云环境下基于改进蚁群算法的任务调度

何长杰,白治江
(上海海事大学 信息工程学院,上海 201306)

摘 要: 云计算是能够提供动态资源池、虚拟化和高可用性的计算平台,达到可扩展性和高可用性两个重要目标。其中云计算任务调度负责为用户的计算任务分配合适的资源,成为云计算一个核心问题。由于云计算任务调度问题是 NP-hard 问题,近些年提出的启发式(heuristics)算法和元启发式(meta-heuristics)算法取得了良好的效果,蚁群算法作为元启发式算法有良好的鲁棒性和并行性,适合求解组合优化问题。元启发式算法相比较启发式算法求解精度高,但算法运行时间长,基于精英策略的蚁群算法虽能加快收敛速度却易陷入局部最优。为了扩大蚁群搜索空间,防止算法陷入局部最优解,对信息素更新和最优路径奖励进行了改进。实验证明改进后的蚁群算法降低了最大完成时间和不平衡程度,提高了云资源的利用率。

关键词: 云计算; 任务调度; 元启发式算法; 蚁群算法
中图分类号: TP393 **文献标识码:** A **文章编号:** 1673-629X(2018)12-0013-04
doi: 10.3969/j.issn.1673-629X.2018.12.003

Task Scheduling Based on Improved Ant Colony Algorithm in Cloud Environment

HE Chang-jie, BAI Zhi-jiang
(School of Information Engineering, Shanghai Maritime University, Shanghai 201306, China)

Abstract: Cloud computing is a computing platform that provides dynamic resource pooling, virtualization and high availability, achieving scalability and high availability of two important goals. Among them, the cloud computing task scheduling is responsible for allocating the appropriate resources for the user's computing tasks and becomes a core issue of cloud computing. Because cloud computing task scheduling is NP-hard problem, heuristics and meta-heuristics algorithms in recent years are proposed to achieve better results. The ant colony algorithm as a meta-heuristic algorithm has strong robustness and parallelism, which is suitable for solving combinatorial optimization problems. Compared with the heuristic algorithm, meta-heuristic algorithm has higher solution precision, but runs for a long time. Although the ant colony algorithm based on the elite strategy can speed up the convergence speed, it is easy to fall into the local optimum. In order to expand the ant colony search space and prevent the algorithm from falling into the local optimal solution, the pheromone update and the optimal path reward are improved. Experiment shows that the improved ant colony algorithm can reduce the maximum completion time and degree of unbalance, and improve the cloud resource utilization.

Key words: cloud computing; task scheduling; meta-heuristic; ant colony algorithm

0 引 言

云计算是一种商业计算模型,它将计算任务分配在由大量廉价计算机构成的资源池上,使各种应用系统能够根据需要获取计算力、存储空间和信息服务[1]。它通过虚拟化技术将软硬件资源形成一个巨大的资源池[2],当用户需要这些资源时,通过某种调度方式,将资源池中的资源分配给用户任务,云中的资源就像煤气、水和电一样,取用方便,费用低廉。云计算环境下的资源分配实际上根据调度目标将资源分配给云任务[3],调度目标不同,采取的分配策略也会不同,如以最优跨度、成本最低、服务质量、负载均衡等为调度目标。

目前云计算任务调度是 NP 完全问题[4],不少学者提出一些改进的调度算法,如 Li Kun 等[5]提出了 LBACO 算法,考虑虚拟机的计算能力、带宽和负载均衡,使得任务执行时间少且负载更加均衡。左利云

收稿日期: 2018-01-03 **修回日期:** 2018-05-06 **网络出版时间:** 2018-06-29
基金项目: 国家自然科学基金(71471110)
作者简介: 何长杰(1993-),男,硕士研究生,研究方向为云计算; 白治江,博士,副教授,研究方向为模式识别、人工智能、并行与分布式处理。
网络出版地址: http://kns.cnki.net/kcms/detail/61.1450.TP.20180629.1708.080.html

