基于蚂蚁群体智能的多目标虚拟机合并优化

李玉萍 陈丽娜
(商丘师范学院信息技术学院 河南 商丘 476000)

摘 要 为了优化数据中心中虚拟机的合并过程,提高物理主机利用率和降低虚拟机的迁移代价,利用蚂蚁群体智能方法提出一种新的多目标虚拟机合并算法。该算法基于重要性按序优化了两个目标,第一目标是最大化虚拟机合并过程中的主机释放量。同时,由于虚拟机迁移是资源密集型操作,第二目标选择最小化虚拟机迁移量。通过修正的蚂蚁搜索过程,最终得到了最优的虚拟机合并效果。与两种代表性蚂蚁算法进行实验对比,结果表明,在所有四个实验场景下,新算法在多数场景和参数配置条件下得到的主机释放量、虚拟机迁移量、包装效率以及算法运行时间均有更佳表现。

关键词 虚拟机合并 多目标优化 虚拟机迁移 蚂蚁种群

中图分类号 TP393 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.08.041

MULTI-OBJECTIVE VIRTUAL MACHINE CONSOLIDATION OPTIMIZATION BASED ON ANT COLONY INTELLIGENT

Li Yuping Chen Lina
(*School of Information Technology, Shangqiu Normal University, Shangqiu 476000, Henan, China*)

Abstract In order to optimizate virtual machines consolidation process in data center, improve the physical hosts utilization and reduce the virtual machines migraiton cost, a novel multi-objective virtual machines consolidation algorithm using ant colony intelligent is designed in this paper. It optimizes two objectives that are ordered by their importance. The first and foremost objective is the proposed algorithm is to maximize the number of released physical hosts. Moreover, since virtual machine migration is a resource-intensive operation, is also tries to minimize the number of virtual machine migration. Our algorithm finally obtains the optimal virtual machine consolidation effect through modified ant search process. Some constrast experiments are carried on with the other two kinds of typical ant algorithms. The experimental results show that, in all four test scenarios, under the condition of most scenarios and parameter configuration, our new algorithm has a better performance on the number of released physical hosts, the number of virtual machine migration, the packing efficiency and the algorithm running time.

Keywords Virtual machine consolidation Multi-objective optimization Virtual machine migration Ant colony

0 引 言

数据中心由数以万计的物理主机 PM 组成,这些物理资源会导致大量能耗,不仅会增加运营成本,还会导致巨大的碳排放[1]。同时,数据中心的高能耗主要是由资源利用不充分导致的[2]。虚拟化使性能独立的虚拟机 VM 可共享同一主机,以改善资源利用率,而虚拟机合并及空闲主机的休眠或低功耗转换则可以进一步改善资源利用率和能耗。虚拟机合并是改善数据中心能效的最有效方式[3],利用动态的虚拟机迁移的合并技术可以将虚拟机部署于数量更少的主机上。虚拟机合并主要通过迁移手段实现,在不同主机间的虚拟机迁移可以关闭闲置主机[4]。虚拟机合并的性能度量指标主要包括:释放的主机数量(需最大化)、虚拟机迁移次数(需最小化)以及算法运行时间(需最小化)。

收稿日期:2019-01-03。国家自然科学基金项目(11501345);河南省高等学校重点科研项目(19B520023)。**李玉萍**,讲师,主研领域:人工智能,大数据处理。**陈丽娜**,副教授。

虚拟机合并问题本身是 NP 问题,求解技术如混合整数线性规化方法[5],但其寻优时间达到指数级。其次是元启发式方法,可在多项式时间复杂度内有效搜索可行解空间。

自适应元启发式蚂蚁群体智能优化方法是求解虚拟机合并的有效方法,已有的研究多集中于通过单目标单群体的聚合目标函数 AOF 来联立多目标[6],AOF 方法的好处在于可以降低时间复杂度,通过限制可行解的子空间搜索来改善算法运行效率。然而,该方法的不妥之处在于无法准确地赋予各个目标需要考虑的权重,仅能通过主观意识赋值。且 AOF 可能无法以恰当的方式联立优化目标。已有的蚂蚁群体虚拟机合并算法的 AOF 均未对优化目标进行优先关系排序。

本文设计了一种最大化主机释放量且虚拟机迁移量最小的虚拟机迁移方法实现虚拟机合并,其主要贡献在于:1) 实现了多目标多群体优化。算法扩展已有虚拟机合并单目标单群体优化思想,实现多目标多群体的蚂蚁群体优化。2) 降低了搜索空间。算法通过设置约束在未降低解质量前提下排除了合并过程中的部分迁移方案,使得算法更快收敛,可扩展性增强。

1 相关工作

蚂蚁群体优化 ACO 启发式方法受蚂蚁觅食行为启发,其从食物源至蚂蚁巢的食物运输会跟随路径上的信息素这类化学物质进行追踪。该方法允许蚂蚁之间直接通信以寻找起始点最优的路径。虽然每个蚂蚁可以找到一个可行解,但高质量解是来源于间接通信和蚂蚁之间的全局协作得到的结果。ACO 算法的类型包括蚂蚁系统 AS、蚂蚁群体系统 ACS 以及最大最小蚂蚁系统 MMAS,其中,ACS 是当前最优的解决方案。已有虚拟机部署和合并研究中,文献[7]利用 ACO 解决了非线性资源分配问题,得到受限资源下任务的最优分配。文献[8]利用改进的 ACO 进行多目标资源分配求解。文献[9]利用单目标单群体 MMAS 算法求解虚拟机合并问题。文献[10]将向量代数机制整合到 ACS 中,优化了资源利用率。文献[11-12]设计了单目标单群体 ACS 算法进行虚拟机合并求解。文献[13]利用多目标 ACS 解决两个目标优化问题:能耗最小和资源浪费最小,但方法并没有进行虚拟机迁移,仅在初始部署考虑优化目标。

