文章编号:1671 - 6833(2017)04 - 0051 - 06

基于概率自适应蚁群算法的云任务调度方法

王俊英[1,2], 颜芬芬[1,2], 陈 鹏[1], 董方敏[1,2], 臧兆祥[1]

(1. 三峡大学 计算机与信息学院,湖北 宜昌 443002; 2. 三峡大学 湖北省水电工程智能视觉监测重点实验室,湖北 宜昌 443002)

摘 要: 针对基本蚁群算法在求解云任务调度问题时易陷入局部最优的不足,提出一种任务分配概率自适应的蚁群算法. 算法根据任务量的大小对任务进行降序排序. 定义了任务分配集中度,引入了概率自适应调整因子对任务分配过于集中的资源节点的分配概率进行调整. 结果表明,相对基本蚁群算法及改进蚁群算法,该算法有效地缩短了任务完成时间,且算法的执行效率、收敛速度均有一定程度的改善.

关键词: 云计算;任务调度;蚁群算法;概率自适应

中图分类号: TP391 **文献标志码**: A doi:10.13705/j.issn.1671 - 6833.2017.01.018

0 引言

云计算是将计算任务分布在大量计算机构成的数据中心,通过网络、硬件及系统软件等为用户提供服务[1]. 云计算模式涵盖范围非常广泛,从底层的软、硬资源聚集管理到虚拟化计算池,乃至通过网络提供各类计算的服务. 它将带来生活、生产方式和商业模式的深刻改变,已成为当前社会关注的热点. 任务调度是云计算研究的核心问题之一,对云计算中心的整体性能发挥影响很大[2]. 由于云任务调度是一个 N-P 难题,对于用户提交的任务,云任务调度器可根据用户对计算、存储和网络等的特定需求将资源动态划分给用户,仅采用简单的分配调度方法,如轮转算法、加权轮转法和最小负载优先法等,很难满足用户及云计算中心性能要求[3]. 近些年仿生智能算法备受各界学者的关注,已在连续空间约束优化[4]、任务分配[5]、图形分割[6]等问题上得到广泛应用. 20 世纪 90 年代意大利学者 Dorigo 通过模拟蚂蚁的群体行为提出蚁群算法(ant colony optimization, ACO)[7]. 由于其正反馈、分布式并行、鲁棒性及可扩展性等特点,在解决组合优化问题上,具有较大优势[8]. 为进一步提高 ACO 算法性能,文献[9]结合遗传算法全局收敛的特点,将遗传算法融入到蚁群算法中,以避免算法局部收敛;文献[10]提出自适应路径选择和信息素更新的蚁群算法,提高了算法的全局搜索能力及收敛速度;文献[11]改进算法局部更新策略,提出一种高效鲁棒的蚁群算法;文献[12]提出遗传蚁群算法,以遗传算法随机产生的二进制编码个体,对蚁群算法的路径选择概率进行调整. 然而,这些算法在实际运行过程中,仍有一定的局限性,解的质量也有待进一步提高. 针对上述问题,笔者深入研究了蚁群算法机制,分析了云环境的特殊性质,提出基于概率自适应蚁群算法(probability adaptive ant colony optimization, PAACO)的云任务调度方法.

1 云计算任务调度问题描述

笔者对云任务调度为将 m 个相互独立的任务分配到 n 个异构资源节点上,使得任务完成时间较短. 为深入分析问题,建立如下云任务调度模型.

定义 1 设任务集合 $TASK = \{T_1, T_2, \cdots, T_m\}$,表示在当前时刻任务队列共有 m 个相互独立的任务,任务大小用百万指令(million instructions, MI)表示,一个任务只能在一个资源节点上处理.

定义 2 由于云任务粒度越小对总任务的完

收稿日期:2016 - 11 - 19;**修订日期**:2016 - 12 - 30

基金项目:国家自然科学基金资助项目(61272236、61272237、61502274);湖北省自然科学基金资助项目(2015CFB336);三峡大学人才科研启动基金项目(KJ2011B011、KJ2013B064)

作者简介:王俊英(1971—),女,湖北黄梅人,三峡大学副教授,博士,主要从事人工智能与模式识别方面的研究,E-mail: wannjy@qq.com.

