软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(4):916−932 [doi: 10.13328/j.cnki.jos.004965]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于混合遗传模拟退火算法的 SaaS 构件优化放置*

孟凡超[1,2], 初佃辉[1], 李克秋[2], 周学权[3]

[1](哈尔滨工业大学(威海) 计算机科学与技术学院,山东 威海 264209)

[2](大连理工大学 计算机科学与技术学院,辽宁 大连 116024)

[3](哈尔滨工业大学(威海) 经济管理学院,山东 威海 264209)

通讯作者: 孟凡超, E-mail: mfc@hitwh.edu.cn

摘 要: 目前,对于 SaaS 优化放置问题的研究都是假定云环境中的虚拟机的种类和数量都是确定的,即,在限定的资源范围内进行优化.然而,在公有云环境下,SaaS 提供者所需要的云资源数量是不确定的,其需要根据 IaaS 提供者所提供的虚拟机种类以及被部署的 SaaS 构件的资源需求来确定.为此,站在 SaaS 提供者角度,提出一种新的 SaaS 构件优化放置问题模型,并采用混合遗传模拟退火算法(hybrid genetic and simulated annealing algorithm,简称 HGSA)对该问题进行求解.HGSA 结合了遗传算法和模拟退火算法的优点,克服了遗传算法收敛速度慢和模拟退火算法容易陷入局部最优的缺点,与单独使用遗传算法和模拟退火算法相比,实验结果表明,HGSA 在求解 SaaS 构件优化放置问题方面具有更高的求解质量.所提出的方法为 SaaS 服务模式的大规模应用提供了理论与方法的支撑.

关键词: 软件即服务(SaaS);SaaS 构件优化放置;虚拟机网络图;混合遗传模拟退火算法

中图法分类号: TP311

中文引用格式: 孟凡超,初佃辉,李克秋,周学权.基于混合遗传模拟退火算法的 SaaS 构件优化放置.软件学报,2016,27(4):916−932. http://www.jos.org.cn/1000-9825/4965.htm

英文引用格式: Meng FC, Chu DH, Li KQ, Zhou XQ. Solving SaaS components optimization placement problem with hybird genetic and simulated annealing algorithm. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):916−932 (in Chinese). http://www.jos.org.cn/1000-9825/4965.htm

Solving SaaS Components Optimization Placement Problem with Hybird Genetic and Simulated Annealing Algorithm

MENG Fan-Chao[1,2], CHU Dian-Hui[1], LI Ke-Qiu[2], ZHOU Xue-Quan[3]

[1](School of Computer Science and Technology, Harbin Institute of Technology at Weihai, Weihai 264209, China)

[2](School of Computer Science and Technology, Dalian Institute of Technology, Dalian 116024, China)

[3](School of Economics and Management, Harbin Institute of Technology at Weihai, Weihai 264209, China)

Abstract: Current researches on SaaS (software as a service) optimization placement mostly assume that the types and number of virtual machines are constant in cloud environment, namely, the optimization placement is based on the restricted resource. However, in actual situation the types and number of virtual machines are unknown, and they need to been calculated according to the resource requirement of components deployed. To address the issue, from the view of SaaS providers, this paper proposes a new approach to SaaS optimization placement problem that not only is applied to initial deployment of SaaS, but also is applied to component dynamic deployment in the

* 基金项目: 国家科技支撑计划(2014BAF07B02); 国家自然科学基金(61432002); 山东省重大科技专项(2015ZDXX0201B02); 山东省自然科学基金(2015ZRA10032)

Foundation item: National Key R&D Plan Project of China (2014BAF07B02); National Natural Science Foundation of China (61432002); Major Scienece & Technology Specific Project of Shandong Province (2015ZDXX0201B02); Natural Science Foundation of Shandong Province (2015ZRA10032)

收稿时间: 2015-06-30; 修改时间: 2015-10-15, 2015-11-30; 采用时间: 2015-12-07; jos 在线出版时间: 2016-01-13

CNKI 网络优先出版: 2016-01-14 13:16:30, http://www.cnki.net/kcms/detail/11.2560.TP.20160114.1316.013.html

running phase of SaaS. A hybrid genetic and simulated annealing algorithm (HGSA) is used in this approach that combines the advantages of genetic algorithm and simulated annealing algorithm, and overcomes the problems of the premature of genetic algorithm and the lower convergence speed. Compared with the separated using of genetic algorithm and simulated annealing algorithm, the experimental results show that HGSA has higher quality in solving the problem of SaaS component optimization placements. The approach proposed in this paper will provide the support of theory and method for the large-scale application of SaaS service mode.

Key words: software as a service (SaaS); SaaS component optimization placements; virtual machine network graph; hybrid genetic and simulated annealing algorithm

随着云计算产业的发展,云计算逐步从概念走向实际应用,越来越多的传统软件提供商正在将其应用向云服务模式迁移.这些应用被部署在 IaaS(infrastructure as a service)提供商或者自己的云基础设施上,通过互联网,以按需、易扩展的方式向客户提供服务.被部署在云中的应用是以 SaaS(software as a service)模式对外提供服务.在 SaaS 服务模式下,使用 SaaS 应用的客户不需要购买完整的软件产品,也不需要配备相应的硬件系统和维护人员,只需通过互联网,按需租用即可,这对于成本预算有限、技术条件不足的中小企业来说,具有很强的吸引力.与传统的软件模式相比,SaaS 具有部署时间短、成本低、使用方便等优势,因此,其市场应用前景非常广阔,被认为是未来应用软件的发展趋势[1].

当传统软件提供商在向 SaaS 服务模式迁移时,所面临的一个重要问题是:如何根据需要部署的 SaaS 应用中每个构件的资源需求来确定最优的 SaaS 构件放置方案,即:选择哪些类型的虚拟机?每种虚拟机的数量是多少?每个被选择的虚拟机上放置哪些构件?如何放置才能够在满足每个构件资源需求的条件下,使得总的云资源使用成本最低?制定优化的 SaaS 构件放置方案是提高云资源利用率、降低运营成本的一项重要手段,具有十分重要的应用价值.尽管目前已经提出了许多 SaaS 构件优化放置方法,但是这些方法都假定虚拟机的类型和数量是确定的,即,在限定的云资源上进行优化放置,因此在实际应用中还存在许多问题.例如,SaaS 提供者在将其 SaaS 应用部署到 IaaS 提供者的云环境时,其只知道 IaaS 提供商所发布的虚拟机种类和价格,例如,亚马逊 EC2 的 m1.small,m1.large,m2.x large 等,但是所需要的每种类型的虚拟机的数量是未知的,它需要根据构件的资源需求来确定.尽管目前许多 IaaS 提供商也提供了相应的虚拟机选择方案,但是这些方法大都是简单的资源匹配,很难满足大型复杂企业级 SaaS 应用的部署需求.SaaS 构件优化放置问题是一个 NP 问题[2],针对该问题,本文提出一种基于混合遗传模拟退火算法的 SaaS 构件优化放置方法,其目标是提高云资源利用率、降低云资源使用成本.该方法既可以应用于 SaaS 应用的初始部署,也可应用于 SaaS 运行阶段的动态部署.

