一种融合遗传蚁群算法的 ad hoc 云任务卸载算法

余思东[1] 黄 欣[1] 赵志刚[2*]

[1](广西农业职业技术学院信息与机电工程系 广西 南宁 530007)

[2](广西大学计算机与电子信息学院 广西 南宁 530004)

摘 要 针对 ad hoc 云中的任务卸载问题,设计一种多目标任务卸载决策模型。综合考虑任务完成时间、能耗和额外开销进行卸载决策,并选取簇头节点作为集中控制器进行合理的任务分配。提出一种融合遗传算法和蚁群算法的任务卸载算法,利用遗传算法的快速搜索能力得到可行解,将其作为蚁群算法的初始信息素,再利用蚁群算法的正反馈机制实现对任务分配方案的精确求解。仿真结果表明,该算法与随机任务分配算法、异构感知任务分配算法和遗传算法相比,能有效降低任务完成时间和能量消耗。

关键词 ad hoc 云 多目标 任务卸载 遗传算法 蚁群算法

中图分类号 TP393 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2020.11.031

A TASK OFFLOADING ALGORITHM OF AD HOC CLOUD MERGING GENETIC ALGORITHM AND ANT COLONY ALGORITHM

Yu Sidong[1] Huang Xin[1] Zhao Zhigang[2*]

[1](*Department of Information and Electromechanical Engineering, Guangxi Agriculture Vocational and Technical College, Nanning 530007, Guangxi, China*)

[2](*College of Computer and Electronics Information, Guangxi University, Nanning 530004, Guangxi, China*)

Abstract Aiming at the task offloading problem in ad hoc cloud, this paper designs a multi-objective task offloading decision model. The task completion time, energy consumption and overhead are comprehensively considered to make the offloading decision and select the cluster head node as the centralized controller for reasonable task assignment. This paper also proposes a task offloading algorithm based on genetic algorithm and ant colony algorithm. We utilized the fast search capability of genetic algorithm to obtain the feasible solution, and then used it as the initial pheromone of ant colony algorithm. According to the positive feedback mechanism of ant colony algorithm, we obtained the task assignment scheme accurately. The simulation results show that the proposed algorithm can effectively reduce task completion time and energy consumption compared with random task assignment algorithm, heterogeneity-aware task allocation algorithm and genetic algorithm.

Keywords ad hoc cloud Multi-target Task offloading Genetic algorithm Ant colony algorithm

0 引 言

近年来,随着移动互联网技术的迅速发展,移动云计算(Mobile Cloud Computing,MCC)[1]逐渐成为了缓解移动设备资源受限问题的有效方法,并且越来越受到学术界的重视。MCC 为移动设备提供了高效的远程计算服务,移动用户可以通过 MCC 将计算密集型的任务卸载到远程云[2]、本地微云[3]和 ad hoc 云[4]处理,任务完成后再将计算结果返回,以此降低任务在本地处理的开销,缓解移动设备资源受限的问题。在 MCC 的应用场景中,远程云虽然计算能力强,但会带来延迟和开销较高的问题;本地微云计算能力强且通信带宽大,但却无法保证用户能够随时随地接入;而基

收稿日期:2019-06-10。国家自然科学基金项目(61363067);广西高校中青年教师科研基础能力提升项目(2019KY1408)。
余思东,副教授,主研领域:计算机网络技术。**黄欣**,副教授。**赵志刚**,教授。

于 ad hoc 网络[5]架构的 ad hoc 云作为无预设基础设施下 MCC 的一种特殊扩展,用户可以通过与附近的移动设备共享其相对丰富的闲置资源进行任务卸载,具有部署快速和扩展灵活等优点[6]。由于 ad hoc 网络中节点的移动性、异构性,以及动态变化的网络拓扑,如何设计动态高效的任务卸载算法成为了亟待解决的问题。

在 ad hoc 云系统中,任务卸载不仅需要考虑完成时间和能量消耗,还要考虑自身的电池能量、信道环境和可用资源节点信息等。虽然现有工作对卸载决策和资源分配都分别进行了较为细致的分析研究,但任务的卸载决策和资源分配显然不能分开处理。此外,受信道环境、电池续航能力等因素的影响,系统中可用资源节点在为终端提供服务时显然会消耗自身资源,这将严重影响其参与任务卸载的积极性,并且系统中任务的动态到达性以及任务完成后被移除的随机性,会产生资源分配不均衡等问题。

为实现高效的任务卸载,现有研究为 MCC 设计了各种策略让移动用户决定是否将任务卸载到远程执行。为了最小化用户开销,文献[7]提出了一种基于博弈论的多代理节点资源分配方案,但该方案仅将延迟开销作为约束条件而忽略了其对用户体验的影响,并且没有考虑时变信道的影响。文献[8]则尝试通过 Lyapunov 优化来实现电量消耗、处理速度和服务价格的平衡,但仅考虑了对延迟不敏感的应用,并且使用传输队列的大小间接度量应用程序的延迟。文献[9]在决定任务卸载时综合考虑了能耗、延迟和价格因素,但没有考虑任务的传输调度和无线信道的时变特性。