已有 ACO 的虚拟机合并方法与本文算法的不同在于:已有方法趋向于利用 AOF 的单目标单群体 ACO 算法尝试联立多目标,而本文算法直接利用相互独立的两个蚂蚁群体设计多目标的 ACS 算法。群体一最大化释放主机数量,群体二最小化虚拟机迁移量。并且在解空间的搜索上通过设计三种约束进行了优化,极大地降低了最优解的搜索时间。

2 多目标虚拟机合并蚂蚁算法 MOACS

2.1 基本说明

本文提出一种多目标虚拟机合并蚂蚁算法 MOACS,算法的第一个目标是最大化释放主机的数量 $|P_R|$,第二个目标是最小化虚拟机迁移数量 nM。由于拥有更大主机释放数量的全局最优迁移方案 Φ^+ 总是优于较低主机释放量的迁移方案 Φ^+ 的,即使该方案中的虚拟机迁移量大于后一迁移方案,即最大化 $|P_R|$ 优先于最小化 nM。本文用到的其他相关符号及说明如表 1 所示。

表 1 符号说明

符号	参数含义
M	虚拟机迁移方案集合
P	物理主机集合
P_R	执行迁移方案 Φ 时释放的主机集合
T_k	未被蚂蚁 k 穿过的元组集合
V	虚拟机集合
$C_{p_{\mathrm{de}}}$	目标主机 p_{de} 的总体能力向量
N	主机的邻居
p_{de}	元组中的目标主机
p_{so}	元组中的源主机
q	均匀分布的随机变量
S_{crk}	迄今为止蚂蚁 k 的最优得分
$U_{p_{\mathrm{de}}}$	目标主机 p_{de} 已利用的能力向量
$U_{p_{\mathrm{so}}}$	源主机 p_{so} 已利用的能力向量
U_v	虚拟机 v 已利用的能力向量
η	启发值
τ	信息素数量
τ_0	初始信息素数量
Φ	虚拟机迁移方案
Φ^+	全局最优虚拟机迁移方案
Φ^+_{nM}	ACS_{nM} 得到的迄今为止最优的迁移方案
$\Phi^+_{P_R}$	$ACS_{\|P_R\|}$ 得到的迄今为止最优的迁移方案
Φ_k	蚂蚁 k 定义的迁移方案
Φ^m_k	蚂蚁 k 定义的临时迁移方案
$\Delta^{nM}_{\tau_s}$	Φ^+_{nM} 中的元组附加信息素数量
$\Delta^{P_R}_{\tau_s}$	$\Phi^+_{P_R}$ 中的元组附加信息素数量
β	决定 η 相对重要性的参数
ρ	局部更新规则中信息素的衰变参数
q_0	决定搜索相对重要性的参数
nA	同时建立迁移方案的蚂蚁数量
nI	蚂蚁世代的数量

2.2　算法基本原理

MOACS 算法结构如图 1 所示,算法通过协调两个不同单目标优化种群的运行,实现最终双目标的同步优化。优化主机释放量通过群体 $ACS_{|P_R|}$ 进行优化,优化虚拟机迁移量通过群体 ACS_{nM} 进行优化。两个群体独立运行并利用独立的信息素和启发值,但两个群体会协作产生全局最优迁移方案 Φ^+。

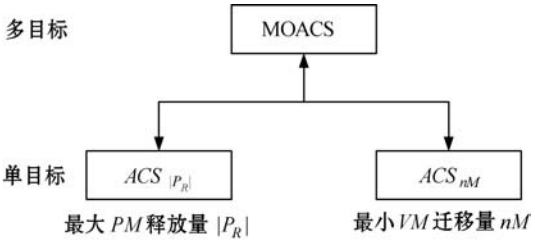

图 1　算法结构

虚拟机合并中,每个主机 p 上可寄宿多个虚拟机 v。至少寄宿一个虚拟机的主机称为潜在源主机,主机和虚拟机均具有资源利用率属性,包括 CPU 利用率和内存利用率。MOACS 算法还引入了主机邻居概念。邻居表示相互排斥的 P 的子集。一个邻居是一个抽象实体,代表数据中心中位于邻近位置的主机集合,那么,数据中心的主机可以构成主机的一个邻居。一个虚拟机可被迁移至任意邻居 N 的其他主机上。因此,源主机的邻居内的和以外的其他主机也是一个潜在目标主机,也具有资源利用率属性。基于此,MOACS 算法建立一个元组集合 T,每个元组 $t \in T$ 由三个元素组成:源主机 p_{so}、虚拟机 v 及目标主机 p_{de}。

