计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220779
60(12): 2770−2782, 2023

一种联合时延和能耗的依赖性任务卸载方法

张俊娜[1,2] 鲍 想[1] 陈家伟[1] 赵晓焱[1] 袁培燕[1] 王尚广[3]
[1](河南师范大学计算机与信息工程学院 河南新乡 453007)
[2](智慧商务与物联网技术河南省工程实验室(河南师范大学) 河南新乡 453007)
[3](网络与交换技术国家重点实验室(北京邮电大学) 北京 100876)
(jnzhang@htu.edu.cn)

A Dependent Task Offloading Method for Joint Time Delay and Energy Consumption

Zhang Junna[1,2], Bao Xiang[1], Chen Jiawei[1], Zhao Xiaoyan[1], Yuan Peiyan[1], and Wang Shangguang[3]
[1](*School of Computer and Information Engineering*, *Henan Normal University*, *Xinxiang*, *Henan* 453007)
[2](*Henan Provincial Engineering Laboratory of Smart Commerce and Internet of Things Technology* (*Henan Normal University*), *Xinxiang*, *Henan* 453007)
[3](*State Key Laboratory of Networking and Switching Technology* (*Beijing University of Posts and Telecommunications*), *Beijing* 100876)

Abstract Edge computing deploys computing and storage resources on the edge of the network closed to users, so that users can offload high-latency and energy-intensive applications to the edge of the network for execution to reduce application latency and local energy consumption. Existing offloading research usually assumes that the offloaded tasks are independent of each other, and the edge server caches all the services required for task execution. However, in real scenarios, there are often dependent between tasks, and edge servers can only cache limited services due to their limited storage resources. To this end, we propose a dependent task offloading method that balances latency and energy consumption (i.e., cost) under the constraints of limited computing resources and service caches on edge servers. First, the constraints in the research problem are relaxed to be transformed into a convex optimization problem. A convex optimization tool is used to find the optimal solution, which is used to calculate the priority of offloading tasks. Then, the tasks are offloaded to the edge server with the least cost according to the priority. If multiple dependent tasks are offloaded to different edge servers, an improved particle swarm optimization is used to solve the optimal transmission power of edge servers to minimize the total cost. Finally, sufficient experiments are performed based on real datasets to verify the effectiveness of the proposed method. The experimental results show that the proposed method can reduce the total cost by approximately 8% to 23% compared with other methods.

Key words edge computing; task offloading; task dependency; service cache; improved particle swarm optimization

摘 要 边缘计算通过在靠近用户的网络边缘侧部署计算和存储资源,使用户可将高延迟、高耗能应用程序卸载到网络边缘侧执行,从而降低应用延迟和本地能耗.已有的卸载研究通常假设卸载的任务之间相互独立,且边缘服务器缓存有执行任务所需的所有服务.然而,在真实场景中,任务之间往往存在依赖关系,且边缘服务器因其有限的存储资源只能缓存有限的服务.为此,提出一种在边缘服务器计算资源和服务缓存有限的约束下,权衡时延和能耗(即成本)的依赖性任务卸载方法.首先,松弛研究问题中的约束将其转换为凸优化问题;采用凸优化工具求最优解,并用解计算卸载任务的优先级.然后,按照优先级将

收稿日期: 2022-09-01; **修回日期**: 2023-01-30
基金项目: 国家自然科学基金项目(61902112, 62072159); 河南省科技攻关项目(222102210011)
This work was supported by the National Natural Science Foundation of China (61902112, 62072159) and the Science and Technology Development Foundation of Henan Province (222102210011).
通信作者: 赵晓焱(121095@htu.edu.cn)

任务卸载到成本最小的边缘服务器，若多个依赖任务卸载到不同的边缘服务器，为了使总成本最小，则采用改进粒子群算法求解边缘服务器的最佳传输功率. 最后，为了验证所提方法的有效性，基于真实数据集进行了充分的实验. 实验结果表明，所提方法与其他方法相比能够降低总成本 8%~23%.

关键词 边缘计算；任务卸载；依赖性任务；服务缓存；改进粒子群优化

中图法分类号 TP393

随着物联网的快速发展和万物互联时代的到来，智能移动设备在人们的日常生活中越来越普及. 同时，诸如虚拟现实、人脸识别、图像处理等[1]低时延、高能耗的应用程序得到了广泛应用. 但移动设备受电池容量、计算和存储能力等诸多限制，直接运行上述应用程序会带来较高的延迟和能耗，用户体验极差[2]. 为了提升用户体验，边缘计算（edge computing, EC）应运而生，其在距离移动设备最近的边缘网络提供计算和存储资源，目的是减少延迟来确保高效的网络操作和服务交付[3]. 通过将移动设备中部分或全部计算任务卸载到边缘服务器处理，可以解决移动设备在资源存储、计算能力以及电池容量等方面存在的不足，并降低应用的延迟.

近年来，许多学者针对任务卸载进行了研究[4-7]，然而这些研究也存在一些局限性：1）只考虑单一优化目标. 即只优化卸载任务的完成时间或能耗. 如文献 [8] 只考虑卸载任务的完成时间，无法满足用户对能耗的需求.2）假定卸载的多个任务相互独立. 现实中很多应用程序包含的任务往往存在依赖关系. 例如，人脸识别应用程序包含特征提取和特征分类任务，且“特征提取”任务的输出是“特征分类”任务的输入. 因此，只有“特征提取”任务成功执行，才能开始执行“特征分类”任务.3）默认边缘服务器缓存有卸载任务所需的所有服务. 文献 [9−10] 均研究依赖性任务的卸载问题，但均假设边缘服务器缓存有卸载在其上的所有任务所需的服务. 现实生活中，任务的执行往往需要特定执行环境. 例如，人脸识别应用程序中“特征提取”任务，只能被卸载到配置有相应机器学习服务的边缘服务器才能被成功执行. 若边缘服务器上没有配置该服务，将导致卸载任务无法执行. 而边缘服务器存储资源有限，只能缓存有限服务，因此卸载任务时需考虑边缘服务器缓存服务约束.4）假定边缘服务器具有无限存储和计算资源. 文献 [11−12] 均不考虑边缘服务器的存储和计算资源，而现实中边缘服务器提供的资源是有限的，卸载任务时必须考虑资源受限问题，否则会出现任务卸载不成功的风险.

综上所述，已有任务卸载研究往往忽略任务间的依赖关系，默认边缘服务器缓存有任务所需的所有服务和具有无限资源，且大多只优化单一用户需求. 为了解决上述局限性，本文研究在边缘服务器计算资源和服务缓存有限的情形下，权衡时延和能耗的依赖任务卸载问题，并提出了联合时延和能耗的依赖性任务卸载方法（dependent task offloading method for joint time delay and energy consumption, DTO-TE）. 首先，放松优化函数中的二进制变量为连续变量，并修改相关约束，将问题转换成凸优化问题进行求解. 其次，将得到的小数解作为卸载概率进行迭代运算使其恢复成二进制解，作为应用程序中每个任务的可行卸载决策，并根据该决策计算任务的卸载优先级. 最后，按照任务优先级从高到低依次将任务卸载到成本最小的边缘服务器. 若多个依赖任务卸载到不同的边缘服务器，则采用改进粒子群优化方法（particle swarm optimization, PSO）求解边缘服务器的最佳传输功率，从而获得最小的传输时延和传输能耗.

本文的主要贡献有 4 点：

1）在边缘服务器计算资源和服务缓存有限的约束下，建立了任务依赖模型、网络模型、完成时间模型和能耗模型，并采用线性加权的方法将对时延和能耗的共同优化转化为对成本优化的单目标优化问题.

2）针对非凸的 NP-hard 问题，通过松弛约束中的二进制变量将其转换成凸优化问题. 采用凸优化工具求得可行卸载决策，从而确定任务的最优卸载顺序.

3）若将多个依赖任务卸载到不同的边缘服务器，边缘服务器间因传输数据产生成本. 为了最小化总成本，采用改进的粒子群算法求解边缘服务器的最佳传输功率.

4）基于真实数据集，本文方法与已有的方法进行了充分的实验对比. 实验结果表明，本文方法比其他对比方法在总成本上减少 8%~23%.

1 相关工作

近年来，任务卸载作为 EC 关键技术之一受到了广泛的关注. 针对任务卸载问题，许多学者提出了解决方法或研究思路，并取得了较好的研究成果. 下面

将对一些与本文研究密切相关的成果予以分析.

