



基于布谷鸟搜索的虚拟机放置算法

姜栋瀚, 林海涛

(海军工程大学电子工程学院, 湖北 武汉 430033)

摘要: 针对虚拟机放置问题, 引入了布谷鸟搜索算法。首先, 将虚拟机放置方案映射为鸟巢, 并按照适应度高低将其分成顶巢和底巢。其次, 通过扰动函数对底巢和顶巢进行扰动。最后, 通过选择、迭代得到最佳放置方案。该算法可用于云数据中心的物理机整合, 使放置物理机数量最小化。通过 Cloudsim 进行仿真, 仿真结果表明, 比起重排序分组遗传算法、分组遗传算法、改进的最小加载和改进的降序首次适应算法, 提出的方法不仅避免了局部最优, 而且具有更高的性能优势。

关键词: 云数据中心; 布谷鸟搜索; 虚拟机; 放置方案

中图分类号: TN911

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2017215

Virtual machine placement algorithm based on cuckoo search

JIANG Donghan, LIN Haitao

School of Electronic Engineering, Naval University of Engineering, Wuhan 430033, China

Abstract: A cuckoo search algorithm was introduced for virtual machine placement. Firstly, the virtual machine placement program was mapped to the nest, and according to the level, the fitness would be divided into top and bottom nest. Secondly, the bottom nest and the top nest were disturbed by the disturbance function. Finally, by selecting, iterations got the best placement scheme. The algorithm was used for physical integration of cloud data centers, minimizing the number of physical machines placed. The algorithm is simulated by Cloudsim and the results show that the proposed method not only avoids the local optimum, but also has higher performance advantages than the re-ordered grouping genetic algorithm, the group genetic algorithm, the improved least load algorithm and the improved first fit decrease algorithm.

Key words: cloud data center, cuckoo search, virtual machine, placement program

1 引言

为了应对日益增长的需求, 诸如亚马逊、谷歌、华为等大型公司已经建立了大型云数据中心。随着云数据中心的日渐庞大, 迫切需要其具有更少的物理机、高利用率、高可用性、高灵活性、

低时间跨度、低硬件和操作成本。功率的大量消耗, 一方面会增加云数据中心的能量成本, 另一方面过多的热耗散会增加硬件故障率^[1]。因此, 降低能耗显著影响云数据中心的可用性、生产力以及可靠性。

虚拟机(virtual machine, VM)放置问题可

以看作一个装箱问题^[2], 将不同尺寸的物品包装到给定容量的箱子中, 使得使用箱子的数量最小化。在VM放置问题中, 要装箱的物品为VM, 其大小取决于资源利用率。箱子是物理机, 其容量为物理机利用率阈值。最后, 尺寸由资源类型(如CPU、内存等)表示。图1表示将3个虚拟机放置到具有两个维度(即CPU和内存)的物理机中。

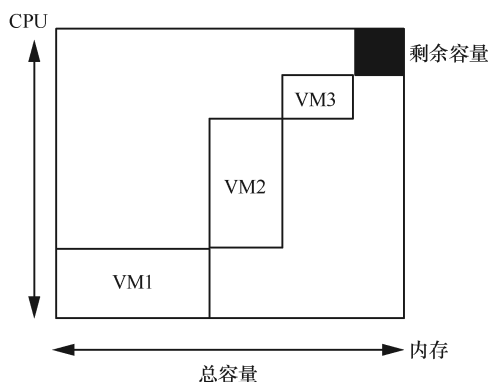


图1 物理机中放置3台虚拟机的资源映射示例

用于物理机整合的VM放置问题算法可以大致分为以下几类。

一类是基于经典装箱算法, 通过排序理论对VM放置问题建模。目前, 用于装箱问题的经典算法主要有降序最佳适应(best fit decreasing, BFD)算法和降序首次适应(first fit decreasing, FFD)算法^[3]。这两种算法都是首先将虚拟机和物理机降序排列。不同的是, FFD算法将虚拟机放置到第一个能满足虚拟机资源需求的物理机中, 而BFD算法是将虚拟机放置到最适合虚拟机资源需求的物理机中。Maruyama等人^[4]提出了8种不同方法, 根据多维尺寸对物品进行排序, 将排序列表中的每个物品装在可容纳它的第一个箱子中。如果没有一个箱子可以容纳该物品, 则将新的箱子引入当前放置方案中, 直到所有物品被成功装箱。最小负载(least load, LL)算法^[5]尝试通过将物品分配给最小负载的箱子来平衡现有负载。Ajiro等人^[6]提出了一种改进的降序首次适应(IFFD)算法和改进的最小负载(ILL)算法。实