等[6]提出RCMM调度算法,将任务和资源等级的乘积作为组合值进行Min-Min算法调度,改进算法负载比原始Min-Min更加均衡。张春艳等[7]提出基于分组多态蚁群算法,侦察蚂蚁负责初始化信息素,搜索蚂蚁负责搜索解,使得平均完成时间降低。查英华等[8]提出增强蚁群算法,兼顾任务调度的最短完成时间和负载均衡,对信息素初始化和局部更新,以及节点选择进行改进,减少了算法运行时间,负载更加均衡。王芳等[9]为了解决蚁群算法收敛速度慢和容易陷入局部最优解的缺陷,概率选择时引入混沌扰乱策略,且自适应调整信息素挥发因子,实验表明改进后的算法时间跨度更小。

为了防止蚁群算法陷入局部最优解,文中进行了以下改进:对于没有访问的路径不蒸发信息素,增大其被搜索的概率;奖励局部最优路径及其近邻路径等多条路径,增大了蚁群搜索空间,防止蚁群算法陷入局部最优。

1 云计算任务调度

云计算调度模型如图1所示,分为两级调度,分别是云任务调度和虚拟机调度。其中云任务调度表示在理想或者极优的资源分配情况下,一个Vm应能以共享模式分配给多个终端用户的多个任务。虚拟机调度为Vm寻找合适的Host,再把Vm创建到这个Host上。文中仅仅考虑云任务调度算法。

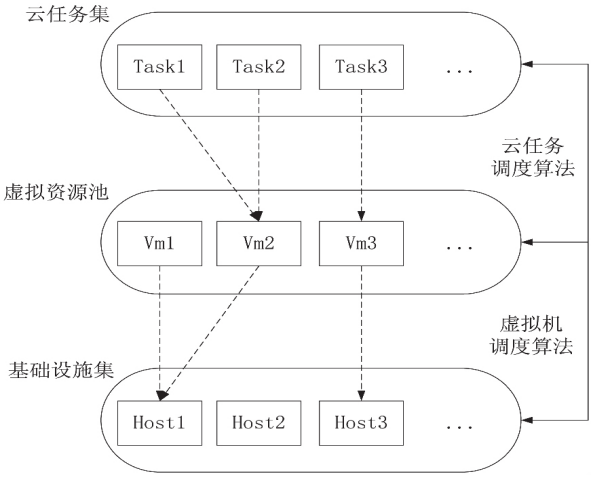

图1 云计算调度模型

云计算任务调度问题本质是根据某些调度目标将n个任务分配到m个虚拟机上,n通常大于m。使得云平台负载更加均衡,任务完成时间最小,尽可能提高云资源利用率。

任务集合$\mathrm{TASK}=\{\mathrm{Task}_1,\mathrm{Task}_2,\cdots,\mathrm{Task}_n\}$表示当前任务队列有$n$个任务,任务指令长度用百万指令(million instructions,MI)表示。

虚拟机集合$\mathrm{VM}=\{\mathrm{Vm}_1,\mathrm{Vm}_2,\cdots,\mathrm{Vm}_m\}$表示云中有$m$个资源节点。虚拟机处理能力用每秒执行的百万指令数(million instructions per second,MIPS)表示。

TASK到VM的分配关系可以用分配矩阵X表示为:

$$\boldsymbol{X}=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \tag{1}$$

其中,x_{ij}表示Task_j和Vm_i的对应关系,$x_{ij}\in\{0,1\}$,$\sum_{i=1}^{m}x_{ij}=1$,$i\in\{1,2,\cdots,m\}$,$j\in\{1,2,\cdots,n\}$。将Task_j分配到Vm_i上,则$x_{ij}=1$,否则$x_{ij}=0$。Task_j在Vm_i上的执行时间用ET_{ij}表示,ET矩阵表示为:

$$\mathbf{ET}=\begin{bmatrix} \mathrm{et}_{11} & \mathrm{et}_{12} & \cdots & \mathrm{et}_{1n} \\ \mathrm{et}_{21} & \mathrm{et}_{22} & \cdots & \mathrm{et}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{et}_{m1} & \mathrm{et}_{m2} & \cdots & \mathrm{et}_{mn} \end{bmatrix} \tag{2}$$

$$\mathrm{et}_{ij}=\frac{\mathrm{TaskLength}_j}{\mathrm{VmMips}_i} \tag{3}$$

其中,$\mathrm{TaskLength}_j$表示任务j的指令长度;VmMips_i表示虚拟机i的处理能力。

2 蚁群算法

蚁群算法是Marco Dorigo在1992年提出的一种元启发式仿生算法,模拟蚁群觅食行为过程中发现最短路径的现象。蚂蚁在觅食中释放一种称为信息素(pheromone)的化学物质,蚁群之间通过信息素的浓度变化进行信息交流和相互协作,浓度越高的路径选择的概率越大。虽然每个蚂蚁个体智能有限,但通过群体的搜索、协作和涌现现象可以解决单个个体无法解决的问题。这种方法被很多研究者用于求解大多数NP-Hard优化问题,如旅行商(traveling salesman problem,TSP)、二次分配等组合优化问题[10]。蚁群算法求解TSP的过程描述如下:

Step1:创建antNum只蚂蚁,迭代次数g,初始化信息素矩阵、可访问表、禁忌表以及相关参数。

Step2:随机将蚂蚁分布在n个城市,将初始分配城市节点加入禁忌表Tabu_k,第k只蚂蚁从当前城市i选择下一个城市j。概率选择公式如下:

$$p_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}{\sum_{k\in\mathrm{allowed}_k}[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{ij}(t)]^{\beta}}, & \text{if } i\in\mathrm{allowed}_k \\ 0, & \text{else}\end{cases} \tag{4}$$

其中:$p_{ij}^{k}(t)$表示蚂蚁k从城市i到城市j的选择

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,其中 $\eta_{ij}(t)=1/d_{ij}$,d_{ij} 表示城市 i 到城市 j 的距离; allowed_k 为可访问表,表示蚂蚁 k 允许访问的下一个城市集合蚂蚁 k 访问后的城市放入禁忌表 Tabu_k 中。

Step3: 第 k 只蚂蚁搜索完成获得一条路径后,根据解的质量获取信息素增量矩阵 $\Delta\tau_{ij}^k$,定义如下:

$$\Delta\tau_{ij}^k=\begin{cases}\dfrac{Q}{\text{Length}_k}, & \text{if 第 } k \text{ 只蚂蚁经过}(i,j)\\ 0, & \text{else}\end{cases} \tag{5}$$

其中,$\Delta\tau_{ij}^k$ 表示第 k 只蚂蚁更新城市 i 到城市 j 的信息素增量矩阵; Q 为信息素强度常量; Length_k 为第 k 只蚂蚁搜索路径的长度。

Step4: antNum 只蚂蚁都搜索完成各自获得一条路径后,对信息素浓度进行更新。信息素更新公式如下:

$$\tau_{ij}(\text{new})=(1-\rho)\tau_{ij}(\text{old})+\Delta\tau_{ij} \tag{6}$$

$$\Delta\tau_{ij}=\sum_{k=1}^{\text{antNum}}\Delta\tau_{ij}^k \tag{7}$$

其中,ρ 表示信息素的挥发因子; $1-\rho$ 表示信息素的残留因子。

Step5: 若满足迭代结束条件,则直接打印最优路径结束。否则,重新初始化蚂蚁,跳转到 Step2。

3　蚁群算法求解云计算任务调度

将任务到虚拟机的一次分配作为蚂蚁搜索对象[8],用$(\text{Task}_j,\text{Vm}_i)$表示一个节点,当所有任务分配给虚拟机之后就形成一系列节点组成的路径。

3.1　启发式因子设置

$$\eta_{ij}=\frac{1}{d_{ij}} \tag{8}$$

$$d_{ij}=\frac{\text{TotalLength}_i}{\text{VmMips}_i}+\text{et}_{ij} \tag{9}$$

在 TSP 问题中,d_{ij} 表示城市 i 到城市 j 的距离,而在云计算任务调度问题中,d_{ij} 表示任务 j 分配给虚拟机 i 后的完成时间,包括分配之前虚拟机 i 已经执行的时间和分配之后任务 j 在虚拟机 i 的执行时间。TotalLength_i 表示已经分配给虚拟机 i 的任务指令总长度[11]。et_{ij} 表示任务 j 分配给虚拟机 i 的执行时间,由式 3 计算可得。

