云环境下基于改进蚁群算法的虚拟机批量部署研究

杨 星 马自堂 孙 磊
(解放军信息工程大学电子技术学院 郑州 450004)

摘 要 针对云计算中虚拟机批量部署问题,在定义虚拟机与服务器匹配距离的基础上,使用蚁群优化思路进行部署方案搜索,并有针对性地对蚁群算法进行了扩展改进。首先在蚁群算法随机比例规则中加入性能感知策略,以尽量避免将相同性能偏好的虚拟机部署在同一台服务器上,造成对硬件资源竞争的危险。同时增加了单一蚂蚁信息素更新规则,以减少错误先验知识对蚂蚁后续选择的误导。通过在 CloudSim 中的仿真实验,对算法参数选择进行了研究。与现有部署算法相比,本算法具有更好的系统负载均衡性能和资源利用率,以及比基本蚁群算法更快的收敛速度。

关键词 云计算,虚拟机,蚁群优化,信息素,负载均衡
中图法分类号 TP301 **文献标识码** A

Research on Extended Ant Colony Optimization Based Virtual Machine Deployment in Infrastructure Clouds

YANG Xing MA Zi-tang SUN Lei
(Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou 450004, China)

Abstract Aiming at the virtual machine deployment problem in the cloud computing environment, based on the defining of the match-distance between the virtual machine and the server, the ant colony optimization(ACO) was used to research the deployment scheme. And the ACO was extended and modified for the deployment problem. Using the probabilistic tour decision with performance apperceive policy, the virtual machines with the same performance interest are designedly placed in different servers to reduce the competition of the hardware resources. And using the single ant pheromone update rules, the misdirection of the inaccurate heuristic information is avoided. The parameter values for the arithmetic were researched with the experiments in CloudSim. Finally, the performance of the extended ACO was compared with that of the ranking deployment arithmetic and the original ACO. The experimental results show that the extended ACO meets the need of the system load balancing better, and accelerates the convergence to the original ACO.

Keywords Cloud computing, Virtual machine, Ant colony optimization, Pheromone, Load balancing

为了解决网络应用爆炸式膨胀的数据量和企业日益增长的高昂 IT 设施维护成本[1],在 Google、Amazon 等公司的推动下,云计算应运而生,并迅速得到工业界和学术界的广泛关注。其按需支付、即时服务、动态扩展以及对用户屏蔽底层技术细节等特点[2],革命性地改变了人们对于 IT 资源的使用方式,在为使用者提供高质量服务的同时,降低了 IT 的运营和维护成本。以 Amazon 的 EC2[3] 为代表的基础设施,即服务(Infrastructure as a Service, IaaS)是云计算的一个主要服务形式,其将硬件资源(包括物理硬件资源和虚拟化资源)动态按需提供给用户。支撑该层服务的技术体系主要包括虚拟化技术和相关的资源动态管理与调度技术[4]。

传统的虚拟机创建步骤过于繁琐,无法满足云计算的快速部署需求,因而更为快捷的部署技术——虚拟器件技术[5]广泛地被云计算所采用。虚拟器件是一个包括预安装、预配置的操作系统、中间件和应用程序的最小化的虚拟机。其以镜像模式存储在镜像仓库中,当需要部署应用时,只需把相应的镜像拷贝到目标物理机上,便可以开启其中虚拟机应用。

多虚拟环境的管理通常采用集中管理方式[6],简化的系统架构如图 1 所示。为提高虚拟机的部署效率,云计算系统通常会采用并行或协同批量部署技术,将多个虚拟机同时部署到多个物理服务器上,由前端管理服务器或服务器群统一调度。

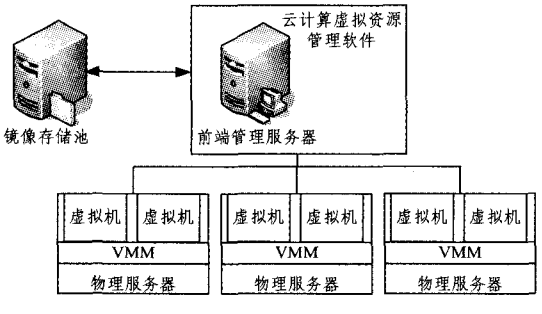

图 1 多虚拟机集中管理

现有的多机环境虚拟化中已有相应的批量部署策略,文献[7]研究了多机环境下虚拟机批量部署问题,总结了顺序部署和均衡部署两种常用的部署策略。但这两种部署方法均是

到稿日期:2011-10-31 返修日期:2012-02-17 本文受武器装备预研重点基金项目(9140A15060311JB5201)资助。
杨 星(1986—),男,硕士生,主要研究方向为云计算、系统优化,E-mail:mars654@hotmail.com;马自堂(1962—),男,教授,主要研究方向为信息安全、密码系统工程;孙 磊(1973—),男,博士,副研究员,主要研究方向为云计算基础设施可信增强、可信虚拟化技术。