文献 [13] 研究小蜂窝网络下单个边缘服务器的任务卸载问题，并提出了一种高效的低时延云边端协同卸载方案，该方案不仅可以降低任务的总处理时延，还可以处理大量的用户请求. 文献 [14] 在多基站的 EC 场景下，研究多用户任务卸载问题，该问题以最小化所有用户最大任务执行时间为目标，提出了一种启发式计算卸载方法. 文献 [8] 研究了兼顾边缘服务器服务缓存的依赖任务卸载问题，以最小化卸载任务的完成时间为目标，提出基于凸优化的依赖任务卸载方法. 该方法将每个任务卸载到完成时间最小的边缘服务器上，从而优化整个应用程序的完成时间. 文献 [15] 在超密集组网的 EC 场景下，联合优化卸载决策和资源分配问题，提出了基于坐标下降法的任务卸载方案. 然而，文献 [13−15] 只考虑单一优化目标，即只优化时延或能耗，无法满足用户对时延和能耗的共同需求.

文献 [16] 针对超密集网络下独立任务的卸载问题，提出一种基于双深度 Q 网络的在线卸载方法，通过特定的神经网络估计每个动作所获得的奖励值，获得最优的任务卸载策略以及最佳的 CPU 频率和发射功率，从而最小化任务的完成时间和终端能耗. 文献 [17] 研究在边缘服务器资源受限情形下的独立任务卸载问题，为了有效利用 EC 的计算资源和提高用户体验，以最小化所有移动设备的时延和能耗为目标，提出了一种高效的李亚普诺夫在线算法，获得了最优的任务卸载和数据缓存决策. 文献 [18] 研究在多个边缘服务器覆盖场景下的独立任务卸载问题. 该文献设计了一种概率优先的先验卸载模型，将问题转换成平衡时延和能耗的效用函数，提出基于遗传算法的联合卸载比例和传输功率的分布式先验卸载机制. 文献 [16−18] 均假定卸载的多个任务相互独立，然而现实应用程序中任务间往往存在依赖关系.

文献 [9] 研究依赖任务的调度决策问题. 在应用程序完成时间的约束下，将调度决策问题构建成以应用程序执行成本最小化为目标的优化函数，提出启发式多用户重分配卸载方法以获取最优的卸载决策，从而优化任务的完成时间. 文献 [10] 提出一种面向多用户的串行任务动态卸载策略，为所有任务做出近似最优的卸载策略，从而使平均完成时间和移动设备的平均能耗达到近似最优. 文献 [19] 研究在完成时间受到预定期限的约束下依赖任务的卸载问题，以最小化移动设备的能耗为目标，提出了一种高效的自适应卸载算法，确定哪些任务必须被卸载及卸载到哪个边缘服务器. 文献 [9−10,19] 虽然考虑了任务之间的依赖关系，但均假设边缘服务器上缓存有执行卸载任务的服务. 因边缘服务器的存储能力有限，往往只能缓存有限的服务.

文献 [20] 针对边缘服务器只能缓存有限服务的情形，研究依赖任务的卸载和调度问题. 为了降低所有卸载任务的完成时间，提出一种求解最优任务分配和高效调度的方法. 文献 [11] 研究具有时间约束的依赖任务卸载问题. 为了最小化应用程序完成时间，提出了基于贪婪调度的个体时间分配算法. 文献 [12] 研究在服务缓存受限的 EC 系统中依赖任务的调度问题. 为了解决任务对边缘服务器资源的竞争，将任务调度定义为以最小化时延为目标的优化问题，并在此基础上，提出了上下文感知的贪婪任务调度算法以获得最佳的调度决策. 文献 [11−12, 20] 均假设边缘服务器的计算资源是无限的，可执行任意数量的任务. 但现实中边缘服务器往往具有有限的资源.

为了解决上述局限性，本文研究在边缘服务器计算资源和服务缓存有限的情形下，权衡时延和能耗的依赖任务卸载问题，并提出了 DTO-TE 方法.

2 系统建模和问题描述

本节首先介绍边缘服务器间的网络模型和任务间的依赖模型；然后详细描述了任务卸载到边缘服务器执行的完成时间模型和能耗模型；最后描述了在边缘服务器计算资源和服务缓存有限的情形下，依赖性任务的卸载问题，并将其构建成以最小化总成本为目标的优化函数.

2.1 网络模型

本文建立的网络模型如图 1 所示，边缘网络由若干个基站组成，每个基站均部署 1 个 EC 服务器，不同 EC 服务器具有不同的处理和通信能力，且通过无

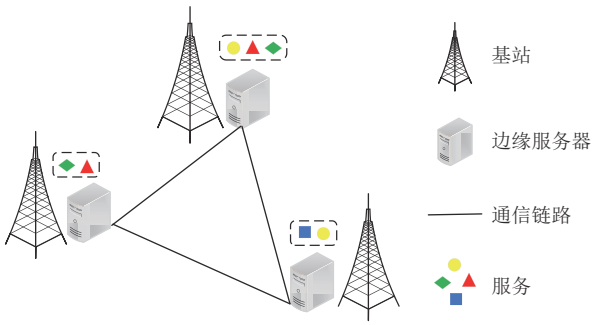


Fig. 1 Network model

图 1 网络模型

线方式相互通信. 虚线方框中内容表示边缘服务器缓存的服务，不同形状的图形表示不同的服务；实线表示边缘服务器之间通过局域网或蜂窝网连接.

集合$M=\{m_1,m_2,\cdots,m_l,\}$表示 EC 网络中的集，其中 l 表示边缘服务器的数量. 从边缘服务器 m 传输数据到边缘服务器m'的通信延迟为$t_{mm'}$. 当$m=m'$时，表示为同一服务器，此时通信时延为 0.每个边缘服务器最大限度地缓存服务，M_v 表示可以处理任务 v 的边缘服务器集，即任务 v 只能卸载到 M_v 中的边缘服务器上才能被执行. 此外，每个边缘服务器具有有限的计算资源，采用 CPU 周期表示. 边缘服务器 m 具有的计算资源为 $C(m)$. t_{vm} 表示任务 v 卸载到边缘服务器 m 上的执行时间，r_{vm} 表示边缘服务器 m 执行任务 v 每秒所需要的 CPU 周期.

2.2 任务依赖模型

本文研究将 1 个应用程序卸载到 EC 环境中. 假定应用程序能被划分成多个依赖任务，1 个任务仅能被卸载到 1 个边缘服务器，且任务执行需要计算资源和相应服务的支持. 如图 2 所示，假设应用程序由 4 个任务组成. 只有菱形代表的服务可为任务 2 提供执行环境，那么任务 2 只能被卸载到缓存有菱形代表的服务的边缘服务器上. 此外，任务 2 与任务 1 存在依赖关系，即任务 2 是任务 1 的后继任务，那么当且仅当任务 1 执行完毕，且执行结果被传输到任务 2 卸载的边缘服务器后，任务 2 才可以执行.

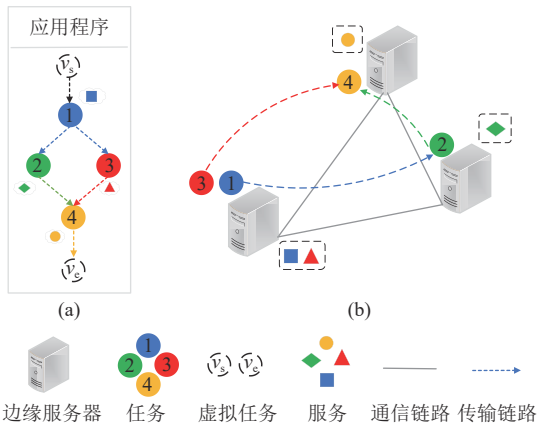


Fig. 2 Task dependent model

图 2 任务依赖模型

本文采用有向无环图(directed acyclic graph, DAG)，$G=(V, E)$表示任务之间的依赖关系，其中 V 表示任务集，E 表示边集. 为了便于描述，为应用程序构建 DAG 时增加 2 个虚拟任务 v_s 和 v_e. v_s 表示开始任务，没有前驱任务；v_e 表示结束任务，没有后继任务. 这 2 个任务无需被卸载，即在用户设备执行时所需时间为 0，那么应用(包含 n 个任务)DAG 的任务集为 $V=\{v_s,\ v_1,v_2,\cdots,v_n,v_e\}$. 有向边$(v,v')$表示任务$v$和$v'$之间存在依赖关系，即任务$v$的输出是任务$v'$的输入. 用$d_{vv'}$表示任务$v$到任务$v'$需传输的数据量.表 1 总结了在本文中使用的关键符号.

