DOI:10.13878/j.cnki.jnuist.20230731005

沈正林[1]　吴涛[1]　周启钊[1]　陈曦[2]

移动边缘计算中基于混合人工蜂群算法的计算卸载策略

摘要

计算卸载是移动边缘计算(Mobile Edge Computing,MEC)中的关键技术.针对多用户多 MEC 服务器场景中计算卸载策略的不足,本文提出一种混合人工蜂群算法(Artificial Reverse Sine-Cosine,ARSC).首先,使用反向学习策略初始化种群,优化种群的初始解;然后,在雇佣蜂阶段利用正余弦算法的全局最优引导信息,提升算法的局部搜索能力;最后,为了平衡算法的全局搜索能力和局部搜索能力,引入动态感知因子对算法的步长因子进行改进.仿真实验结果表明,相比基于粒子群算法的卸载策略、基于人工蜂群算法的卸载策略,ARSC 策略在系统时延、系统能耗、收敛性等指标上均有所改善.

关键词

移动边缘计算;计算卸载;人工蜂群算法;正余弦算法;多用户多 MEC

中图分类号 TN925.5;TP18

文献标志码 A

0　引言

近年来,随着物联网、云计算、大数据等技术的高速发展,用户数据规模呈指数型增长.传统的云计算模式需要上传大量本地数据到云服务器,但由于云服务器部署离本地终端较远,传输过程中的响应延迟、能量损耗、网络干扰、数据安全等问题难以避免.为了解决云计算中心离终端设备较远带来的时延、能耗等问题,学者们提出将云的功能向网络边缘转移[1].在接近移动终端设备的网络边缘端,移动边缘计算(Mobile Edge Computing,MEC)作为提供信息技术服务且具有计算能力的一种新型网络结构和计算范式被提出.

边缘计算并不是取代云计算,而是作为传统云计算的延伸.它将部分数据处理和服务迁移到离用户更近的网络边缘节点上,从而降低网络延迟、提高用户体验和数据安全性.然而,移动设备仍然面临着资源和能耗方面的限制,无法满足一些复杂或者实时性要求高的任务[2].因此,如何将这些计算任务卸载到边缘服务器进行处理,成为边缘计算中备受关注的关键技术.

国内外研究人员针对边缘计算中的计算卸载进行了大量研究.文献[3]将工业物联网环境下的任务卸载问题建模为一个多用户多边缘服务器问题,使用粒子群算法(Particle Swarm Optimization,PSO)对任务进行卸载.文献[4]针对多设备 MEC 系统中计算任务的执行时间以及终端设备的能耗最小化问题,构建了一个云服务器辅助的多 MEC 服务器计算卸载模型,并提出一种改进的人工蜂群算法(Artificial Bee Colony algorithm,ABC).文献[5]针对能量和频谱资源稀缺的问题,提出一种基于差分进化算法(Differential Evolution,DE)的任务卸载方法,该方案考虑了能源站的发射功率和边缘服务器的计算能力等约束条件,建立了无线能源供应的边缘计算网络模型.文献[6]将工业生产线中的计算卸载问题建模为多用户、多 MEC 时延优化问题,为了消除本地设备使用能耗来换取时延的问题,通过惩罚函数平衡时延与能耗,并提出一种基于粒子群算法优化的卸载策略.

任务卸载策略求解过程属于 NP 难问题[7],MEC 研究中大多将其转化为多因素优化问题.文献[8]在物联网场景下,使用遗传算法作为任务卸载策略,显著降低了设备的能耗和数据传输的时延.为了解决边缘计算中任务卸载调度的优化问题,文献[9]对蝙蝠群算法(BA)进行改进,提出一种改进混沌蝙蝠群协同卸载方法,该方案大大

收稿日期 2023-07-31

资助项目 四川省重点研发计划(23ZDYF0171,23RKX0645)

作者简介

沈正林,男,硕士生,主要从事边缘计算研究.1843963412@qq.com

吴涛(通信作者),女,博士,教授,主要从事并行计算、边缘计算与网络安全等方面的研究.wut@cuit.edu.cn

1　成都信息工程大学　计算机学院,成都,610225

2　西南民族大学　计算机科学与技术学院,成都,610225

减少了任务完成的时延,从而满足任务实时处理的需求.文献[10]通过对模型中的三个目标分别进行归一化,消除维度的影响,对模型进行了改进,并引入灰狼优化算法(GWO)求解模型.文献[11]基于非正交多址的多址边缘计算系统,研究了卸载决策、子信道分配、发射功率和计算资源分配的联合优化问题,使用鲸鱼群算法(WOA)解决了发射功率和子信道分配的问题.文献[12]将多代理生成对抗模仿学习和马尔可夫策略相结合以逼近专家性能,实现算法的在线执行,最后结合非支配排序遗传算法Ⅱ(NSGA-Ⅱ),对时延和能耗进行联合优化.

现有研究中的MEC卸载策略模型缺少对排队等待时延的考虑,无法有效均衡边缘服务器的计算资源.为了降低任务积压在服务器中导致系统时延增加的影响,采用积压队列模型,以防止边缘服务器中的任务队列因积压过多的计算任务导致系统总的时延增加,并针对多用户多MEC系统卸载模型,提出一种混合人工蜂群算法(Artificial Reverse Sine-Cosine,ARSC).ARSC使用反向学习策略对人工蜂群算法的初始化阶段进行优化,并充分考虑正余弦算法收敛性强的优点,通过融合正余弦算法,提升算法的局部搜索能力.同时,引入动态感知因子改进算法的步长因子,进一步平衡算法的全局搜索能力和局部探索能力.