1 相关工作

针对 SaaS 优化放置问题,国内外学者已经展开了许多研究工作.根据放置的粒度,可分为租户放置、构件放置和应用放置这 3 个层次.

租户放置是指建立租户与 SaaS 应用或构件实例之间的服务关系,即,哪些租户的服务请求被分配到哪些 SaaS 应用或构件实例上执行.租户优化放置的一个主要目标是:在满足每个租户服务质量需求的条件下,使得现有的云资源数量能够支持更多的租户和并发用户请求.针对租户放置问题,目前已提出了多种优化策略和算法,例如启发式算法[2]、遗传算法[3]、准入控制策略[4]、分等级策略[5]等.

应用放置是指,建立 SaaS 应用与 IaaS 层的虚拟机之间的部署关系[6],应用放置主要适合于粒度比较小的 SaaS 应用,而对于大型复杂企业级 SaaS 应用来说,由于单个虚拟机的资源能力难以满足整个 SaaS 应用的资源需求,因此需要将 SaaS 应用分解为一组可独立部署的构件,这些构件被放置在多个虚拟机上运行.此时,SaaS 应用放置问题可转化为 SaaS 构件放置问题.SaaS 构件放置是指建立 SaaS 应用中的构件与 IaaS 层的虚拟机之间的部署关系[7],即,哪些构件被部署在哪个虚拟机上执行.由于同一个 SaaS 应用被分布式地部署在多个虚拟机上执行,因此,如何提高整个应用的性能是一个需要考虑的重要问题[8].构件优化放置的主要目标是,在保证每个租户的服务质量需求的前提下提高云资源利用率、降低云资源使用成本.

本文主要研究构件优化放置问题,针对该类问题,目前已经提出了许多相应的方法.Yusoh 提出了一种组合

SaaS 应用放置方法(SPP),SPP 分为初始放置和资源优化两个过程:初始放置的目标是确定如何将服务/数据构件放置到相应的计算/存储服务器上,使得整个 SaaS 应用的执行时间最少;资源优化的目标则是通过调整运行阶段构件的放置方案来提高资源的利用率、减少运营成本[7].Moens采用特征模型来描述 SaaS 应用的构件及其关系,并采用整数线性规划和启发式方法对问题进行求解[9].Zhu 针对云数据中心的应用构件放置问题,提出了一个基于MIP的求解算法[10].Jin针对跨多个云数据中心的应用构件放置问题提出了一种分布式方法,并给出了相应的模拟分析方法[11].Sailer 提出了一个基于图的云服务放置方法,该方法采用图来描述业务支持的服务(BSS)、操作支持的服务(OSS)以及它们之间的映射,并提出了一种基于服务需求和数据中心资源类型的云资源分配算法[12].

尽管目前已经存在许多求解构件优化放置的算法,但是这些算法都是将虚拟机的类型和数量看作是固定的常量,该类问题可以抽象为多背包问题(multiple knapsack problem,简称 MKP),MKP 是一个 NP 问题[13].目前,求解 MKP 的方法主要有精确算法[13]、启发式算法[14]、遗传算法[15]和蚁群克隆算法[16]等.MKP 的目标是使装入多个背包中的物品的总价值最大或者物体放置得更佳均衡,由于背包的数量是有限的,因此有些物体不能被装入任何包中.本文所提出的 SaaS 构件优化放置问题也可以看作是一个特殊的背包问题,我们称其为多维多类背包问题(multidimensional multiple-class knapsack problem,简称 MMCKP),其中,构件可以看作是物品,而虚拟机模板则看作是背包的类型,不是具体的背包.另外,物品和背包的属性都是多维的.与传统的 MKP 问题不同的是,MMCKP 要求所有的物品都必须被放入到一个具体的背包中,其目标是使用背包的成本最低.另外,由于 SaaS 构件优化放置问题不仅要考虑虚拟机的使用成本,而且还要考虑不同虚拟机之间的网络通信成本,因此,其问题的复杂度要比 MKP 更高,目前并未见到相关文献解决此类问题.

SaaS 构件放置问题是一个 NP 问题,当问题的规模较大时,采用枚举等精确算法的运行时间代价比较高,很难满足实际应用的需求.因此,需要采用相应的近似算法对其进行求解.目前,求解 SaaS 构件优化放置问题的主要近似方法有启发式算法[2,9]和遗传算法(genetic algorithm,简称 GA)[7]等进化算法.由于 GA 在生成初始解时已经采用了相应的启发式规则,因此在解决构件放置问题时,其求解质量比单纯采用启发式算法的效果要好. GA 是解决离散优化问题的一种常用的方法,它通过选择、交叉、变异等进化操作来实现群体的协同进化.GA 具有较强的全局搜索能力,但其局部搜索能力较差,容易产生“早熟”收敛问题[17].由于本文所提出的构件放置问题中虚拟机的数量是不确定的,因此染色体中每个基因的取值范围较大,采用交叉操作会使虚拟机的范围变大,其求解质量较低.另外,由于本文所提出的构件优化问题不仅考虑新放置的构件和新创建的虚拟机,同时还需要考虑已创建的虚拟机和已放置的构件,因此在产生新解时,许多虚拟机已经放置了构件,尽管这些虚拟机的可用资源量已经不具备放置资源需求量较高的大粒度构件的条件,但仍有可能放置一些资源需求量较小构件的条件.因此在解的搜索过程中,如果只采用遗传算法的随机交叉、变异等操作来产生新解,其错误率较高,进而影响算法的效率和求解质量.

为了克服遗传算法在求解 SaaS 构件优化放置问题方面的不足,本文将遗传算法与模拟退火算法(simulated annealing algorithm,简称 SA)相结合,取长补短,提出了一种求解 SaaS 构件优化放置问题的混合遗传模拟退火算法(hybrid genetic and simulated annealing algorithm,简称 HGSA).SA 是对热力学中退火过程的模拟,它是一种在某一给定的初始温度下,通过缓慢下降温度参数,并根据相应的概率接受准则在解空间中随机寻找目标函数全局最优解的优化方法.SA 具有较强的局部搜索能力,但其全局把握能力较弱[18].HGSA 是将 GA 与 SA 相结合而构成的一种优化算法,它克服了 GA 收敛速度慢和 SA 容易陷入局部最优的缺点,从而提高了问题的求解质量[19].混合遗传模拟退火算法目前已在许多领域得到应用,并取得了较好的效果[20~22].例如:Ren 采用混合遗传模拟退火算法以解决电子商务供应链环境下具有返程的本地库存车辆路线优化问题[20];Deb 利用混合遗传模拟退火算法解决计算机辅助计划中零件装配序列规划问题[21];McCall 提出了一种并行混合遗传模拟退火算法来求解计算网格中的 Q3AP 问题[22].由于问题领域的不同,因此针对不同问题的混合遗传模拟退火算法的结构和流程也不完全相同.本文针对云计算领域中 SaaS 构件放置问题的特点,提出了一种基于二元组的染色体编码方法来表示虚拟机类型和虚拟机,并设计了 4 种邻域搜索规则来改善种群中每个个体的质量,避免了交叉操作

所导致解的质量下降问题.与单独使用遗传算法和模拟退火算法相比,实验结果表明,HGSA 具有较高的求解质量.

