软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(6):1889−1908 [doi: 10.13328/j.cnki.jos.005705]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

移动边缘计算中资源受限的串行任务卸载策略*

刘 伟[1,2], 黄宇成[1], 杜 薇[1,2], 王 伟[3]

[1](武汉理工大学 计算机科学与技术学院,湖北 武汉 430070)

[2](交通物联网技术湖北省重点实验室,湖北 武汉 430070)

[3](同济大学 计算机科学与技术系,上海 200092)

通讯作者: 杜薇, E-mail: whutduwei@whut.edu.cn

摘 要: 云计算和移动互联网的不断融合,促进了移动云计算的产生和发展,但是其难以满足终端应用对带宽和延迟的需求.移动边缘计算在靠近用户的网络边缘提供计算和存储能力,通过计算卸载,将终端任务迁移至边缘服务器上面执行,能够有效降低应用延迟和节约终端能耗.然而,目前针对移动边缘环境任务卸载的主要工作大多考虑单个移动终端和边缘服务器资源无限的场景,这在实际应用中存在一定的局限性.因此,针对边缘服务器资源受限下的任务卸载问题,提出了一种面向多用户的串行任务动态卸载策略(multi-user serial task dynamic offloading strategy,简称 MSTDOS).该策略以应用的完成时间和移动终端的能量消耗作为评价指标,遵循先来先服务的原则,采用化学反应优化算法求解,充分考虑多用户请求对服务器资源的竞争关系,动态调整选择策略,为应用做出近似最优的卸载决策.仿真结果表明,MSTDOS 策略比已有算法能够取得更好的应用性能.

关键词: 移动边缘计算;资源受限;串行任务;任务卸载;资源分配

中图法分类号: TP316

中文引用格式: 刘伟,黄宇成,杜薇,王伟.移动边缘计算中资源受限的串行任务卸载策略.软件学报,2020,31(6):1889−1908. http://www.jos.org.cn/1000-9825/5705.htm

英文引用格式: Liu W, Huang YC, Du W, Wang W. Resource-Constrained serial task offload strategy in mobile edge computing. Ruan Jian Xue Bao/Journal of Software, 2020,31(6):1889−1908 (in Chinese). http://www.jos.org.cn/1000-9825/5705.htm

Resource-Constrained Serial Task Offload Strategy in Mobile Edge Computing

LIU Wei[1,2], HUANG Yu-Cheng[1], DU Wei[1,2], WANG Wei[3]

[1](School of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China)

[2](Hubei Key Laboratory of Transportation Internet of Things, Wuhan 430070, China)

[3](Department of Computer Science and Technology, Tongji University, Shanghai 200092, China)

Abstract: The continuous integration of cloud computing and mobile Internet promotes the generation and development of mobile cloud computing (MCC), but it is difficult to meet the demand for bandwidth and delay of terminal applications. Mobile edge computing (MEC) provides computing and storage capabilities at the edge of the user's network. By computing offloading, the terminal task is migrated to the edge server for execution, which can effectively reduce application delay and conserve terminal energy consumption. However, this has certain limitations in practical applications for existing works that focus on a single mobile terminal and assume the server's resources are sufficient for task offloading on MEC environment. This study focuses on the task offloading problem under the resource-constrained MEC environment and proposes a multi-user serial task dynamic offloading strategy (MSTDOS). The strategy uses the completion time of the application and the energy consumption of the mobile terminal as evaluation indicators, follows the principle of first come first served,

* 基金项目: 国家自然科学基金(61672384); 教育部人文社科基金(16YJCZH014); 中央高校基本科研业务费专项资金(2016III028, 2017III028-005)

Foundation item: National Natural Science Foundation of China (61672384); Ministry of Education Project of Humanities and Social Sciences (16YJCZH014); Fundamental Research Funds for the Central Universities (2016III028, 2017III028-005)

收稿时间: 2018-06-26; 修改时间: 2018-08-21; 采用时间: 2018-10-18

uses a chemical reaction optimization algorithm to solve, while can make a near-optimal offloading strategy for the application by consider the interactionamong multiple terminals and dynamically adjust the selection decision. Simulation results show that MSTDOS strategy can achieve better application performance than existing algorithms.

Key words: mobile edge computing; limited resources; serial task; task offloading; resource allocation

据思科视觉网络指数预测,到 2021 年,全球移动数据流量每月将达到 49 艾字节(EB),这对移动终端的性能和移动网络的数据传输率提出了新的要求[1].此外,一类新的应用程序也应运而生,诸如人脸识别[2]、移动增强现实[3]等,这类应用对延迟比较敏感,且需要大量的资源.然而,移动终端的计算和存储资源有限,尤其是电池的续航周期较短,运行这类应用程序会带来较高的延迟,增加移动终端的能耗.而移动云计算(mobile cloud computing,简称 MCC)[4]的出现,为解决这些问题提供了一种有效方式,通过任务卸载(也称计算卸载或计算迁移)[5,6]将应用程序的部分计算任务卸载到远程云端,减少了应用程序的完成时间,提高了移动终端的续航能力.

然而在传统移动云计算中,移动终端通过互联网与远程公有云连接,如阿里云、腾讯云等,由于集中式部署的远程公有云距离移动终端较远,在两者之间通过互联网进行数据传输往往会带来比较高的网络延迟[7],这对于延迟敏感型的移动应用程序无法容忍.为了解决 MCC 存在的高延迟问题,诞生了一种新的计算模式——移动边缘计算(mobile edge computing,简称 MEC)[8],在靠近移动终端的网络边缘部署边缘服务器,将远程云端提供的 IT 服务和计算能力延伸到距离移动终端更近的位置,移动终端便能够利用边缘服务器强大的资源,并以较低的网络延迟与边缘服务器进行数据传输.当终端提交卸载请求时,可以直接在边缘服务器上处理而不需要发送给远程云端,节省了应用完成时间和终端能耗.MEC 既有强大的计算和存储能力,又有靠近用户的高带宽、低时延的优势,能够显著地提高用户的实时体验和满意度.国内相关学者在边缘计算体系结构等领域取得了一定的研究成果[9].但是,边缘计算在移动性管理、虚拟化技术和计算卸载等领域还存在一定的技术挑战[10].

学者们针对边缘服务器环境中的任务卸载问题,从不同的角度开展研究工作[11−24].然而,这些工作大多针对单个移动终端或边缘服务器资源无限的场景.与传统的云计算相比,边缘服务器提供的计算、网络及存储资源是有限的.此外,当有大量用户同时向边缘服务器提交卸载请求时,用户之间会对边缘服务器的资源产生竞争,导致资源的不足而带来应用性能的衰减.因此,考虑多用户同时向资源受限的边缘服务器提交卸载请求的场景是非常有必要的.但是,与已存在的工作相比,这将导致计算卸载情况更加复杂.

本文针对边缘服务器资源受限场景下的任务卸载和资源分配问题展开研究,提出了一种面向多用户的串行任务动态卸载策略(MSTDOS).此策略考虑了多用户同时请求的竞争关系和任务卸载决策与服务器资源分配之间的相互影响,以应用的平均完成时间和移动终端的平均能量消耗作为评价指标,采用化学反应优化算法求解.该策略遵循先来先服务原则,合理分配边缘服务器的资源,减少任务在服务器上的等待时间,从而使用户得到一个更好的卸载结果,提高用户体验和应用性能.

本文的主要贡献如下.

1) 解决了边缘服务器资源受限时的多终端串行任务卸载问题.本文建立了串行任务模型、应用完成时间模型和终端能耗模型,旨在降低应用平均完成时间和终端平均能耗;同时,采用线性加权的方法将多目标优化问题归一为单目标优化问题,并证明该问题属于 NP-hard;
2) 提出了一种面向多用户的串行任务动态卸载策略 MSTDOS.该策略基于化学反应优化算法,在满足组件间依赖关系的前提下,充分考虑多用户的竞争关系和任务卸载决策与资源分配的相互影响,提高了应用性能和用户体验;
3) 验证了 MSTDOS 策略的有效性.本文选取现实生活中的人脸识别应用作为评估对象,仿真实验表明:在边缘服务器资源受限的多终端任务卸载场景下,MSTDOS 策略比存在的方法在应用性能提升上有明显优势.

本文第 1 节介绍国内外的研究现状.第 2 节描述任务卸载的场景及相关概念.第 3 节阐述具体的系统模型和问题定义.第 4 节重点介绍本文提出的卸载策略.第 5 节分析实验环境和评估结果.最后,对本文工作进行总结和展望.

1 相关工作

近年来,任务卸载[11−24]问题在学术界引起了广泛的关注.为了扩展移动终端的计算能力,延长终端的续航周期,相关研究者们提出了多种任务卸载框架,如 MAUI[11],ThinkAir[12],CloneCloud[13]等,关于计算迁移的研究方法和策略也开展了相关工作.浙江大学的邓水光等人[14]考虑到移动云环境中用户的移动性,提出了一种包含容错机制的卸载系统,保障了应用的稳定执行,并采用遗传算法为应用做出卸载决策.北京大学的黄罡团队[15]针对Web 应用中由复杂脚本导致任务计算量过大的问题,将 Web 应用动态变化的特征和计算卸载相结合,提出了一种应用运行时的卸载策略从而得到 Web 应用的卸载结果.南洋理工大学的张维文等人[16]研究了移动云环境中基于随机无线信道的任务卸载问题,分别采用穷举法和 one-climb 策略为应用做出卸载决策,确定在截止时间约束下能耗最优的选择结果.文献[17]针对由多用户资源竞争而导致的高能耗问题,提出了一种任务联合执行策略,将联合执行应用的优化问题建模为最小化终端能耗问题,并采用遗传算法来处理该优化问题.武汉大学的傅建明团队[18]对 CM-MCC 和 CA-MCC 两类弹性移动云计算方式展开了研究,分析了其实现流程及存在的问题;同时,针对弹性移动云计算的安全问题展开分析,并研究相应防御方法.北京邮电大学的王尚广团队[19]针对已有服务选择工作没有考虑到移动终端的状态和历史特征信息的问题,提出了一种状态感知和稳定性感知的选择方法,分别对任务在移动终端和云端运行两种情况建模,并设计了一种选择算法决定最优的服务.