1 系统模型

本研究构建了一个多用户多MEC系统架构,如图1所示.系统包含用户层、边缘层.边缘层有 m 个基站,MEC=$\{1,2,3,\cdots,m\}$,每个基站上部署一个边缘服务器.本研究主要着眼于移动边缘计算网络中卸载任务的调配,使计算资源的调配更加高效合理,因此移动设备产生的计算任务采用二元任务迁移模型.二元任务迁移模型是指移动设备产生的单个任务是不可拆分的,任务要么被卸载到边缘服务器,要么在本地终端进行计算.每个任务由一个三元组构成,$t_i=\{w_i,f_i,\mathrm{cpu}_i\}$,其中,$w_i$ 表示任务的数据量,f_i 表示每bit数据需要的cpu时钟周期数,cpu_i 表示任务所需要的计算资源.

1.1 时延模型

任务在不同的设备上执行有不同的时延模型:当任务在本地移动设备上执行,只需要考虑任务在移动设备上产生的计算时延;当任务被卸载到边缘服务器上执行,需要考虑任务上传的传输时延、服务器上的等待时延、计算时延、下传的传输时延.因为

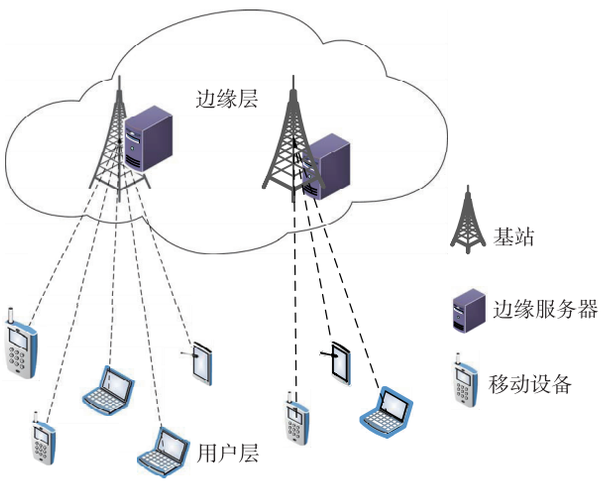

图1 系统架构

Fig. 1 System architecture

下传时延远远低于上传时延,故本文不考虑下传的传输时延.

1.1.1 本地计算时延模型

当移动设备在本地执行的时候,其任务执行时间如式(1)所示,其中,$f_{\mathrm{local},i}$ 为移动设备的cpu时钟频率.

$$T_{\mathrm{local},i}=\frac{w_i f_i}{f_{\mathrm{local},i}}. \tag{1}$$

1.1.2 积压队列模型

边缘服务器节点的计算能力是有限的,当一个服务节点中积压大量计算任务时,会产生等待时延,从而导致系统时延的增加.为了有效利用边缘服务器中的计算资源来减少等待时延,所以引入积压队列模型.当任务在边缘服务器中的计算时延超过在本地计算时延时,边缘服务器拒接卸载当前任务.边缘节点中任务队列的执行过程如图2所示.

任务1、任务2、任务3,任务4到达边缘服务器节点的时间分别为 t_1,t_2,t_3,t_4.假设,任务1~3到达服务器节点时,服务器的计算资源能够满足任务1~3立刻执行,而任务4到达边缘服务器节点时计算资源不能够满足任务4立即执行,需要等待其他任务执行完成后释放计算资源才能执行.假设任务1释放计算资源后能够满足任务4的需求,并且任务4在边缘服务器的计算时延不超过本地计算时延,则任务4的等待时延为任务1的执行完成时延 t_5 与任务4的到达时延 t_4 的差值.如果任务4在边缘服务器中的计算时延超过本地计算时延,则拒接卸载.

任务在边缘服务器中的计算时延存在如下两种情况:

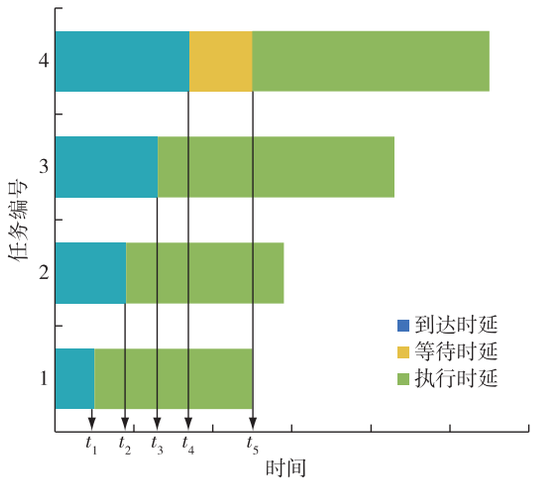

图 2　边缘节点任务队列执行过程

Fig. 2　Computing process of edge node tasks

1）当任务到达边缘服务器后,若当前边缘服务器的计算资源不能满足任务立即执行,此时需要等待任务队列中其他任务执行完成后方可参与计算,任务时延($T_{\text{mec},i}$) 如式(2) 所示:

$$T_{\text{mec},i} = T_{\text{trans},i} + T_{\text{wait},i} + T_{\text{execute},i},$$
$$\text{s.t.}\quad \begin{cases} \text{cpu}_i > \text{cpu}_j, \\ T_{\text{mec},i} \leqslant T_{\text{lcaol},i}. \end{cases} \tag{2}$$