2 SaaS 构件优化放置问题描述

SaaS 构件优化放置是指在 SaaS 应用的初始部署或者运行阶段,SaaS 提供者根据部署的每个构件的资源需求以及现有云环境中构件的放置情况,确定一个当前的最优放置方案.图 1 描述了 SaaS 构件优化问题的框架.从软件部署角度来看,一个 SaaS 应用可以抽象为一个图结构,称为构件关系图,其中,图的顶点代表构件,边代表构件之间的连接关系.同时,SaaS 构件放置方案也可表示为一个图结构,称为虚拟机网络图,其中,图的顶点代表包含一组构件的虚拟机,边则代表虚拟机之间的通信链路,因此,构件放置可以看作是对构件关系图的划分,每个被划分的子图被映射为一个由相应的虚拟机模板所创建的虚拟机,构件之间的连接关系被映射为虚拟机之间的通信链路.虚拟机是由相应的虚拟机模板创建的,不同的虚拟模板具有不同的资源配置和价格.由同一虚拟机模板所创建的虚拟机具有相同的资源配置和价格.如果被放置在两个不同虚拟机上的构件之间存在连接关系,则这两个虚拟机之间存在通信链路.为了降低网络通信成本,我们尽量将具有紧密耦合关系的构件放置到一个虚拟机中.但是虚拟机的资源需求会受到其所属的虚拟机资源能力的约束,因此,SaaS 构件优化放置的关键就是确定一个最优的构件关系图划分,该划分使得虚拟机网络的资源使用成本最低.该问题是一个 NP 问题.

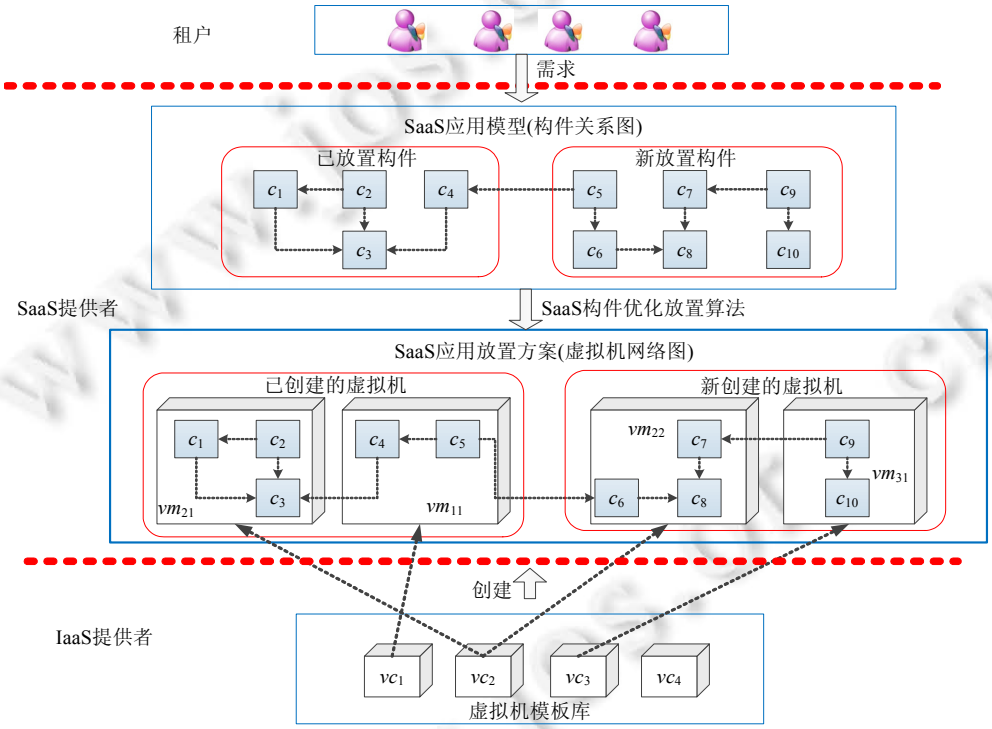

Fig.1 Frame of SaaS component optimal placement problem

图 1 SaaS 构件优化放置问题框架

下面我们给出 SaaS 构件优化放置问题中所涉及到的一些基本概念的定义.

定义 1(构件关系图). 构件关系图可以定义为 $G=(C,E)$,其中,

- $C=C_A\cup C_B$ 为构件的集合,C_A 为已放置的构件集合,C_B 为新放置的构件集合;
- $E=\{(c,c')|c,c'\in C\}$ 为构件之间的连接关系集,对于 $\forall(c,c')\in E$,表示从 c 到 c' 存在连接关系,对于 $\forall c\in C$,$r_c(c)$ 为 c 的计算资源消耗量,$r_m(c)$ 为 c 的内存资源消耗量,$r_d(c)$ 为 c 的存储资源消耗量;对于 $\forall(c,c')\in E$,

$r_t(c,c')$表示从 c 到 c'的网络通信带宽消耗量.

构件放置的一个难点在于,事先并不知道每种虚拟机的数量,因此,我们首先需要确定每种虚拟机模板所要创建的虚拟机的数量.假设有 n 种虚拟机模板,令 $VC=\{vc_1,vc_2,\dots,vc_n\}$表示虚拟模板的集合,其中,$vc_i$ 表示第 i (i=1,2,…,n)种虚拟机模板,令 $r_c(vc_i)$表示 vc_i 的计算资源提供量,$r_m(vc_i)$表示 vc_i 的内存资源提供量,$r_d(vc_i)$表示 vc_i 的存储资源提供量,$p(vc_i)$为 vc_i 的资源使用成本.

