计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202110803
60(2): 371−385, 2023

移动边缘计算中基于云边端协同的任务卸载策略

张文柱　余静华
（西安建筑科技大学信息与控制工程学院　西安　710055）
（wzzhang@xauat.edu.cn）

Task Offloading Strategy in Mobile Edge Computing Based on Cloud-Edge-End Cooperation

Zhang Wenzhu and Yu Jinghua
（*College of Information and Control Engineering*, *Xi'an University of Architecture and Technology*, *Xi'an* 710055）

Abstract　In order to make use of limited computing resources to provide efficient computing services, a cloud-edge-end collaborative task offloading framework based on Docker is proposed In MEC (mobile edge computing) to solve the problems of multi-access MEC collaborative offloading and computing resource allocation. In order to improve the execution rate of tasks and the resource utilization of each node, preprocessing of tasks: Kahn algorithm added to the requirements of full rank matrix sets the execution sequence of output tasks in combination with the parallel calculation of tasks. Based on these, the task computing mathematical models of end, edge and cloud are established respectively, and the objective functions of the joint optimization system delay and energy consumption are designed by assigning weights. In order to solve the optimal offloading decision, the parameter of "full merit rate" and particle bee are introduced to propose APS (artificial particle swarm) algorithm as offloading decision algorithm. The experiments show that multi-task processing proves the effectiveness of APS algorithm. Compared with the five modes of local computing, edge computing, cloud computing, end-edge union computing and random processing, the low latency and low energy consumption of the proposed scheme proves its advantages in providing efficient services under multi-access conditions.

Key words　mobile edge computing; cloud-edge-end collaboration; task dependence; full merit rate; artificial particle swarm algorithm; delay and energy consumption

摘　要　在移动边缘计算（mobile edge computing, MEC）中，为了利用有限的计算资源提供高效的计算服务，提出一种基于 Docker 的云—边—端协同任务卸载框架，解决多接入 MEC 协同卸载、计算资源分配问题.为提高任务的执行速率和各节点资源利用率，对任务进行预处理，如在 Kahn 算法中加入行满秩矩阵要求并结合任务并行计算设定输出任务执行序列；分别建立端、边、云任务计算模型，分配权重设计联合优化系统延迟与能耗的目标函数；为求解最优卸载决策，引入“全优率”参数和粒子蜂设计人工粒子蜂群（artificial particle swarm, APS）算法作为卸载决策算法.实验表明，多任务处理证明了 APS 的有效性. 多接入条件下，相比于本地计算、边缘计算、云计算、端—边联合和随机处理 5 种模式，所提方案的低延时和低能耗表现证明了其提供高效服务的优越性.

收稿日期：2021−08−02；**修回日期**：2022−03−30
基金项目：国家自然科学基金项目（72071153）；陕西省重点研发计划项目（2021GY-066）；陕西省自然科学基础研究计划项目（2020JM-489）
This work was supported by the National Natural Science Foundation of China (72071153), the Key Research and Development Program of Shanxi Province (2021GY-066), and the Natural Science Basic Research Plan of Shanxi Province (2020JM-489).
通信作者：余静华（1173286484@qq.com）

关键词 移动边缘计算;云边端协同;任务依赖;全优率;人工粒子蜂群算法;延迟和能耗

中图法分类号 TP393

随着万物互联时代的到来,智能终端设备数量和其产生的数据量都急剧增长[1].Cisco 云预测,2021年全球范围内将有超过 500 亿个的终端设备,这些设备每年产生的计算型数据总量将超 84 ZB,高计算应用需求越来越多[2].移动终端设备因其有限的电量和计算能力不能满足人们日益增长的应用需求,因此,如何提供高计算能力服务和服务质量成为一个研究重点.移动云计算(mobile cloud computing, MCC)[3]通过将移动终端设备的计算任务转移到计算能力强大的云中心对任务进行集中处理[4],成为提供高计算能力的解决方案.但 MCC 有其固有的局限性[5]:云中心与远距离终端之间的传播时限使其无法高效处理终端设备产生的敏感时延性质的数据.因此,MCC 不适合服务于时延敏感的应用.

欧洲电信标准协会(ETSI)在 2014 年提出移动边缘计算模式[6].移动边缘计算是将计算和存储资源部署在移动网络边缘侧,为用户提供云计算能力、低时延和高带宽的网络服务[7],结合计算卸载技术很好地平衡了 MCC 的局限性.未来的无线系统中,小基站、平板电脑等计算能力与计算机服务器相当的设备都可作为边缘节点[8].

计算卸载技术[9]指终端设备将部分或全部计算任务交给边缘云或中心云处理,为移动终端设备在资源存储、计算性能以及能效等方面存在的不足提供解决方案,是移动边缘计算(mobile edge computing, MEC)的关键技术之一.其中,卸载决策是计算卸载技术的核心问题[10],卸载决策决定任务是否卸载、卸载什么以及卸载到哪里的问题[11].在多终端、多任务的复杂场景下,卸载决策的制定需要综合任务计算量、数据传输量、边缘云计算能力和资源利用率等诸多因素.传统的卸载方式依靠虚拟机实现,而研究表明[12],Docker 容器比虚拟机更轻量,节省了时间和能耗,它将任务程序代码、任务依赖项和环境概要文件打包到镜像副本中构建容器镜像,操作方便.在一个完备的执行架构中,根据任务的差异化制定最优的卸载决策能有效提高任务的执行效率,但依靠单一的终端设备计算、边缘计算和云计算都不能满足复杂场景的需求,不能同时平衡时延与能耗.因此,本文的目标是综合端—边—云的优势,实现以下目标:在最优化时延与能耗的情况下,将时延敏感的任务卸载至边缘节点计算,时延容忍的大数据量任务在云中心处理,时延容忍的小数据量任务在终端处理.本文的主要贡献有 3 个方面:

1)提出一个基于 Docker 的云—边—端协同的移动边缘计算框架,建立了任务依赖关系联合并行计算的任务执行序列,在并行计算设定上分别建立了本地计算、边缘计算和云计算的计算时间和能耗模型;

2)在多用户服务场景中,通过跨端—边—云的协作方式,制定任务最小计算时间和最低能耗的目标卸载决策,为实现减小系统计算时间的同时降低系统能耗的目标,设计了人工粒子蜂群(artificial particle swarm, APS)算法搜索最佳的卸载策略;

3)通过仿真实验,对比人工蜂群算法、粒子群算法和随机卸载算法验证了 APS 算法的性能;对比本地计算、边缘计算、云计算、本地—边缘联合计算和随机处理等计算模式,验证本系统的性能.

1 相关工作

近年来,随着 MEC 的应用和卸载决策的研究不断深入,网络环境复杂度的提高和终端设备数据量的增大所带来的挑战也越来越严峻.尤其对于移动边缘计算模型的建立和卸载策略的制定,要满足任务的执行条件,当可用节点较少或卸载决策制定因素不完善时,会产生高时延甚至处理失败.

一些特定应用背景下 MEC 系统模型和卸载决策的研究成果已经落地.文献 [13] 提出了一种面向多节点的自动驾驶边缘卸载框架来提供效率和隐私导向的自动驾驶卸载服务,在多边缘节点中进行卸载调度策略的实时求解.文献 [14] 针对智慧城市领域提出了在能耗和响应时间之间进行权衡的任务卸载优化策略,包括 2 个子目标的建模和启发式双目标优化算法.文献 [15] 针对智能家居场景中的 MEC,引入边缘节点的泛在感知能力,在保证数据安全的基础上实现边缘节点资源的分配.文献 [16] 研究双无人机辅助 MEC 系统的安全问题,通过抗干扰帮助地面终端设备计算卸载的任务,联合了优化通信资源、计算资源和无人机轨迹进行任务卸载.

从 MEC 应用可知,任务卸载模型和计算卸载决策是 MEC 中的研究热点,二者共同决定任务的执行位置与分配资源,主要研究方向有 3 个:时延、能耗、权衡时延与能耗.文献 [17] 提出一个任务卸载和异构

资源调度的联合优化模型，最小化用户的设备能耗、任务执行时延和成本来求解最优的任务卸载算法.文献 [18] 为了使时延敏感的处理任务在近终端用户执行，提出支持多个物联网服务提供商在一个通用 MEC 平台部署的兼容方案，在平台内引导物联网数据传输.文献 [19] 提出缓存辅助边缘计算的卸载决策制定与资源优化方案，通过建立最小化用户在任务执行时最高能耗值，得到理想的卸载决策集合与资源分配方案.文献 [20] 制定了在迁移能量预算的条件下尽量减少任务的平均完成时间的目标，设计了基于强化学习的分布式任务迁移算法.文献 [21] 考虑多用户的 MEC 系统以最小化终端能耗为目标，通过联合优化任务传输功率、局部计算和边缘计算能力来研究任务卸载问题.

然而，文献 [13−21] 研究都依赖单一的计算模式，在通信资源有限的条件下，会因通信链路堵塞造成更大的时延.因此提出本文第一个改进点:建立多终端、多任务的云—边—端协作任务执行模型，并加入任务计算结果传回终端设备的时延异步性因素；联合时延与能耗，通过实验为时延与能耗分配最优权重进行计算卸载，保证此模型具备实际意义.

对于任务卸载策略的研究大多处于粗粒度范畴，随着任务需求的提升，近年来也出现了考虑任务的差异化并对其进行预处理的研究，表现为将任务进行划分或建立任务依赖关系.文献 [22] 将一个任务划分为多个子任务卸载给多个 MEC 节点，通过预测每个候选 MEC 节点上每个任务的总处理时间决定子任务卸载位置.文献 [23−24] 研究单用户 MEC 系统、移动设备包含一个由多计算组件或具有依赖关系的任务组成的应用程序，根据依赖执行顺序计算出最优的任务卸载策略.文献 [25] 研究多用户任务卸载，为提高资源利用率，提出基于建立任务依赖关系的卸载调度器.文献 [26] 先研究 2 个设备间的任务依赖与相关性，然后扩展到复杂交互的多用户场景，实现最优的任务卸载策略和资源分配.文献 [27] 提出一种用于多移动应用的依赖任务卸载框架,将有依赖约束的计算密集型任务自适应地卸载到 MEC 和云.任务划分的研究适用于资源需求大、数据量大和数量偏少的任务集，而在密集复杂环境中划分任务会导致计算量呈指数增长，于系统服务无益.