在能够预测服务器可部署虚拟机个数的前提下运行的，而云计算具有用户申请的随机性和多样性，很难确定某一服务器可部署的虚拟机个数。均衡部署虽然对负载均衡有一定的考虑，但也仅仅停留在对虚拟机部署个数的均衡方面，而无法直接应用于云计算环境。

文献[8]中提出了一种基于元区间的云计算基础设施服务调度算法，在为计算节点划分元区间的基础上，给出了一种部署调度算法，其采用内存与磁盘大小比率来确定分配标杆，然后按照确定的标杆对物理资源和用户请求资源从大到小排队，依次进行部署。该方法对小规模的云计算部署应用具有不错的效果，元区间的思想很难适应大规模的云计算部署应用，并且其选择的性能参考标准仅限于两种，具有一定的局限性。

目前，大多开源的 IaaS 管理软件都使用贪婪算法进行虚拟机部署。各个 IaaS 服务提供商的虚拟机部署算法都是不公开的[9]，但它们通常都会考虑以下两点部署要素：1)提高资源利用率，降低能源开销，尽量不启动新的物理服务器；2)尽可能地让不同资源消耗的虚拟机互补，让不同类型业务的虚拟机分配在同一台物理服务器上。本文部署方法也是借鉴以上思路。

1 虚拟机批量部署问题建模

1.1 性能匹配距离

虚拟机与服务器的映射关系为 $n\to1$，一台虚拟机只能部署在一台服务器上，而一台服务器上可以部署多台虚拟机或不部署虚拟机。在进行虚拟机部署时，首先要确定服务器是否能满足此虚拟机的性能需求。设$\{Ser_{1p},\cdots,Ser_{mp}\}$为服务器 Ser_j 同类剩余性能集合，$\{Vm_{1p},\cdots,Vm_{np}\}$为虚拟机 Vm_i 相对应的同类需求性能集合。当$(Ser_{jp}-Vm_{ip})(p=1,2,\cdots,k)$存在负值($k$ 为参考性能个数)时，将服务器 Ser_j 视为无法满足虚拟机 Vm_i 的性能需求，而不考虑将 Vm_i 部署在 Ser_j 上；其为正值时，需要定量衡量虚拟机与各服务器之间的匹配关系，来为是否进行部署提供参考。为此本文提出匹配距离 $Mat_{vm_i ser_j}$(以下简写为 Mat_{ij})的概念。

首先为了规范地比较虚拟机与服务器，对服务器和虚拟机性能进行归一化处理。服务器性能归一化算法为：$S_{jp}=\frac{Ser_{jp}-Ser_p^{\min}}{Ser_p^{\max}-Ser_p^{\min}}$，其中 $Ser_p^{\max}$ 为服务器同类性能 p 的最大值，$Ser_p^{\min}$ 为最小值；同样，虚拟机期望性能归一化算法为：$V_{ip}=\frac{Vm_{ip}-Vm_p^{\min}}{Vm_p^{\max}-Vm_p^{\min}}$。若最大值与最小值相同，则参数均视为 1。

匹配距离是指待部署虚拟机需求性能和服务器空闲性能之间的欧式距离，算法为 $Mat_{ij}=\sqrt{\sum_{p=1}^{k}(S_{jp}-V_{ip})^2}$。可以看出，匹配距离越小，服务器 Ser_j 越能适应虚拟机 Vm_i 的性能需求。为了更好满足下文算法，对 Mat_{ij} 设置下限为 0.1，若匹配距离小于下限则均视为 0.1，以防止匹配距离为 0 的现象出现。

在批量部署过程中，匹配距离是一个变化的可预测值。若预计将虚拟机 Vm_i 部署在服务器 Ser_j 上，则在考虑 Ser_j 与其他虚拟机的匹配距离时，需使用 Ser_j 部署 Vm_i 后的剩余性能做计算。

