计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800088

一种基于遗传算法的多边缘协同计算卸载方案

高基旭　王　珺
南京邮电大学通信与信息工程学院　南京 210000
(gjxaiou@gmail.com)

摘　要　边缘计算(Edge Computing,EC)作为云计算的补充,在处理 IOT 设备产生的计算任务时可以保证计算的延时符合系统的要求。针对在传统卸载场景中,由于计算任务到达存在空窗期导致异地边缘云存在空闲状态,造成异地边缘云利用不充分的问题,文中提出了一种基于遗传算法的多边缘与云端协同计算卸载模型(Genetic Algorithm-based Multi-edge Collaborative Computing Offloading Model,GAMCCOM)。该计算卸载方案联合本地边缘和异地边缘进行任务卸载,并采用遗传算法进行求解,从而得到同时考虑时延和能耗的最小的系统代价。通过仿真实验结果可知,在综合考虑卸载系统的时延消耗和能量消耗的情况下,该方案相比基本的三层卸载方案系统整体代价降低了 23%,在只考虑时延消耗和只考虑能量消耗的情况下依然分别能够降低系统代价 17% 和 15%。因此针对边缘计算的不同卸载目标,GAMCCOM 卸载方案对系统代价均有比较优秀的降低效果。

关键词:边缘计算;物联网;计算卸载;协同卸载;遗传算法
中图法分类号　TP393

Multi-edge Collaborative Computing Unloading Scheme Based on Genetic Algorithm

GAO Ji-xu and WANG Jun
School of Communication and Information Engineering,Nanjing University of Posts and Telecommunications,Nanjing 210000,China

Abstract　As a supplement to cloud computing,edge computing can ensure that the calculation delay meets system requirements when processing computing tasks generated by IOT equipment. Aiming at the problem of insufficient utilization of the remote edge cloud due to the empty window period of the computing task in the traditional offloading scenario,a genetic algorithm-based multi-edge and cloud collaborative computing offloading model (Genetic Algorithm-based Multi-edge Collaborative Computing Offloading Model,GAMCCOM) is proposed. This computing offloading solution combines local edge and remote edge to perform task offloading and uses a genetic algorithm to get the minimum system cost under consideration of both delay and energy consumption at the same time. The results of simulation experiments show that when considering the time delay consumption and energy consumption of the unloading system,the overall cost of this scheme is reduced by 23% compared with the basic three-layer unloading scheme. In the case of considering time delay consumption and energy consumption respectively,the system cost can still be reduced by 17% and 15% respectively. Therefore,the GAMCCOM offloading method can effectively reduce the system cost for different offloading targets of edge computing.

Keywords　Edge computing,Internet of things,Computing unloading,Collaborative unloading,Genetic algorithm

1　引言

随着物联网、大数据以及人工智能的发展,人们需要对不断产生的海量数据进行及时的处理反馈。根据思科研究报告,到 2020 年全球将有超过 50 亿台设备终端实现在线互联[1],因此移动边缘计算作为一种新兴分布式架构被提出[2]。边缘计算作为云计算的一种补充,其设备主要部署于接近产生计算数据源的用户端网络边缘,用于将原有云计算中由大型的中心云节点处理的大型服务划分为相对较小的计算模块,然后分别交给不同的边缘节点进行计算处理。因为相比于云数据中心而言,边缘设备的部署位置通常更加接近数据与计算任务的产生地,所以直接在边缘端处理计算服务可以减少任务传输计算带来的时延与能耗,同时对于计算量需求过大的资源,仍然可以将其传输到计算能力更强的云服务中心进行处理。因此,边缘计算是一种弥补云计算的高带宽需求、高时延、低可靠性等缺点的有效手段[3]。

边缘计算的主要研究重点和难点在于计算任务的卸载[4]。同样,其挑战在于需要在距离数据产生地较近但是计

到稿日期:2020-08-16　返修日期:2020-10-18　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。
通信作者:王珺(wang_jun@njupt.edu.cn)

算处理能力较弱的边缘端和距离较远但是处理能力较强的网络中心端选择一个确定的计算卸载节点来处理任务，即在系统算法上通常需要选择合适的卸载算法来实现最小化系统代价。整个系统的卸载方案决策需要考虑设备计算能力、任务传输延迟、网络环境、计算任务关系等。

在现有的边缘计算任务卸载架构中，较多边缘设备节点存在计算任务空窗期，即在靠近本地终端对应的本地边缘云周围通常存在大量的异地边缘云，它们与本地边缘云的距离较近并拥有一定的计算能力，但是在自身计算需求不高的情况下它们的空闲计算能力却没有被充分利用[5]。多边缘协同卸载计算模型将异地边缘设备节点的计算能力纳入整个边缘端卸载体系，从而使得边缘端的卸载体系的计算能力得到提升，可以处理更多的计算任务，尽可能地减少传输到中心云的任务，降低系统时间代价。

针对上述问题，本文引入本地边缘云和异地边缘云之间的协同卸载，即卸载到本地边缘上的计算任务可以选择卸载到异地边缘云或者中心云上进行计算。然后将多边缘协同卸载问题建模为最小化时间与能耗的加权模型，从而获得系统最小总执行代价。遗传算法相比其他的启发式算法具有与问题领域无关且快速随机的搜索能力，同时搜索的过程从群体出发，具有潜在的并行能力，因此可以提升搜索过程的速度。因而本文最后利用遗传算法寻找最优解，进而获取最优的卸载决策方案。

本文第 2 节详细介绍了计算卸载研究的相关工作；第 3 节建立了边缘云之间协同计算卸载问题的数学模型；第 4 节详细描述了遗传算法，并结合本卸载方案寻找最优卸载决策；第 5 节通过仿真实验的结果讨论和对比本方案的优势；最后总结全文并展望未来。

2 相关工作

计算卸载的主要任务是将产生自终端的计算任务通过计算卸载算法选择不同的卸载目的地，从而缓解终端设备在能量消耗以及计算能力方面的不足，以达到获得系统整体最小代价的目标。现有研究根据系统需求不同通常主要将计算卸载决策目标划分为 3 类：以降低时延为目标、以降低能耗为目标和以权衡时延和能耗最小化为目标。

现有研究通常为仅局限于本地终端、边缘云、云中心的 3 级卸载方案。文献[6]设计了一个可以满足本地执行、部分卸载、完全卸载的计算卸载模型，该模型在每一个时隙内决定是否将缓冲任务卸载到边缘服务器，最后综合所有时隙的卸载决策形成最终的卸载决策链。其使用马尔可夫决策过程对每个任务的平均时延和设备的平均功耗进行分析，并使用一维搜索算法找到最优随机计算卸载策略。文献[7]通过研究卸载内容之间的依赖关系对最终计算卸载决策的影响，来确定本地和远程设备上寻找任务的最佳匹配，提出了一种不同于整数规划和启发式算法的全多项式时间近似方案(Fully Polynomial Time Approximation Scheme，FPTAS)，同时使用在线算法来学习未知的动态环境，该方案相比于采用启发式算法的方法在时延和 CPU 的计算时间上都有所优化。在通常情况下，因为计算任务存在不连续的情况，所以处理当前任务的本地边缘会在较长时间处于空闲状态，而该本地边缘相对于其他任务又属于异地边缘，因此可以融合其他任务的本地边缘进行联合计算卸载处理。

