计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200600129

基于延迟接受的多用户任务卸载策略

毛莺池　周　彤　刘鹏飞
河海大学计算机与信息学院　南京 211100
(yingchimao@hhu.edu.cn)

摘　要　随着人工智能的应用对计算资源的要求越来越高,移动设备由于计算能力和存储能量有限而无法处理这类有实时性需求的计算密集型应用。移动边缘计算(Mobile Edge Computing,MEC)可以在无线网络边缘提供计算卸载服务,达到缩短时延和节约能源的目的。针对多用户依赖任务卸载问题,在综合考虑时延与能耗的基础上建立用户依赖任务模型,提出了基于延迟接受的多用户任务卸载策略(Multi-User Task Offloading Based on Delayed Acceptance,MUTODA),用于解决时延约束下最小化能耗的任务卸载问题。该策略通过非支配的单用户最优卸载策略和解决资源竞争的调整策略两个步骤的不断迭代,来解决多用户任务卸载问题。实验结果表明,相比基准策略和启发式策略,基于延迟接受的多用户任务卸载策略能够提高约8%的用户满意度,节约30%～50%的移动终端能耗。

关键词:移动边缘计算;任务卸载;博弈论;任务依赖性
中图法分类号　TP399

Multi-user Task Offloading Based on Delayed Acceptance

MAO Ying-chi,ZHOU Tong and LIU Peng-fei
College of Computer and Information,Hohai University,Nanjing 211100,China

Abstract　With the application of artificial intelligence,the demand for computing resources is higher and higher. Due to the limited computing power and energy storage,mobile devices can not deal with this kind of computing intensive applications with real-time requirements. Mobile edge computing (MEC) can provide computing offload service at the edge of wireless network,so as to reduce the delay and save energy. Aiming at the problem of multi-user dependent task offloading,a user dependent task model is established based on the comprehensive consideration of delay and energy consumption. The multi-user task offloading strategy based on delay acceptance (MUTODA) is proposed to solve the problem of minimizing energy consumption under delay constraints. MUTODA solves the problem of multi-user task offloading through two steps of non dominated single user optimal offloading strategy and adjustment strategy to solve resource competition. The experimental results show that compared with the benchmark strategy and heuristic strategy,the multi-user task offloading strategy based on delayed acceptance can improve about 8% user satisfaction and save 30%～50% of the energy consumption of mobile terminals.

Keywords　Mobile edge computing,Task offloading,Game theory,Task interdependence

1　引言

当下设备环境中,越来越多的计算密集型应用要求低延迟,如虚拟现实、增强现实、在线游戏等。这些应用往往需要较大的计算量和较高能耗,这导致计算能力和电池电量有限的移动设备难以处理。移动云计算(Mobile Cloud Computing,MCC)利用云的丰富的存储空间和计算能力等资源优势,将任务卸载到远程云中,利用云强大的计算能力,为用户提供更丰富的资源和更好的使用体验[1]。这在一定程度上缓解了由移动设备计算能力不足带来的时延和能耗问题。然而,大量数据传到云会导致较大的网络传输开销,利用现有的基础设施很难缩短远程云服务器和用户设备之间的网络时延。

为了解决上述问题,欧洲电信标准化协会(European Telecommunication Standards Institute,ETSI)于2014年提出了移动边缘计算(Mobile Edge Computing,MEC)[2],其将MEC服务器密集地部署在移动用户附近,使计算和存储等资源更靠近用户。由于移动边缘计算可以在无线网络边缘以低时延、低能耗的方式提供计算卸载、无线缓存等服务,因此在虚拟现实、物联网、车联网等许多场景中具有广阔的应用前景。

MEC系统模型主要包括移动用户设备、基站、MEC服务器[3-4]。移动设备通过无线网络与基站进行通信,MEC服务

到稿日期:2020-06-20　返修日期:2020-11-19　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划课题(2018YFC0407105);国家自然科学基金重点项目(61832005);华能集团总部科技项目(HNKJ19-H12)
This work was supported by the China National Key R&D Program (2018YFC0407105), National Natural Science Foundation of China (61832005) and Technology Project of China Huaneng Group Headquarters(HNKJ19-H12).
通信作者:周彤(1172267343@qq.com)

器通常由服务运营商部署在基站附近，MEC 服务器是类似于远程云的小型服务器，它具备计算、存储等一系列云计算中心拥有的功能。计算任务卸载是移动边缘计算的重要支撑技术，用户通过无线信道将任务卸载到 MEC 服务器上进行处理，实现缩短时延、降低移动设备能耗以及节约任务处理成本等目标。同时，MEC 服务器可以承担转发工作，将用户任务发送到远程云中进行处理。相比移动云计算，MEC 任务不再需要经过核心网络，具有低时延、低能耗等优点，有效提升了用户体验感。

由于 MEC 服务器的资源有限，难以处理覆盖范围内所有用户的卸载任务，因此需要制定有效的卸载策略，以尽可能地满足用户需求，提高 MEC 服务器计算资源的利用率。此外，每个用户的需求不同，会产生任务的差异性。用户能够以部分卸载或完全卸载的形式将任务卸载到 MEC 服务器以及远程云，但是当组成任务的子任务具有依赖关系时，任务卸载问题会变得更加复杂，如何解决依赖性任务的卸载问题是一个新的挑战。

本文在任务具有依赖关系的场景下，研究多用户的任务卸载问题。在综合考虑时延与能耗的基础上建立细粒度的任务卸载模型，提出依赖任务场景下的多用户任务卸载策略，以解决时延约束下最小化能耗的任务卸载问题。首先，通过非支配遗传算法，求解每个用户在单用户场景下的最优解；其次，针对遗传算法的特点，通过加入非支配的评判方案和按概率的选择机制提高收敛速度；然后，基于稳定匹配中延迟接受的思想，提出了调整策略；最后，通过不断迭代，解决了多用户任务卸载问题。

本文第 2 节介绍了任务卸载的相关工作；第 3 节描述了基于延迟接受的多用户依赖任务卸载策略的系统模型；第 4 节给出了该策略的总体思路和具体实现过程；第 5 节通过仿真实验评估了该卸载策略的有效性；最后总结全文。