目前对 ad hoc 云中任务卸载的研究仍处于初步阶段,仅有很少的研究针对这种特殊的分布式架构提出了有效的任务卸载方案。为解决计算密集型应用卸载问题,文献[10]提出了一种在线批调度启发式方法,动态地选择移动设备进行任务卸载。针对网络中节点的异构特性,文献[11]根据能耗和延迟约束设计了一种异构感知的任务分配算法,但没有考虑节点的移动性和网络拓扑的动态变化特性。为使移动用户能够获得最优的任务卸载决策,文献[12]提出了一种基于马尔可夫决策过程的卸载模型,动态决定将任务卸载到可用的资源节点。为降低用户能耗和计算成本,文献[13]提出了一种基于微云辅助的任务分配机制,制定了一个两阶段 Stackelberg 博弈来确定节点提供的执行单元数量,而用户将以此确定补偿策略,但是该机制在任务分配时并未考虑可用移动设备的资源异构特性。文献[14]提出了一种用于异构移动云的通用卸载框架,它使用移动设备、附近的小型云和远程云来提高 MCC 服务的性能和可用性,然而,所提出的框架并没有充分考虑 ad hoc 云的特性。为了满足移动设备的延迟限制,文献[15]提出了一种节能任务调度方案,并针对不同的任务,设计了一种自适应概率调度器,不仅能满足延迟约束,而且能在执行计算密集型实时应用程序时最小化能耗,但其计算复杂度较高。

以上研究主要针对最小化用户能耗和延迟等目标来设计解决方案,但没有综合考虑用户的多目标需求以及 ad hoc 云中的多因素影响进行任务卸载决策和资源分配。鉴于此,本文综合考虑用户多目标需求设计任务卸载决策模型,并提出一种融合遗传算法和蚁群算法的任务分配算法,综合考虑用户需求动态决策,充分利用遗传算法的快速搜索能力和蚁群算法的正反馈机,根据网络中节点的实际状态进行高效的任务分配,最后通过仿真验证了所提算法的有效性。

1 系统模型及卸载决策

1.1 系统模型

为了不失一般性,本文选取 ad hoc 网络中的一个分簇[16]作为 ad hoc 云系统,系统中只存在一个簇头节点,其他节点则为簇内成员节点,并且簇头节点往往具有距离簇内成员节点较近、相对移动性较低、能量稳定等特点。假设在 ad hoc 云系统中有一个移动用户(Mobile User, MU)节点和 N 个移动微云(Mobile Cloudlet, MC)节点,MU 有一组计算密集型任务 M 需要处理,如果仅仅依靠自身处理这些任务很可能会耗尽自身资源并且带来较高的延迟,十分影响用户体验。而其他 N 个 MC 节点则拥有相对丰富的闲置资源(如计算能力和电池电量),在 Wi-Fi 覆盖范围内,MU 可以将部分或全部任务卸载到 MC 节点上执行,任务完成后再将计算结果返回,从而实现对自身资源的扩展。在资源受限的情况下,各节点设备不会无偿地为其他节点提供资源,而分布式的网络架构也很难实现最大化利用系统资源,因此本文采用集中式策略来进行任务卸载过程的统一调度控制,并选取需要实时与所有节点进行信息交互的簇头节点作为集中控制器。MU 和 MC 节点均通过与集中控制器的信息交互来实现整个任务卸载的过程。由于 MU 的任务卸载模型几乎完全涵盖了实际移动应用程序的各个方面,其较高的复杂性可能会使问题难以解决,因此为了简化计算,本文设计如图 1 所示的任务卸载模型。

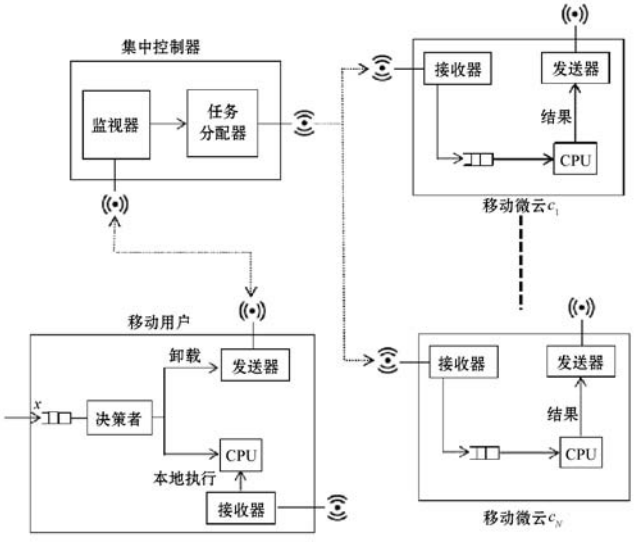

图 1　任务卸载模型

选定系统中的簇头节点作为集中控制器,MU 根据需求将任务按照优先级排序,并且当任务序列达到上限时,后续到达的任务将被放弃执行。对比任务在本地执行的开销和卸载到 MC 的开销之后,MU 决定是将任务卸载还是留在本地执行,并且将需要卸载的任务队列信息发送给集中控制器。集中控制器负责收集节点信息,并根据节点的实际状态将 MU 决定卸载的任务分配给相应的 MC,任务执行完成后 MC 再将结果返回给 MU。在本文中,集中控制器虽然同样具备 MC 的功能,但由于其作为簇头节点需要通过与网络中所有节点进行周期性的信息交互来实时动态地管理网络中所有节点的信息,因此默认其只负责集中调度控制,而不作为任务卸载请求和计算资源提供设备,具体流程如下:

(1) 各节点设备周期性地向集中控制器发送自身设备信息,包括计算能力、通信能力、可用电池资源、任务队列等信息。

(2) 集中控制器根据收集到的节点信息对节点编号并将信息整理至可用资源列表,为后续任务卸载做准备。

(3) MU 根据需求向集中控制器发送任务卸载请求。

(4) 集中控制器将收集到的节点信息和网络状态发送给 MU。

(5) MU 根据收到的信息作出卸载决策并将需要卸载的任务队列信息发送给集中控制器。

(6) 集中控制器采用合理的资源分配算法将任务分配给相应的 MC 并将信息反馈给 MU。

(7) MU 根据集中控制器的反馈信息将需要卸载任务的数据发送给相应的 MC。

(8) MC 在收到数据后进行处理,完成后将结果返回给 MU 并将信息反馈给集中控制器。

(9) 集中控制器将已完成任务从队列中移除,重复以上过程直到任务队列为空。

1.2　卸载决策

在资源有限的情况下,网络中的节点不会无偿地为其他用户提供服务,因此影响 MU 进行任务卸载决策的因素主要有任务完成时间、能耗和需要额外支付给 MC 的开销。完成时间主要包括两部分:传输时间和计算时间;能耗也包括两部分:传输能耗和执行能耗;额外支付给 MC 的开销主要取决于其计算速度和剩余可用负载。

假设 MU 决定将大小为 D 编号为 m 的任务卸载到编号为 n 的 MC 上执行,根据香农极限理论,最大数据传输速率可以表示为:

$$R_n = B\log(1 + SINR_n) \tag{1}$$

式中:R_n 为数据传输速率;B 为带宽;$SINR_n$ 为信号与干扰加噪声比,通常表示无线链路的质量,可以通过式(2)计算:

$$SINR_n = \frac{P_n^{\mathrm{t}} H_n}{\sigma_n^2 B + \sum_{i \neq n} P_i^{\mathrm{t}} H_i} \tag{2}$$

式中:P_n^{t} 为数据传输功率;H_n 为信道增益;σ_n^2 为噪声功率。则传输时间 T_n^{t} 为:

$$T_n^{\mathrm{t}} = D/R_n \tag{3}$$

传输能耗为:

$$E_n^{\mathrm{t}} = P_n^{\mathrm{t}} T_n^{\mathrm{t}} \tag{4}$$

需要额外支付给 MC 节点的开销定义为:

$$X_n = \lambda f_n^{\mathrm{c}} + \delta L_n \tag{5}$$

式中:f_n^{c} 为 MC 的 CPU 时钟频率;L_n 为 MC 的剩余可用负载量;λ、δ 为权重系数并且可以根据 MC 的实际状态设置。显然,计算速度越快剩余可用负载越大的 MC 会消耗 MU 更多的开销,但同时也会降低任务完成时间和能耗。任务 m 在 MC 上的执行时间为:

$$T_n^{\mathrm{e}} = D/f_n^{\mathrm{c}} \tag{6}$$

执行能耗为:

$$E_n^{\mathrm{e}} = kT_n^{\mathrm{e}} \tag{7}$$

式中:k 为 MC 的处理功率。本文暂不考虑任务完成后数据回传阶段产生的能量消耗和传输时间,因为返回结果的大小应当远小于输入数据的大小。因此,这一阶段的消耗很小,可以忽略。根据以上计算定义函数 $f_{\mathrm{c}}(x)$ 为任务卸载的总开销,则:

$$f_{\mathrm{c}}(x) = \alpha(T_n^{\mathrm{t}} + T_n^{\mathrm{e}}) + \beta E_n^{\mathrm{t}} + \gamma X_n \tag{8}$$

式中:α、β、γ 为完成时间、能耗、额外支付的开销这三个因素占总开销的权重系数,并且可以根据 MU 的实际需求动态设置,$\alpha + \beta + \gamma = 1$。同理,如果 MU 选择将

任务留在本地执行,则执行时间和能耗分别为:

$$T_m^l = D/f_m^l \tag{9}$$

$$E_m^l = gT_m^l \tag{10}$$

式中:f_m^l为 MU 的 CPU 时钟频率;g 为 MU 的处理功率。则任务在本地执行的总开销为:

$$f_l(x) = \mu T_m^l + \eta E_m^l \tag{11}$$

式中:μ、η 为执行时间、能耗占总开销的权重系数,并且同样可以根据 MU 的实际状态设置,$\mu + \eta = 1$。当且仅当$f_c(x) < f_l(x)$时,MU 才会将任务卸载到 MC 执行,反之则留在本地执行。

