

# 边缘计算环境下改进蚁群算法的任务调度算法

张云飞<sup>1</sup>,高岭<sup>1</sup>,丁彩玲<sup>2</sup>,赵辉<sup>2</sup>,金帅<sup>2</sup>,高全力<sup>1</sup>

(1. 西安工程大学 计算机科学学院,陕西 西安 710048;

2. 山东如意毛纺服装集团股份有限公司,山东 济宁 272044)

**摘要:**针对边缘计算环境下边缘节点间资源差距过大且任务分配的负载不均衡等问题,提出了一种基于蚁群优化算法的任务调度方法。方法以不同任务对于CPU、内存、带宽等计算资源的需求情况的差异作为任务选择边缘节点的约束条件,以边缘云达到整体的负载均衡为目标,通过改进启发式因子、信息素的更新等条件提高算法的整体计算效率,降低计算时间,最后通过利用蚁群算法实现任务在边缘环境下的合理分配得出最优分配方式。方法能够避免相同类型的任务部署在同一节点中以提高任务执行效率和运算资源利用率。仿真实验结果表明,该算法在相同的节点数量下可以分配更多的任务,而且相同的任务数量下边缘节点整体具有较低的负载不均衡度,提高了资源的利用率,降低了任务整体的计算时间。

**关键词:**边缘计算;任务调度;负载均衡;蚁群算法;多目标

中图分类号:TP393

文献标识码:A

文章编号:1673-629X(2021)09-0086-06

doi:10.3969/j.issn.1673-629X.2021.09.015

## Improved Task Scheduling Algorithm of Ant Colony Algorithm in Edge Computing

ZHANG Yun-fei<sup>1</sup>,GAO Ling<sup>1</sup>,DING Cai-ling<sup>2</sup>,ZHAO Hui<sup>2</sup>,JING Shuai<sup>2</sup>,GAO Quan-li<sup>1</sup>

(1. School of Computer Science,Xi'an Polytechnic University,Xi'an 710048,China;

2. Shandong Jining Ruyi Woolen Textile Co.,Ltd.,Jining 272044,China)

**Abstract:**In the edge computing environment,the resource gap between edge nodes is too large and the load is unbalanced in task allocation,therefore,a task scheduling method based on ant colony optimization algorithm is proposed. The differences in the demands of different tasks on CPU,memory,bandwidth and other computing resources are taken as the constraints for task selection of edge nodes,and the overall load balance of edge cloud is taken as the goal. The overall computing efficiency of the algorithm is improved and the computing time is reduced by improving the conditions such as heuristic factors and pheromone update. Finally,the optimal allocation method is obtained by using the ant colony algorithm to realize the reasonable allocation of tasks in the edge environment. The method can avoid the deployment of tasks of the same type in the same node to improve task execution efficiency and computing resource utilization. The simulation experiment shows that the proposed algorithm can allocate more tasks under the same number of nodes,and the overall edge nodes have lower load imbalance under the same number of tasks,which improves the utilization of resources and reduces the overall task calculating time.

**Key words:**edge computing;task scheduling;load balance;ant colony algorithm;multi-objective

## 0 引言

由于现有的计算架构难以满足以无人驾驶为代表的移动端应用对于低时延、高数据量的需求,边缘计算作为云计算等现有运算模式的补充被提了出来。边缘计算就是在靠近用户终端的地方设置计算/存储节点

以此来满足移动端用户的需求,可以有效缓解云中心以及网络传输中的压力,提高数据处理的实时性。目前边缘计算中计算和任务的调度、体系结构、安全性等问题是领域内的热点研究方向,软件定义网络<sup>[1]</sup>、区块链<sup>[2]</sup>、机器学习<sup>[3]</sup>等新兴的技术也应用到了边缘计算

收稿日期:2020-12-01

修回日期:2021-04-02

**基金项目:**国家自然科学基金青年项目(61902300);陕西省自然科学基金(2019JQ-850);陕西省重点产业创新链(群)(2020ZDLGY07-05);2019年度山东省重点研发计划重大科技创新工程(厅市联合)项目(2019TSLH0209)