2 相关工作

任务决策的结果由卸载决策目标决定。目前，边缘环境中的任务卸载目标主要集中为 3 类：降低时延、减少能耗以及权衡时延和能耗达到良好的卸载效果。Wang 等[5]通过将时延和能耗的加权和达到最小化作为研究目标，改进遗传算法并实现计算卸载。Liu 等[6]通过分析每个任务的平均延迟和移动设备的平均功耗，将计算卸载任务建模为一个功率受限的时延最小化问题，提出了一种有效的一维搜索算法来寻找最优的任务调度策略。Wu 等[7]以最小化完成所有终端用户计算任务的总时延为目标，提出了一种计算最优卸载解的有效算法，使得任务执行总时延最小。Xu 等[8]基于拍卖理论提出了一种资源分配机制。Wang 等[9]提出了一种能耗最优的部分卸载方案，使得用户(UE)根据任务的最大允许时延调整其计算能力，在满足应用的执行时延的同时通过动态电压缩放技术来实现最小化能量消耗。Yu 等[10]通过博弈论的功率分配算法提高计算卸载性能。Sardellitti 等[11]研究了计算资源和无线资源的联合优化，提出了一种基于逐次凸逼近的迭代算法，在满足时延约束的同时最小化能耗。Mao 等[12]针对多 UE 情况下的能耗和执行时延之间的权衡问题，提出了基于 Lyapunov 优化的在线算法。Wang 等[13]针对能耗和时延之间的折中问题，将此问题建模为整数规划问题，分别提出了一种两阶段优化算法和一种迭代改进算法，以达到优化计算效率的目的。

时延是影响用户体验最主要的因素，高额的时延会直接影响用户体验感。同时，移动设备的能源存储有限，减少移动设备的能耗有利于设备的长久工作。综上可知，时延和能耗都是影响用户体验的重要指标。因此，本文综合考虑时延和能耗，在尽可能满足更多用户时延需求的前提下最小化用户总能耗。

卸载任务本身的特点对卸载决策也有重要影响，有的任务可以被划分为多个子任务，有的子任务可以被卸载，而有的子任务只能在移动设备上运行。此外，不同子任务之间可能存在复杂的依赖关系，分为独立任务卸载和依赖任务卸载。Meng-Hsi 等[14]考虑了通用的多个用户多个独立任务的计算卸载场景。其联合计算资源和通信资源，以最小化所有用户的能耗、计算和时延的总体成本为目标，提出了一种有效的多用户多任务卸载算法。仿真结果表明，在不同的参数设置下，该算法能提供近乎最优的性能，并且可以带来可观的成本效益。Ning 等[15]研究了多个依赖任务场景下最小化所有用户任务响应时延的问题，将多用户计算卸载问题归结为一个混合整数线性规划问题，设计了一种迭代启发式的资源分配算法，动态地进行卸载决策。Guo 等[16]研究了多个依赖任务在完成时延的硬约束下实现节能的计算卸载问题，提出了一种分布式的动态卸载和资源调度的节能算法。

在多个用户任务卸载场景中，任务是否具有依赖性对卸载策略的制定有巨大的影响。在实际场景中，任务间多有依赖关系，因此本文针对多用户依赖任务卸载问题提出了有效的解决策略，满足了用户任务的时延要求，并且能够节约卸载过程中用户设备的能耗。

3 系统模型

3.1 网络模型

图 1 给出了具有任务依赖关系的多用户任务卸载网络模型。基站靠近用户和 MEC 服务器，用户卸载请求发送到基站，再由基站卸载到 MEC 服务器或云服务器。

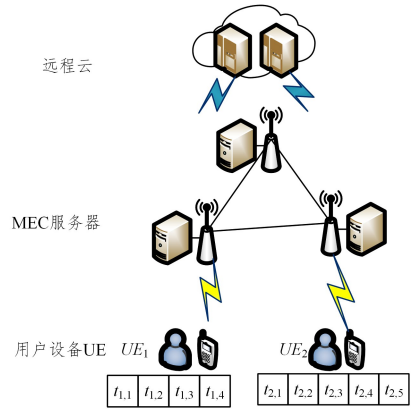


图 1 具有任务依赖关系的多用户任务卸载示例

Fig. 1 Example of multi-user task offload with task dependencies

MEC 服务器的计算能力比用户设备强，可以快速完成卸载调度与任务计算。在某个很小的间隙内且面临多个用户的计算卸载任务的情况下，由于每个用户的任务有最大容忍时

延，除了直接在本地计算之外，可以将每个用户任务分割为几个具有依赖关系的子任务，选择部分或者全部的子任务并将其卸载到 MEC 服务器或者远程云中完成计算。在通信方面，我们不考虑用户之间的干扰，即每个用户都有理想的信道资源。

3.2 任务模型

本文对任务模型进行构建，并将其定义为 $G=(Task_i, De_i, Cy_i, Da_i, Tc_i, delay_i, M_i)$。假设某用户 i 具有用户任务 G_i，那么 $Task_i$ 表示用户 i 任务的集合，$|Task_i|$ 表示用户任务的数量，每个子任务 $t_{i,j}$ 表示用户 i 的第 j 个子任务。De_i 表示任务之间依赖关系的集合。例如，$de_{i,j,k}$ 表示任务 $t_{i,j}$ 和任务 $t_{i,k}$ 存在依赖关系，$t_{i,j}$ 的输入数据依赖 $t_{i,k}$ 的输出。Cy_i 表示用户 i 每个任务所需计算周期数的集合，每个任务的计算周期数用 $cycles_i$ 表示。Da_i 表示用户 i 每个任务计算所需数据规模的集合，每个任务的需求数据用 $data_{i,j,k}$ 表示，并且任务 $t_{i,j}$ 依赖任务 $t_{i,k}$。Tc_i 表示依赖任务之间传输数据带来的通信开销，任务 $t_{i,j}$ 与任务 $t_{i,k}$ 之间的通信开销为：

$$tc_{i,j,k}=\frac{data_{i,j,k}}{r_{i,j,k}} \tag{1}$$

其中，$r_{i,j,k}$ 表示用户 i 的任务 $t_{i,j}$ 和任务 $t_{i,k}$ 之间的传输速率。若任务 $t_{i,j}$ 和任务 $t_{i,k}$ 在同一个计算节点上运算，则两者之间的通信开销忽略不计。$delay_i$ 表示用户 i 要求完成所有任务的最大容忍时延。M_i 表示用户 i 可选择的计算节点的集合，最初 M_i 包含 $M+2$ 个计算节点，假设用户任务是有向无环的。图 2 给出了一个用户任务示例。可以得知，在有依赖关系的任务之间，任务有着严格的执行顺序。边 $de_{i,j,k}$ 确定任务 $t_{i,j}$ 和任务 $t_{i,k}$ 之间的偏序关系，下面将进一步给出相关定义。

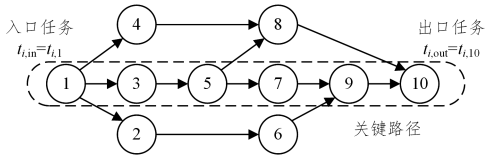


图 2 用户任务示例

Fig. 2 Example of user task

定义 1（前置任务） $t_{i,j}\succ t_{i,k}$，表示任务 $t_{i,j}$ 在任务 $t_{i,k}$ 之前计算，任务 $t_{i,j}$ 为任务 $t_{i,k}$ 的前置任务，任务 $t_{i,k}$ 为任务 $t_{i,j}$ 的后置任务。

