1$;

步骤 3 计算状态 S_t 是否满足终止条件,如果满足终止条件,返回步骤 2;

步骤 4 根据贪婪策略,从动作空间 A 中选取当前状态 S_t 的值函数 $Q(S_t,a_t)$ 最大的动作 a_p,若 $Q(S_t,a_t)$ 为最大的数量超过 2,则随机从对应的几个动作中选取一个 a_t 作为 a_p;

步骤 5 产生[0,1]之间的随机数 ε,如果 $\varepsilon<\min(\mathrm{epi},(1+\mathrm{step})/(10+\mathrm{step}))$,则 $a=a_p$,反之从有效动作空间 A 中随机选取一动作 a_r,使 $a=a_r$;

步骤 6 执行动作 a,进入下一状态 S_{t+1},获得即时奖励 r;

步骤 7 由式(6)更新 $Q(S_t,a_t)$;

步骤 8 $S_t=S_{t+1}$,$\mathrm{step}=\mathrm{step}+1$,如果 S_t 不满足终止条件,转步骤 4,否则 S_t 为当前情节的终态,当前情节结束,并令 $n=n+1$;若情节数 n 达到设定值 N,算法结束;否则,返回步骤 2 继续执行.

2.2 深度前馈神经网络

采用深度全连接前馈神经网络构造多任务部分迁移策略生成器,其输入维度为 N,代表确定任务的 N 个可迁移组件数据量;由于每一个组件均有可能在移动设备 CPU 执行或者迁移到 MEC 服务器执行,所以任务分配过程中的总策略数为 2^N,输出维度为 2^N,其激活函数使用“sigmoid”;隐藏层数为 H,其激活函数使用“ReLU”. 深度神经网络的策略生成过程本质上是多分类的过程,将任务的 N 个可迁移组件数据经过正规化后输入神经网络,经过前向传播得到输出结果并利用 softmax 算法进行概率转换得到每一个分类的概率,将得到的结果使用交叉熵损失函数计算代价,为避免过拟合,使用 L2 正则化损失函数并使用反向传播算法更新神经网络参数,不断训练,使神经网络产生较高的准确率.

2.3 基于机器学习的随机任务迁移算法

该算法是前述算法模块的整合与使用,可分为训练阶段和使用阶段. 如图 3 所示,在训练阶段,算法的执行流程为对每一个随机任务,根据前述规则划分为 N 个可迁移组件和 2 个不可迁移组件,利用 Q 学习算法得到任务的最优执行策略,并将其存入数据库作为训练样本,当数据足够多时,用来训练深度前馈神经网络,以数据驱动的方式不断地循环训练算法得到近似最优迁移策略. 在算法使用阶段,可以跳过 Q 学习寻找单任务最优策略的过程,直接将任务的各个组件参数输入神经网络,经过网络的一次前向传播即可得到随机任务的近似最优迁移策略,使得算法可以高效地生成随机任务的近似最优迁移策略.