由于集中式部署的远程公有云距离移动终端较远,他们之间的数据传输将带来较高的网络延迟.借助于移动边缘计算,将计算和存储服务部署在移动网络的边缘,能有效弥补传统云计算的不足.韦恩州立大学的施巍松等人[8]针对集中式数据处理方式无法满足大量数据的实时处理需求问题,提出了面向边缘设备的边缘计算模型,将其与现有的云计算模式相结合,提升了数据处理的效率.香港科技大学的张军等人[20]针对移动边缘计算中终端电量不足的问题,提出了基于李雅普诺夫优化的动态任务卸载策略,综合考虑应用的卸载策略、移动终端执行的时钟频率和云端执行的传输功耗,为应用做出最优的卸载决策.西安电子科技大学的汪彦婷等人[21]针对移动边缘计算中的部分任务卸载问题,分别从单个边缘服务器场景和多个边缘服务器场景来考虑任务的卸载问题,并采用动态电压频率调节技术来优化移动终端能耗,提出的卸载算法降低了应用延迟和终端能耗.

然而,上面的研究工作都是考虑单个移动终端或边缘服务器资源无限的场景,这在实际应用中存在一定的局限性.香港理工大学的曹建农等人[22]针对云端资源受限的多终端任务卸载问题,提出了一种卸载策略SearchAdjust.该策略先假设云端资源无限然后再作调整,有效提高了应用性能和用户体验.西南大学的郭松涛等人[23]针对时间约束下的移动终端能耗最优问题,提出了一种节能的动态卸载和资源调度策略 eDors.该策略由任务卸载、CPU 时钟频率控制和传输功率分配这 3 个子算法组成,有效降低了终端能耗.关于多信道干扰的多终端任务卸载策略的问题,中山大学的陈旭等人[24]采用博弈论的方法为所有应用做出近似最优的卸载决策,从而在满足应用完成时间约束时,达到移动终端的能耗最优.

本文针对边缘服务器资源受限条件下多终端的串行任务卸载问题,考虑多个用户请求之间的相互影响,研究如何高效分配边缘服务器的资源,为所有应用做出近似最优的卸载决策,使得所有应用的平均完成时间和移动终端的平均能耗达到近似最优.

2 任务卸载场景及介绍

图 1 呈现了多终端任务卸载的场景,该场景主要包括 3 个部分:移动终端、移动网络和边缘服务器.本文的研究工作主要集中在移动终端和边缘服务器端.多终端的任务卸载过程如下:对于移动终端,产生并运行移动应用程序,通过移动网络向边缘服务器提交卸载请求;对于边缘服务器端,接收来自移动端的任务卸载请求,分配合适的资源处理应用请求,处理完成后通过移动网络将最终的结果返回到移动终端.

接下来主要介绍在任务卸载过程中涉及到的相关概念的定义.

(1) 移动应用程序

移动应用程序是移动终端运行的任务,此类任务一般分为独立任务和依赖任务两种:独立任务表示应用程

序的结构是不可分割的,只由单个任务构成,如 N 皇后问题、矩阵运算问题等;依赖任务表示应用程序可自动划分为多个具有相互依赖关系的子任务,子任务之间存在数据交互,如人脸识别应用、QR 二维码应用等.本文以依赖任务中的串行移动应用程序为研究对象.假设一个任务可划分为 n 个子任务,即 $Task=\{1,2,\ldots,n\}$.第 k 个子任务能够开始执行,除了要分配有足够的计算、存储和网络资源外,还需其前驱组件 $k-1$ 已执行完成.串行任务模型已被许多研究者所采用[2,17,22].具体介绍见第 3.1 节.

(2) 移动终端

移动终端是产生应用程序的环境,本文假设每个移动终端都只运行一个相同的应用程序.用 N 表示提交卸载请求的移动终端数量,可以表示为 $MDs=\{md_1,md_2,\ldots,md_N\}$,而每个移动终端可以表示成一个八元组:

$$md_i=\{i,capacity_i^m,power_i,P_{i,comp}^{cpu},P_{i,idle}^{cpu},P_{i,send},P_{i,recv},P_{i,idle}^{net}\},$$

其中,i 表示移动终端的编号,$i\in[1,N]$; $capacity_i^m$ 表示移动终端的计算能力(每秒执行的指令数,单位为 MIPS); $power_i$ 表示移动终端 i 当前的可用电量; $P_{i,comp}^{cpu}$和$P_{i,idle}^{cpu}$ 分别表示移动终端的 CPU 在工作状态和空闲状态下的功率;$P_{i,send}$,$P_{i,recv}$ 和 $P_{i,idle}^{net}$ 分别表示移动终端的网络接口在发送数据、接收数据和空闲状态下的功率.

(3) 边缘服务器

边缘服务器部署在靠近移动终端的网络边缘,为任务卸载提供计算、网络和存储等服务.本文只考虑部署在单个固定地理位置的边缘服务器.选取 M 个配置相同的虚拟机表示边缘服务器资源的集合,即 $VMs=\{VM_1,VM_2,\ldots,VM_M\}$,每个虚拟机可以表示为一个四元组:$VM_i=\{i,capacity_i,B_i,wait_i\}$,其中,$i$ 表示虚拟机的编号,$i\in[1,M]$; $capacity_i$ 为虚拟机 i 的计算能力,表示每秒执行的指令数;B_i 表示当前时刻移动终端与边缘服务器通信的网络带宽;$wait_i$ 表示组件卸载到虚拟机 i 上需要等待的时间.假设所有虚拟机的处理能力和传输能力均一致,并不会随着任务量的上升而改变.

(4) 移动网络

移动网络表示移动终端和边缘服务器之间数据的通信方式,为组件的卸载提供了可能.移动终端可以通过蜂窝网络(如 4G,5G[25])或者无线接入点(Wi-Fi)方式,将应用程序的部分组件卸载到边缘服务器执行.

(5) 卸载策略

卸载策略表示应用程序的卸载结果,对于每个终端运行的应用程序,其卸载策略可以表示为一个 n 维向量:$S_i=\{x_{i,1},x_{i,2},\ldots,x_{i,n}\}$,其中,$n$ 为应用 i 包含的组件个数.向量中的每个值 $x_{i,j}$ 代表应用 i 的组件 j 是在本地执行还是边缘服务器执行,其取值为 0 或 1:$x_{i,j}=0$ 表示应用 i 的组件 j 在本地执行,$x_{i,j}=1$ 表示组件卸载到边缘服务器执行.

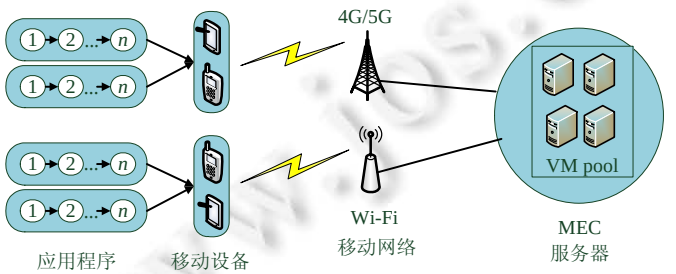


Fig.1 An illustration of task offloading with multiple mobile terminals

图 1 多终端任务卸载场景

3 系统模型及问题定义

3.1 任务模型

本文的研究对象是由若干个串行组件组成的移动应用程序.如图 2 所示,该应用由 n 个组件构成,其中,组件 0 和组件 $n+1$ 为虚拟组件,表示数据输入和结果输出的组件,且固定在本地执行.本文采用线性链表 $L=\{V,E\}$来表

示组件之间的依赖关系,每个节点 $i\in V$ 示移动应用程序的一个组件,每条有向边 $e(i,j)\in E$ 表示组件 i 和组件 j 之间的依赖关系.可以看出,组件 j 只有等待其前驱组件 i 执行完成后才能开始执行.

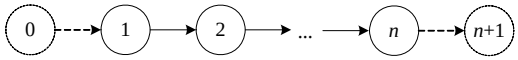


Fig.2 An illustration of serial task model for mobile applications

图 2 移动应用的串行任务模型

串行应用程序在实际生活中存在着广泛的应用,如人脸识别应用[2]、视频字符识别和国际象棋等.人脸识别应用包括图片输入、人脸检测、图像预处理、脸部特征提取、人脸匹配和结果输出等 6 个顺序执行组件,前一个组件计算的输出结果作为后一个组件的输入数据,当所有组件执行完成并输出最终结果,表示应用执行完毕.视频字符识别应用包括视频输入、关键帧提取、文本区域定位、字符识别、OCR 识别和文本输出等 6 个组件.人机对战中的中国象棋或国际象棋,机器每走一步,其后台的计算都可以划分为几个串行的任务:首先计算每颗棋子能够移动的位置,然后计算每个棋子的最优移动位置,最后计算最优的移动棋子[17].

3.2 完成时间模型

完成时间[26]是评价应用性能的一个重要指标,表示应用程序从数据输入到结果输出整个过程所需的时间.接下来,分别从应用组件在本地执行和边缘服务器执行来讨论组件的执行时间.

