一种面向移动边缘计算的多用户细粒度任务卸载调度方法

崔玉亚[1,2],张德干[1,2],张 婷[3],杨 鹏[1,2],朱浩丽[1,2]
(1. 天津理工大学天津市智能计算及软件新技术重点实验室,天津 300384; 2. 天津理工大学计算机视觉与系统省部共建教育部重点实验室,天津 300384; 3.天津体育学院体育经济与管理学院,天津 301617)

摘 要: 在移动边缘计算中(Mobile Edge Computing, MEC),任务卸载可以有效地解决移动设备资源受限的问题,但是将全部任务都卸载到边缘服务器并非最优.本文提出一种面向移动边缘计算的多用户细粒度任务卸载调度新方法,把计算任务看作一个有向无环图(Directed Acyclic Graph, DAG),对节点的执行位置和调度顺序进行了优化决策.考虑系统的延迟把计算卸载看作一个约束多目标优化问题(Constrained Multi-object Optimization Problem, CMOP),提出了一个改进的NSGA-Ⅱ算法来解决CMOP.所提出的算法能够实现本地和边缘的并行处理从而减少延迟.实验结果表明,算法能够在实际应用程序中做出最优决策.

关键词: 移动边缘计算;计算卸载;约束多目标优化问题;细粒度卸载调度;NSGA-Ⅱ
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112(2021)11-2202-06
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20201039

A Multi-User Fine-Grained Task Offloading Scheduling Approach of Mobile Edge Computing

CUI Yu-ya[1,2],ZHANG De-gan[1,2],ZHANG Ting[3],YANG Peng[1,2],ZHU Hao-li[1,2]
(1. *Tianjin Key Laboratory of Intelligent Computing & Novel Software Technology, Tianjin University of Technology, Tianjin* 300384, *China*;
2. *Ministry of Education Key Laboratory of Computer Vision and System, Tianjin University of Technology, Tianjin* 300384, *China*;
3.*School of Sports Economics and Management, Tianjin University of Sport, Tianjin* 301617, *China*)

Abstract: In mobile edge computing (MEC), task offloading can solve the problem of resource constraint on mobile devices effectively, but it is not optimal to offload all tasks to edge servers. In this paper, a multi-user fine-grained task offloading scheduling approach of mobile edge computation is proposed. The computation task is regarded as a directed acyclic graph (DAG), and task nodes' execution location and scheduling order are optimized. Considering the delay of the system, the computation offloading is considered as a constrained multi-objective optimization problem (CMOP), and an improved NSGA-Ⅱ algorithm is proposed to solve the CMOP. The proposed algorithm can realize local and edge parallel processing to reduce delay. The experimental results show that the algorithm can make the optimal decision in practical applications.

Key words: mobile edge computing; computation offloading; constrained multi-objective optimization problem (CMOP); fine-grained task offloading scheduling; NSGA-Ⅱ

1 引言

近几年随着工业物联网的不断发展,IoT设备在工业中的应用越来越广泛,例如智慧工厂、自然语言识别、自动驾驶等[1~5].这些都要求物联网设备具有处理计算密集型和延迟关键型任务的能力,目前的物联网设备由于电池、存储、计算能力的限制很难满足这些需求[6~16].MEC能够有效的降低移动设备的延迟,解决移动设备资源受限的问题[17].

计算卸载可以有效的缓解移动设备资源受限的问

收稿日期:2020-09-21;修回日期:2021-05-13;责任编辑:李勇锋
基金项目:国家自然科学基金(No.61571328);天津市重大科技专项(No.15ZXDSGX00050,No.16ZXFWGX00010);天津市科技支撑重点项目(No.17YFZCGX00360);天津市自然科学基金(No.18JCZDJC96800,No.18JCYBJC19300);天津市科技创新和131人才团队(No.TD13-5025,No.2015-23)

题. 将整个计算任务都卸载到边缘服务器处理并不是最优的,因为卸载需要额外的通信消耗,并且有些任务必须要在本地执行(GPS、I/O设备、加速器). 可以把一些复杂的任务看作由许多相互依赖的子程序组成的,移动设备根据策略选择哪些子程序在本地执行哪些在边缘服务器执行.