验证明, IFFD和ILL优于标准FFD和LL算法。但是, 由于上述算法每次映射只能将一台虚拟机放置在物理机上, 在规模较大时, 难以保证计算效率。

另一类是基于群体优化算法优化放置问题。例如在分组遗传算法^[7](generational genetic algorithm, GGA)中, 使用基于组的编码方案, 解决了经典遗传算法难以处理的分组问题。Wilcox等人^[8]提出了重排序分组遗传算法(reordered grouping genetic algorithm, RGGA), 其使用基于组的编码和序列编码两种编码方案, 采用GGA的适应度函数解决了多维度装箱问题。参考文献[9]介绍了基于粒子群优化的虚拟机放置算法, 实现了能效整合, 减少了虚拟机使用数量和迁移次数。但是上述算法在规模较大时, 容易陷入局部最优, 难以得到全局最优解。

此外, 参考文献[10]中介绍和评估了基于蚁群优化的虚拟机放置算法, 根据每个物理机的处理能力对虚拟机进行了优化放置。但是其所提算法一般选择近距离的物理机, 且过多考虑任务序列, 具有明显的局限性。参考文献[11]提出了一种VM分裂分配算法, 使用多个更小的虚拟机代替单个大型虚拟机, 以更小的粒度进行资源分配, 从而减少了物理机使用数量。然而, 在算法中, 一个应用程序要部署多个小型虚拟机, 需要为每个小型虚拟机分配足够的资源, 在运行时间方面难以保证。

综上所述, 现有虚拟机放置算法存在诸多不足。尤其是在大规模资源映射时, 目前的解决方案经常陷入局部最优, 效率难以保证。本文将布谷鸟搜索算法(cuckoo search algorithm, CSA)稍加改进, 创新性地应用于虚拟机放置问题, 所提出的方法既能避免局部最优问题, 又具有较高的性能优势。

2 布谷鸟搜索算法

布谷鸟搜索算法是由Yang X S和Deb在2009年



开发的元启发式算法^[12]，它基于布谷鸟寻巢的寄生行为来求解优化问题。一方面，布谷鸟（寄主鸟）通过随机选择在其他鸟窝主人（宿主鸟）的巢中产卵。另一方面，宿主鸟一旦发现外来蛋，该鸟蛋就被抛出或者该巢被丢弃。原来鸟窝里的蛋表示原来的解，布谷鸟蛋表示新的解，整个过程意味着用新的、更好的解来取代原有解。布谷鸟繁殖过程有3个理想化的规则简化。

- 每只布谷鸟一次只产一个蛋，并将其随机放在巢中。
- 适应度较好的高品质鸟巢会传承给下一代。
- 鸟窝的主人可以发现外来鸟蛋的概率为 $P_a \in [0,1]$ 。这些鸟巢会被新巢取代，可表示为： $N_{\text{new}} = n \cdot P_a$ 。其中， n 表示鸟巢个数， N_{new} 表示被新鸟巢替换的个数，新巢代表新的随机解决方案。

在这些条件下，布谷鸟通过莱维飞行的方法进行局部和全局搜索。莱维飞行是一个服从莱维分布的随机游走过程。使用莱维飞行进行资源选择的原因如下。

- 莱维分布具有无穷的均值和方差，因此莱维飞行能够有效地探索搜索空间。
- 莱维飞行是随机过程，即下一状态仅取决于先前状态。
- 莱维飞行具有尺度不变性，用于模拟聚类问题。

莱维飞行可表示为随机方程： $x_i^{t+1} = x_i^{(t)} + \alpha \otimes L(\lambda)$ ，其中， $x_i^{(t)}$ 和 x_i^{t+1} 分别表示在时间 t 和 $t+1$ 的资源。 α 是步长的缩放因子， $L(\lambda)$ 是莱维飞行的随机搜索向量。显然，如果步长大，则下一个解 x_i^{t+1} 将远离当前解 $x_i^{(t)}$ 。

CS算法在很多方面优于其他元启发式算法，例如其设计可以调整少数参数，例如底巢的百分比和莱维飞行步长。此外，对于复杂优化问题，CS算法已被证明能够避免陷入局部最优，明显优于遗传算法和粒子群优化算法^[13]。

在本文中，采用布谷鸟搜索算法解决虚拟机放置问题。首先，基于CSA设计了物理机整合算法，基于适应度评估来确定放置方案的质量。其次，提出新的扰动函数，使CSA能够有效地穿越较大空间。