3.2.1 本地执行

本地执行表示组件(i,j)在移动终端上执行,$ST_{i,j}$,$FT_{i,j}$ 分别表示组件(i,j)的开始执行时间和结束时间,RT_i 记录移动终端 i 可以开始执行任务的时间.由于组件(i,j)必须等待其前驱组件$(i,j-1)$执行完成后才能开始执行,则组件(i,j)在本地的开始执行时间表示为

$$ST_{i,j}=\begin{cases}0, & pred(i,j)=\varnothing\\ \max\{TR_i, FT_{i,j-1}+T_{i,j-1,j}\}, & \text{others}\end{cases} \tag{1}$$

其中,$pred(i,j)$表示组件(i,j)的前驱组件的集合;$T_{i,j-1,j}$ 表示组件$(i,j-1)$和(i,j)之间的数据传输时间,计算公式如下:

$$T_{i,j-1,j}=\begin{cases}0, & x_{i,j-1}=x_{i,j}\\ \dfrac{data_{i,j-1,j}}{B_i}, & \text{others}\end{cases} \tag{2}$$

其中,$data_{i,j-1,j}$ 表示组件$(i,j-1)$和组件(i,j)之间数据传输的大小.

用 $t_{i,j}^m$ 表示组件(i,j)在本地执行的时间,因此,组件(i,j)的完成时间为

$$FT_{i,j}=ST_{i,j}+t_{i,j}^m \tag{3}$$

3.2.2 边缘服务器执行

边缘服务器执行表示组件通过移动网络卸载到边缘服务器上执行.对于组件(i,j),如果前驱组件没有在边缘服务器上执行,由于应用之间对边缘服务器资源的竞争使用,则需要考虑等待时间;反之,则不需要考虑等待时间.因此,组件(i,j)在边缘服务器上的开始执行时间可以表示为

$$ST_{i,j}=\begin{cases}FT_{i,j-1}+T_{i,j-1,j}, & x_{i,j-1}=1\\ FT_{i,j-1}+T_{i,j-1,j}+\min\limits_{1\leqslant v\leqslant M} wait_{i,v}, & \text{others}\end{cases} \tag{4}$$

其中,$wait_{i,v}$ 表示应用 i 在虚拟机 v 中的等待时间,其等待时间的长短主要取决于虚拟机 v 中排队在应用 i 之前的应用的服务时间之和.这里用 $service_{k,v}$ 表示应用 k 在虚拟机 v 上的服务时间,每个应用维护一个链表 $List_k$,表示应用 k 卸载到边缘服务器上的组件的集合.则应用 i 的等待时间可以通过如下公式得到:

$$wait_{i,v}=\sum service_{k,v}=\sum(FT_{List_{k,rear}}-ST_{List_{k,head}}) \tag{5}$$

用 $t_{i,j}^c$ 表示组件(i,j)在边缘服务器上的执行时间,此时,组件(i,j)对应的完成时间:

$$FT_{i,j} = ST_{i,j} + t_{i,j}^{c} \tag{6}$$

因此,应用 i 的完成时间:

$$MakeSpan_i = FT_{i,exit} - ST_{i,entry} \tag{7}$$

其中,$FT_{i,exit}$ 表示应用程序出口组件的结束时间,$ST_{i,entry}$ 表示应用程序入口组件的开始执行时间.

3.3 能耗模型

移动终端包括屏幕、CPU、蓝牙、GPS、ROM、RAM、网络接口等[27]多个部件,本文主要考虑移动终端 CPU 和网络接口产生的能耗.

3.3.1 计算能耗

移动终端的 CPU 主要包括运行和空闲两种状态.如果 CPU 有任务执行,就会一直处于运行状态,没有任务处理时,CPU 则会进入空闲状态.CPU 处于运行状态的能耗计算如下:

$$E_{i,comp}^{cpu} = P_{i,comp}^{cpu} \cdot T_{i,comp}^{cpu} \tag{8}$$

其中,$T_{i,comp}^{cpu}$ 为 CPU 处于运行状态的时间:

$$T_{i,comp}^{cpu} = \sum_{j \in V_i} t_{i,j}^{m} \cdot (1 - x_{i,j}) \tag{9}$$

CPU 处于空闲状态的时间:

$$T_{i,idle}^{cpu} = MakeSpan_i - T_{i,comp}^{cpu} \tag{10}$$

则 CPU 的空闲能耗:

$$E_{i,idle}^{cpu} = P_{i,idle}^{cpu} \cdot T_{i,idle}^{cpu} \tag{11}$$

移动终端 i 总的计算能耗:

$$E_i^{cpu} = E_{i,comp}^{cpu} + E_{i,idle}^{cpu} \tag{12}$$

3.3.2 传输能耗

传输能耗主要是移动终端和边缘服务器之间进行数据传输产生的,包括数据发送能耗、数据接收能耗和网络接口空闲能耗.对于一个应用而言,当存在两个满足前驱后继关系的相邻组件都在移动端或边缘服务器上执行时,传输能耗为零;只有当两个组件在不同的位置执行时,才存在传输能耗.移动终端发送数据到边缘服务器产生数据发送能耗,接收边缘服务器返回的数据产生数据接收能耗.则对应的能耗分别表示为

$$E_{i,send}^{net} = P_{i,send}^{net} \cdot T_{i,send}^{net}, E_{i,recv}^{net} = P_{i,recv}^{net} \cdot T_{i,recv}^{net} \tag{13}$$

其中,$T_{i,send}^{net}$ 和 $T_{i,recv}^{net}$ 分别表示移动终端的网络接口处于发送数据和接收数据状态的时间.因此,网络接口处于空闲状态的时间为

$$T_{i,idle}^{net} = MakeSpan_i - T_{i,send}^{net} - T_{i,recv}^{net} \tag{14}$$

对应的空闲能耗:

$$E_{i,idle}^{net} = P_{i,idle}^{net} \cdot T_{i,idle}^{net} \tag{15}$$

移动终端 i 总的传输能耗:

$$E_i^{net} = E_{i,send}^{net} + E_{i,recv}^{net} + E_{i,idle}^{net} \tag{16}$$

因此,单个移动终端的总能耗:

$$E_i = E_i^{cpu} + E_i^{net} \tag{17}$$

3.4 问题定义

在第 3.2 节和第 3.3 节中分别建立了单个应用的完成时间模型和单个移动终端的能耗模型,$MskeSpan_i$ 表示应用 i 的完成时间,E_i 表示移动终端 i 产生的能耗.上述时间和能耗模型都受卸载决策 S 的影响,不能通过独立计算同时达到最小值,其中,$S=\{S_1,S_2,\ldots,S_N\}$.本文针对边缘服务器资源受限条件下多终端的串行任务卸载问题,旨

在提高串行应用性能的同时降低移动终端能耗,采用线性加权的方法来规划联合目标函数.因此,原问题可以定义为

$$F1:\min_{S}\frac{1}{N}\sum_{i=1}^{N}(\lambda_t\cdot MakeSpan_i+\lambda_e\cdot E_i) \tag{18}$$

满足以下约束条件:

(a) $ST_{i,j}\geqslant ST_{i,j-1}+(1-x_{i,j-1})\cdot t^m_{i,j-1}+x_{i,j-1}\cdot t^c_{i,j-1}+T_{i,j-1,j},\forall i\in[1,N],\forall j\in[1,n]$;

(b) $ST_{k,j,v}\geqslant FT_{i,j,v},\forall k,i\in[1,N],\forall j\in[1,n],\forall v\in[1,M]$;

(c) $\sum_{i=1}^{N}B_i\leqslant B,\forall i\in[1,N]$;

(d) $x_{i,j}\in\{0,1\},\forall i\in[1,N],\forall j\in[1,n]$;

(e) $x_{i,entry}=0,x_{i,exit}=0,\forall i\in[1,N]$.

上述λ_t和λ_e分别代表应用完成时间和终端能耗所占的权重,有$\lambda_t,\lambda_e\in(0,1)$,且满足$\lambda_t+\lambda_e=1$.其中:限制条件(a)保证了组件$(i,j)$只有等待其前驱组件$(i,j-1)$执行完成才能开始执行;限制条件(b)表示边缘服务器中每个虚拟机在同一时间只能处理一个任务,即应用 k 和应用 i 都有任务在虚拟机 v 上执行,应用 k 表示应用 i 后面的应用,则应用 k 的任务只有等待应用 i 的任务执行完成后才能开始执行;限制条件(c)表示所有任务所分配的带宽量应满足边缘服务器提供的网络带宽的约束;限制条件(d)表示应用组件的执行位置只能是 0 或 1.根据应用系统模型,应用的输入数据由移动终端产生,输出结果将返回到移动终端,限制条件(e)表明应用的入口组件和出口组件只能在本地执行.

命题 1. $F1$ 问题属于 NP-hard.

证明:考虑 $F1$ 问题的一种特殊情况,参数$\lambda_t=1,\lambda_e=0$,并忽略组件之间传输时间.此时,应用 i 的任务完成时间仅由所有组件的执行时间构成.计算如下:

$$Time_i=\sum_{j=1}^{n}(x_{i,j}\cdot t^c_{i,j}+(1-x_{i,j})\cdot t^m_{i,j}) \tag{19}$$

其中,$t^c_{i,j}$和$t^m_{i,j}$分别表示组件(i,j)在移动设备和服务器上面的执行时间.因此,问题 $F1$ 等价于:

$$\min_{S_i}\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n}(x_{i,j}\cdot t^c_{i,j}+(1-x_{i,j})\cdot t^m_{i,j}) \tag{20}$$

满足上述约束条件(a)~约束条件(e).

可以看出:该特殊情况为组件在本地执行或边缘服务器执行的完成时间最小化问题,等价于 0-1 背包问题.我们知道,0-1 背包问题属于 NP-hard 问题[28].因此,$F1$ 问题的特殊情况也是 NP-hard 问题,故 $F1$ 问题也属于 NP-hard[29].