2　系统模型

我们考虑的是多用户多服务器的细粒度卸载场景,$M=\{1,2,3,...,\mathrm{NI}\}$个IoT,$K=\{1,2,3,...,k\}$个服务器节点,$N=\{1,2,3,...,n\}$个信道. IoT在一个时间段最多只能连接一个信道,同一个信道的IoT可能会发生信道间的干扰,并且每个IoT只能选择一个MEC服务器执行任务. 用一个有向无环图$G_m=(V_m,E_m)$来表示计算任务,其中$V_{m,v}=(v_{m,1},v_{m,2},v_{m,3},...v_{m,v})$表示IoT设备$m$的所有子任务集合,$e_m=(v_{m,i},v_{m,j})\in E_m$表示节点$v_{m,i}$和$v_{m,j}$之间的依赖关系,其中$v_{m,i}$和$v_{m,j}$是邻居节点,如果$v_{m,i}$是$v_{m,j}$的直接父节点,则$v_{m,i}$必须在$v_{m,j}$之前完成. 每个节点都可以用$T_{m,i}=(b_{m,i},c_{m,i},d_{m,i})$表示,其中$b_{m,i}$表示IoT设备$m$的第$i$个节点的数据大小,$c_{m,i}$表示本地执行节点$i$消耗的CPU资源,$d_{m,i}$表示在边缘服务器执行节点$i$消耗的CPU资源. 定义一个决策变量$S_{m,i}$,$S_{m,i}=0$表示节点$i$在本地执行,$S_{m,i}=1$表示节点$i$在边缘服务器执行.

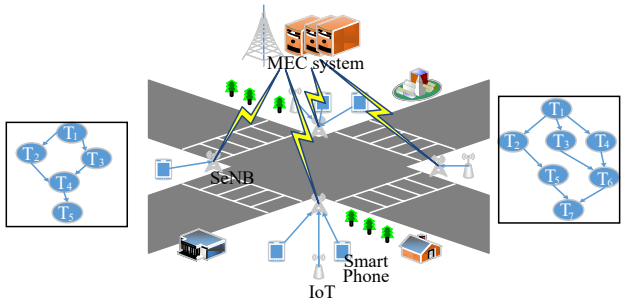

图1　系统模型

将任务节点划分为两个不相交的集合,一个在本地执行$V_{m,\mathrm{local}}$一个在边缘服务器执行$V_{m,\mathrm{server}}$,$V_{m,\mathrm{local}}\cap V_{m,\mathrm{server}}=\varnothing$并且$V_{m,\mathrm{local}}\cup V_{m,\mathrm{server}}=V_m$. 把需要卸载的节点融合在一起传输到边缘服务器,我们用$E_{m,\mathrm{partition}}$表示DAG被划分边的集合. 在不同位置执行的节点$v_{m,i}$和$v_{m,j}$会产生额外的通信代价($v_{m,i}$和$v_{m,j}$是邻居节点). 额外的通信主要与传输速率和传输的数据大小有关,我们用$w_{m,n(e(v_{m,i},v_{m,j}))}$表示IoT设备$m$通过信道$n$传输数据的延迟.

IoT设备和MEC服务器中配备有数据缓存器,用于存储等待执行的数据. 我们用$\mathrm{DB}_{\mathrm{local},m}$表示IoT设备$m$缓存器中数据的计算延迟,$\mathrm{DB}_{\mathrm{server},k}$表示边缘服务器$k$缓存器中数据的计算延迟.

$$\mathrm{DB}_{\mathrm{local},m}=\frac{\mathrm{db}_{m,l}u_{m,l}}{F_m^{\mathrm{local}}} \tag{1}$$

其中,$\mathrm{db}_{m,l}$表示本地缓存区中的数据大小,$u_{m,l}$表示本地执行缓存数据所需消耗的平均CPU资源,F_m^{local}表示IoT设备m的计算能力(即,每秒CPU的圈数).

$$\mathrm{DB}_{\mathrm{server},k}=\frac{\mathrm{db}_{k,c}u_{k,c}}{F_k^{\mathrm{server}}} \tag{2}$$

其中,$\mathrm{db}_{k,c}$表示MEC服务器缓存区中的数据大小,$u_{k,c}$表示MEC服务器执行缓存数据所需消耗的平均CPU资源,F_k^{server}表示MEC服务器k的可用计算资源.

定义1　T_m表示IoT设备m执行计算任务的延迟,T_m包括四个部分:1)本地节点的执行延迟$\mathrm{TL}_{m,i}$;2)边缘服务器节点的执行延迟$\mathrm{TS}_{m,i,k}$;3)$w_{m,n}(e(v_{m,i},v_{m,j}))$;4)缓存区等待时间.