Table 1 Key Symbols Meaning

表 1 关键符号意义

符号	意义
V	任务集合
M	边缘服务器集合
$d_{vv'}$	依赖任务v和v'之间传输的数据量
$R_{mm'}$	边缘服务器m与边缘服务器m'的通信速率
M_v	可以处理任务v的边缘服务器集
$C(m)$	边缘服务器m的总 CPU 周期数
r_{vm}	边缘服务器 m 执行任务 v 每秒需要的 CPU 周期
z_v^m	任务v的卸载决策
$Pred(v)$	任务v的直接前驱任务集合
$Succ(v)$	任务v的直接后继任务集合
t_v	任务的开始执行时间
$t_{mm'}$	边缘服务器m到m'的传输时延
t_{vm}	边缘服务器m执行任务v所需时间
T	应用程序的完成时间
P_m	边缘服务器m的传输功率
$P_{\max}$	边缘服务器的最大传输功率
κ	边缘服务器的能量系数
e_{vm}	边缘服务器m执行任务v所需能耗
$e_{mm'}$	边缘服务器m到m'的传输能耗
E	应用程序的总能耗

2.3 完成时间模型

由 2.2 节可知，应用程序的完成时间从 v_s 开始执行，至 v_e 执行完毕.v_s 和 v_e 为虚拟节点，所需的执行时间为 0，那么 v_s 执行结束时刻即为应用程序开始时刻，v_e 开始执行时刻即为应用程序结束时刻. 当 v_e 的所有前驱任务均执行完毕，并把结果传回到用户设备时，用户设备可开始执行. 由此可知，应用程序的完成时间可由组成其的任务完成时间及任务间的依赖关系得出.

通过 2.2 节构造的应用程序的 DAG 可知，任务 v 能够执行时需满足 2 个条件：1)任务 v 所有前驱任务均执行完毕，且把执行结果传输至其被卸载的边缘服务器；2)卸载任务 v 的边缘服务器有计算资源执行任务 v.

依赖任务被卸载至相同边缘服务器，数据传输

时延为0；卸载至不同边缘服务器时，需要传输的数据量为$d_{v'v}$，则传输时延可表示为

$$t_{m'm}=\begin{cases}\dfrac{d_{v'v}}{R_{m'm}}(\forall\langle v',v\rangle\in E),\ m'\neq m,\\ 0,\qquad\qquad m'=m,\end{cases}\tag{1}$$

其中$R_{m'm}$是边缘服务器m'到边缘服务器m的传输速率，$t_{m'm}$表示m'与m之间的传输时延，可通过香农公式[10]计算得到：

$$R_{m'm}=B\mathrm{lb}\left(1+\frac{P_{m'}h_{m'm}}{N_0}\right),\tag{2}$$

其中B是信道带宽，$P_{m'}$是边缘服务器m'的传输功率，$h_{m'm}$是m'和m之间的信道增益，N_0表示噪声功率.

任务v可能由多个前驱组成，那么任务v接收到所有前驱任务执行结果的时刻为

$$D_v=\max_{v'\in Pred(v)}\{t_{v'}+t_{v'm'}+t_{m'm}\},\tag{3}$$

其中$Pred(v)$表示任务v的直接前驱任务集合，$t_{v'}$表示任务v'的开始执行时刻，$t_{v'm'}$表示任务v'在边缘服务器m'上所需执行时间.

此外，本文研究场景中，边缘服务器串行执行卸载至其上的任务. 当任务v卸载到边缘服务器m时，需要等待排在其前面的所有任务执行完才能被执行. 假如排在任务v之前的任务为k，用SL_m表示任务k被卸载的边缘服务器m执行完毕的时刻，那么有

$$SL_m=t_k+t_{km},\tag{4}$$

其中t_k表示任务k真正的开始时刻，t_{km}表示任务k在边缘服务器m上的执行时间.

因此，任务v在边缘服务器m上的开始时刻为

$$t_v=\max(SL_m,D_v).\tag{5}$$

通过上述分析，应用程序的完成时间可表示为

$$T=\max\left\{t_v+\sum_{m\in M}z_v^m t_{vm},v\in V\right\},\tag{6}$$

其中二进制变量z_v^m表示任务v是否卸载到边缘服务器m上，$z_v^m=1$表示是，$z_v^m=0$表示否. t_{vm}表示任务v在边缘服务器m上的执行时间.

2.4 能耗模型

执行应用程序的能耗包括在用户设备和在边缘服务器上产生的能耗. 本文只关注边缘服务器上产生的能耗、可能产生执行卸载任务的执行能耗，以及传输具有依赖关系任务之间的执行结果的传输能耗.

任务v在边缘服务器m的执行能耗e_{vm}可表示为

$$e_{vm}=\kappa t_{vm},\tag{7}$$

其中κ为边缘服务器的能量系数，表示边缘服务器的执行能力，κ与边缘服务器的具体参数有关. 文献[18]对κ的期望值进行了计算，κ与真实测量值较相似. 因此，本文对κ的取值为文献[18]计算的期望值.

假设2个依赖任务(v,v')分别被卸载到边缘服务器m'和边缘服务器m时，传输能耗可表示为

$$e_{m'm}=\begin{cases}P_{m'}t_{m'm},&m'\neq m,\\ 0,&m'=m,\end{cases}\tag{8}$$

其中$P_{m'}$表示边缘服务器m'的传输功率.

因此，边缘服务器的总能耗可以表示为

$$E=\sum_{v\in V}\left\{\sum_{m\in M}z_v^m\left(\sum_{m'\in M'}e_{m'm}+e_{vm}\right)\right\},\tag{9}$$

其中M'表示任务v所有前驱任务所在的边缘服务器集合.

2.5 研究问题描述

本文研究在边缘服务器计算资源和服务缓存有限的约束下，联合时延和能耗的依赖任务卸载问题. 引入时延和能耗的偏好权重变量$\omega(\omega\in[0,1])$，其取值由用户对时延和能耗的需求决定[21]. 如果对时延的需求超过能耗时，例如延迟敏感型应用人脸识别，用户更关注时延时，$\omega<0.5$；反之，例如执行应用的设备为电池容量有限的移动设备，用户更关注能耗时，$\omega\geqslant 0.5$. 因此，根据式(6)(9)，权衡时延和能耗的应用程序总成本可定义为

$$C=\omega E+(1-\omega)T.\tag{10}$$

综合以上分析，本文研究问题可以优化为

$$\begin{aligned}&\min\ C\\ &\text{s.t. }\mathrm{C}_1:z_v^m\in\{0,1\},&&\forall v\in V,m\in M;\\ &\quad\mathrm{C}_2:\sum_{m\in M_v}z_v^m=1,&&\forall v\in V;\\ &\quad\mathrm{C}_3:t_{v'}+t_{v'm'}+t_{m'm}\leqslant t_v,&&\forall\langle v',v\rangle\in E;\\ &\quad\mathrm{C}_4:t_v-t_{v'}\geqslant t_{v'm},&&\forall v,v'\in V,m\in M;\\ &\quad\mathrm{C}_5:t_{vm}r_{vm}\leqslant C(m),&&\forall m\in M;\\ &\quad\mathrm{C}_6:0<P_m\leqslant P_{\max},&&\forall m\in M;\end{aligned}\tag{11}$$

其中：C_1表示任务的卸载决策，取$z_v^m=1$表示任务v被卸载到边缘服务器m上，反之取$z_v^m=0$；C_2表示每个任务只能卸载到缓存有其所需服务的边缘服务器；C_3表示依赖任务之间的执行顺序；C_4表示同一边缘服务器上任务的执行顺序；C_5表示任务需要卸载到满足其计算资源的边缘服务器；C_6表示边缘服务器传输功率的取值范围.

文献[8]研究在边缘服务器计算资源和服务缓存有限的约束下，时延最优的依赖性任务卸载问题，并证明了研究问题为NP-hard问题. 与文献[8]研究问题相比，本文优化目标增加了能耗，因此，本文研究问题也属于NP-hard问题.

3 联合时延和能耗的依赖性任务卸载方法

为了求解本文研究的 NP-hard 问题，本节提出一种联合时延和能耗的依赖性任务卸载方法，该方法可分 3 步：1)松弛二进制变量. 松弛式(11)中 C1，将问题转换为凸优化问题，即转化 NP-hard 问题，并使用凸优化工具进行求解，为每个卸载任务获得多个介于 [0,1] 的松弛解；2)计算卸载任务的优先级. 选择每个任务松弛解中最大值作为其可行解，通过迭代运算确定每个任务的卸载优先级；3)完成卸载任务. 按照优先级从高到低把任务卸载到边缘服务器，若多个依赖任务可卸载到不同的边缘服务器，则采用改进的粒子群方法求解边缘服务器的最佳传输功率.