$$T = (p_{so}, v, p_{de}) \tag{1}$$

MOACS 算法如算法 1 所示。初始时,全局最优迁移方案 Φ^+ 未知,Φ^+ 为空集(步骤 2)。步骤 1 的主循环过程迭代至停止条件,即所有剩余主机均充分利用或在给定的连续迭代次数到达时无法进一步改进性能为止。每一次迭代中,两个群体 $ACS_{|P_R|}$ 和 ACS_{nM} 试图根据各自目标寻找全局最优迁移方案 Φ^+。步骤 3 中,$ACS_{|P_R|}$ 寻找拥有更高主机释放量的迁移方案,步骤 4 中,ACS_{nM} 寻找拥有更少虚拟机迁移量的迁移方案。当寻找到一个改进迁移方案时全局最优迁移方案 Φ^+ 被更新。如果 $ACS_{|P_R|}$ 找到一个拥有更高主机释放量的迁移方案(步骤 5),Φ^+ 根据 $ACS_{|P_R|}$ 中得到的迄今为止最优迁移方案进行更新,即 $\Phi^+_{P_R}$(步骤 6)。同样地,当 ACS_{nM} 找到一个拥有更低虚拟机迁移量且与 Φ^+ 中至少一样的主机释放量的迁移方案时(步骤 7 - 步骤 8),Φ^+ 根据 ACS_{nM} 中得到的迄今为止最优迁移方案进行更新,即 Φ^+_{nM}(步骤 9)。最后,在主循环中每次迭代结束时,虚拟机将根据全局最优迁移方案 Φ^+ 进行合并,释放的主机将终止,即步骤 10。

算法 1　Multi-objective Ant Colony System-MOACS 算法

1. While until a stopping criterion is met do
2. $\Phi^+ = \varnothing$　　//全局最优迁移解赋空集
3. $\Phi^+_{P_R} = ACS_{|P_R|}$　　//得到 $ACS_{|P_R|}$ 算法的迁移解
4. $\Phi^+_{nM} = ACS_{nM}$　　//得到 ACS_{nM} 算法的迁移解
5. if $\Phi^+ = \varnothing \vee f(\Phi^+_{P_R}) > f(\Phi^+)$ then　　//$ACS_{|P_R|}$ 解 PM 释放量占优
6. $\Phi^+ = \Phi^+_{P_R}$　　//更新 Φ^+
7. if $f(\Phi^+_{nM}) \geqslant f(\Phi^+)$ then　　//ACS_{nM} 解 PM 释放量至少占优
8. if $g(\Phi^+_{nM}) > g(\Phi^+)$ then　　//ACS_{nM} 解 VM 迁移量占优
9. $\Phi^+ = \Phi^+_{nM}$　　//更新 Φ^+
10. consolicate VMs according to Φ^+ and terminate released PM
11. end while

算法 1 的时间复杂性分析。算法的计算时间主要取决于元组 $|T|$ 的数量,为了降低计算时间,MOACS 设计了三种约束,可通过移除部分元组降低搜索空间。第一重约束确保仅未充分利用的主机可作为源主机,第二重约束仅允许未充分利用主机可作为目标主机,即排除从利用充分的主机上迁移和迁移至充分利用的主机上两种情况,这可以防止已利用充分的主机出现超载。而从利用充分主机上进行虚拟机迁移也不可能导致源主机的终止,进而不会降低总主机请求数量。第三重约束通过防止邻居内迁移进一步限制 $|T|$ 的大小。因此,作为通用规则,一个虚拟机仅能被迁移至其源主机的邻居内的主机上。该规则唯一例外是一个邻居仅有一个主机,此时,该主机上的虚拟机可被迁移至任意邻居的任意主机上。通过这三种规则的定义与利用,可以在不降低解质量的同时极大地缩短算法的计算时间。

算法 1 的空间复杂度分析。MOACS 算法空间复杂度为 $O(|T|)$。而且,最差空间复杂度对应于具体的虚拟机合并场景,与主机的利用程度无关。此时,每个主机可视为一个目标主机。最差情况下元组的最大数量计算为:

$$\max|T| = |P||V|(|N| - 1) \tag{2}$$

式中:$|P|$ 为主机数量,$|V|$ 为虚拟机数量,$|N|$ 为邻居大小。由于实际的虚拟机合并场景中通常包括一个或多个充分利用的主机,而所提算法可以排除从该类主机进行的迁移,故元组数量 $|T|$ 通常小于最差场景中的数量。

2.3 基于 PM 释放量最大化的优化算法 $ACS_{|PR|}$

$ACS_{|P_R|}$的目标是优化主机释放量$|P_R|$，即算法目标函数为：

$$\max f(\Phi) = |P_R| \tag{3}$$

由于虚拟机合并的首要目标是最小化活动主机的数量，目标函数的定义将根据主机释放量进行定义。且当执行迁移方案时，将利用一种约束，通过将迁移限制在仅为释放主机 P_R 集合以外的主机上，以降低虚拟机迁移量 nM，即：

$$p_{\text{de}} \in P | p_{\text{de}} \notin P_R \tag{4}$$

算法中，当其上的所有虚拟机被迁移后，该主机才视为已释放主机。因此，释放主机集合 P_R 可定义为：

$$P_R = \{p \in P | V_p = \varnothing\} \tag{5}$$

式中：V_p 表示运行于主机 p 上的虚拟机集合。因此，当不再寄宿任意虚拟机时，一个主机仅包含于释放主机集合 P_R 中。

虚拟机合并问题中，蚂蚁根据式(1)定义的元组存放信息素。nA 个蚂蚁中的任一蚂蚁利用随机状态转换规则选择可穿过的下一个元组，$ACS_{|P_R|}$中的状态转换规则即为伪随机比例规则。根据该规则，蚂蚁 k 选择下一元组的规则为：

$$s = \begin{cases} \text{argmax}_{u \in T_k}\{[\tau_u] \cdot [\eta_u]^\beta\} & q \leqslant q_0 \\ S & \text{其他} \end{cases} \tag{6}$$