**作者简介:**张云飞(1995-),男,硕士,研究方向为边缘计算;通信作者:高岭(1964-),男,教授,博士,研究方向为边缘计算、智能信息处理;高全力(1988-),男,副教授,博士,研究方向为边缘计算、智能推荐系统。

中,并且边缘计算同样地也在医疗<sup>[4]</sup>、智能城市<sup>[5]</sup>、虚拟现实<sup>[6]</sup>、智慧工业<sup>[7]</sup>、无人驾驶<sup>[8]</sup>、环境监测<sup>[9]</sup>等方面取得了不错的进展。

经过这几年的发展,许多人也在边缘计算的不同领域做出了许多的贡献,为后续的发展提供了强有力的支持。文献[10]总结了边缘计算的背景、概念、发展现状、核心技术以及面临的问题。文献[11]考虑到有效技术传输和处理所需的资源,于是提出了云-边耦合的新式架构,通过边缘层对物联网设备的完整分析管道进行动态管理从而生成大型数据集以达到最终目的。文献[12]则设计了一个轻量级的调度器,它利用深度神经网络来实现数据中心和移动设备的自动分配,通过这种方式来实现云中心和边缘节点之间的数据交换。以上这些工作均是在讨论云边之间的协同与交互问题,但是并没有关注边缘设备间的交互与任务分配问题。

由于边缘计算中数据的异构性、计算和存储资源的局限性,导致利用有限的资源实现对于时延敏感应用的有效支撑是目前亟需解决的关键问题。那么,能否实现任务的合理调度是解决上述问题的思路之一。目前针对任务调度已有部分研究者提出了一些解决方案,具体有:文献[13]使用动态资源调度算法实现边缘端的资源分配;利用资源匹配算法来将云数据中心的资源分配在边缘上,通过二者的协作实现边缘计算的低时延和高服务质量。文献[14]通过改变Storm的Task实例的排序分配方式以及二者的映射关系实现全局的优化调度;同时,针对拓扑任务并发度的缺陷,提出了一种基于蝙蝠算法的调度策略,以满足边缘节点之间的高实时性处理要求。文献[15]是在边缘网络上搭建一个实时数据处理的应用程序,希望通过这种方式减少数据传输量来降低能耗。文献[16]

将数据块的最佳放置和任务的最佳调度相结合,以减少提交任务的计算延迟和响应时间,改善边缘计算用户的体验。以上这些算法从不同的需求角度提高了任务的完成效率,但较少有人关注由于任务调度的不合理导致的计算资源的负载不均衡,使得边缘环境下有限的资源无法得到充分利用,进而提高了用户的响应时间。文中将在蚁群算法的基础上,以任务的资源需求为约束实现边缘端任务分配的负载均衡,让有限的边缘资源发挥最大的功效。

## 1 边缘计算

### 1.1 任务调度

边缘计算的研究现在仍然处于研究的初级阶段,不同应用领域的人都对它的定义有着自己的理解,但是其内容实质已然确定:在靠近数据源的网络边缘存储和处理数据,与云端协作为用户提供低时延、高效的服务。而任务调度便是实现边缘计算十分重要的一个环节,一个优秀的调度算法可以充分利用设备上的计算和存储资源,实现数据传输的最小化和应用程序执行性能的最大化,在有限的资源中为用户创建更优秀的服务效果。

边缘计算的典型任务调度场景如图1所示,可以描述为 $n$ 个相互独立的任务分配到 $m$ 个不同的计算能力的边缘节点上, $n$ 通常大于 $m$ 。其中每个边缘节点的计算资源差距很大而且任务 $n$ 的需求也将呈现不同的状态。根据需要的优化目标,建立任务与边缘节点之间的匹配关系,实现任务分配的最优。

为了简化任务的调度,作如下假设:

- (1) 默认任务之间为相互独立的没有依赖关系;
- (2) 每个任务的需求都是已知的;
- (3) 任务执行中不存在间断执行。

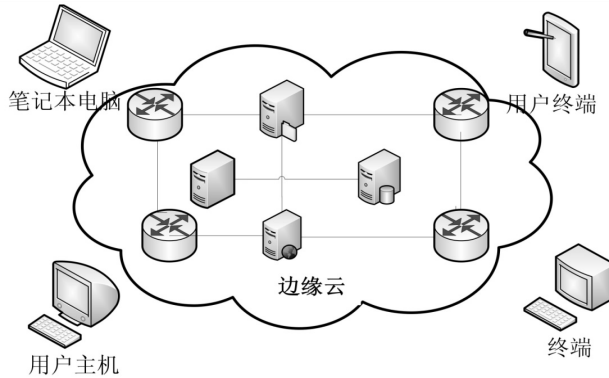


图1 边缘网络调度模型

### 1.2 问题定义

在某一时刻一个边缘节点接收到的任务数为 $n$ ,表示为 $T_N = \{t_{n_1}, t_{n_2}, \dots, t_{n_n}\}$ ;任务 $i$ 对于资源的需求描述为 $(d_{cpu_i}, d_{gpu_i}, d_{mem_i})$ ,其中 $d_{cpu_i}$ 表示该任务对于CPU

的需求, $d_{gpu_i}$ 表示该任务对于GPU的需求, $d_{mem_i}$ 表示该任务对于内存的需求;而该节点需要将这些任务合理分配至其所在的边缘网络当中,边缘网络中可分配的计算节点数量为 $m$ ,表示为 $P_M = \{p_{m_1}, p_{m_2}, \dots, p_{m_m}\}$ ;

节点  $j$  中可用的资源描述为  $(r_{\text{cpu}_j}, r_{\text{gpu}_j}, r_{\text{mem}_j})$ , 其中  $r_{\text{cpu}_j}$  表示该节点 CPU 可用量,  $r_{\text{gpu}_j}$  表示该节点 GPU 的可用量,  $r_{\text{mem}_j}$  表示该节点内存的可用量。

任务  $T_N$  到节点  $P_M$  的分配关系可以用矩阵  $A$  表示为:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad (1)$$

这里的  $a_{ij}$  表示为任务  $T_i$  和节点  $P_j$  的对应关系,  $a_{ij} \in \{0, 1\}$ ,  $i \in \{0, 1, \dots, n\}$ ,  $j \in \{0, 1, \dots, n\}$ , 如果  $a_{ij} = 1$  则表示任务  $i$  分配在了节点  $j$  上。

## 2 基于改进蚁群算法的多目标优化边缘计算任务调度

文中是在标注蚁群算法的基础上, 对于信息素的初始化、更新, 以及启发因子的设置等进行优化改进, 以达到目标任务以及适应边缘计算环境下的要求。

### 2.1 定义问题解

文中对边缘节点的负载均衡性采用负载不均衡度来度量, 该值在  $0 \sim 1$  之间取值, 该值越小则表示边缘节点的任务的负载分布越均匀, 系统的整体性能越高。具体表示为:

$$N = \frac{1}{n} \sum_{i=1}^n |U_{\text{no}_i^{\text{CPU}}} - U_{\text{avg}_i^{\text{CPU}}}| + \frac{1}{n} \sum_{i=1}^n |U_{\text{no}_i^{\text{GPU}}} - U_{\text{avg}_i^{\text{GPU}}}| + \frac{1}{n} \sum_{i=1}^n |U_{\text{no}_i^{\text{MEM}}} - U_{\text{avg}_i^{\text{MEM}}}| \quad (2)$$

其中,  $U_{\text{no}_i^{\text{CPU}}}$ 、 $U_{\text{no}_i^{\text{GPU}}}$ 、 $U_{\text{no}_i^{\text{MEM}}}$  分别表示边缘节点  $P_{M_i}$  ( $i = 1, 2, \dots, n$ ) 对于 CPU、GPU、内存的使用率;  $U_{\text{avg}_i^{\text{CPU}}}$ 、 $U_{\text{avg}_i^{\text{GPU}}}$ 、 $U_{\text{avg}_i^{\text{MEM}}}$  分别表示整体边缘节点对于 CPU、GPU、内存的平均使用率。