1.2 数学模型

问题描述：将 n 个相互独立的虚拟机$(vm_1,vm_2,\cdots,vm_n)$部署在 m 个服务器$(ser_1,ser_2,\cdots,ser_m)$上，虚拟机 VM_i 与服务器 Ser_j 的匹配距离为 Mat_{ij}。虚拟机部署的最优解，是最大限度使用物理资源，且达到系统的负载均衡，获得虚拟机与服务器最小匹配距离和。虚拟机批量部署中，虚拟机与服务器之间的映射匹配问题是一个典型组合优化的 NP-hard 问题。其数学模型可以描述为：

1. 待部署虚拟机构成有限集 $VM=(vm_1,vm_2,\cdots,vm_n)$，目标服务器构成有限集 $SER=(ser_1,ser_2,\cdots,ser_m)$。

2. 有限集 $L=\{l_{vm_i ser_j}\mid(vm_i,ser_j)\in CONN\}$是 $CONN$ 中各元素间关系集合，其中 $|L|\leqslant n\times m$，$CONN$ 是笛卡尔积(Cartesian product)$VM\times SER$，$l_{vm_i ser_j}$ 是一个 0，1 编码变量，其取值为：

$$l_{vm_i ser_j}=\begin{cases}1, & \text{虚拟机 } vm_i \text{ 部署在服务器 } Ser_j \text{ 上}\\ 0, & \text{其他}\end{cases}$$

为方便叙述，下文将 $l_{vm_i ser_j}$ 简写为 l_{ij}。L 满足约束 $\sum_{j=1}^{m}l_{ij}=1$(本文暂不考虑存在所有服务器都不满足某虚拟机性能需求的情况，此情况可以采用返回队列重新部署，并申请云计算服务管理中心开启新服务器来解决)，$\sum_{i=1}^{n}l_{ij}\geqslant0$。

3. $Mat(l_{ij},t)$是部署状况 t 参数化的 $l_{ij}\in L$ 匹配距离方程，即虚拟机 VM_i 与服务器 Ser_j 在处于部署状况 t 时的匹配距离。

4. $s=\langle l_{ip},l_{jq}\cdots l_{kk}\cdots\rangle$是关系 L 中所有取值为 1 的元素组成的有序数列，当每个虚拟机 $vm_i(i=1,2,\cdots,n)$对应的 l_{ij} 有且仅有一个，同时其中任意 l_{ij} 对应的 $Mat_{ij}>0$，则称 s 为一个可行部署方案，其方案的匹配距离为：

$$MAT(s)=\sum_{l_{ij}\in s}Mat(l_{ij},t)\qquad(1)$$

所有可行部署方案 s 构成问题的可行方案域 S。

5. 批量部署问题就是求虚拟机与服务器最佳匹配距离的问题，目标函数可以描述为：

$$DEP^{best}(s)=\min MAT(s)\ (s\in S)$$

2 改进蚁群优化

2.1 蚁群优化算法

意大利学者 Dorigo 等人在 20 世纪 90 年代首次提出的基本蚁群算法[10](Ant System，AS)，是现今各种蚁群优化算法的原型。基本蚁群算法模仿蚂蚁寻食时的群体行为，采用人工蚂蚁行走路径来表示问题可行解。人工蚁群中各个蚂蚁相互独立地在问题解空间中进行搜索，以概率方式选择解的部件，并在其所选位置留下一定信息素(pheromone)。后续的蚂蚁再次进行搜索时，会以较大概率选择信息素较多的解，并留下更多的信息素，形成正反馈机制。照此方法进行多次迭代搜索后，信息素会在问题解的最优路径上逐渐增多，而其他路径上的信息素会随着迭代次数的增加而挥发，后续蚂蚁的搜索路径逐渐向最优解收敛。整个寻优过程，单个人工蚂蚁使用一种结构上的贪婪启发法搜索可行解。每只蚂蚁都能寻找出一个问题的解，但选择能力有限，系统的各个个体间通过信息素进行信息交换，形成集体自催化行为，最终找到问题的最优解路径。

基本蚂蚁算法收敛速度较慢，且会出现过早停滞的现象。

近年来，对算法的改进主要集中在提高算法性能和使其适用于更多优化问题上[11]。对于性能的改进，较为成功的有基于优化排序蚂蚁系统（Rank-Based Version of Ant System，AS_{rank}）[12]、蚁群系统（Ant Colony System，ACS）[13]、最大-最小蚂蚁系统（Max-Min Ant System）[14]等。至今蚁群优化算法已在许多实际的组合优化问题中取得很好的应用效果，包括最初的旅行商问题（TSP）以及二次分配问题（QAP）、车间任务调度问题（JSP）、图着色问题（GCP）以及网络路由问题等。