在利用周围空闲设备进行卸载资源计算方面，文献[8-10]主要侧重于研究多用户情况下的用户之间的相互卸载，一方面在边缘服务器资源有限的情况下，通过 P2P 进行协同计算从而减轻服务器的负担，另一方面在用户之间共享计算资源，可以平衡计算工作量和计算能力在用户之间的不均匀分布。其中文献[8]提出了一种四时隙联合计算和通信协作协议，其通过 MEC 服务器不仅可以计算用户卸载下来的部分任务，还可以充当中继节点将任务转发给其他 MEC 服务器。文献[10]提出了一种基于 Lyapunov 优化和博弈论方法的在线节点卸载框架，其通过小规模基站之间的协作来处理网络空间中不均匀的计算卸载问题。

在利用异地边缘节点的空余计算能力方面，文献[11]考虑到部分边缘服务器本身的计算资源有限，提出了分层的边缘计算部署架构，通过在边缘服务器的邻近区域引入一个备份计算服务器来解决边缘服务器的计算资源问题，同时采用 Stackelberg 博弈论的方法实现了多层级的卸载方案。文献[12]只针对卸载模型中边缘端和中心云的协同卸载部分，提出了一种基于双边博弈的计算卸载方法，其采用先通过强连通分量的集群聚合，然后进行多对一的匹配博弈，来实现最小化计算时延的目标。而该方案将中间本地终端设备、本地边缘、异地边缘以及中心云完整地整合到同一个计算卸载协同方案中，不仅实现了对异地边缘节点的空余计算能力的利用，同时还因为本地边缘和异地边缘在设备和处理方式的高度相似性，降低了任务在不同架构上转换的成本。文献[13]主要针对应用程序组件进行分类，然后在应用程序分区卸载算法的基础上利用空余的边缘节点进行任务计算卸载，但是此算法仅针对系统时延进行了优化，且没有给出系统模型与应用程序组件的数目和可用边缘节点数目之间的关系，使得卸载算法的可重用推广性有所降低。

综上所述，以上计算卸载方案大多要么仅优化计算卸载过程中的计算决策，要么只对三层卸载架构中的某些部分进行协作。而本文提出的方案在卸载时考虑将部分任务卸载到空闲的异地边缘节点。本文首先将卸载问题建模成一个通用的卸载代价模型；然后在综合考虑时延和能耗的情况下，将任务分别卸载到本地终端节点、本地边缘节点、异地边缘节点以及中心云；最后通过遗传算法得到最优解，从而得到时延和能耗均衡代价最小的卸载方案。

3 计算卸载系统模型

本节首先描述了多边缘协同卸载模型适用的系统场景；然后分别获取卸载系统各部分的时延和能量消耗代价；最终阐述了整合为完整的计算卸载系统模型，同时对于计算任务使用有向加权图的方式进行了构建描述。

3.1 系统场景

传统的计算卸载整体架构一般分为3层：第1层为物联网终端设备；第2层为由边缘服务器组成的边缘云，其中边缘服务器一般采用SCC(Small Cell Cloud)方案实现，即通过在基站中增加小型基站管理器SCM(Small Cell Manager)来实现资源的计算和管理；第3层为计算能力较强的中心云。三层架构中的任务传输可以采用4G、5G、WiFi以及蓝牙等方式进行传输。终端设备上产生的计算任务根据计算卸载算法可以选择在本地终端上执行、传输到边缘云上执行或者传输到中心云上执行。但是传统计算卸载方案无法充分利用异地边缘服务器上的剩余计算能力。

在本方案中，我们考虑的是一个多边缘协同卸载的计算卸载场景，该场景在传统的3层架构的基础上，通过本地边缘和中心云联合异地边缘云来实现对任务的协同计算卸载。首先根据边缘云与产生数据的终端距离的远近，将边缘云划分为本地边缘和异地边缘。因此在该场景中，任务可以选择在本地终端上执行，或者卸载到本地边缘端上执行，然后卸载到本地边缘端的任务可以进一步卸载到异地边缘或者中心云上执行。异地边缘相对于中心云，虽然在计算能力上略有差距，但是因其靠近本地边缘，传输时延较低，所以可以将计算任务卸载二次卸载到异地边缘云上。这样一方面不会因为本地边缘端计算能力不足而造成在任务二次卸载到中心云时需要较大的传输时延，同时也在一定程度上充分利用周围拥有空闲计算能力的边缘服务器。系统整体卸载场景如图1所示。

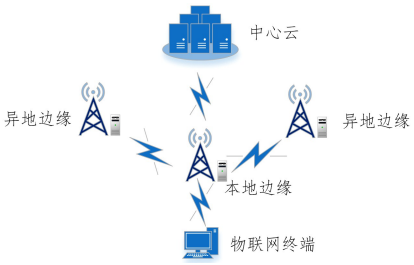

图1 多边缘协同计算卸载场景

Fig. 1 Multi-edge collaborative computing offloading scenario

本方案中的每个计算任务之间拥有关联和依赖关系，因此在本地终端上产生的计算任务可以划分为 $M=\{1,2,3,\cdots,m\}$ 个子任务，这 m 个子任务可能部分在本地执行，部分会卸载到本地边缘云上进行计算，还有可能卸载到远端边缘云和中心云上运行。每一个任务都包含输入数据量大小(D)、计算工作量(L)和反馈计算任务结果的数据量大小(S)3个属性。因此可以使用由 M 个顶点的加权图 $G=(V,E)$ 表示所有子任务之间的关系，图中每个顶点 $i\in V$ 对应其中一个子任务。因为终端设备产生的部分任务可能包含保密信息，具有一定的隐私性，不应该传输卸载到其他设备上进行计算，所以使用变量 $isUnload$ 来标记该任务是否可以卸载。图中每条边 $edge(i,j)$ 表示第 i 个和第 j 个任务之间的存在相互关系。图中箭头指向表示任务之间的依赖关系，它是任务 i 和任务 j 之间的优先约束，即必须先完成任务 i 才能完成任务 j，子任务关系示意图如图2所示。