式中：argmax 返回$[\tau][\eta]^\beta$得到其最大值的元组，$q \in [0,1]$为均匀分布随机变量，$q_0 \in [0,1]$为一个参数，S 为根据式(7)的概率分布选择的随机变量，蚂蚁 k 选择元组 s 穿越的概率 $prob_s$ 定义为：

$$prob_s = \begin{cases} \dfrac{[\tau_s] \cdot [\eta_s]^\beta}{\sum\limits_{u \in T_k} [\tau_s] \cdot [\eta_s]^\beta} & s \in T_k \\ 0 & \text{其他} \end{cases} \tag{7}$$

元组 s 的启发值 η_s 定义为：

$$\eta_s = \begin{cases} \dfrac{(U_{p_{\text{de}}} + U_v)}{C_{p_{\text{de}}}} & U_{p_{\text{de}}} + U_v \leqslant C_{p_{\text{de}}} \\ 0 & \text{其他} \end{cases} \tag{8}$$

可知，拥有最小未利用能力的目标主机将获得最高启发值。因此，启发值偏向于导致主机利用不足降低的虚拟机迁移，而 $U_{p_{\text{de}}} + U_v \leqslant C_{p_{\text{de}}}$可防止虚拟机迁移带来目标主机 p_{de} 的超载。

式(6)和式(7)中的随机状态转换规则偏向于选择拥有更高信息素浓度的元组。式(6)中 $q \leqslant q_0$ 的第一种情况选择能够得到$[\tau][\eta]^\beta$ 最大值的元组，而第二种情况则根据式(7)选择元组。第一种情况有助于蚂蚁快速的收敛于高质量解，而第二种情况则有助于蚂蚁通过基于宽度的搜索空间拓展避免蚂蚁的搜索停滞。除了随机状态转换规则，$ACS_{|P_R|}$还利用了全局和局部信息素踪迹蒸发规则。全局信息素踪迹蒸发规则利用于所有蚂蚁完成其迁移的一次迭代的结束时，定义为：

$$\tau_s = (1-\alpha) \cdot \tau_s + \alpha \cdot \Delta_{\tau_s}^{P_R} \tag{9}$$

式中：$\alpha \in (0,1]$为信息素衰变参数，$\Delta_{\tau_s}^{P_R}$为附加信息素量，仅给予 $ACS_{|P_R|}$中迄今为止最优迁移方案中的元组，即 $\Phi_{P_R}^+$，定义为：

$$\Delta_{\tau_s}^{P_R} = \begin{cases} |P_R| & s \in \Phi_{P_R}^+ \\ 0 & \text{其他} \end{cases} \tag{10}$$

局部信息素踪迹更新规则应用于当蚂蚁穿越该元组做出自身的迁移方案时的元组，定义为：

$$\tau_s = (1-\rho) \cdot \tau_s + \rho \cdot \tau_0 \tag{11}$$

式中：$\rho \in (0,1]$类似于 α，τ_0 表示初始信息素，计算为主机数量$|P|$与近似最优解$|\Phi|$的乘积的逆值，即：

$$\tau_0 = (|\Phi| \cdot |P|)^{-1} \tag{12}$$

$ACS_{|P_R|}$中伪随机比例规则和全局信息素踪迹更新规则可以使搜索更有指向性，伪随机比例规则偏向于选择更高信息素和更高启发值的元组，因此，蚂蚁可以在迄今全局最优解的极邻近区域搜索高质量解。此外，局部信息素踪迹更新规则可补充搜索与全局最优解相距较远区域中的其他高质量解。这是由于无论何时蚂蚁穿越一个元组，应用局部信息素更新规则，元组信息素均会损耗，对于其他蚂蚁的吸引会变差。因此，这有助于避免所有蚂蚁停滞于搜索相同解或过早收敛于子最优解。

$ACS_{|P_R|}$算法过程如算法 2 所示。算法利用式(1)产生元组集合 T，并利用式(12)设置元组信息素为初始信息素 τ_0(步骤 2)。然后，算法迭代运行 nI 次(步骤 3)，每次迭代 i 建立新一代 nA 个蚂蚁，同时建立其迁移方案(步骤 4 - 步骤 20)。每个蚂蚁 k 在$|T|$个元组上进行迭代(步骤 6 - 步骤 18)。通过式(7)计算选择下一元组穿越的概率(步骤 7)，然后，基于式(6)和式(7)计算概率和随机状态转换规则，每个蚂蚁选择一个元组 t 进行穿越(步骤 8)，并添加 t 至其暂时迁移方案 Φ_k^m 中(步骤 9)。在 t 上应用式(11)和式(12)的局部信息素踪迹更新规则(步骤 10)。如果 t 表示的迁移没有使目标主机 p_{de}发生超载，则源主机 $U_{p_{\text{so}}}$和 t 中的目标主机 $U_{p_{\text{de}}}$ 的已利用能力向量将根据迁移进行更新(步骤12)。然后，式(3)的目标函数将应用于 Φ_k^m 上进行计算。如果获得的目标值高于蚂蚁迄今为止的最优值 S_{crk}(步骤 13)，则将 t 添加至蚂蚁定义的迁移