我们通过 n 个非负整型决策变量 $S=\{s_1,s_2,\dots,s_n\}$来表示由虚拟机模板所创建的虚拟机数量,如果 s_i=0,表示没有选择 vc_i 创建虚拟机.令 $VM=VM_A\cup VM_B=VM_1\cup VM_2\cup\dots\cup VM_n$ 表示放置构件的虚拟机集合,其中,VM_A 表示已创建的虚拟机集合,VM_B 表示需要新创建的虚拟机集合,VM_i 表示由 vc_i 所创建的虚拟机集合,如果 s_i=0,则有 $VM_i=\varnothing$,即,不存在类型为 vc_i 的虚拟机;否则,$VM_i=\{vm_{i1},vm_{i2},\dots,vm_{in_i}\}$,其中,$vm_{ij}$($j$=1,2,…,$n_i$)表示由 vc_i 所创建的第 j 个虚拟机.对于$\forall vm_{ij}\in VM_i$,令 $r_c(vm_{ij}),r_m(vm_{ij})$和 $r_d(vm_{ij})$分别表示 vm_{ij} 的计算、内存和存储资源提供量.由同一虚拟机模板所创建的虚拟机具有相同的资源配置和价格,即:

$$r_c(vm_{ij})=r_c(vc_i),r_m(vm_{ij})=r_m(vc_i),r_d(vm_{ij})=r_d(vc_i),p(vm_{ij})=p(vc_i).$$

集合 VM_i(i=1,2,…,n)可划分为两个子集:已创建的虚拟机集合 VM_{Ai} 和新创建的虚拟机集合 VM_{Bi},即 $VM_i=VM_{Ai}\cup VM_{Bi}$.令 $n_{Ai}=|VM_{Ai}|$表示 VM_i 中已创建的虚拟机数量,$n_{Bi}=|VM_{Bi}|$表示 VM_i 中新创建的虚拟机数量,因此,$n_i=n_{Ai}+n_{Bi}$.为了处理方便,我们规定 VM_i 中的前 n_{Ai} 个虚拟机为已创建的,后 n_{Bi} 个虚拟机为新创建的,n_{Bi} 的数量在新构件放置开始时是不确定的,其需要根据放置结果来确定,因此,

$$VM_A=VM_{A1}\cup VM_{A2}\cup\dots\cup VM_{An},VM_B=VM_{B1}\cup VM_{B2}\cup\dots\cup VM_{Bn}.$$

例如,在图 1 所示的放置场景中:$C_A=\{c_1,c_2,c_3,c_4\}$为已放置的构件集合,其中,c_1,c_2 和 c_3 被放置在由虚拟机模板 vc_2 所创建的虚拟机 vm_{21} 中,c_4 被放置在由 vc_1 所创建的虚拟机 vm_{11} 中;$C_B=\{c_5,c_6,c_7,c_8,c_9,c_{10}\}$为新放置的构件集合,通过制定新的放置方案,$c_5$ 将被放置在 vm_{11} 上,c_6,c_7,c_8 将被放置在由虚拟机模板 vc_2 所创建的虚拟机 vm_{22} 中,c_9 和 c_{10} 将被放置在由虚拟机模板 vc_3 所创建的虚拟机 vm_{31} 中.

为了提高云资源的利用率,我们尽量将 C_B 中的构件放置在 VM_A,这样可以减少新创建虚拟机的数量,进而可以降低云资源的使用成本.

定义 2(SaaS 构件放置方案). SaaS 构件放置方案可以定义从构件集合 C 到虚拟机集合 VM 的映射函数 $\rho:C\to VM$.对于$\forall c\in C,\rho(c)\in VM$ 表示 c 所放置的虚拟机;对于$\forall vm_{ij}\in VM,C(vm_{ij})=\{c\in C|\rho(c)=vm_{ij}\}$表示放置在虚拟机 vm_{ij} 上的构件集合.

例如,在图 1 所示的放置场景中,

$VM=\{vm_{11},vm_{21},vm_{22},vm_{31}\},C(vm_{11})=\{c_4,c_5\},C(vm_{21})=\{c_1,c_2,c_3\},C(vm_{22})=\{c_6,c_7,c_8\},C(vm_{31})=\{vm_9,vm_{10}\}$.

在构件关系图中,如果两个构件之间存在连接关系,并且这两个构件被放置在不同的虚拟机上,则这两个虚拟机之间存在网络通信.根据放置在两个不同虚拟机之间的构件连接关系,我们可以定义虚拟机网络图.

定义 3(虚拟机网络图). 虚拟机网络图可以定义为 $VN=(VM,L_{VM})$,其中,VM 为放置 C 中的构件所需要的虚拟机集合;$L_{VM}=\{(vm_{ij},vm_{st})|\exists c\in C(vm_{ij}),c'\in C(vm_{st}):(c,c')\in E\wedge vm_{ij}\neq vm_{st}\}$表示 VM 中虚拟机之间的通信链路的集合,对于$\forall(vm_{ij},vm_{st})\in L_{VM}$,表示 vm_{ij} 和 vm_{st} 之间存在通信链路.

对于$\forall(vm_{ij},vm_{st})\in L_{VM},E(vm_{ij},vm_{st})=\{(c,c')|c\in C(vm_{ij}),c'\in C(vm_{st}),(c,c')\in E\}$为从 vm_{ij} 到 vm_{st} 的构件连接关系集,则从 vm_{ij} 到 vm_{st} 的通信流量可以定义为

$$r_t(vm_{ij},vm_{st})=\sum_{(c,c')\in E(vm_{ij},vm_{st})}r_t(c,c').$$

因此,虚拟机网络的通信流量可定义为 $r_t(L_{VM})=r_t(L_{VM})=\sum_{(vm_{ij},vm_{st})\in L_{VM}}r_t(vm_{ij},vm_{st})$.

基于上述分析,虚拟机网络的成本可以定义为 $c(VN)=c(VM)+c(L_{VM})$,其中,$c(VM)=\sum_{i=1}^{n}\sum_{j=1}^{s_i}p(vm_{ij})$ 为虚拟机的使用成本,$c(L_{VM})=p_t(r_t(L_{VM}))$为虚拟机网络的通信成本.在这里,$p_t(T)$表示虚拟机网络通信计费价格函数,如何确定 $p_t(T)$,需要根据具体的网络通信计费模式来确定.本文的目标就是确定一种使得 $c(VN)$最小的 SaaS 构件放置

方案,因此,SaaS 构件优化放置问题可以描述为

$$\textbf{Min } c(VN)=c(VM)+c(L_{VM}) \tag{1}$$

Subject to:

$$S=\{s_1,s_2,...,s_n\},s_i\in N(i=1,2,\ldots,n) \tag{2}$$

$$VM=VM_1\cup VM_2\cup...\cup VM_n$$

$$VM_i=\begin{cases}\varnothing, & s_i=0\\ \{vm_{i1},vm_{i2},...,vm_{is_i}\}, & s_i>0\end{cases} \tag{3}$$

$$\rho(c)\in VM,\forall c\in C \tag{4}$$

$$C(vm_{ij})\neq\varnothing\wedge C(vm_{ij})\subseteq C,\forall vm_{ij}\in VU_B \tag{5}$$

$$E(l)\neq\varnothing\wedge E(l)\subseteq E,\forall l\in L_{VM} \tag{6}$$

$$\sum_{c\in C(vm_{ij})} r_c(c)\leqslant r_c(vm_{ij})-r_c^0(vm_{ij}),\forall vm_{ij}\in VM \tag{7}$$

$$\sum_{o\in C(vm_{ij})} r_m(c)\leqslant r_m(vm_{ij})-r_m^0(vm_{ij}),\forall vm_{ij}\in VM \tag{8}$$

$$\sum_{c\in C(vm_{ij})} r_d(c)\leqslant r_d(vm_{ij})-r_d^0(vm_{ij}),\forall vm_{ij}\in VM \tag{9}$$