2 融合遗传蚁群算法的任务分配算法

合理的任务分配算法能充分利用节点的计算资源,提升整个网络的系统性能。但当网络中存在大量节点时,用传统的数学方法来解决这类任务分配问题往往需要大量的计算时间,而启发式算法可以在相对较短的时间内获得接近最优解的结果。相比于其他启发式算法,遗传算法由于前期收敛速度快且较为稳定而广泛应用于任务调度领域,但却存在易陷入局部收敛而无法获得全局最优解的问题;蚁群算法则具有良好的适应能力和正反馈机制,在求解多目标组合优化问题上有着独特的优势,但由于初始信息素积累较慢容易导致迭代时间较长。因此本文采用融合遗传算法和蚁群算法的方法来解决任务分配问题,核心思想是利用遗传算法的快速收敛能力得到可行解,在其迭代速度降低到一定水平而陷入局部最优之前,将求得的可行解对应转化为蚁群算法的初始信息素并开始新的迭代过程。这样不仅可以避免使遗传算法陷入局部收敛,也解决了蚁群算法初始信息素积累缓慢的问题,从而实现两种算法的有机结合。

2.1 基于遗传算法的快速搜索

遗传算法[17]通过模拟生物进化过程搜索最适应环境的个体,将任务分配问题的求解问题转化为染色体种群编码问题,根据设计的适应度函数值对相应的任务分配方案进行评估,利用选择、交叉和变异等遗传操作建立迭代过程,使产生的新一代个体在经过不断的迭代进化之后,性能不断优于上一代,逐渐求解出接近最优方案的任务分配方案。

2.1.1 染色体编码

为了能通过遗传算法进行迭代寻找合理的任务分配方案,首先需要将任务分配问题进行编码,形成染色体种群,从而建立实际任务分配方案和染色体种群编码之间的联系。染色体长度即为任务队列的最大容量 l,其中的元素对应相应的任务,元素的值代表该任务被分配到的 MC 节点的编号,编号由集中控制器统一分配管理,并且一般为固定不变的值。将染色体编码表示为 $G_i = (g_1, g_2, \cdots, g_l)$,其中 $i = 1, 2, \cdots, S$,S 为算法编码空间的大小即任务分配方案的大小;g_j表示任务分配结果,并且 $g_j \in [1, N]$,$j \in [1, l]$,例如当 $G = (3,2,1,2,3)$时,表示任务 1 和 5 将被分配给编号为 3 的 MC 节点执行,任务 2 和 4 被分配给编号为 2 的 MC 节点执行,任务 3 被分配给编号为 1 的 MC 节点执行,特别的是,当选择将任务留在本地执行时 g_j为 0,其他情况以此类推。

2.1.2 种群初始化

为加快算法的收敛速度,假设初始种群中的每个任务按照一定的随机概率分配给 MC 执行,且 MC 被选择执行的概率为:

$$P_i = \frac{E_i^j}{\sum_{j=1}^{N} E_i^j} \tag{12}$$

式中:E_i^j表示任务 i 分配给编号为 j 的 MC 执行时所需要消耗的能量,主要包括数据在传输过程中的能耗与在设备 j 上的执行能耗。

2.1.3 适应度函数设计

合理的适应度函数设计能够有效评估种群中染色体的环境适应能力,同时适应度函数的值也对应着任务分配方案性能优劣的评判,并最终决定着求解出的任务分配方案是否接近最优解的判断。由于遗传算法中的适应度函数值一般为非负值,因此本文结合式(8)定义适应度函数为:

$$F(x) = \alpha(T_n^t + T_n^e)^{-1} + \beta(E_n^t)^{-1} + \gamma(X_n)^{-1} \tag{13}$$

式中:α、β、γ 分别表示完成时间、能耗、需要额外支付的开销的权重系数。

2.1.4 遗传操作

选择、交叉和变异是三种基本的遗传算法操作,分别对应着自然界中生物繁衍过程中的正常交配、跨种群杂交和基因突变等现象,是保证遗传算法能够不断产生新的优秀个体的根本。对其具体设计如下:

(1) 选择操作模拟了自然界中的自然选择过程,能够根据种群中个体的适应度函数值将具有较大值的个体筛选出来,以便更好地从种群中获得相对优秀的个体。本文根据适应度函数值比重选择的形式和轮盘赌的方式来实现选择操作。对于染色体群 $G = \{G_1, G_2, \cdots, G_S\}$,个体 $G_j \in G$ 的适应度值为 $F(G_j)$,则其选择概率为:

$$P_s = \frac{F(G_j)}{\sum_{j=1}^{S} F(G_j)} \tag{14}$$

式(14)决定了更新后的种群染色体的概率分布情况。

(2) 交叉操作模拟了自然界中生物有性繁殖的基因重组过程,下一代的个体可以通过这种交叉重组的方式获得上一代的优秀基因,从而产生比上一代更加适应环境的个体。本文采用面向知识领域的位交叉来模拟交叉操作,并以此来计算个体染色体上每一位基因的适应度函数值,并结合式(8)定义基因的适应度函数为:

$$h(x) = \alpha(T_n^{\mathrm{t}} + T_n^{\mathrm{e}}) + \beta E_n^{\mathrm{t}} + \gamma X_n \tag{15}$$