文献 [22−27] 研究中建立任务依赖关系的方法大多面向单用户系统，通过传统的有向无环图（directed acyclic graph, DAG）进行任务拓扑排序，再扩展到多个设备的 DAG 调用. 在每个处理器仅执行 1 个任务的情况下多个任务面临等待，且调用多个 DAG 进行任务分配无法解决有依赖任务与无依赖任务的区别处理，造成不必要的时延浪费.基于文献 [22−27] 分析，提出本文的第 2 个改进点：1)将任务依赖关系与任务并行处理相结合，以最优最长链路拓扑排序的 Kahn 算法为基础，加入行满秩矩阵结果要求，使算法输出的任务依赖矩阵的行元素直接表明任务的优先级. 第一优先级任务为无依赖任务与依赖初始任务的集合，优先级则代表任务的执行顺序；2)再结合云—边协同架构的并行处理原则，将表征任务依赖的优先级矩阵抽象成任务执行序列，每个序列将同步执行，大大缩短任务的排序处理时延.

卸载策略求解过程属于 NP 难问题，MEC 研究中大多将其转化为多因素优化问题，近年来使用较多的求解方式是群体智能算法.文献 [28−29] 研究了面向智能物联网的 MEC 应用，采用遗传（genetic algorithm, GA）算法优化计算成本与能耗进而得到卸载决策.文献 [30−31] 为了减少多接入环境和高需求应用的能耗，利用模拟退火（simulated annealing, SA）算法全局寻优能力求解卸载决策，使复杂优化问题便于处理. 文献 [32] 结合灰狼和混合鲸鱼优化算法实现卸载决策的 3 个目标优化，并对比 GA 说明了混合算法在考虑多因素求解卸载决策时的优越性.文献 [33] 为预测多服务器 MEC 的自适应服务质量，采用人工蜂群（artificial bee colony, ABC）算法求解策略后再将各服务器结果进行整体对比选择任务执行节点.在多接入边缘协同环境中，应用最为广泛的是粒子群（particle swarm optimization, PSO）算法.文献 [34−35] 研究了多 MEC 服务器对多任务的卸载决策制定，采用 PSO 得到任务迁移决策和计算资源分配方案.文献 [36] 应用 PSO 作为卸载决策算法处理计算密集型和时延敏感型任务，且对比说明 PSO 相比于 GA 与 SA 的优越性. 但 PSO 在处理异构边缘节点和复杂约束优化问题时全局寻优收敛时间太长，因此需融入局部寻优能力强的算法平衡 PSO 的缺陷，如 ABC 是群体智能中最成功的局部寻优算法之一，ABC 与 PSO 的混合更有利于发挥群体智能算法在 MEC 中的作用.

基于文献 [28−36] 研究提出本文的第 3 个改进点为：结合 ABC 算法的局部寻优能力与 PSO 算法的全局寻优能力，组合成 APS 算法用于求解云—边—端协同框架下联合时延与能耗的卸载决策；从降低算法复杂度的角度出发，通过引入“全优率”参数进行迭代结果的适应度函数判断，决定个体蜂的类型与蜜源搜索方式. 表 1 为现有的适用于复杂约束的 PSO-

ABC 混合算法相关研究与本文方法的对比，进一步说明本文算法特点.

Table 1 Comparison of Researches on ABC-PSO Hybrid Algorithm and the Proposed Method

表 1 ABC-PSO 混合算法相关研究与本文方法对比

方法	混合方式与特点	有无参数	复杂度程度
文献 [37] 方法	将 PSO 寻优结果作为 ABC 的初始种群；后期收敛快，容易陷入局部极优.	无	一般
文献 [38] 方法	ABC 优化后的蜜源作为 PSO 搜索空间；前期易遗漏最优解.	无	一般
文献 [39] 方法	3 个 ABC 循环更新蜜源后用 PSO 寻优，保证蜜源优化，但循环更新耗时太长.	无	高
文献 [40] 方法	PSO 与 ABC 以是否产生侦查蜂交替更新；同步搜索无法保证最优解质量.	无	高
本文方法	ABC 搜索后对比蜜源和全局最优解；前后期互为参考与 PSO 交替转换.	有	低

2 系统架构

本文提出一个基于 Docker 的轻量级、虚拟化的云—边—端协同卸载架构，它包含 K 个云中心服务器、M 个边缘服务器和 L 个移动终端设备.边缘服务器部署在近移动终端设备侧的微型基站上；远程云中心服务器附加在宏基站上，其与边缘服务器相比，它的存储和计算能力更强大.设云中心集合表示为 $K=\{1,2,\cdots,k\}$, 边缘节点集合表示为 $M=\{1,2,\cdots,m\}$, 移动终端设备集合表示为 $L=\{1,2,\cdots,l\}$.同时假设每个终端设备与边缘节点和云中心相关联，移动设备与基站之间的链路集表示为 $R=\{1,2,\cdots,r\}$.

如图 1 所示，系统架构由卸载调度、卸载决策算法和任务执行部分(移动设备侧、边缘节点侧和云中心侧)组成.首先，移动终端设备生成任务集合并制定执行矩阵；其次，由卸载决策模块收集任务的执行要求和边缘节点的资源使用情况，通过卸载决策算法得到任务的执行方案；最终，将队列中的全部任务分别发送到本地计算任务集合、边缘节点，计算任务集合和云中心计算任务集合.经过卸载模块计算，无需卸载的任务在本地进行计算，需要卸载的任务通过卸载调度模块将任务程序代码、数据和 Dockerfile 文件发送至边缘节点或云中心，构建任务镜像并创建容器进行任务计算，计算完成后将结果返回原终端设备.

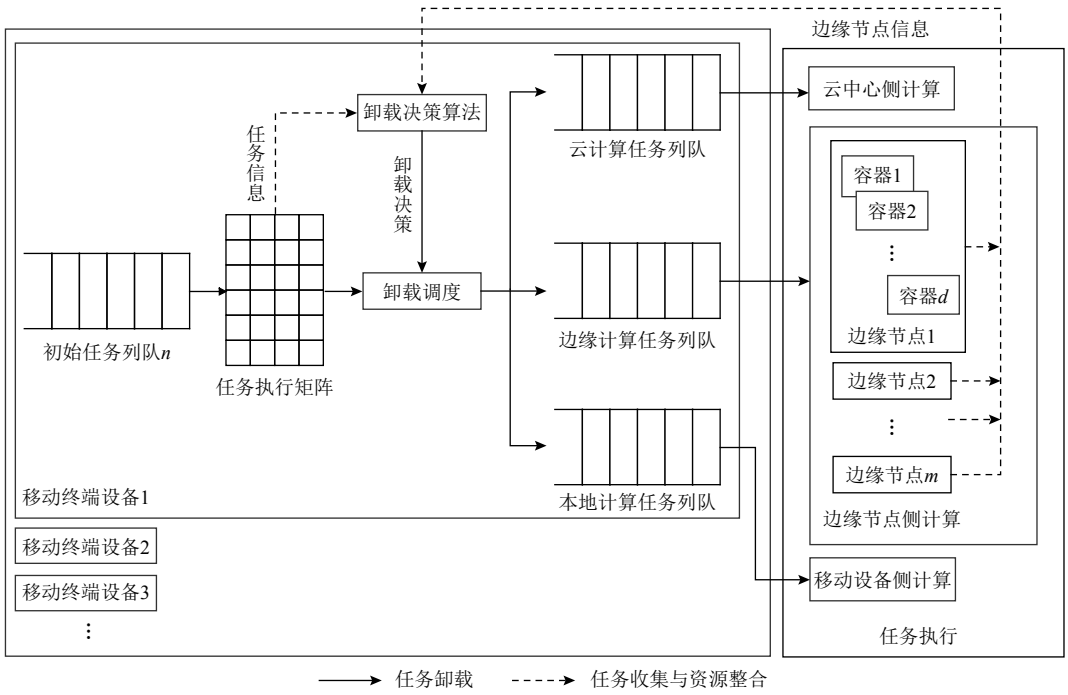

Fig. 1 Cloud-edge-end system architecture

图 1 云—边—端系统架构

3 系统模型和卸载决策制定

基于图 1 所示的多用户移动边缘计算场景，卸载决策模块为基站覆盖服务区的 N 个终端用户提供任务执行方案.每个终端用户都可以选择在本地计算任务，或将任务卸载至边缘节点或云中心执行.任务模型、本地计算模型、边缘计算模型、云计算模型在本节进行说明.

3.1 任务依赖模型与并行计算说明

3.1.1 任务模型

假设任务是完整的且不可再划分，移动终端设备产生 n 个待处理任务 $Task=\{T_1, T_2, \cdots, T_n\}$，每个任务 T_j 有 6 个基本属性：

$$T_j=\{w_j, f_{\min.j}, cpu_j, mem_j, data_j, t_{\max.j}\}. \quad (1)$$

其中：w_j 表示任务 j 的计算量，通过 CPU 周期度量；$f_{\min.j}$ 表示任务 j 需要的最小计算能力；cpu_j 和 mem_j 分别表示任务 j 执行时需要的 CPU 资源和内存资源；$data_j$ 表示任务 j 卸载时需要传输的数据量，包括过程代码和输入参数等信息；$t_{\max.j}$ 表示完成任务 j 允许的最大时延.