4 算法描述

本文研究的任务卸载问题是典型的多目标组合优化问题[30],属于 NP-hard 问题,可以采用启发式算法求解该类问题.目前存在多种启发式算法,如遗传算法(GA)[31]、粒子群算法(PSO)[32]、蚁群算法(ACO)[33]和化学反应优化算法(CRO)[34]等.相比于遗传算法,化学反应优化算法能避免过早地陷入局部最优,获取全局近似最优解,同时以更快的速度收敛于最优解[35].化学反应优化算法[34]最早是在 2010 年由 Lam 和 Li 提出来的,灵感来源于大自然中的化学反应.根据实际应用表明:化学反应算法能够有效地解决多目标优化问题,并且在二次分配问题、网格任务调度问题、人工神经网络训练、资源受限项目调度问题等方面得到广泛的应用.针对边缘服务器资源受限的多终端任务卸载问题,本文采用化学反应优化算法求解.

4.1 算法框架

化学反应优化算法中,分子具有 3 个必要属性:分子结构(structure,简称 S)、分子势能(potential energy,简称 PE)和分子动能(kinetic energy,简称 KE).分子结构 S 表示任务卸载策略的一个解空间;分子势能 PE 表示分子结

构的稳定性,定义为目标函数的值;分子动能 *KE* 使得分子势能趋向更高的状态,避免目标函数值过早地陷入局部最优解而继续寻找全局最优解.第 4.2 节~第 4.4 节主要从问题编码、适应度函数和操作算子设计这 3 个角度介绍化学反应优化算法.问题编码定义了卸载策略与分子结构之间的关系;适应度函数是评估分子好坏的标准,分子的适应度值越小,代表所求问题的解越优,因此,设计一种好的适应度函数有利于获得最优解;操作算子是算法搜索解空间的关键步骤,不同的操作算子能够改变分子结构,避免算法过早陷入局部最有解,同时搜索全局最优解.第 4.5 节介绍具体的算法设计,第 4.6 节和第 4.7 节分别对算法的复杂度和收敛性进行分析.

4.2 问题编码

化学反应优化算法是一种通过模拟化学反应过程搜索最优解的方法,问题解空间用分子集合表示,分子是由若干个原子编码组成.本文研究的移动应用程序是由串行执行的组件构成,每个组件执行位置分为移动终端执行和边缘服务器执行,分别用 0 和 1 表示.移动应用程序的卸载决策表示为分子,组件的执行位置用原子表示.

图 3 是移动应用程序及其对应的分子结构,该应用程序包含 5 个组件,分子结构表示应用程序的卸载策略,分子中每个原子的值代表组件的卸载位置.图中是一种可能的分子结构 S={1,1,0,1,0},表示组件 3、组件 5 在移动终端上执行,组件 1、组件 2、组件 4 卸载到边缘服务器执行.

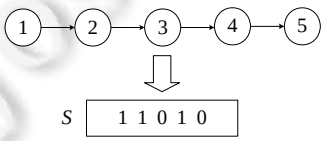


Fig.3 Mobile application and corresponding molecule

图 3 移动应用程序及对应的分子结构

4.3 适应度函数

本文评价卸载策略性能的指标是平衡所有应用的平均完成时间和移动终端的平均能耗.对于一种卸载策略 S,对应的应用完成时间用 T 表示,移动终端能耗用 E 表示,将两个不同量纲的目标进行归一化处理,并采用线性加权方式将双目标转化为单目标,则适应度函数[36]可以表示为如下形式:

$$Fitness(S)=\alpha\cdot\frac{T-T_{\min}}{T_{\max}-T_{\min}}+\beta\cdot\frac{E-E_{\min}}{E_{\max}-E_{\min}} \tag{21}$$

其中,$T_{\max}$ 和 $T_{\min}$ 分别代表分子种群中所有分子对应的应用完成时间的最大值和最小值,$E_{\max}$ 和 $E_{\min}$ 分别代表分子种群中所有分子对应的移动终端能耗的最大值和最小值.α和β分别代表应用完成时间和移动终端能耗的比例所占权重,其取值主要由移动终端的可用电量来决定,$\alpha,\beta\in(0,1)$,且满足$\alpha+\beta=1$.当移动终端电量充足时,完成时间为主要优化目标,参数设置条件为$\alpha>\beta$;当电量不足时,主要优化移动终端能耗,设置$\alpha<\beta$.然而,在实际场景中,完成时间和能耗的权重的合理分配可根据多属性决策理论[24,34]来确定.

4.4 操作算子设计

本节介绍化学反应优化算法的 4 种基本操作算子,包括单分子碰撞、单分子分解、分子间碰撞和分子合成.问题编码中介绍过每个分子结构表示一种卸载策略,不同的操作算子对分子结构会产生变化,应用程序的卸载策略也会随之改变.单分子碰撞和分子间碰撞对原分子结构改变较小,旨在原卸载策略的邻域空间搜索更优的卸载策略;单分子分解和分子合成提供搜索更大解空间的机会,避免卸载策略过早地陷入局部最优解.下面详细介绍 4 种基本操作算子的具体设计.

4.4.1 单分子碰撞

单分子碰撞是指单个分子与容器壁发生碰撞并反弹的过程.对于分子 S,发生单分子碰撞后产生新的分子 S',新分子 S'与原分子 S 在分子结构上相差不大,表示碰撞过程是在分子 S 的邻域空间搜索最优解.

具体设计如图 4 所示:从原分子 S 中随机选择一个原子,改变其执行位置,剩余部分保留不变.对应到卸载策

略中,原先的卸载策略经过单分子碰撞后,随机改变应用某个组件的执行位置,产生新的卸载策略.

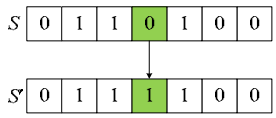


Fig.4 On-wall ineffective collision

图 4 单分子碰撞

4.4.2 单分子分解

单分子分解是指单个分子与容器壁发生碰撞后分解为两个或多个分子的过程.本文假设分解后产生两个分子.对于分子 *S*,经过单分子分解后产生新的分子 *S*1 和 *S*2,新分子 *S*1 和 *S*2 与原分子 *S* 在分子结构上差别较大,表示分解过程是探索更大的解空间,避免结果过早地收敛于局部最优解.

具体设计如图 5 所示:在原分子中随机选择一个分解点 *point*,新分子 *S*1 保留原分子 *S* 的[*start*,*point*]部分的原子,新分子 *S*2 保留原分子 *S* 的[*point*,*end*]部分的原子,*S*1 和 *S*2 剩余的原子位随机生成.对应到卸载策略中,原卸载策略经过单分子分解,随机选择一个组件作为分解点,将分解点的前部分组件和后部分组件分别保留到两个新分子的对应位置,剩余组件的执行位置随机产生,从而得到新的卸载策略.

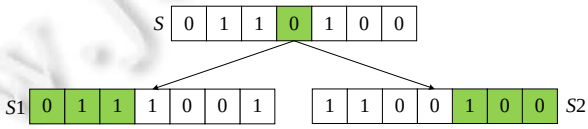


Fig.5 Decomposition

图 5 单分子分解

4.4.3 分子间碰撞

分子间碰撞是指两个分子相互碰撞并反弹的过程.对于分子 *S*1 和 *S*2,发生分子间碰撞后产生新分子 *S*1′和 *S*2′,新分子 *S*1′和 *S*2′与原分子在分子结构上比较相似,表示碰撞过程是在分子 *S*1 和 *S*2 的邻域空间搜索最优解.

具体设计如图 6 所示:从原分子中选择两个碰撞点 *point*1 和 *point*2,交换原分子 *S*1 和 *S*2 对应碰撞点的原子的值,形成新的分子.对应到卸载策略中,原卸载策略经过分子间碰撞,随机交换两个卸载策略中对应两个组件的执行位置,得到两个新的卸载策略.

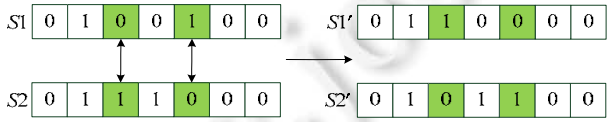


Fig.6 Inter-Molecular ineffective collision

图 6 分子间碰撞

4.4.4 分子合成

分子合成是指两个或多个分子相互碰撞并合成一个分子的过程.本文假设是两个分子进行分子合成操作.对于分子 *S*1 和 *S*2,经过分子合成后产生新的分子 *S*,新分子 *S* 和原分子在分子结构上相差较大,表示合成使得分子种群呈现多样化,扩大解的搜索空间.

具体设计如图 7 所示:随机生成一个合成点 *point*,新分子继承原分子 *S*1 的[*start*,*point*]部分的原子和原分子 *S*2 的[*point*,*end*]部分的原子.对应到卸载策略中,原卸载策略经过分子合成后,随机选择一个组件作为合成点,将第 1 种卸载策略的合成点的前部分组件和第 2 种卸载策略对应合成点的后部分组件保留,合成后得到新的卸载策略.

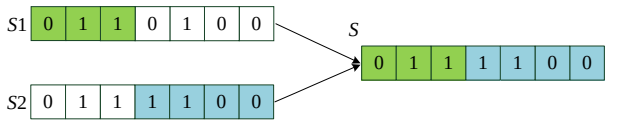


Fig.7 Synthesis

图 7 分子合成

4.5 算法设计

本文提出了一种面向多用户的串行任务动态卸载策略 MSTDOS.针对边缘服务器资源受限的多终端任务卸载问题,MSTDOS 算法为所有应用做出近似最优的卸载决策,使得在减少应用完成时间的同时降低移动终端能耗.

算法 1 中,首先,MSTDOS 对边缘服务器上所有虚拟机的状态进行初始化(line 1),保证所有虚拟机可以最早开始执行任务的时间为零,其次,进入迭代过程,为所有应用做出近似最优的卸载决策.在每次迭代过程中,遍历所有的虚拟机,选择可以最早开始执行任务的虚拟机 v,记录该虚拟机的最早开始时间为 $RT_{\min}$(line 3~line 7),此时,将虚拟机 v 作为应用 i 卸载组件到边缘服务器的候选虚拟机.确定应用 i 卸载的候选虚拟机后,采用化学反应优化算法 $CRO(i,RT_{\min})$为应用 i 做出任务卸载决策(CRO 算法的具体过程在算法 2 中介绍)(line 7),并返回应用 i 的近似最优卸载策略,将其添加到所有应用的策略集 *StrategyList* 中(line 8),并更新虚拟机的最早开始执行时间(line 9).最后,迭代过程结束,得到所有应用的近似最优卸载策略.