定义 2（前驱任务） $t_{i,j}\succ_{De} t_{i,k}$，任务 $t_{i,j}$ 为任务 $t_{i,k}$ 的前置任务，并且两者具有依赖关系；存在一条边 $de_{i,j,k}$，表示两者的依赖关系；称任务 $t_{i,j}$ 为任务 $t_{i,k}$ 的前驱任务。

定义 3（入口任务） $pre(t_{i,k})$ 表示在任务图中，具有 $t_{i,j}\succ t_{i,k}$ 前置关系的所有 $t_{i,j}$ 的集合。$pre(t_{i,k})=\emptyset$ 表示子任务 $t_{i,k}$ 没有前置任务，称 $t_{i,k}$ 为用户 i 的入口任务，记为 $t_{i,in}$。

定义 4（出口任务） $suc(t_{i,j})$ 表示在任务图中，具有 $t_{i,j}\succ t_{i,k}$ 前置关系的所有 $t_{i,k}$ 的集合。$suc(t_{i,j})=\emptyset$ 表示任务 $t_{i,j}$ 没有后置任务，称 $t_{i,j}$ 为出口任务，记为 $t_{i,\text{out}}$。

定义 5（关键路径） 在任务图中，从入口任务开始到出口任务结束，计算开销最大的路径。关键路径将决定任务的响应时间。

定义 6（开始时间） $ST(t_{i,j})$ 表示在任务图中入口任务开始执行到子任务 $t_{i,j}$ 被调度并准备执行的时间点。

定义 7（结束时间） $ET(t_{i,j})$ 表示在任务图中入口任务开始执行到子任务 $t_{i,j}$ 被调度并执行完成的时间点。

由定义 6 和定义 7 可得：

$$ET(t_{i,j})=ST(t_{i,j})+\frac{cycles_i}{C_k} \tag{2}$$

$$ST(t_{i,j})=ET(t_{i,j})+s_{i,j,k},\ t_{i,j}\succ_{De} t_{i,k} \tag{3}$$

其中，C_k 表示 MEC 服务器 k 的计算能力，$k=0$ 时（即 C_0）表示本地设备的计算能力，$k=M+1$ 时（即 C_{M+1}）表示远程云的计算能力。

3.3 时延和能耗模型

3.3.1 时延模型

本文假设用户的回传数据远小于计算时传输的数据，忽略计算结果的回程开销。用户任务 i 的响应时延为 $ET(t_{i,\text{out}})-ST(t_{i,\text{in}})$，$delay_i$ 表示使用户 i 满意的任务最大计算时间。本文的目标是在满足用户延时的前提下，使得用户终端能耗最小，最大化所有用户的满意度（Degree of Satisfaction，DoS）。

为了保证用户满意度，使尽量多的用户任务响应时延短于或等于最大容忍时延，即最大化所有用户的满意度（DoS）。定义单个用户的满意度 DoS_i 为：

$$DoS_i=\begin{cases}1, & ET(t_{i,\text{out}})-ST(t_{i,\text{in}})\leqslant delay_i\\ 0, & ET(t_{i,\text{out}})-ST(t_{i,\text{in}})>delay_i\end{cases} \tag{4}$$

当响应时延 $ET(t_{i,\text{out}})-ST(t_{i,\text{in}})\leqslant delay_i$ 时，用户 i 的满意度为 1，反之为 0。

3.3.2 能耗模型

本文只考虑用户终端设备的能耗，以减少终端所有用户设备总能耗为目标。假设有 N 个用户，用户 i 的能耗由 3 部分组成：传输能耗、计算能耗以及用户节点在等待 MEC 服务器计算机结果时的空闲能耗。

$$e_i=e_i^C+e_i^T+e_i^W \tag{5}$$

其中，e_i 表示用户 i 的能耗，e_i^C 表示用户 i 的计算能耗，e_i^T 表示用户 i 的传输能耗，e_i^W 表示用户 i 等待 MEC 计算结果时的空闲能耗。

假设存在 M 个 MEC 服务器和 1 个远程云服务器，加上本地用户设备，对于用户 i 有 $M+2$ 个计算节点。$t_{i,j}=0$ 表示子任务 $t_{i,j}$ 在用户本地设备上进行计算，$t_{i,j}=k(1\leqslant k\leqslant M)$ 表示子任务 $t_{i,j}$ 在 MEC 服务器中进行计算，$t_{i,j}=M+1$ 表示子任务 $t_{i,j}$ 在远程云中进行计算。则用户任务 i 的计算能耗 e_i^C 为：

$$\begin{aligned}&e_i^C=\sum_{j=1}^{|Task_i|}\alpha_{i,j}\cdot P_i^C\cdot\frac{cycles_i}{C_0}\\&\alpha_{i,j}=\begin{cases}1, & t_{i,j}=0\\ 0, & t_{i,j}\neq 0\end{cases}\end{aligned} \tag{6}$$

其中，P_i^C 表示用户任务 i 的计算功率，$\alpha_{i,j}=1$ 表示任务 $t_{i,j}$ 在本地进行计算，$\alpha_{i,j}=0$ 表示不在本地运算。用户任务 i 的传输能耗 e_i^T 为：

$$e_i^T=\sum_{j=1}^{|Task_i|}\alpha_{i,j}\cdot P_i^T\cdot d_{i,j,k},\alpha_{i,j}=1\ \text{且}\alpha_{i,j}=0,t_{i,j}\succ_{De} t_{i,k} \tag{7}$$

$$e_i^w=\sum_{j=1}^{|Task_i|}|\alpha_{i,j}-1|\cdot P_i^w\cdot\frac{cycles_i}{C_k},\alpha_{i,j}=1\ \text{或}\ 0 \tag{8}$$

其中，P_i^T 表示用户任务 i 的传输功率，P_i^w 表示用户的待机功率。最后可得所有用户的总能耗 e_N 为：

$$e_N=\sum_{i=1}^{N}e_i^C+\sum_{i=1}^{N}e_i^T+\sum_{i=1}^{N}e_i^w=e_N^C+e_N^T+e_N^w \tag{9}$$

3.4 目标函数

本文的目标函数针对多用户任务的场景进行构建，其中 $\max \frac{1}{N}\sum_{i=1}^{N} DoS_i$ 表示最大化用户的满意度，即满足更多用户的需求。

$$\max \frac{1}{N}\sum_{i=1}^{N} DoS_i \tag{10}$$

$\min e_i$ 表示卸载策略满足用户需求时，最小化该用户的能耗；当不能满足用户满意度时，忽略能耗影响因素，通过 $\min(ET(t_{i,\text{out}})-ST(t_{i,\text{in}}))$ 来最小化用户任务的延迟。具体公式如下。

$$\min e_i, DoS_i=1$$

$$\min(ET(t_{i,\text{out}})-ST(t_{i,\text{in}})), DoS_i=0$$

4 基于延迟接受的多用户任务卸载策略

从目标函数可以看出，多用户的部分任务卸载问题是带有约束条件的整数规划问题，该问题为 NP-hard 问题，解空间总数为 $(\sum_{i=1}^{N} L_i)^{M+2}$。其中，$\sum_{i=1}^{N} L_i$ 表示 N 个用户的任务数目之和，L_i 表示第 i 个用户的任务数目。当问题规模较大时，常规数学方法求解的复杂度过高。为了兼顾搜索的全局性和效率，本文提出了基于延迟接受的启发式多用户任务卸载策略(Multi User Task Offloading Based on Delayed Acceptance，MUTODA)，它通过不断的迭代得到近似最优解，迭代过程分为两步：1)单用户任务卸载策略(Single User Task Offloading，SUTO)；2)调整策略(Adjustment Strategy，AS)。实验结果表明，该算法具有更优的调度性能。