3.1 松弛变量和定义优先级

由式(11)中约束 C_1 可知，本文研究问题为非凸问题. 将 C_1 中二进制变量 z_v^m 松弛为 [0,1](即 C_7)，即假设每个卸载任务可被划分为若干个子任务，且可被卸载到不同的边缘服务器并行执行. 因此优化问题可转换为

$$\begin{aligned}
&\min C\\
\text{s.t.}\quad & C_7: z_v^m \in [0,1], && \forall v \in V, m \in M;\\
& C_2 \text{至} C_6\\
& C_2: \sum_{m\in M_v} z_v^m = 1, && \forall v \in V;\\
& C_3: t_{v'} + t_{v'm'} + t_{m'm} \leqslant t_v, && \forall \langle v', v\rangle \in E;\\
& C_4: t_v - t_{v'} \geqslant t_{v'm}, && \forall v, v' \in V, m \in M;\\
& C_5: t_{vm} r_{vm} \leqslant C(m), && \forall m \in M;\\
& C_6: 0 < P_m \leqslant P_{\max}, && \forall m \in M.
\end{aligned} \tag{12}$$

引理 1. 式(12)为凸优化问题.

证明. 由凸优化问题的判定准则可知，若式(12)同时满足 3 个条件时为凸优化问题：

1)目标函数为凸函数. 目标函数为多元变量相加的一次函数，因此其 Hesse 矩阵为零矩阵. 又因零矩阵为半正定矩阵，所以目标函数为凸函数.

2)不等式约束为凸函数. 式(12)中有多个不等式约束，但证明方法类似. 现以最复杂的不等式约束 C_3 为例进行证明. 令 $c(x) = t_{v'} + t_{v'm'} + t_{mm'} - t_v$，可以看出 $c(x)$ 为多元相加的一次函数，Hesse 矩阵为 0，因此约束 C_3 为凸函数. 同理，其他不等式约束也是凸函数.

3)等式约束为仿射函数. 将式(12)中约束 C_2 转换成 $\sum_{m\in M_v} z_v^m - 1 = 0$，令 $g(x) = \sum_{m\in M_v} x_m - 1$，可得出 $g(x)$ 为仿射函数.

综上所述，式(12)为凸优化问题. 证毕.

由于式(12)中传输功率是未知的，为了能求解凸优化问题，先将传输功率固定为其范围内的随机数(其最优值求解见 3.2 节). 借助 CPLEX[22] 凸优化工具在多项式时间求得最优解 $\widetilde{z_v^m}$. 由于 $\widetilde{z_v^m}$ 为连续变量，所以求得的解 $0 \leqslant \widetilde{z_v^m} \leqslant 1$. 但本文中任务是不可拆分的，需把松弛后得到的结果恢复成原始问题的二进制值.

可将松弛后获得的 $\widetilde{z_v^m}$ 看作任务 v 卸载到边缘服务器 m 上的概率，因此可选择若干个值中最大的 $\widetilde{z_v^m}$，将其取整为 1，作为任务 v 的可行卸载决策. 然后将 $\widetilde{z_v^m}$ 作为已知解重新代入式(12)进行求解. 经过若干次迭代后，可获得所有任务可行的卸载决策.

通过求得的每个任务可行的卸载决策，将 DAG 每条边的权值设为 $w_{v'v} = t_{v'm} + e_{v'm}$，与结束任务相连的边的权值设为 0. 任务 v 到结束任务的最大权值定义为任务 v 的优先级，采用 $L(v)$ 表示，按照优先级倒序存放在队列 Q 中. 按照队列 Q 卸载任务，不仅能保留任务间的依赖关系，还将 DAG 中最长路径上的任务优先卸载执行.

3.2 卸载任务

为了方便描述，再定义一些变量，用 $R(m)$ 表示边缘服务器 m 的剩余资源，$M_v(R)$ 表示同时满足任务 v 的计算资源和服务缓存约束的边缘服务器集合. 根据 3.1 节得到的队列 Q 依次将任务卸载到总成本最小的边缘服务器. 每卸载一个任务 v，用式(13)~(17)更新其实际的开始时间 $AST(v, m_v)$、完成时间 $ACT(v, m_v)$、能耗 $SEC(v, m_v)$、总能耗 $TEC(v, m_v)$ 和剩余资源 $R(m_v)$.

$$AST(v, m_v) = \max\left(SL_m, \max_{v'\in Pred(v)}\left(ACT(v', m_{v'}) + t_{m_{v'}m_v}\right)\right), \tag{13}$$

$$ACT(v, m_v) = AST(v, m_v) + t_{vm_v}, \tag{14}$$

$$SEC(v, m_v) = \sum_{m_{v'}\in M'_v} e_{m_{v'}m_v} + e_{vm}, \tag{15}$$

$$TEC(v, m_v) = TEC(v, m_v) + SEC(v, m_v), \tag{16}$$

$$R(m_v) = R(m_v) - t_{vm_v} r_{vm_v}. \tag{17}$$

在 2.1 节中，为了能够对式(12)求解，假设传输功率为随机值. 随着传输功率取值的增大，时延会变短，但能耗会增加；反之随着传输功率取值的减小，能耗会降低，但又会增加时延. 为了能够使应用程序的总成本最小，应求边缘服务器的最佳传输功率. 由文献 [23] 可知，边缘服务器间最佳传输功率问题为 NP-hard 问题，因此只能采用启发式算法进行求解. 基于 PSO 不依赖问题规模，设置参数少、收敛速度快

的特点, 本文采用改进的 PSO 算法求解边缘服务器间的最佳传输功率.

3.2.1 PSO 算法

PSO 是基于群体的进化方法, 源于对鸟群捕食的行为研究, 基本思想是通过群体中个体之间的协作和信息共享来寻找最优解. 本文采用的 PSO 在 1 维的搜索空间随机产生V个粒子构成原始种群$X=\{x_1, x_2, \cdots, x_v\}$. 对应于本文研究问题, 搜索空间为边缘服务器间传输功率的可取值范围. 每个粒子包含位置和速度 2 个属性, 位置表示搜索空间中一个解, 即边缘服务器间传输功率一个取值; 速度表示粒子在搜索空间中寻找最佳传输功率的方向和大小. 寻优过程中第k个粒子搜索到的最优位置表示为p_{best_k}, 全局搜索过程中粒子群搜索到的最优位置表示为g_{best}, 第k个粒子的位置和速度更新的计算过程[24]分别为

$$x_k(t+1)=x_k(t)+v_k(t+1), \tag{18}$$

$$\begin{aligned} v_k(t+1)=&wv_k(t)+c_1 rand()\left(p_{\text{best}_k}(t)-x_k(t)\right)+\\ &c_2 rand()\left(g_{\text{best}}(t)-x_k(t)\right), \end{aligned} \tag{19}$$

其中w表示惯性权重, c_1和c_2为学习因子, $rand()$表示[0,1]的随机数, t表示迭代次数.

虽然经典 PSO 算法收敛速度较快, 但存在容易陷入局部最优解等缺点[25], 究其原因是因为经典 PSO 算法采用固定的惯性权重. 较大的惯性权重有利于跳出局部解, 便于全局搜索; 而较小的惯性权重则有利于对当前的搜索区域进行精确局部搜索. 当惯性权重的值恒定时, 迭代前期缺少全局搜索能力或迭代后期缺少局部搜索能力, 易造成 PSO 方法陷入局部最优解, 或后期在全局最优解附近产生振荡现象. 为了平衡全局和局部搜索能力, 本文改进了经典 PSO 算法.

3.2.2 改进的 PSO 算法

本文采用如式(20)所示的惯性权重线性递减的调整方法进行寻优:

$$w=w_{\max}-\frac{t\times(w_{\max}-w_{\min})}{t_{\max}}, \tag{20}$$

其中$w_{\max}$, $w_{\min}$分别表示w的最大值和最小值, t表示当前迭代次数, $t_{\max}$表示最大迭代次数, 通常取$w_{\max}=0.9$, $w_{\min}=0.4$[25].

3.2.3 适应度函数构造

PSO 算法采用适应度评价优化目标. 本文将任务在边缘服务器上执行产生的时延和能耗作为优化目标. 构造适应度函数为

$$fitness=\omega\left(\sum_{m'\in M'}e_{m'm}+e_{vm}\right)+(1-\omega)(t_v+t_{vm}), \tag{21}$$

其中$fitness$值越小, 越优, 反之亦然.

3.3 方法实现

通过 3.2 节的分析, 本节将详细描述联合时延和能耗的依赖性任务卸载方法, 即 DTO-TE 方法. 该方法为应用程序中所有卸载任务做出最优的卸载决策, 从而优化任务卸载产生的总成本.