算法 1. 多用户串行任务动态卸载策略 MSTDOS.

输入:所有移动终端的卸载请求、移动应用程序;

输出:所有应用的近似最优任务卸载策略.

1. $RT_{1...v}\leftarrow 0$ //初始化所有虚拟机的最早开始执行时间
2. **for each** application $i\in ApplicationList$
3. $RT_{\min}\leftarrow MAXVALUE$
4. **for each** $VM_v\in VMList$ //选择最早开始执行任务的虚拟机
5. **if** $RT_v<RT_{\min}$ **then**
6. $RT_{\min}\leftarrow RT_v$
7. $strategy[i]\leftarrow CRO(i,RT_{\min})$ //采用 CRO 算法为应用 i 做出任务卸载决策
8. $StrategyList\leftarrow strategy[i]$ //将卸载策略 $strategy[i]$添加到 *StrategyList*
9. $updateVmRT(v)$ //更新虚拟机的最早开始执行时间
10. **end for**
11. **return** *StrategyList*

算法 2 详细介绍了通过化学反应优化算法 CRO 为应用做出任务卸载决策的过程.CRO 算法包括 3 个阶段:初始化、迭代和结果输出.在初始化阶段(line 1~line 11)进行相关参数的定义(line 1),并随机生成包含 *PopSize* 个分子的分子种群(line 2~line 11).在迭代阶段(line 12~line 31)进行 *MaxIter* 次迭代过程,每一次迭代过程中,从 4 种基本操作中选择一种操作发生反应.将参数 *MoleColl* 定义为分子间发生反应的概率,生成一个随机数 $t\in(0,1)$,如果 $t>MoleColl$,则发生单分子反应(line 15~line 21);否则发生分子间反应(line 22~line 28).单分子反应时,随机选择一个分子,如果分子满足分解条件,则发生单分子分解操作;否则发生单分子碰撞操作,单分子反应结束时更新策略集 *Strategies*.分子间反应时,随机选择一对分子,判断分子对是否同时满足分子合成操作:满足则发生单分子合成操作,否则发生单分子碰撞操作.在结果输出阶段(line 32, line 33),从分子种群中选择具有最小势能的分子,作为应用 i 的近似最优卸载策略.

算法 2. 化学反应优化算法 $CRO(i,RT_{\min})$.

输入:移动终端 i 的请求、移动应用程序;

输出:应用 i 的近似最优任务卸载策略.

初始化 *PopSize*,*MaxIter*,*Molecule*,*KELossRate*,*InitialKE*,*Strategies*
j←1
while j≤*PopSize*
    Rand(*S*)　　//随机生成一个卸载策略 *S*
    MakeSpan(*S*)←*compMakeSpan*(*S*,$RT_{\min}$)　　//根据公式(7)计算完成时间
    Energy[*S*]←*computeEnergy*(*S*)　　//根据公式(16)计算能耗
    KE[*S*]←*InitialKE*(*S*)　　//初始化分子势能
    PE[*S*]←*computeFitness*(*S*)　　//根据适应度函数公式计算分子的适应度
    Strategies←*S*　　//将卸载策略 *S* 添加到策略集 *Strategies* 中
    j←j+1
end while
j←1
while j≤*MaxIter*
    t←*random*(0,1)
    if t>*MoleColl* **then**　　//发生单分子反应
        S←*Strategies*　　//从策略集 *Strategies* 中随机选择一个分子 *S*
        if *checkDecomp*(*S*) **then**
            status←*decompose*(*S*,*s*1,*s*2)　　//单分子分解
        else
            status←*ineff_coll_on_wall*(*S*,*s*)　　//单分子碰撞
        Strategies→*S*′　　//将新策略添加到 *Strategies*,替换旧策略 *S*
    else　　//发生分子间反应
        ⟨*S*1,*S*2⟩←*Strategies*　　//从策略集 *Strategies* 中随机选择一对⟨*S*1,*S*2⟩
        if *checkSynth*(*S*1) & *checkSynth*(*S*2) **then**
            status←*synthesis*(*S*1,*S*2,*s*)　　//分子合成
        else
            status←*inter_ineff_wall*(*S*1,*S*2,*s*1,*s*2)　　//分子间碰撞
        Strategies←⟨*S*1′,*S*2′⟩　　//将新策略添加到 *Strategies*,替换旧策略 *S*1 和 *S*2
    $MinPE_{strategy}$←min*PE*(*S*)　　//选择具有最小 *PE* 的策略
    j←j+1
end while
strategy[i]←*MinPE_strategy*
retrun *strategy*[i]

4.6 算法的时间复杂度分析

设应用的个数为 N,边缘服务器的虚拟机个数为 M,分子种群的规模为 *PopSize*,最大迭代次数为 *MaxIter*,应用的组件个数为 n.算法 1 中,MSTDOS 算法为每个应用做决策主要包括两个步骤:首先,从边缘服务器中选择合适的虚拟机,作为卸载应用的候选虚拟机,其时间复杂度为 $O(M)$;其次,采用 CRO 算法为当前应用做出具体的任务卸载决策,其中,初始化分子种群的时间复杂度为 $O(PopSize\times n)$,单分子碰撞操作的时间复杂度为 $O(1)$,单分子分解操作的时间复杂度为 $O(n)$,分子间碰撞的时间复杂度为 $O(1)$,分子合成的时间复杂度为 $O(n)$,基本操作迭代的次数为 Max*Iter*,因而,CRO 算法的时间复杂度为 $O(PopSize\times n+\text{Max}Iter\times n^2)$.

因此,算法 1 总的时间复杂度为 $O(N\times(M+PopSize\times n+\text{Max}Iter\times n^2))$.

4.7 算法的收敛性分析

MSTDOS 算法的收敛性主要受化学反应算法的基本操作影响.参考文献[37]中建立的有限吸收马尔可夫链模型,将每个分子结构表示成一种状态,用 X 表示所有状态空间的集合.对于一种状态,经过基本操作Λ后会改变其状态,令 $\Psi_\Lambda(x)$表示分子 x 经过操作Λ后可能出现的状态的集合,且 $\Psi_\Lambda(x)\subseteq X$,二元组$(X,(\Psi_\Lambda(x),x\in X))$能够抽象成一个有向无环图 $G(V,E)$,顶点集 V 用 X 表示,边集 E 则根据 $\Psi_\Lambda(x)=\{i|(x,i)\in E\}$得到.如图 8 所示为经过操作$\Lambda$后状态的转换过程,可以得出,状态 x_1 经过操作Λ可以转换成 x_2,x_3 和 x_4,即 $\Psi_\Lambda(x_1)=\{x_2,x_3,x_4\}$,以此类推,$\Psi_\Lambda(x_2)=\{x_2,x_5,x_6\}$,$\Psi_\Lambda(x_3)=\{x_1,x_2,x_6\}$,$\Psi_\Lambda(x_4)=\{x_3,x_5\}$,$\Psi_\Lambda(x_5)=\{x_2,x_6\}$,$\Psi_\Lambda(x_6)=\{x_2\}$.

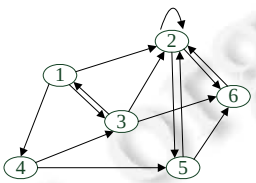


Fig.8 A state transition under operator Λ

图 8 操作Λ下的状态转换

定义 1(最佳可达性[37]). 设 $G(V,E)$为 CRO 算法的解图,若$\forall x\notin X-X_{opt}$,则经过一定的状态转换,至少存在一个最优解 $x_{opt}\in X_{opt}$,使得 $x=x_0\to x_1\to x_2\to\ldots\to x_l=x_{opt}$,其中,$l$ 为从 x 到 x_{opt} 状态转换的次数,则称 $G(V,E)$为最佳可达.

定理 1. CRO 收敛于全局最优解的必要条件是 $G(V,E)$最佳可达.

证明:CRO 中每种操作算子都存在一个解图,用 $G(V,E^{owi})$,$G(V,E^{dec})$,$G(V,E^{imi})$,$G(V,E^{syn})$分别表示单分子碰撞、单分子分解、分子间碰撞和分子合成的解图,$G(V,E)$表示 CRO 算法的解图,则满足 $E=E^{owi}\cup E^{dec}\cup E^{imi}\cup E^{syn}$.假设解图 $G(V,E)$不是最佳可达,其解图的任意非最优解都无法通过状态转化到达最优解,则存在解 $x'\in X-X_{opt}$.CRO 算法中初始种群产生的解都是非最优解 x'的概率为 $1/\|X\|^n$,$\|X\|$表示 X 的基,n 为初始种群大小.于是,从 x'转换到最优解的概率为

$$P\{x_{t+1}\in X_{opt}|x_t\notin X_{opt}\}=0,t\geq 0,$$

则不存在一个从非最优解到最优解的转换过程,因此,产生最优解的概率计算如下:

$$\lim_{t\to\infty}P\{x\in X_{opt}\}\leqslant 1-\frac{1}{\|X\|^n}\leqslant 1.$$

由此可得:当 $G(V,E)$不满足最佳可达的条件时,CRO 算法不能收敛于全局最优解.证毕. □

5 仿真实验与结果分析

5.1 仿真实验环境

(1) 移动终端

本文的移动终端选取 Oppo A37、乐视 3、Vivo Y67 和小米 6 这 4 款手机,其详细的参数配置见表 1,具体的功率值是通过移动终端的 power-profile.xml 文件获取得到.