4.1 总体思路

博弈论稳定匹配的核心思想就是延迟接受和当前偏序最优。本文借鉴这个思想，联合文献[15]中卸载策略的思路，先求得单用户的最优策略，再用调整策略拒绝无法满足的用户，然后通过不断的迭代，获得所有用户的卸载决策。首先依次对所有用户运用 SUTO 算法，直到所有用户都得到最优调度解；接着，用调整策略对拥塞节点中的用户进行调整；然后，对拥塞队列中的用户重复上述步骤；直到无须进行调整，算法终止。整体算法的流程如图 3 所示。

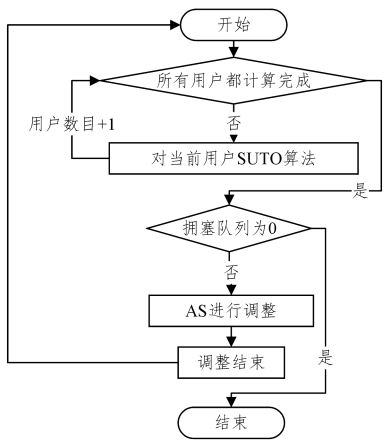


图 3 基于延迟接受的卸载策略流程

Fig. 3 Flow of offload strategy based on delayed acceptance

SUTO 以带精英策略的快速非支配排序遗传算法(Non-dominated Sorting Genetic Algorithm II，NSGA-II)为基础，利用其非支配排序方法对个体的适应度进行评估，通过选择、交叉和变异等遗传操作求得单用户情况下的最优卸载策略。NSGA-II 用快速非支配排序对每个个体进行评估，降低了算法的复杂度，加入了拥挤度的概念，在非支配排序后的同一非支配层中其作为筛选个体的标准[15]。最后，NSGA-II 引入精英策略，扩大采样空间，使父代中的优良品种可以进入下一代，保证种群朝着正确的方向发展。SUTO 算法的流程如图 4 所示。

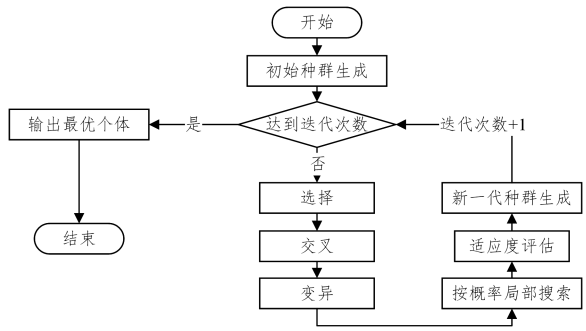


图 4 STUO 单用户任务卸载算法的流程

Fig. 4 Flow of STUO single user task offload algorithm

AS 对拥塞节点进行了调整，当出现拥塞节点时，对选择该节点的所有用户进行评估，排除评估结果最差的一个或多个用户任务，同时这些用户进入拥塞队列，重新进行最优化求解，不断进行迭代，直到拥塞队列中没有用户存在。AS 策略的流程如图 5 所示。

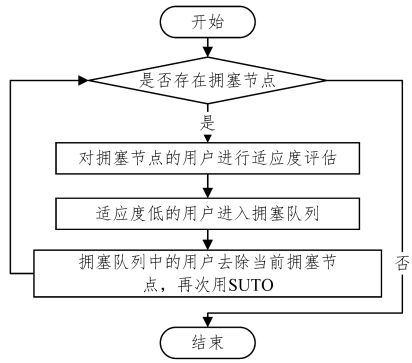


图 5 AS 调整策略的流程

Fig. 5 Flow of AS adjustment strategy

4.2 单用户任务卸载策略

4.2.1 个体编码与初始种群生成

在单用户任务卸载问题中，每个个体采用整数编码，编码长度为 L，即有 L 维分量，每个分量代表任务的卸载地点，共有 $M+2$ 种选择。分量为 0 表示任务在本地运行，分量为 $M+1$ 表示任务卸载到远程云，分量为 $1-M$ 表示任务卸载到对应编号的 MEC 服务器中。

图 6 给出了多用户任务卸载的一个场景，UE_1 有 4 个子任务，UE_2 有 5 个子任务，经过卸载决策决定了每个用户任务的卸载地点。从图中可以看出，对于 UE_1，任务 $t_{1,1}$ 在 MEC 服务器 M_1 中运行，$t_{1,1}=1$；任务 $t_{1,2}$ 在 MEC 服务器 M_3 中运行，$t_{1,2}=3$；任务 $t_{1,3}$ 经服务节点 M_1 转发到 MEC 服务器 M_2 中运行，$t_{1,3}=2$；任务 $t_{1,4}$ 在本地运行，$t_{1,4}=0$。对于 UE_2，任务 $t_{2,1}$ 在本地运行，$t_{2,1}=0$；任务 $t_{2,2}$ 和 $t_{2,3}$ 在 MEC 服务器 M_2 中运行，$t_{2,2}=t_{2,3}=2$；任务 $t_{2,4}$ 在 MEC 服务器 M_1 中运行，$t_{2,4}=1$；任务

$t_{2,5}$在 MEC 服务器M_3中运行，$t_{2,5}=3$。图 7 给出了任务卸载策略对应的个体编码。

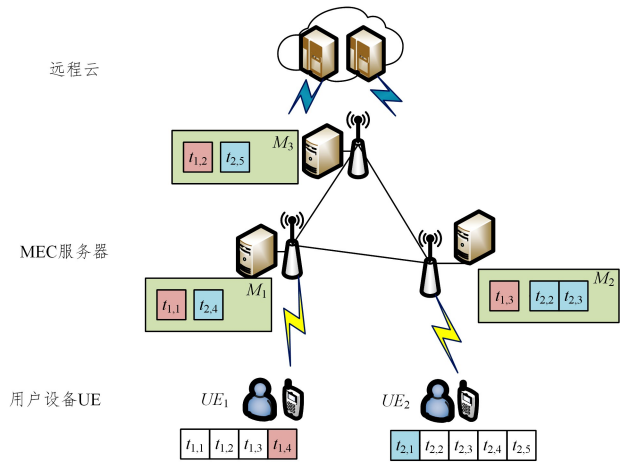


图 6 多用户任务卸载示例

Fig. 6 Multi-user task uninstall example

0	2	2	1	3
$t_{2,1}$	$t_{2,2}$	$t_{2,3}$	$t_{2,4}$	$t_{2,5}$

图 7 个体编码

Fig. 7 Individual code

遗传算法的目标是使种群进化到最优种群或者接近最优种群。作为迭代进化的起点，初始种群的质量对算法的性能影响很大。在初始种群均匀覆盖良好的情况下，种群中的个体不聚集，并且可以扩散到一个相对较大的解空间，因此在这种情况下得到较好解的概率较大。如果初始种群是完全随机产生的，那么种群的质量就不能得到有效保证。因此，在初始种群生成过程中，不仅需要加入随机个体，还可以适当加入已初筛过的个体，以保证种群朝着目标方向进化，并且可以加快种群的收敛速度。