3.1.2 任务依赖模型

同一设备产生的不同任务通常具有一定的依赖关系，本文通过建立任务执行矩阵来提高处理任务效率.当任务生成后，将任务如何执行的初始状态抽象成 DAG；再将 DAG 进行拓扑排序生成任务执行矩阵；最后根据卸载策略判断任务的执行位置.本文设计的任务依赖关系算法以双维护集 Kahn 拓扑算法为基础，融入行满秩矩阵要求，在保证最优且唯一的同时避免出现分支多组合路径.为与任务并行计算设定相结合，此算法的目标是在多种拓扑顺序的情况下选出多行少列的依赖关系矩阵结果作为任务调度方案，最大限度确保有依赖任务有序执行，无依赖任务快速执行.任务依赖矩阵结果同时也是任务优先等级的划分结果，每一行向下递减一个优先级.执行伪代码如算法 1.

算法 1 .任务依赖关系算法 /*对输入的有向无环图生成拓扑排序后的执行矩阵 $\boldsymbol{L}$（该矩阵中行向量线性无关）.

输入：有向无环图 G, $G=(V, E)$, V 表示任务集，E 表示任务之间依赖关系链路集；

输出：执行矩阵 $\boldsymbol{L}$.

① 建立可放置已排序任务的空矩阵 $\boldsymbol{L}$；

② 将没有入度链路的任务放置在集合 S 中，$S=\{S_1, S_2, \cdots, S_n\}$, S_n 代表循环中第 n 轮判断无入度链路的任务；

③ while $S \neq \varnothing$

④ 将 S_n 中任务放进 $\boldsymbol{L}$ 第 n 行 (从 S_1 开始顺序执行)；

⑤ 从 S_n 中删去任务 v_1；

⑥ for 每个通过一条链路 e 从任务 v_1 到达任务 v_2

⑦ 从 E 中删去链路 e；

⑧ if 任务 v_2 没有其他的入度链路

⑨ 将任务 v_2 放入 S_{n+1} 中；

⑩ end if

⑪ end for

⑫ end while

⑬ if E 中存在任意一个循环

⑭ return 错误；

⑮ else

⑯ return $\boldsymbol{L}$;

⑰ end if

任务之间的初始状态关系如图 2 所示，根据拓扑算法建立执行矩阵式(2)，假设由卸载决策算法得到任务的决策矩阵式(3)，其中，“1”表示任务在终端设备执行，“−1”表示任务在边缘节点执行，“0”表示任务在云中心执行.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & & 14 & 10 \\ 12 & & & 11 \\ 15 & 16 & & 13 \\ & & & 17 \\ 18 & & & \end{pmatrix}, \quad (2)$$

$$\begin{pmatrix} 1 & -1 & -1 & 1 \\ -1 & -1 & 0 & 1 \\ -1 & & -1 & 1 \\ -1 & & & -1 \\ 1 & -1 & & 0 \\ & & & -1 \\ -1 & & & \end{pmatrix} \quad (3)$$

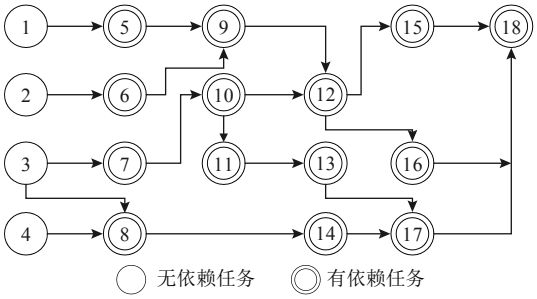

Fig. 2 Initial state relationship among tasks

图 2 任务之间的初始状态关系

3.1.3 任务并行计算说明

采用 Docker 封装卸载的形式使并行计算成为可能.当边缘节点 i 满足任务 j 需求的存储资源但不满足任务 j 的 CPU 计算资源需求时，结合时延与能耗，判断是否等待边缘节点 i 计算完 1 个或若干任务后释放 CPU 资源以达到任务需求，若合适仍将任务卸载到此边缘节点.任务到达边缘节点 i 后先储存到内存中，等待满足 CPU 资源后参与计算.并行计算过程

如图 3 所示:

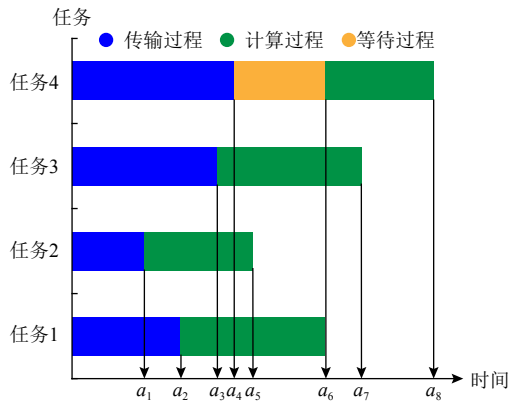

Fig. 3 Parallel computing process of edge node

图 3 边缘节点并行计算过程

图 3 中 a_1, a_2, a_3, a_4 分别为任务 1~4 抵达节点 i 的时间.假设任务 1~3 抵达节点 i 时, 节点可用内存资源和计算资源大于任务所需资源, 则任务 1~3 立即执行; 假设任务 4 抵达节点时节点可用计算资源小于任务所需计算资源, 则任务 4 不能立即执行.对比任务 1~3 的计算时间, 任务 2 最先计算完毕, 其次任务 1 完成计算; 假设任务 1 释放 CPU 计算资源后该节点满足任务 4 的计算需求, 则任务 4 开始执行, 时间 a_6 为任务 1 的完成时间.我们将考虑任务之间的并行依赖关系后制定的卸载决策表示为 $ST=\{st_1, st_2, \cdots, st_n\}$, st_j 表示任务 j 的执行地点.

任务并行计算的重要基础是任务依赖矩阵的生成, 矩阵行向量传达了任务的优先级执行顺序, 行向量之间的无依赖关系任务就可以依据资源情况在相关节点并行计算.当设备数量与任务数量较多时, 结合任务依赖于并行计算, 实现同一时间计算出不同设备上同一优先级的任务结果, 真正将多维分散无序计算变为 $m\times1$ 维有序执行.

$$\begin{pmatrix} \text{优先级1} \\ \text{优先级2} \\ \vdots \\ \text{优先级}m \end{pmatrix} \begin{pmatrix} a & b & \cdots & c & d \\ e & f & \cdots & g & h \\ i & & \cdots & j & k \\ \vdots & \vdots & & \vdots & \vdots \\ l & m & \cdots & & n \\ o & & \cdots & p & q \\ r & s & \cdots & t & \end{pmatrix}_{m\times n} \xrightarrow{\text{并行计算}} \begin{pmatrix} \text{计算序列1} \\ \text{计算序列2} \\ \vdots \\ \text{计算序列}m \end{pmatrix}_{m\times1}.$$

3.2 本地计算模型

任务在终端设备进行计算的时延如式(4).其中 f_l 表示终端设备的计算能力, cpu_l 表示终端设备可用 CPU 资源, mem_l 表示终端设备可用内存资源.任务在终端设备进行计算的能耗为 E^{ED}, $\sigma_1\times f_l^2$ 表示 CPU 循环 1 个周期所产生的能耗, σ_1 表示由芯片结构而定的开关电容.

$$T^{\mathrm{ED}}=\frac{w_j}{f_l},$$
$$\text{s.t.}\begin{cases} f_l > f_{\min.j}, \\ T^{\mathrm{ED}} < t_{\max.j}, \\ cpu_j < cpu_l, \\ mem_j < mem_l, \end{cases} \tag{4}$$
$$E^{\mathrm{ED}}=\sigma_1\times f_l^2\times w_j. \tag{5}$$

3.3 边缘节点计算模型

本系统中 M 个边缘服务器的集合为 $Edge=\{Ed_1, Ed_2, \cdots, Ed_m\}$, 每个边缘服务器的基本属性如式(6)所示.$f_i$ 表示边缘节点 i 的最大计算能力, cpu_i 表示边缘节点 i 的可用 CPU 资源, mem_i 表示边缘节点 i 的可用内存资源, R_i 是边缘节点与移动设备间的数据传输速度, P_i 为边缘节点 i 的发射功率.

$$Ed_i=\{f_i, cpu_i, mem_i, R_i, P_i\}. \tag{6}$$

考虑任务卸载至边缘节点时存在并行计算的情况, 假设模型中并行计算等待过程的能耗忽略不计, 只讨论产生的等待延时, 则任务计算完成并将结果返回移动终端设备的总时延 T^{EN} 计算存在 2 种情况, 如式(7)所示:

$$T^{\mathrm{EN}}=\begin{cases} T^{\mathrm{trans}}_{\mathrm{ED}\to\mathrm{EN}}+T^{\mathrm{compu}}_{\mathrm{EN}}+T^{\mathrm{trans}}_{\mathrm{EN}\to\mathrm{ED}}, \\ \text{s.t. } cpu_j\leqslant cpu_i, mem_j\leqslant mem_i; \\ T^{\mathrm{trans}}_{\mathrm{ED}\to\mathrm{EN}}+T^{\mathrm{compu}}_{\mathrm{EN}}+T^{\mathrm{wait}}+T^{\mathrm{trans}}_{\mathrm{EN}\to\mathrm{ED}}, \\ \text{s.t. } cpu_j > cpu_i, mem_j > mem_i. \end{cases} \tag{7}$$

其中 $T^{\mathrm{trans}}_{\mathrm{ED}\to\mathrm{EN}}$ 表示任务 j 由移动设备卸载到边缘节点 i 的传输时间, $T^{\mathrm{compu}}_{\mathrm{EN}}$ 表示边缘节点 i 计算任务的时间, T^{wait} 表示并行计算时任务的等待时间, $T^{\mathrm{trans}}_{\mathrm{EN}\to\mathrm{ED}}$ 表示任务结果由边缘节点 i 传输回移动终端的时间.