3 用于物理机整合的布谷鸟搜索算法

将布谷鸟搜索算法用于虚拟机放置问题，目标是 최소화物理机数量，这通常被称为物理机整合。在物理机整合中，使用最少数量的物理机将VM放置到云数据中心。随后，关闭没有分配VM的物理机，以减少散热和降低功耗。本文将用于物理机整合的CSA，与IFFD^[6]、ILL^[6]、GGA^[7]和RGGA^[8]的性能进行比较。

3.1 问题描述

这里的云数据中心被视为完全虚拟化，以便所有应用程序在虚拟机上运行。CPU和内存维度用于表征VM和物理机节点。为简洁起见，本文暂不考虑磁盘大小等维度。如果物理机托管多个虚拟机，则物理机CPU利用率看作其托管的虚拟机CPU利用率之和，物理机内存利用率看作其托管VM的内存利用率之和。此外，假设该云数据中心中所有物理机具有相同的容量。

假设有 n 个VM放置在 m 个物理机上，所有VM请求都可以由单个物理机处理。变量 i 和 j 分别表示单个VM和物理机，集合 I 和 J 分别表示所有VM和物理机。令 ρ_i^c 表示VM i 的CPU请求， ρ_i^m 表示VM i 的内存请求。 T_{cj} 为物理机 j 的CPU利用率最大容量， T_{mj} 为内存利用率的最大容量。此外，定义以下两个二进制决策变量。

物理机分配变量 y_j ：如果物理机 j 正在使用，则 $y_j=1$ ，否则为0。

虚拟机分配变量 $x_{i,j}$ ：如果VM i 被放置在物理机 j 中，则 $x_{i,j}=1$ ，否则为0。

由于本节算法的目的是放置给定VM，使得使用的物理机数量最小，所以放置问题可以

表示为:

$$\min f(y) = \sum_{j=1}^m y_j \quad (1)$$

约束条件为:

$$\sum_{i=1}^n \rho_i^c \cdot x_{i,j} \leq T_{cj} \cdot y_j, \forall j \in J \quad (2)$$

$$\sum_{i=1}^n \rho_i^m \cdot x_{i,j} \leq T_{mj} \cdot y_j, \forall j \in J \quad (3)$$

$$\sum_{j=1}^m x_{i,j} = 1, \forall i \in I \quad (4)$$

$$y_j, x_{i,j} \in \{0,1\}, \forall i \in I, \forall j \in J \quad (5)$$

约束式(2)和式(3)保证虚拟机放置不超过物理机的容量阈值,约束式(4)确保VM被放置在一个物理机中,约束式(5)表示变量 $x_{i,j}$ 和 y_j 的域。

3.2 算法介绍

3.2.1 编码

CSA采用与遗传算法类似的染色体编码,基因代表物理机。如果编码为: $A=\{4,5,6\}$, $B=\{1,2,7\}$, $C=\{3,8\}$, $D=\{0,9,10\}$ 。其中A、B、C、D表示物理机,0~10表示虚拟机。那么,该鸟巢可以表示为: $\{4,5,6\} \{1,2,7\} \{3,8\} \{0,9,10\}$ 。

3.2.2 适应度函数

启发式算法的主要特征之一是定义良好的适应度函数。对于VM放置问题,首先考虑的适应度是用于放置VM的物理机数量。然而,这种适应度并不实用。由于空间限制,通过使用这种适应度函数,所有个体将具有相同的适应度。本文将VM利用物理机剩余空间的能力作为适应度函数,能填充剩余空间的VM为高适应度,不能填充该空间的VM为低适应度。给定VM的适应度函数为:

$$f_1 = \frac{\rho_i^c + \rho_i^m}{(T_c - \sum_{k=1, k \neq i}^n \rho_k^c) + (T_m - \sum_{k=1, k \neq i}^n \rho_k^m)} \quad (6)$$

其中,分子表示VM的CPU和内存请求,分母

表示假设物理机的剩余CPU和内存容量。假设在物理机中放置3个虚拟机,设置阈值 T_c 和 T_m 为90%。如图2所示,根据式(6)计算得到3个VM的适应度都为1,因为它们完全满足物理机容量。相反,如图3所示,计算得到VM1的适应度为0.46,这是因为在图3中VM1未充分利用物理机中的剩余空间。因此,式(6)能够建模VM在物理机中的放置情况。最后,计算放置在物理机中的VM的平均适应度作为物理机的适应度。鸟巢(放置方案)的总适应度为放置方案中包含的所有VM的平均适应度。