成时间程度影响越小,笔者根据任务大小对其按降序排列,以此序列作为任务调度顺序.

定义3 云计算中心的可用资源包括服务器、普通PC及各种处理单元.假设云计算中有n个计算资源,表示为$RE=\{R_1,R_2,\cdots,R_n\}$,处理能力用百万指令每秒(million instructions per second, MPIS)表示.设定任务数量大于资源数量,即$m>n$,更符合云计算中心调度任务的实际情况.

定义4 任务在资源节点上预期完成时间集合用矩阵$\boldsymbol{ETC}$表示,$\boldsymbol{ETC}=\begin{bmatrix} etc_{11} & \cdots & etc_{1n} \\ \vdots & \ddots & \vdots \\ etc_{m1} & \cdots & etc_{mn} \end{bmatrix}$,通过$etc_{ij}=S_{T_i}/P_{R_j}$估计任务完成时间,$S_{T_i}$表示第$i$个任务的大小,$P_{R_j}$表示第$j$个资源节点的处理能力.

设定矩阵$\boldsymbol{X}$的大小为$m\times n$,表示任务与资源的映射关系,即一种任务分配方案,x_{ij}为$\boldsymbol{X}$中元素,$x_{ij}\in\{0,1\}$,若第i个任务分配到第j个资源节点上,x_{ij}为1,否则为0.将m个任务分配到n个资源上,任务的总完成时间:$F_{\text{time}}=\max\limits_{j=1}^{n}\left(\sum\limits_{i=1}^{m}x_{ij}\cdot etc_{ij}\right)$,优化目标:$\min F_{\text{time}}$.

2 基于蚁群算法的云任务调度

蚁群算法的实质为蚂蚁通过个体之间的信息素交流达到合作.笔者以经典的TSP为例,说明运用蚁群算法对云任务调度的过程.

构建图$G=(C,L)$来描述平面上的n个城市节点,C为城市集合,L表示城市节点间的距离集合.假设有m只蚂蚁,第k只蚂蚁已经访问过的城市节点存放于$Tabu_k$禁忌表中,$Allowed_k=\{C-Tabu_k\}$,表示第k只蚂蚁未到达的城市节点集合.d_{ij}表示城市i与城市j之间的距离,$\eta_{ij}=1/d_{ij}$,表示从城市i到城市j的启发式期望程度,τ_{ij}表示在t时刻路径(i,j)上信息素残留量.在初始时刻,令各路径上信息素含量$\tau_{ij}(0)=w$,w为常数,则在t时刻,第k只蚂蚁从城市i转移到城市j的概率为:

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}{\sum\limits_{j=1}^{n}[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}, j\in Allowed_k;\\ 0,\text{else},\end{cases}\tag{1}$$

式中:α为信息素影响力因子,反映了路径累积的信息素含量对蚂蚁路径选择的影响程度;β为启发信息影响因子,反映了蚂蚁在行进过程中,启发信息对路径选择的影响程度.通常取$\alpha=1$,$\beta=2\sim5$.为避免信息素无限累积,采用如下方式对信息素更新:

$$\tau_{ij}(t+1)=(1-\rho)\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij}^{k},\tag{2}$$

$$\Delta\tau_{ij}^{k}=\begin{cases}\dfrac{Q}{L_k},\text{边}(i,j)\text{在第}k\text{只蚂蚁构建的路径}T_k\text{上};\\ 0,\text{else}\end{cases}\tag{3}$$

式(2)表示在下一时刻,路径(i,j)上信息素浓度,$\rho\in(0,1]$表示信息素挥发因子,通常取$\rho=0.5$;$\Delta\tau_{ij}^{k}$表示第k只蚂蚁向边(i,j)释放的信息素量.式(3)中,Q为常数,表示信息素强度,通常取$Q=1$;L_k表示路径T_k的长度[13].

由于云计算环境的独特性,在运用基本蚁群算法解决云任务调度问题时,还需将TSP问题与云任务调度问题进行差异性分析.两者的差异性为:

1)在TSP问题中,对于蚂蚁到达过的城市节点,蚂蚁不能重复访问;云环境中,对于已经被分配过的任务不能再次被分配.