DTO-TE 的具体实现过程如算法 1.

算法 1. DTO-TE 方法.

输入: 任务集合V, 边缘服务器集合M, DAG;

输出: 任务的卸载决策和应用程序的总成本.

①根据 3.2 节计算每个任务的优先级$L(v)$;

②按照任务优先级降序排列, 并保存到卸载队列Q;

③初始化$AST(v)=ACT(v)=TEC(v)=0$;

④初始化边缘服务器的剩余资源$R(m)=C(m)$;

⑤ while $Q\neq\varnothing$

⑥ v=$Q.top()$, 将任务v产生的能耗$SEC(v)$置 0;

⑦ for $m\in M_v(R)$ do

⑧ 用 3.2.2 节提出的改进 PSO 算法计算 m 和 v 前驱任务所在的边缘服务器间的最佳传输功率;

⑨ 计算保存v在边缘服务器m的总成本;

⑩ end for

⑪ 将任务卸载到成本最小的边缘服务器;

⑫ 根据式(14)(16)(17)更新ACT, TEC, $R(m)$;

⑬ 保存v的卸载决策并从队列Q删除任务v;

⑭ end while

算法 1 的分析过程分为 2 个部分: 1)获取任务的优先级, 按照优先级降序保存到队列Q, 并初始化任务和边缘服务器信息(行③④).2)判断Q是否为空, 若非空将其中的任务按顺序卸载到总成本最小的边缘服务器上. 用改进的 PSO 算法求解边缘服务器间的最佳传输功率(行⑤~⑪). 更新任务的完成时间、总能耗和最终卸载的边缘服务器的剩余资源, 并从队列中删除已卸载任务, 继续循环, 直至循环结束(行⑪~⑭).

算法 2 给出了改进的 PSO 算法求得最优传输功率的过程.

算法 2. 改进的 PSO 算法.

输入: 任务v前驱任务所在的边缘服务器和任务v所在的边缘服务器;

输出: 最佳传输功率.

①初始化种群规模$PopSize=20$, 加速因子$c_1=2$, $c_2=2$, 迭代次数为$MaxIter$;

② 根据式(21),计算每个粒子的适应度值;

③ for $j=1$ to $PopSize$

④ 初始位置x_j和初始速度v_j在搜索空间范围内随机产生;

⑤ $p_{\text{best}_j}=x_j$; /*初始化每个粒子的历史最优解*/

⑥ end for

⑦ $g_{\text{best}_j}=\max\left(p_{\text{best}_j}\right)$; /*初始化全局最优解*/

⑧ for $i=1$ to $MaxIter$

⑨ for $j=1$ to $PopSize$

⑩ 根据式(20)更新权重;

⑪ 根据式(18)(19)更新粒子速度v_i和位置x_i;

⑫ 计算适应度函数得出适应度值$fitness(x_i)$;

⑬ 更新粒子的历史最优解;

⑭ 更新全局最优解;

⑮ end for

⑯ end for

⑰ 最佳传输功率=g_{best_i}.

算法 2 可分为 5 个部分. 1)定义了种群规模、加速因子、迭代次数和适应度函数(行①②)并随机生成每个粒子的初始位置(传输功率的初始值)、初始速度、每个粒子的极值和全局最优解(行③~⑦).2)进行迭代(行⑧~⑯), 迭代部分可分为 3 个阶段: 更新权重; 更新每个粒子的位置、速度; 计算当前位置的适应度值(当前传输功率对应的成本)(行⑩~⑫). 3)比较当前粒子位置所得适应度值是否小于当前粒子历史最优解的适应度值, 若是, 则更新粒子的历史最优解; 否则, 当前粒子历史最优解不变(行⑬). 4)比较当前粒子历史最优解的适应度值是否小于全局最优解的适应度值, 若是, 则更新全局最优解(行⑭). 5)迭代结束后, 将全局最优解作为最佳传输功率(行⑰).

3.4 算法 1 的时间复杂度分析

假设应用程序中的任务数量为n, 边缘服务器数量为l, 每个任务的前驱任务数量为P, 粒子种群规模为$PopSize$, 最大迭代次数为$MaxIter$. 算法 1 中, 行①~④求解任务的卸载顺序, 并初始化信息, 其时间复杂度为$O(n^2)$; 行⑤~⑬, 按照队列中任务顺序进行卸载, 但卸载过程中需用改进的 PSO 算法求解边缘服务器间的传输功率. 由于本文采用的 PSO 算法是一维的, 故时间复杂度为$O(MaxIter\times PopSize+PopSize)$. 所以算法 1 中行⑤~⑬的时间复杂度为$O(n\times l\times P\times PopSize\times(MaxIter+1))$.

综上所述, 算法 1 总的时间复杂度为$O\left(n^2+n\times l\times P\times PopSize\times(MaxIter+1)\right)$.

4 实验验证

4.1 实验设置

本文实验配置为: 华硕灵耀 S4000UA, Intel$^{\text{R}}$ Core$^{\text{TM}}$i5-7200U CPU@ 2.5~2.7 GHz, RAM 8192 MB, Windows 10 中文版 64 b, 实验环境为 Python3.9.0.

在 EC 网络场景中[26], 假设网络中有 1 个应用程序需要卸载. 该应用程序可分为若干个任务, 根据任务之间的依赖关系, 使用拓扑排序算法生成 DAG. 使用谷歌数据集[27]模拟每个任务在边缘服务器上的处理时间和所需的处理资源. 为了便于实验, 本文采用文献 [8] 中的做法, 将数据集中的任务执行时间和所需的处理资源统一映射至区间(1,10). 缓存某一服务的边缘服务器数量并设置为边缘服务器总数量的 50%. 边缘服务器间信道带宽和背景噪声功率设置参考文献 [28], 边缘服务器的能量系数、最大传输功率和信道增益参考文献 [18], 具体数值如表 2 所示.

Table 2 Experimental Parameters

表 2 实验参数

实验参数	取值
信道带宽B/GHz	1
背景噪声功率N/dBm	−174
边缘服务器的能量系数κ	16
最大传输功率$p_{\max}$/dBm	20
边缘服务器间信道增益$h_{mm'}$	10^{-6}
执行时延t_{vm}	(1,10)

4.2 对比方法

由于没有和本文方法完全一致的方法, 故修改与本文类似的 3 种已有方法作为对比方法.

对比方法 1: 文献 [8] 提出的方法 CP, 其在边缘服务器具有服务缓存和有限资源的约束下, 优化应用程序的完成时间. 根据本文的研究问题, 在 CP 方法的优化目标中加入能耗.

对比方法 2: 文献 [11] 提出的基于贪婪卸载的个体时间分配方法 ITAGS, 其目标是在满足完成时间约束的同时最小化通信和计算成本. 为了能使 ITAGS 与本文所提方法进行比较, 将 ITAGS 的约束条件修改为本文所提方法的约束条件.

对比方法 3: 文献 [12] 提出的一种上下文感知的

贪婪任务调度方法 CaGTS, 其目标是最小化任务的响应时间. 将 CaGTS 的优化目标和约束条件修改为本文方法的优化目标和约束条件.

4.3 数值归一化

相较于时延, 能耗的值较大, 两者不在同一数量级, 导致时延在实验结果中的占比小, 对任务的卸载决策影响较小. 为了使时延和能耗在同一数量级上, 需要对时延和能耗进行归一化. 由于时延和能耗中存在未知变量, 如传输功率作为一种常用的归一化方法不适用于本文.

本文采用文献 [29] 的方法, 将时延扩大到与能耗同一数量级, 即$E/(\alpha\times T)\approx 1$, 归一化因子$\alpha$为正数. 对应用程序包含的任务数量取不同值, 并设定 DAG 边数是任务数量的 2 倍, 边缘服务器的数量为 10.采用不同的α值分别进行实验, 实验结果如图 3 所示. 随着α的增加不同任务数量下的$E/(\alpha\times T)$呈下降趋势. 当$\alpha=10$时, 不同任务数量下的$E/(\alpha\times T)\approx 1$. 因此, 在下面实验中, 均将时延扩大 10 倍后再计算应用程序的总成本.