蚁群优化算法具有很好的鲁棒性和可扩展性，大规模批量部署问题本身也可以抽象为一个 NP 组合问题。本文将扩展改进蚁群优化算法，以解决云计算中虚拟机批量部署问题。

2.2 带有性能感知的服务器路径机制

算法每次迭代循环都有 n 只人工蚂蚁，分别从不同的虚拟机开始“旅行”（部署）。蚂蚁一次“旅行”需要进行 n 步，每完成其所在位置的虚拟机与服务器的部署映射便为完成一步，所走的步数计为 t。蚂蚁完成一次“旅行”便获得了一个部署方案 s，如果某只蚂蚁在“旅行”中遇到部署失败的情况，那么这只蚂蚁就视为“死亡”而不返回部署方案。蚂蚁在每一步部署时，以概率方式选择服务器，选择机制如下。

为减少将同一性能偏好的虚拟机部署在同一台服务器上而造成对服务器硬件资源的竞争风险，应尽量减少相同性能偏好虚拟机部署在同一台服务器上，因而在服务器选择过程中，为蚂蚁添加了性能感知特性。首先在一批虚拟机部署任务到达时，按照用户需求对虚拟机进行性能分类，分为计算密集型 CVm、网络密集性 NVm、存储密集型 SVm 和无特殊需求型 OVm。

在蚂蚁进行一次“旅行”时，每完成一次虚拟机在服务器上的部署，便参照部署虚拟机类型为服务器打上标签，部署分为计算密集型服务器 $CSer$、网络密集型 $NSer$、存储密集型 $SSer$，无特殊需求的便不打标签。同时为蚂蚁添加服务器规避列表 $tSer_{vmi}$。在蚂蚁旅行到虚拟机 Vm_i 进行服务器选择时，遵循以下逻辑判断是否将目标服务器 Ser_j 加入规避列表：

$$Ser_j \in tSer_{vmi} \Leftrightarrow (Vm_i \in CVm \wedge Ser_j \in CSer) \vee (Vm_i \in NVm \wedge Ser_j \in NSer) \vee (Vm_i \in SVm \wedge Ser_j \in SSer)$$

本文通过减小蚂蚁选择禁忌列表中虚拟机概率的方法，来避免同一台服务器部署两台相同性能偏好虚拟机的情况。蚂蚁 a 在第 c 次迭代搜索时，按照如下概率选择目标服务器：

$$p_a(i,j)=\begin{cases}\dfrac{\tau_{ij}(c)\cdot[\eta_{ij}(t)]^{\beta}}{\sum\limits_{l\notin tSer}\tau_{il}(c)\cdot[\eta_{il}(t)]^{\beta}+\sum\limits_{k\in tSer}\tau_{\min}\cdot[\eta_{ik}(t)]^{\beta}}, & j\notin tSer_{vmi}\\ p_a'(i,j), & j\in tSer_{vmi}\end{cases} \tag{2}$$

$$p_a'(i,j)=\frac{\tau_{\min}\cdot[\eta_{ij}(t)]^{\beta}}{\sum\limits_{l\notin tSer}\tau_{il}(c)\cdot[\eta_{il}(t)]^{\beta}+\sum\limits_{k\in tSer}\tau_{\min}\cdot[\eta_{ik}(t)]^{\beta}} \tag{3}$$

式中，$\tau_{ij}(c)$ 表示在进行第 c 次迭代时，虚拟机 i 映射到服务器 j 的信息素；$\eta_{ij}(t)$ 为启发因子，用来描述蚂蚁 a 在第 t 步将虚拟机 Vm_i 部署在服务器 Ser_j 上的期望程度，本文取匹配距离的倒数作为启发因子，即 $\eta_{ij}(t)=\dfrac{1}{Mat(l_{ij},t)}$；$\tau_{\min}$ 为最小信息素浓度，其值将在下一节信息素更新规则中进行讨论。β 体现了信息素与启发因子在蚂蚁选择中的相对重要性，$\beta\geqslant 0$。