2)在TSP问题中,蚂蚁访问城市节点的总距离影响信息素更新;云环境中,通过资源与任务的较优组合所得的相关系数表示信息素浓度.

3)在云计算环境中,资源对任务的启发信息用资源的能见度表示,资源能见度包括资源的计算能力、通信能力、存储能力等相关参数.

3 基于PAACO算法的云任务调度

ACO算法具有正反馈、分布式并行的特点,但易陷入局部最优.在运用ACO算法对任务进行调度时,蚂蚁根据任务与资源节点间信息素浓度及资源对任务的启发信息来分配任务,若信息素含量过高,任务分配到资源节点的集中程度则过大,可能使算法因过强的正反馈信息陷入局部最优.

为避免算法局部收敛,当算法累计多次未更新最优解,则认为算法已陷入局部最优,为使算法跳出局部最优解,笔者定义任务分配集中度概念,引入任务分配概率自适应调整因子,对任务分配集中度过高的任务分配概率进行调整,避免算法由于过强的正反馈信息导致局部收敛.

通过分析可知,由于算法在迭代后期,将逐渐趋于收敛,此时任务分配集中程度已相对较高,若再引入概率自适应调整因子对任务分配概率进行调整,将导致算法收敛速度慢甚至无法收敛,因此本文PAACO算法中的任务分配概率自适应调整方法仅在迭代前期实施.

3.1　任务分配集中度

笔者首先定义任务分配集中度的概念. 假设在上一代已生成最优路径矩阵中,对于任务 i,每只蚂蚁遍历各个资源节点对任务进行一次分配,设任务 i 分配到资源节点 $j(j=1,2,\cdots,n)$ 对应的蚂蚁个数依次为 $\theta_{i1},\theta_{i2},\cdots,\theta_{ij},\cdots,\theta_{in}$,即有 θ_{ij} 只蚂蚁将任务 i 分配到资源节点 j 上,蚂蚁总数量为 m',则任务分配集中度定义为:

$$d_i=\max_{j=1}^{n}\left(\frac{\theta_{ij}}{m'}\right), \tag{4}$$

即 i 任务与资源节点映射关系 (T_i,R_j) 中对应的蚂蚁数量与总蚂蚁数量比值的最大值.

3.2　任务分配概率自适应调整方法

PAACO 算法中通过判断任务分配集中程度,引入任务分配概率自适应调整因子对大于任务分配集中度阈值的任务分配概率进行调整,任务分配概率自适应调整因子定义为:

$$T_{ij}=\begin{cases}c\cdot e^{1-\theta_{ij}/m'}, d_i>\mu;\\ 1,\text{else.}\end{cases} \tag{5}$$

其中,μ 表示任务 i 分配到资源节点的集中度阈值,$\mu\in(0,1)$;c 为常数,用来控制调整因子变化的幅度. 在上一代生成的最优路径矩阵中,对于任务与资源映射关系 (T_i,R_j) 中对应的蚂蚁数量 θ_{ij} 越大,表示蚂蚁对任务 i 分配时越集中于资源节点 j 上,则调整因子 T_{ij} 值越小,反之,调整因子值越大;θ_{ij}/m' 表示映射关系 (T_i,R_j) 中对应蚂蚁个数相对总蚂蚁数量的比值. 笔者通过任务分配概率自适应调整因子对任务分配概率进行调整,降低映射关系 (T_i,R_j) 中对应蚂蚁数量相对总蚂蚁数量较高的任务分配概率,增大蚂蚁数量相对总蚂蚁数量较低的任务分配概率,从而达到降低任务分配集中度的效果,提高算法的搜索范围,加快算法的收敛速度.

引入任务分配概率自适应调整因子后的任务分配概率 p_{ij}^k 表示为:

$$p_{ij}^k(t)=\begin{cases}\dfrac{T_{ij}\cdot[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}{\sum_{j=1}^{n}T_{ij}\cdot[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}, i\in AllowedTask_k;\\ 0,\text{else.}\end{cases} \tag{6}$$

其中,$\eta_{ij}(t)$ 为启发信息,表示任务被分配到资源节点的期望程度,令 $\eta_{ij}(t)=P_{R_j}$;α、β 分别为信息素影响力因子和资源固有属性影响力因子;$AllowedTask_k$ 表示可分配任务列表,对于已经被分配的任务存放于禁忌表 $Tabu_k$ 中.