本地节点执行的延迟:每个IoT设备的处理能力不同,本地IoT设备m的计算时间$\mathrm{TL}_{m,i}$计算公式如下:

$$\mathrm{TL}_{m,i}=\frac{c_{m,i}}{F_m^{\mathrm{local}}} \tag{3}$$

边缘服务器节点执行延迟:本文,我们假设一个MEC服务器有多个信道,每个信道可以连接多个MEC服务器,所有访问MEC服务器的用户都可以共享计算资源. 边缘服务器k的计算时间$\mathrm{TS}_{m,i,k}$公式如下:

$$\mathrm{TS}_{m,i,k}=\frac{d_{m,i}}{F_k^{\mathrm{server}}} \tag{4}$$

相关联节点的额外传输延迟:如果节点$v_{m,i}$和$v_{m,j}$($v_{m,i}$和$v_{m,j}$是邻居节点)被划分在不同的分区执行$v_{m,i}\in V_{m,\mathrm{local}},v_{m,j}\in V_{m,\mathrm{server}}$,则两个节点之间需要额外的传输时间.

总延迟T_m:将部分任务卸载到边缘服务器上执行,可以实现任务的并行处理. 令$\mathrm{ST}_{m,i},\mathrm{ET}_{m,i}$分别表示节点$v_{m,i}\in V_m$的开始时间和结束时间. 我们有:

$$\mathrm{ET}_{m,i}=\mathrm{ST}_{m,i}+T_{m,i}^{\mathrm{cmp}} \tag{5}$$

其中$T_{m,i}^{\mathrm{cmp}}$表示$v_{m,i}$的执行时间,$v_{m,i}$既可以在本地执行也可以在边缘服务器执行,因此$T_{m,i}^{\mathrm{cmp}}$的计算公式为:

$$\begin{aligned}T_{m,i}^{\mathrm{cmp}}&=(1-S_{m,i})\mathrm{TL}_{m,i}+S_{m,i}\mathrm{TS}_{m,i,k}\\&=(1-S_{m,i})\frac{c_{m,i}}{F_m^{\mathrm{local}}}+S_{m,i}\frac{d_{m,i}}{F_k^{\mathrm{server}}}\end{aligned} \tag{6}$$

$\mathrm{ST}_{m,i}$主要取决于$v_{m,i}$前置节点的完成时间以及缓存区的延迟. $\mathrm{ST}_{m,i}$的计算公式如下:

$$\mathrm{ST}_{m,i}\begin{cases}0 & ,i=1\\ \max\left\{\begin{matrix}(1-S_{m,i})\mathrm{DB}_{\mathrm{local},m}+S_{m,i}\mathrm{DB}_{\mathrm{server},k},\\ \max\limits_{p\in P_{m,i}}\left[\mathrm{ET}_{m,p}+\omega_{e_m}w_{m,n}(e(v_{m,p},v_{m,i}))\right]\end{matrix}\right\} & ,i>1\end{cases} \tag{7}$$

如果i是第一个任务节点,则$\mathrm{ST}_{m,i}\equiv 0$. 其中$P_{m,i}$是节点$v_{m,i}$直接前置任务节点集合. 其中$\omega_{e_m}$可以表示如下:

$$\omega_{e_m}=\begin{cases}1, & \text{if } e_m\in E_{m,\text{partition}}\\ 0, & \text{if } e_m\notin E_{m,\text{partition}}\end{cases} \tag{8}$$

因此,IoT设备m总的完成时间就是最后一个子任务的结束时间减去第一个任务的开始时间.

$$T_m=\mathrm{ET}_{m,v}-\mathrm{ST}_{m,1} \tag{9}$$

3 调度约束和卸载策略

3.1 调度约束

考虑到各个节点之间的执行优先级以及执行期限,需要满足以下约束:

A)运行截止期限约束:最后一个节点的完成时间不能大于整个任务的计算时间,在本模型中第一个任务和最后一个任务都是在本地执行的. 约束表示为:

$$0<\max\left\{\begin{matrix}DB_{\text{local},m},\\ \max\limits_{p\in P_{m,v}}\left[\mathrm{ET}_{m,p}+\omega_{e_m}w_{m,n}(e(v_{m,p},v_{m,v}))\right]\end{matrix}\right\}+\mathrm{TL}_{m,v}\leqslant T_m \tag{10}$$

$\mathrm{TL}_{m,v}$表示最后一个节点在本地的执行延迟.