理论上，加入性能感知策略对于算法搜索最短路径会有额外限制，这将影响算法的搜索性能。但在实际应用中，往往制定的偏好性能均是其需求较高的性能，按照本文的匹配距离计算方法，相同性能偏好的虚拟机应该尽量分开部署。因而对于云计算中虚拟机批量部署问题，加入性能感知策略可以加快其收敛速度，这将在下面试验中得到验证。

2.3 信息素更新规则

蚂蚁进行搜索时，会释放新的信息素，同时原有的信息素会随着迭代次数增加而挥发。本文制定了两种信息素更新规则：信息素全局更新规则和单一蚂蚁更新规则。

信息素全局更新规则，只有获得全局最优解的蚂蚁所途径的虚拟机服务器组合时才允许增加信息素。这是为了使后续蚂蚁搜索范围更加集中在已发现的最优解周围内，以增加算法的收敛速度。全局信息素增加方式如下所示：

$$\tau_{ij}(c+1)\leftarrow\rho\tau_{ij}(c)+\Delta\tau^{best} \tag{4}$$

式中，ρ 是全局信息素衰减参数，$0<\rho<1$，$\Delta\tau(i,j)$ 为信息素增加量，确定如下：

$$\Delta\tau^{best}=\begin{cases}\sigma/MAT^{sb}(c), & \text{如果 } l_{ij} \text{ 属于全局最优解}\\ 0, & \text{否则}\end{cases}$$

式中，$MAT^{sb}(c)$ 是第 c 此迭代搜索完成为止发现的全局最优解匹配距离，σ 是调和常数，取值 1。其他路径的信息素只按照衰减参数进行挥发。

同时，因为蚂蚁在“旅行”中一旦选择一个虚拟机与服务器 r 的匹配，其在后续部署中，服务器 r 对应其他虚拟机的匹配距离 $Mat(l_{ir},t)(i=1,2,\cdots,n)$ 就会发生变化，先验知识对下一步虚拟机部署会产生误导，于是本算法引入单一蚂蚁信息素更新策略，以对 $Mat(l_{ir},t)(i=1,2,\cdots,n)$ 发生变化的组合节点处的信息素进行调节。单一蚂蚁信息素更新是一种蚂蚁自感知策略，只对引起变化的单一蚂蚁及其正在进行的搜索过程有效，而不会影响到并行的其他蚂蚁以及后续蚂蚁的搜索行为。其更新方式为：

$$\tau(i,r)\leftarrow\tau(i,r)+\lg\left(\frac{Mat(l_{ir},0)}{Mat(l_{ir},t)}\right)\Delta\tau(i,j) \tag{5}$$

2.4 信息素限制

为了防止算法过早收敛于局部极优解，每个虚拟机与服务器映射组合的信息素限制在$[\tau_{\min},\tau_{\max}]$之间，超出范围的值将被设定为上、下限：

$$\begin{cases}\tau_{ij}(c)=\tau_{\min}, & \tau_{ij}(c)<\tau_{\min}\\ \tau_{ij}(c)=\tau_{\max}, & \tau_{ij}(c)>\tau_{\max}\end{cases} \tag{6}$$

这样可以避免某匹配组合上的信息素浓度远远大于其他组合，以便于算法寻找新的最优解，并避免出现过早停滞的现象，加快了算法收敛速度。

首先对信息素上限 $\tau_{\max}$ 取值进行推导。系统中最大的信息素浓度的部署组合，应该是在每一次迭代循环后都按照 $DEP_{best}(s)$ 进行信息素全局更新部署组合。在进行了 N_c 次迭代搜索后，根据式(4)得，最大可能的信息素为：

$$\tau_{ij}^{\max}(N_c)=\sum_{k=1}^{N_c}\rho^{N_c-k}\frac{\sigma}{DEP^{best}(s)}+\rho^{N_c}\tau_{ij}(0)$$

随着 N_c 的增加，信息素最大值单调递增。且由于 $1>\rho>0$，可以求得：$\lim\limits_{N_c\to\infty}\tau_{ij}^{\max}(N_c)=\dfrac{1}{1-\rho}\dfrac{\sigma}{DEP^{best}(s)}$，其为信息素