运用 PAACO 算法对任务进行分配时,调整信息素更新公式为:

$$\tau_{ij}(t+1)=(1-\rho)\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij}^k, \tag{7}$$

$$\Delta\tau_{ij}^k=\begin{cases}\dfrac{Q}{F_{\text{time}}^k}, x_{i,j}^k=1;\\ 0,\text{else.}\end{cases} \tag{8}$$

式中:F_{time}^k 表示第 k 种分配方案的所得任务总完成时间;$x_{i,j}^k=1$ 表示第 k 种方案中第 i 个任务分配被分配到第 j 个资源节点上.

3.3　基于 PAACO 算法的任务调度流程

根据 PAACO 算法建立的任务调度模型,算法输入:任务集合 $TASK$,资源集合 RE,蚁群规模 m',启发信息影响因子 α,信息素影响因子 β,信息素挥发因子 ρ 等参数,算法输出:一个大小为 $m\times n$ 的矩阵,表示任务的分配方案. 蚂蚁根据启发信息及路径信息素含量,在任务和资源节点构成的网状结构上爬行,当所有的任务分配完成,则表示爬行结束. 运用 PAACO 算法进行云任务调度的具体步骤如下:

步骤 1　初始化信息素矩阵,定义任务分配概率自适应调整方法终止代数为 ND,$ND\leqslant\sigma\cdot NC\_\min$,$(0<\sigma\leqslant 0.5)$,设定算法停滞阈值 $\varepsilon=F_{\text{time}}^{NC-\lambda}-F_{\text{time}}^{NC}$,当迭代次数超过最小迭代次数,且连续迭代 λ 次更新的任务总完成时间差小于阈值 ε,则终止迭代;

步骤 2　定义可分配任务列表 $AllowedTask$、禁忌表 $Tabu$,给 m' 只蚂蚁分配任务列表;

步骤 3　判断当前迭代次数 NC 是否小于 ND,是,则执行步骤 4,否,则执行步骤 5;

步骤 4　判断算法是否陷入局部最优,是,则根据任务分配度集中度阈值引入任务分配概率自适应调整因子,根据式(6)及轮盘赌策略对任务 i 分配资源;否,则执行步骤 5;

步骤 5　不引入动态调整因子,根据任务分配概率及轮盘赌策略对任务 i 分配资源节点;

步骤 6　更新信息素及最优解 $bestF$,令 $bestF=\min(F_{\text{time}}^k)$,更新当前迭代次数 NC;

步骤 7　判断是否达到终止条件,是,则统计数据,输出结果,否,则执行步骤 2.

4 仿真结果与分析

4.1 实验设计

CloudSim[14]仿真平台由澳大利亚墨尔本大学网格实验室提出,平台通过DatacenterBroker模拟了一个代理,用户通过在该类中添加自定义算法来实现云任务调度仿真.笔者搭建了Cloudsim-3.0仿真平台,在DataCenterBroker类中构造PAACO任务调度算法,在该平台中创建Task-Scheduling.java类,调用Cloudlet类及Vm类实现对任务属性及资源属性的定义,调用编写的任务调度算法实现对云任务调度的模拟仿真.取8组实验数据对算法进行检验,设定任务及资源属性参数如表1所示.

表1 任务及资源参数设置

Tab.1 Task and resource parameter settings

参数名	参数值
任务长度	$1\times10^6\sim1\times10^8$ MI
任务数量	$250\times N$,N为正整数且$N\in\{1,2,\cdots,8\}$
资源数量	50
资源性能	$1\times10^3\sim1\times10^4$ MIPS

4.2 实验分析

为保证实验的有效性,实验结果为算法运行20次所得数据的平均值.试验中蚁群数量取10,参数α和β在常用范围内取值,运行PAACO算法及基本蚁群算法(ACO),可得两种算法均在$\alpha=1$、$\beta=2$时所得实验效果最好;因笔者不讨论信息素挥发因子对算法的影响,故取中间值$\rho=0.5$;通过多次实验可见,算法收敛速度较快,实验过程经过30次迭代后逐渐收敛,因此将迭代次数设定为30,算法停滞阈值$\varepsilon=10$.本文算法主要通过设定概率自适应调整方法终止代数占算法执行最小代数的比例σ及任务集中度阈值μ,以实现算法最优.