分配完成后,虚拟机 i 的任务指令总长度需要加上当前分配到的任务 j 的指令长度,计算公式如下:

$$\text{TotalLength}_i=\text{TotalLength}_i+\text{TaskLength}_j \tag{10}$$

3.2　改进信息素更新规则

经典蚁群算法的信息素更新规则会对全部路径进行信息素挥发,对于蚂蚁未经过的路径,该路径的信息素由于挥发越来越少,趋近于零,导致之后蚂蚁几乎不会选择该路径,降低了蚁群算法的搜索空间[12]。因此文中对信息素更新规则进行改进,迭代过程中,对于没有访问的路径不再挥发信息素。信息素更新公式如下:

$$\tau_{ij}(\text{new})=\begin{cases}(1-\rho)\tau_{ij}(\text{old})+\Delta\tau_{ij}, & \text{if } \Delta\tau_{ij}\neq 0\\ \tau_{ij}(\text{old}), & \text{else}\end{cases} \tag{11}$$

其中,$\Delta\tau_{ij}$ 表示一次迭代过程中根据各个蚂蚁搜索的路径以及路径的质量进行信息素更新。$\Delta\tau_{ij}=0$ 表明该路径没有信息素更新,亦没有蚂蚁访问,没有访问的路径不再挥发信息素。

3.3　定义问题解

所有蚂蚁遍历完成获得分配方案后,虚拟机 i 总的执行时间即负载定义为:

$$L_i=\frac{\text{TotalLength}_i}{\text{VmMips}_i} \tag{12}$$

TSP 问题中将路径的长度作为解,路径越短,获得的解越好。在云计算任务调度问题中,由于各个虚拟机并行执行,更多地考虑所有任务执行完成的最大时间[13],因此 t 次迭代时第 k 只蚂蚁获得的解定义如下:

$$L_k(t)=\max\{L_i\} \tag{13}$$

迭代过程中最优解定义为 L^*,表示如下:

$$L^*=\min_{1\leqslant k\leqslant \text{antNum}}\{L_k\} \tag{14}$$

3.4　最优解奖励改进

基于精英的蚁群算法对最优路径进行额外奖励,虽然能够加快收敛速度,却容易陷入局部最优解,这是由于精英蚁群算法仅仅奖励局部最优解路径这一条路径,导致这一条路径上的信息素偏高。云计算环境中许多虚拟机的处理能力相当,因此这些虚拟机之间处理相同的任务集所需的执行时间也相当。文中对最优解路径上的 Vm_* 搜索与之处理能力相当的虚拟机,交换两个虚拟机上的任务集得到新的路径,此时的路径定义为近邻路径,对最优路径和近邻路径多条路径进行信息素奖励,有利于发现近邻解,扩大搜索范围。同时给予其他路径很少的奖励,防止最优解的路径与其他路径的信息素差异太大,陷入局部最优。奖励公式如下:

$$\tau_{ij}(t)=\tau_{ij}(t)+\frac{Q}{L^*}\times\left(1-\frac{|\text{VmMips}_i-\text{VmMips}_*|}{\text{VmMaxMips}}\right) \tag{15}$$

其中,VmMips_i 虚拟机 i 的处理能力; VmMips_* 是最优解 L^* 中 Task_j 分配到的 Vm_* 的处理能力; VmMaxMips 是所有虚拟机中处理能力的最大值。

4 实验及结果分析

文中选择墨尔本大学 Gridbus 项目组提出的云计算仿真软件 Cloudsim[14] 3.0 进行仿真实验，构造蚂蚁类 Ant 和蚁群类 ACO，重载 DatacenterBroker 类。云仿真器的参数设置如表 1 所示。

表 1 CloudSim 中云环境的参数设置

实体类型	参数	值
任务	指令长度	5 000 ~ 15 000
	任务总数	100 ~ 300
虚拟机	虚拟机总数	50
	MIPS	500 ~ 2 000
数据中心	主机数	10

为了保证实验的准确性，将任务数据和虚拟机数据保存在文件中，进行实验时从文件中读取。蚁群算法 α 值越大，搜索易过早陷入局部最优，β 值越大越接近贪婪规则，α 一般取值为 1，β 一般取值为 5。蚁群算法参数设置如表 2 所示。

表 2 蚁群参数设置

参数	值
蚂蚁数	10
迭代次数	10
α	1.0
β	5.0
ρ	0.5
Q	100

定义云中资源不平衡程度(degree of imbalance，DI)：

$$DI = \frac{L_{max} - L_{min}}{L_{avg}} \tag{16}$$

其中，L_{max} 是任务分配完成之后集群中最大的虚拟机负载；L_{min} 是集群中最小的虚拟机负载；L_{avg} 是虚拟机平均负载。DI 值越小，集群中的负载就越均衡。