在上面的形式化描述中:公式(1)为问题的优化目标,表示SaaS构件放置方案,即,虚拟机网络图 VN 的成本最低;公式(2)给出了一组非负整型决策变量 $S=\{s_1,s_2,...,s_n\}$,其中,$s_i\in N$ 表示由第 i 类虚拟机模板 vc_i 所创建的虚拟机数量,N 为非负整数集合;公式(3)是基于公式(2)的决策变量所构造的虚拟机集合 $VM=VM_1\cup VM_2\cup\ldots\cup VM_n$,其中,$VM_i$ 表示由第 i 类虚拟机模板 vc_i 所构造的虚拟机集合,$vm_{ij}(j=1,2,\ldots,s_i)$表示 VM_i 中第 j 个虚拟机,VM_i 和 vm_{ij} 是需要根据决策变量 S 来动态创建的;公式(4)表示对于 C 中的任何构件,都被放置到 VM 中的一个虚拟机上;公式(5)表示 VM 中的任何虚拟机所包含的构件都不能为空,并且都是 C 中的构件;公式(6)表示 L_{VM} 中的任何通信链路所包含的构件连接都不能为空,并且都是 E 中的构件连接;公式(7)到公式(9)是对虚拟机资源需求的约束,分别表示虚拟机所包含的构件的计算、内存和存储资源不能超过该虚拟机资源的提供能力,其中,$r_c^0(vm_{ij}),r_m^0(vm_{ij})$ 和 $r_d^0(vm_{ij})$ 分别表示 vm_{ij} 的计算、内存和存储资源预留量.虚拟机的资源利用率不能超过一定的阈值,因此需要预留一部分资源作为缓冲以保证应用运行的质量,虚拟机的资源预留量与虚拟机的类型有关.

3 基于 HGSA 的 SaaS 构件优化放置算法

针对 SaaS 构件优化放置问题,本文采用混合遗传模拟退火算法(HGSA)对其进行求解.与遗传算法类似,HGSA 也是从一组随机产生的初始解开始进行全局最优解搜索,它首先通过遗传算法(GA)中的选择和交叉操作来产生一组新个体,然后,针对每个新产生的个体采用模拟退火算法(SA)进行邻域搜索,并以其结果作为下一代种群中的个体.HGSA 将 GA 与 SA 的优点充分结合起来,从而提高了 SaaS 构件优化放置问题的求解质量.

3.1 解的编码

HGSA 采用基于构件的编码方式,染色体中每一个基因代表一个构件,染色体的长度为构件的数量$|C|$,基因的取值代表构件所放置的虚拟机,每个虚拟机采用一个二元组(i,j)来表示,其中,i 表示虚拟机类型编号,j 表示虚拟机编号.每一类虚拟机都是从 1 开始编号,并且要保持编号的连续性.令 P 表示一条染色体,P 可分为两个部分:已放置构件的基因片段 P_A 和新放置构件的基因片段 P_B,其中,P_A 中的每个基因代表 C_A 中的一个构件,P_B 中的每一个基因代表 C_B 中的一个构件.例如,针对图 1 所示的 SaaS 构件放置方案,其所对应的编码如图 2 所示.

1	2	3	4	5	6	7	8	9	10
(2,1)	(2,1)	(2,1)	(1,1)	(1,1)	(2,2)	(2,2)	(2,2)	(3,1)	(3,1)
c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}

Fig.2 Coding of SaaS component placement project

图 2 SaaS 构件放置方案编码

3.2 解初始化

解的初始化是指生成种群中的每一条初始染色体,本文采用基于贪心策略的启发式方法来生成初始染色体,具体步骤如下:

Step 1. 按照构件的编号顺序生成一条初始染色体 P,其中,P_A 中每个构件的基因取值为该构件所在的虚拟机编码,P_B 中的每个构件的基因取值为空.

Setp 2. 随机产生一个 VM_A 中虚拟机序列 S_{VM} 和 C_B 中构件序列 S_C.

Step 3. 按照顺序取出 S_C 中的每一个构件 c,依次检查 S_{VM} 中的每一个虚拟机是否能够放置 c:如果满足放置条件,则将构件 c 放置到一个匹配的虚拟机上(即,c 的基因取值为该虚拟机编号);如果 S_{VM} 中的每一个虚拟机都不能放置构件 c,则随机创建一个新的虚拟机,并将该虚拟机编号加入到序列 S_{VM} 的末尾,然后将构件 c 放置到新创建的虚拟机上(即,构件 c 的基因取值为新创建的虚拟机编号).重复上述过程,直到 S_C 中的每一个构件都被放置完成.

按照上述步骤,可以生成初始种群中的每一条染色体.解初始化的时间复杂度为 $O(s\cdot|C_B|\cdot|VM|)$,其中,s 为种群的数量,$|C_B|$为新放置构件数量,$|VM|$为虚拟机的数量上限.

3.3 新解产生规则

HGSA 采用 SA 中的邻域搜索策略来实现解的局部搜索能力.在 SA 中,新解产生是旧解向邻域搜索的结果,其搜索规则决定了邻域的范围.新解产生规则的设计与实际问题密切相关,而不是简单的随机搜索.针对本文所提出的 SaaS 构件优化放置问题,由已知解向邻域搜索的操作对象主要有构件和虚拟机,因此,我们设计了如下 4 种新解产生规则,其中,规则(1)和规则(2)属于构件粒度层次的邻域搜索,规则(3)和规则(4)属于虚拟机粒度层次的邻域搜索.

(1) 改变构件放置位置

随机选择一个基因 k,设其取值为(i,j),为 k 重新赋予一个新值(i',j'),其对应的放置操作为:将构件 c_k 从虚拟机 vm_{ij} 迁移到 $vm_{i'j'}$ 上,这要求 $vm_{i'j'}$ 的可用资源能够满足放置 c_k 的资源需求.如果虚拟机 vm_{ij} 中只包含 c_k 一个构件,则可以将 vm_{ij} 从虚拟机网络图 VN 中删除.图 3(a)描述了改变构件放置位置产生新解过程的示意图,它表示将构件 c_6 从虚拟机 vm_{22} 迁移到虚拟机 vm_{12} 上.