传输时延($T_{\text{trans},i}$) 如式(3) 所示,其中,$\gamma_{i,j}$ 表示移动设备与边缘服务器之间的传输速率. 香农公式(4) 中,W 表示移动设备与边缘服务器之间的传输带宽,p_i 表示移动设备的发射功率,$H_{i,j}$ 表示移动设备和边缘服务器之间的信道增益,N_0 表示噪声功率.

$$T_{\text{trans},i} = \frac{w_i}{\gamma_{i,j}}, \tag{3}$$

$$\gamma_{i,j} = W \cdot \text{lb}\left(1 + \frac{p_i \cdot H_{i,j}}{N_0}\right). \tag{4}$$

执行时延($T_{\text{execute},i}$) 如式(5) 所示,其中,$f_{\text{mec},j}$ 表示边缘服务器的 cpu 时钟频率.

$$T_{\text{execute},i} = \frac{w_i f_i}{f_{\text{mec},j}}. \tag{5}$$

边缘服务器中的积压任务队列 $S(j)$ 采用先进先出队列模型(First Input First Output,FIFO),任务遵循先来先服务原则,积压队列如图 3 所示.通过任务队列可以计算任务的等待时延($T_{\text{wait},i}$),如式(6) 所示. 首先从任务队列中找到最快释放资源并且能够满足任务 i 需求的任务完成时间 $T_{\text{mec},k}$,然后再根据任务 i 的到达时间 $T_{\text{trans},i}$,两者的差值即任务 i 的等待时延.

$$T_{\text{wait},i} = \begin{cases} 0, & \text{s.t. cpu}_i \leqslant \text{cpu}_j; \\ T_{\text{mec},k} - T_{\text{trans},i}, & \text{s.t. cpu}_i > \text{cpu}_j. \end{cases} \tag{6}$$

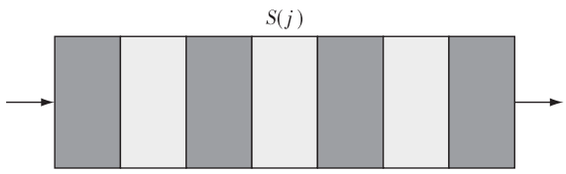

图 3　积压队列模型

Fig. 3　Backlog queue model

2）当任务到达边缘服务器后,若边缘服务器剩余资源满足任务所需计算资源,即 $\text{cup}_i \leqslant \text{cpu}_j$,$T_{\text{mec},i} \leqslant T_{\text{local},i}$,此时任务可以立即执行,任务完成时间如式(7) 所示:

$$T_{\text{mec},i} = T_{\text{trans},i} + T_{\text{execute},i},$$
$$\text{s.t.}\quad \begin{cases} \text{cpu}_i < \text{cpu}_j, \\ T_{\text{mec},i} \leqslant T_{\text{lcaol},i}. \end{cases} \tag{7}$$

1.2　能耗模型

ARSC 算法设计的初衷是降低移动设备的时延和能耗,所以在能耗模型中只考虑移动设备的执行能耗和传输能耗.

任务在本地执行的能耗和任务的传输能耗分别如式(8)和(9)所示,其中,$T_{\text{local},i}$ 表示任务的本地执行时延,$T_{\text{trans},i}$ 表示任务传输时延,$p_{\text{local},i}$ 表示移动设备的本地计算功率,p_i 表示移动设备的发射功率.

$$E_{\text{local},i} = T_{\text{local},i} \cdot p_{\text{local},i}, \tag{8}$$

$$E_{\text{trans},i} = T_{\text{trans},i} \cdot p_i. \tag{9}$$

任务总的时延 T_{all} 和总的能耗 E_{all} 分别如式(10) 和(11) 所示,其中,$k = 1$ 代表任务在本地执行,$k = 0$ 代表任务在边缘服务器中执行.

$$T_{\text{all}} = \sum_{t=0}^{n} (kT_{\text{local},i} + (1-k)T_{\text{mec},i}), \tag{10}$$

$$E_{\text{all}} = \sum_{t=0}^{n} (kE_{\text{local},i} + (1-k)E_{\text{trans},i}), \tag{11}$$

1.3　目标函数

把当前系统模型转化为有约束条件下联合时延和能耗的最小化问题,目标函数 Q 如式(12) 所示.

$$Q = \exp\left(-\left(\theta_1 \frac{T_{\text{L}} - T_{\text{all}}}{T_{\text{L}}} + \theta_2 \frac{E_{\text{L}} - E_{\text{all}}}{E_{\text{L}}}\right)\right), \tag{12}$$

$$\text{s.t.}\quad \begin{cases} T_{\text{all}} < T_{\text{L}}, \\ E_{\text{all}} < E_{\text{L}}, \\ T_{\text{mec},i} \leqslant T_{\text{local},i}, \\ \text{cpu}_i < \text{cpu}_j, \\ \theta_1, \theta_2 \in [0,1], \\ \theta_1 + \theta_2 = 1. \end{cases} \tag{13}$$

其中:T_L 为所有任务在本地设备执行的时延;E_L 为所有任务在本地设备执行的能耗;T_{all} 和 E_{all} 越小,目标函数值 Q 越小;系数 θ_1 和 θ_2 分别表示时延和能耗在系统中所占的比重,为了平衡时延和能耗,本文 θ_1 和 θ_2 取值都为 0.5.