可能的最大值。本文取 $MAT^{gb}(c)$ 代替 $DEP^{best}(s)$，可得：

$$\tau_{max}=\frac{1}{1-\rho}\frac{\sigma}{MAT^{gb}(c)} \tag{7}$$

将其近似视为信息素上限。可见 τ_{max} 是一个动态值，当每次获得新的全局最优解时便对 τ_{max} 进行一次更新。参考文献[14]，本算法将 τ_{min} 值近似设置为：

$$\tau_{min}=\frac{\tau_{max}}{2n} \tag{8}$$

式中，n 为待部署虚拟机的数量。

在进行信息素初始化时，若将各个虚拟机与服务器组合信息素设为较小的值，则各部署方案信息素路径之间的差异会随着迭代增加得非常迅速。因而，本文将各个组合的信息素设置为 τ_{max}，以减缓各部署方案间信息素差异，使蚂蚁能够探索到更多的方案。因而在第一次迭代时，将各路径信息素初始化为一个较大的值 $n\cdot m$，求出 τ_{max}；在第二次迭代时，将各路径信息素再次初始化为 τ_{max}，之后进入正常的搜索循环中。

2.5 基于蚁群优化的虚拟机批量部署算法描述

1)算法初始化：获得待部署虚拟机数量 n 和可选目标服务器数量 m。初始化算法参数：β、ρ 以及循环上限 N_c。设蚂蚁个数为 n，分别指定从 n 个虚拟机的位置开始搜索。设初始化迭代循环次数 $c=0$，并将各个路径信息素初始值定为 $\tau(0)=n\cdot m$。

2)每只蚂蚁根据式(2)和式(3)选择其每一步虚拟机与服务器的部署匹配方案，并按照式(5)进行单一信息素更新。直至所有蚂蚁都完成了对 n 个虚拟机的部署，并获得 n 个部署方案。按照式(1)计算出各个部署方案的匹配距离，并通过比较获得最佳部署方案和最小匹配距离值 $MAT^{gb}(c)$。按照式(7)、式(8)分别确定信息素上、下限 τ_{max}、τ_{min}。

3)如果 $c<N_c$，则 $c=c+1$；其他跳转执行步骤 7)

4)如果 $c=1$，则执行步骤 5)；如果 $1<c<N_c$，则跳转执行步骤 6)；

5)将各路径信息素浓度初始化为 $\tau(1)=\tau_{max}$，跳转至步骤 2)。

6)若全局最小匹配距离发生变化，按照式(4)、式(6)进行信息素的全局更新，跳转至步骤 2)

7)退出循环，并输出最优部署方案。

进行一次虚拟机与服务器组合选择时，需要考虑虚拟机与服务器的所有可能组合。一共有 n 只蚂蚁进行 N_c 此循环，因而该算法的时间复杂度为 $O(N_c\times n\times n\times m)$。

3 仿真试验与分析

3.1 算法参数选定分析

本文采用澳大利亚墨尔本大学的网格实验室和 Gridbus 项目提出的云仿真平台 CloudSim[15] 作为试验仿真工具，使用版本为 CloudSim-2.1.1。通过对 VmAllocationPolicy 类中的管理虚拟机放置方法 allocateHostForVm(Vm vm)进行编写，实现虚拟机放置算法，并同时需要相应修改 Vm 和 Datacenter 类。重新编译 CloudSim，获得仿真环境。并在此基础上编写仿真程序。

首先建立一个 Datacenter 包含 10 个服务器(Host)，选取 CPU、内存、硬盘和网络带宽 4 个性能作为匹配距离衡量参数，如表 1 所列。并提交 25 个虚拟机(Vm)部署申请，如表 2 所列。

表 1 服务器列表

ID	CPU/个	CPU/MIPS	内存/Gb	硬盘/Gb	带宽/Mb/s
0	2	2000	4	500	1000
1	2	3000	4	500	1000
2	2	3000	4	750	1500
3	2	4000	4	500	2000
4	2	3000	6	500	1500
5	4	4000	6	1000	1000
6	4	8000	6	1000	1500
7	4	8000	8	1000	1500