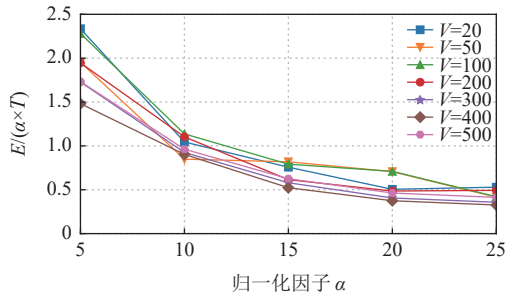


Fig. 3 Effect of α on $E/(\alpha\times T)$

图 3 α对$E/(\alpha\times T)$的影响

4.4 参数分析

为了降低参数对实验结果的影响, 本节对权重因子、边缘服务器数量和迭代次数设置不同的值进行实验, 并观察这些参数对实验结果的影响. 为了保证实验结果的公正性, 本文每类实验结果均取运行 100 次实验后的平均值.

4.4.1 权重因子ω的影响

设定实验环境中边缘服务器的数量为 10, 应用程序的任务数量为 500, DAG 边的条数为 1 000. 变化ω的取值, 采用本文方法和 3 种对比方法进行实验. 时延随ω变化结果如图 4(a)所示, 可以看出ω取值与时延成正比关系; 能耗随ω变化结果如图 4(b)所示, ω取值与能耗成反比关系. 也就是说, 当$\omega<0.5$时, 意味着用户更偏好时延, 此时, 可增大用户设备到边缘服务器的传输功率, 降低任务的传输时间; 相反, 当$\omega\geqslant 0.5$时, 意味着用户对能耗的需求较高, 此时, 边缘服务器可以适当增大应用程序的完成时间, 以降低边缘服务器的能耗. 为了验证本文方法既能满足用户时延偏好又能满足能耗偏好, 下面每类时延对ω分别取值为 0.2 和 0.8.

此外, 从图 4 还可以看出, 无论ω怎么取值, 本文方法实验结果均优于 3 个对比方法.

4.4.2 边缘服务器数量对总成本的影响

边缘环境中边缘服务器的数量对实验结果也会产生影响. 数量较少时, 每个边缘服务器需要处理的任务会较多. 又因为边缘服务器串行处理卸载在其上的任务, 所以会增加任务的等待时延, 从而增加应用程序的总时延. 真实场景中因为成本问题, 网络提供商也不可能部署特别多的边缘服务器. 所以本节通过实验, 试图找到一种合适的服务器数量设置, 既不会增加过多的等待时延, 又较符合真实场景中边缘服务器的数量部署.

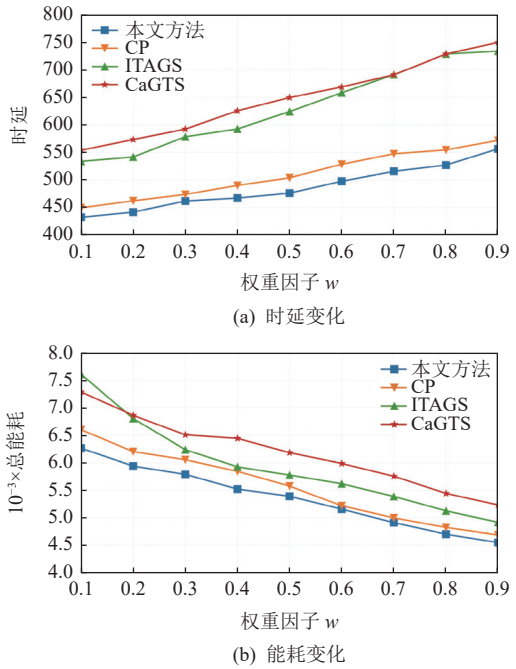
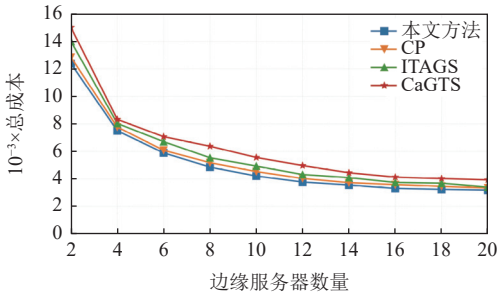


Fig. 4 Effect of weights on time and energy consumption

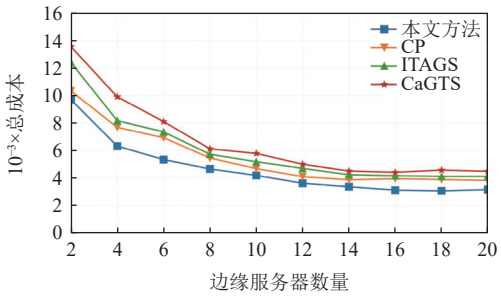
图 4 权重对时间和能耗的影响

设定应用程序包含的任务数量为 500, DAG 边的条数为 1 000, ω分别取值为 0.2 和 0.8, 变化边缘服务器的数量, 采用 4 种方法分别进行实验, 实验结果如图 5 所示. 随着边缘服务器数量的增加, 4 种方法的总成本都明显下降. 主要因为当边缘服务器数量为 2 时, 所有任务只能卸载到 2 个边缘服务器上, 且卸载到同一边缘服务器的概率变大, 因此, 任务在边缘服务器上的等待时间变长, 从而增加了总成本. 而随

着边缘服务器数量的增加，任务可选择卸载的边缘服务器数量也随之增加，对边缘服务器资源的竞争也在降低，因此，降低了任务在边缘服务器的等待时间，从而降低了总成本. 但当边缘服务器数量不断增加时，总成本降低幅度变得很小且趋于平稳，这是因为当边缘服务器增加到一定数量时，在固定任务数下，系统资源变得充足且可以满足所有任务. 本文方法考虑从最长路径卸载任务，可以减少任务在边缘服务器的等待时间；此外，考虑待卸载任务的前驱任务卸载决策和边缘服务器间的最佳传输功率. 因此，与其他 3 种方法相比，本文方法总成本最小. 从图 5 可以看出，边缘服务器数量小于 10 时，4 种方法的总成本降低速度较快；当边缘服务器数量大于 10 时，4 种方法的总成本降低速度放缓. 因此，服务器数量较佳设置应为 10.



(a) ω=0.2 时的总成本变化



(b) ω=0.8 时的总成本变化

Fig. 5 Effect of the number of edge servers on the total costs

图 5 边缘服务器数量对总成本的影响

4.4.3 PSO 迭代次数的影响

对应用程序包含的任务数量取不同的值，并设定 DAG 边数是任务数量的 2 倍，采用不同的迭代次数分别进行实验，实验结果如图 6 所示. 随着迭代次数的增加，不同任务数量下的应用程序总成本均呈下降趋势. 而当迭代次数大于 20 时，总成本下降的趋势很小. 继续迭代可能会求得更优的传输功率，但对总成本的影响很小. 而多余的迭代次数会增加算法的运行时间. 因此，为了尽可能降低总成本，在接下来的实验中，设置 PSO 的迭代次数为 20.

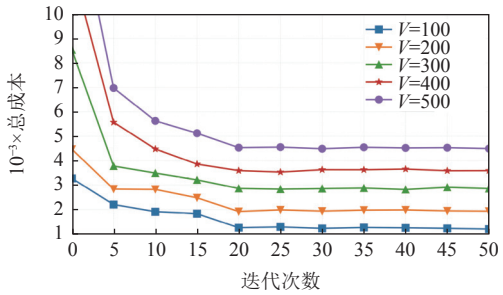


Fig. 6 Effect of PSO iteration numbers on total costs

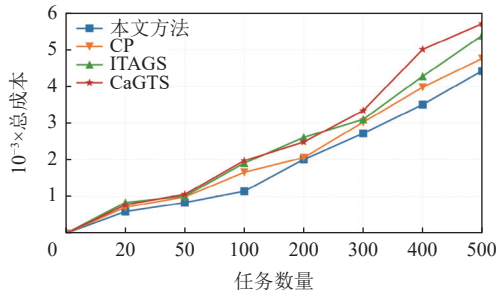
图 6 PSO 迭代次数对总成本的影响

4.5 实验对比

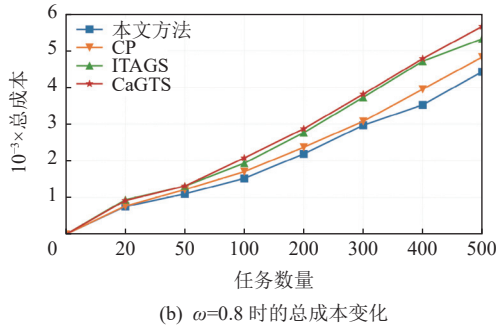
本节通过改变应用程序中任务数量、应用程序大小和任务间依赖关系，并设定边缘服务器数量为 10，ω分别取值为 0.2 和 0.8，PSO 迭代次数为 20.采用本文方法与 3 种对比方法进行实验对比.