Table 1 Mobile terminals configuration

表 1 移动终端配置

类型	状态	功率(mW)			
		Oppo A37	乐视 3	Vivo Y67	小米 6
CPU	活跃	130	204	86	290
	空闲	5	4	4.5	9
网络接口	活跃	80	100	65	110
	空闲	4.5	12	10	4

(2) 边缘服务器

采用由实验室 4 台服务器搭建的 Openstack 集群环境,Openstack 的版本为 Juno,4 台服务器的型号都是 Lenovo ThinkServer RD330,CPU 是两个核心数为四核的 Intel Xeon E5-2620,主频是 2.1GHz,内存大小是 32G,硬盘容量是 500G,操作系统是 Ubuntu 14.04 的 linux 系统.

(3) 网络环境

边缘服务器的内部节点之间是通过带宽为 56Gbps 的 InfiniBand 高速交换机进行连接,交换机的型号是 Mellanox SX602.边缘服务器与外部设备通过 TP-Link 的千兆以太网交换机连接,交换机型号为 TL-SG1024T.

(4) 移动应用程序

本文选用人脸识别应用[2]作为实验验证的应用程序.如图 9 所示,人脸识别应用主要包括 6 个顺序执行的组件,分别是图片输入、人脸检测、图像预处理、脸部特征提取、人脸匹配和结果输出.将每个组件用序号进行编号,其中,No.0 和 No.5 分别代表人脸识别应用程序的开始和结束组件,固定在移动终端执行.实验中,选取一张 240KB 大小的包含人脸图像的图片作为输入数据.

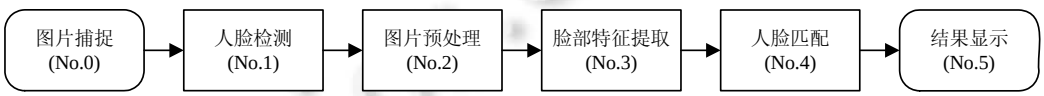


Fig.9 Face recognition applications

图 9 人脸识别应用

首先,分别在给定的移动终端和服务器上运行人脸识别应用,测出应用的每个组件的执行时间,表 2 给出了每个组件的平均执行时间.

Table 2 Average execution time of each component

表 2 应用程序组件的平均执行时间

时间(ms) 组件号 / 设备和虚拟机	No.1	No.2	No.3	No.4
Oppo A37	21	11	18	505
乐视 3	14	9	13	441
Vivo Y67	29	15	24	536
小米 6	12	8	11	393
虚拟机	8	4	6	230

然后,对组件之间传输的数据大小进行了测量,测量结果见表 3.

Table 3 Transmission data size among components

表 3 组件之间的传输数据大小

边	数据传输大小(kB)
No.0→No.1	240
No.1→No.2	240
No.2→No.3	100
No.3→No.4	25
No.4→No.5	25

5.2 实验结果分析

本节将提出的 MSTDOS 策略与已有的 AllInMobile 策略[24]、SearchAdjust 策略[22]和 MDSA 策略[38]进行比较.其中,AllInMobile 策略表示应用的所有组件都在本地执行,不受边缘服务器资源和网络带宽的影响.SearchAdjust 策略是一种多终端离线卸载策略,先假设边缘服务器资源无限,做出应用的卸载决策,再对不满足资源限制条件的组件进行延迟执行,保证所有任务都能够正常执行完成.MDAS 策略是一种多维搜索调整策略,在边缘服务器资源受限时,通过判断计算资源和网络资源是否发生冲突,来动态调整任务的卸载位置以达到降

低延迟的效果.接下来,文献[22,38]分别通过调整移动终端数量、虚拟机数量和网络带宽这 3 个参数对上述 4 个策略的性能进行对比分析,以及测试权重因子α和β对 MSTDOS 卸载策略性能的影响和测试 MEC 服务器性能瓶颈.

5.2.1 移动终端数量的影响

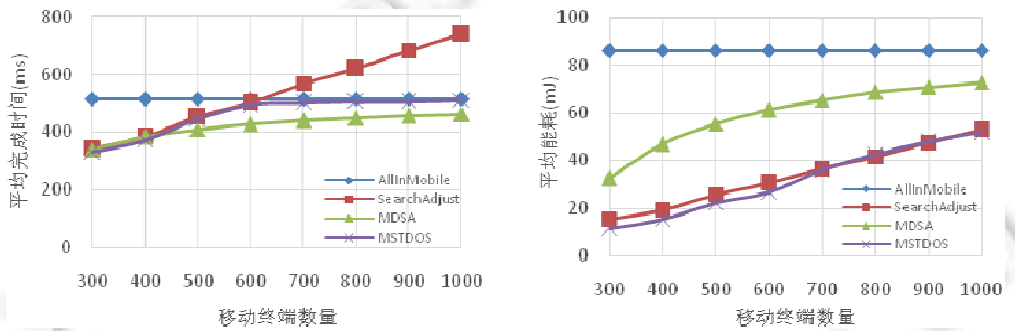
本组实验研究应用数量对不同卸载策略的影响,具体的参数配置见表 4.

Table 4 Environment configuration of experiment 1

表 4 实验 1 的环境配置

实验配置	参数
虚拟机数量	200
网络带宽	8Mbps
移动终端型号	Oppo A37、乐视 3、Vivo Y67、小米 6
移动设备可用电量	不充足

本实验中,边缘服务器有 200 个虚拟机,网络带宽取 8Mbps,移动终端选取上面提到的 4 款手机,主要研究移动终端数量不断增加时对卸载策略性能的影响.具体的实验结果如图 10 所示.



(a) Average completion time (b) Average energy consumption

Fig.10 Impact of mobile terminals' number on average completion time and average energy consumption

图 10 移动终端数量对平均完成时间和平均能耗的影响

从图 10(a)和图 10(b)可知:随着移动终端数量的增加,SearchAdjust 策略、MDSA 策略和 MSTDOS 策略的应用平均完成时间和终端平均能耗都会明显的增加;而 AllInMobile 策略所有组件都在本地执行,不受移动终端数量的影响,且其平均完成时间和平均能耗开销较大.其中,MSTDOS 策略与 MDSA 策略相比,两者平均完成时间较接近,有一点劣势;但在平均能耗上有较大的优势.主要是由于在移动终端可用电量不充足时,MSTDOS 策略以牺牲一部分完成时间的代价来获得更低的平均能耗,而 MDSA 策略只关注于减少平均完成时间,忽视了终端能耗.其中,MSTDOS 策略与 SearchAdjust 策略相比,当移动终端数量较小时,两者平均完成时间非常接近,在平均能耗上有点优势;当移动终端数量较大时,在平均完成时间有明显优势,而平均能耗很接近.因为当移动终端数较少时,边缘服务器资源充足,MSTDOS 策略和 SearchAdjust 策略都能取得较好的性能.但移动终端数量的增加,大量的移动终端会同时向边缘服务器提交卸载请求,MSTDOS 策略考虑到移动终端之间卸载策略的影响,能够动态地调整某些应用组件的执行位置,缩短了组件在边缘服务器上的等待时间;而 SearchAdjust 策略并不会做出类似 MSTDOS 策略的调整过程,对于卸载到边缘服务器的组件只考虑了延迟执行,这会增加额外的等待时间,导致应用性能的衰减.

5.2.2 边缘服务器虚拟机数量的影响

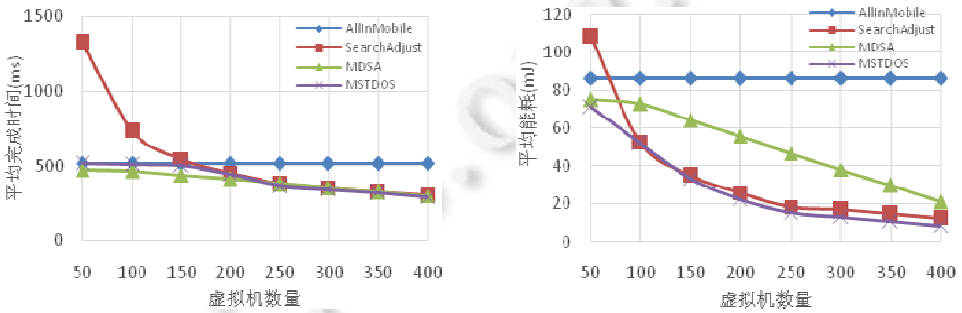
本组实验研究边缘服务器虚拟机个数对不同卸载策略的影响,具体参数配置见表 5.

Table 5 Environment configuration of experiment 2

表 5 实验 2 的环境配置

实验配置	参数
移动终端数量	500
网络带宽	8 Mbps
移动终端型号	Oppo A37、乐视 3、Vivo Y67、小米 6
移动设备可用电量	不充足

本实验中,移动终端数量为 500 个,网络带宽为 8Mbps,移动终端选取上面提到的 4 款手机,主要研究边缘服务器上虚拟机不断增加时对卸载策略性能的影响.具体的实验结果如图 11 所示.



(a) Average completion time (b) Average energy consumption

Fig.11 Impact of virtual machines' number on average completion time and average energy consumption

图 11 虚拟机数量对平均完成时间和平均能耗的影响

从图 11(a)和图 11(b)可知:随着边缘服务器虚拟机数量的增加,SearchAdjust 策略、MDSA 策略和 MSTDOS 策略的应用平均完成时间和终端平均能耗都会明显的降低.主要由于随着虚拟机数量增多,应用对虚拟机资源的竞争得以缓解,边缘服务器能提供足够的资源同时服务多个终端的请求.而 AllInMobile 策略所有组件都在本地执行,虚拟机数量的变化对其性能没有影响,且其平均完成时间和平均能耗开销较大.其中,MSTDOS 策略与 MDSA 策略相比,两者平均完成时间较接近;但在平均能耗上有较大的优势.主要是由于两者策略的专注点不同,MSTDOS 策略是折中平均完成时间和平均能耗,在移动终端可用电量不充足时,为了降低平均能耗而牺牲一部分完成时间;但 MDSA 策略主要针对减少平均完成时间,没有考虑如何降低终端能耗,从而取得了最优的平均完成时间,而导致平均完成能耗比较高.其中,MSTDOS 策略与 SearchAdjust 策略相比,当虚拟机数量较小时,在平均完成时间和平均能耗上都有优势;当虚拟机数量较大时,平均完成时间和平均能耗都非常接近.因为虚拟机数量较少时,用户数大于虚拟机数量,导致用户请求对虚拟机资源的竞争激烈.此时,MSTDOS 策略会采取动态调整机制为应用做出近似最优的卸载决策,尽可能地降低所有应用的平均完成时间和终端平均能耗,而 SearchAdjust 策略对于卸载到边缘服务器的请求,采用推迟执行的方法,应用之间的等待时间会大大增加,导致应用性能较差.但虚拟机数量较大时,边缘服务器资源充足,能够同时处理的用户请求量大.此时,MSTDOS 策略和 SearchAdjust 策略在平均完成时间和平均能耗上都能取得较好的性能.