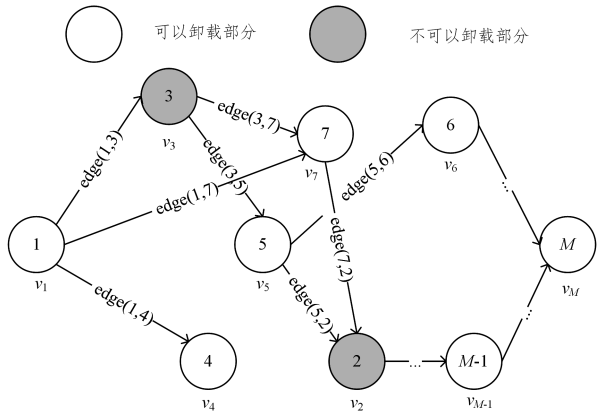

图2 任务关系的有向加权图

Fig. 2 Directed weighted graph of task relationship

3.2 计算卸载模型

3.2.1 模型参数描述

在本文提出的多边缘协同卸载方案中，由于各个边缘云之间卸载的数据量较小，它们之间任务数据传输速率设置为定值 R_e。同时因为本地计算能力有限，很大比例的计算任务需要卸载到本地边缘端，并且存在多个边缘云连接中心云的现象，所以本方案将终端设备与本地边缘之间以及边缘云和中心云之间的计算任务传输速率设置为变量 R_c。边缘云第 n 号上的第 k 个执行任务的时间参数列表如表1所列，其中 n 为边缘云的编号，k 为任务的序列号，l 表示本地终端。

表1 边缘 n 上任务 k 的时间参数列表

Table 1 Time parameter list of task k on edge n

参数名称	参数含义	参数名称	参数含义
$T_{k,l}^{wl}$	任务在本地终端等待时间	$T_{k,n}^{we}$	任务在异地边缘等待时间
$T_{k,l}^{l}$	任务在本地终端执行时间	$T_{k,n}^{e}$	任务在异地边缘执行时间
$T_{k,n}^{wl}$	任务在本地边缘等待时间	$T_{k,n}^{re}$	任务从异地边缘反馈时间
$T_{k,n}^{tl}$	传输到本地边缘时间	$T_{k,n}^{tl}$	任务传输到中心云上时间
$T_{k,n}^{l}$	任务在本地边缘执行时间	$T_{k,n}^{wc}$	任务在中心云上等待时间
$T_{k,n}^{rl}$	任务从本地边缘反馈时间	$T_{k,n}^{c}$	任务在中心云上执行时间
$T_{k,n}^{te}$	任务传输到异地边缘时间	$T_{k,n}^{rc}$	任务从中心云反馈时间

关于能耗方面，设备的能量消耗一般与计算卸载到该设备的任务量、再次卸载到其他设备带来的传输任务量以及CPU的单位能耗有关。为了简化最终卸载方案模型，本文默认CPU在处理以及传输任务的通信过程中始终以最大的CPU频率运行。同时每台设备都有一个静态额定能耗，该值与设备的工作负载无关，只要设备开启运行即会产生用于维持设备的基本功能运作。

此外，为使同一个系统中的各个部分能够协同工作且能发挥总体最大效能，其边缘端(无论是本地边缘还是异地边缘)一般采用同样的设备，所以本地边缘和异地边缘设备的静态功率和在最高CPU时钟频率状态下运行的单位能耗均相同。一方面，因为存在计算结果返回，而计算结果的数据量通常很小，所以所有带来的处理能耗都忽略不计。另一方面为了简化模型，接收计算数据所带来的能耗也都忽略不计。各个节点的能耗参数如表2所列。

表 2 各个节点能耗参数列表

Table 2 List of energy consumption parameters of each node

参数名称	参数含义
K_l	本地终端在 CPU 运行时的计算单位容量对应能耗
K_e	边缘云在 CPU 运行时的计算单位容量对应能耗
K_c	中心云在 CPU 运行时的计算单位容量对应能耗
γ_l	本地终端静态能耗
γ_e	边缘云静态能耗
γ_c	中心云静态能耗

3.2.2 时延模型

(1)本地终端

因为任务本身由本地终端设备产生，所以在本地终端上计算不涉及到计算任务的传输时间，只需要计算任务在本地终端上计算的时间即可。设$f_{m,l}$表示任务 m 在本地终端上的计算能力，该计算能力和终端设备的 CPU 时钟频率成正比，同时设定在本地终端 l 上的计算任务量为$L_{m,l}$，因此任务在本地终端的执行时间$T^{cl}_{m,l}=L_{m,l}/f_{m,l}$。

同时本方案中各个计算子任务之间存在关联和依赖关系，即在任务 m 在本地终端 l 上执行之前，该任务之前所有与之相关的前驱任务应该执行完毕。因为在本文多边协同卸载方案中，一个任务可以选择在本地终端执行、卸载到本地边缘上执行、进一步卸载到异地边缘上执行、卸载到中心云上执行，所以任务在本地终端上执行前的等待时间为所有相关前驱任务完成时间的最大值，即$T^{wl}_{m,l}=\max\limits_{k\in pred(m)}\max\{T^{cl}_{k,l},T^{cl}_{k,n},T^{ce}_{k,n},T^{cc}_{k,n}\}$，其中 $pred(m)$表示在执行任务 m 之前需要完成的相关联任务集合。

(2)边缘云与中心云

对于边缘云和中心云来说，其总体任务计算时间主要包括 3 个步骤的时间消耗。步骤 1 是计算任务传输阶段，本地终端设备将任务传输到本地边缘，本地边缘将任务传输到异地边缘端，本地边缘将任务传输到中心云。步骤 2 为任务计算阶段，即各个边缘云和中心云将计算卸载到自身设备上。步骤 3 为任务反馈阶段，即各个接收计算任务的边缘云和中心云需要将计算结果反馈回去。

1)任务传输阶段

为了计算任务的传输时间，设定任务输入计算量为$D_{m,n}$。同时因为在本方案中本地边缘与异地边缘之间传输卸载的数据量较小，所以设定边缘云之间的传输速率R_e为定值常量。另一方面，因为本地终端到本地边缘之间以及本地边缘云到中心云之间传输的数据量较大，所以将其传输速率设置为变量R_c，这里的R_c和传输任务量以及网络传输速率有关，需要考虑到信道带宽等因素。最后可以得到传输到本地边缘时间$T^{tl}_{m,n}$和传输到中心云时间$T^{tc}_{m,n}$分别为$T^{tl}_{m,n}=T^{tc}_{m,n}=D_{m,n}/R_c$，传输到异地边缘时间$T^{te}_{m,n}=D_{m,n}/R_e$。

2)任务计算阶段

本文设定任务 m 卸载到边缘云 n 的计算量为$L_{m,n}$；$f_{m,n}$表示任务 m 在边缘 n 上的计算处理能力，该值同样与各个边缘云的 CPU 时钟周期成正比；$f_{m,c}$表示中心云的计算能力。因此该阶段任务在本地边缘执行时间$T^{el}_{m,n}=L_{m,n}/f_{m,n}$，在异地边缘任务执行时间$T^{ce}_{m,n}=L_{m,n}/f_{m,n}$，在中心云上执行时间为$T^{cc}_{m,n}=L_{m,n}/f_{m,c}$。