2 基于 ARSC 算法的卸载决策

2.1 基本的人工蜂群算法

1) 初始化阶段:设定蜜源数量 N_S、问题维度 D、最大迭代次数 G_{max}、最大失败尝试次数 N_{limit},根据式(14)初始化蜜源位置.其中,X_j^{max}、X_j^{min} 分别表示各维度的上下界,rand(0,1) 表示[0,1] 上均匀分布的随机数.

$$X_{i,j}=X_j^{min}+\text{rand}(0,1)\times(X_j^{max}-X_j^{min}),\quad i\in\{1,2,\cdots,N_S\},j\in\{1,2,\cdots,D\}. \tag{14}$$

2) 雇佣蜂阶段:雇佣蜂更新蜜源位置的公式如式(15)所示.其中,ϕ 为[-1,1] 之间均匀分布的随机数,$X_{i,j}^{new}$ 表示新的蜜源位置.比较新蜜源位置和旧蜜源位置,使用贪婪策略保留较好的蜜源.

$$X_{i,j}^{new}=X_{i,j}+\varphi(X_{i,j}+X_{k,j}),k\in\{1,2,\cdots,N_S\}. \tag{15}$$

3) 跟随蜂阶段:跟随蜂收集雇佣蜂传递的蜜源信息按照式(16)计算跟随每一个雇佣蜂的概率,然后以轮盘赌策略的方式来选择跟随对象.其中:$f(X_i)$ 表示最小化问题的函数值;fitness_i 代表适应度值(式(17)),适应度值越大的蜜源越优秀,越容易被选择.

$$p_i=\frac{\text{fitness}_i}{\sum_{j=1}^{N_S}\text{fitness}_j}, \tag{16}$$

$$\text{fitness}_i=\begin{cases}1+|f(X_i)|, & f(X_i)<0;\\ \dfrac{1}{1+f(X_i)}, & f(X_i)\geqslant 0.\end{cases} \tag{17}$$

4) 侦察蜂阶段:当蜜源位置超过 N_{limit} 次都没有更新,则雇佣蜂会放弃该蜜源转化为侦察蜂,并按照式(14)重新初始化蜜源位置.

2.2 混合人工蜂群算法

2.2.1 反向学习策略

传统的人工蜂群算法初始化蜜源位置采用随机策略,而蜜源位置在解空间的分布直接影响算法的收敛速度和最终结果.为了改善蜜源位置,采用反向学习(Opposition-Based Learning,OBL)对初始解进行优化.反向学习[13]是一种优化策略,基本思想是通过计算当前解的反向解来扩大搜索范围,以此来找出问题更好的解.

根据式(18)生成反向解,然后使用贪婪策略从正向解和反向解的集合 H 中选出 N_S 个目标函数值最优的解作为初始种群 S_0.

$$X_{i,j}^{reverse}=X_j^{max}+X_j^{min}-X_{i,j}, \tag{18}$$

$$H=\{X_1,X_2,\cdots,X_{N_S},X_{N_S+1}^{reverse},\cdots,X_{2N_S}^{reverse}\}. \tag{19}$$

2.2.2 正余弦策略

基本人工蜂群算法(ABC)及其搜索策略侧重于提升全局搜索能力[14],但由于 ABC 在每一代不直接使用全局最优信息,只是存储其最优信息,导致其局部搜索能力较弱[15].为了增强算法的局部搜索能力,在人工蜂群算法的雇佣蜂阶段引入正余弦算法(Sine-Cosine Algorithm,SCA)[16].这种方法通过采用正余弦算法中的全局最优解的引导信息来增强算法的局部搜索能力,利用正余弦算法的正余弦模型的振荡特性来平衡探索与开发能力.混合人工蜂群算法的雇佣蜂阶段蜜源位置更新策略为

$$X_{i,j}^{t+1}=\begin{cases}X_{i,j}^t+r_1\sin r_2\cdot|r_3X_{best}-X_{i,j}^t|, & r_4<0.5;\\ X_{i,j}^t+r_1\cos r_2\cdot|r_3X_{best}-X_{i,j}^t|, & r_4\geqslant 0.5.\end{cases} \tag{20}$$

其中:r_2 是[0,2π] 之间的随机数,表示当前个体更新时朝向或远离当前最优个体所能达到的最远距离;r_3 是[0,2] 之间的随机数,决定了当前已知最优个体影响当前个体的程度;r_4 是[0,1] 之间的随机数,表示在正弦函数和余弦函数之间的随机取值,解除了移动步长和移动方向可能的耦合关系;r_1 是当前个体在迭代中的移动步长,

$$r_1=a-\frac{at}{t_{max}}. \tag{21}$$

其中:a 是一个常数;t_{max} 是最大迭代次数.当步长因子 r_1 较高时,表示有利于全局搜索,当步长因子较低时有利于局部搜索.因此,步长因子对寻找最优解至关重要,故在此处引入动态感知因子(Dynamic Perception,DP)[17].动态感知因子可以帮助算法在前期阶段提升算法的全局搜索能力,扩大搜索的解空间,在算法后期提升算法的局部搜索能力,提升算法获取最优解的能力.

$$\text{DP}=\begin{cases}a\left(1-\left(\dfrac{t}{t_{max}}\right)^2\right), & t>\dfrac{t_{max}}{2};\\ a\left(1-\dfrac{t}{t_{max}}\right)^2, & t\leqslant\dfrac{t_{max}}{2}.\end{cases} \tag{22}$$

2.2.3 编码和解码

为了使人工蜂群算法和正余弦算法能够应用于边缘计算卸载问题,需要对算法进行离散化处理.