方案 Φ_k 中(步骤 15)。然后,当所有蚂蚁完成迁移方案时,迁移方案被添加至 M 中(步骤 16),每个迁移方案 Φ_k 将利用式(3)的目标函数进行评估,并选择迄今为止得到的全局最优方案作为 Φ^+(步骤 17),且利用式(9)和式(10)中的全局信息素踪迹更新规则对所有元组进行更新(步骤 18)。最后,当所有迭代 i 完成后,$ACS_{|P_R|}$ 返回迄今为止得到的最优方案 $\Phi^+_{P_R}$(步骤 19)。

算法 2　$ACS_{|P_R|}$ 算法

1. $\Phi^+_{P_R}=\varnothing, M=\varnothing$　　//初始化
2. $t\in T|\tau_t=\tau_0$　　//元组信息素初始化
3. for $i\in[1,nI]$ do
4. for $k\in[1,nA]$ do
5. while $|\Phi^m_k|<|T|$ do
6. $\Phi^m_k=\varnothing, \Phi_k=\varnothing, S_{crk}=0$
7. compute $prob_s$ by using (7)
　　//计算蚂蚁的穿越概率
8. choose a tuple $t\in T$ to traverse by using (6)
9. $\Phi^m_k=\Phi^m_k\cup\{t\}$　　//更新临时迁移方案
10. apply local update rule in (11) on t //更新元组
11. if the migration in t does not overload destination PM p_{de} then　　//元组的迁移未导致主机超载
12. update used capacity vectors $U_{p_{so}}$ and $U_{p_{de}}$ in t
13. if $f(\Phi^m_k)>S_{crk}$ then
14. $S_{crk}=f(\Phi^m_k)$
15. $\Phi_k=\Phi_k\cup\{t\}$　　//更新蚂蚁 k 的迁移方案
16. $M=M\cup\{\Phi_k\}$
17. $\Phi^+_{P_R}=\arg\max_{\Phi k\in M}\{f(\Phi_k)\}$
　　//基于式(3)得到全局最优方案
18. apply global update rule in (9) on all $s\in T$
　　//更新所有元组
19. return $\Phi^+_{P_R}$

算法 2 的时间复杂度分析。$ACS_{|P_R|}$ 的时间复杂度为 $O(nI\times|T|^2)$,nI 表示蚂蚁代数,$|T|$ 表示元组数量。从算法 2 可知,步骤 3 的主循环需要迭代 nI 次,步骤 4 的第二重循环并不会增加时间复杂度,由于蚂蚁会同步建立其迁移方案。步骤 5 的循环需要迭代 $|T|$ 次,步骤 7 的概率计算需要在 $|T|$ 上进行迭代。综上可得算法的时间复杂度为 $O(nI\times|T|^2)$。

2.4　基于 VM 迁移量最小化的优化算法 ACS_{nM}

ACS_{nM} 的目标寻找虚拟机迁移量更少且至少与 Φ^+ 中主机释放量 P_R 相当的迁移方案,即算法目标函数为:

$$\max\ g(\Phi)=(nM)^{-1} \tag{13}$$

由于虚拟机迁移是资源密集型操作,ACS_{nM} 的目标函数可定义为迁移量的逆值。

ACS_{nM} 中的蚂蚁同样利用式(6)和式(7)的伪随机比例规则选择需穿越的下一元组。且式(8)中的启发值更偏向于拥有更多的已利用能力向量的虚拟机 U_v。这样,更可能导致虚拟机迁移量降低的虚拟机迁移操作将得到更高的启发值。因此,式(8)中的启发值支持 ACS_{nM} 中式(13)定义的目标函数。

ACS_{nM} 群体同样利用式(11)中的局部信息素踪迹更新规则,然而,全局信息素踪迹更新规则需要重新定义为:

$$\tau_s=(1-\alpha)\cdot\tau_s+\alpha\cdot\Delta^{nM}_{\tau_s} \tag{14}$$

式中:$\Delta^{nM}_{\tau_s}$ 表示附加信息素量,仅给予 ACS_{nM} 中迄今为止的最优迁移方案中的元组,即 Φ^+_{nM},定义为:

$$\Delta^{nM}_{\tau_s}=\begin{cases}(nM)^{-1} & s\in\Phi^+_{nM}\\ 0 & \text{其他}\end{cases} \tag{15}$$

ACS_{nM} 算法过程如算法 3 所示。多数 ACS_{nM} 算法步骤与 $ACS_{|PR|}$ 相似(步骤 1 - 步骤 12)。步骤 13 利用式(3)定义的主机释放量目标函数取代式(13)的虚拟机迁移量目标函数。然而,当步骤 17 从 ACS_{nM} 中选择迄今为止的最优迁移方案时(即 Φ^+_{nM}),所有蚂蚁定义的迁移方案 $\Phi_k\in M$ 将首先利用式(3)定义的主机释放量目标函数进行评估,然后利用式(13)定义的虚拟机迁移量目标函数进行评估。因此,拥有最多主机释放量和更少虚拟机迁移量的迁移方案将被选择为最优迁移方案 Φ^+_{nM}。此后,算法将利用式(14)和式(15)中的全局信息素踪迹更新规则在所有元组上进行信息素更新,即步骤 18。最终,算法返回 Φ^+_{nM},即步骤 19。