(2) 交换构件放置位置

随机选择两个取值不同的基因 k 和 l,交换 k 和 l 的值,其所对应的放置操作为:将构件 c_k 迁移到构件 c_l 所在的虚拟机上,同时将构件 c_l 迁移到构件 c_k 所在的虚拟机上,即,互换 c_k 和 c_l 的放置位置.迁移后的虚拟机应满足构件放置的资源约束条件.图 3(b)描述了交换构件放置位置产生新解过程的示意图,它表示将构件 c_5 从虚拟机 vm_{11} 取出放置到虚拟机 vm_{22} 中,同时将构件 c_8 从虚拟机 vm_{22} 取出放置到虚拟机 vm_{11} 中.

(3) 从高价虚拟机向低价虚拟机迁移

设 vm_{ij} 为虚拟机网络图 VN 中的一个虚拟机,若存在比 vm_{ij} 价格低且能够放置其上所有构件的虚拟机模板 $vc_{i'}$,则创建 $vc_{i'}$ 的一个实例 $vm_{i'j'}$,将 vm_{ij} 上的所有构件都迁移到 $vm_{i'j'}$ 上,并将所有取值为(i,j)的基因值都替换为(i',j'),然后,再将 vm_{ij} 从虚拟机网络图 VN 中删除.由于 $vm_{i'j'}$ 的成本要比 vm_{ij} 低,因此通过该操作虚拟机网络 VN 的成本降低了 $p(vm_{ij})-p(vm_{i'j'})$.图 3(c)描述了从高价虚拟机向低价虚拟机迁移过程示意图,它表示存在比虚拟机 vm_{31} 价格低且能够放置其上所有构件的虚拟机模板 vc_1,因此可以创建 vc_1 的一个实例 vm_{12}(由于已存在编号为 1 的虚拟机 vm_{11},为了保持编号的连续性,新创建的虚拟机编号应为 2),然后将 vm_{31} 中的构件 c_9 和 c_{10} 迁移到新创建的虚拟机 vm_{12} 中,最后,将 vm_{31} 从虚拟机网络图中删除.

(4) 从小型虚拟机向大型虚拟机迁移

设 vm_{ij} 为虚拟机网络图 VN 中的一个虚拟机,若存在能够放置 vm_{ij} 上所有构件的虚拟机 $vm_{i'j'}$,则将 vm_{ij} 上的所有构件都迁移到 $vm_{i'j'}$ 上,并将这些构件的基因值都替换为(i',j'),然后,将 vm_{ij} 从虚拟机网络图 VN 中删除.由于减少了一个虚拟机 vm_{ij},而且 vm_{ij} 与 $vm_{i'j'}$ 之间的通信由虚拟机之间转变为虚拟机内部(本文假定虚拟机内部的

通信不产生成本),因此,通过该操作虚拟机网络图 VN 的成本低了 $p(vm_{ij})+p_t(r_t(vm_{ij},vm_{i'j'}))$.图 3(d)描述了从小型虚拟机向大型虚拟机迁移过程的示意图,它表示存在一个能够放置虚拟机 vm_{21} 上所有构件的虚拟机 vm_{31},因此,可以将 vm_{21} 上的所有构件都迁移到 vm_{31} 上,并将 vm_{21} 从虚拟机网络图中删除.迁移后,vm_{31} 上包含 5 个构件 c_9,c_{10},c_1,c_2 和 c_3,其中,c_9 和 c_{10} 为迁移之前 vm_{31} 中已存在的构件,c_1,c_2 和 c_3 是从 vm_{21} 迁移过来的构件.

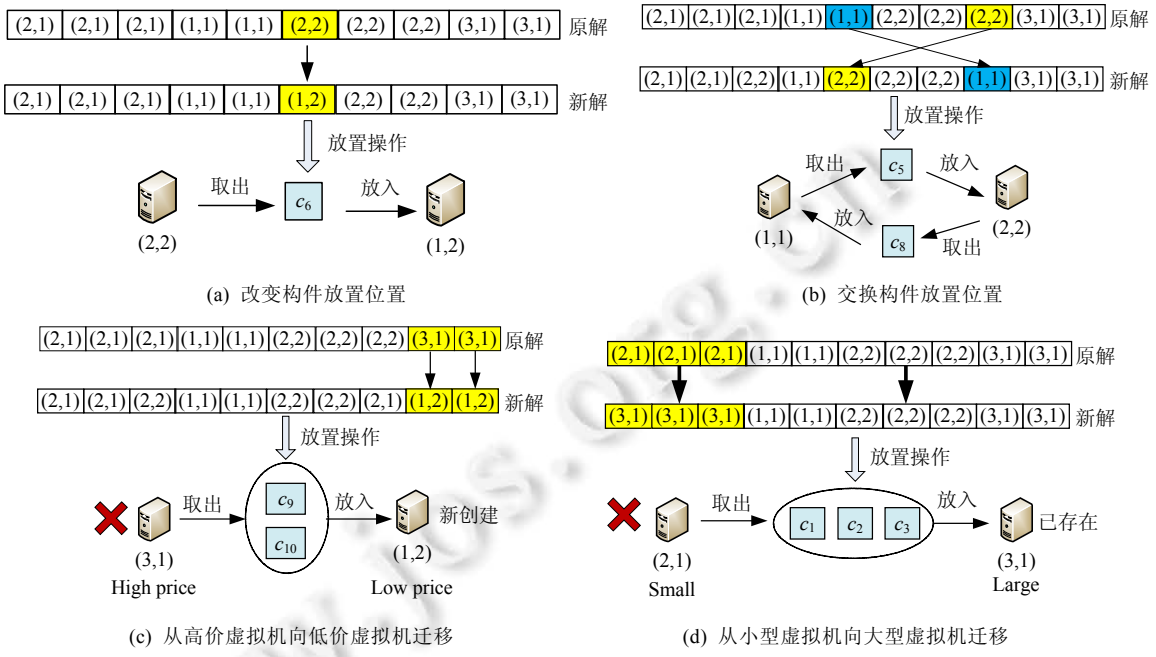

Fig.3 Four types rules of creating new solutions

图 3 4 种新解产生规则

3.4 选择与交叉操作

HGSA 采用遗传算法中的选择和交叉操作来提高解的全局搜索能力,其中,选择操作采用比例选择策略,交叉操作采用双切交叉法.

(1) 选择操作

选择操作采用比例选择策略和旋轮法来实现,其目的是使适应度高的个体具有更高的生存概率.