假设边缘服务器的数量为 m、任务数量为 n,卸载决策如式(23)所示,X 表示一个卸载策略,其中,x_i 的取值表示分配到边缘服务器或者本地设备.

$$X=\{x_1,x_2,\cdots,x_n\},\quad x\in[0,m]. \tag{23}$$

1)对人工蜂群蜜源位置更新策略进行离散化:式(24)中的 $V_{i,j}$ 表示位移的增量.假设边缘服务器数量 m 为5,任务数量 n 为5,$V_{i,j}$ 的取值范围是[-5,5].$V_{i,j}$ 大于0表示计算位置向后移动,$V_{i,j}$ 小于0表示向前移动,$V_{i,j}$ 等于0表示任务继续在当前服务器进行计算.人工蜂群算法位置更新策略如式(25)所示,为了保证更新后的位置值在区间[0,5],需要对相加后的位置值取余.

$$V_{i,j}=\varphi(X_{i,j}+X_{k,j}), \tag{24}$$

$$X_{i,j}=\begin{cases}(X_{i,j}+V_{i,j})\%m, & v_{i,j}\geqslant 0;\\ (X_{i,j}+V_{i,j}+m)\%m, & v_{i,j}<0.\end{cases} \tag{25}$$

2)对正余弦算法的位置更新进行离散化:式(26)中的 $V_{i,j}$ 表示向着最优解移动的方向和大小.为了保证 $V_{i,j}$ 的值在区间[-5,5],当 $V_{i,j}$ 的值比-5低时,令 $V_{i,j}=-5$,当 $V_{i,j}$ 的值大于5时,令 $V_{i,j}=5$.正余弦算法的位置更新公式如式(27)所示.

$$V_{i,j}=\begin{cases}r_1\cdot\sin r_2\cdot|\,r_3X_{\text{best}}-X_{i,j}^t\,|, & r_4<0.5;\\ r_1\cdot\cos r_2\cdot|\,r_3X_{\text{best}}-X_{i,j}^t\,|, & r_4\geqslant 0.5.\end{cases} \tag{26}$$

$$X_{i,j}^{t+1}=\begin{cases}(X_{i,j}^t+V'_{i,j})\%m, & v_{i,j}\geqslant 0;\\ (X_{i,j}^t+V'_{i,j}+m)\%m, & v_{i,j}<0.\end{cases} \tag{27}$$

2.2.4 算法伪代码

ARSC 算法如下所示:

ARSC 算法

1) 初始化蜜源个数 N_S,雇佣蜂和跟随蜂的数量与蜜源数量相等.初始化迭代次数 $G_{\max}$ 和蜜源最大开采次数 N_{limit}.
2) For i = 1 to N_S
3) 利用反向学习策略初始化蜜源,并从蜜源中选取目标函数值最优的蜜源存储.
4) END
5) For n = 1 to $G_{\max}$
6) For i = 1 to N_S
7) 在跟随蜂阶段使用融合了正余弦算法的位置更新策略,对蜜源位置进行更新.更新全局最优蜜源,并记录蜜源的开采次数.
8) END
9) 通过蜜源的适应度值,计算每一个蜜源被选择的概率.
10) For m = 1 to N_S
11) 跟随蜂阶段使用轮盘赌策略选择一个蜜源,在这个蜜源周围进行探索,更新全局最优蜜源,并记录蜜源的开采次数.
12) END
13) DO
14) 判断蜜源的开采次数是否达到最大限制 N_{limit},当达到最大开采次数产生一个新蜜源,替换掉旧蜜源.
15) While $i < N_S$
16) END
17) 输出全局最优蜜源

3 仿真结果分析

3.1 仿真环境和参数

本文所有算法通过 Python3 实现,实验环境为 Windows10 64 位操作系统,Intel Core i5-11400H CPU,16 GB 内存.仿真用到的主要参数如表1所示.

表1 参数设置

Table 1 Parameter setting

参数	数值
移动设备计算功率 $p_{\text{local},i}$/W	1 ~ 5
移动设备发射功率 p_i/W	6 ~ 10
移动设备 cpu 时钟频率 $f_{\text{local},i}$/GHz	1 ~ 3
每 bit 数据需要的时钟周期 f_i/(cycle/bit)	800 ~ 1 200
传输宽带 W/MHz	10
边缘服务器计算速率 $f_{\text{mec},j}$/GHz	4 ~ 8
传输信道增益 $H_{i,j}$	2×10^{-10} ~ 2×10^{-6}
信道高斯噪声功率 N_0/W	1×10^{-9}
任务所需要的计算资源 cpu_i/GHz	1 ~ 3
边缘服务器最大计算资源 cpu_j/GHz	8

3.2 结果分析

根据表1的仿真参数进行实验,本文采用本地卸载策略(LA)、随机卸载策略(RA)、基于粒子群算法的卸载策略(PSO)、基于人工蜂群算法的卸载策略(ABC)、混合人工蜂群算法卸载策略(ARSC)来分析系统总的时延和能耗.