在计算得到每一位基因的适应度函数值后,可以根据染色体上对应位置基因的适应度函数值大小对基因进行筛选,并选择将适应度函数值较小的基因保留下来。

(3) 变异操作模拟了自然界中普遍存在的基因突变现象,合理的设计不仅有利于种群多样性的丰富,还能防止算法在获得最优解之前提前收敛。与另外两种遗传操作相比,变异操作仅充当可选择性的非必要辅助手段,仅在求得的任务分配方案不合理时才会使用。这里同样采用面向领域知识的位变异来模拟变异操作,类似地,基因的变异概率依然与适应度函数值大小相关,可以定义为:

$$P_l^i = 1 - \frac{h_i}{\sum_{j=1}^{l} h_j} \tag{16}$$

式中:h_i表示染色体上第 i 位基因的适应度函数值。显然,任务完成时间越长、能量消耗和额外开销越高的任务,分配结果对应的基因更容易发生变异。

2.1.5　种群更新操作

按照适应度函数值从大到小对种群中个体进行排序,并用前 T 个个体替代上一代种群中的对应个体,从而形成新的种群。

2.2　基于蚁群算法的精确求解

蚁群算法具备较好的鲁棒性、并行性和正反馈特性,可以很好地应用于任务调度问题[18]。但是当需要求解的问题规模较大时,由于该算法的本质决定了信息素的产生需要消耗大量的时间,容易导致算法收敛速度较慢,因此需要结合遗传算法前期的快速搜索能力实现对任务分配方案的精确求解。为了将任务分配问题对应为蚂蚁种群的外出觅食问题,需要将能够充分反映节点计算资源丰富度的硬件性能信息对应为蚁群算法的初始信息素。

2.2.1　算法初始化

将信息素的值赋给 MC,并且信息素值的大小表示 MC 节点上可用资源的多少。根据前期遗传算法输出的最终结果可以解析出对应的任务分配方案,将该方案对应地转化为 MC 节点的初始信息素值,根据影响任务执行的异构因素定义每个 MC 节点的初始信息素值为:

$$\tau_i(0) = \tau_i^{\mathrm{C}} - \tau_i^{\mathrm{G}}(0) \quad i = 1,2,\cdots,N \tag{17}$$

式中:τ_i^{C}为设定的信息量常数,主要取决于需要求解问题规模的大小,并且与 MC 节点的计算速度、负载和电池能量相关;$\tau_i^{\mathrm{G}}(0)$为根据遗传算法输出的任务分配方案所转换的对应信息量,表示分配给对应 MC 节点的计算任务负载大小。

2.2.2　转移概率设计

转移概率的大小是将下一个任务分配给系统中合适的 MC 节点的主要参考依据,这里同样根据节点的异构性将节点的计算速度、剩余可用负载和剩余电池能量作为其性能优劣的主要判断依据,因此在 t 时刻的转移概率的设计主要取决于每个节点的计算速度、剩余可用负载、剩余电池能量和信息素值。则转移概率可以表示为:

$$P_i(t) = \begin{cases} \dfrac{[\tau_i(t)]^{\omega} \times [A_i(t)]^{\varphi}}{\sum_j [\tau_i(t)]^{\omega} \times [A_i(t)]^{\varphi}} & i,j \in N \\ 0 & i,j \notin N \end{cases} \tag{18}$$

式中:$\tau_i(t)$表示 t 时刻编号为 i 的 MC 的信息素值;$A_i(t)$为 t 时刻 MC 的相对性能;ω、φ 分别为两者的权重系数。$A_i(t) = \lambda f_i^{\mathrm{c}} + \delta L_i + \xi E_{\mathrm{b}}$,其中:$f_i^{\mathrm{c}}$为节点的计算速度;$L_i$为节点的剩余可用负载;$E_{\mathrm{b}}$为剩余电池能量;$\lambda$、$\delta$ 和 ξ 为权重系数。显然计算速度越快、剩余可用负载越大和电池能量越高的节点更有可能为 MU 提供服务。

2.2.3　信息素更新

当外出觅食的蚂蚁从节点转移时,节点上的信息素浓度就会相应地发生变化,因此需要及时更新信息素的值才能提高蚁群算法的收敛精度。这里根据求得的可行解对应的任务分配方案及任务的实际完成情况,对求得的所有路径上的信息素值进行更新,则信息素值的更新依据可以定义为:

$$\tau_i(t+1) = \tau_i(t) + \Delta\tau_i(t) \quad i = 0,1,\cdots,N \tag{19}$$

式中:当任务执行成功时有 $\Delta\tau_i(t) = k_{\mathrm{s}} f_{\mathrm{c}}(x)$,当任务执行失败时有 $\Delta\tau_i(t) = k_{\mathrm{l}} f_{\mathrm{c}}(x)$,$k_{\mathrm{s}}$、$k_{\mathrm{l}}$分别为任务执行成功、失败的奖惩因子,并且 $k_{\mathrm{s}} > k_{\mathrm{l}}$。

2.3　融合算法流程描述

为充分利用遗传算法的全局快速搜索能力和蚁群算法良好的正反馈机制,对它们进行优势互补,在完成了基于遗传算法的快速搜索和基于蚁群算法的精确求解设计之后,给出融合算法的详细流程,如算法 1