1)任务 j 抵达边缘节点 i 后, 若边缘节点 i 当前剩余资源满足任务计算所用资源, 即 $cpu_j\leqslant cpu_i$, $mem_j\leqslant mem_i$.此时任务 j 可立即执行, 其完成时间如式(8)所示:

$$T^{\mathrm{EN}}=T^{\mathrm{trans}}_{\mathrm{ED}\to\mathrm{EN}}+T^{\mathrm{compu}}_{\mathrm{EN}}+T^{\mathrm{trans}}_{\mathrm{EN}\to\mathrm{ED}},$$
$$\text{s.t.}\begin{cases} f_i > f_{\min.j}, \\ T^{\mathrm{EN}} < t_{\max.j}, \\ cpu_j\leqslant cpu_i, \\ mem_j\leqslant mem_i. \end{cases} \tag{8}$$

$T^{\mathrm{trans}}_{\mathrm{ED}\to\mathrm{EN}}$ 的计算方式如式(9)所示, 假设移动设备与边缘服务器之间有 q 条链路, 且每条链路只允许有 1 台设备占用.由香农公式式(10)得 R_i, 其中, W 为

信道之间带宽，P_l 表示移动设备的发射功率，τ^2表示噪声功率，I_q 为链路的干扰功率.

$$T_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}=\frac{data_j}{R_i}, \tag{9}$$

$$R_i=W\times\mathrm{lb}\left(1+\frac{P_l}{\tau^2+I_q}\right). \tag{10}$$

$T_{\mathrm{EN}}^{\mathrm{compu}}$和$T_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}$的计算方式分别为式(11)和式(12)，其中$data_{\mathrm{re}}^{\mathrm{EN}}$表示边缘节点任务计算的结果数据.

$$T_{\mathrm{EN}}^{\mathrm{compu}}=\frac{w_j}{f_i}, \tag{11}$$

$$T_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}=\frac{data_{\mathrm{re}}^{\mathrm{EN}}}{R_i}. \tag{12}$$

任务在边缘节点进行计算的能耗如式(13)所示，$E_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}$表示任务 j 由移动设备卸载到边缘节点i的能耗，$E_{\mathrm{EN}}^{\mathrm{compu}}$为边缘节点 i 计算任务j的能耗，$E_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}$表示边缘节点 i 计算任务 j 得到的结果返回终端设备的能耗，以上 3 部分的能耗计算方式分别为式(14)、式(15)和式(16).

$$E^{\mathrm{EN}}=E_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}+E_{\mathrm{EN}}^{\mathrm{compu}}+E_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}, \tag{13}$$

$$E_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}=P_l\times T_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}, \tag{14}$$

$$E_{\mathrm{EN}}^{\mathrm{compu}}=\sigma_2\times f_i^2\times w_j, \tag{15}$$

$$E_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}=P_i\times T_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}. \tag{16}$$

2)任务 j 抵达边缘节点 i 后，若边缘节点i当前剩余资源小于任务计算所用资源，即 $cpu_j>cpu_i$，$mem_j>mem_i$.此时任务 j 需要等待此节点执行完优先级高的任务并释放足够的计算资源后方可参与计算，其完成时间为式(17).等待时间可根据已建立的任务执行矩阵得式(18).

$$T^{\mathrm{EN}}=T_{\mathrm{ED}\to\mathrm{EN}}^{\mathrm{trans}}+T_{\mathrm{EN}}^{\mathrm{compu}}+T^{\mathrm{wait}}+T_{\mathrm{EN}\to\mathrm{ED}}^{\mathrm{trans}}, \tag{17}$$

$$T^{\mathrm{wait}}=T_{front\in n}^{\mathrm{compu}}+\left|T_{front}^{\mathrm{trans}}-T_{behind}^{\mathrm{trans}}\right|. \tag{18}$$

$T_{front\in n}^{\mathrm{compu}}$表示当前任务之前到达节点$i$的任务的计算时间；$T_{front}^{\mathrm{trans}}$表示在当前任务之前到达节点$i$的任务的传输时间；$T_{behind}^{\mathrm{trans}}$表示当前任务之后到达节点$i$的任务的传输时间.

3.4 云中心计算模型

本系统中部署 K 个云服务器，因为云中心的CPU资源和内存资源远高于终端设备和边缘节点，本文仅考虑云中心的 3 个属性.其中，f_k 表示云中心的最大计算能力，R_k 为云中心与移动设备间的数据传输速度，P_k 表示云中心的发射功率.

$$C_k=\{f_k,R_k,P_k\}. \tag{19}$$

任务卸载到云中心的时延为式(20)中的T^{C}，式(21)中$T_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}$表示任务 j 由移动设备卸载到云中心的时间，式(22)中$T_{\mathrm{C}}^{\mathrm{compu}}$表示任务 j 在云中心的执行时间，式(23)中$T_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}$为任务 j 在云中心计算的结果数据返回终端设备的时间.其中$data_{\mathrm{re}}^{\mathrm{C}}$为云中心计算任务的结果数据.

$$\begin{aligned}&T^{\mathrm{C}}=T_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}+T_{\mathrm{C}}^{\mathrm{compu}}+T_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}},\\&\text{s.t. } T^{\mathrm{C}}<t_{\max.j}.\end{aligned} \tag{20}$$

$$T_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}=\frac{data_j}{R_k}, \tag{21}$$

$$T_{\mathrm{C}}^{\mathrm{compu}}=\frac{w_j}{f_k}, \tag{22}$$

$$T_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}=\frac{data_{\mathrm{re}}^{\mathrm{C}}}{R_k}. \tag{23}$$

云中心的能耗E^{C}计算公式如式(24)所示，时延对应的卸载传输能耗$E_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}$、执行能耗$E_{\mathrm{C}}^{\mathrm{compu}}$和回程传输能耗$E_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}$计算公式分别为式(25)、式(26)和式(27)

$$E^{\mathrm{C}}=E_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}+E_{\mathrm{C}}^{\mathrm{compu}}+E_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}, \tag{24}$$

$$E_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}=P_l\times T_{\mathrm{ED}\to\mathrm{C}}^{\mathrm{trans}}, \tag{25}$$

$$E_{\mathrm{C}}^{\mathrm{compu}}=\sigma_3\times f_c^2\times w_j, \tag{26}$$

$$E_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}=P_k\times T_{\mathrm{C}\to\mathrm{ED}}^{\mathrm{trans}}. \tag{27}$$

3.5 联合时间与能耗的卸载决策目标函数

综上，可以得到所有任务的云—边—端协同卸载执行计算需要的时间T_{all}描述和能耗E_{all}描述分别为式(28)和式(29)，其中 $\beta_{\mathrm{ED}},\beta_{\mathrm{EN}},\beta_{\mathrm{C}}\in\{0,1\}$. 规定任务在 1 个位置执行，$\beta_{\mathrm{ED}},\beta_{\mathrm{EN}},\beta_{\mathrm{C}}$这 3 个参数只取 0 或 1，根据任务的执行位置决定云、边、端时间与能耗的参数取值.

$$T_{\mathrm{all}}=\sum_{j=1}^{n}\beta_{\mathrm{ED}}\times T_j^{\mathrm{ED}}+\beta_{\mathrm{EN}}\times T_j^{\mathrm{EN}}+\beta_{\mathrm{C}}\times T_j^{\mathrm{C}}, \tag{28}$$

$$E_{\mathrm{all}}=\sum_{j=1}^{n}\beta_{\mathrm{ED}}\times E_j^{\mathrm{ED}}+\beta_{\mathrm{EN}}\times E_j^{\mathrm{EN}}+\beta_{\mathrm{C}}\times E_j^{\mathrm{C}}. \tag{29}$$

为了实现云—边—端协同卸载模式下计算时间和能耗的联合优化，设计了科学评价任务卸载效果的目标函数Q.其中，θ_1和θ_2分别表示时延和能耗在目标函数中所占的比重，二者最优取值通过实验判定.$T_{\mathrm{all}}^{\mathrm{ED}}$和$E_{\mathrm{all}}^{\mathrm{ED}}$分别为所有任务均在终端设备计算的时间和能耗；$T_{\mathrm{all}}^{\mathrm{C}}$和$E_{\mathrm{all}}^{\mathrm{C}}$分别表示所有任务均在云中心计算的时间和能耗.

$$Q=\theta_1\times\left(1-\frac{2\times T_{\mathrm{all}}}{T_{\mathrm{all}}^{\mathrm{ED}}+T_{\mathrm{all}}^{\mathrm{C}}}\right)+\theta_2\times\left(1-\frac{2\times E_{\mathrm{all}}}{E_{\mathrm{all}}^{\mathrm{ED}}+E_{\mathrm{all}}^{\mathrm{C}}}\right). \tag{30}$$

由式(30)可知，系统计算时间和能耗的优化问题转化为有约束条件下目标函数Q的最优化问题，即通过搜索卸载决策方案获得Q最大值.优化目标及其约束条件如式(31)所示：

$$\max Q,\quad \text{s.t.}\begin{cases}\theta_1,\theta_2\in[0,1],\\ \theta_1+\theta_2=1,\\ \beta_{\mathrm{ED}}+\beta_{\mathrm{EN}}+\beta_{\mathrm{C}}=1,\\ mem_j<mem_i,\\ f_l>f_{\min.j},\\ f_i>f_{\min.j},\\ T_{\mathrm{all}}<t_{\max.j},\\ j\in\{0,1,2,\cdots,n\},\\ i\in\{0,1,2,\cdots,m\},\\ \text{式(4)},\\ \text{式(8)},\\ \text{式(20)}.\end{cases}\tag{31}$$

4 基于 APS 算法求解卸载决策

4.1 APS 算法说明

本文设计的人工粒子蜂群算法包含 5 个要素：蜜源、引领蜂、跟随蜂、粒子蜂和侦察蜂.APS 采用原人工蜂群算法进行初始化，随机生成初始解x_i $(i=1,2,\cdots,SN)$.在搜索阶段，引领蜂依据式(32)寻找新蜜源.

$$x'_{ij}=x_{ij}+\varphi(x_{ij}-x_{kj}),\tag{32}$$

其中$j\in\{1,2,\cdots,D\}$，$k\in\{1,2,\cdots,SN\}$且$i\neq k$，φ为 [−1,1] 之间的随机数.x'_{ij}是新蜜源位置，x_{ij}是当前蜜源位置，x_{kj}是当前蜜源邻域中的一个随机蜜源位置，比较生成的新解$x'_i=\{x'_{i1},x'_{i2},\cdots,x'_{iD}\}$和初始化的解$x_i=\{x_{i1},x_{i2},\cdots,x_{iD}\}$的适应度值，采用贪婪选择策略保留较好的解.跟随蜂按式(33)所描述的概率公式选择蜜源，其中fit_i表示x_i的适应度值，$fit_i=Q$，P'_i表示选择概率.

$$P'_i=\frac{fit_i}{\sum_{j=1}^{SN}fit_j}.\tag{33}$$

APS 融合 ABC 算法和粒子群算法，解决了 ABC 搜索效率低和提前陷入局部最优的问题.主要体现在引领蜂变异为侦查蜂阶段，APS 改变了 ABC 算法的位置更新策略，根据全局最优解的位置和个体最优解的位置来搜索蜜源，跳出局部最优.