设计具体的初始化种群的过程如下：首先，通过基于最早完成时间的启发式方法（Earliest Finish Algorithm，EFA）形成任务分配结果，并将其加入初始种群中；然后，通过个体随机算法（Individual Random Algorithm，IRA）生成任务分配结果，并将其加入初始种群中；不断重复以上这两步，直到种群中个体数量达到设定规模 populationSize。

算法 1 Population initialization 算法

输入：populationSize

输出：种群 population

```
1. for each i=1 to populationSize do
2.     x_i=EFA()
3.     x_{i+1}=IRA()
4.     Add x_i and x_{i+1} to population
5. end for
6. paretoOptimality=population.get(0)
7. return population
```

4.2.2 帕雷托解与适应度评估

本文在任务卸载过程中考虑了多个目标，多目标优化与单目标优化有很大的区别。对于单目标，若某个解比其他的解都好，那么该解是最优解，通常是全局最大值或最小值。对于多目标优化问题，通常存在一个解集，这些解相互之间无法比较，它们的特点是可以通过减少某个函数值来改进其他的目标函数，这种解称作非支配解或帕雷托最优解。

定义 8（帕雷托支配关系） 对于最小化多目标问题，n 个目标分量$f_i(i=1,\cdots,n)$组成的向量为$\overline{\boldsymbol{f}}(\overline{X})=(f_1(\overline{X}),f_2(\overline{X}),\cdots,f_n(\overline{X}))$，给定任意两个决策变量$\overline{X}_u,\overline{X}_v\in U$：

当且仅当对于$\forall i\in\{1,\cdots,n\}$，都有$f_i(\overline{X}_u)<f_i(\overline{X}_v)$，则$\overline{X}_u$支配$\overline{X}_v$。

当且仅当对于$\forall i\in\{1,\cdots,n\}$，有$f_i(\overline{X}_u)\leqslant f_i(\overline{X}_v)$，且至少存在一个$j\in\{1,\cdots,n\}$，使得$f_j(\overline{X}_u)<f_j(\overline{X}_v)$，则$\overline{X}_u$弱支配$\overline{X}_v$。本文将支配与弱支配统一称为支配关系。

定义 9（帕雷托最优解） 对于最小化多目标问题，n 个目标分量$f_i(i=1,\cdots,n)$组成的向量为$\overline{\boldsymbol{f}}(\overline{X})=(f_1(\overline{X}),f_2(\overline{X}),\cdots,f_n(\overline{X}))$，$\overline{X}_u\in U$ 为决策变量，若$\overline{X}_u$为帕雷托最优解，则满足：

当且仅当不存在$\overline{X}_v\in U$，$v=f(\overline{X}_v)=(v_1,\cdots,v_n)$支配$u=f(\overline{X}_u)=(u_1,\cdots,u_n)$。

本文中的目标函数有两个：时延和能耗。在适应度评估中采用 NSGA-II 中的非支配排序作为个体适应度的评估方法。非支配排序根据个体的非劣解水平对种群进行分层，其作用是引导搜索向帕雷托最优解集的方向进行。它是一个循环的非支配值分层过程，首先找出当前种群中非支配值最小的个体集，记为第一层；然后，找寻剩余种群中非支配值最小的个体集，记为第二层；不断重复，直到所有个体全部分层完成。在同一层中，引入拥挤度的概念，拥挤度表示同层个体之间在解的空间上聚集的程度，拥挤度越大，个体的多样性就越好。这里，个体x_i的拥挤度为个体x_{i-1}和x_{i+1}之间的欧氏距离。在拥挤度计算完成之后，对同层个体进行拥挤度降序排序，拥挤度大的个体被选择进入下一代的概率越高。

适应度评估过程为：首先，对 population 使用 NSGA-II 所提出的方法进行非支配排序（Non Dominated Ordering，NDO）（见算法 2），生成排序后的新种群 population′；接着，淘汰父代中不可行的个体，即删除分配任务中不符合时延要求的个体；然后，从 population′中选出支配层最低拥挤度最大的个体x_0，用x_0更新帕雷托最优个体 paretoOptimality，此处的最优个体x_0则是符合时延情况下的能耗最小的个体；最后，算法返回非支配排序后的种群 population′和更新后的 paretoOptimality。适应度评估算法如算法 3 所示。

算法 2 nonDominatedOrdering 算法

输入：种群大小 populationSize

输出：非支配排序后的新种群 population′$=\{F_1,F_2,\cdots,F_n\}$

```
1. for each p∈populationSize
2.   S_p=∅//被 p 支配的个体集合
3.   n_p=0 //支配 p 的个体数目
4.     for each q∈populationSize
5.     if p≺q then // ≺表示支配关系
6.   S_p=S_p∪{q}
7.     else if q≺p then
8.       n_p=n_p+1
9.     if n_p=0 then
10.      F_1=F_1∪{p}
11. i=1 //初始化非支配解层级计数器
12. while F_i≠∅
```

```
13. Q=Ø//用来存储下一级的非支配解
14.   for each p∈F_i
15.     for each q∈S_p
16.       n_q=n_q-1
17.       if n_q=0 then
18.         Q=Q∪{q}
19.   i=i+1
20. F_i=Q
```

算法 3 Population evaluation 算法

```
输入:populationSize
输出:评估后的新种群 population,帕雷托最优个体 paretoOptimality
1. population'=nonDominatedOrdering(population)
2. delete(population')
3. x_0=getParetoBest (population')
4. paretoOptimality=x_0
5. return population'
```