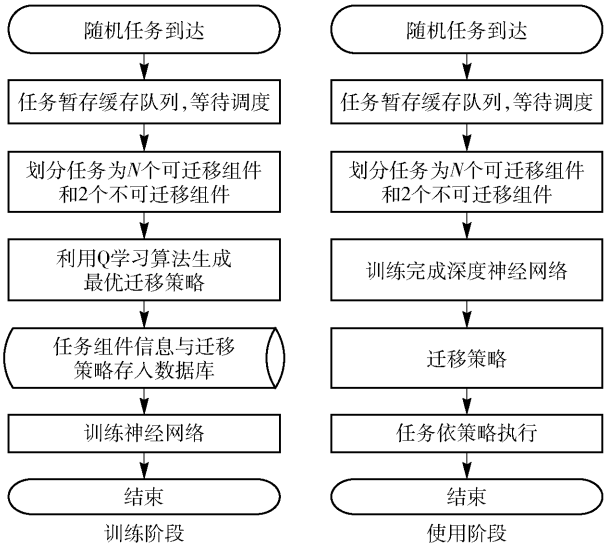

图 3 迁移算法训练阶段和使用阶段流程图

3 仿真结果

对提出的算法在 python 环境中利用 tensorflow

框架进行了编码实现，并且通过仿真实验对其性能进行了评估. 在仿真中，假定深度神经网络中隐藏层数 $H=2$，分别包含 1 024 个和 512 个神经元. 确定任务的子模块数 $N=9$，任务子模块数据大小和任务的负载量服从均匀分布，即 $c_i \sim \mathrm{Unif}([0,2c_{\mathrm{avg}}])$，$u_i \sim \mathrm{Unif}([0,2u_{\mathrm{avg}}])$，其中：$c_{\mathrm{avg}}=10$ kbit；$u_{\mathrm{avg}}=797.5$ cycles/bit. 此外，设置移动设备 CPU 频率 $f_{\mathrm{loc}}=1$ GHz，其功率 $p_{\mathrm{loc}}=320$ mW，MEC 服务器的 CPU 频率 $f_{\mathrm{ser}}=3$ GHz，对于移动设备传输模块，其参数设置为 $g_0=-40$ dB，$L_0=1$ m，$L=100$ m，$\theta=4$，$\omega=3$ MHz，$N_0=-174$ dBm/Hz，$P_{\mathrm{tx}}=100$ mW[4].

为验证算法的有效性和泛化能力，引入 3 种任务调度算法作为对比，包括随机任务随机迁移算法、基于 Johnson 的最优任务迁移算法[3]和移动设备本地执行算法. 随机生成算法为对所有随机到达的任务，除了不可迁移组件在移动设备执行外，对所有可迁移组件随机分配到移动设备 CPU 和 MEC 服务器执行；基于 Johnson 的最优任务迁移算法对所有随机到达的任务，按照传输时间和执行时间进行任务重新排序，按照排序结果将任务依次迁移到 MEC 服务器执行；移动设备本地执行算法为将所有任务的所有子模块均在移动设备 CPU 执行.

在包含 1 万个任务的测试集上使用基于机器学习的 MEC 随机任务迁移算法与前述算法在时延能耗加权和上的性能，如图 4 ~ 图 6 所示. 可以看到，随着任务数量的不断增加，移动设备任务完成时延、能耗、时延能耗加权和与任务数量呈类线性增长，但是，不同任务分配策略对时延与能耗加权和的影响也越来越大. 在图 4 中，所提出的基于机器学习的随机任务迁移算法时延最小，为 534.5，比本地执行算法(876.0)时延降低约 39%.

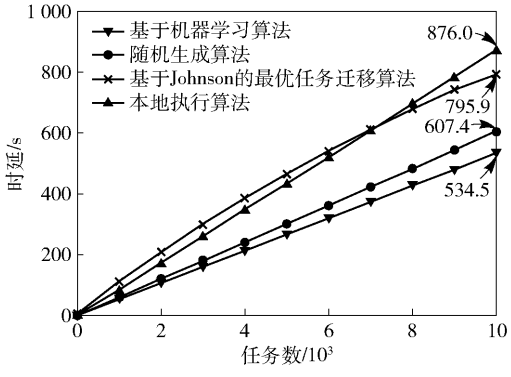

图 4　测试集上的任务完成时延与任务数

图 5 展示了 4 种不同算法在能耗上的表现，可以看到，除了基于 Johnson 的最优任务迁移算法外，所提出的算法能耗最低. 在图 6 中，处理完全部 1 万个任务时，本地执行算法时延与能耗加权和高达 1 156.3，基于 Johnson 的最优任务迁移算法略低，为 886.4，随机生成算法比基于 Johnson 的最优任务迁移算法低 94.3，为 792.1，基于机器学习的任务迁移算法时延与能耗加权和最低，为 715.7，大约比本地执行策略降低 38.1%，比最优任务迁移 Johnson 算法降低 19.3%，比随机任务随机生成策略降低 9.6%.