B)优先级约束:若节点$v_{m,i}$是$v_{m,j}$的直接父节点,则$v_{m,i}$的执行优先级要比$v_{m,j}$高,$v_{m,i}$的优先级计算如下:

$$\mathrm{priority}(v_{m,i})=T_{m,i}^{\mathrm{cmp}}+\max_{v_{m,j}\in\mathrm{succ}(v_{m,i})}\mathrm{priority}(v_{m,j}) \tag{11}$$

其中,$\mathrm{priority}(v_{m,i})$表示节点$v_{m,i}$的优先权,$\mathrm{succ}(v_{m,i})$表示的是节点$v_{m,i}$的直接后继节点集合. 通过从最后一个任务节点$v_{m,v}$开始遍历DAG递归计算优先级. 最后一个节点的优先级可以计算为:

$$\mathrm{priority}(v_{m,v})=T_{m,v}^{\mathrm{cmp}} \tag{12}$$

因此,$\mathrm{priority}(v_{m,i})>\mathrm{priority}(v_{m,j})$.

C)节点完成期限约束:每一个任务节点$v_{m,j}$都必须在其所有先前任务都完成以及处理完组件本身之后才算结束,节点$v_{m,j}$的开始时间不得早于节点$v_{m,i}$的结束时间. 如果两个任务节点不在同一个分区执行还需要传输时间,我们给定这个约束如下:

$$\max\left\{\begin{matrix}(1-S_{m,j})\mathrm{DB}_{\text{local},m}+S_{m,j}\mathrm{DB}_{\text{server},k},\\ \left[\mathrm{ET}_{m,i}+\omega_{e_m}w_{m,n}(e(v_{m,j},v_{m,i}))\right]\end{matrix}\right\}\geqslant\mathrm{ST}_{m,i}+T_{m,i}^{\mathrm{cmp}} \tag{13}$$

$\omega_{e_m}w_{m,n}(e(v_{m,i},v_{m,j}))$表示两个节点的通信延迟,由式(8)可知,如果$e_m=(v_{m,i},v_{m,j})\notin E_{m,\text{partition}}$表示两个节点都在同一位置处理,不需要额外的通信时间,反之则需要.

3.2 卸载策略

任务节点的划分情况会根据IoT设备选择的信道和服务器节点动态变化,IoT设备的平均延迟消耗可以计算如式(14)所示. 其中$C_{m,n,k}$表示如果用户通过信道n将任务卸载到服务器k上执行则$C_{m,n,k}=1$,否则,$C_{m,n,k}=0$.

$$T=\frac{1}{M}\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{k=1}^{K}C_{m,n,k}\left\{\begin{matrix}\max\left\{\begin{matrix}\mathrm{DB}_{\text{local},m},\\ \max\limits_{p\in P_{m,v}}\left[\begin{matrix}\mathrm{ST}_{m,p}+(1-S_{m,p})\dfrac{c_{m,p}}{F_m^{\text{local}}}\\ +S_{m,p}\dfrac{d_{m,p}}{F_k^{\text{server}}}\\ +\omega_{e_m}w_{m,n}(e(v_{m,p},v_{m,v}))\end{matrix}\right]\end{matrix}\right\}\\ +\dfrac{d_{m,v}}{F_m^{\text{local}}}\end{matrix}\right. \tag{14}$$

最终,我们将IoT设备的细粒度卸载看作一个约束多目标优化问题(CMOP). 以最小化平均延迟为目的,最优卸载策略O_1可以表示如下:

$$\begin{aligned}&O_1\underset{\{S_{m,i},C_{m,n,k}\}}{:}\min\{T\}\\ &\text{s.t.}\\ &C1\text{: }S_{m,i}\in\{0,1\}\\ &C2\text{: }C_{m,n,k}\in\{0,1\}\\ &C3\text{: }\sum_{1}^{N}C_{m,n,k}=1\\ &C4\text{:(15)}\\ &C5\text{:(18)}\\ &C6\text{: }\mathrm{priority}(v_{m,i})>\mathrm{priority}(v_{m,j})\\ &C7\text{: }\forall m\in M,\forall n\in N,\forall k\in K\end{aligned} \tag{15}$$

约束$C1$表示节点在本地执行或在边缘服务器执行. $C2$表示IoT设备m是否通过信道n连接MEC服务器节点k. $C3$表示一个IoT在一次选择时间最多可以连接一个信道. $C6$表示如果节点$v_{m,i}$是$v_{m,j}$的直接父节点,则节点$v_{m,i}$的执行优先级要比$v_{m,j}$高.