算法 3　ACS_{nM} 算法

1. $\Phi^+_{P_R}=\varnothing, M=\varnothing$　　//初始化
2. $t\in T|\tau_t=\tau_0$　　//元组信息素初始化
3. for $i\in[1,nI]$ do
4. for $k\in[1,nA]$ do
5. while $|\Phi^m_k|<|T|$ do
6. $\Phi^m_k=\varnothing, \Phi_k=\varnothing, S_{crk}=0$
7. compute $prob_s$ by using (7)
　　//计算蚂蚁的穿越概率
8. choose a tuple $t\in T$ to traverse by using(6)
9. $\Phi^m_k=\Phi^m_k\cup\{t\}$//更新临时迁移方案
10. apply local update rule in (11) on t//更新元组
11. if the migration in t does not overload destination PM p_{de} then　　//元组的迁移未导致主机超载
12. update used capacity vectors $U_{p_{so}}$ and $U_{p_{de}}$ in t
13. if $f(\Phi^m_k)>S_{crk}$ then
14. $S_{crk}=f(\Phi^m_k)$
15. $\Phi_k=\Phi_k\cup\{t\}$　　//更新蚂蚁 k 的迁移方案

16. $M = M \cup \{\Phi_k\}$

17. $\Phi_{nM}^{+} = \arg\max_{\Phi_k \in M}\{f(\Phi_k)\} \wedge \arg\max_{\Phi_k \in M}\{g(\Phi_k)\}$

//基于式(13)得到全局最优方案

18. apply global update rule in (14) on all $s \in T$

//更新所有元组

19. return Φ_{nM}^{+}

算法 3 的时间复杂度分析。ACS_{nM}的时间复杂度与 $ACS_{|P_R|}$相似,由于 ACS_{nM}与 $ACS_{|P_R|}$两个群体均是同步且独立运行。

3 仿真实验

3.1 实验说明

本节测试 MOACS 的性能,对比算法选取同为蚁群系统的优化,包括单目标单群体的虚拟机合并算法 Feller-ACO[9] 及 ACS[8] 算法。虚拟机合并的输入参数包括:主机数量、合并虚拟机的数量、虚拟机的 CPU 利用率、虚拟机的内存需求以及每个虚拟机的当前位置。为了观察算法可扩展性和适应性,在主机上建立四种不同场景测试:1) 低 CPU 低内存请求;2) 高 CPU 大内存请求;3) 高 CPU 小内存请求;4) 低 CPU 大内存请求。实验利用随机负载、同质虚拟机和同质主机进行测试。实验参数见表 2 和表 3。对于场景 1),合并的虚拟机数量设置为 1 000,主机数量为 100,比例为 10∶1,其他三种场景设置 1 000 个虚拟机和 200 个主机,比例为 5∶1。邻居规模设置为 5,邻居随机选择。表 3 是与蚁群智能优化过程相关的参数设置。实验中需要考虑的指标包括:合并后最大化的释放主机数量、包装效率(释放的主机数量与总主机数量的比值)、合并后最小化的虚拟机迁移量以及算法的求解时间。

表 2 场景设计

测试算法		CPU 利用率	
		低/Low	高/High
内存	小/Small	场景 1 |V| = 1 000 |P| = 100 |N| = 5 运行次数 = 10	场景 3 |V| = 1 000 |P| = 200 |N| = 5 运行次数 = 10
	大/Large	场景 4 |V| = 1 000 |P| = 200 |N| = 5 运行次数 = 10	场景 2 |V| = 1 000 |P| = 200 |N| = 5 运行次数 = 10

表 3 蚂蚁群体参数设置

α	β	ρ	q_0	nA	nI
0.1	2.0	0.1	0.9	10	2

3.2 实验结果

1) 释放主机量与包装效率。

图 2 是算法在不同实验下得到的释放主机数量的箱形图,表 4 是相应的数值结果,包括包装效率。结果表明,MOACS 算法可以比 Feller-ACO 多释放 25% ~ 37% 的主机。在场景 1 中,MOACS 释放了 15 个主机(10 次测试的中位值),而 Feller-ACO 仅释放 11 个主机。由于包装效率需由释放主机量推导,该指标也具有类似的趋势。

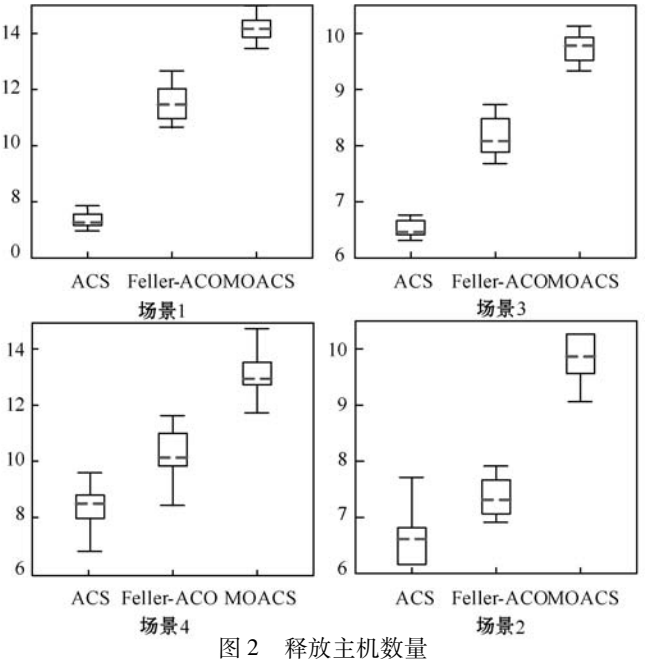

图 2 释放主机数量

表 4 释放主机量

场景	参数	ACS	MOACS	Feller-ACO	Change	p-value
1	中位值	9	15	11	136%	0.005
	标准差	0.63	0.52	0.82	–	–
	包装效率	9%	15%	11%	–	–
2	中位值	7	11	8	137%	0.005
	标准差	0.67	0.67	0.99	–	–
	包装效率	3.5%	5.5%	4%	–	–
3	中位值	7	10	8	125%	0.004
	标准差	0.67	0.52	0.82	–	–
	包装效率	3.5%	5%	4%	–	–
4	中位值	7	11	8	137%	0.005
	标准差	0.88	0.88	0.74	–	–
	包装效率	3.5%	5.5%	4%	–	–