APS 算法中引入了式(34)“全优率”参数和粒子蜂对 ABC 与 PSO 进行融合，全优率的含义为：每一个个体在迭代过程中的适应度值相对于全局最优解的概率.假设连续几代蜜源适应度不变时所有的引领蜂都为粒子蜂，通过衡量全优率，将引领蜂分成 2 类：一类是全优率大于 1 的蜜蜂变异为侦查蜂，在当前蜜源附近搜索；另一类全优率小于 1 的蜜蜂变异为粒子蜂，迭代按照 PSO 算法的式(34)进行迭代.

$$P_i^{\text{full-m}}=\sum_{k=0}^{\lim it}\frac{\left|x_{\mathrm{g}}-x_i^{k+1}\right|}{\left|x_{\mathrm{g}}-x_i^{k}\right|},\tag{34}$$

其中x_i^k表示第i个粒子蜂迭代到k代时的位置，x_{g}表示所有粒子蜂中全局最优解的位置.当引领蜂转变为侦查蜂时按照式(35)更新位置，当引领蜂转变为粒子蜂时按照式(36)和式(37)更新位置.$rand(0,1)$表示 [0,1] 间的随机数，$x_j^{\max}$和$x_j^{\min}$是参数变量的最大值与最小值.其中，v_i^k为第 i 个粒子蜂迭代到第 k 代时的速度，w 为惯性权值，c_1和c_2都为正常数，通常取$c_1=c_2=2$，x_{p}为粒子蜂 i 找到的最优解.

$$x''_{ij}=x_j^{\min}+rand(0,1)\left(x_j^{\max}-x_j^{\min}\right),\tag{35}$$

$$x_i^{k+1}=x_i^k+v_i^{k+1},\tag{36}$$

$$v_i^{k+1}=w\times v_i^k+c_1\times rand(0,1)\left(x_{\mathrm{p}}-x_i^k\right)+c_2\times rand(0,1)\left(x_{\mathrm{g}}-x_i^k\right).\tag{37}$$

4.2 算法描述及流程

算法 2. 为基于 APS 算法的卸载决策.

步骤 1. 初始化参数，包括种群规模、引领蜂和跟随蜂数量、最大迭代次数、限定蜜源更新次数、搜索空间等；

步骤 2. 随机生成初始蜜源，要求初始蜜源表示的卸载决策方案均满足各任务对内存的需求；

步骤 3. 进行邻域搜索，通过式(33)分别计算初始蜜源和领域更新后蜜源的适应度，采用贪婪选择方法保留较好的蜜源；

步骤 4. 根据引领蜂分享的信息，判断是否为跟随蜂分配蜜源，如果分配则记录下最好的蜜源位置；

步骤 5. 如果不分配则为跟随蜂选择 1 个蜜源，然后搜索新的蜜源计算适应度函数进行贪婪选择，并保留较好的蜜源；

步骤 6. 判断是否达到最大蜜源更新次数，“是”则记录最好的蜜源位置，“否”则回到步骤 5；

步骤 7. 为避免陷入局部最优，进行当前最好蜜源的全优率判断，全优率大于 1 则产生侦查蜂，使用式(4)更新蜜源位置；全优率小于 1 则产生粒子蜂，使用式(5)和式(6)更新蜜源位置；

步骤 8. 判断是否达到最大迭代次数，“是”则进行步骤 9，“否”则回到步骤 3；

步骤 9. 输出最优卸载决策 ST.

算法 2 流程图如图 4 所示.

5 仿真实验与结果分析

5.1 仿真实验介绍

在本节中，部署了一个云—边—端协同的网络结

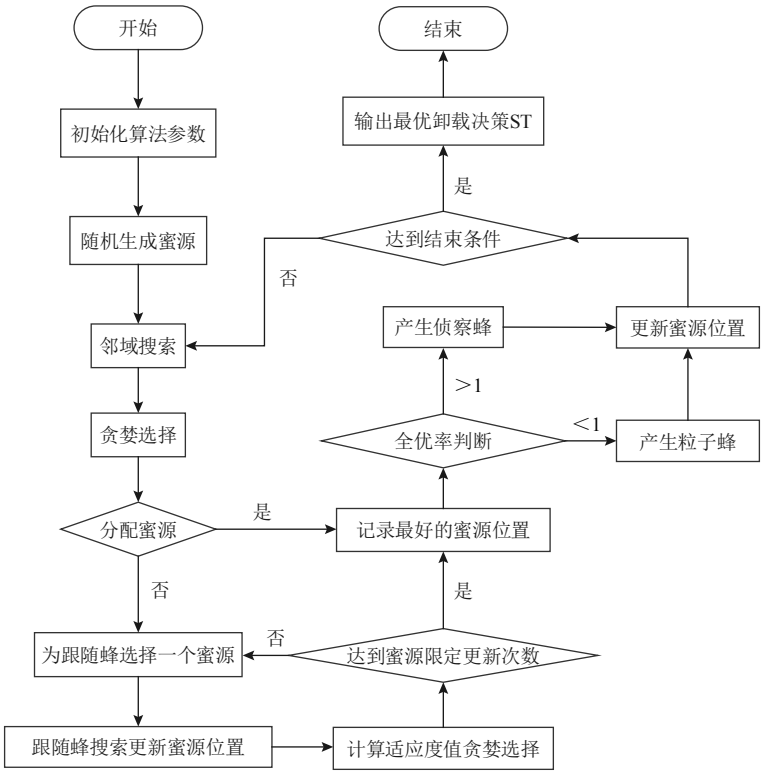

Fig. 4 Flow chart of APS algorithm

图 4 APS 算法流程图

构仿真，提出的任务卸载架构包括 1 个服务半径为 1 km 的云中心服务器、4 个服务半径为 500 m 的边缘服务器和 50 个移动终端设备，总覆盖面积为 2 km×2 km.移动终端设备和边缘服务器随机分布在覆盖区域中.本文采用 Python 语言进行编程仿真，仿真实验中使用的相关参数参考文献 [41] 设定，如表 2 所示，实验内容如表 3 所示.

为验证以上实验内容对系统性能的影响，将本文方案与 5 种任务处理方式作比较:

1）Local: 任务只在终端设备进行处理计算.

2）Edge: 任务随机卸载到边缘节点上计算.

3）Cloud: 任务全部卸载到云中心执行.

4）Local-Edge: 任务在本地和边缘节点执行.

5）Random: 任务在端—边—云随机执行.

5.2 实验结果与分析

5.2.1 θ_1，θ_2的取值实验

图 5 和图 6 分别表征了系统在处理 100 个随机任务的情况下，θ_1取不同的值时系统的计算时间和能耗情况.从图 5 和图 6 可知，当θ_1=0.6 时，系统计算时间最短；θ_1=0.5 时，系统能耗最低.同时从 2 个结果图的变化趋势可以看出，θ_1的取值对于系统计算时间的影响较大，能耗的变化相比于计算时间来讲比较平缓，尤其是当θ_1=0.4, 0.5, 0.6 时，能耗的变化微小.因此

Table 2 Experimental Parameters Setting

表 2 实验参数设定

参数	取值
终端设备数量 L	50
边缘节点数量 M	4
信道带宽 W/MHz	20
信道噪声功率 τ^2/W	2×10^{-13}
链路之间干扰功率 I_q /W	2×10^{-13}
任务计算量 w_j /(cycle·s^{-1})	[10,1 000]
任务输入数据量 $data_j$ /MB	[0.5,5]
任务所需最小计算能力 $f_{\min,j}$ /GHz	[2.0,20]
任务所需内存资源 mem_j /MB	[50,500]
任务所需 CPU 资源 cpu_j /GHz	[1.0,3.0]
任务最大容忍时延 $t_{\max,j}$ /s	[0.5,1.5]
终端设备计算能力 f_l /GHz	4
终端设备发射功率 P_l /W	[0.1,0.5]
开关电容σ_1/μF	0.01
边缘节点最大计算能力 f_i /GHz	50
边缘节点可用 CPU 资源 cpu_i /GHz	[3.0,30.0]
边缘节点可用内存资源 mem_i / MB	[1,40]
边缘节点与设备间传输速度 R_i / Mbps	[1,10]
边缘节点发射功率 P_i /W	4
云中心最大计算能力 f_k /GHz	100
云中心与设备间传输速度 R_k /Mbps	[1,10]
云中心发射功率 P_k /W	10
θ_1和θ_2初始值	0.5,0.5

Table 3 Experimental Contents Setting

表 3 实验内容设定

项目	实验内容
1	采用 APS 算法评估目标函数中时间占比和能耗的占比，并得出最优的θ_1, θ_2值.
2	从计算时间和能耗角度对比 APS，ABC，PSO 以及随机卸载算法（RAN）的表现，评估 APS 卸载决策算法的性能.
3	研究终端数量和边缘节点数量对协同卸载系统性能的影响.
4	研究任务的配置文件对系统性能的影响.
5	研究任务最大延迟和最小计算能力对系统性能的影响.
6	研究终端设备最大发射功率对系统性能的影响.