4.2.3 选择操作

选择操作是对种群个体进行优胜劣汰的过程，适应度高的个体被选择进入下一代群体中的概率大，否则遗传概率小，在选择阶段采用 NSGA-II 中的精英策略选择算法。首先，对父代以及子代组合成的种群进行非支配排序和个体拥挤度计算，排除其中不可行的个体；接着，按照非支配排序从低到高的顺序，将整层种群依次放入新种群中，直到放入某一层时出现种群大小超出种群规模 1/2 的情况；然后根据个体拥挤度由大到小的顺序继续放入新种群，直到种群数量达到 1/2 的规模；当无法达到种群规模的 1/2 时，采用锦标赛法(Tournament)实现选择操作，具体步骤为从种群中随机选择 a 个个体(每个个体被选择的概率相同)，根据每个个体的适应度值，选择其中适应度值最好的个体进入下一代种群；重复该操作，直到新的种群规模达到预设值。

算法 4 Population selection 算法

```
输入:populationSize,population
输出:新种群 newPopulation
1. population'=nonDominatedOrdering(population)
2. c=0 //记录非支配层
3. for each i=1 to R do //R 为非支配总层数
4.   if Size(newPopulation+F_i)<populationSize/2
5.     Add F_i to newPopulation
6.   else c=i //记录当前非支配层
7.   break //跳出循环
8.   endif
9. endfor
10. congestionRanking(F_c) //c 层拥挤度降序排序
11. for each t=1 to Size(F_c) do
12.   if Size(newPopulation)<populationSize/2
13.     x_t=getFrom(F_c) //获得拥挤度最低的个体
14.     Add x_t to newPopulation
15.   end if
16. end for
17. if Size(newPopulation)<populationSize/2
18.   x_t=tournamentAlgorithm()//锦标赛算法
19.   Add x_t to newPopulation
20. end if
21. return newPopulation
```

4.2.4 交叉和变异操作

基因重组和基因变异是生物学中保证物种多样性的两种方法，而交叉操作和变异操作分别是模仿这两个过程的遗传算法。交叉操作不仅可以保证父代中优良个体的基因可以遗传给下一代，还可以保证物种的多样性，具体操作如下：首先，确定父亲与母亲两个父代个体；接着，随机选择一个交叉点；然后，父亲交叉点的前段基因遗传给儿子，母亲交叉点的后段基因遗传给儿子，这样就形成了一个完整的新个体；最后，母亲交叉点的前段基因遗传给女儿，父亲交叉点的后段基因遗传给女儿，形成另一个新个体。

变异操作使得新生后代的基因可能会产生突变，突变基因不同于父代的基因，大大提高了种群的多样性。具体操作如下：对于每一个新生后代，首先选择一个基因位，然后按照变异概率来进行基因位的突变，其余位上的基因保持不变。

4.3 调整策略

当某个 MEC 服务器资源比较充裕时，其很有可能成为大多数用户的选择对象，然而过多的计算任务会超出 MEC 服务器的最大资源限制，称这样的 MEC 服务器为拥塞节点。针对拥塞节点，本文提出了一种基于稳定匹配中延迟接受思想的调整策略。

首先，对拥塞节点的任务按照用户进行分类并根据用户标识进行排序；接着，按照权值从大到小排序，依次剔除用户在该拥塞节点的任务，直到总体计算任务不会超过该节点的计算能力限制；然后，被剔除的用户进入拥塞队列，拥塞队列中的用户依次重新计算当前计算节点之外的最优卸载决策；最后，重复此过程，直到拥塞队列中没有用户存在。用户 i 在计算节点 k 上的权值 $\omega_{k,i}$ 为：

$$\omega_{k,i}=\omega_1 \cdot delay_i+\omega_2 \cdot \sum_{i=1}^{num_{i,k}} cycles_{i,j}$$
$$t_{i,j}=k \tag{11}$$

其中，ω_1 和 ω_2 为系数，$\sum_{i=1}^{num_{i,k}} cycles_{i,j}$ 表示用户 i 在节点 k 上所有任务的计算周期数。算法 5 给出了节点调整策略的过程。

算法 5 节点调整策略

```
输入:ω_1,ω_2,L_i,t_i,j,R_k
输出:拥塞队列 congestionQueue
1. for each i=0 to R do
2.   ω_k,i=userWeight(ω_1,ω_2,L_i,t_i,j)
3. Add ω_k,i to weightQueue
4. end for
5.   weightRanking(weightQueue)
6. for each i=0 to R do
7.     if sizeCycles(allTask)>R_k then
8. Delete(T_i)
9.       Add T_i to congestionQueue
10.      updateNode(T_i)
11.  end if
12.  end for
13. return congestionQueue
```

5 实验验证

5.1 实验设置

(1)实验参数设置

本文通过计算机仿真实验评估基于稳定匹配思想的启发式算法的计算卸载策略的性能，实验平台为 IDEA。为了构建仿真场景，实验随机生成任务的拓扑结构与边缘的 MEC 服务器，并将 UE 随机分配给各个 MEC 服务器。实验仿真参数[15,18-23]的设置如表 1 所列。

表 1 主要参数设置

Table 1 Main parameter setting

参数	参数取值
边缘服务器数量 K	20
用户任务数量 N	50
每个用户子任务的数量 L	7～13
任务输入数据规模 $data_i$/MB	0.1～2
子任务计算量 $cycles_i$/cycles	[40,100]M
容忍时延 $delay_i$/s	0.1～2
用户设备 CPU 处理能力 C_0/GHz	0.2～1
MEC 服务器处理能力 C_k/GHz	5～6
远程云 CPU 处理能力 C_R/GHz	15
MEC 服务器最大计算资源限制 R_k^{Max}/cycles	[4000,6000]M
最大带宽 B/MHz	20
阴影衰落系数速率 σ_c	10
用户设备传输功率 P_i^T/W	1
噪声功率 N_0/(dB/Hz)	−118.4
基站传输功率 P_k^T/W	10.1
用户设备计算功率 P_i^C/(J/cycles)	8.6×10^{-8}

(2)评估指标

本节将会从用户满意度、策略执行时间和用户总能耗 3 个方面进行评估。用户满意度表示能够在容忍时延内完成的用户任务数量与总用户任务数量的比值；策略执行时间表示每种卸载策略对各种信息进行综合处理，得出卸载决策计算所用的时间；用户总能耗表示执行卸载决策时所有用户所消耗的能量总和。

(3)对比实验

选择本地运行策略(Local Computing，LC)、远程云卸载策略(Remote Cloud，RC)、基于粒子群算法的卸载策略(Particle Swarm Optimization，PSO)以及基于遗传算法的启发式卸载策略(Genetic Algorithm，GA)进行对比实验，从用户满意度、策略执行时间和用户总能耗 3 个方面评估基于稳定匹配与遗传算法的启发式多用户任务卸载策略的有效性。

1)本地运行策略(Local Computing，LC)：用户子任务都在移动设备上运行。

2)远程云卸载策略(Remote Cloud，RC)：用户子任务全部卸载到远程云中心运行。

3)基于粒子群算法的卸载策略(Particle Swarm Optimization，PSO)：运用启发式的算法来最大化用户满意度。

4)基于遗传算法的启发式卸载策略(Genetic Algorithm，GA)：每个任务可以在本地、MEC 服务器或者远程云上进行计算，运用启发式的方法以最大化用户满意度为目标来求得较优解，这里采用启发式的遗传算法。