4 多用户细粒度任务卸载调度方法

基于改进的NSGA-Ⅱ卸载调度决策方法可描述如下:

(1)初始化种群:我们将染色体的每个基因对应一个IoT设备,染色体上的基因数量就是IoT设备数量. IoT设备m的计算任务由ξ_m个节点组成,相对应基因的值可以表示为$\text{value}\in\{0,1,2,\ldots,2^{\xi_m}-1\}$. 将value转换为二进制,表示$m$中任务节点的卸载决策.

传统的随机初始化无法预测收敛速度,本文使用反向学习(Opposition-Based Learning, OBL)初始化种群. 同时考虑每个个体的反向体,这样个体和反向个体靠近最优解的几率就是50%,然后选择靠最优解最近的个体作为种群的初始个体.

(2) 约束处理:初始化种群后,我们根据式(14)来计算目标函数值. 本文提出的是约束的多目标优化问题(CMOP),存在染色体不满足式(15)中的约束,这些解为不可行解,但是我们不能完全抛弃不可行解,因为NSGA-Ⅱ是随机搜索的算法,丢弃不可行解可能会导致NSGA-Ⅱ陷入局部最优,将不可行解纳入搜索过程可以挖掘更多信息. 因此,还需要计算个体的约束违规程度DOV. 在演化的过程中同时考虑可行解和不可行解,我们通过快速非支配排序方法在不可行约束空间中寻找DOV较小的解,在可行约束空间中寻找最优解.

(3) 选择操作:初始化种群后,快速非支配排序算法根据初始解的质量以及拥挤度n_{cd}进行排序. 首先我们根据约束支配规则对种群进行分层,将所有的染色体分配到不同的前沿. 为了保证种群的多样性我们还需要估计每个解的拥挤度,我们根据平均延迟T和平均能耗E计算解ς两侧的两个点的平均距离. 将同一前沿上距离解ς最近的两个解看作顶点,以形成矩形. 解ς的拥挤度n_{cd}就是这个矩形的平均边长. 注意,如果ς是当前Pareto前沿的第一个或者最后一个解,$n_{cd,\varsigma}$为∞. 因此,对于任意两个解x_1和x_2,满足以下任一条件,则x_1优于x_2:

① x_1的支配等级小于x_2.

② x_1与x_2的支配等级相同,但$n_{cd,x_1} > n_{cd,x_2}$.

(4) 自适应交叉和变异算子:进化过程中动态调整交叉和变异的概率可以提高算法的收敛速度和收敛精度. 对于高适应度个体给予较低的交叉变异概率,这有利于保存种群中优良的个体,对于低适应度的个体给予高的交叉变异概率,这有利于劣质个体的改变.

(5) 更新种群:通过交叉变异产生新的子代种群后,我们使用第二步来计算他们的目标函数值和DOV. 然后根据快速非支配排序将整个种群(包含父种群和子种群)进行排序,并且计算他们的拥挤度n_{cd}. 然后根据非支配排序等级和n_{cd}进行排序,最优染色体被保留生成下一代. 当算法满足终止条件时,将最优前沿的解转换成二进制作为最佳决策. 否则,返回第三步,不断的迭代直到满足终止条件.

5　实验结果和分析

5.1　实验环境

为了验证I-NSGA-Ⅱ卸载调度算法性能,我们使用Matlab 2016a进行实验. 如图1所示,建立一个真实的网络场景. 每个IoT设备都运行真实的应用程序,如人脸识别应用,视频处理,语音识别等应用. 信道的时变性遵循瑞利分布,我们根据文献[6~9, 18]设置实验参数. IoT设备数量为40~120,信道数量为4~8,MEC服务器数量为{4, 6, 8, 10, 12},基站的覆盖范围为50m~100m.

5.2　对比实验

随机分配(random):将IoT中的任务节点随机划分为本地执行和在边缘服务器执行.

NSGA-Ⅱ:在考虑计算任务大小、信道传输速率、边缘服务器计算资源的情况下,使用传统的NSGA-Ⅱ多目标优化算法对多用户的任务节点进行划分.