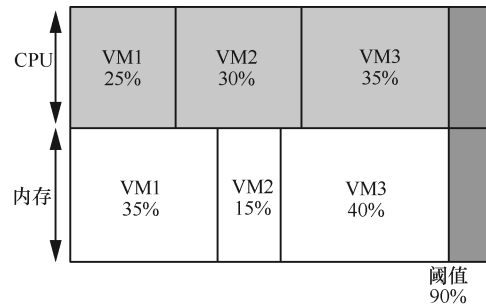


图2 高适应度虚拟机放置

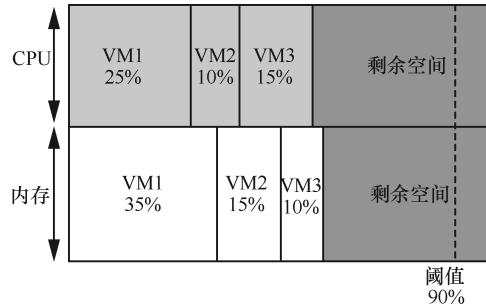


图3 低适应度虚拟机放置

3.2.3 扰动函数

(1) 扰动函数Function_1

Function_1 用于底部鸟巢扰动。该函数接收单个鸟巢作为输入,从莱维分布^[13]生成数字 x : $x = (1-u)^{-1/\alpha}$, 其中 $u \in [0,1]$, $\alpha = g^{1/6}$, g 为迭代次数。然后,从种群中删除 x 个物理机。由于删除物理机中包含的虚拟机被丢出鸟巢,需要重新排入。对丢失的虚拟机进行排序,通过首次适应算(first fit, FF)算法将排序的虚拟机重新排入放置方案中。



(2) 扰动函数Function_2

Function_2 用于顶部鸟巢扰动。该函数接收单个鸟巢作为输入, 计算鸟巢中物理机的适应度。基于适应度将物理机划分为顶部组和底部组。实验测试表明, 选择物理机适应度的前 75% 作为顶部组成员, 剩余 25% 作为底部组, 可以获得较好的结果。随后, 从鸟巢中删除底部组的物理机。对丢失的虚拟机进行排序, 并通过首次适应算法排入放置方案中, 最终得到的放置方案由函数返回。

3.2.4 用于 VM 放置的 CSA

算法分为 3 个部分: 在初始化阶段, 设置算法初始参数, 将群体分成顶部巢和底部巢; 在第二阶段, 每个底部巢被扰动和替换; 在第三阶段, 每个顶巢也被扰动。从群体中随机选择鸟巢, 如果扰动顶巢的适应度优于随机选择的巢, 则替换该巢, 直到达到最大迭代。本文在扰动底部巢后再次对群体进行排序, 这一步骤进一步增加了数据随机性。

算法主要步骤如下。

步骤 1 算法初始化, 设置底部巢中的种群分数 P_a 和最大迭代次数 $C_{\max}$ 。通过最初获得 VM 请求的 M 个随机序列, 生成大小为 M 的初始种群。

步骤 2 使用 FF 算法^[14]对随机序列进行排序, 以获得优化解。结束时, 获得 M 个不同的放置方案作为初始种群。

步骤 3 通过式 (6) 中的适应度函数计算 VM 的平均适应度, 从而得到每个巢的适应度, 迭代 $C_{\max}$ 次。这个过程将种群分成顶部巢和底部巢。适应度在前 75% 的种群作为底部巢的成员, 剩余的巢作为顶部巢的成员。使用这种排序方法以避免在算法中排序计算的复杂化。

步骤 4 对于底部的每个鸟巢, 使用扰动函数 Function_1 生成一个新鸟巢。生成的新鸟巢用于替换原来的底部鸟巢。然后将种群再次分成顶部巢和底部巢。

步骤 5 对于每个顶部鸟巢, 使用扰动函数

Function_2 从当前顶部鸟巢生成新的鸟巢 X_k 。

步骤 6 从整个种群中随机生成鸟巢 X_r 。如果新创建的鸟巢 X_k 的适应度优于 X_r 的适应度, 则用它替换 X_r 。

步骤 7 存储到适应度最高的鸟巢, 重复外部循环, 直到达到最大迭代次数。

输入: 底部种群系数 $P_a=0.75$, 最大迭代次数 $C_{\max}$

输出: 最优放置方案

Initialize Parameter (CSAParameter); //初始化参数

Code Nests (Nests); //初始化种群

Calculate Fitnesss(Nests); //计算每个鸟巢的适应度

for $g=1$ to n ; //迭代

Generate TopNest();

Generate BottomNest(); //将种群分为顶部鸟巢和底部鸟巢

for $X_i \in \text{BottomNest}$; //对于底部鸟巢, 使用扰动函数 1 生成新鸟巢

$X_j = \text{Function_1}(X_i)$;

$X_i = X_j$;

$f(X_i) = f(X_j)$;

end for;

Generate TopNest();