3.2.1 不同任务数量下的能耗和时延

为了比较不同任务数量下的各卸载决策的能耗和时延,本组实验参数设置为任务数量分别为50、

100、150、200、250,边缘服务器数量设置为 10 个,任务的数据量为 20 MB.实验结果如图 4 所示.由图 4 可知,在不同任务数量下 ARSC 方法的时延和能耗都优于其他方法.在任务数量为 250 时,ARSC 方法相较于本地卸载和随机卸载分别降低 51.3%、21.8%的时延和 90.3%、47.2%的能耗;相较于 ABC 方法和 PSO 方法,在任务数量为 250 的则分别降低 16.3%、16.6%的时延和 11.9%、31.5%的能耗.由此可见,ARSC 算法的局部搜索能力和全局搜索能力相较于 ABC 算法都得到了改善.

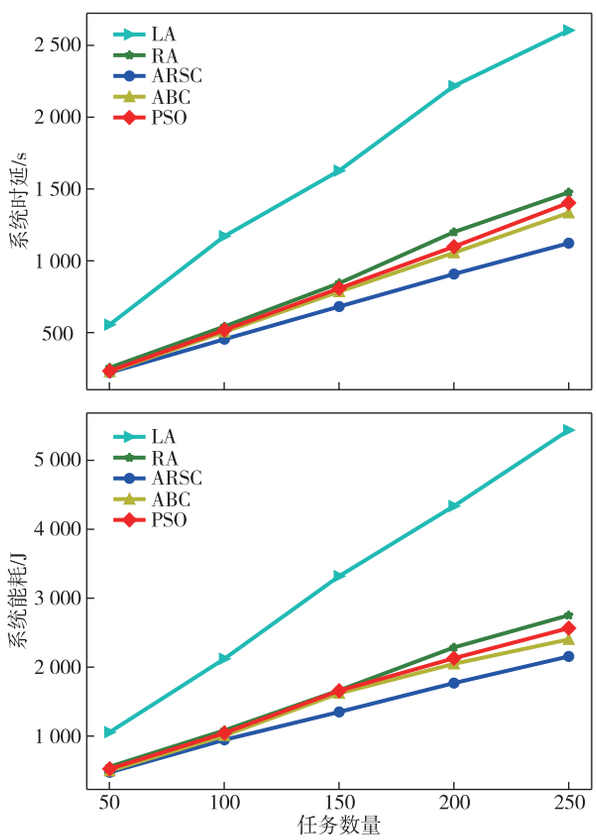

图 4　不同任务数量下的系统时延和能耗

Fig. 4　System latency and energy consumption for different number of tasks

3.2.2　各方法收敛性对比

为了比较 ARSC、ABC、PSO 三种算法的收敛性,设置任务数量为 50,边缘服务器数量为 10,实验结果如图 5 所示.由图 5 可知,ARSC 算法有很强的全局搜索能力,在算法的前期不断寻找全局最优解,且在后期拥有良好的局部搜索能力.在时延上,ARSC 相较于 ABC 和 PSO 算法,分别降低 6.4%和 11.7%;在能耗上,ARSC 算法相较于 ABC 算法和 PSO 算法,分别降低 5.2%和 6.6%.