同样，为了计算任务 m，需要其依赖的所有任务执行完毕，而该任务可能处于本地终端设备、本地边缘云、异地边缘云以及云中心这 4 个地方，因此在计算等待延迟时间时，不仅需要考虑任务在对应设备的执行时间，还需要考虑任务卸载到该设备上的时间，而因为任务在本地端产生，所以本文认为任务卸载到本地设备的等待时间为 0。由此任务 m 在本地边缘等待时间为$T^{wl}_{m,n}=\max(T^{tl}_{m,n},\max\limits_{k\in pred(m)}\max\{T^{cl}_{k,l},T^{cl}_{k,n},T^{ce}_{k,n},T^{cc}_{k,n}\})$，任务在异地边缘等待时间为$T^{we}_{m,n}=\max(T^{te}_{m,n},\max\limits_{k\in pred(m)}\max(T^{cl}_{k,l},T^{cl}_{k,n},T^{ce}_{k,n},T^{cc}_{k,n})])$，任务在中心云上等待时间为$T^{wc}_{m,n}=\max(T^{tc}_{m,n},\max\limits_{k\in pred(m)}\max\{T^{cl}_{k,l},T^{cl}_{k,n},T^{ce}_{k,n},T^{cc}_{k,n}\})$。

3)任务运行结果反馈阶段

在任务反馈阶段，需要将卸载到本地边缘、异地边缘以及中心云上的任务运行结果反馈到本地终端设备，设置反馈的计算数据量为$S_{m,n}$，它们之间的数据传输速率仍然为R_e和R_c。因此本地边缘云的计算任务反馈时间$T^{rl}_{k,n}=S_{m,n}/R_e$，异地边缘云的计算任务反馈时间$T^{re}_{m,n}=S_{m,n}/R_e$，中心云的计算反馈时间为$T^{rc}_{m,n}=S_{m,n}/R_c$。

3.2.3 计算卸载的时延目标

本方案通过增加本地边缘与异地边缘之间的协同卸载，提出了边缘云以及中心云的协同卸载方案。该方案的计算卸载目标一共包括两个：最小化计算时延和最小化能量消耗。其中计算时延模型如下文所述。

针对本地终端计算，模型的任务等待时间为$T^{wl}_{m,l}\geqslant a_{k,l}T^{cl}_{k,l}+b_{k,n}T^{cl}_{k,n}+\sum\limits_{i=1}^{N}c^i_{k,n}T^{ce}_{k,n}+d_{k,n}T^{cc}_{k,n}$，$k\in pred(m)$。对于单一任务的卸载，设备具有唯一性，即每个任务最终只能在一台设备上进行计算执行，不可再次拆分，因此用$a_{k,l}+b_{k,n}+\sum\limits_{i=1}^{N}c^i_{k,n}+d_{k,n}=1$ 和$a_{k,l}\in\{0,1\}$，$b_{k,n}\in\{0,1\}$，$c^i_{k,n}\in\{0,1\}$，$d_{k,n}\in\{0,1\}$分别表示任务最终在本地终端、本地边缘、异地边缘以及中心云上的决策结果，其中 0 表示不执行，1 表示执行。

任务 m 在本地终端上执行的完成时间是本地终端计算执行时间和为了完成当前的任务而等待前驱任务完成的时间的和。因此本地终端上执行的完成时间$Z^l_{m,l}$的计算公式如下：

$$Z^l_{m,l}=T^{cl}_{m,l}=T^{el}_{m,l}+T^{wl}_{m,l} \tag{1}$$

针对本地边缘计算，只有在任务完全卸载到本地边缘(所以等待时间肯定大于等于任务传输到本地边缘的时间)时或者在任务 m 所有相关的直接前驱任务都已经完全执行完成之后，本地边缘才能开始执行任务 m。因此等待时间$T^{wl}_{m,n}\geqslant T^{tl}_{m,n}$，即$T^{wl}_{m,n}\geqslant a_{k,l}T^{cl}_{k,l}+b_{k,n}T^{cl}_{k,n}+\sum\limits_{i=1}^{N}c^i_{k,n}T^{ce}_{k,n}+d_{k,n}T^{cc}_{k,n}$，其中，$k\in pred(m)$。

在边缘云 n 上的任务 m 在本地执行的任务完成时间是本地边缘计算中的本地计算执行时间加上等待前驱任务的时间加上等待结果反馈的时间的总和。因此本地边缘计算的完成时间$Z^l_{m,n}$的计算公式如下：

$$Z^l_{m,n}=T^{cl}_{m,n}=T^{el}_{m,n}+T^{wl}_{m,n}+T^{rl}_{m,n} \tag{2}$$

针对异地边缘计算，因为只有在任务完全卸载到同一个异地边缘或者在任务 m 所有相关的直接前驱任务都已经在

边缘执行完成之后，异地边缘才能开始执行任务 m，因此等待时间 $T_{m,n}^{we} \geqslant T_{m,n}^{re}$ 即 $T_{m,n}^{we} \geqslant a_{k,l} T_{k,l}^{cl} + b_{k,n} T_{k,n}^{cl} + \sum_{i=1}^{N} c_{k,n}^{i} T_{k,n}^{ce} + d_{k,n} T_{k,n}^{cc}$，$k \in pred(m)$。

同上，该公式也需要满足卸载设备的唯一性。同理，源自边缘 n 的任务 m 的异地边缘执行的完成时间是异地边缘计算的执行时间、等待前驱任务完成的时间、等待结果反馈的时间之和，所以异地边缘计算的总时间 $Z_{m,n}^{e}$ 的计算公式如下：

$$Z_{m,n}^{e} = T_{m,n}^{ce} = T_{m,n}^{ee} + T_{m,n}^{we} + T_{m,n}^{re} \tag{3}$$

中心云计算的总时间和异地边缘相似，因此中心云计算的总时间 $Z_{m,n}^{c}$ 为：

$$Z_{m,n}^{c} = T_{m,n}^{cc} = T_{m,n}^{ec} + T_{m,n}^{wc} + T_{m,n}^{rc} \tag{4}$$

整个系统的时间为本地终端花费时间、本地边缘花费时间、异地边缘花费时间、中心云花费时间之和，总的花费时间 T_n 的值为：

$$T_n = \sum_{m=1}^{M} Z_{m,n} = \sum_{m=l}^{M} a_{m,l} Z_{m,l}^{l} + b_{m,n} Z_{m,n}^{l} + c_{m,n} Z_{m,n}^{e} + d_{m,n} Z_{m,n}^{C} \tag{5}$$

为了保证任何一个任务只能在一个地方执行，有以下约束条件：$a_{m,l} + b_{m,n} + c_{m,n} + d_{m,n} = 1$ 和 $a_{m,l} \in \{0,1\}$，$b_{m,n} \in \{0,1\}$，$c_{m,n} \in \{0,1\}$，$d_{m,n} \in \{0,1\}$。

综上，整体而言该问题是一个混合整数非线性优化问题。总的时间上的目标函数的具体计算公式如下：

$$Min \sum_{n=1}^{N} T_n = \min \sum_{n=1}^{N} \sum_{m=1}^{M} a_{m,l} Z_{m,l}^{l} + b_{m,n} Z_{m,n}^{l} + c_{m,n} Z_{m,n}^{e} + d_{m,n} Z_{m,n}^{c} \tag{6}$$