### 2.2 启发式因子的设置

$\eta_{ij}$  表示的是启发式因子, 主要表示的是任务  $i$  布置在边缘节点  $j$  的期望强度,  $\eta_{ij}$  的值越大则任务布置在该节点的可能性也越大:

$$\eta_{ij} = Q/M_{D_{ij(0)}} \quad (3)$$

其中,  $Q$  为任意常数 (建议选择值在 1 附近选择),  $M_{D_{ij(0)}}$  则可以定义为等待分配的任务对于所需求的资源与节点空闲资源之间的余弦相似度, 相当于是任务  $i$  与节点  $j$  之间的相似程度用二者之间的夹角表示。夹角越小, 则代表着二者之间的相似程度越高, 任务分配在该节点的可能性也越高:

$$M_{D_{ij}} = \frac{\sum_{a=1}^A (S_i^a \times Q_j^a)}{\sqrt{\sum_{a=1}^A (S_i^a)^2} + \sqrt{\sum_{a=1}^A (Q_j^a)^2}} \quad (4)$$

其中,  $A$  表示边缘节点可以为任务提供的资源种类的个数 (文中以 CPU、GPU、内存三种资源为需求资源);  $S_i^a$  为边缘节点  $P_{M_i}$  的第  $a$  种资源的空闲量;  $Q_j^a$  为任务  $T_{N_j}$  的第  $a$  种资源的需求量。

根据公式 (4) 可以得出: 待分配任务与边缘节点之间的匹配度越小, 节点越能适应任务对于节点的性能要求, 该任务分配在此节点的可能性也应该越高。在任务的分配过程中, 匹配度也在随着时间的变化在不停的变化, 令  $W_{D_i}(t)$  表示时刻  $t$  边缘节点  $P_{M_i}$  与任务  $T_{N_j}$  的匹配度, 如果任务  $T_{N_j}$  放置在边缘节点  $P_{M_i}$  上, 则在下次边缘节点  $P_{M_i}$  与其他任务的匹配度计算要在分配了任务  $T_{N_j}$  所需资源后的空闲资源进行计算。

任务与节点之间的分配为多对一的映射关系, 在任务的分配中需要保证任务对于每一种资源的需求量都应该小于节点空闲的资源量, 通过对比任务对 CPU、GPU 和内存的需求和节点可利用资源进行对比防止将任务分配在资源不足的节点上面, 即:

$$\sum_i v_{\text{cpu}_i} < r_{\text{cpu}_i} \quad (5)$$

$$\sum_i v_{\text{gpu}_i} < r_{\text{gpu}_i} \quad (6)$$

$$\sum_i v_{\text{mem}_i} < r_{\text{mem}_i} \quad (7)$$

其中,  $v_{\text{cpu}_i}$ 、 $v_{\text{gpu}_i}$ 、 $v_{\text{mem}_i}$  分别表示任务  $T_{N_j}$  在边缘节点  $P_{M_i}$  上 CPU、GPU、内存的使用量;  $r_{\text{cpu}_i}$ 、 $r_{\text{gpu}_i}$ 、 $r_{\text{mem}_i}$  分别表示边缘节点  $P_{M_i}$  关于 CPU、GPU、内存的可使用量。

### 2.3 信息素的更新

初始蚁群算法的信息素更新对于全部的路径挥发, 这样会导致经常未走过的节点的信息素会越来越低甚至趋近于零。所以, 文中对于没有走过的路径不进行挥发, 具体更改如下:

$$\tau_{ij}(\text{new}) = \begin{cases} (1 - \rho)\tau_{ij}^i(t) + \rho\Delta\tau(t), & \text{if } \Delta\tau_{ij} \neq 0 \\ \tau_{ij}^i(t), & \text{else} \end{cases} \quad (8)$$

其中,  $\rho$  表示信息素的挥发因子,  $1 - \rho$  表示信息素的残留因子。  $\Delta\tau_{ij} = \sum_{k=1}^m \tau_{ij}^k$ , 若  $\Delta\tau_{ij} = 0$  则表示该任务没有分配在该节点上面。