所示。

算法 1 融合算法

输入：随机产生的初始种群。

输出：全局最优的任务分配方案。

步骤 1 参数初始化。

步骤 2 初始化种群。

步骤 3 计算种群中个体的适应度值。

步骤 4 选择适应度值高的个体作为父代染色体。

步骤 5 按概率进行遗传操作。

步骤 6 判断是否满足约束条件，是则转到下一步，否则将适应度值函数值较高的个体加入种群并返回步骤 3。

步骤 7 将染色体种群中适应度函数值最高的个体作为最优解输出并进行下一步。

步骤 8 根据步骤 7 的最优解输出初始化蚁群。

步骤 9 根据转移概率选择下一节点。

步骤 10 更新信息素。

步骤 11 计算适应度值。

步骤 12 判断是否满足约束条件，是则输出结果，否则返回步骤 9 重新进行迭代操作。

3 仿真对比分析

为验证本文融合遗传蚁群算法的任务卸载算法（Genetic Ant Colony Algorithm，GACA）的性能，将其与随机任务分配算法（Random Task Assignment Algorithm，RTA）、异构感知任务分配算法（Heterogeneity-aware Task Allocation Algorithm，HTA）[9] 和遗传算法（Genetic Algorithm，GA）进行仿真对比，RTA 是将任务完全随机地分配给可用节点，而 HTA 则是根据节点性能和任务大小进行任务分配。

仿真场景中假设所有移动设备随机分布在一个边长为 100 m 的正方形区域内，并且区域内所有节点都在 Wi-Fi 覆盖范围内缓慢移动，选取其中 10 个 MC 节点的位置信息如表 1 所示，作为集中控制器的簇头节点位于区域正中并保持静止。无线信道参数设置为：带宽 $B=5$ MHz，噪声功率 $\sigma_n^2=50$ dBm，信道增益 $H_n=d_n^{-\varepsilon}$，其中：d_n 为 MC 和 MU 的距离；$\varepsilon=4$ 为路径损耗因子。假设 MU 的数据传输功率为 $P_n^t=1.5$ W，计算速度为 600MIPS（Million Instructions Per Second，MIPS，即 CPU 每秒处理的百万级的机器语言指令数），MC 的计算速度分别设置为 800MIPS、1 200MIPS 或1 600MIPS，任务设置为 6 组共 30 个，如表 2 所示。通过任务完成情况来验证所提算法的有效性，衡量指标为任务完成时间和能量消耗，其余参数根据节点的实际状态设置。

表 1 MC 的位置

节点	位置	节点	位置
1	46,25	6	49,73
2	42,64	7	92,27
3	62,67	8	36,54
4	70,18	9	58,70
5	74,15	10	24,86

表 2 任务设置

组	任务大小（指令数/10^{12}）				
1	0.002 6	0.045 2	0.233 9	0.739 2	1.171 2
2	0.001 8	0.044 8	0.236 6	0.732 3	1.119 0
3	0.002 9	0.043 8	0.231 4	0.732 1	1.257 5
4	0.001 6	0.044 0	0.234 7	0.730 2	1.337 1
5	0.002 7	0.043 8	0.240 0	0.731 1	1.442 7
6	0.002 8	0.044 4	0.233 5	0.737 2	1.401 8

任务完成时间是影响用户体验最重要的因素，图 2 为 GACA、RTA、HTA、GA 在相同任务设置情况下的平均完成时间随着节点数量不断变化的关系曲线。

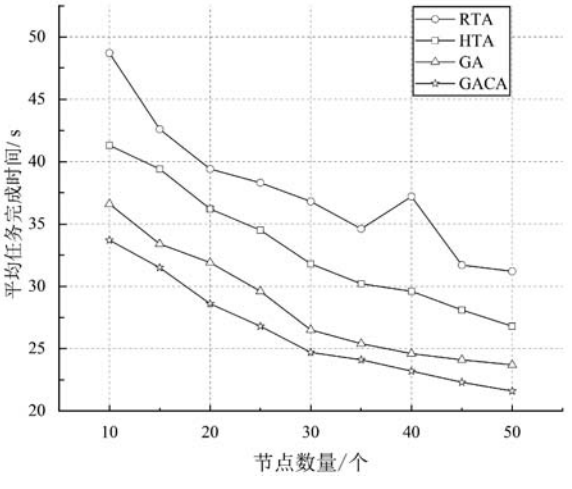

图 2 平均任务完成时间对比

随着节点数量的增加，任务的平均完成时间总体呈下降趋势，这是因为任务数量固定的情况下，节点数量的增加会显著提高任务分配的效率。其中：对所有任务进行完全随机分配的 RTA 性能较差并且不够稳定，尽管节点数量在不断增加，RTA 也无法充分地利用可用资源；HTA 根据节点设备的性能进行任务分配，忽略了节点间的通信开销，虽然也能不断降低任务执行时间，但总体效果一般；GA 与 GACA 更加有效，

并且当系统中可用节点设备数量较少时,这两种算法下任务的平稳完成时间相差不明显,这是因为当系统中的可用 MC 节点数量较少时,蚁群算法对于任务分配方案的精确求解能力受到了一定的限制,而遗传算法可以在前期利用全局搜索能力快速得出较优的任务分配方案。随着系统中可用资源节点设备数量的不断增加,由于 GACA 相比于 GA 具有更精确的求解能力,所以任务的平均完成时间呈现出更明显的下降趋势。

能量消耗同样是影响用户体验的另一重要因素,由于节点能量有限,因此较低的能耗能支持设备运行更长的时间,图 3 为本文 GACA 和 RTA、HTA、GA 在相同任务设置情况下的平均任务执行能耗随着节点数量不断变化的关系曲线。