表 2 用户提交虚拟机列表

ID	CPU/个	CPU/MIPS	内存/Gb	硬盘/Gb	带宽/Mb/s	性能偏好
0	1	250	0.5	120	400	NVm
1	1	300	0.5	250	300	OVm
2	1	500	1	120	100	CVm
3	1	300	0.5	120	300	CVm
4	1	200	0.5	75	250	OVm
5	1	250	1	250	300	CVm
6	1	300	1	300	400	NVm
7	1	100	1	250	300	SVm
8	2	300	2	120	300	CVm
9	2	250	1	300	200	SVm
10	2	150	2	250	300	CVm
11	2	200	2	120	200	OVm
12	2	300	2	200	100	OVm
13	2	500	2	250	400	OVm
14	2	500	1	120	300	OVm
15	2	400	2	250	300	NVm
16	2	300	1	75	100	NVm
17	2	200	1.5	120	300	OVm
18	2	400	1.5	120	300	CVm
19	2	400	2	75	300	NVm
20	2	300	2	250	400	CVm
21	2	500	2	300	400	SVm
22	2	100	1	120	300	OVm
23	2	200	2	120	300	OVm
24	2	500	2	250	300	OVm

当虚拟机与服务器 CPU 匹配时，首先判断服务器所包含的 CPU 数量是否大于等于虚拟机请求的 CPU 数量，若小于则不考虑进行部署；若大于等于，则取 CPU 数量乘以每个 CPU 的 MIPS 值，将其作为 CPU 综合性能，来计算虚拟机与服务器匹配距离。仿真试验没有考虑最低性能预留，允许性能剩余量为 0 的情况出现。

首先对算法参数设定进行试验分析，采取的方法是每设定一组参数，进行 10 次试验，取 10 次试验中的最优结果、最差结果和平均结果进行比较。

表 3 为参数 β 和挥发系数 ρ 对算法结果的影响。在取相同的 β 时，可以看出，随着 ρ 减小，试验效果有优化的趋势；这个结果可以分析为，本文采取的迭代次数较少，过大的 ρ 值会使路径上信息素挥发速度较慢，在少量的迭代次数中，最优路径与较差路径之间的信息素浓度区别并不明显，使得搜索范围无法围绕在最优解周围；但过小的 ρ 会导致信息素挥发过快，且导致算法过早停滞。取相同 ρ 时，在 $\beta=2$ 处，算法性能取得较好的结果。当 $\beta=0$ 时，质量虽然有所下降，但并不十分明显，在可以接受的范围内。因而在工程实现上可以考虑

将其设置为0，以减少部署算法开销。

表3 算法参数对算法结果的影响(迭代500次)

初始化参数值		匹配距离最优结果	匹配距离平均结果	匹配距离最差结果
$\beta=2$	$\rho=0.9$	12.894	12.981	13.012
$\beta=2$	$\rho=0.7$	12.525	12.743	13.823
$\beta=2$	$\rho=0.6$	12.332	12.413	12.652
$\beta=2$	$\rho=0.5$	11.824	11.937	12.025
$\beta=2$	$\rho=0.3$	12.614	12.714	12.729
$\beta=1$	$\rho=0.5$	12.113	12.249	12.496
$\beta=5$	$\rho=0.5$	12.263	12.366	12.501
$\beta=0$	$\rho=0.5$	12.158	12.209	12.431

3.2 算法比较分析

下面进一步验证本算法的有效性，将其与简单贪婪算法、基本蚁群算法进行比较。本文算法与基本蚁群算法取算法参数 $\beta=2,\rho=0.5$，进行500次迭代。贪婪算法部署流程如下：

(1) 按照本文归一化方法，并按照虚拟机性能偏好设定权值，通过加权运算求出相应服务器各性能之和，从中选出性能最好的主机，并尝试在其上创建虚拟机。

(2) 如果(1)失败了，并且还有主机没有尝试过，就排除当前选择的这台主机，重返部署队列执行(1)。

为了验证本算法的收敛性，首先与基本蚁群算法进行比较。每进行50次迭代本文算法和基本蚁群算法得到的全局最优解情况如图2所示。可以看出，本文算法在收敛速度和防止过早停滞方面明显优于基本蚁群算法，能够获得更好的搜索结果。

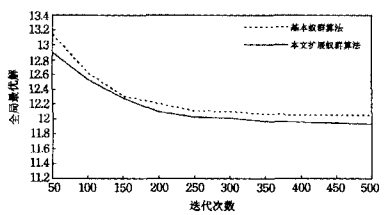

图2 本文扩展蚁群算法与基本蚁群算法收敛情况比较

本文取负载均衡因子 LB 作为系统状态衡量标准，将3种算法的试验结果进行比较。为了更直观地比较，对4个性能特征分别求出各自的负载均衡因子，算法以CPU为例，$LB.Cpu=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Uti_i.Cpu-SysUti.Cpu)^2}$，其中 $Uti_i.Cpu$ 为资源 i 的CPU利用率，$SysUti.Cpu$ 为系统总的CPU利用率，n 为服务器个数。仿真结果如图3所示。