### 2.4 算法总体流程

输入: 边缘节点集合  $P_M = \{P_{m_1}, P_{m_2}, \dots, P_{m_n}\}$ , 任务需求集合  $T_N = \{t_{n_1}, t_{n_2}, \dots, t_{n_n}\}$ , 信息素启发因子  $\alpha$ , 期望启发因子  $\beta$ , 信息素挥发率  $\rho$  等初始需求。

输出: 最优的任务分配方案和负载不均衡度。

Step1: 初始化蚂蚁个数 AntNum 并且按照任务提

的顺序将任务的需求资源附加在每一个任务上,迭代次数  $\maxIter$ , 启发式因子  $\alpha$  和  $\beta$  以及相关参数。

Step2: 随机将  $n$  个携带着任务需求的蚂蚁分配在随机节点上面(因为不同任务可以放置在同一节点上,所以禁忌表  $Tabu_k$  仅为不满足公式(5)到公式(7)的节点), 计算第  $k$  只蚂蚁将任务  $i$  分配在节点  $j$  的概率。之后利用轮盘赌的方式从满足条件的节点中随机选择一个节点作为该任务的分配节点, 并将该任务部署在该节点上。概率选择公式为:

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{k \in allowed_k} [\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta} & \text{if } i \in allowed_k \\ 0, & \text{else} \end{cases} \quad (9)$$

其中,  $p_{ij}^k(t)$  表示第  $k$  只蚂蚁中任务  $i$  选择节点  $j$  的概率,  $\tau_{ij}(t)$  表示路径  $(i, j)$  的信息素浓度,  $\eta_{ij}(t)$  表示路径  $(i, j)$  的启发式因子, 如公式(3)。allowed<sub>k</sub> 为任务  $i$  的可分配列表。

Step3: 第  $k$  只蚂蚁完成所有的任务分配之后, 对所分配的节点按照公式(6)进行局部更新。

Step4: 完成该蚂蚁的部署之后按照公式(2)计算该分配方案的负载不均衡度, 并与历史记录对比并记录最优分配方案和最小负载不均衡度。

Step5: 判断所有的蚂蚁是否全部结束, 如果有蚂蚁没有完成则跳至 Step2; 如果所有的蚂蚁均完成了本次迭代, 计算出全局最优解并保存, 最后按照最优的放置方案根据公式(8)进行信息素的全局更新。

Step6: 判断是否满足迭代次数或者满足了预先设置的负载均衡度, 算法迭代结束返回最优路径解。否则, 返回 Step2。

### 3 实验分析

#### 3.1 仿真实验参数配置

为了验证算法的有效性, 进行仿真实验。利用轮询算法(PA)、最大内存分配算法(MMA)以及文中算法(IAC)进行比较, 在一台计算机(四核, 2.5 GHz 的 CPU 主频, 8.0 GB 的内存)上采用 python 语言编程进行模拟仿真, 验证算法的性能。将任务分配后边缘节点的整体负载不均衡度(公式(2))以及该边缘云的最大承载任务数作为最终判断算法优劣的依据。

为了使硬件设备的计算和存储能力有更好可视性, 所以对硬件和任务需求的数据进行了数字化描述。由于边缘节点的异构性, 所以 CPU 与 GPU 的处理能力在 100 ~ 200 间随机取得, 内存的存储能力也在 300 ~ 600 间随机取得; 根据文献[17]的流量模型, 其中假设任务的 15% 为大型任务, 每个任务的 CPU、GPU 需求量在 30 ~ 70 之间随机取得, 内存需求也在

50 ~ 200 间随机取得; 小型任务的数量在 85%, 每个任务的 CPU、GPU 需求量在 1 ~ 30 之间随机取得, 内存需求也在 3 ~ 50 间随机取得。

由于蚁群算法的参数对算法的性能有着较大的影响, 因此对算法的各种参数组合进行实验并对比。采用控制变量法, 分别对  $\alpha$ 、 $\beta$ 、 $\rho$  设定数值, 设置任务数量为 200(均为大型任务, 可以使得对比变化更明显), 边缘节点数为 100, 迭代次数为 50, 取 10 次平均结果作为最终结果。