5.2.3 网络带宽的影响

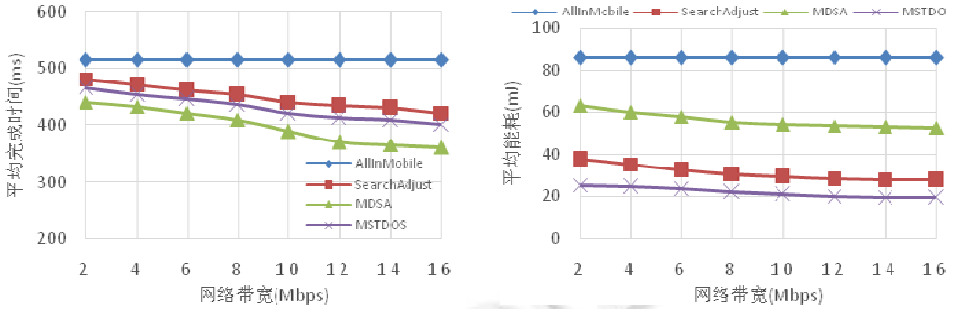
本组实验研究网络带宽对不同卸载策略的影响,具体参数配置见表 6.

Table 6 Environment configuration of experiment 3

表 6 实验 3 的环境配置

实验配置	参数
移动终端数量	500
虚拟机数量	200
移动终端型号	Oppo A37、乐视 3、Vivo Y67、小米 6
移动设备可用电量	不充足

本实验中,移动终端数量有 500 个,边缘服务器提供 200 个虚拟机,移动终端选取上面提到的 4 款手机,主要研究网络带宽不断增加时对卸载策略性能的影响.实验具体结果如图 12 所示.



(a) Average completion time (b) Average energy consumption

Fig.12 Impact of network bandwidth on average completion time and average energy consumption

图 12 网络带宽对平均完成时间和平均能耗的影响

从图 12(a)和图 12(b)可知:AllInMobile 策略由于所有的组件都在本地执行,网络带宽的变化对其没有影响,且其平均完成时间和平均能耗的开销较大.而 SearchAdjust 策略、MDSA 策略和 MSTDOS 策略的应用平均完成时间和终端平均能耗都与网络带宽呈负相关关系.随着网络带宽的增加,数据传输的延迟会减小,更多的组件会卸载到边缘服务器上执行,从而降低了应用平均完成时间和终端平均能耗.3 种策略的性能随着网络带宽的增加最初变化明显,而后趋近于平稳.主要是因为网络带宽决定着移动终端与边缘服务器之间数据的传输速率与应用传输时间成反比,随着网络带宽的增大,对应用传输时间的影响越来越小,从而对决策性能的提升逐渐减低.其中,MSTDOS 策略与 MDSA 策略相比,在平均完成时间上存在劣势;但在平均能耗上有较大的优势.主要是由于 MSTDOS 策略侧重于平均完成时间和平均能耗的均衡,而 MDSA 策略侧重于减少平均完成时间.在移动终端可用电量不充足时,前者为了得到较低的平均能耗以牺牲部分完成时间为代价,而后者只关注于减少平均完成时间,忽视了终端能耗.其中,MSTDOS 策略与 SearchAdjust 策略相比,在平均完成时间和平均能耗上都有一定的优势.主要是由于边缘服务器资源有限,用户请求会对资源产生竞争.此时,MSTDOS 策略考虑到多个终端之间会相互影响,能够根据边缘服务器的状态信息调整应用的卸载策略,为所有应用做出较优的卸载决策,而 SearchAdjust 策略只是延迟组件的执行,并没有采取相应措施进行调整.

5.2.4 权重因子α和β的影响

本组实验研究权重因子α和β对平均完成时间和平均能耗的影响.α和β的取值依赖于移动终端当前可用电量的多少,且满足$\alpha+\beta=1$.具体参数配置见表 7.

Table 7 Environment configuration of experiment 4

表 7 实验 4 的环境配置

实验配置	参数
移动终端数量	500
虚拟机数量	200
网络带宽	8Mbps
移动终端型号	Oppo A37、乐视 3、Vivo Y67、小米 6

本实验中,移动终端数量有 500 个,边缘服务器提供 200 个虚拟机,网络带宽取 8Mbps,移动终端选取上面提到的 4 款手机,主要研究移动终端可用电量充足和不足时对卸载策略性能的影响,实验具体结果如图 13 所示.

从图 13 中可以看出:随着权重因子α的减少,用户的平均完成时间是增加的,存在负相关关系;而终端平均能耗却相应减少,存在正相关关系.主要原因是:当α的值较大时,意味着移动设备有着充足的电量,用户更关心减少应用的完成时间;相反,当α的值较小时,β的值较大,意味着移动设备电量不充足.在这种情况下,执行能耗的权重

比平均完成时间的更高,才能有效地降低应用执行能耗.

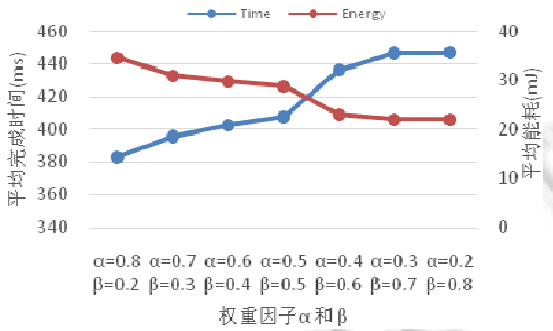


Fig.13 Impact of mobile terminal's available batter power on average completion time and average energy consumption

图 13 移动终端可用电量对平均完成时间和平均能耗的影响

5.2.5 MEC 服务器性能瓶颈测试

本组实验研究服务器的性能瓶颈,即对于一定数量的虚拟机能够支持的最大用户数.实验配置见表 8.

Table 8 Environment configuration of experiment 5

表 8 实验 5 的环境配置

实验配置	参数
网络带宽	8Mbps
移动终端型号	Oppo A37、乐视 3、Vivo Y67、小米 6
移动终端可用电量	不充足

为了验证 MEC 服务器的性能瓶颈,本文选择与 AllInMobile 策略进行实验比较.在 AllInMobile 策略中,人脸识别应用的所有组件都在移动终端上执行,通过实验测得应用平均执行时间是 515ms.人脸识别应用的执行时间一般分布在 370 ms~620ms[39],AllInMobile 策略的结果在合理范围内.最终实验测试结果如图 14 所示.

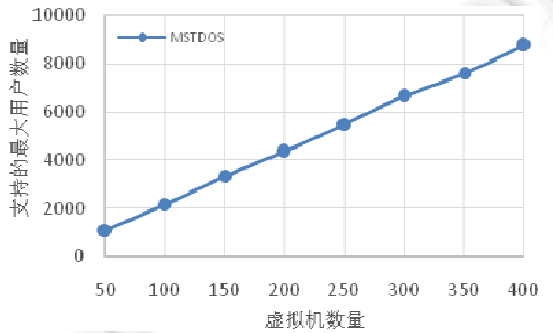


Fig.14 Performance bottlenecks of MEC server

图 14 MEC 服务器的性能瓶颈

如图 14 所示:当 MEC 服务器的虚拟机数量变化时,通过调节提交卸载请求的用户数目来测试 MEC 服务器的性能瓶颈.对于 MEC 服务器上指定数量虚拟机,不断地调节移动用户数目,当 MSTDOS 策略的执行性能低于 AllInMobile 策略的执行性能时停止调节,将此时的用户数量作为 MEC 服务器支撑的最大用户数.例如:在 MEC 服务器拥有 50 台虚拟机的情况下,当用户数量小于 1 100 时,MSTDOS 策略的执行性能高于 AllInMobile 策略的执行性能;而一旦用户数量超过 1 100 时,MSTDOS 策略的执行性能将会下降,同时低于 AllInMobile 策略的应

用性能.因此,我们能够得出一个结论:有 50 台虚拟机时,MEC 服务器最多能够支撑 1 100 个用户,则将此环境下 1 100 个用户作为 MEC 服务器的性能瓶颈.

6 总结与展望

本文针对边缘服务器资源受限的多终端任务卸载问题,提出了一种面向多用户的串行任务动态卸载策略 MSTDOS.该策略根据收集的移动终端信息和边缘服务器状态信息,动态地为所有应用做出近似最优的卸载决策.最后进行一系列实验,对 MSTDOS 的性能进行了评测.实验结果表明:本文提出的卸载策略在边缘服务器资源受限的多终端任务卸载场景下,在平均完成时间方面要优于 SearchAdjust 策略和 AllInMobile 策略,较劣于 MDSA 策略;而在平均能耗方面与 SearchAdjust 策略比较接近,比 AllInMobile 策略和 MDSA 策略有明显优势.