设 $P_i(i=1,2,\ldots,s)$为种群中的一个染色体,s 为种群规模,$f(P_i)$为 P_i 的适应度,适应度采用虚拟机网络图成本来表示.令 VN_i 为 P_i 所对应的虚拟机网络图,则 $f(P_i)=c(VN_i)$,$c(VN_i)$为 VN_i 的成本.为了便于选择操作,我们采用正规化技术将适应值映射到(0,1)区间.设 $F(P_i)$为映射后的正规化适应度,则有:

$$F(P_i)=\frac{f_{\{\max\}}-f(P_i)+r}{f_{\{\max\}}-f_{\{\min\}}+r},$$

其中,$f_{\{\min\}}$和 $f_{\{\max\}}$分别为种群中的最小和最大适应度;$r\in(0,1)$是一个常数,它可以根据实际情况进行设定,我们设定 r=0.8.根据比例选择策略,P_i 被选择的概率为 $F(P_i)\Big/\sum_{i=1}^{s}F(P_i)$.得到每个染色体的选择概率后,采用旋轮法来实现选择操作,共旋转 s 次,每次转轮时,随机产生$\xi_k\in(0,1)$,如果 $\sum_{j=1}^{i-1}F(P_i)\Big/\sum_{i=1}^{s}F(P_i)\leqslant\xi_k<\sum_{j=1}^{i}F(P_i)\Big/\sum_{i=1}^{s}F(P_i)$,则选择染色体 P_i.

(2) 交叉操作

交叉操作采用双切交叉法,对于两个选定的染色体 P_1 和 P_2,随机选取两个切点 k_1 和 k_2,交换 k_1 和 k_2 之间的

基因编码,生成两个新的染色体 P_1' 和 P_2'.图 4 描述了交叉操作过程的示意图.

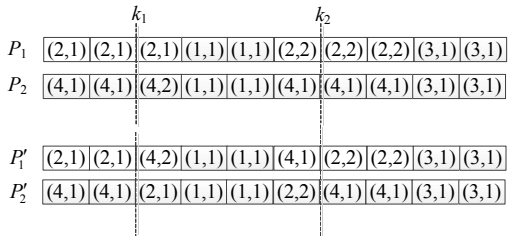

Fig.4 Crossover operation

图 4 交叉操作

如果新生成的染色体不满足虚拟机资源约束条件,则需要对其进行修复,染色体修复过程如下:

Step 1. 针对每个不满足资源约束的虚拟机,生成该虚拟机所放置构件的一个序列,然后,按照该序列依次取出每个构件,直到虚拟机满足资源约束为止.

Step 2. 对于从所有不满足资源约束条件的虚拟机取出的所有构件,随机生成一个构件序列 S_C.同时,对于所有已经使用的虚拟机,随机生成一个虚拟机序列 S_{VM}.

Step 3. 按照顺序取出 S_C 中的每一个构件 c,依次检查 S_{VM} 中的每一个虚拟机是否能够放置 c:如果满足放置条件,则将构件 c 放置到第 1 个匹配的虚拟机上(即,c 的基因取值为该虚拟机编号);如果 S_{VM} 中的每一个虚拟机都不能放置构件 c,则随机创建一个新的虚拟机,并将该虚拟机编号加入到序列 S_{VM} 的末尾,然后将构件 c 放置到新创建的虚拟机上(即,构件 c 的基因取值为新创建的虚拟机编号).重复上述过程,直到 S_C 中的每一个构件都被放置完成.

3.5 算法流程

基于 HGSA 的 SaaS 构件优化放置算法流程如图 5 所示,其具体步骤如下:

Step 1. 设置算法参数,包括种群规模 s、初始温度 T_{high}、退火系数α、终止温度 T_{low}、交叉概率 p_c、内循环次数 l、最优染色体 P_{best}.

Step 2. 设置当前温度 $t=T_{high}$.

Step 3. 根据第 3.1 节所提出的种群初始化操作,产生初始种群 $Pop=\{P_1,P_2,\ldots,P_s\}$,并将 Pop 中的最优解保存在 P_{best} 中.

Step 4. 设置当前内循环次数 $n=0$.

Step 5. 根据第 3.2 节所提出的比例选择策略,从当前种群 Pop 中选择 s 个染色体,并用其替换原染色体.

Step 6. 按照交叉概率 p_c,从当前种群 Pop 中选择$\lfloor s/2\rfloor$对染色体进行交叉操作,产生$\lfloor s/2\rfloor$对新染色体,令 $Pop'=\{P_1',P_2',\ldots,P_s'\}$ 为新染色体的集合,其中,P_i' 为 P_i 的子染色体.对于 Pop'中每个不满足资源约束条件的染色体,按照第 3.4 节所提出的策略对其进行修复.

Step 7. 对于新种群 Pop'中的每个染色体 P_i' ($i=1,2,\ldots,s$),令 VN_i 为 P_i' 所对应的虚拟机网络图,VM_i 为 VN_i 中所包含的虚拟机集合,依次检索 VM_i 中的每个虚拟机,然后按照第 3.3 节所提出的规则(3)和规则(4)对其进行迁移操作,产生新的染色体 P_i'',如果 $f(P_i'')<f(P_i')$,则用 P_i'' 替换 P_i';如果 $f(P_i')<f(P_{best})$,则用 P_i' 替换 P_{best}.

Step 8. 对于新种群 Pop'中的每个染色体 P_i' ($i=1,2,\ldots,s$),利用 Metropolis 准则判断其是否替换当前种群 Pop 中其所对应的父染色体 P_i,令 $\Delta f=f(P_i')-f(P_i)$:

- 如果$\Delta f<0$,则用 P_i' 替换 P_i;
- 否则,产生一个随机数$\xi\in(0,1)$:如果 $\exp(-\Delta f/(\alpha\cdot t))>\xi$,则用 P_i' 替换 P_i;如果 $f(P_i)<f(P_{best})$,则用 P_i 替换 P_{best}.

Step 9. 对于当前种群 Pop 中的每个染色体 $P_i(i=1,2,\ldots,s)$,按照第 3.3 节所提出的规则(1)和规则(2)对其进行构件的迁移和交换操作,产生新的染色体 P_i',利用 Metropolis 准则判断是否接受该新染色体 P_i'.令 $\Delta f = f(P_i') - f(P_i)$:

- 如果$\Delta f<0$,则用 P_i' 替换 P_i;
- 否则,产生一个随机数$\xi\in(0,1)$:如果 $\exp(-\Delta f/(\alpha\cdot t))>\xi$,则用 P_i' 替换 P_i;如果 $f(P_i)<f(P_{best})$,则用 P_i 替换 P_{best}.

Step 10. 令 $n=n+1$,如果 $n<l$,转到 Step 5;否则,执行 Step 11.

Step 11. 令 $t=\alpha\cdot t$,如果 $t>T_{low}$,转到 Step 4;否则,执行 Step 12.

Step 12. 输出最优解 P_{best}.