(1) 保持  $\rho$  的数值不变, 分别对  $\alpha$  和  $\beta$  进行取值, 具体的影响结果如表 1 所示。 $\alpha$  越大, 蚂蚁在某个点容易陷入局部最短路径, 虽然收敛速度会增大, 但是搜索过程中随机性减弱, 易陷入局部最优;  $\beta$  越大, 蚂蚁越容易选择原有路径, 搜索的随机性减少, 也会陷入局部最优。由表 1 可以看出, 在  $\rho$  不变的情况下,  $\alpha = 4$ ,  $\beta = 5$  时算法的性能最优, 负载不均衡度也最小。

表 1  $\alpha$ 、 $\beta$  对算法的影响

$\alpha$	$\beta$	负载不均衡度
0.7	1	0.380 1
0.8	1	0.388 5
1	2	0.362 2
2	3	0.367 5
3	5	0.357 6
4	5	0.349 5
5	7	0.360 7

(2) 保持  $\alpha$  和  $\beta$  不变, 对  $\rho$  进行取值, 影响结果如表 2 所示。由表 2 可以看出,  $\rho$  的最优选择分别在 0.2 和 0.6 的周围, 此时的算法比较优秀。但是  $\rho = 0.2$  在计算过程中的运算时间大于  $\rho = 0.6$ , 所以最终选择  $\rho = 0.6$ 。

表 2  $\rho$  对算法性能的影响

$\rho$	负载不均衡度
0.05	0.381
0.1	0.378
0.2	0.355
0.4	0.366
0.5	0.36
0.6	0.353
0.7	0.371
0.8	0.362

#### 3.2 实验结果与分析

为了验证算法可以较为优秀的对边缘节点做到负载均衡, 分别对文献[18]的轮询算法(RR)、传统蚁群



算法(ACO)以及文中算法(IAC)进行比较,得到最后全部为大型任务、文献[17]的混合任务以及全部为小型任务的最大承载任务数,见图2~图4,以及相同任务数量上三种算法的负载均衡度的对比,见图5。