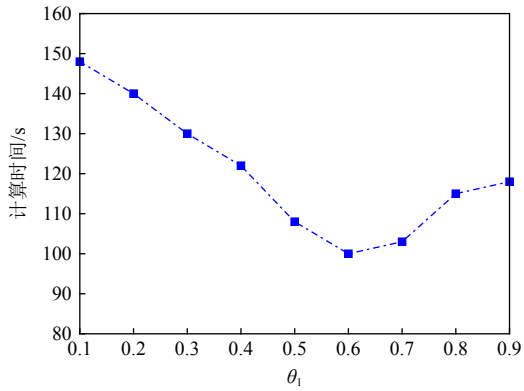

Fig. 5 The relationship of value θ_1 and the systematic calculation time

图 5 θ_1的取值与系统计算时间的关系

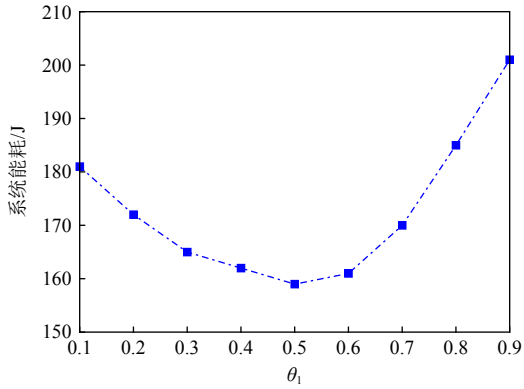

Fig. 6 The relationship of value θ_1 and the systematic energy consumption

图 6 θ_1的取值与系统能耗的关系

综合以上情况分析，取θ_1=0.6 时具有最合适的延时与能耗表现.

5.2.2 卸载决策算法验证

在处理不同的任务量并经过 100 次迭代计算后，4 种卸载决策算法的计算时间和能耗对比分别如图 7 和图 8 所示.由图 7 可知，当任务数量较少时，APS 算法虽然具备一定的优越性，但同比 ABC 和 PSO 差距较小；随着任务数量的增多，APS 相比于 ABC 算法、PSO 算法和 RAN 算法，计算时间分别降低了 19.6%，29.6%，51.9%.图 8 展示了系统能耗的变化，对比不同数量的任务，APS 算法的能耗都在不同程度上降低，其中，相比于 ABC 算法能耗最大能降低 12.8%，相比于 PSO 算法能耗最大能降低 40.3%，相比于 RAN 算法能耗最大能降低 50.2%.由此可见，APS 算法具有不可比拟的优越性，极大提升了整个系统性能.

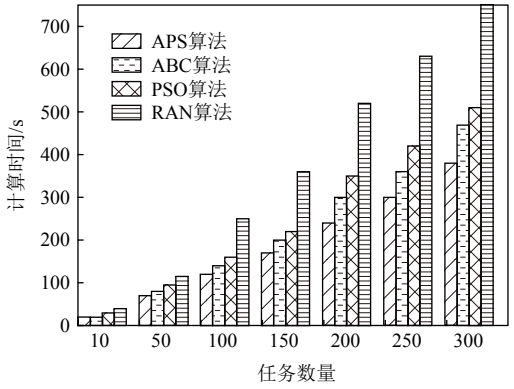

Fig. 7 Calculation time of four algorithms for different numbers of tasks

图 7 不同任务数量下 4 种算法的计算时间

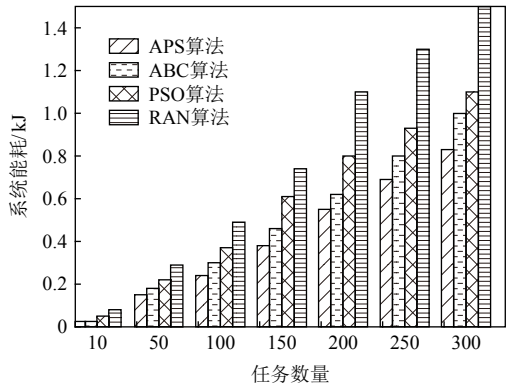

Fig. 8 Systematic energy consumption of four algorithms for different numbers of tasks

图 8 不同任务数量下 4 种算法的系统能耗

5.2.3 用户数量对系统的影响

图 9 和图 10 分别评估了终端设备的数量对系统总时延和系统能耗的影响.从图 9 中可以看出，所有方案的总时延都随着终端设备数量的增加而上升，但 Joint 的总时延始终是最小的.当终端设备数量增多时，所产生的任务需要的计算资源也迅速增多，结合任务的数据量等特性，系统依据卸载模型将一些任务从终端设备卸载到边缘节点或云中心处理，以获得最小的服务时延.此外，发现当用终端数量逐渐增多时，边缘节点计算方案的总时延大于本地计算

方案的总时延，这是由于任务在卸载过程中信道和边缘节点计算资源的竞争没有遵循合理的卸载顺序，导致任务在传输过程和计算过程都存在不必要的等待环节，因此时延越来越高.

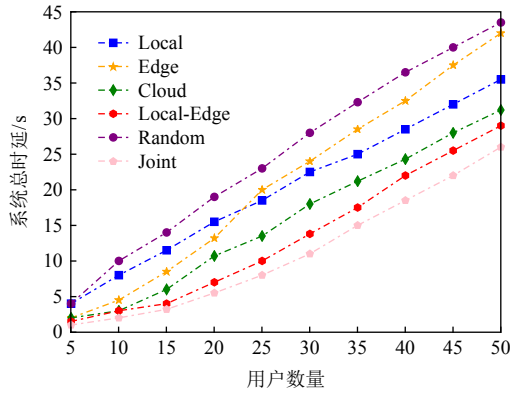

Fig. 9 Effect of the number of end users on the total systematic delay

图 9 终端用户数量对系统总时延的影响

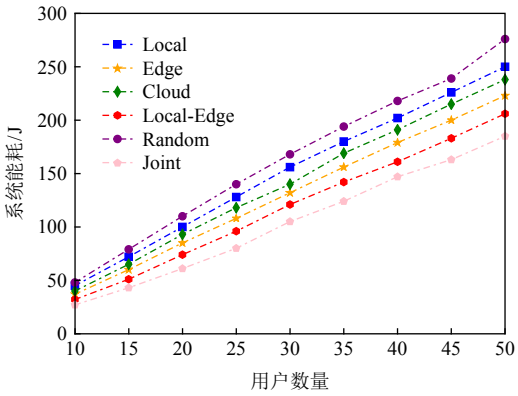

Fig. 10 Effect of the number of end users on the systematic energy consumption

图 10 终端用户数量对系统能耗的影响

从图 10 中可以看出，系统能耗也随着终端数量的增多而增多，且 Joint 的能耗表现具有很强的优越性.Edge 的能耗低于 Cloud 的能耗的原因在于：不同于系统时延，任务卸载到边缘节点的方案存在滞后等待，当任务到达 Edge 后，等待过程的能耗可忽略不计.相比于远距离的 Cloud，Edge 的传输能耗显著降低，因此，总体上 Edge 的能耗低于 Cloud 的能耗.

5.2.4 任务的配置文件对系统性能的影响

这里评估了任务所需计算资源量和任务输入数据量对系统总体时延和系统能耗的影响(终端设备数量为 30).2 种配置下各方案的时延表现如图 11 和图 13 所示.6 种方案的总体时延都随任务计算资源量的增加而增加，但 Joint 的总时延始终是最小的.由于本地计算不涉及任务的卸载和传输，因此时延保持为一个稳定的数值.而 Edge 模式随着卸载任务和数据量的增加时，出现争夺通信资源和边缘节点计算资源的情况，因此其时延要高于云计算和本地计算.从 2 种任务配置因素得知，本文系统可快速处理一些要求计算资源量高和输入数据量高的任务.

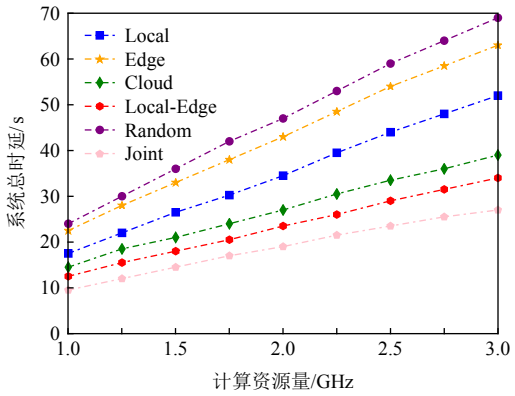

Fig. 11 Effect of computing resources required by tasks on systematic delay

图 11 任务所需计算资源对系统时延的影响

如图 12 所示，当任务所需计算资源量增加时，6 种方案的能耗随之上升，但 Joint 的能耗远小于其他系统的能耗.通过计算平均任务所需资源量，Joint 能耗分别比 Random 降低了 58.28%，比 Local 降低 55.60%，比 Cloud 降低 43.54%，比 Edge 降低 24.46%，比 Local-Edge 降低 16%.由图 14 得，随着任务输入数据量的增加，本地计算不涉及数据输入，能耗逐渐稳定增大，Joint 尤其表现优越.因此，当面临大数据量输入时，Joint 的能耗表现远远优于其他模式，其中相比 Random 最高降低了 52.72%，比 Local 最高降低 29.20%，比 Cloud 最高降低 26.92%，比 Edge 最高降低 22.45%，

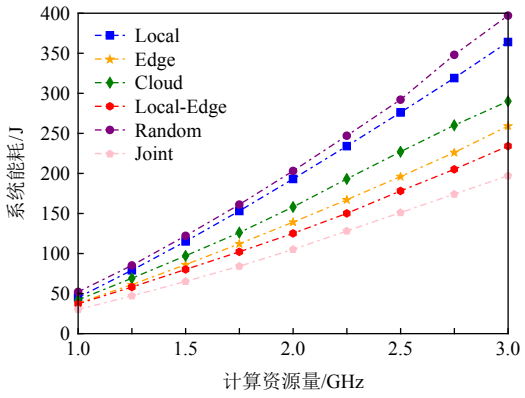

Fig. 12 Effect of computing resources required by tasks on systematic energy consumption