3.2.4 能耗模型

针对本地终端，本文设 $f_{m,l}$ 表示任务 m 在本地终端上的计算能力，该计算能力和终端设备的 CPU 时钟频率成正比，同时设定任务 m 在本地终端 l 上的计算任务量为 $L_{m,l}$，即为了处理卸载到本地终端上的任务 m 的任务量 $L_{m,l}$，设备在以最高时钟频率 $f_{m,l}$ 运行的情况下对应产生的计算能量消耗为 $E_{k,l}^{c,l} = \gamma_l + L_{m,l} * k_l$。

因为在通常情况下计算所需的数据就是在本地终端上产生的，所以卸载决策阶段会将部分数据卸载到本地边缘端进行处理，因此数据传输到本地边缘的过程同样会消耗部分能量。为了计算任务的传输能耗，设定任务输入到本地边缘的计算量为 $D_{m,n}$，同时因为计算能耗中已经包含基本能耗，所以不再添加，因此最终的传输能耗计算公式为 $E_{k,n}^{t,l} = D_{m,n} * k_l$。

每个任务只能选择在当前设备执行或者卸载到其他设备上，每一个任务所带来的能量消耗不超过执行该任务的传输能耗。因此本地终端设备上的总能耗为计算能耗和传输任务到本地边缘带来的传输能耗中的最大值，具体计算公式如下：

$$E_{m,l}^{l} = \max\{a_{m,l} * E_{m,l}^{c,l}, (b_{m,n} + \sum_{i=1}^{N} C_{m,n}^{i} + d_{m,n}) * E_{m,n}^{t,l}\} \tag{7}$$

设定任务 m 卸载到本地边缘云 n 的计算量为 $L_{m,n}$。因为本地边缘会将部分任务卸载到异地边缘和中心云上，设卸载的数据量为 $D_{m,n}$，所以针对本地边缘，计算 $L_{m,n}$ 任务带来的计算能耗为 $E_{m,n}^{c,l} = \gamma_e + L_{m,n} * k_e$，本地边缘传输到异地边缘带来的能耗为 $E_{m,n}^{t,e} = D_{m,n} * k_e$。同样因为一个任务只能选择在本地边缘执行或者传输到异地边缘或中心云上执行，所以取两者的较大值为本地边缘上的能量消耗，具体计算公式如下：

$$E_{m,n}^{l} = \max\{b_{m,n} * E_{m,n}^{c,l}, (\sum_{i=1}^{N} C_{m,n}^{i} + d_{m,n}) * E_{m,n}^{t,e}\} \tag{8}$$

同样设置从本地边缘卸载到异地边缘上的任务量大小为 $L_{m,n}$，该任务只会带来异地边缘的计算能耗 $E_{k,n}^{c,e} = \gamma_e + L_{m,n} * k_e$，因此异地边缘端的能耗的计算公式如下：

$$E_{m,n}^{e} = (\sum_{i=1}^{N} C_{m,n}^{i}) * E_{m,n}^{c,e} \tag{9}$$

同理，输入中心云的 $L_{m,n}$ 大小的任务量带来的中心云的计算能耗为 $E_{k,n}^{c,c} = \gamma_c + L_{m,n} * k_c$，因此中心云的能耗的计算公式如下：

$$E_{m,n}^{c} = d_{m,n} * E_{m,n}^{c,c} \tag{10}$$

因此整个系统的能耗的计算公式如下：

$$E_n = \sum_{m=1}^{M} a_{m,l} E_{m,l}^{l} + b_{m,n} E_{m,n}^{l} + c_{m,n} E_{m,n}^{e} + d_{m,n} E_{m,n}^{c} \tag{11}$$

能耗上总的目标函数的具体计算公式如下：

$$Min \sum_{n=1}^{N} E_n = \min \sum_{n=1}^{N} \sum_{m=1}^{M} a_{m,l} E_{m,l}^{l} + b_{m,n} E_{m,n}^{l} + c_{m,n} E_{m,n}^{e} + d_{m,n} E_{m,n}^{c} \tag{12}$$

整个系统的目标函数的计算公式如下：

$$Min \sum_{n=1}^{N} \{\beta T_n + (1-\beta) E_n\} \tag{13}$$

系统的约束条件的具体计算公式如下：

$$\sum_{m=1}^{M} (a_{m,l} T_{m,l}^{cl} + b_{m,n} T_{m,n}^{cl} + \sum_{i=1}^{N} c_{m,n}^{i} T_{m,n}^{ce} + d_{m,n} T_{m,n}^{cc}) \leqslant T_{n,\max} \tag{14}$$

$$\sum_{m=1}^{M} a_{m,l} E_{m,l}^{l} + b_{m,n} E_{m,n}^{l} + c_{m,n} E_{m,n}^{e} + d_{m,n} E_{m,n}^{c} \leqslant E_{n,\max} \tag{15}$$

$$a_{k,l} T_{k,l}^{cl} + b_{k,n} T_{k,n}^{cl} + \sum_{i=1}^{N} c_{k,n}^{i} T_{k,n}^{ce} + d_{k,n} T_{k,n}^{cc} \leqslant T_{m,l}^{ul}, k \in pred(m) \tag{16}$$

$$T_{m,n}^{tl} \leqslant T_{m,n}^{ul}$$

$$a_{k,l} T_{k,l}^{cl} + b_{k,n} T_{k,n}^{cl} + \sum_{i=1}^{N} c_{k,n}^{i} T_{k,n}^{ce} + d_{k,n} T_{k,n}^{cc} \leqslant T_{m,n}^{ul} \tag{17}$$

$$T_{m,n}^{te} \leqslant T_{m,n}^{we}$$

$$a_{k,l} T_{k,l}^{cl} + b_{k,n} T_{k,n}^{cl} + \sum_{i=1}^{N} c_{k,n}^{i} T_{k,n}^{ce} + d_{k,n} T_{k,n}^{cc} \leqslant T_{m,n}^{we} \tag{18}$$

$$T_{m,n}^{tc} \leqslant T_{m,n}^{wc}$$

$$a_{k,l} T_{k,l}^{cl} + b_{k,n} T_{k,n}^{cl} + \sum_{i=1}^{N} c_{k,n}^{i} T_{k,n}^{ce} + d_{k,n} T_{k,n}^{cc} \leqslant T_{m,n}^{wc} \tag{19}$$

$$a_{m,l} \in \{0,1\}, b_{m,n} \in \{0,1\}, c_{m,n} \in \{0,1\}, d_{m,n} \in \{0,1\} \tag{20}$$

$$a_{m,l} + b_{m,n} + c_{m,n} + d_{m,n} = 1 \tag{21}$$

其中，约束(14)表示总时间小于所需的最大完成时间 $T_{n,\max}$。约束(15)表示总能耗小于所需的最大能量消耗 $E_{n,\max}$。约束(16)确保了任务 m 只能在其所有相关的直接前驱任务完全执行完毕之后才能执行。约束(17)－(19)表示任务卸载完成之后才能执行，即对于任务 m 只有在任务完全卸载到本地边缘云、异地边缘云或者中心云之后才能开始在本地边缘云、异地边缘云和中心云上执行，或者等待所有与任务 m 相关的前驱任务已完全在本地边缘云、异地边缘云端或者中心云上执行完成后才能执行。约束(20)中的 4 个变量中分别使用 0 和 1 来表示计算任务是否在本地终端执行以及是否卸载到本地边缘、异地边缘、中心云上。约束(21)保证计算任务只能在本地终端或者本地边缘云服务器或者异地边缘云服务器或者中