4.5.1 任务数量对总成本的影响

对应用程序包含的任务数量取值，并设定 DAG 的边的数量是任务数量的 2 倍，分别采用 4 种方法进行实验，实验结果如图 7 所示. 随着任务数量的增加，4 种方法的总成本随之增加，且 CaGTS 增加得最多. 究其原因为 CaGTS 总将任务卸载到执行时间最小的边缘服务器，不考虑前驱任务的卸载位置，导致产生大量传输能耗，从而增加总成本.ITAGS 和 CaGTS 在任务数量较少时，总成本很接近，但随着任务数量增加，ITAGS 的总成本小于 CaGTS，主要因为 ITAGS 每次贪婪卸载任务时，会兼顾前驱任务的卸载位置，从



(a) ω=0.2 时的总成本变化



(b) ω=0.8 时的总成本变化

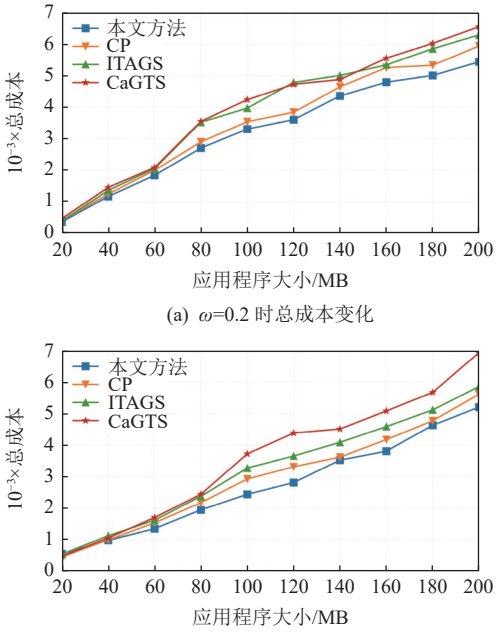
Fig. 7 Effect of task quantity on total costs

图 7 任务数量对总成本的影响

而减少了边缘服务器间的通信成本.CP 和 ITAGS 相比，在任务数量较少的时候，两者的总成本很接近，但随着任务数量的逐渐增大，CP 明显优于 ITAGS，究其原因是 CP 优先卸载执行时间长且距离结束任务远的任务，因此进一步降低了总成本. 本文方法不仅先卸载距离结束任务远的任务，还兼顾前驱任务的卸载位置. 此外，本文方法在进行依赖性任务的数据传输时，采用最佳的传输功率降低了总成本. 因此，此类实验本文方法结果最佳.

4.5.2 应用程序大小对总成本的影响

设定应用程序包含的任务数量为 100，任务大小随机生成，执行时间根据谷歌数据集中任务大小和执行时间的比值求得，DAG 边数为 200，采用 4 种方法分别进行实验，实验结果如图 8 所示. 因为本文实验中应用程序任务数量固定不变，所以随着应用程序大小的增加，任务的平均大小也会增加，边缘服务器处理任务的时延和能耗会随之变大，且边缘服务器处理任务所需的资源也在增加. 因此，4 种方法的总成本也随之增加.



(a) ω=0.2 时总成本变化

(b) ω=0.8 时总成本变化

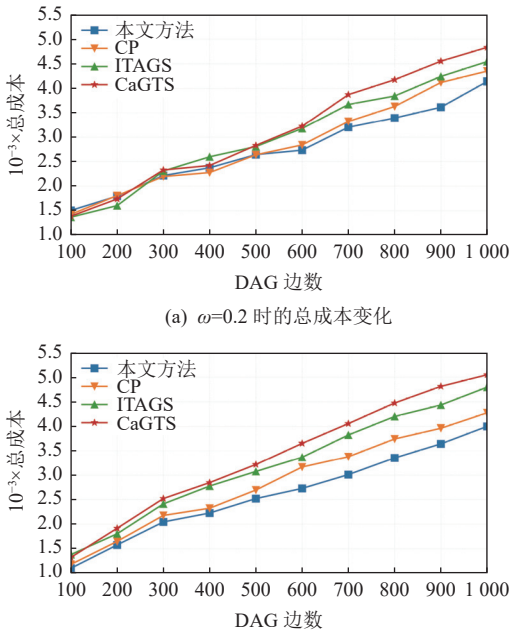
Fig. 8 Effect of App sizes on total costs

图 8 应用程序大小对总成本的影响

由于 CaGTS 和 ITAGS 都不考虑任务卸载的优先级，导致距离结束任务较远的任务可能后执行，会产生较长的等待时间. 与 CaGTS 和 ITAGS 不同，CP 考虑了前驱任务的卸载位置，且优先卸载距离结束任务远的任务，这将大大降低任务在边缘服务器的等待时间. 但由于 CP 的传输功率是固定的，当任务大小增加时，任务间传输数据的成本就会增加，进而增加总成本. 本文方法和 CP 相比，在应用程序较小时，总成本非常相近. 因为这 2 种方法的边缘服务器处理任务的时间相同.CP 是根据任务的执行时间对传输时间进行调整，因此在应用程序较小时，传输时间也相对较小. 但随着应用程序增大，边缘服务器执行任务的时间也增大，因此 CP 的传输时间也会增大，从而对总成本影响较大. 本文方法无论在应用程序多大时，均能求得边缘服务器的最佳传输功率，从而降低总成本.

4.5.3 任务依赖关系对总成本的影响

设定应用程序的任务数量为 300，依赖关系，也就是 DAG 边数从 100 递增至 1 000，实验结果如图 9 所示.DAG 边数越多，任务间依赖关系越复杂，致使边缘服务器间的传输次数的增加，从而导致这 4 种方法总成本增加. 当 DAG 边较少时，4 种方法的总成本都很相近. 这是因为任务之间依赖较少时，边缘服务器间的传输次数也较少，此时本文方法通过获取最佳传输功率从而降低总成本的优势不明显. 此外，本文方法为得到最佳传输功率，还需额外增加计算时延，从而导致在$\omega = 0.2$(用户更偏好时延)，边数为 100~200 时，本文方法性能略差于对比方法. 然而，随着 DAG 边数的增加，边缘服务器间的通信成本随之增大，此时本文方法明显优于其他 3 种方法.



(a) ω=0.2 时的总成本变化

(b) ω=0.8 时的总成本变化

Fig. 9 Effect of task dependency relationship on total costs

图 9 任务的依赖关系对总成本的影响

相比于 ITAGS 和 CaGTS，本文方法优先卸载到

结束任务路径最长的任务，可以减少任务在边缘服务器的等待时延；并且卸载任务时考虑到当前任务的前驱任务，通信成本也会降低. 本文方法和 CP 的总成本总体上接近，特别是在 DAG 边数在 100~500 之间，这是因为这 2 种方法都优先卸载最长路径上的任务且考虑前驱任务的卸载决策，但随着边数不断的增多，本文方法明显优于 CP. 这是因为任务之间的依赖关系变得越来越复杂，且任务的前驱任务也变多了，边缘服务器之间传输数据的次数也变得频繁，由于本文方法考虑边缘服务器间传输数据时的最佳传输功率，可以获得最小的传输时延和能耗，从而降低总成本. 因此，本文方法取得的实验结果最佳. 由此可知，本文方法更适合应用依赖任务较多的情形.

5 总结与展望

本文研究了在边缘服务器计算资源和服务缓存有限的约束下，权衡时延和能耗的依赖性任务卸载问题，并提出了 DTO-TE 方法.DTO-TE 通过松弛研究问题的约束将其转换成凸优化问题，并求得松弛解；根据松弛解求得满足约束的可行解，从而确定任务的卸载优先级；按照优先级依次将任务卸载到成本最小的边缘服务器上. 实验结果充分表明，本文方法在本文设置的各类实验中均获得了最优的结果.

虽然本文方法较对比方法有一定的优势，但是仍存在一些不足：1)本文方法只进行了基于真实数据的仿真实验，因实验设备的限制并没有在真实场景下进行实验验证；2)本文方法只建立了边—端卸载模型，只适合较小规模应用程序，对于大规模应用程序，可以考虑建立云—边—端协同卸载模型；3)本文方法只研究 1 个应用程序卸载，实际应用中存在协同卸载多个应用程序的场景. 因此，下一步工作拟研究云—边—端模型中的多个应用程序的协同卸载，并在真实场景下对研究问题进行实验验证.

作者贡献声明：张俊娜提出论文思路，并全程指导实验设计、结果分析和论文撰写；鲍想负责实验的实现和论文的撰写；陈家伟和赵晓焱参与实验设计和论文撰写；袁培燕和王尚广修改论文.