Generate BottomNest(); //将种群再次分成顶部鸟巢和底部鸟巢

for $X_i \in \text{TopNest}$ //对于顶部鸟巢, 使用扰动函数 2 生成新鸟巢

$X_k \leftarrow \text{Function_2}(X_i)$;

$X_r = \text{SelOneNest}$; //从整个种群中随机选择一个鸟巢 X_r

if $f(X_k) = f(X_r)$; //如果新创建鸟巢 X_k 的适应度优于 X_r 的适应度, 则用 X_k 替换 X_r

$X_r = X_k$;

$f(X_r) = f(X_k)$;

end if;

end for;

Store OptimalSolution();//存储最好的鸟巢
(放置方案)

end for

4 实验分析

为了生成实验数据集,需要在多个相关性实验中收集随机CPU和内存请求序列。为简单起见,在MATLAB R2012b中生成实验数据集,在设定环境中配置相应参数。所采用的方法是使用概率 P ,令服务器的CPU和内存利用率都大于或等于参考值,或者两种利用率都小于参考值。以下是用于生成CPU和内存利用请求实例的伪代码。

输入: R_c 、 R_m 、 P 、 n

输出: 实验数据集

for $i=1$ to n ;

$\rho_i^c \leftarrow \text{rand}(2R_c)$;

$\rho_i^m \leftarrow \text{rand}(2R_m)$;

$r \leftarrow \text{rand}(0,1)$;

if $(r < P, \rho_i^c \geq R_c)$ or $(r \geq P, \rho_i^c < R_c)$;

$\rho_i^m \leftarrow \rho_i^m + R_m$;

end if;

end for

以上过程中,函数 $\text{rand}(x)$ 返回均匀分布在 $[0,x]$ 中的随机数。 R_c 和 R_m 分别表示CPU和内存利用率参考值,实验中设置为 $R_c = R_m = 45\%$ 。 P 是用于控制CPU和内存利用率的强弱相关性。 P 的值设置为0、0.25、0.5、0.75和1,相关性由负相关到正相关。增加 P 的值,则生成随机VM请求。对于每个参考值,生成5组VM请求,每组代表不同的相关性。VM请求数量 n 设置为500,产生5个VM请求集合,每个VM请求集合中的CPU和内存维度具有不同相关性。每个相关强度生成100个VM请求实例,算法用于生成VM实例映射到物理机的放置方案。实验中,所有物理机的资源容量阈值设置为 $T_c = T_m = 90\%$ 。

布谷鸟搜索算法(CSA)、重排序分组遗传算法(RGGA)和分组遗传算法(GGA)的种群大小设置为25。由于本文迭代法所需的迭代次数属于不能通过计算得知的一类,为非确定值。理论上来说,迭代次数越多,结果越精确,时间越久。但是对本文来说,主要目标是减少使用的物理机数量,可行解空间不是特别大,模拟实验中发现,当迭代次数为100次时,本文算法所用的物理机数量已经远远低于其他算法,而且呈下降趋势。后面的迭代收敛实验中,算法也很快得到收敛,足以说明其收敛性。因此,最大迭代次数设置为100。

RGGA的交叉率和突变率分别设置为0.8和0.1, GGA的交叉率和突变率也设置为0.8和0.1。由于GGA适应度测量被设计为解决单维装箱问题,所以GGA使用RGGA的适应度函数。对于改进最小负载(improved least loaded, ILL)算法和改进首次适应(improved first fit decrease, IFFD)算法的参数设置:IFFD和ILL的最大迭代次数分别为VM请求数量 n 的30%和10%。

之后比较了本文所提出的CSA与IFFD^[6]、ILL^[6]、GGA^[7]和RGGA^[8]的性能,性能指标包括:执行时间(单位为s)、用于放置的物理机平均数 m 和物理机整合比 m/L_B 。 L_B 是可用于放置的物理机数量的理论下限,计算式为:

$$L_B = \max \left\{ \left[\left(\sum_{i=1}^n \rho_i^c \right) / T_c \right], \left[\left(\sum_{i=1}^n \rho_i^m \right) / T_m \right] \right\} \quad (7)$$

随着用于放置的物理机的数量 m 接近理论下限 L_B ,物理机整合比(m/L_B)逐渐收敛到1。

执行时间与内存和CPU相关性如图4所示。从图4中可以看出,虽然CSA的执行时间略高于RGGA,但是与其他算法相比具有良好的执行效果。

放置的物理机平均数与内存和CPU相关性及物理机整合比与内存和CPU相关性分别如图5和图6所示。从图5和图6中可以看出,CSA所用的物理机平均数量和平均整合比优于其他算法,并且随着