2）VM 迁移量。

图 3 是算法得到虚拟机迁移量结果,表 5 同步给出数值结果。可以看出,MOACS 在此目标上也是优于 Feller-ACO 的,MOACS 仅利用 Feller-ACO 虚拟机迁移量的 82% 即可得到更好的包装效率。在场景 1 中,MOACS 拥有 189 次虚拟机迁移,而 Feller-ACO 拥有 226 次迁移。

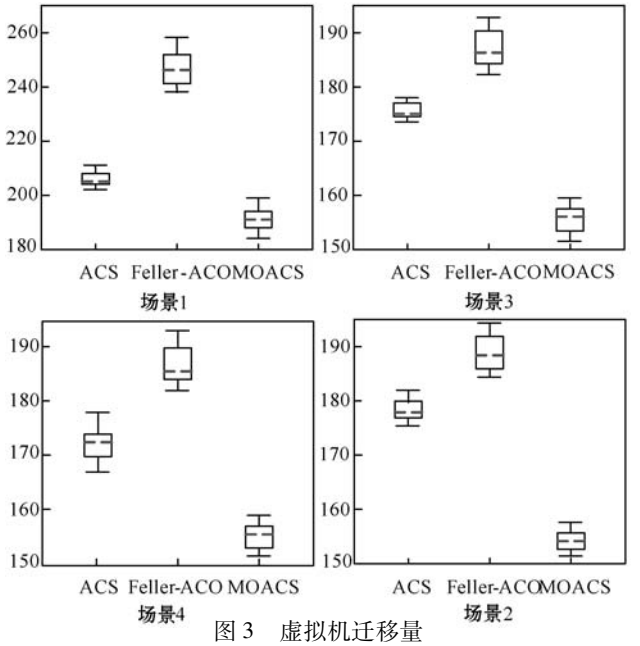

图 3　虚拟机迁移量

表 5　虚拟机迁移量

场景	参数	ACS	MOACS	Feller-ACO	Change	p-value
1	中位值	9	15	11	136%	0.005
	标准差	0.63	0.52	0.82	–	–
	包装效率	9%	15%	11%	–	–
2	中位值	7	11	8	137%	0.005
	标准差	0.67	0.67	0.99	–	–
	包装效率	3.5%	5.5%	4%	–	–
3	中位值	7	10	8	125%	0.004
	标准差	0.67	0.52	0.82	–	–
	包装效率	3.5%	5%	4%	–	–
4	中位值	7	11	8	137%	0.005
	标准差	0.88	0.88	0.74	–	–
	包装效率	3.5%	5.5%	4%	–	–

3）执行时间与可扩展性。

该指标度量算法搜索全局最优解的执行时间。图 4 是基于场景 2 三种算法得到的结果,此时设置的主机数量由 50 至 500 以步长 50 递增,虚拟机数量由 250 至 2 500 以步长 250 递增。可以看出,MOACS 优于 Feller-ACO,而 ACS 是优于 MOACS 的,由于其仅是单目标优化,搜索时间更短,而 MOACS 需要在两个目标上进行搜索。

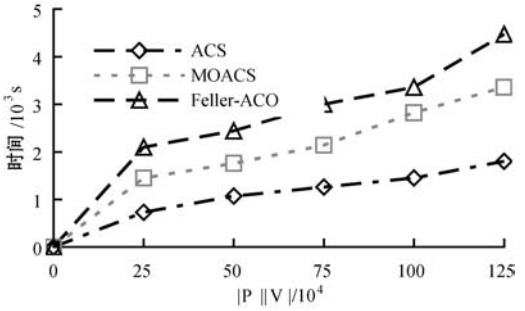

图 4　算法运行时间

进一步,实验在场景 1 下利用 $|P| = 1\ 000$ 和 $|V| = 1\ 000$ 进行了比较分析,如表 6 所示。该场景下,ACS 得到的时间多少 1 分钟,比较而言,MOACS 几乎运行 2 分种,而 Feller-ACO 需要 6 分种。同时还观察到,时间变量的标准差均是较小的,前两种算法的时间差则差别较大。

表 6　场景 1 下的算法执行时间

|P||V|	参数	ACS/min	MOACS/min	Feller-ACO/min
10^5	中值	1.03	1.99	5.92
	标准差	0.03	0.08	0.17

4）结论分析。

总体来看,MOACS 在所有测试指标下均优于 Feller-ACO,包括释放的主机量、包装效率以及虚拟机迁移量和执行时间。Feller-ACO 利用单目标单群体下 AOF,联合了两种不同的目标,包括释放主机量和虚拟机迁移量,MOACS 利用多目标算法,考虑的是两个独立的蚂蚁群体分别进行优化。Feller-ACO 中的 AOF 利用多个参数决定全局优化过程两个目标的相对重要性,该方法的不足在于:1）无法准确找到 AOF 中合适的参数值;2）AOF 可能无法做到不同目标间的合理联立。例如:MOACS 中最大化释放主机量是占优于最小化虚拟机迁移量的,而 Feller-ACO 中的 AOF 不支持目标间的占优关系。此外,MOACS 采用了搜索空间的约束机制(三种约束),可以极大地降低算法的运行时间,且不会影响到解的质量。同时,MOACS 与 ACS 相比,拥有更多的主机释放量和更少的虚拟机迁移量,但其运行较 ACS 更慢。比较 ACS 与 Feller-ACO,ACS 更慢且虚拟机迁移量更少,尽管该算法没有实现相同的包装效率。