参 考 文 献

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

[2] Wang Haipei, Lin Zhipeng, Lv T. Energy and delay minimization of partial computing offloading for D2D-assisted MEC systems [C/OL] // Proc of the 13th IEEE Wireless Communications and Networking Conf. Piscataway, NJ: IEEE, 2021 [2022-12-02].https://ieeexplore.ieee.org/document/9417536

[3] Hu Yuncao, Patel M, Sabella D, et al. Mobile edge computing—A keytechnology towards 5G [J/OL]. World Class Standards, 2015 [2022-12-02].https://infotech.report/Resources/Whitepapers/f205849d-0109−4de3−8c47-be52f4e4fb27_etsi_wp11_mec_a_key_technology_towards_5g.pdf

[4] Hu Junyan, Li Kenli, Liu Chubo, et al. Game-based task offloadingof multiple mobile devices with QoS in mobile edge computing systems of limited computation capacity [J/OL]. ACM Transactions on Embedded Computing Systems, 2020 [2022-12-02].https://dl.acm.org/doi/abs/10.1145/3398038

[5] Alfakih T, Hassan M M, Gumaei A, et al. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA[J]. IEEE Access, 2020, 8: 54074−54084

[6] Choi J. Random access-based multiuser computation offloading for devices in IoT applications[J]. IEEE Internet of Things Journal, 2022, 9(21): 22034−22043

[7] Li Xiang, Fan Rongfei, Hu Han, et al. Joint task offloading and resource allocation for cooperative mobile edge computing under sequential task dependency[J]. IEEE Internet of Things Journal, 2022, 9(23): 24009−24029

[8] Zhao Gongming, Xu Hongli, Zhao Yangming, et al. [C] //Proc of the 39th IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2020: 1997−2006

[9] Fan Yinuo, Zhai Linbo, Wang Hua. Cost-efficient dependent task offloading for multiusers[J]. IEEE Access, 2019, 7: 115843−115856

[10] Liu Wei, Huang Yucheng, Du Wei, et al. Resource-constrained serial task offloading strategy in mobile edge computing[J]. Journal of Software, 2020, 31(6): 1889−1908 (in Chinese)
(刘伟, 黄宇成, 杜薇, 等. 移动边缘计算中资源受限的串行任务卸载策略[J]. 软件学报, 2020, 31(6): 1889−1908)

[11] Sundar S, Liang Ben. Offloading dependent tasks with communication delay and deadline constraint [C] //Proc of the 37th IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2018: 37−45

[12] Cai Lingfeng, Wei Xianglin, Xing Changyou, et al. Failure-resilient DAG task scheduling in edge computing[J]. Computer Networks, 2021, 198: 108361−108377

[13] Hossain M D, Huynh L N, Sultana T, et al. Collaborative task offloading for overloaded mobile edge computing in small-cell networks [C] //Proc of the 34th Int Conf on Information Networking. Piscataway, NJ: IEEE, 2020: 717−722

[14] Zhang Liping, Chai Rong, Yang Tiantian, et al. Min-max worst-case design for computation offloading in multi-user MEC system [C] // Proc of the 39th IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2020: 1075−1080

[15] Zhang Haibo, Li Hu, Chen Shanxue, et al. Task offloading and resource optimization based on mobile edge computing in ultra-dense networks[J]. Journal of Electronics and Information, 2019, 41(5): 1194−1201 (in Chinese)
(张海波, 李虎, 陈善学, 等. 超密集网络中基于移动边缘计算的任

务卸载和资源优化[J]. 电子与信息学报, 2019, 41(5): 1194−1201)
[16] Zhang Yameng, Liu Tong, Zhu Yanmin, et al. A deep reinforcement learning approach for online computation offloading in mobile edge computing [C/OL] //Proc of the 28th ACM Int Symp on Quality of Service. New York: ACM, 2020 [2022-12-04].https://ieeexplore.ieee.org/document/9212868
[17] Zhang Ni, Guo Songtao, Dong Yifan, et al. Joint task offloading and data caching in mobile edge computing networks[J]. Computer Networks, 2020, 182: 107446−107467
[18] Wang Jin, Wu Wenbing, Liao Zhuofan, et al. A probability preferred priori offloading mechanism in mobile edge computing[J]. IEEE Access, 2020, 8: 39758−39767
[19] Mazouzi H, Achir N, Boussetta K. Elastic offloading of multitasking applications to mobile edge computing [C] //Proc of the 22nd Int Conf on Modeling, Analysis and Simulation of Wireless and Mobile Systems. New York: ACM, 2019: 307−314
[20] Liu Liuyan, Tan Haisheng, Jiang S H C, et al. Dependent task placement and scheduling with function configuration in edge computing [C/OL] //Proc of the 27th ACM Int Symp on Quality of Service. New York: ACM, 2019 [2022-12-04].https://ieeexplore.ieee.org/document/9068608
[21] Ko S W, Kim S J, Jung H, et al. Computation offloading and service caching for mobile edge computing under personalized service preference[J]. IEEE Transactions on Wireless Communications, 2022, 21(8): 6568−6583
[22] Cplex II. V12.1: User's manual for CPLEX[J]. International Business Machines Corporation, 2009, 46(53): 157−263
[23] Barney B. Introduction to parallel computing[J]. Lawrence Livermore National Laboratory, 2010, 6(13): 10−159
[24] Yang Wei, Li Qiqiang. A review of particle swarm optimization algorithms[J]. Chinese Engineering Science, 2004, 6(5): 87−94 (in Chinese)
(杨维, 李歧强. 粒子群优化算法综述[J]. 中国工程科学, 2004, 6(5): 87−94)
[25] Hu Wang, Li Zhishu. A simpler and more effective particle swarm optimization algorithm[J]. Journal of Software, 2007, 18(4): 861−868 (in Chinese)
(胡旺, 李志蜀. 一种更简化而高效的粒子群优化算法[J]. 软件学报, 2007, 18(4): 861−868)
[26] Zhang Wenzhu, Yu Jinghua. Task offloading strategy in mobile edge computing based on cloud-edge-end cooperation [J]. Journal of Computer Research and Development, 2023, 2: 371−385(in Chinese)
(张文柱, 余静华. 移动边缘计算中基于云边端协同的任务卸载策略[J]. 计算机研究与发展, 2023, 2: 371−385)
[27] Reiss C, Tumanov A, Ganger G R, et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis [C/OL] //Proc of the 3rd ACM Symp on Cloud Computing. New York: ACM, 2012 [2022-12-06].https://dl.acm.org/doi/abs/10.1145/2391229.2391236
[28] Chi Guoxuan, Wang Yumei, Liu Xiang, et al. Latency-optimal task offloading for mobile-edge computing system in 5G heterogeneous networks [C/OL] //Proc of the 87th IEEE Vehicular Technology Conf. Piscataway, NJ: IEEE, 2018 [2022-12-04].https://ieeexplore.ieee.org/document/8417606
[29] Zhang Jiao, Hu Xiping, Ning Zhaolong, et al. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks[J]. IEEE Internet of Things Journal, 2017, 5(4): 2633−2645

Zhang Junna, born in 1979. PhD, associate professor, master supervisor. Member of CCF. Her main research interests include edge computing and service computing.
张俊娜, 1979 年生. 博士, 副教授, 硕士生导师. CCF 会员. 主要研究方向为边缘计算、服务计算.

Bao Xiang, born in 1997. Master candidate. Student member of CCF. His main research interests include edge computing and service computing.
鲍　想, 1997 年生. 硕士研究生. CCF 学生会员. 主要研究方向为边缘计算、服务计算.

Chen Jiawei, born in 1998. Master candidate. Student member of CCF. His main research interests include edge computing and service computing.
陈家伟, 1998 年生. 硕士研究生. CCF 学生会员. 主要研究方向为边缘计算、服务计算.

Zhao Xiaoyan, born in 1981. PhD, associate professor, master supervisor. Member of CCF. Her main research interests include edge computing and D2D communication.
赵晓焱, 1981 年生. 博士, 副教授, 硕士生导师. CCF 会员. 主要研究方向为边缘计算、D2D 通信.

Yuan Peiyan, born in 1978. PhD, professor, master supervisor. Member of CCF. His main research interests include edge computing and crowd sensing.
袁培燕, 1978 年生. 博士, 教授, 硕士生导师. CCF 会员. 主要研究方向为边缘计算、群智感知.

Wang Shangguang, born in 1982. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interests include service computing, mobile cloud computing, and Internet of vehicles and network security.
王尚广, 1982 年生. 博士, 教授, 博士生导师. CCF 杰出会员. 主要研究方向为服务计算、移动云计算、车联网与网络安全.