心云服务器这 4 个服务器中的一个上执行。其中系统的最大完成时间 $T_{n,\max}$ 和最大能量消耗 $E_{n,\max}$ 为系统预先设定值，本地终端、本地边缘云、异地边缘云的 CPU 频率是根据具体场景进行设定的动态值。最后通过求解各个部分的时间和能耗值使式(13)的系统总代价达到最小。

4 遗传算法

对系统数学模型进行分析发现本文方案所求解的目标问题为 NP-Hard 问题[14]，因此直接计算求解得到最佳计算卸载决策的难度比较大。而启发式算法的优点在于它比盲目型的搜索法更高效，启发式函数可以在较短的时间内得到一个搜索问题的最优解，对于 NP 问题，其亦可在多项式时间内得到一个较优解。因此，本文通过基于改进的遗传算法来进行模型求解。

遗传算法(Genetic Algorithm，GA)是一种通过借鉴进化生物学而发展起来的随机全局搜索和优化方法[15]。因为该算法直接以群体中的所有个体为对象，即直接对结构对象进行操作，所以不存在求导和函数连续性的限定，同时其利用随机化技术指导对一个被编码的参数空间进行高效搜索，因此不需要确定的规则就可以获取指导优化的搜索空间，同时其还可以根据种群选择来自动调整搜索方向，拥有隐性的并行性和更加优良的全局寻优能力。另一方面，相比其他的启发算法，遗传算法通过基因突变来尽量避免在求解过程中陷入局部最优的情况，使得结果更加趋近于全局最优解，因此其一般用于解决复杂的非线性优化问题。该算法的主要步骤包括：遗传、突变、自然选择和杂交等。针对本方案中的计算卸载场景，需要对遗传算法中的参数编码、初始种群、适应度函数、遗传操作控制、控制参数等主要步骤进行针对性的设定，从而使其满足求解本文所提方案的目标模型的需求。针对边缘计算任务的关联性特点，为了防止过早陷入局部最优，在原始的遗传算法的基础上，本文通过增加“突变”步骤来引入新的变量，从而使得卸载算法形成新的性状，同时通过实验仿真得到最优的突变因子，以保证优化之后的遗传算法能够得到计算卸载方案的最优解。

4.1 参数编码与初始种群设定

在本文提出的协同卸载的计算场景中，主要目标是选取最优的计算卸载策略，因此本文将每一种可能的计算卸载方案作为遗传算法中的个体，同时将计算卸载方案进行编码之后作为遗传算法中的染色体，并将每个计算任务最终的卸载设备作为遗传算法中的基因。

另一方面，因为每个具体的任务可以被划分为 M 个子任务，所以每条染色体由 M 个基因组成，而每个子任务经过卸载决策之后可以选择在本地终端设备计算、本地边缘计算、异地边缘计算和传输到中心云计算，因此每个基因包含 4 种可能值：$-2,-1,0,1$，即 M 个基因组成的染色体也由 4 种值按照某种排序组成。本文将初始种群设定为 M。

4.2 适应度函数

遗传算法中的适应度用来度量每个物种对于生存环境的适应性，通过适应度函数的值来判断并淘汰适应程度较差的生物。式(22)中分母部分即为每个卸载方案的目标函数，因此本文中每个卸载方案的适应度 $f(n)$ 是目标函数模型的倒数。适应度函数值越高时，目标函数模型的值就越小，表示该计算卸载方案的整体代价越小，即该卸载方案越优。$f(n)$ 的具体计算公式如下：

$$f(n)=\frac{1}{Min\sum_{n=1}^{N}\beta T_n+(1-\beta)E_n} \tag{22}$$

在计算完适应度函数后，根据计算值淘汰适应度较差的、一般的染色体，即卸载方案，保留适应度较高的染色体，这样经过若干次迭代后的染色体的质量越来越高，即卸载方案的整体代价越来越小。

4.3 交叉过程

遗传算法通过交叉过程来迭代产生新的染色体，交叉过程需要从符合适应度函数的未被淘汰的染色体中选择两条，然后将这两条染色体的某个位置切断并且拼接到一起形成新的染色体，即新的染色体包含来自两条染色体的部分基因。

为了保留更加优良的基因，在选择来源染色体时，本文按照式(23)来计算每一条染色体的适应度概率：

$$p(i)=\frac{f(i)}{\sum_{i=0}^{M}f(i)} \tag{23}$$

其中，$p(i)$ 表示染色体 i 被选择的概率，$f(i)$ 为通过适应度函数计算得到的染色体 i 的适应度，$\sum_{i=0}^{M}f(i)$ 为所有染色体的适应度之和。

当适应度较高的染色体被选择作为下一代父母染色体的概率越大时，迭代产生的卸载方案也会变得越优质。交叉过程示例如图 3 所示。

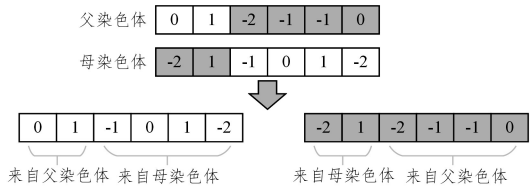

图 3 交叉过程示例

Fig. 3 Example of crossover process

4.4 变异过程

遗传算法中的交叉过程可以保证每次迭代进化留下相对优良的基因，但是因为其仅仅是对上一步的结果集进行选择，所以在交换和多次进化之后会出现“早熟”的现象，从而只能得到近似于局部最优解。为了得到全局最优解，本文在交叉形成一个新的染色体后，在新的染色体上随机选择若干个基因，然后随机修改基因的值，从而通过给现有的染色体引入新的基因来突破当前搜索的限制，有利于算法得到全局最优解。

4.5 终止条件

因为遗传算法需要设置终止条件，所以每次染色体变异之后需要计算其适应度函数值。如果已经到达了设定的最大迭代次数或者很长时间内适应度值一直保持不变，那么整个遗传算法流程终止，即得到符合条件的全局最优解，否则继续进行迭代。完整的算法流程如图 4 所示。