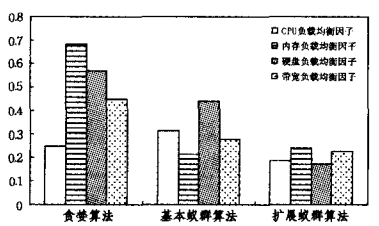

图3 本文算法与贪婪算法、基本蚁群算法比较

由于短板现象的存在，一台服务器某一性能负载过大，就会导致其他性能负载很低，而无法进行利用的现象，因而各个性能之间的负载均衡是影响系统资源利用率的一个重要标准。可以看出，贪婪算法各个性能负载均衡情况都较差，且各性能之间负载情况有很大差异；而基本蚁群算法虽然一定程度上提高了各个性能的负载均衡性，但性能之间的负载差异还是比较大，其资源利用率仍然不够充分。而本文算法在系统各个性能的负载均衡以及性能之间的负载均衡方面都取得了很好的结果，达到了很高的系统负载均衡性和资源利用率。

结束语 本文使用蚁群算法进行虚拟机批量部署，并有针对性地对蚁群算法进行了扩展和改进，提出了具有性能感知的路径选择机制，解决了相同性能偏好虚拟机对于硬件资源的竞争问题；制定了单一蚂蚁信息素更新策略，提高了算法收敛性。通过试验证明，该算法在系统负载均衡和资源利用率方面明显优于现有的贪婪算法和基本蚁群算法，同时，它比基本蚁群算法拥有更好的收敛性能和搜索能力。因此，本文提出的扩展蚁群优化算法可以很好地解决云计算环境下虚拟机批量部署的问题。

参考文献

[1] Gillen A, Broussard F W, Perry R, et al. Optimizing infrastructure: the relationship between it labor costs and best practices for managing the windows desktop[EB/OL]. http://download.microsoft.com/download/a/4/4/a4474b0c-57d8-41a2-afe6-32037fa93ea6/IDC_windesktop_IO_whitepaper.pdf. 2007

[2] Mell P, Grbace T. The NIST Definition of Cloud Computing [R]. National Institute of Standards and Technology, 2011

[3] Amazon. EC2[EB/OL]. http://aws.amazon.com/ec2, 2011

[4] 罗军舟，金嘉辉，宋爱波. 东方云计算：体系架构和关键技术[J]. 通信学报，2009，20(5)：1337-1348

[5] DMTF. Open Virtualization Format Specification[R]. DSP0243, DMTF, 2009

[6] Konstantinou A V, Eilam T, KALANTAR M, et al. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds[C]// VTDC'09. Barcelona: Spain, June 15, 2009

[7] 袁金艳. 多虚拟机快速部署机制的研究[D]. 武汉：华中科技大学，2008(5)

[8] 汤小春，刘健. 基于元区间的云计算基础设施服务的资源分配算法研究[J]. 计算机工程与应用，2010，46(34)：237-241

[9] 雷葆华，饶少阳，江峰，等. 云计算解码：技术架构和产业运营[J]. 电子工业初版社，2011(4)：65-66

[10] Dorigo M, Maniezzo, Colorni A. The ant system: Optimization by a colony of cooperating agents[C]// IEEE Trans. System Man Cybernet. 1996(B26): 29-41

[11] Dorigo M, CARO G D. Ant Colony Optimization: A New Meta-Heuristic[C]// Proceedings of the 1999 Congress on Evolutionary Computation. Washington DC, USA, 1999: 1470-1477

[12] Bullnheimer, Hartl R F, Strauss C. A New Rank-Based Version of the Ant System: A Computational Study[R]. POM-03/97. Institute of Management Science, University of Vienna, Austria. Accepted for publication in the Central European Journal for Operations Research and Economics

[13] Dorigo M, Gambardella L M. Ant Colony System: Acooperative learning approach to the traveling salesman problem[J]. IEEE Transactions on Evolutionary Computation, 1997, 1(1): 53-66

[14] Stutzle T, Hoos H. Improvements on the ant system: introducing MAX-MIN ant system[C]// Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms. Springer Verlag, Wien, 1997: 245～249

[15] Calheiros R N, Ranjan R, De Rose C A F, et al. loudsim: A novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services [R]. GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory. The University of Melbourne, Australia, March 13, 2009