本文研究的任务卸载问题默认移动终端的位置是固定的,因此,将来的工作可以考虑用户移动性导致网络连接的变化,将任务卸载问题与移动性管理问题相结合来开展研究工作.此外,当前的研究工作主要考虑了边缘服务器只部署在单个固定的地理位置,其资源有限的特点会影响多终端任务卸载的应用性能,因而可以考虑部署在多个地理位置的边缘服务器场景下,研究多个终端的任务卸载问题.

References:

[1] Cisco. Cisco visual networking index: Forecast and methodology, 2016−2021. 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] Soyata T, Muraleedharan R, Funai C. Cloud-Vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In: Proc. of the IEEE Symp. on Computers and Communications. IEEE Computer Society, 2012. 59−66. [doi: 10.1109/ISCC.2012.6249269]

[3] Deffeyes S. Mobile augmented reality in the data center. IBM Corp, 2011,55(5):487−491. [doi: 10.1147/JRD.2011.2163278]

[4] Cui Y, Song J, Miao CC. Mobile cloud computing research progress and trends. Chinese Journal of Computers, 2017,40(2): 273−295 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2017.00273]

[5] Kumar K, Lu YH. Cloud computing for mobile users: Can offloading computation save energy. Computer, 2010,43(4):51−56. [doi: 10.1109/MC.2010.98]

[6] Zhang WL, Guo B, Shen Y. Computation offloading on intelligent mobile terminal. Chinese Journal of Computers, 2016,39(5): 1021−1038 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2016.01021]

[7] Satyanarayanan M, Bahl P, Davies N. The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing, 2009, 8(4):14−23. [doi: 10.1109/MPRV.2009.82]

[8] Shi WS, Sun H, Cao J. Edge computing: An emergying computing model for the internet of evergything era. Journal of Computer Research and Development, 2017,54(5):907−924 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2017.20160941]

[9] Deng XH, Li DS, Wu F. 2018 edge computing topic. Journal of Computer Research and Development, 2018,55(3):447−448 (in Chinese with English abstract).

[10] Zhao ZM, Liu F, Cai ZP, Xiao N. Edge computing: Platforms, applications and challenges. Journal of Computer Research and Development, 2018,55(2):327−337 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2018.20170228]

[11] Cuervo E, Balasubramanian A, Cho DK. MAUI: Making smartphones last longer with code offload. In: Proc. of the Int'l Conf. on Mobile Systems, Applications, and Services. ACM, 2010. 49−62. [doi: 10.1145/1814433.1814441]

[12] Kosta S, Aucinas A, Hui P. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: Proc. of the IEEE Int'l Conf. on Computer Communications. INFOCOM, 2012. 945−953.

[13] Chun BG, Ihm S, Maniatis P. CloneCloud: Elastic execution between mobile device and cloud. In: Proc. of the Conf. on Computer Systems. ACM, 2011. 301−314. [doi: 10.1145/1966445.1966473]

[14] Deng S, Huang L, Taheri J. Computation offloading for service workflow in mobile cloud computing. IEEE Trans. on Parallel and Distributed Systems, 2015,26(12):3317−3329. [doi: 10.1109/TPDS.2014.2381640]

[15] Yu M, Huang G, Wang X. JavaScript offloading for Web applications in mobile-cloud computing. In: Proc. of the IEEE Int'l Conf. on Mobile Services. 2015. 269−276. [doi: 10.1109/MobServ.2015.46]

[16] Zhang W, Wen Y, Wu DO. Collaborative task execution in mobile cloud computing under a stochastic wireless channel. IEEE Trans. on Wireless Communications, 2014,14(1):81−93. [doi: 10.1109/TWC.2014.2331051]

[17] Liu X, Li JB, Yang Z. A task collaborative policy in mobile cloud computing. Chinese Journal of Computers, 2017,40(2):364−377 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2017.00364]

[18] Li PW, Fu JM, Li SB. Elastic mobile cloud computing: State of the art and security analysis. Journal of Computer Research and Development, 2015,52(6):1362−1377 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2015.20140227]

[19] Zhou A, Wang S, Li J. Optimal mobile device selection for mobile cloud service providing. Journal of Supercomputing, 2016,72(8): 3222−3235.

[20] Mao Y, Zhang J, Letaief KB. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE Journal on Selected Areas in Communications, 2016,34(12):3590−3605. [doi: 10.1109/JSAC.2016.2611964]

[21] Wang Y, Sheng M, Wang X. Mobile-Edge computing: Partial computation offloading using dynamic voltage scaling. IEEE Trans. on Communications, 2016,64(10):4268−4282. [doi: 10.1109/TCOMM.2016.2599530]

[22] Yang L, Cao J, Cheng H. Multi-User computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. on Computers, 2015,64(8):2253−2266. [doi: 10.1109/TC.2014.2366735]

[23] Guo S, Xiao B, Yang Y. Energy-Efficient dynamic offloading and resource scheduling in mobile cloud computing. In: Proc. of the IEEE Int'l Conf. on Computer Communications. INFOCOM, 2016. 1–9. [doi: 10.1109/INFOCOM.2016.7524497]

[24] Chen X, Jiao L, Li W. Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE Trans. on Networking, 2016,24(5):2795−2808. [doi: 10.1109/TNET.2015.2487344]

[25] Hu YC, Patel M, Sabella D. Mobile edge computing—A key technology towards 5G. ETSI White Paper, 2015,11(11):1−16.

[26] Ra MR, Sheth A, Mummert L. Odessa: Enabling interactive perception applications on mobile devices. In: Proc. of the Int'l Conf. on Mobile Systems, Applications, and Services. ACM, 2011. 43−56. [doi: 10.1145/1999995.2000000]

[27] Perrucci GP, Fitzek FHP, Widmer J. Survey on energy consumption entities on the smartphone platform. In: Proc. of the Vehicular Technology Conf. IEEE, 2011. 1−6. [doi: 10.1109/VETECS.2011.5956528]

[28] Truong TK, Li K, Xu Y. Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem. Applied Soft Computing, 2013,13(4):1774−1780. [doi: 10.1016/j.asoc.2012.11.048]

[29] Garey MR, Johnson DS. "Strong" NP-completeness results: Motivation, examples, and implications. Journal of the ACM, 1978, 25(3):499−508. [doi: 10.1145/322077.322090]

[30] Hu HY, Liu RH, Hu H. Multi-Objective optimization for task scheduling in mobile cloud computing. Journal of Computer Research and Development, 2017,54(9):1909−1919 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2017.20160757]

[31] Holland JH. Adaptation in natural and artificial systems. Quarterly Review of Biology, 1975,6(2):126−137.

[32] Kennedy J, Eberhart, R. Particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Neural Networks. 2002,4(8):1942−1948. [doi: 10.1109/ICNN.1995.488968]

[33] Bonabeau E, Dorigo M, Theraulaz G. Inspiration for optimization from social insect behavior. Nature, 2000,406(6791):39−42.

[34] Lam AYS, Li VOK. Chemical-Reaction-Inspired metaheuristic for optimization. IEEE Trans. on Evolutionary Computation, 2010, 14(3):381−399. [doi: 10.1109/TEVC.2009.2033580]

[35] Pandharipande SL, Dixit AK. Comparative study of performance of chemical reaction optimization with genetic algorithm. Int'l Journal of Computer Applications, 2014,107(8):1−8.

[36] Altiparmak F, Gen M, Lin L. A genetic algorithm approach for multi-objective optimization of supply chain networks. Computers and Industrial Engineering, 2006,51(1):196−215.

[37] Lam AYS, Li VOK, Xu J. On the convergence of chemical reaction optimization for combinatorial optimization. IEEE Trans. on Evolutionary Computation, 2013,17(5):605−620.

[38] Yang L, Liu B, Cao J. Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. In: Proc. of the IEEE Int'l Conf. on Cloud Computing. 2017. 246−253. [doi: 10.1109/CLOUD.2017.39]

[39] Satyanarayanan M. The emergence of edge computing. Computer, 2017,50(1):30−39.

附中文参考文献:

[4] 崔勇,宋健,缪葱葱.移动云计算研究进展与趋势.计算机学报,2017,40(2):273−295. [doi: 10.11897/SP.J.1016.2017.00273]

[6] 张文丽,郭兵,沈艳.智能移动终端计算迁移研究.计算机学报,2016,39(5):1021−1038. [doi: 10.11897/SP.J.1016.2016.01021]

[8] 施巍松,孙辉,曹杰.边缘计算:万物互联时代新型计算模型.计算机研究与发展,2017,54(5):907−924. [doi: 10.7544/issn1000-1239.2017.20160941]

[9] 邓晓衡,李东升,吴帆.2018 边缘计算专题前言.计算机研究与发展,2018,55(3):447−448.

[10] 赵梓铭,刘芳,蔡志平,肖侬.边缘计算:平台、应用与挑战.计算机研究与发展,2018,55(2):327−337. [doi: 10.7544/issn1000-1239.2018.20170228]

[17] 柳兴,李建彬,杨震.移动云计算中的一种任务联合执行策略.计算机学报,2017,40(2):364−377. [doi: 10.11897/SP.J.1016.2017.00364]

[18] 李鹏伟,傅建明,李拴保.弹性移动云计算的研究进展与安全性分析.计算机研究与发展,2015,52(6):1362−1377. [doi: 10.7544/issn1000-1239.2015.20140227]

[30] 胡海洋,刘润华,胡华.移动云计算环境下任务调度的多目标优化方法.计算机研究与发展,2017,54(9):1909−1919. [doi: 10.7544/issn1000-1239.2017.20160757]

刘伟(1978－),男,湖北襄阳人,博士,副教授,CCF 专业会员,主要研究领域为移动边缘计算,绿色云计算.

杜薇(1978－),女,博士,副教授,CCF 专业会员,主要研究领域为服务计算,移动边缘计算.

黄宇成(1996－),男,硕士生,主要研究领域为移动边缘计算.

王伟(1979－),男,博士,副教授,博士生导师,主要研究领域为云计算与大数据.