文献[18]:在卸载之前使用多合并计算排序分割(MCSS)算法将终端设备上的任务分为本地和卸载两部分. 使用改进后的Kuhn-MunKras(KM)为每个用户匹配一个最低传输的MEC服务器和信道,最后通过优化本地CPU频率来减少本地执行消耗.

首先,我们分析延迟对比情况,从图2(a)可以看出,随着IoT数量的增加,四种算法的平均延迟都在增加,I-NSGA-Ⅱ(proposed)的平均延迟最低. 这是因为我们在进行任务划分的过程中考虑了边缘服务器的计算能力以及信道的传输速率,这样可以很大程度的提高任务节点划分的准确性和实时性,并且通过改进传统NSGA-Ⅱ提升了算法的收敛速度,在演化的过程中同时考虑可行解和不可行解增加了可行解的范围. 传统的NSGA-Ⅱ算法不管是在种群初始化还是约束处理方面与本文相比都存在较多不足. 文献[18]对任务节点划分后才进行信道选择和最优MEC服务器匹配,这将导致节点划分不够准确,而节点划分是卸载的重要组成部分. random随机分配任务执行节点的位置,没有经过任何算法的优化,因此在延迟方面表现最差. 图2(b)的分析与2(a)类似. 从图2(c)可以看出随着MEC服务器数量的增加,四种算法的平均延迟都在降低,这是因为MEC服务器的增加可以为IoT提供充足的计算资源,卸载到边缘处理的任务也就越多.

6　总结

本文提出一种面向移动边缘计算应用的多用户细粒度任务卸载调度方法. 将计算任务看作一个DAG,通过考虑任务之间的依赖性,计算任务的大小,本地计算资源以及服务器的计算能力将任务节点划分为本地和边缘两个部分,并为相关联节点增加调度约束. 为了最小化能耗和延迟,我们将卸载调度策略看作一个CMOP,并且提出一个改进的NSGA-Ⅱ算法来提高收敛速度增大解的空间. 此外,通过大量的实验证明了改进的NSGA-Ⅱ算法在卸载调度策略上的性能. 本文提出的卸载调度策略能够实现节点执行位置和调度顺序的最优决策,通过本地和边缘的并行处理减少系统的延迟和能耗.

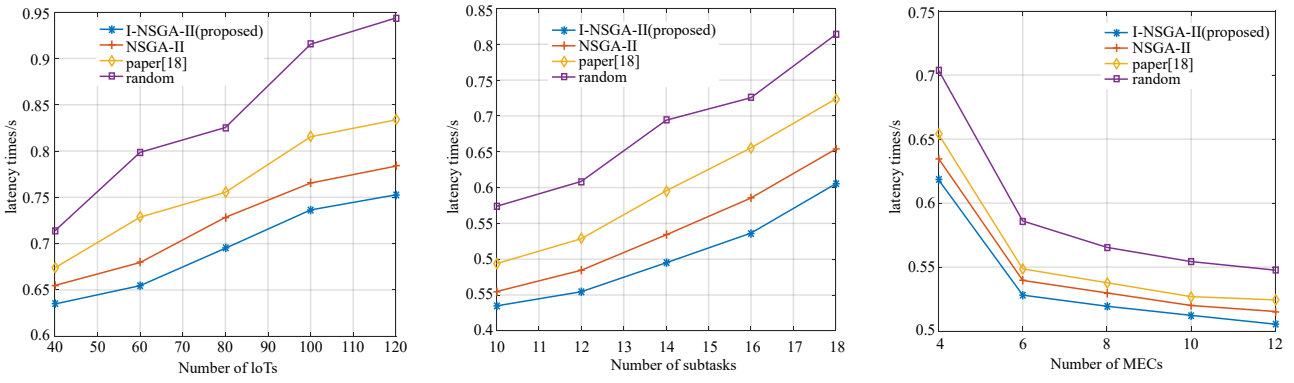

图2 (*a*) 不同数量IoT四种算法的平均延迟对比;(*b*)不同数量子任务四种算法的平均延迟对比;(*c*)不同数量MEC服务器四种算法的平均延迟对比

参考文献

[1] GAO J X. Novel approach of distributed & adaptive trust metrics for MANET[J]. Wireless Networks, 2019, 25(6): 3587 – 3603.