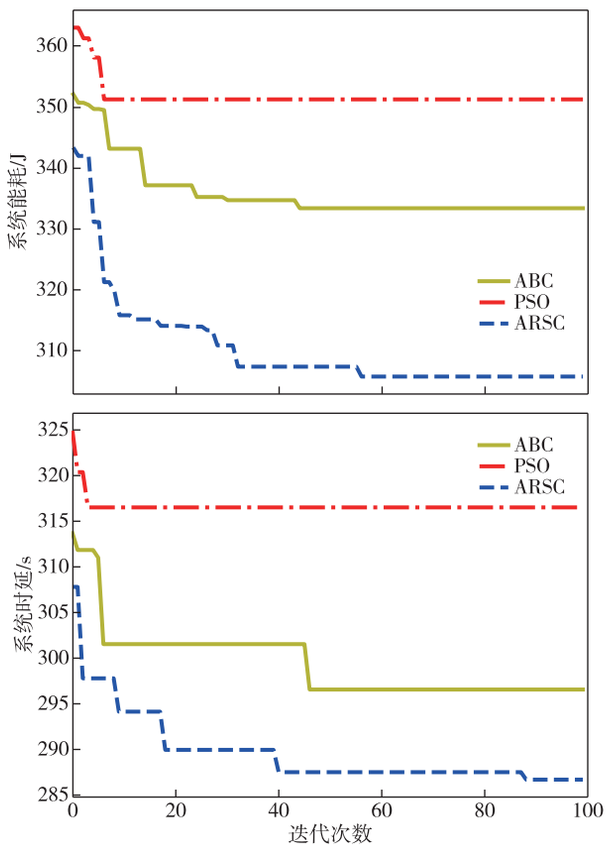

图 5　方法收敛性比较

Fig. 5　Comparison of algorithm convergence

3.2.3　任务的等待时延分析

为了分析任务所需计算资源增多时,是否造成系统的等待时延增加.本组实验参数设置任务数量为 50,任务的数据量为 20 MB,边缘服务器的数量为 10,任务所需资源量为 1 、1.5、2、2.5 和 3 GHz.由图 6 可知.当任务所需计算资源为 0.5、1 和 1.5 GHz 时,任务的等待时延在总的系统时延中所占比重较少,此时的边缘服务器资源能够满足同时执行多个任务,所以系统总的时延增加较少.当任务所需的计算资源达到 1.5 GHz 以上时,等待时延在总的系统时延中占比较大,系统总的等待时延增加较多,此时的边缘服务器资源不能同时满足多个任务同时执行,造成统总的时延和等待时延的增加.ARSC 卸载策略相比其他卸载策略总的等待时延最小.

3.2.4　不同任务数据量下的时延和能耗

为了比较不同任务数据量下的各卸载决策的能耗和时延,本组实验参数设置为任务数量为 50,边缘服务器数量设置为 10 个,任务的数据量分别 7、14、21、28 和 35 MB.实验结果如图 7 所示.由图 7 可知,随着任务的数据量增加本地卸载策略的时延和能耗

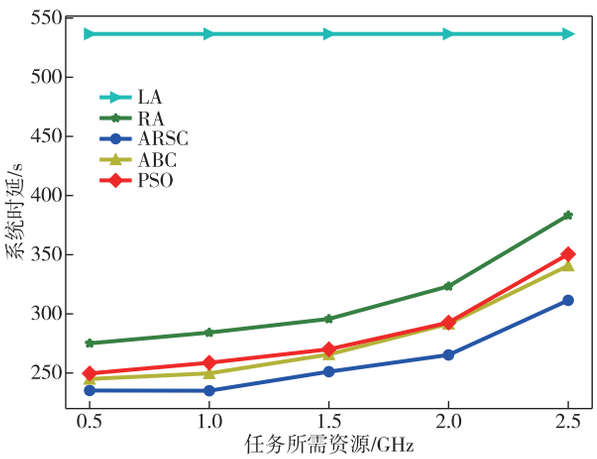

图 6　任务所需计算资源对系统时延的影响

Fig. 6　Impact of computational resources required for tasks on system latency

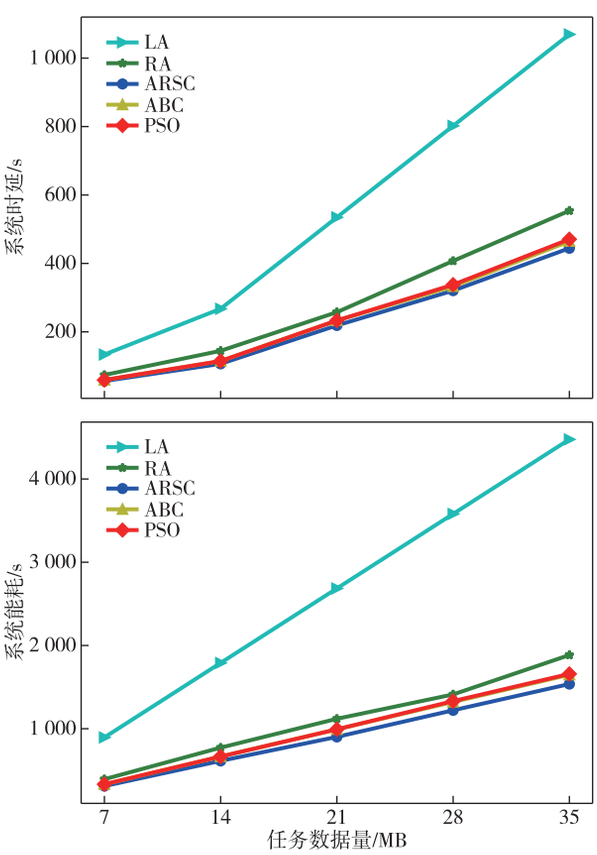

图 7　不同任务数据量下的系统时延和能耗

Fig. 7　System latency and energy consumption for different task data volumes

会快速增加,而基于 ARSC 方法的卸载策略的时延和能耗相对于其他方法都是最优的.在数据量为 35 MB 下,相较于 LA、RA、PSO、ABC 卸载策略,时延分别降低 61.6%、19.1%、5.4%、2.5%,能耗分别降低 66.1%、25.8%、8.8%、8.3%.

4　结论

本文针对多用户多 MEC 场景下计算卸载问题进行了研究.首先,将该场景下的计算卸载问题转化为联合时延和能耗最小化模型,并提出积压队列模型,防止边缘服务器中积压过多计算任务导致的系统总时延的增加.其次,提出一种混合人工蜂群算法卸载策略 ARSC,该算法利用正余弦算法的全局最优引导信息提升人工蜂群算法的局部搜索能力,同时使用反向学习策略优化算法的初始阶段,并引入动态感知因子进一步平衡算法的全局搜索能力和局部搜索能力.最后,对算法进行离散化处理,使其符合边缘计算应用模型.通过实验分析表明,ARSC 算法相较于 ABC、PSO 算法,在系统时延、能耗、系统总的代价等指标上均有改善.