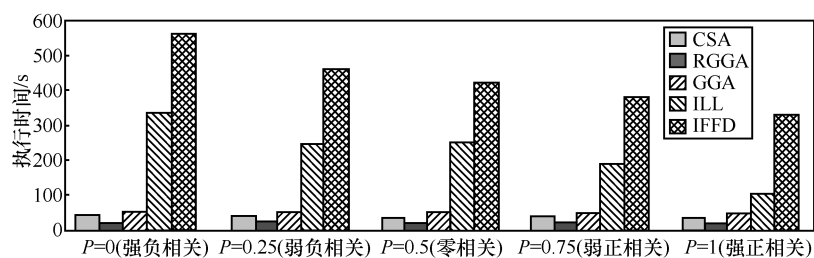


图4 执行时间与内存和CPU相关性

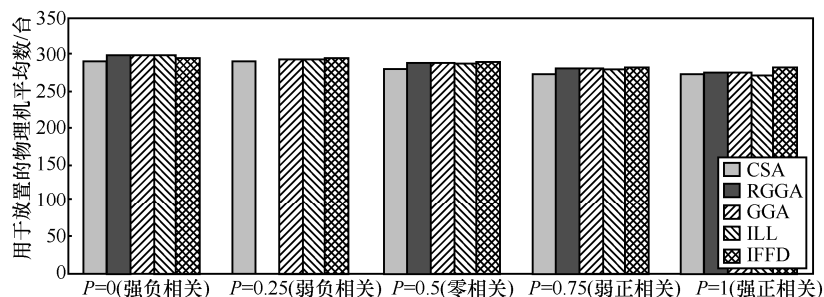


图5 放置的物理机平均数与内存和CPU相关性

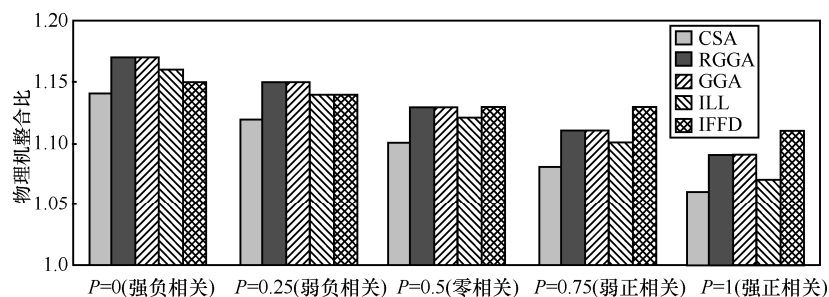
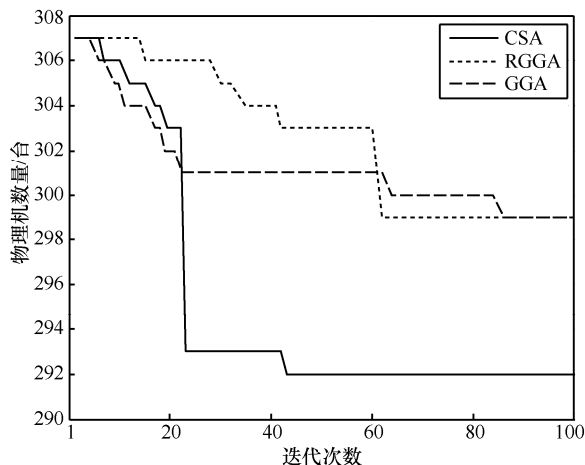


图6 物理机整合比与内存和CPU相关性

相关性增加,放置所需的物理机数量减少。这是因为当CPU和内存请求之间呈负相关时,每个物理机空间大量浪费,必须使用大量物理机来容纳VM。

$P=0$ (强负相关)、 $P=0.5$ (零相关)、 $P=1$ (强正相关) 时物理机数量与迭代次数的关系分别如图7、图8和图9所示。从图7~图9中可以看出,本文将所提出CSA的收敛速率与RGGA、GGA算法进行比较。从生成数据集中,选择 $P=0$ (强负相关)、 $P=0.5$ (零相关) 和 $P=1$ (强正相关) 3种情况,令 $R_c=R_m=45\%$, $n=500$ 。每个实验使用相同的初始种群,运行10次后绘制最佳曲线。可以发现,在少量迭代之后,CSA具有比RGGA和GGA更快的收敛速度。迭代结束时,CSA使用物理机数量明显少于RGGA和GGA算法。

图7 $P=0$ (强负相关) 时物理机数量与迭代次数的关系

使用物理机平均数量与迭代次数的关系如图10所示。该实验设置 $R_c=R_m=45\%$, $n=500$, $P=0$ 。从图10中可以看出,每次迭代的物理机平均数量