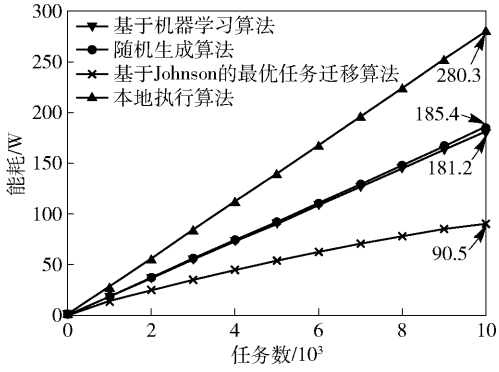

图 5　测试集上的移动设备能耗与任务数

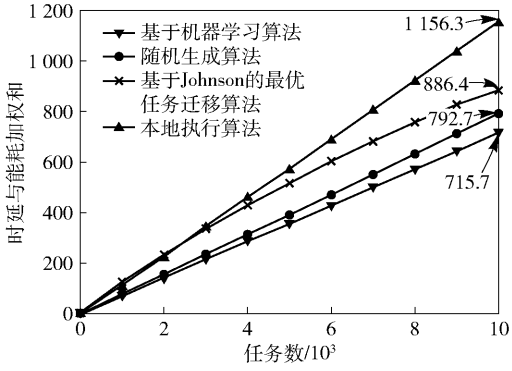

图 6　测试集上的时延与设备能耗加权和与任务数

4　结束语

提出了在单用户 MEC 系统中基于机器学习的随机任务迁移算法，通过将随机任务切分成可迁移组件和不可迁移组件，在算法训练阶段，利用 Q 学习生成该任务的最优迁移策略作为训练样本来训练深度前馈神经网络，完成最优策略的学习. 在算法使用阶段，新的随机任务到达时仅经过一次神经网络前向传播就可以生成该任务的近似最优分配策略，极大地加速了最优策略的生成. 仿真结果表明，

该算法在经过大量任务的训练后非常有效,并且在新任务到达时也能够生成近似最优策略,表现出了很好的泛化能力. 未来的研究工作主要包括加入对任务缓存队列优先级的控制、对移动设备无线信道传输功率的控制和将任务组件间数据依赖关系的处理由平行依赖扩展为一般性依赖.

参考文献:

[1] Barbarossa S, Sardellitti S, Lorenzo P D. Communicating while computing: distributed mobile cloud computing over 5G heterogeneous networks[J]. IEEE Signal Processing Magazine, 2014, 31(6): 45-55.

[2] Huang D, Wang P, Niyato D. A dynamic offloading algorithm for mobile computing[J]. IEEE Transactions on Wireless Communications, 2012, 11(6): 1991-1995.

[3] Mao Y, Zhang J, Letaief K B. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems[C]// Wireless Communications and Networking Conference. [S.l.]: IEEE, 2017: 1-6.

[4] Liu J, Mao Y, Zhang J, et al. Delay-optimal computation task scheduling for mobile-edge computing systems[C]// IEEE International Symposium on Information Theory. [S.l.]: IEEE, 2016: 1451-1455.

[5] Mao Y, You C, Zhang J, et al. A survey on mobile edge computing: The communication perspective[J]. IEEE Communications Surveys and Tutorials, 2017, 19(4): 2322-2358.

[6] Christopher J C H, Watkins, Peter D. Q-learning[J]. Machine Learning, 1992(8): 279-292.

[7] 刘晓平, 杜琳, 石慧. 基于Q学习的任务调度问题的改进研究[J]. 图学学报, 2012, 33(3): 11-16.
Liu Xiaoping, Du Lin, Shi Hui, et al. Improvement of task scheduling based on Q-learning[J]. Journal of Graphics, 2012, 33(3): 11-16.