为逐一验证任务排序方法及概率自适应调整方法的有效性,笔者将仅采用概率自适应调整的蚁群算法命名为PAACOI算法.取任务量为1 000,对参数σ及μ的取值进行调整,以检验参数值对所提算法的寻优能力及收敛速度的影响.所得任务完成时间及迭代次数如表2所示.

由表2可知,当PAACO算法与PAACOI算法的参数值相同时,PAACO算法所得任务完成时间均优于PAACOI算法,说明对任务进行降序排序能有效地缩短任务完成时间.

多次实验可得,当$\sigma=0.25$,$\mu\in[0.3,0.6]$时,PAACO算法与PAACOI算法所得任务完成时间及算法的迭代次数均能取得较好值,而当$\mu=0.3$时,任务完成时间与算法迭代次数均得到最好值.

表2 任务完成时间及迭代次数

Tab.2 The task completion time and iteration times

参数值		PAACO		PAACOI	
		完成时间/s	迭代次数	完成时间/s	迭代次数
$\sigma=0.25$	$\mu=0.1$	26 655	36	26 771	39
	$\mu=0.3$	25 737	33	26 351	35
	$\mu=0.6$	26 397	36	26 463	37
	$\mu=0.9$	26 519	38	26 562	38
$\mu=0.3$	$\sigma=0.35$	26 570	36	26 623	36
	$\sigma=0.45$	26 716	37	26 789	38

当$\sigma=0.25$,$\mu=0.1$及$\mu=0.9$时,算法所得任务完成时间及迭代次数均相对较差,此种情况是由于将任务集中度阈值设定过低,不能较好体现任务分配集中程度的差异,过高则不能达到执行概率自适应调整方法条件,故均不利于算法寻优.

当$\mu=0.3$时,σ越大,即概率自适应调整方法影响的迭代次数越多,算法表现的效果越不理想,此种情况是由于算法在迭代中期较优的任务分配方案所对应的信息素在逐渐累积,若继续运用所提方法对算法进行调整,将不利于较优分配方案对应的信息素累积,从而导致算法不能收敛到全局最优解.

将本文PAACO算法及PAACOI算法(参数均取最优值)与ACO算法及文献[9]中的MACO算法进行对比实验,所得任务完成时间如表3所示.

表3 任务完成时间

Tab.3 The task completion time s

任务数量	PAACO	PAACOI	ACO	MACO
250	7 811	7 905	7 960	7 949
500	14 178	14 200	14 384	14 309
750	20 362	20 405	20 719	20 744
1 000	25 737	26 351	26 678	26 638
1 250	32 866	33 129	33 714	33 572
1 500	39 228	39 419	40 020	39 449
1 750	45 372	45 584	45 842	45 634
2 000	52 004	52 093	52 534	52 326

由表3数据可知,在任务量为250~2 000时,PAACOI算法和PAACO算法的任务总完成时间均低于ACO算法及MACO算法,说明PAACOI算法能有效地缩短任务总完成时间.而结合了任务排序方法的PAACO算法可以更有效地缩短任务完成时间,且在任务量改变的情况下,算法均具

备较好的寻优能力. 而 MACO 算法虽将遗传算法融入到蚁群优化算法的每一次迭代中,增加了种群的多样性,但由于采用的交叉、变异策略对算法的影响力度有限,并不能使算法很好地获得全局最优解. 由表 3 可知,PAACO 算法相对 ACO 算法,任务平均总完成时间降低了 1.77%,相对 MACO 算法平均总完成时间降低了 1.27%.

为了验证算法的执行效率,笔者统计了 4 种算法的收敛代数及执行 8 个任务集各算法的花费时间. 实验结果如表 4 所示,执行时间如图 1 所示.