5)基于延迟接受的多用户任务卸载策略(MUTODA)：引用稳定匹配的思想，运用启发式算法求得每个用户的最优解，使用调整策略不断进行调整。

本文选择基于粒子群算法(PSO)的卸载策略作为本文实验的对比策略。选择该算法的原因是 PSO 与遗传算法有许多相似之处，例如两者都需要种群初始化，都使用适应值评价系统且根据适应值进行一定的搜索。同时，我们希望在对比相似度高的 PSO 算法上突出本文算法的优势。

粒子群算法原本是通过模拟鸟群觅食行为而发展起来的一种基于群体协作的随机搜索算法。该算法初始化为一群随机粒子(随机解)，然后通过迭代找到最优解，在每一次迭代中粒子通过跟踪两个“极值”来更新自己。第一个极值就是粒子本身找到的最优解，这个解为个体极值 pBest；另一个极值是整个种群找到的最优解，这个解是全局极值 gBest。另外，也可以不用整个种群而只用其中一部分最优粒子的邻居进行算法训练，那么在所有邻居中的极值就是局部极值。

5.2 实验结果

5.2.1 用户满意度

图 8 给出了不同卸载策略下任务数目对用户满意度的影响。可知，LC 和 RC 策略与用户任务数无关；随着用户任务数量的增多，PSO，GA 和 MUTODA 策略的用户满意度都呈现降低趋势。相比 PSO 策略，MUTODA 策略能够充分考虑每一个 MEC 服务器对每一位用户的影响，延迟接受的策略使得不满足需求的用户数目尽可能少，因此其用户满意度比 PSO 策略高出约 8%，而 GA 策略从整体上直接进行卸载任务的划分，没有求解单用户最优，也没有调整策略，其效果比 PSO 策略还要差一些。

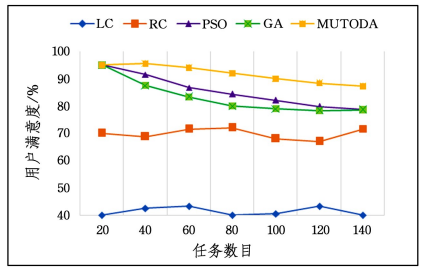
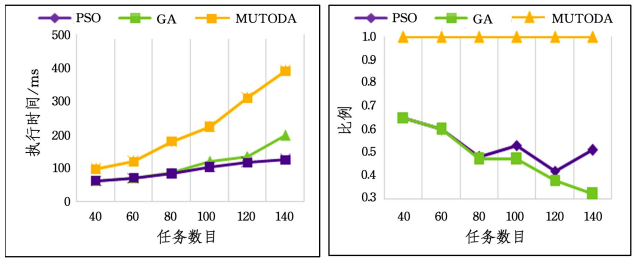


图 8 任务数目对用户满意度的影响

Fig. 8 Impact of number of tasks on user satisfaction

图 9(a)给出了用户数目对策略执行时间的影响，LC 和 RC 策略的执行时间为 0，不予考虑。随着用户数目的增长，PSO，GA 和 MUTODA 策略的执行时间呈现非线性的增长。由于执行时间与策略运行的设备计算能力相关，因此本文用比例关系来描述三者的变化。图 9(b)中的比例值表示某策略执行时间与 MUTODA 策略执行时间的比值。PSO 策略和 GA 策略的比例呈现减小趋势，说明虽然在用户满意度上 MUTODA 策略能够达到较高的水平，但是其执行时间较长。



(a) (b)

图 9 任务数目对策略执行时间的影响

Fig. 9 Impact of number of tasks on strategy execution time

图 10 显示了任务计算量对用户满意度的影响。RC 策略在远程云中计算任务，因此任务计算量对 RC 策略无明显影响。随着任务计算量的增加，LC，PSO 和 MUTODA 策略的用户满意度都呈现了下降趋势。此外，LC 和 PSO 策略的用户满意度明显低于 MUTODA 策略，这是由于 MUTODA 运用延迟接受的启发式方法，使得不满足用户满意度的任务数目趋向最少。

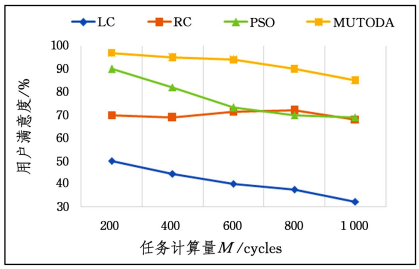


图 10 任务计算量对用户满意度的影响

Fig. 10 Effect of task calculation on user satisfaction

图 11 给出了 MEC 资源最大限制对用户满意度的影响。随着边缘服务器最大资源限制的增加，LC 和 RC 策略不受影响，而 PSO，GA 和 MUTODA 策略的用户满意度都以一定的幅度增加。当服务器容量超过 12 时，三者的用户满意度增长趋势非常小或者停止增加，因为边缘服务器也会受到计算能力、传输速率以及用户最大容忍时延等因素的限制。

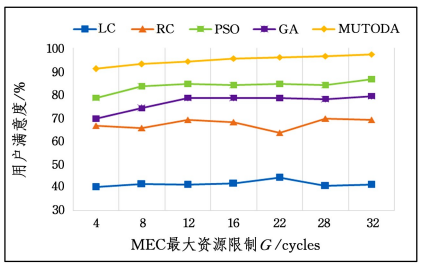


图 11 MEC 最大资源限制对用户满意度的影响

Fig. 11 Impact of MEC maximum resource limits on user satisfaction

5.2.2 能源消耗

用户总能耗表示执行卸载决策时所有用户所消耗的能量总和，能耗受到用户任务数量、传输数据规模等因素的影响。图 12 给出了数据规模对能耗的影响。

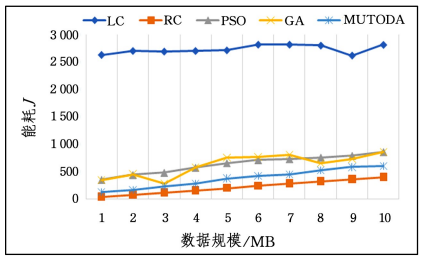


图 12 数据规模对能耗的影响

Fig. 12 Impact of data scale on energy consumption

由于 LC 策略在本地计算任务无需传输，因此能耗与数据规模无关。RC 策略虽然能够实现最小的能耗，但是用户满意度较低。随着数据规模的增大，PSO 和 MUTODA 策略的能耗都随之增大，GA 策略整体上呈增加趋势，原因是传输的数据量增多导致传输能耗增大。相比 PSO 策略，MUTODA 策略能保持较小的能耗，因为 MUTODA 策略寻求多个可行解中能耗近似最小的解，使得用户端的能耗较小。对比 PSO 策略和 GA 策略可知，GA 策略不如 PSO 稳定，GA 策略的能耗随数据规模变化而较大幅度地变化。