图 12 任务所需计算资源对系统能耗的影响

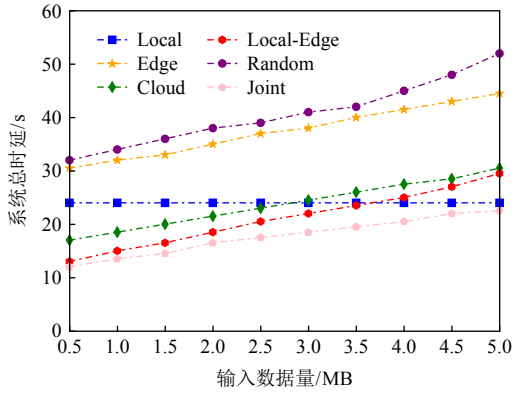

Fig. 13 Effect of the amount of date input by tasks on systematic delay

图 13 任务输入数据量对系统时延的影响

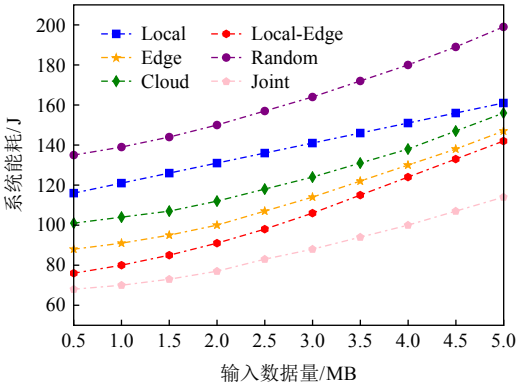

Fig. 14 Effect of the amount of date input by tasks on the systematic energy consumption

图 14 任务输入数据量对系统能耗的影响

比 Local-Edge 最高降低 19.82%.

5.2.5 任务的固定属性对系统性能的影响

这里评估了任务最大时延和任务所需最小计算能力对系统时延和能耗的影响.如图 15 所示，当任务的时延要求逐渐增大时，将有更多任务在本地执行，同时所需计算资源量大的任务卸载到边缘节点，因此系统总时延随着时延容忍的增加而减少. Joint 相比其他方案的总时延总是保持最小，图 16 中能耗也呈现出和时延状态相似的表现，Joint 方案对应总体时延和能耗始终维持低数值，具有一定优越性.

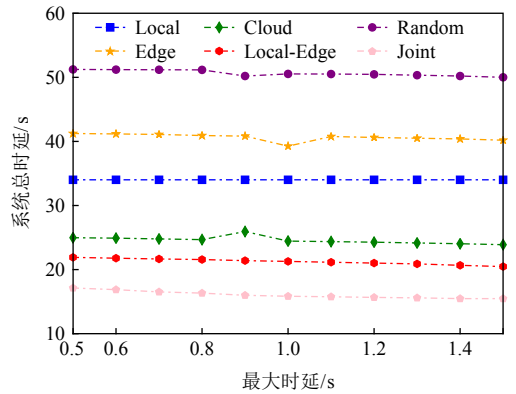

Fig. 15 Effect of the maximum delay of tasks on the systematic delay

图 15 任务最大延迟对系统延时的影响

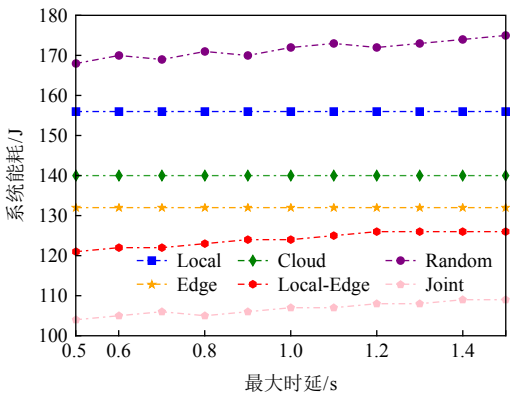

Fig. 16 Effect of the maximum delay of tasks on the systematic energy consumption

图 16 任务最大延迟对系统能耗的影响

图 17 和图 18 反映了任务所需的最小计算能力对系统的影响.由图 17 得知，当最小所需计算能力不断增加时，任务的执行位置越来越受限，从而导致本地设备不能参与计算，所有任务为了卸载至边缘服务器或者云中心，全部都去竞争有限的信道资源与网络资源，因此系统总体时延性能变差.从图 18 可知，虽然 Joint 的能耗在高计算能力需求时在一定程度上与边缘计算模式和云计算相近，但充分满足了系统时延的要求.综上所述，与其他 5 种方案相比，Joint 方案在大规模计算情况下具备很强的自适应性.

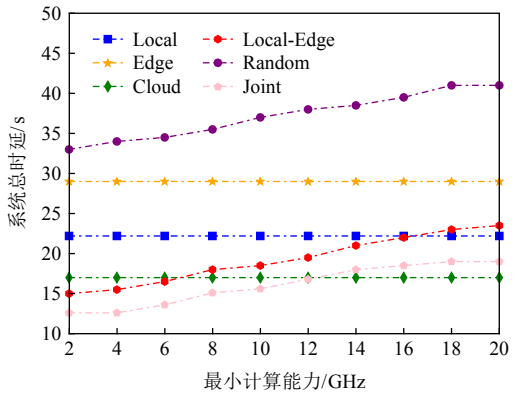

Fig. 17 Effect of the minimum computing power required by tasks on the systematic delay

图 17 任务所需最小计算能力对系统延时的影响

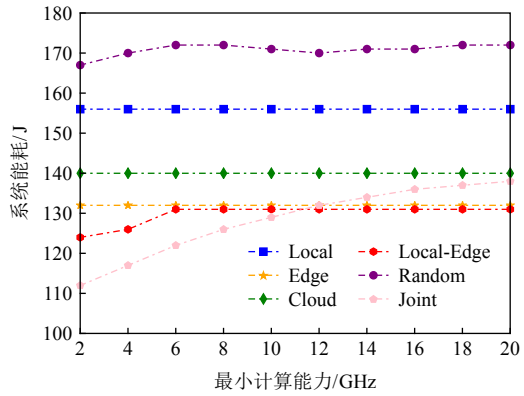

Fig. 18 Effect of the minimum computing power required by tasks on the systematic energy consumption

图 18 任务所需最小计算能力对系统能耗的影响

5.2.6 终端最大发射功率对系统性能的影响

图 19 和图 20 分别显示了终端设备最大发射功率对系统时延与能耗的影响.由图 19 可知随着终端设备最大发射功率的不断增加，发射功率的范围变大，终端设备的可用功率相应增加，因此任务卸载速度变快，系统总时延变小.由图 20 得知，终端最大发射功率增加幅度小，但系统时延下降较快，因此系统能耗总体上呈下降趋势.Joint 方案相比于其他方案不论是时延还是能耗都显现出一定的优势，终端设备最大发射功率在合理范围内调整时，Joint 系统性能始终最优.

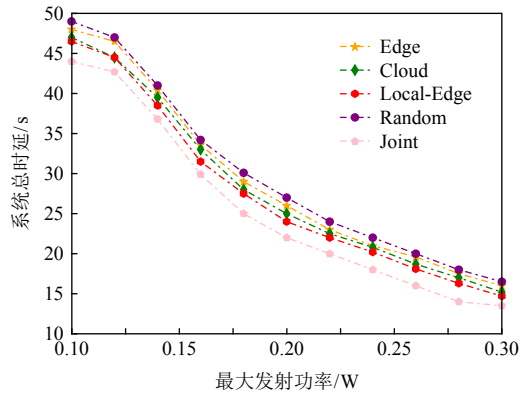

Fig. 19 Effect of the maximum transmitting power of end on the systematic delay

图 19 终端设备最大发射功率对系统延迟的影响

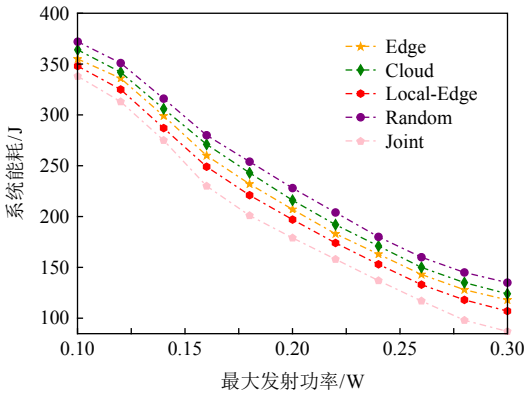

Fig. 20 Effect of the maximum transmitting power of end on the systematic energy consumption

图 20 终端设备最大发射功率对系统能耗的影响

6 总 结

在本文中，我们部署了一个云—边—端协同计算架构，此架构可实现移动终端设备的任务择优在本地、边缘节点或云中心处理.本文在以往研究成果的基础上考虑了任务之间存在的依赖关系、任务并行计算以及任务结果回程问题，基于此分别建立了端、边、云任务计算模型，设计联合优化时延与能耗的目标函数；其次，在 ABC 算法和 PSO 算法基础上引入全优率指数与粒子蜂，设计了 APS 算法求解联合优化时延与能耗的任务卸载决策；最后，在多接入情况下，构造最小化任务计算时间和能耗的目标问题，在评估各项资源的情况下实现自适应地卸载任务.仿真结果表明，本文方案在多终端、多任务情况下可提供低时延、低能耗的服务.未来本研究计划通过联合优化时延和能耗进一步优化任务执行矩阵，并探索 5G 网络环境下，结合人工智能技术动态预测和评估计算资源的利用和服务质量.

作者贡献声明：张文柱指导论文写作，并参与实验设计和实验数据分析；余静华完成数据分析和论文的写作.