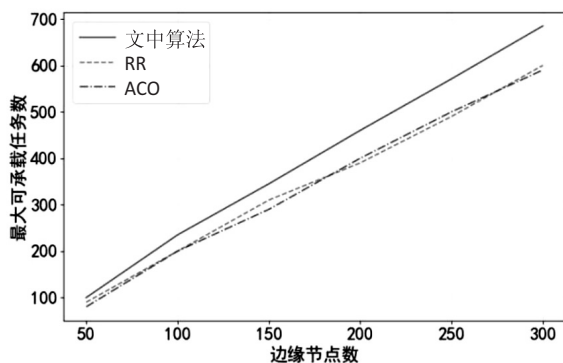


图2 大型任务时各算法最大承载任务数对比

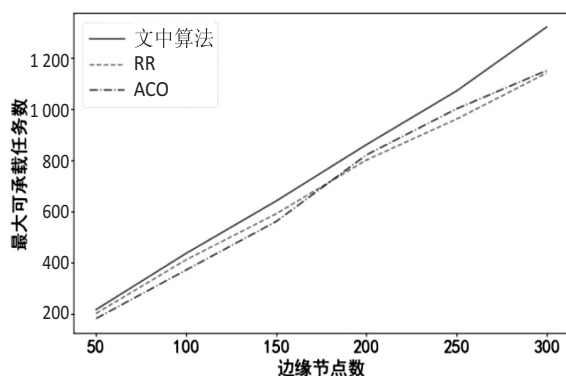


图3 混合任务时各算法最大承载任务数对比

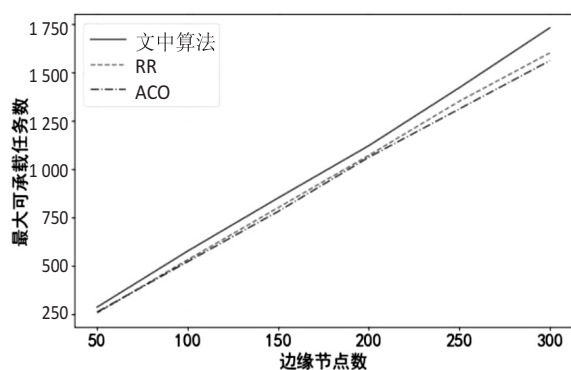


图4 小型任务时各算法最大承载任务数对比

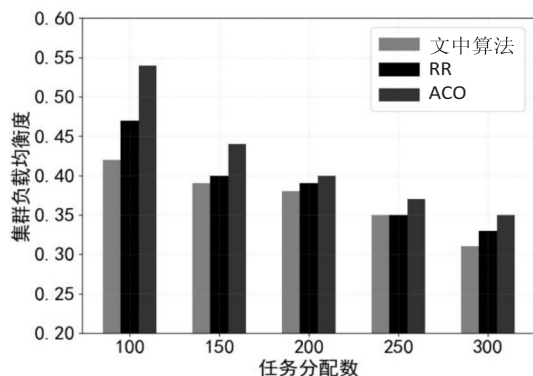


图5 不同任务数时各算法负载均衡度对比

由上面三幅图均可以明显看出,文中提出的算法相对其他两种算法提高了最大任务可承载数,提高了边缘节点的使用效率,从而降低了网络中心总流量,降低了对用户的响应需求的平均响应时间;而且IAC算法每次会有多个近似最优解生成,可以根据需求自由选择。通过三个图的对比可以看出,全部为小型任务时各算法差距很小,当加入大型任务时可以看出文中提出的算法更加优秀。同时若是对负载需求不高的情景也可以通过减少蚂蚁数的方式提高分配效率,降低算法的运行时间。

由图5可知,在边缘节点数不变的情况下,文中所提出的IAC算法的负载均衡度在随着任务数的增长过程中均优于剩余的两种算法,且优秀程度也有着较为明显的差距。这是由于随着任务数量的增多在分配过程中分配的可能性也就越多,也更有可能出现优秀的分配方式,所以算法的整体负载均衡度呈现一个下降的趋势。综上可以得出结论,IAC算法无论是从负载均衡度还是任务的分配数上相对于剩余两种算法均有一个良好的表现。

## 4 结束语

边缘节点的负载均衡是边缘计算下一个十分重要的问题。目前方法在资源分配方面主要以时延为准则忽略了边缘环境的局限性,文中针对边缘环境下资源的局限性,首先对边缘节点的任务分配问题进行描述;然后设计了一种改进的蚁群算法用来解决该问题。该算法将任务对资源的需求作为启发因子,改变了信息素更新方式,加快了收敛速度,提高了最终解的质量。通过仿真实验结果表明,算法可以有效降低节点的负载不均程度。

## 参考文献:

- [1] SON Jungmin, HE Tianzhang, RAJKUMAR B. CloudSimSDN - NFV: modeling and simulation of network function virtualization and service function chaining in edge computing environments [J]. Software: Practice and Experience, 2019, 49(12): 1748-1764.
- [2] ZHEN Yan, LIU Hanyong. Distributed privacy protection strategy for MEC enhanced wireless body area networks [J]. Digital Communications and Networks, 2020, 6(2): 229-237.
- [3] TANG Yayuan, GUO Kehua, MA Jianhua, et al. A smart caching mechanism for mobile multimedia in information centric networking with edge computing [J]. Future Generation Computer Systems, 2019, 91: 590-600.
- [4] GRECO L, PERCANNELLA G, RITROVATO P, et al. Trends in IoT based solutions for health care: moving AI to