[2] CHEN C, CUI Y Y. New method of energy efficient subcarrier allocation based on evolutionary game theory[J]. Mobile Networks and Applications, 2021, 26(2): 523 – 536.

[3] Liu S, Liu X H, Zhang T. Adaptive repair algorithm for TORA routing protocol based on flood control strategy[J]. Computer Communications, 2020, 151(1): 437 – 448.

[4] GE H. New multi-hop clustering algorithm for vehicular Ad Hoc networks[J]. IEEE Transactions on Intelligent Transportation Systems, 2019, 20(4):1517 – 1530.

[5] ZHANG T. A new method of data missing estimation with FNN-based tensor heterogeneous ensemble learning for Internet of vehicle[J]. Neurocomputing, 2021, 420(1): 98 – 110.

[6] LIU S. Novel unequal clustering routing protocol considering energy balancing based on network partition & distance for mobile education[J]. Journal of Network and Computer Applications, 2017, 88(15):1 – 9.

[7] ZHANG T. Novel self-adaptive routing service algorithm for application of VANET[J]. Applied Intelligence, 2019, 49(5):1866 – 1879.

[8] YANG J N, MAO G Q. Optimal base station antenna downtilt in downlink cellular networks[J]. IEEE Transactions on Wireless Communications, 2019, 18(3): 1779 – 1791.

[9] ZHANG T. Novel optimized link state routing protocol based on quantum genetic strategy for mobile learning[J]. Journal of Network and Computer Applications, 2018, (122):37 – 49.

[10] 张德干. 一种基于前向感知因子的WSN能量均衡路由方法[J]. 电子学报, 2014, 42(1): 113 – 118.
ZHANG De-gan. A kind of routing protocol based on forward-aware factor for energy balance[J]. Acta Electronica Sinica, 2014, 42(1): 113 – 118. (in Chinese)

[11] 张德干. 一种基于Q-Learning策略的自适应车联网路由新算法[J]. 电子学报, 2018, 46(10): 2325 – 2332.
ZHANG De-gan. A kind of new routing algorithm with adaptivity for mobile IoT based on Q-Learning[J]. Acta Electronica Sinica, 2018, 46(10): 2325 – 2332. (in Chinese)

[12] Wang X, Song X D. A new clustering routing method based on PECE for WSN[J]. EURASIP Journal on Wireless Communications and Networking, 2015, (162): 1 – 13.

[13] Liu S, Liu X H. Novel dynamic source routing protocol (DSR) based on genetic algorithm bacterial foraging optimization (GA-BFO)[J]. International Journal of Communication Systems, 2018, 31(18): 1 – 20.

[14] Tang Y M, Cui Y Y. Novel reliable routing method for engineering of Internet of vehicles based on graph theory [J]. Engineering Computations, 2019, 36(1): 226 – 247.

[15] Liu S. Dynamic analysis for the average shortest path length of mobile Ad Hoc networks under random failure scenarios[J]. IEEE Access, 2019, 7: 21343 – 21358.

[16] Wu H, Zhao P Z. New approach of multi-path reliable transmission for marginal wireless sensor network[J]. Wireless Networks, 2020, 26(2): 1503 – 1517.

[17] Cui Y Y, Zhang T, Chen L. Novel method of mobile edge computation offloading based on evolutionary game strategy for IoT devices[J]. AEU-International Journal of Elec-

tronics and Communications, 2020, 118(5): 1 – 13.

[18] Rui L L. Computation offloading in a mobile edge communication network: a joint transmission delay and energy consumption dynamic awareness mechanism[J]. IEEE Internet of Things Journal, 2019, 1(1): 99.

作者简介

崔玉亚　男,1992年生,江苏淮安人,天津理工大学计算机科学与工程学院在读博士生,研究兴趣为网络通信、物联网、无线传感器网络、移动边缘计算等.

E-mail:844511468@qq.com

张德干　男,1970年生,湖北黄冈人,天津理工大学计算机科学与工程学院教授/博士,博导,研究兴趣为物联网、无线传感器网络、移动边缘计算、云计算等.

E-mail:zhangdegan@tsinghua.org.cn

张　婷(通信作者)　女,1972年,河北唐山人,天津体院学院体育经济与管理学院教授/博士,硕导,研究兴趣为物联网、无线传感器网络、移动边缘计算、大数据等.

E-mail:zhangtingts@163.com