参考文献

References

[1] 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

[2] 董思岐,李海龙,屈毓锛,等.移动边缘计算中的计算卸载策略研究综述[J].计算机科学,2019,46(11):32-40
DONG Siqi, LI Hailong, QU Yuben, et al. Survey of research on computation unloading strategy in mobile edge computing[J]. Computer Science, 2019, 46(11): 32-40

[3] Liang L, Xiao J T, Ren Z, et al. Particle swarm based service migration scheme in the edge computing environment[J]. IEEE Access, 2020, 8: 45596-45606

[4] 章呈瑞,柯鹏,尹梅.改进人工蜂群算法及其在边缘计算卸载的应用[J].计算机工程与应用,2022,58(7):150-161
ZHANG Chengrui, KE Peng, YIN Mei. Improved artificial bee colony algorithm and its application in edge computing offloading[J]. Computer Engineering and Applications, 2022, 58(7): 150-161

[5] Sun Y Y, Song C H, Yu S M, et al. Energy-efficient task offloading based on differential evolution in edge computing system with energy harvesting[J]. IEEE Access, 2021, 9: 16383-16391

[6] 罗斌,于波.移动边缘计算中基于粒子群优化的计算卸载策略[J].计算机应用,2020,40(8):2293-2298
LUO Bin, YU Bo. Computation offloading strategy based on particle swarm optimization in mobile edge computing[J]. Journal of Computer Applications, 2020, 40(8): 2293-2298

[7] Islam A, Debnath A, Ghose M, et al. A survey on task offloading in multi-access edge computing[J]. Journal of

Systems Architecture,2021,118:102225
[8] Abbasi M, Mohammadi-Pasand E, Khosravi M R. Intelligent workload allocation in IoT-fog-cloud architecture towards mobile edge computing [J]. Computer Communications,2021,169:71-80
[9] 简琤峰,陈家炜,张美玉.面向边缘计算的改进混沌蝙蝠群协同调度算法[J].小型微型计算机系统,2019,40(11):2424-2430
JIAN Chengfeng,CHEN Jiawei,ZHANG Meiyu.Improved chaotic bat swarm cooperative scheduling algorithm for edge computing [J]. Journal of Chinese Computer Systems,2019,40(11):2424-2430
[10] Feng S L,Chen Y J,Zhai Q H,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
[11] Pham H G T,Pham Q V,Pham A T,et al.Joint task offloading and resource management in NOMA-based MEC systems: a swarm intelligence approach [J]. IEEE Access,2020,8:190463-190474
[12] 刘建华,李炜,刘佳嘉,等.基于多代理模仿学习的普适边缘计算资源分配[J].南京信息工程大学学报,2024,16(1):83-96
LIU Jianhua,LI Wei,LIU Jiajia,et al.Resource allocation for pervasive edge computing based on multi-agent imitation learning[J].Journal of Nanjing University of Information Science & Technology,2024,16(1):83-96
[13] Tizhoosh H R.Opposition-based learning: a new scheme for machine intelligence[C]//International Conference on Computational Intelligence for Modelling,Control and Automation and International Conference on Intelligent Agents,Web Technologies and Internet Commerce. November 28 - 30, 2005, Vienna, Austria. IEEE, 2006:695-701
[14] 匡芳君,金忠,徐蔚鸿,等.Tent 混沌人工蜂群与粒子群混合算法[J].控制与决策,2015,30(5):839-847
KUANG Fangjun,JIN Zhong,XU Weihong,et al.Hybridization algorithm of Tent chaos artificial bee colony and particle swarm optimization[J]. Control and Decision,2015,30(5):839-847
[15] Alatas B.Chaotic bee colony algorithms for global numerical optimization[J].Expert Systems with Applications,2010,37(8):5682-5687
[16] Mirjalili S.SCA: a sine cosine algorithm for solving optimization problems[J].Knowledge-Based Systems,2016,96:120-133
[17] 雍龙泉,黎延海,贾伟.正弦余弦算法的研究及应用综述[J].计算机工程与应用,2020,56(14):26-34
YONG Longquan,LI Yanhai,JIA Wei.Literature survey on research and application of sine cosine algorithm[J].Computer Engineering and Applications,2020,56(14):26-34

Computational offloading strategy based on hybrid artificial bee colony algorithm in mobile edge computing

SHEN Zhenglin[1] WU Tao[1] ZHOU Qizhao[1] CHEN Xi[2]
1 School of Computer Science,Chengdu University of Information Technology,Chengdu 610225,China
2 School of Computer Science and Technology,Southwest Minzu University,Chengdu 610225,China

Abstract Computational offloading is an essential technology in Mobile Edge Computing (MEC).To address the shortage of computational offloading strategies in multi-user and multi-MEC server scenarios,this paper proposes a hybrid artificial bee colony approach (Artificial Reverse Sine-Cosine,ARSC).First,the opposition-based learning strategy is used to initialize the population and optimize the initial solution of the population.Then the global optimal bootstrap information of the sine-cosine algorithm is exploited to improve the local search capability in the employed bee stage.Finally,to balance the global and local search capability of the approach,the step size factor is adapted by introducing dynamic perception.Simulation results show that the proposed ARSC approach outperforms offloading strategies based on particle swarm algorithm and artificial bee colony algorithm in convergence,latency,and energy consumption.

Key words mobile edge computing(MEC);computation offloading;artificial bee colony algorithm;sine-cosine algorithm;multi-user and multi-MEC server