- the edge[J]. Pattern Recognition Letters, 2020, 135: 346–353.
- [5] LORIMER P A K, DIEC V M F, KANTARCI B. COVERS-UP: collaborative verification of smart user profiles for social sustainability of smart cities[J]. Sustainable Cities and Society, 2018, 38: 348–358.
- [6] YANG Yousung, LEE J, KIM N, et al. Social-viewpoint adaptive caching scheme with clustering for virtual reality streaming in an edge computing platform[J]. Future Generation Computer Systems, 2020, 108: 424–431.
- [7] CARVALHO A, O' MAHONY N, KRPALKOVA L, et al. Edge computing applied to industrial machines[J]. Procedia Manufacturing, 2019, 38: 178–185.
- [8] LIU Luning, LU Zhaoming, WANG Luhan, et al. Large-volume data dissemination for cellular-assisted automated driving with edge intelligence[J]. Journal of Network and Computer Applications, 2020, 155: 102535.
- [9] CHENG Chen, ZHOU Hui, CHAI Xuchao, et al. Adoption of image surface parameters under moving edge computing in the construction of mountain fire warning method[J]. PLOS ONE, 2020, 15(5): 1–16.
- [10] 施巍松, 张星洲, 王一帆, 等. 边缘计算: 现状与展望[J]. 计算机研究与发展, 2019, 56(1): 69–89.
- [11] VERDERAME L, MERELLI I, MORGANTI L, et al. A secure cloud-edges computing architecture for metagenomics analysis[J]. Future Generation Computer Systems, 2020, 111: 919–930.
- [12] KANG Y, HAUSWALD J, GAO C, et al. Neurosurgeon: collaborative intelligence between the cloud and mobile edge[J]. ACM SIGPLAN Notices, 2017, 52(4): 615–629.
- [13] TANG Hengliang, LI Chunlin, BAI Jingpan, et al. Dynamic resource allocation strategy for latency-critical and computation-intensive applications in cloud-edge environment[J]. Computer Communications, 2019, 134: 70–82.
- [14] 简铮峰, 平靖, 张美玉. 面向边缘计算的 Storm 边缘节点调度优化方法[J]. 计算机科学, 2020, 47(5): 277–283.
- [15] ZAHAF H E, BENYAMINA A E H, OLEJNIK R, et al. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms[J]. Journal of Systems Architecture, 2017, 74: 46–60.
- [16] LI Chunlin, BAI Jingpan, TANG Jianghang. Joint optimization of data placement and scheduling for improving user experience in edge computing[J]. Journal of Parallel and Distributed Computing, 2018, 125: 93–105.
- [17] BENSON T, AKELLA A, MALTZ D A. Network traffic characteristics of data centers in the world[C]//Proc of the 10th ACM SIGCOMM conference on internet measurement. Melbourne, AU: ACM, 2010: 267–280.
- [18] TAN T, KIDDLE C. An assessment of eucalyptus version 1.4[R]. Calgary: University of Calgary, 2009.
- +++++
- (上接第85页)
- ed Abstracts, 2000, 19(1): 798–801.
- [7] 罗宇平. 基于 Min-Min 改进后的网格调度算法[J]. 微电子学与计算机, 2009, 26(3): 86–88.
- [8] 马丽, 刘高原. 一种改进型 Min-Min 调度算法[J]. 计算机工程与应用, 2012, 48(16): 69–73.
- [9] 樊程, 苏若凡. 基于负载均衡的任务调度优化算法[J]. 计算机工程与设计, 2017, 38(6): 1532–1535.
- [10] 王观玉. 网格计算中任务调度算法的研究和改进[J]. 计算机工程与科学, 2011, 33(10): 186–190.
- [11] 阳王东, 王昊天, 张宇峰, 等. 异构混合并行计算综述[J]. 计算机科学, 2020, 47(8): 5–16.
- [12] 王宇耕, 肖鹏, 张力, 等. 基于负载预测的自适应权值负载均衡算法[J]. 计算机工程与设计, 2019, 40(4): 1033–1037.
- [13] 方娟, 章佳兴. 基于负载均衡的 CPU-GPU 异构计算平台任务调度策略[J]. 北京工业大学学报, 2020, 46(7): 782–787.
- [14] KHAWATREH S A. An efficient algorithm for load balancing in multiprocessor systems[J]. International Journal of Advanced Computer Science and Applications, 2018, 9(3): 160–164.
- [15] KUROWSKI K, OLEKSIK A, PIATEK W, et al. Impact of urgent computing on resource management policies, schedules and resources utilization[J]. Procedia Computer Science, 2012, 9: 1713–1722.