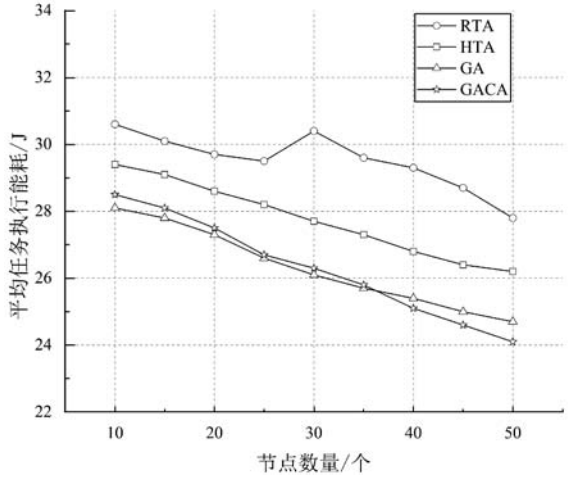

图 3　平均任务执行能耗对比

RTA 的性能依然较差并且不够稳定,任务的平均完成能耗相比于其他算法依然存在较大差距;随着节点数量的增加,HTA 虽然也能降低能耗,但总体效果不明显,这是因为尽管节点数量不断增加,而 HTA 依然优先选择执行速度快的节点,从而导致了较高的能耗;GACA 与 GA 具有明显的优势,但平均任务执行能耗优化性能相差不大,甚至 GA 在节点数量较少时表现更好。这是由于 GACA 为了提升网络整体性能,在适应度函数的设计上忽略了一定程度的能耗性能,但是随着系统中 MC 节点设备数量的增加,蚁群算法所具备的精确求解能力对最优任务分配方案求解的影响逐渐增大,GACA 的平均任务完成能耗呈现出明显的下降趋势。

迭代次数是衡量启发式算法性能优劣的重要指标,图 4 为 GA 和 GACA 在相同节点数量和任务设置的情况下平均任务完成时间随着迭代次数逐渐增加的变化曲线对比情况。随着迭代次数的增加,任务的平均完成时间显著降低,而相比于 GA,GACA 性能更佳,收敛速度更快。这是由于 GACA 融合了遗传算法的全局搜索能力和蚁群算法的反馈能力,收敛速度加快,得到更优的任务分配方案,克服了 GA 易获得局部最优解的缺陷,缩短任务分配问题求解时间,取得更好的效果。

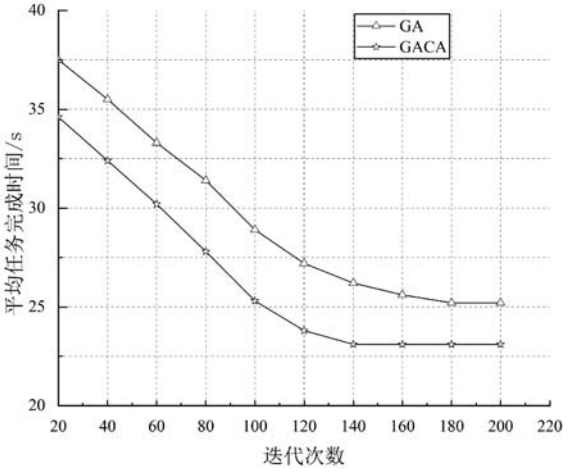

图 4　平均任务完成时间对比

4　结　语

本文研究 ad hoc 云环境下的任务卸载算法,为了有效降低任务完成的时间和能耗,设计基于用户多目标需求的任务卸载决策模型,并选取网络中具有稳定能量和较低移动性的簇头节点作为集中控制器进行卸载决策后的任务分配;提出融合遗传算法和蚁群算法的高效任务卸载算法,综合利用遗传算法的快速搜索能力和蚁群算法的正反馈机制,迅速得到更优的任务分配方案。仿真结果表明,相比于随机任务分配算法、异构感知任务分配算法和遗传算法,本文算法能有效降低任务的平均执行时间和能量消耗,实现高效的任务卸载,从而提升整个网络的性能。

参　考　文　献

[1] Mollah M B, Azad M A K, Vasilakos A. Security and privacy challenges in mobile cloud computing: survey and way ahead [J]. Journal of Network & Computer Applications, 2017, 84:34 - 54.

[2] Kumar K, Lu Y H. Cloud computing for mobile users: can offloading computation save energy? [J]. Computer, 2010, 43(4):51 - 56.

[3] Verbelen T, Simoens P, Turck F D, et al. Cloudlets: bringing the cloud to the mobile user [C]//ACM Workshop on Mobile Cloud Computing and Services, 2012.

[4] Chen M, Hao Y, Li Y, et al. On the computation offloading at ad hoc cloudlet: architecture and service modes [J]. IEEE Communications Magazine, 2015, 53(6):18 - 24.

(下转第 199 页)

chine breakdowns[J]. IEEE Transactions on Cybernetics, 2019,49(1):184 – 197.

[14] Deb K, Abouhawwash M, Seada H. A computationally fast convergence measure and implementation for single-, multiple-, and many-objective optimization[J]. IEEE Transactions on Emerging Topics in Computational Intelligence, 2017, 1(4):280 – 293.