表 4　算法收敛代数

Tab. 4　Iteration times of algorithm convergence

任务数量	PAACO	PAACOI	ACO	MACO
250	35	35	36	36
500	36	36	39	37
750	35	36	37	37
1 000	33	35	38	38
1 250	36	36	38	37
1 500	37	38	39	39
1 750	35	37	38	38
2 000	36	37	38	37

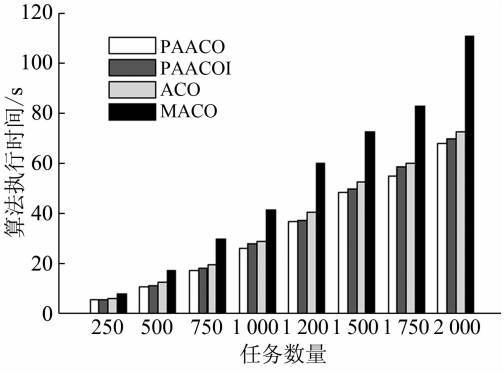

图 1　算法执行时间

Fig. 1　Execution time of algorithm

由表 4 可知,本文 PAACO 算法及 PAACOI 算法的收敛代数均低于 ACO 算法及 MACO 算法,表明任务分配概率自适应调整方法提高了算法的寻优能力,可使算法具备较快的收敛速度. 本文 PAACO 算法相对 ACO 算法,收敛速度提高了 6.60%,相对 MACO 算法提高了 5.35%.

由图 1 可知,PAACO 算法执行时间均低于 ACO 算法及 MACO 算法,说明对任务排序,同时采用任务分配概率自适应调整方法能有效地提高算法的执行效率. MACO 算法的执行时间最长,是因为该算法将遗传算法融入到蚁群优化算法的每一次迭代中,额外消耗了较多的时间. 通过实验对比,本文 PAACO 算法平均执行时间相对 ACO 算法降低了 8.84%,相对 MACO 算法降低了 37%.

由于本文 PAACO 算法、PAACOI 算法仅执行效率相对 MACO 算法优势明显,为验证所提算法的有效性,笔者对各算法所得结果进行统计检验.

根据实验数据可知,任务量的变化对本文所讨论的 4 种算法的任务完成、迭代次数及算法执行时间无明显影响,故取任务量为 1 000,将 4 种算法均执行 50 次,对各算法所得的任务完成时间、迭代次数以及执行时间进行分析. 在对样本进行统计检验时,需预知样本的分布特征,笔者运用 Kolmogorov-Smirnov 检验方法,通过 SPSS 软件计算可得 4 种算法所得的样本均符合正态分布.

为进一步确定能否运用独立样本 t 检验方法来检验本文算法所得 3 项检验样本相对 ACO 算法及 MACO 算法所得样本的差异显著性,需分别对由 PAACO 算法、PAACOI 算法相对 ACO 算法、MACO 算法组成的 4 组实验组进行方差齐性检验. 通过 SPSS 软件计算可知,4 组实验组对应的 3 项指标方差均为齐性,由此表明实验可运用独立样本 t 检验方法.

假设 PAACO 算法、PAACOI 算法所得 3 项检验样本相对 ACO 算法及 MACO 算法无显著性差异,设定概率显著性水平 $\alpha = 0.05$,通过独立样本 t 检验方法所得显著性检验值如表 5 所示.

表 5　显著性检验值

Fig. 5　Significant test values

算法对比	任务完成时间	迭代次数	算法执行时间
PAACOI vs. ACO	0.449	0.161	0.016
PAACOI vs. MACO	0.580	0.214	0.000
PAACO vs. ACO	0.223	0.032	0.000
PAACO vs. MACO	0.295	0.049	0.000

由表 5 可知,PAACOI 算法相对 ACO 算法及 MACO 算法,执行时间样本所得的显著性检验值均小于 0.05,因此拒绝原假设,故仅采用概率自适应调整方法的 PAACOI 算法的执行时间相对 ACO 算法和 MACO 算法有显著性差异.

PAACO 算法相对 ACO 算法及 MACO 算法,迭代次数和算法执行时间有显著性差异,表明将任务排序方法和概率自适应调整方法结合,能更为有效地提高算法的效率.