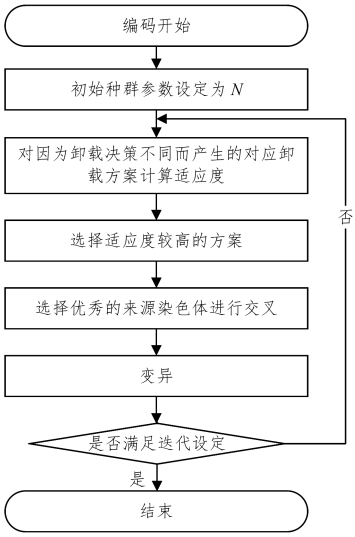

图 4 遗传算法流程

Fig. 4 Genetic algorithm process

5 仿真和评估

5.1 参数设置

根据卸载算法特点，本文仿真场景主要包括一个本地终端节点、一个本地边缘节点、多个异地边缘节点和一个中心云。由于异地边缘节点相对其他某一个本地终端而言就是其自身的本地边缘节点，同时为了方便各个异地边缘节点的通信和计算，通常情况下本地边缘节点和异地边缘节点的硬件设备完全一致。本仿真实验中涉及到较多的物联网设备节点，系统的基本参数设定基于文献[16-17]中提供的参数示例，涉及到协同卸载的部分参数参考文献[18]，并且本文根据仿真环境进行了一定的调整。实验涉及到的遗传算法的参数均通过详细的仿真过程获得。具体实验参数如表 3 所列。

表 3 仿真实验参数

Table 3 Simulation experiment parameters

实验参数	取值
本地终端节点静态 CPU 功耗/mW	10
边缘端节点静态 CPU 功耗/mW	20
中心云节点静态 CPU 功耗/mW	40
本地终端节点运行时 CPU 功耗/mW	90
边缘端节点运行时 CPU 功耗/mW	200
中心云节点运行时 CPU 功耗/mW	400
本地终端节点 CPU 频率/(cycles/s)	1 GHz
边缘(包括本地边缘和异地边缘)节点 CPU 频率 f_e/(cycles/s)	4 GHz
中心云 CPU 频率 f_c/(cycles/s)	8 GHz
本地边缘和异地边缘之间传输速率 R_e/(MB/s)	5
种群数目 M	60
交叉概率	0.8
最大迭代次数	100

5.2 结果分析

首先针对算法本身主要的变量变异概率进行仿真实验，即验证遗传算法的参数对 GAMCCOM 方案的影响。初始种群数目 M 为 60，最大迭代次数为 100，同时设置时延因子 β 为 0.5，所以能耗因子也为 0.5，设置变异概率的概率空间为[0,0.1]，仿真结果如图 5 所示。由图 5 可知，当变异概率在[0,0.026]之间时，系统模型的总体代价不断下降，因为变异概率过低时会导致“早熟”，从而陷入局部最优，无法获得最佳的系统代价，随着变异概率的增加，算法结果会不断趋向于最佳的系统代价。当变异概率在[0.026,0.1]之间时，系统整体的代价处于波动上升阶段，因为当变异概率较大时候会增加染色体中的基因随机性，导致每次迭代之后形成的新的染色体相比于原来的染色体不一定更优，即不一定更加趋近全局最优解，从而无法保证每次求解结果逐步趋近于全局最优解。因此，该部分的系统代价不断波动上升。

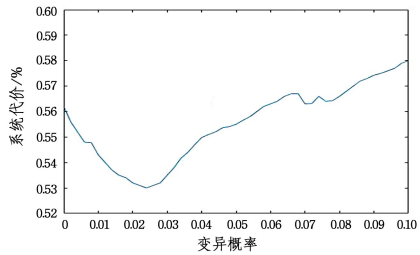

图 5 变异概率分布

Fig. 5 Probability distribution of mutation

然后本文将 GAMCCCO 方案和以下卸载方案进行对比。

方案 1 纯本地卸载(Local)：所有的计算任务均在本地执行。

方案 2 随机卸载(Random)：将本地终端产生的所有任务随机分配到各个节点上执行[19]。

方案 3 传统卸载方案(Tradition)：包含本地节点、边缘节点、中心云 3 层架构的传统计算卸载方案[20]。

方案 4 部分联合卸载(Partial)：该方案仅考虑在边缘节点、中心云架构中使用联合协同卸载方案[11]。

在方案对比的过程中，设置系统代价的时延因子和能耗因子的比例均为 0.5。因为在物联网中，例如在自动驾驶场景下，数据处理的时延和设备的能耗通常均是考虑的重点，所以在该比例之下可以对整个卸载算法做一个衡量。同时根据图 5 仿真结果可知，当变异概率在 0.026 时，整个方案的系统代价最低，因此本次实验对比设置传统卸载方案和本文提出的 GAMCCCO 方案的遗传算法的变异概率均为 0.026。

同时，本次仿真以本地计算卸载的整体系统代价为基准，其他卸载方案的代价分别与本地计算卸载进行对比。仿真结果如图 6 所示。

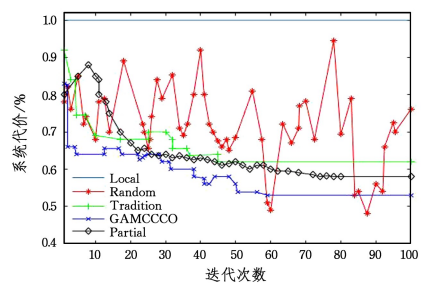

图 6 各个卸载方案对比

Fig. 6 Comparison of various uninstall schemes

由图 6 可知，随机卸载算法相比本地计算卸载实现了较小的系统代价，但是因为其具有很强的随机性，所以系统代价不稳定。部分联合卸载方案中的卸载算法使用了均方误差作为预测模型的依据，因此随着迭代次数的增加，其系统代价不断下降，最终趋于稳定。但是因为其没有将本地终端设备纳入卸载系统，所以即使最终算法实现完全拟合，其系统代价仍然较高。相比于传统的三层卸载方案，本文提出的 GAMCCCO 方案获得了最小的系统代价。虽然随机卸载算法在偶然情况下获得的系统代价更低，但是该算法整体的代价远高于本文方案。因此，总体而言本文提出的 GAMCCCO 计算卸载方案获得了最小的系统代价。

针对方案模型中时延因子和能耗因子对卸载方案的影响，本文采用了上述仿真的参数，即时延因子和能耗因子均为 0.5。下面就时延因子为 1、能耗因子为 0，以及时延因子为 0、能耗因子为 1 的情况进行了仿真比较，结果如图 7、图 8 所示。