[15] Valdez S I, Botello-Aceves S, Becerra H M, et al. Comparison of a concurrent and a sequential optimization methodology for serial manipulators using metaheuristics[J]. IEEE Transactions on Industrial Informatics, 2018, 14(7):3155 – 3165.

[16] Liang J J, Yue C T, Qu B Y. Multimodal multi-objective optimization: A preliminary study[C]//2016 IEEE Congress on Evolutionary Computation(CEC). IEEE, 2016:2454 – 2461.

[17] Liang J J, Guo Q, Yue C. A self-organizing multi-objective particle swarm optimization algorithm for multimodal multi-objective problems[C]//Advances in Swarm Intelligence. Springer, 2018:550 – 560.

[18] Sengupta S, Basak S, Peters R. Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives[J]. Machine Learning and Knowledge Extraction, 2018, 1(1):157 – 191.

[19] 李俊,罗阳坤,李波. 基于异维变异的差分混合粒子群算法[J]. 计算机科学,2018,45(5):208 – 214.

[20] Gong Y J, Li J J, Zhou Y C. Genetic learning particle swarm optimization[J]. IEEE Transactions on Cybernetics, 2017, 46(10):2277 – 2290.

[21] Huang G W, Cai Y G, Cai H. A time varying constrict factor PSO algorithm research[J]. Journal of Computational Methods in Sciences and Engineering Preprint, 2018, 18(3): 1 – 11.

[22] 张宇航,项铁铭,王建成. 基于维度变化的萤火虫优化算法[J]. 工业控制计算机,2017,30(3):20 – 21.

[23] Li Y P, Ni Z W, Jin F F, et al. Research on clustering method of improved glowworm algorithm based on good-point set[J]. Mathematical Problems in Engineering, 2018, 11(1):1 – 8.

[24] 费腾,张立毅,孙云山. 配送中心选址的自适应 Levy 分布混合变异鱼群算法[J]. 计算机应用与软件,2017,34(1):252 – 257.

[25] Kordestani J K, Firouzjaee H A, Meybodi M R. An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems[J]. Applied Intelligence, 2018, 48(1):97 – 117.

[26] Jiang S W, Zhang J, Ong Y S. A simple and fast hypervolume indicator-based multi-objective evolutionary algorithm[J]. IEEE Transactions on Cybernetics, 2017, 45(10):2202 – 2213.

(上接第 191 页)

[5] Li B, Pei Y, Wu H, et al. Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds[J]. Journal of Supercomputing, 2015, 71(8): 3009 – 3036.

[6] Zhou B, Dastjerdi A V, Calheiros R N, et al. A context sensitive offloading scheme for mobile cloud computing service[C]//2015 IEEE 8th International Conference on Cloud Computing, 2015.

[7] Zhao X, Hung W N N, Yang Y, et al. Optimizing communication in mobile ad hoc network clustering[J]. Computers in Industry, 2013, 64(7):849 – 853.

[8] Guan Z, Melodia T. The value of cooperation: minimizing user costs in multi-broker mobile cloud computing networks[J]. IEEE Transactions on Cloud Computing, 2017, 5(4): 780 – 791.

[9] Liu F, Shu P, Lui J C S. AppATP: an energy conserving adaptive Mobile-Cloud transmission protocol[J]. IEEE Transactions on Computers, 2015, 64(11):3051 – 3063.

[10] Shah-Mansouri H, Wong V W S, Schober R. Joint optimal pricing and task scheduling in mobile cloud computing systems[J]. IEEE Transactions on Wireless Communications, 2017, 16(8):5218 – 5232.

[11] Yaqoob I, Ahmed E, Gani A, et al. Heterogeneity-aware task allocation in mobile ad hoc cloud[J]. IEEE Access, 2017, 5:1779 – 1795.

[12] Zhang Y, Niyato D, Wang P. Offloading in mobile cloudlet systems with intermittent connectivity[J]. IEEE Transactions on Mobile Computing, 2015, 14(12):2516 – 2529.

[13] Guo X, Liu L, Chang Z, et al. Data offloading and task allocation for cloudlet-assisted ad hoc mobile clouds[J]. Wireless Networks, 2018, 24(1):1 – 10.

[14] Zhou B, Dastjerdi A V, Calheiros R N, et al. MCloud: a context-aware offloading framework for heterogeneous mobile cloud[J]. IEEE Transactions on Services Computing, 2017, 10(5):797 – 810.

[15] Shi T, Yang M, Li X, et al. An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds[J]. Pervasive & Mobile Computing, 2016, 27:90 – 105.

[16] Karaoglu B, Heinzelman W. Cooperative load balancing and dynamic channel allocation for cluster-based mobile ad hoc networks[J]. Mobile Computing IEEE Transactions on, 2015, 14(5):951 – 963.

[17] Liu Y G, Cui Q, Zhang M, et al. Cloud computing task scheduling strategy based on genetic algorithm[J]. Information Technology, 2017, 8:177 – 180.

[18] Yousafzai A, Chang V, Gani A, et al. Directory-based incentive management services for ad-hoc mobile clouds[J]. International Journal of Information Management, 2016, 36(6):900 – 906.