因此由统计检验结果可得,PAACO 算法相对 ACO 算法、MACO 算法,尽管所得任务完成时间没有显著优势,但迭代次数及执行效率的优化程度较为显著,是一种有效的云任务调度方法.

5　结论

笔者针对基本蚁群算法易陷入局部最优的缺

陷,结合云计算任务调度特点,依据任务量大小对任务进行排序,同时引入任务分配概率自适应调整因子,对任务分配集中度过高的任务分配概率进行调整,不仅有效地避免了算法局部收敛,而且改善了算法的执行效率和收敛速度.实验表明,本文PAACO算法在执行效率上优于ACO算法及MACO算法,任务完成时间也在一定程度上有所降低.目前,该算法只在静态任务调度方面得到了验证,对于云计算资源节点失效等不可预见的动态云计算环境中任务调度问题,算法的有效性还需进一步研究探讨.

参考文献:

[1] ARMBRUST M, FOX A, GRIFFITH R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[2] 田文洪,赵勇.云计算资源调度管理[M].北京:国防工业出版社,2011.

[3] 张建勋,古志民,郑超. 云计算研究进展综述[J]. 计算机应用研究, 2010, 27(2):429-433.

[4] 焦留成, 邵创创, 程志平. 一种求解连续空间约束优化问题的蚁群算法[J]. 郑州大学学报(工学版), 2015, 36(1):20-23.

[5] AL-MAAMARI A, OMARA F A. Task scheduling using PSO algorithm in cloud computing environments [J]. International journal of grid and distributed computing, 2015, 8(5): 245-256.

[6] FU J C, CHEN C C, CHAI J W, et al. Image segmentation by EM-based adaptive pulse coupled neural networks in brain magnetic resonance imaging[J]. Computerized medical imaging and graphics, 2010, 34(4): 308-320.

[7] DORIGO M, BIRATTARI M, STUTZLE T. Ant colony optimization[J]. Computational intelligence magazine, IEEE, 2006, 1(4): 28-39.

[8] MISHRA R, JAISWAL A. Ant colony optimization: A solution of load balancing in cloud[J]. International journal of web & semantic technology, 2012, 3(2): 33-50.

[9] 王永贵,韩瑞莲. 基于改进蚁群算法的云环境任务调度研究[J]. 计算机测量与控制, 2011, 19(5): 1203-1204.

[10] 赵宝江,李士勇,金俊. 基于自适应路径选择和信息素更新的蚁群算法[J]. 计算机工程与应用, 2007, 43(3):12-15.

[11] NAIMI H M, TAHERINEJAD N. New robust and efficient ant colony algorithms: Using new interpretation of local updating process[J]. Expert systems with applications, 2009, 36(1): 481-488.

[12] ZUKHRI Z, PAPUTUNGAN I V. A hybrid optimization algorithm based on genetic algorithm and ant colony optimization[J]. International journal of artificial intelligence & applications, 2013, 4(5): 63-75.

[13] 黄竞伟,朱福喜,康立山.计算智能[M].北京:科学出版社,2010.

[14] CALHEIROS R N, RANJAN R, BELOGLAZOV A, et al. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: practice and experience, 2011, 41(1): 23-50.

Task Scheduling Method Based on Probability Adaptive Ant Colony Optimization in Cloud Computing

WANG Junying[1,2], YAN Fenfen[1,2], CHEN Peng[1], DONG Fangmin[1,2], ZANG Zhaoxiang[1]

(1. College of Computer and Information, China Three Gorges University, Yichang 443002, China; 2. Hubei Key Laboratory of Intelligent Vision Based Monitoring for Hydroelectric Engineering, China Three Gorges University, Yichang 443002, China)

Abstract: The basic ant colony algorithm tended to be trapped in local optimum in solving task scheduling problems of cloud computing. A probability adaptive ant colony optimization was proposed. This algorithm rankd the tasks in descending order according to their size, defines the task concentration degree, and introduces the probability adaptive adjustment factor to adjust the assignment probability of over-concentrated resource noded. The results showed that the proposed algorithm shortened the task completion time, and had some improvements on convergence speed, compared with the Ant Colony Optimization and Modified Ant Colony Optimization.

Key words: cloud computing; task scheduling; ant colony optimization; probability adaptive