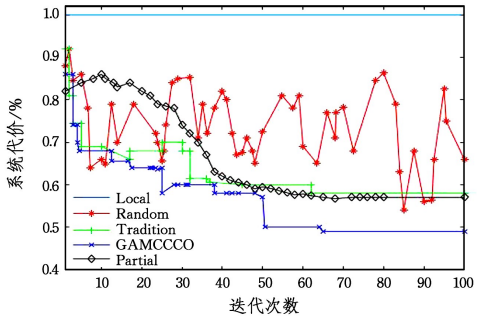

图 7 只考虑时延时各方案的对比

Fig. 7 Comparison of various schemes considering only time delay

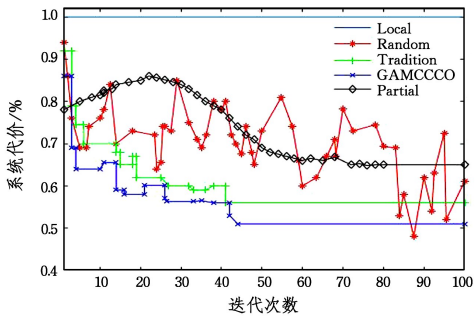

图 8 只考虑能耗时各方案的对比

Fig. 8 Comparison of various schemesconsidering only energy consumption

由图 7 和图 8 可知，无论是在只考虑时延的情况下，还是在只考虑能耗的情况下，GAMCCCO 卸载方案得到的系统代价都小于其他卸载方案。

上述仿真环境中采用的异地边缘节点个数为 4，因为如果异地边缘节点个数为 0 则不能进行协同卸载，所以分别针对边缘节点个数为 1～10 的情况进行了系统仿真，分析在不同异地边缘节点个数的情况下各个方案的系统代价（因为随机卸载的波动性较大，因此取平均值作为其最终代价）。从图 9 的仿真结果可知，本地卸载方案和传统的卸载方案不受异地边缘节点个数的影响，而随机卸载方案的结果受其影响较大，具有很大的随机性。方案 4 中的部分协同卸载随着异地边缘节点个数的增加，其系统代价会下降。而本文提出的 GAMCCCO 卸载方案在相同数据量下，当异地节点个数为 4 时，系统代价处于最优；当节点数大于 4 时，系统代价虽有小幅度波动，但整体保持平稳。因为较多的异地边缘节点会使得计算任务卸载较为分散，增加了任务传输的成本，所以对于最终卸载结果有微弱的影响。

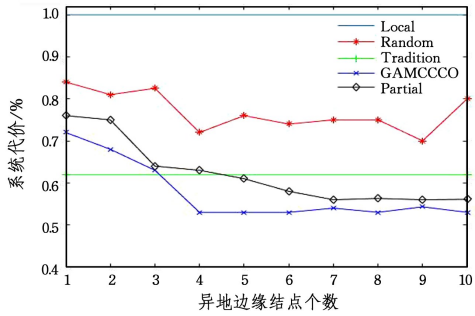

图 9 异地边缘节点个数对于系统代价影响

Fig. 9 Impact of number of remote edge nodes on system cost

结束语 本文针对计算卸载不均衡场景下出现的异地边缘利用率不充分的问题，提出了一种联合本地终端、本地边缘云、异地边缘云、中心云的多边联合卸载方案 GAMCCCO，该方案使用有向加权图构建各个计算任务的直接关系，然后通过引入空闲的异地边缘来降低整个系统的能耗和时间代价。通过仿真可以得到在同等情况下，该方案的系统代价相比其他方案有明显的降低。但是本文方案只考虑了单节点的卸载过程，未来将考虑多节点之间的联合卸载过程，验证在多数据节点联合卸载的情况下系统的性能，从而进一步完善卸载模型。该问题在模型上可以尝试把部分数据源相似的节点归并为同一类节点，在卸载算法上可以采用机器学习的相关算法对一段时间的数据进行统计分析，然后对卸载的数据进行预测，从而使系统达到最优。

参 考 文 献

[1] XIE R C, LIAN X F, JIA Q M, et al. Overview of mobile edge computing offloading technology [J]. Journal on Communications, 2018, 39(11): 138-155.

[2] YUW, HE L F. A Survey on the Edge Computing for the Internet of Things [J]. IEEE Access, 2017, 6(8): 6900-6919.

[3] MARTINA M, ALEKSANDAR A, LVANA P Z, et al. Edge Computing Architecture for Mobile Crowdsensing [J]. IEEE Access, 2018, 6(5): 10662-10674.

[4] MAO Y Y, YOU C S, ZHANG J, et al. A Survey on Mobile Edge Computing: The Communication Perspective [J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322-2358.

[5] PAN J L, JAMES M. Future Edge Cloud and Edge Computing for Internet of Things Applications[J]. IEEE Internet of Things Journal, 2018, 5(1): 439-449.

[6] LIU J, MAO Y Y, ZHANG J, et al. Delay-optimal computation task scheduling for mobile-edge computing systems [J]. IEEE

International Symposium on Information Theory (ISIT), 2016, 1(4): 1451-1455.

[7] YOU C S, HUANG K B. Exploiting Non-Causal CPU-State Information for Energy-Efficient Mobile Cooperative Computing [J]. IEEE Transactions on Wireless Communications, 2018, 17(6): 4104-4117.

[8] CHEN L X, ZHOU S, XU J. Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks [J]. IEEE/ACM Transactions on Networking, 2018, 26(4): 1619-1932.

[9] ZHOU J S, TIAN D X, WANG Y P. et al. Reliability-Optimal Cooperative Communication and Computing in Connected Vehicle Systems [J]. IEEE Transactions on Mobile Computing, 2020, 19(5): 1216-1232.

[10] CAO X W, WANG F, XU J, et al. Joint computation and communication cooperation for energy-efficient mobile edge computing [J]. IEEE Internet of Things Journal, 2019, 6(3): 4188-4200.

[11] DAI Y Y, XU D, MAHARJAN S, et al. Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks [J]. IEEE Internet of Things Journal, 2019, 6(3): 4377-4387.

[12] ZOU S. Research and Implementation of Computing Offloading and Image Cache Method in Edge Computing Platform [D]. Beijing: Beijing University of Posts and Telecommunications, 2019.

[13] WU Z K, JIANG L Y, MU Y R. Research on application unloading algorithm with multi edge nodes [J]. Journal of Nanjing University of Posts and Telecommunications: Natural Science Edition, 2019, 39(4): 96-102.

[14] CHEN B, QUAN G R. NP-Hard Problems of Learning from Examples [C]// 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery. 2008, 2: 182-186.

[15] DEB K, PRATAP A, AGARWAL S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II [J]. IEEE Transactions on Evolutionary Computation, 2002, 6(2): 182-197.

[16] LE G X, DAI Y S, YANG X H, et al. Modeling of trusted collaborative service strategy in edge computing [J]. Computer Research and Development, 2020, 57(5): 1080-1102.

[17] SONG Y Z, STEPHEN S Y, YU R Z, et al. An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications [J]. IEEE International Conference on Edge Computing (EDGE), 2017, 13(5): 32-39.

[18] GUO H Z, LIU J J. Collaborative Computation Offloading for Multiaccess Edge Computing Over Fiber-Wireless Networks [J]. IEEE Transactions on Vehicular Technology, 2018, 67(5): 4514-4526.

[19] GERUTTI G, PRASAD R, BRUTTI A, et al. Compact Recurrent Neural Networks for Acoustic Event Detection on Low-Energy Low-Complexity Platforms [J]. IEEE Journal of Selected Topics in Signal Processing, 2020, 14(4): 654-664.

[20] BADRI H, BAHREINI T, GROSU D, et al. Energy-Aware Application Placement in Mobile Edge Computing: A Stochastic Optimization Approach [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(4): 909-922.

GAO Ji-xu, born in 1996, postgraduate. His main research interests include mobile edge computing and IoT.

WANG Jun, born in 1975, Ph. D, associate professor. Her main research interests include network architecture of IoT and wireless sensor networks.