不会随着迭代而均匀减少。这正是CSA的优点之一，它能够逃避大多数优化问题陷入局部最优。

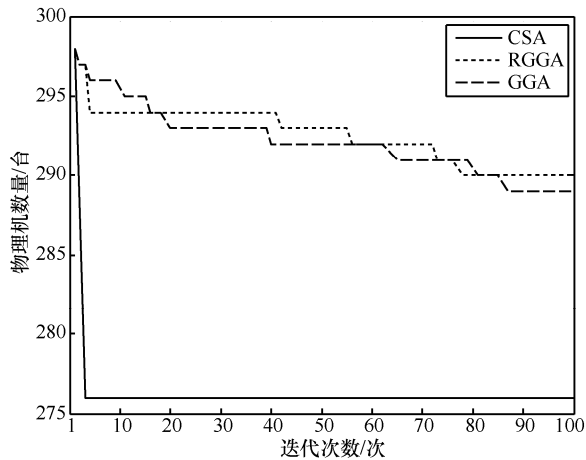


图8 $P=0.5$ (零相关) 时物理机数量与迭代次数的关系

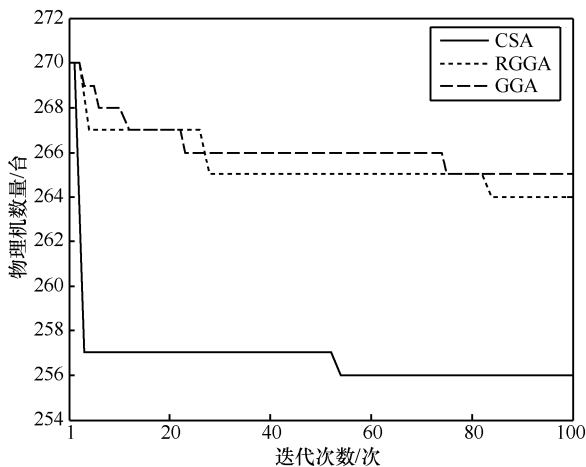


图9 $P=0$ (强负相关) 时物理机数量与迭代的次数

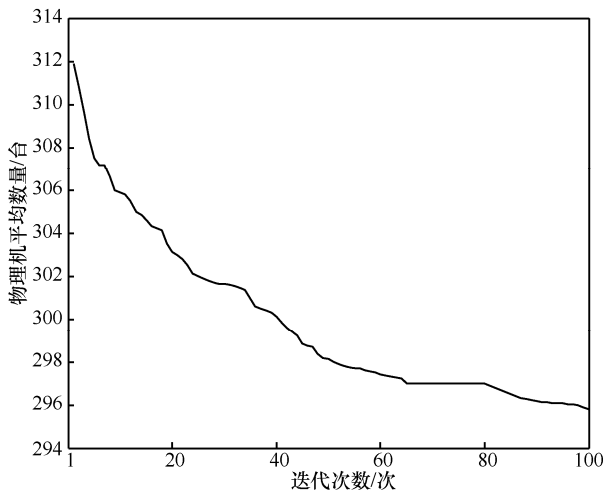


图10 使用物理机平均数量与迭代次数的关系

运行时间与放置虚拟机数量的关系如图11所示。设置 $P=0$ (负相关), 这会使虚拟机放置需要更长的时间。VM的数量从200增加到2000。从图11中可以看出, 当参考值 $R_c=R_m=45\%$ 时, 与设置为25%相比, 需要更多数量的物理机。这是因为当 $R_c=R_m=45\%$ 时, VM利用率请求在 $[0,90\%]$ 范围内, 而设置为25%时, 范围是 $[0,50\%]$ 。另外, 即使在参考值被设置为45%的情况下, 放置2000个VM花费大约600s, 这表明所提出的CSA具有良好的可伸缩性。

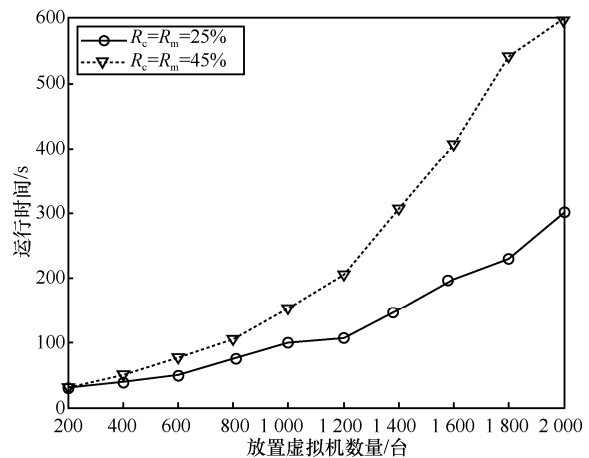


图11 运行时间与放置虚拟机数量的关系

5 结束语

本文基于布谷鸟搜索提出了一种虚拟机放置算法, 将物理机数量最小化。实验结果表明, 比起RGGA、GGA、ILL和IFFD算法, 本文两种算法具有明显的性能优势。在未来工作中, 将关注如何利用VM的动态迁移^[17], 使本文算法发挥更好的效果。