将文中算法和精英蚁群算法进行实验对比。由于蚁群算法的不稳定性，记录十次实验求平均值分别对最大完成时间和不平衡程度两个维度进行比较。

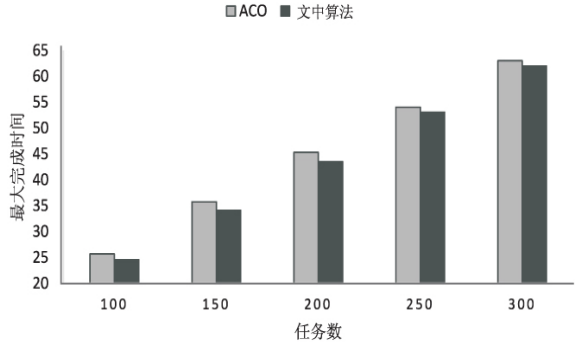

图 2 最大完成时间

图 2 为文中算法和 ACO 算法在不同任务数下的最大完成时间比较。可以看出，文中算法比 ACO 算法求得的最大完成时间更小，这是由于新算法对没有访问的路径不再挥发其信息素，增大其被访问的概率，其次奖励了最优路径及其近邻路径等多条路径，同时减小了最优路径与其他路径信息素的差异，扩大了搜索空间，防止蚁群算法陷入局部最优解。

图 3 为文中算法和 ACO 算法在不同任务数下的不平衡程度比较。可以看出，文中算法不平衡程度普遍优于 ACO 算法，表明新算法更加合理有效地使用资源。随着任务数的增加，云中虚拟机负载不平衡程度减小，这是由于任务之间的指令长度相对差异减小，使得虚拟机上的负载差异也减小，因此不平衡程度也随之减小，且随着任务增加不平衡程度下降趋于平缓。

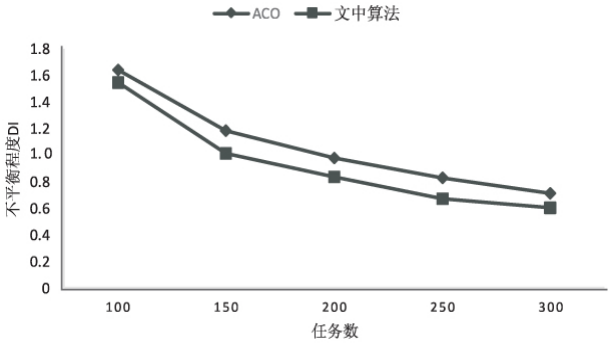

图 3 不平衡程度

5 结束语

元启发式算法相比较启发式算法，虽然求解精度高，但运行时间长。基于精英策略的蚁群算法能够加快收敛速度，却容易陷入局部最优。文中对精英蚁群算法进行了改进，首先对迭代过程中没有访问的路径不蒸发信息素，防止该路径信息素越减越少，增大其被访问的概率，扩大蚁群的搜索空间。其次在最优路径奖励时，对最优路径和近邻路径等多条路径进行奖励，增加了奖励路径的数目，扩大了蚁群搜索空间，同时降低最优路径和其他路径的信息素差值，既增强了最优路径又防止蚁群算法陷入局部最优解。实验证明，该算法在最大完成时间和不平衡程度下能求得更优的解，提高了云资源利用率。

参考文献：

[1] 刘 鹏. 云计算 [M]. 第 2 版. 北京: 电子工业出版社，2011.

[2] 魏 勇，赵开新，张松青，等. 基于改进蚁群算法的云计算任务调度研究 [J]. 火力与指挥控制，2017，42(5)：130-133.

[3] 陈国良，明 仲. 云计算工程 [M]. 北京: 人民邮电出版社，2016.

(下转第 22 页)

参考文献:

[1] XIE Juanying, JIANG Shuai. A simple and fast algorithm for global k-means clustering [C] //Second international workshop on education technology and computer science. Wuhan, China: IEEE, 2010: 36-40.

[2] ALKOFFASH M S. Automatic arabic text clustering using k-means and k-mediods [J]. International Journal of Computer Applications, 2012, 51(2): 5-8.