图 13 给出了任务数目对能耗的影响。可以看出，随着任务数目的增多，除了 GA 策略，其余所有策略的能耗都非线性增大。相比 PSO 策略，MUTODA 策略的能耗明显小很多，这是因为 PSO 容易陷入局部最优，搜索全局最优解的能力不如 MUTODA 策略。MUTODA 策略在满足用户容忍时延的要求下，通过非支配遗传算法的评判选择寻求能耗最小的解，使得终端用户的能耗最小，实现用户设备的长时间工作。

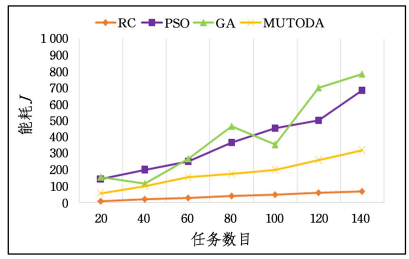


图 13 任务数目对能耗的影响

Fig. 13 Impact of number of tasks on energy consumption

上述实验结果表明，本文提出的基于延迟接受思想的多用户任务卸载策略能够实现最高约 95% 的用户满意度，而且可以有效减少终端能耗；同时，随着用户任务计算量的增大，用户满意度总体呈上升趋势。

结束语 本文针对依赖任务场景中多用户任务卸载的问题，在综合考虑时延与能耗的基础上建立细粒度的任务卸载模型，提出了基于延迟接受的多用户任务卸载策略。本文通过非支配遗传算法，求解每个用户在单用户场景下的最优解；基于稳定匹配的延迟接受的思想，提出了调整策略；通过非支配遗传算法和调整策略的不断迭代，解决了多用户任务卸载问题。仿真实验结果表明，与其他卸载策略相比，基于延迟接受思想的多用户任务卸载策略能够完成最高 95% 的用户满意度，实现更多用户的需求，且能有效减少用户端的能耗，保证用户端设备长久工作，提高用户体验感。

参 考 文 献

[1] CUI Y, SONG J, MIAO C C, et al. Mobile Cloud Computing Reasearch Progress and Trends[J]. Chinese Journal of Computers, 2017, 40(2): 273-295.

[2] PATEL M, NAUGHTON B, CHAN C, et al. Mobile-edge computing introductory technical white paper[M]. Mobile-edge Computing (MEC) Industry Mnitiative, 2014: 1089-7801.

[3] SIEGEL J E, ERB D C, SARMA S E. A survey of the connected vehicle landscape Architecture, enabling technologies, applications, and development areas[J]. IEEE Transactions on Intelligent Transportation Systems, 2017, 19(8): 2391-2406.

[4] SABELLA D, VAILLANT A, KUURE P, et al. Mobile-edge computing architecture: The role of MEC in the Internet of Things[J]. IEEE Consumer Electronics Magazine, 2016, 5(4): 84-91.

[5] WANG Y, GE H B, FENG A Q. Computation Offloading Strate-

gy in Cloud-Assisted Mobile Edge Computing[J]. Computer Engineering,2020,46(8):27-34.

[6] LIU J,MAO Y,ZHANG J,et al. Delay-optimal computation task scheduling for mobile-edge computing systems[C]// 2016 IEEE International Symposium on Information Theory (ISIT). IEEE,2016.

[7] WU Y,QIAN L P,NI K,et al. Delay-Minimization Nonorthogonal Multiple Access Enabled Multi-User Mobile Edge Computation Offloading[J]. IEEE Journal of Selected Topics in Signal Processing,2019,13(3):392-407.

[8] XU J,PALANISAMY B,LUDWIG H,et al. Zenith:Utility-aware Resource Allocation for Edge Computing[C]// IEEE International Conference on Edge Computing. IEEE Computer Society,2017.

[9] WANG Y,SHENG M,WANG X,et al. Mobile-Edge Computing:Partial Computation Offloading Using Dynamic Voltage Scaling[J]. IEEE Transactions on Communications,2016,64(10):4268-4282.

[10] YU X,SHI X Q,LIU Y X. Joint Optimization of Offloading Strategy and Power in Mobile-Edge Computing[J]. Computer Engineering,2020,46(6):20-25.

[11] SARDELLITTI S,SCUTARI G,BARBAROSSA S. Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing[J]. IEEE Transactions on Signal and Information Processing over Networks,2015,1(2):89-103.

[12] MAO Y,ZHANG J,SONG S H,et al. Power-Delay Tradeoff in Multi-User Mobile-Edge Computing Systems[C]// Globecom IEEE Global Communications Conference. IEEE,2016.

[13] WANG W,ZHOU W. Computational offloading with delay and capacity constraints in mobile edge[C]// ICC IEEE International Conference on Communications. IEEE,2017.

[14] MENG-HSI C,BEN L,MIN D. Multi-user Multi-Task Offloading and Resource Allocation in Mobile Cloud Systems[J]. IEEE Transactions on Wireless Communications,2018,17(10):6790-6805.

[15] NING Z,PEIRAN D,KONG X,et al. A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things[J]. IEEE Internet of Things Journal,2018,6(3):4804-4814.

[16] GUO S,LIU J,YANGY,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.

[17] FANG H S. Reasearch of Elitist NSGA and Its Application in Regional Water Resourch Optimal Allocation[D]. North University of China,2008.

[18] MUNOZ O,PASCUAL-ISERTE A,VIDAL J. Joint Allocation of Radio and Computational Resources in Wireless Application Offloading[C]// Future Network and Mobile Summit,2013. IEEE,2013.

[19] OUEIS J,STRINATI E C,BARBAROSSA S. Small cell clustering for efficient distributed cloud computing[C]// IEEE International Symposium on Personal. IEEE,2015.

[20] FAN W,LIU Y,TANG B,et al. Computation offloading based on cooperations of mobile edge computing-enabled base stations[J]. IEEE Access,2018,6:22622-22633.

[21] TAO X,OTA K,DONGM,et al. Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing[J]. IEEE Wireless Communications Letters,2017,6(6):774-777.

[22] KAN T Y,CHIANG Y,WEI H Y. Task offloading and resource allocation in mobile-edge computing system[C]// 2018 27th Wireless and Optical Communication Conference (WOCC). IEEE,2018:1-4.

[23] YU X,LIU Y X,SHI X Q,et al. Mobile Edge Computing Offloading Strategy Under Internet of Vehicles Scenario[J]. Computer Engineering,2020,46(11):29-34,41.

MAO Ying-chi, born in 1976, Ph.D, professor, is a senior member of China Computer Federation. Her main research interests include distributed computing and parallel processing, IoT, and edge intelligence computing.

ZHOU Tong, born in 1997, M.S. candidate. His main research interests include distributed computing, IoT and edge intelligence computing.