参考文献:

- [1] BELOGLAZOV A, BUYYYA R. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers[C]//The 8th International Workshop on Middleware for Grids, Clouds and e-Science, November 29-December 3, 2010, Bangalore, India. New York: ACM Press, 2010: 1-6.
- [2] FARAHNAKIAN F, PAHIKKALA T, LILJEBERG P, et al.



- Energy-aware VM consolidation in cloud data centers using utilization prediction model[J]. IEEE Transactions on Cloud Computing, 2016(99): 1.
- [3] LI C L, CHEN Z L. Bin-packing problem with concave costs of bin utilization[J]. Naval Research Logistics, 2006, 53(4): 298-308.
- [4] MARUYAMA K, CHANG S K, TANG D T. A general packing algorithm for multidimensional resource requirements[J]. International Journal of Parallel Programming, 1977, 6(2): 131-149.
- [5] SIDER A, COUTURIER R. Fast load balancing with the most to least loaded policy in dynamic networks[M]. Netherlands: Kluwer Academic Publishers, 2009.
- [6] AJIRO Y, TANAKA A. Improving packing algorithms for server consolidation[C]//International Computer Measurement Group Conference, December 2-7, 2007, San Diego, USA. Rhineland: DBLP, 2007: 399-406.
- [7] ZHANG K. A generational genetic algorithm with dynamic population size[C]//The 47th Annual Southeast Regional Conference, March 19-21, 2009, Clemson, South Carolina. New York: ACM Press, 2009: 58.
- [8] WILCOX D, MCNABB A, SEPPI K. Solving virtual machine packing with a reordering grouping genetic algorithm[C]//IEEE Congress of Evolutionary Computation, June 5-8, 2011, New Orleans, USA. New Jersey: IEEE Press, 2011: 362-369.
- [9] 李鸿健, 代宇, 刘锐, 等. 云数据中心高性能的虚拟机迁移整合算法研究[J]. 电信科学, 2015, 31(1): 71-77.
LI H J, DAI Y, LIU R, et al. Energy-efficient virtual machine migration and consolidation algorithm in cloud data center[J]. Telecommunications Science, 2015, 31(1): 71-77.
- [10] SHABEERA T P, KUMAR S D M, SALAM S M, et al. Optimizing VM allocation and data placement for data-intensive applications in cloud using ACO metaheuristic algorithm[J]. Engineering Science & Technology an International Journal, 2017, 20(2): 616-628.
- [11] LIU L, XU J, YU H, et al. VMSA: a performance preserving online VM splitting and placement algorithm in dynamic cloud environments[J]. The Journal of Supercomputing, 2016, 72(8): 3169-3193.
- [12] YANG X S. Cuckoo search[M]. Netherlands: Nature-Inspired Optimization Algorithms, 2014.
- [13] DEB S, YANG X S. Cuckoo search via levy flights[J]. Mathematics, 2012, 51(16): 210-214.
- [14] DOSA G. First fit algorithm for bin packing[M]. New York: Springer New York, 2016.
- [15] XUAN H J, JIANG Y. Study of task scheduling based on intelligent algorithm in CloudSim[J]. Value Engineering, 2016, 34(11): 122-129.
- [16] 雷蕾, 李景文, 宫大鹏, 等. 基于Hadoop的OSS域数据建模与采集方法研究[J]. 电信科学, 2015, 31(1): 134-144.
LEI L, LI J W, GONG D P, et al. Study on data modeling and collection in OSS based on Hadoop[J]. Telecommunications Science, 2015, 31(1): 134-144.
- [17] 廖建新. 大数据技术的应用现状与展望[J]. 电信科学, 2015, 31(7): 7-18.
LIAO J X. Big data technology: current applications and prospects[J]. Telecommunications Science, 2015, 31(7): 7-18.

[作者简介]



姜栋瀚 (1992-), 男, 海军工程大学硕士生, 主要研究方向为通信技术与网络。

林海涛 (1974-), 男, 博士, 海军工程大学副教授, 主要研究方向为信息网络管理与规划。