4　结　语

本文提出了一种新的多目标蚂蚁群体算法进行数

据中心的虚拟机合并,算法可以建立虚拟机迁移方案,通过利用不充分的主机上的虚拟机迁移和合并降低物理主机的请求量。在此过程中,目标函数设置为最大化主机释放量与最小化虚拟机迁移量。以大量的实验对算法进行了验证,并与两种已有的蚂蚁优化算法作对比,所提算法在多种实验场景下的性能均是较优的。进一步研究算法实际的能效,本文仅从优化主机使用数量与虚拟机迁移次数上着手优化虚拟机合并,但虚拟机的具体部署结果也会对能效产生影响,此时需要计算具体的部署时的能效问题,然后再设计能效更高的部署算法。

参 考 文 献

[1] Dayarathna M, Wen Y, Fan R. Data Center Energy Consumption Modeling: A Survey[J]. IEEE Communications Surveys & Tutorials, 2017, 18(1):732 - 794.

[2] Beloglazov A, Buyya R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers[J]. Concurrency & Computation Practice & Experience, 2012, 24(13):1397 - 1420.

[3] Farahnakian F, Ashraf A, Pahikkala T, et al. Using Ant Colony System to Consolidate VMs for Green Cloud Computing[J]. IEEE Transactions on Services Computing, 2015, 8(2):187 - 198.

[4] Corradi A, Fanelli M, Foschini L. VM consolidation: A real case based on OpenStack Cloud[J]. Future Generation Computer Systems, 2014, 32(1):118 - 127.

[5] Hwang I, Pedram M. Hierarchical Virtual Machine Consolidation in a Cloud Computing System[C]// IEEE 6th International Conference on Cloud Computing. IEEE, 2013:196 - 203.

[6] Ahmad R W, Gani A, Hamid S H A, et al. A survey on virtual machine migration and server consolidation frameworks for cloud data centers[J]. Journal of Network & Computer Applications, 2015, 52(C):11 - 25.

[7] Pires F L, Baran B. A Virtual Machine Placement Taxonomy [C]// Ieee/acm International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2015:159 - 168.

[8] Farahnakian F, Ashraf A, Liljeberg P, et al. Energy-Aware Dynamic VM Consolidation in Cloud Data Centers Using Ant Colony System [C]// IEEE International Conference on Cloud Computing. IEEE Computer Society, 2014:104 - 111.

[9] Feller E, Morin C, Esnault A. A Case for Fully Decentralized Dynamic VM Consolidation in Clouds[C]// IEEE, International Conference on Cloud Computing Technology and Science. IEEE, 2013:26 - 33.

[10] Ferdaus M H, Murshed M, Calheiros R N, et al. Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic[C]// 20th European International Conference on Parallel Processing (Europar 2014), 2014.

[11] Hu X B, Wang M, Paolo E D. Calculating Complete and Exact Pareto Front for Multiobjective Optimization: A New Deterministic Approach for Discrete Problems[J]. IEEE Transactions on Cybernetics, 2013, 43(3):1088 - 1101.

[12] Ashraf A, Porres I. Using Ant Colony System to Consolidate Multiple Web Applications in a Cloud Environment[C]// Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE, 2014:482 - 489.

[13] Gao Y, Guan H, Qi Z, et al. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing[J]. Journal of Computer & System Sciences, 2013, 79(8):1230 - 1242.

(上接第 240 页)

[6] Chen B, Kai M T, Washio T, et al. Local contrast as an effective means to robust clustering against varying densities [J]. Machine Learning, 2018(2):1 - 25.

[7] 淦文燕,刘冲. 一种改进的搜索密度峰值的聚类算法[J]. 智能系统学报,2017,12(2):229 - 236.

[8] 杨震, 王红军, 周宇. 一种截断距离和聚类中心自适应的聚类算法[J]. 数据分析与知识发现, 2018, 2(3):43 - 52.

[9] 孙昊,张明新,戴娇,等. 基于网格的快速搜寻密度峰值的聚类算法优化研究[J]. 计算机工程与科学,2017, 39(5):964 - 970.

[10] 王飞,王国胤,李智星,等. 一种基于网格的密度峰值聚类算法[J]. 小型微型计算机系统, 2017, 38(5): 1034 - 1038.

[11] 王飞. 基于网格的密度峰值聚类算法研究[D]. 重庆:重庆邮电大学,2017.

[12] 杨洁,王国胤,王飞. 基于密度峰值的网格聚类算法[J]. 计算机应用,2017,37(11):3080 - 3084.

[13] 高诗莹,周晓锋,李帅. 基于密度比例的密度峰值聚类算法[J]. 计算机工程与应用,2017,53(16):10 - 17.

[14] 蒋礼青,张明新,郑金龙,等. 快速搜索与发现密度峰值聚类算法的优化研究[J]. 计算机应用研究,2016, 33(11):3251 - 3254.

[15] 石胜飞. 大数据分析与挖掘[M]. 北京:人民邮电出版社, 2018:166 - 228.