参 考 文 献

[1] Casadei R, Fortino G, Pianini D, et al. Modelling and simulation of opportunistic IoT services with aggregate computing[J]. Future Generation Computer Systems, 2019, 91(2): 252−262

[2] Abolfazli S, Sanaei Z, Ahmed E, et al. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges[J]. IEEE Communications Surveys Tutorials, 2014, 16(1): 337−368

[3] Cui Yong, Song Jian, Liao Congcong, et al. Advances and trends in mobile cloud computing[J]. Chinese Journal of Computers, 2017, 40(2): 273−295 (in Chinese)
(崔勇, 宋健, 缪葱葱, 等. 移动云计算研究进展与趋势[J]. 计算机学报, 2017, 40(2): 273−295)

[4] Haung Y R. A QoE-aware strategy for supporting service continuity in an MCC environment[J]. Wireless Personal Communications, 2021, 116(1): 629−654

[5] Zhou Yuezhi, Zhang Di. Near-end cloud computing: Opportunities and challenges in post-cloud computing era[J]. Chinese Journal of Computers, 2019, 42(4): 677−700 (in Chinese)
(周悦芝, 张迪. 近端云计算: 后云计算时代的机遇与挑战[J]. 计算机学报, 2019, 42(4): 677−700)

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

[7] Zhang Kaiyuan, Gui Xiaolin, Ren Dewang, et al. A review of computing migration and content caching in mobile edge networks[J]. Journal of Software, 2019, 30(8): 2491−2516 (in Chinese)
(张开元, 桂小林, 任德旺, 等. 移动边缘网络中计算迁移与内容缓存研究综述[J]. 软件学报, 2019, 30(8): 2491−2516)

[8] Li Baogang, Si Fangqiang, Zhao Wei, et al. Wireless powered mobile edge computing with NOMA and user cooperation[J]. IEEE Transactions on Vehicular Technology, 2021, 70(2): 1957−1961

[9] Xie Renchao, Lian Xiaofei, Jia Qingmin, et al. A review of moving edge computing offloading techniques[J]. Journal on Communications, 2018, 39(11): 138−155 (in Chinese)
(谢人超, 廉晓飞, 贾庆民, 等. 移动边缘计算卸载技术综述[J]. 通信学报, 2018, 39(11): 138−155)

[10] Zhang Yue, Fu Jingqi. Energy efficient computation offloading strategy with tasks scheduling in edge computing[J]. Wireless Networks, 2021, 27(1): 609−620

[11] Liu Wei, Huang Yucheng, Du Wei, et al. A resource-constrained serial task offloading strategy in mobile edge computing[J]. Journal of Software, 2020, 31(6): 309−328 (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): 309−328)

[12] Zhang Liqing, Guo Dong, Wu Shaoling, et al. An ultra-lightweight container with maximum memory sharing and minimum runtime environment[J]. Journal of Computer Research and Development, 2019, 56(7): 1545−1555 (in Chinese)
(张礼庆, 郭栋, 吴绍岭, 等. 一种最大化内存共享与最小化运行时环境的超轻量级容器[J]. 计算机研究与发展, 2019, 56(7): 1545−1555)

[13] Tang Jie, Yu Rao, Liu Shaoshan, et al. A container based edge offloading framework for autonomous driving[J]. IEEE Access, 2020, 8(1): 33713−33726

[14] Li Shuangyuan. A task offloading optimization strategy in MEC based smart cities[J]. Internet Technology Letters, 2021, 4(1): e158

[15] Huang Qianyi, Li Zhiyang, Xie Wentao, et al. Edge computing in the smart home[J]. Journal of Computer Research and Development, 2020, 57(9): 1800−1809 (in Chinese)
(黄倩怡, 李志洋, 谢文涛, 等. 智能家居中的边缘计算[J]. 计算机研究与发展, 2020, 57(9): 1800−1809)

[16] Xu Yu, Zhang Tiankui, Yang Dingcheng, et al. Joint resource and trajectory optimization for security in UAV assisted MEC systems[J]. IEEE Transactions on Communications, 2021, 69(1): 573−588

[17] Xue Ning, Huo Ru, Zeng Shiqin, et al. MEC task unloading and resource scheduling algorithm based on DRL[J]. Journal of Beijing University of Posts and Telecommunications, 2019, 42(6): 64−69 (in Chinese)
(薛宁, 霍如, 曾诗钦, 等. 基于DRL的MEC任务卸载与资源调度算法[J]. 北京邮电大学学报, 2019, 42(6): 64−69)

[18] Bolettieri S, Bruno R, Mingozzi E. Application-aware resource allocation and data management for MEC assisted IoT service providers[J]. Journal of Network and Computer Applications, 2021, 181(2): 103020

[19] Xue Jianbin, Ding Xuegan, Liu Xingxing. Offload decision and resource optimization for caches aided edge computing[J]. Journal of Beijing University of Posts and Telecommunications, 2020, 43(3): 32−37 (in Chinese)
(薛建彬, 丁雪乾, 刘星星. 缓存辅助边缘计算的卸载决策与资源优化[J]. 北京邮电大学学报, 2020, 43(3): 32−37)

[20] Liu Chubo, Tang Fang, Li Kenli, et al. Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(7): 1603−1614

[21] Meng Yao, Dai Janxin. Energy efficient joint computation offloading and resource allocation in multi-user MEC systems[J]. Journal of Physics: Conference Series, 2020, 1693: 012042

[22] Miao Yiming, Wu Gaoxiang, Li Miao, et al. Intelligent task prediction and computation offloading based on mobile-edge cloud computing[J]. Future Generation Computer Systems, 2020, 102(9): 925−931

[23] Li Yang, Xu Gaochao, Ge Jiaqi, et al. Energy efficient resource allocation for application including dependent tasks in mobile edge computing[J]. KSII Transactions on Internet and Information Systems, 2020, 14(6): 2422−2443

[24] Chai Ming, Li Mingzhu, Yang Tiantian, et al. Dynamic priority based computation scheduling and offloading for interdependent tasks: Leveraging parallel transmission and execution[J]. IEEE Transactions on Vehicular Technology, 2020, 70(10): 10970−10985

[25] Liu Bowen, Xu Xiaolong, Qi Lianyong, et al. Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment[J]. Journal of Systems Architecture, 2021, 114(6): 101970

[26] Yan Jia, Bi Suzhi, Zhang Yingjun, et al. Optimal task offloading and resource allocation in mobile edge computing with inter-user task dependency[J]. IEEE Transactions on Wireless Communications, 2020, 19(1): 235−250

[27] Chen Long, Wu Jigang, Zhang Jun, et al. Dependency-aware computation offloading for mobile edge computing with edge-cloud cooperation[J]. IEEE Transactions on Cloud Computing, 2020, 11(10): 973−991

[28] Abbasi M, Mohammadi E, Khosravi M. Intelligent workload allocation in IoT-Fog-Cloud architecture towards mobile edge computing[J]. Computer Communications, 2021, 169(3): 71−80

[29] Li Wenzao, Wang Fangxin, Pan Yuwen, et al. Computing cost optimization for multi-BS in MEC by offloading[J]. Mobile Networks and Applications, 2020, 25(4): 1628−1641

[30] Zhu Zhengying, Qian Liping, Shen Jiafang, et al. Joint optimisation of UAV grouping and energy consumption in MEC enabled UAV communication networks[J]. IET Communications, 2020, 14(16): 2723−2730

[31] Xie Renchao, Li Zishu, Wu Jun, et al. Energy-efficient joint caching and transcoding for HTTP adaptive streaming in 5G networks with mobile edge computing[J]. China Communications, 2019, 16(7): 229−244

[32] Feng Siling, Chen Yinjie, Zhai Qianhao, et al. Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms[J]. EURASIP Journal on Advances in Signal Processing, 2021, 2021(1): 1−15

[33] Liu Zhizhong, Sheng Quan, Xu Xufei, et al. Context-aware and adaptive QoS prediction for mobile edge computing services[J]. IEEE Transactions on Services Computing, 2019, 30(9): 125−139

[34] Liang Liang, Xiao Jintao, Ren Zhi, et al. Particle swarm based service migration scheme in the edge computing environment[J]. IEEE Access, 2020, 8: 45596−45606

[35] Ma Shuyue, Song Shudian, Zhao Jingmei, et al. Joint network selection and service placement based on particle swarm optimization for multi-access edge computing[J]. IEEE Access, 2020, 8: 160871−160881

[36] You Qian, Tang Bing. Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things[J]. Journal of Cloud Computing, 2021, 10(1): 1−11

[37] Tamilselvan V. A hybrid PSO-ABC algorithm for optimal load shedding and improving voltage stability[J]. International Journal of Manufacturing Technology and Management, 2020, 34(6): 577−597

[38] Gerardo S, Andrade M, Lara-Velázquez P, et al. ABC-PSO: An efficient bioinspired metaheuristic for parameter estimation in nonlinear regression[C] //Proc of the 16th Mexican Int Conf on Artificial Intelligence. Cham: Springer, 2016: 388−400

[39] Singh S, Chauhan P, Singh N. Capacity optimization of grid connected solarfuel cell energy system using hybrid ABC-PSO algorithm[J]. International Journal of Hydrogen Energy, 2020, 45(16): 10070−10088

[40] Han Zidong, Li Yufeng, Liang Junyu. Numerical improvement for the mechanical performance of bikes based on an intelligent PSO-ABC algorithm and WSN technology[J]. IEEE Access, 2018, 6: 32890−32898

[41] Zhou Ping, Shen Ke, Kumar N, et al. Communication efficient offloading for mobile edge computing in 5G heterogeneous networks[J]. IEEE Internet of Things Journal, 2020, 8(13): 10237−10247

Zhang Wenzhu, born in 1970. PhD, professor. His main research interests include wireless communication theory and technology, mobile edge computing.

张文柱, 1970 年生.博士, 教授.主要研究方向为无线通信理论与技术、移动边缘计算.

Yu Jinghua, born in 1997. Master. Her main research interests include mobile edge computing and edge platform technology.

余静华, 1997 年生.硕士.主要研究方向为移动边缘计算和边缘平台技术.