[3] 潘 楚,罗 可. 基于改进粒计算的 K-medoids 聚类算法 [J]. 计算机应用, 2014, 34(7): 1997-2000.

[4] 孙立新,张栩之,邓先瑞,等. 自适应果蝇算法优化模糊均值聚类算法图像分割 [J]. 控制工程, 2016, 23(4): 494-499.

[5] 孙 胜,王元珍. 基于核的自适应 k-medoid 聚类 [J]. 计算机工程与设计, 2009, 30(3): 674-675.

[6] 程 慧,刘成忠. 基于混沌映射的混合果蝇优化算法 [J]. 计算机工程, 2013, 39(5): 218-221.

[7] 印 溪,许 斌,亓 晋. 一种基于禁忌策略的混合优化算法 [J]. 计算机技术与发展, 2017, 27(2): 46-50.

[8] PAIL W T. A new fruit fly optimization algorithm: taking the financial distress model as an example [J]. Knowledge-Based Systems, 2012, 26: 69-74.

[9] DU Tingsong, KE Xianting, LIAO Jiagen, et al. DSLC-FOA: an improved fruit fly optimization algorithm application to structural engineering design optimization problems [J]. Applied Mathematical Modelling, 2017, 55: 314-339.

[10] 韩俊英,刘成忠. 自适应混沌果蝇优化算法 [J]. 计算机应用, 2013, 33(5): 1313-1316.

[11] VERMA J, RICHHARIYA V. A review: salient feature extraction using k-mediods clustering technique [J]. Asian Journal of Computer Science & Information Technology, 2013, 2(3): 11-19.

[12] 王宏智,高学东,赵 杨. 一种群体智能聚类算法研究 [J]. 中国管理信息化, 2013, 16(2): 74-75.

[13] 李 莲,罗 可,周博翔. 一种改进人工蜂群的 K-medoids 聚类算法 [J]. 计算机工程与应用, 2013, 49(16): 146-150.

[14] 马 菁,谢娟英. 基于粒计算的 K-medoids 聚类算法 [J]. 计算机应用, 2012, 32(7): 1973-1977.

[15] 姚丽娟,罗 可,孟 颖. 一种基于粒子群的聚类算法 [J]. 计算机工程与应用, 2012, 48(13): 150-153.

(上接第 16 页)

[4] PACINI E, MATEOS C, GARINO C G. Distributed job scheduling based on swarm intelligence: a survey ☆ [J]. Computers & Electrical Engineering, 2014, 40(1): 252-269.

[5] LI Kun, XU Gaochao, ZHAO Guangyu, et al. Cloud task scheduling based on load balancing ant colony optimization [C] //Sixth annual china grid conference. [s. l.]: IEEE, 2011: 3-9.

[6] 左利云,左利锋. 云计算中基于预先分类的调度优化算法 [J]. 计算机工程与设计, 2012, 33(4): 1357-1361.

[7] 张春艳,刘清林,孟 珂. 基于蚁群优化算法的云计算任务分配 [J]. 计算机应用, 2012, 32(5): 1418-1420.

[8] 查英华,杨静丽. 改进蚁群算法在云计算任务调度中的应用 [J]. 计算机工程与设计, 2013, 34(5): 1716-1719.

[9] 王 芳,李美安,段卫军. 基于动态自适应蚁群算法的云计算任务调度 [J]. 计算机应用, 2013, 33(11): 3160-3162.

[10] DORIGO M, GAMBARDELLA L M. Ant colony system: a cooperative learning approach to the traveling salesman problem [J]. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 53-66.

[11] TAWFEEK M A, EL-SISI A, KESHK A E, et al. Cloud task scheduling based on ant colony optimization [C] //8th international conference on computer engineering & systems. Cairo, Egypt: IEEE, 2013: 64-69.

[12] 袁亚博,刘 羿,吴 斌. 改进蚁群算法求解最短路径问题 [J]. 计算机工程与应用, 2016, 52(6): 8-12.

[13] 黄 俊,王庆凤,刘志勤,等. 基于资源状态蚁群算法的云计算任务分配 [J]. 计算机工程与设计, 2014, 35(9): 3305-3309.

[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 & Experience, 2011, 41(1): 23-50.