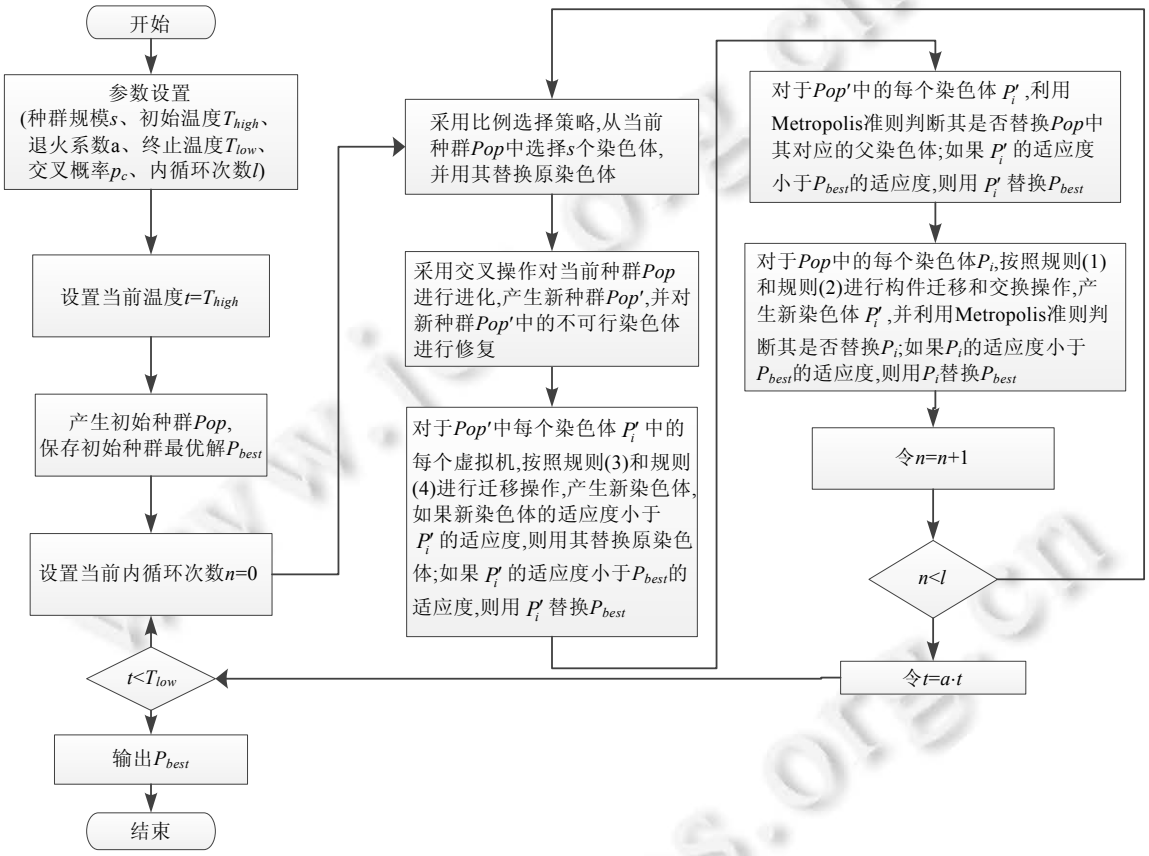

Fig.5 Process diagram of SaaS component optimal placement based on HGSA

图 5 基于 HGSA 的 SaaS 构件优化放置算法流程图

上述算法的时间复杂度为 $O\left(s\cdot l\cdot\log_{\alpha}\left(\frac{T_{low}}{T_{high}}\right)\cdot|C|\cdot|VM|^2\right)$,其中,$|C|$为构件的数量,$|VM|$为所使用的虚拟机数量.由于算法每次执行所产生的虚拟机数量是不确定的,因此,$|VM|$不是固定的常量,$|MV|<\sum_{i=1}^{n}n_{\max}(vc_i)$,其中,$n_{\max}(vc_i)$表示只选择虚拟模板 vc_i 所使用的最大虚拟机数量.对于给定的构件模型和虚拟机模板的集合,该值可以通过实验来确定.

4 实验分析

4.1 算法输入

基于HGSA的SaaS构件优化算法的输入为IaaS提供商所发布的虚拟机模板和SaaS提供商所要部署的构件模型.在本实验中,虚拟机模板参考亚马逊EC2的虚拟机模板类型进行设置,我们主要考虑CPU、内存和存储这3种资源.在EC2中,不同虚拟机模板的资源数量和价格是不同的,我们选择了13种虚拟机模板作为实验对象,其配置见表1.其中,CPU的单位为CU(compute unit),内存的单位是GiB(2^{30}字节),存储的单位是GB,虚拟机价格的单位为$/h,网络通信的价格为0.01$/GB.虚拟机模板的数量可以根据实际需要动态地进行调整.

Table 1 Parameters of EC2 virtual machine template

表1 EC2虚拟机模板参数

类型	vCPU(CU)	内存(GiB)	存储(GB)	价格($/h)
M3.medium	1	3.75	4	0.161
M3.large	2	7.5	32	0.322
M3.xlarge	4	15	80	0.644
M3.2xlarge	8	30	160	1.288
C3.large	2	3.75	32	0.238
C3.xlarge	4	7.5	80	0.477
C3.2xlarge	8	15	160	0.953
C3.4xlarge	16	30	320	1.906
C3.8xlarge	32	60	640	3.813
I2.xlarge	4	30.5	800	1.169
I2.2xlarge	8	61	21 600	2.337
I2.4xlarge	16	122	3 200	4.674
I2.8xlarge	32	244	6 400	9.348

由于不同的SaaS应用其构件模型是不同的,为了验证本文所提出的算法在解决大规模SaaS应用优化放置方面的处理能力,我们采用仿真的方式动态生成构件模型.在本实验中,构件数量的模拟范围为[10,100],对于每个构件,其CPU需求数量的模拟范围为[0.2,2],内存需求数量的模拟范围为[0.5,3],存储需求数量的范围为[50,200],构件之间的连接关系按照概率[0,0.4]随机生成,其通信量模拟范围为[1,5].EC2的计费方式主要分为预留和按需两种方式,这里,我们主要考虑按需计费方式,并设定租期为1天(24h).

4.2 算法参数设置

HGSA中的参数包括种群规模s、初始温度T_{high}、终止温度T_{low}、退火系数α、内循环次数l和交叉概率p_c.不同的参数对算法的性能和质量具有不同的影响,所以首先要对各种参数进行分析,确定每种参数的变化趋势和最佳的取值范围.

(1) 种群规模s

一般来说,种群规模越大越好,因为如果种群太小容易使算法发生早熟,但是种群规模的扩大也将导致运算时间的延长,从而降低了算法的性能.在本实验中,我们设定种群规模为100.

(2) 初始温度T_{high}

一般来说,初始温度要足够大,以此来保证算法在开始时处于平衡状态.但是如果初始温度过高,则接受不可行解的概率就会增大,从而导致算法的运行时间过长.本文采用多次随机搜索所得到的适应函数的平均增量方法来确定初始温度.通过逆推Metropolis准则接受概率公式可得$T_{high}=\frac{-\overline{\Delta f}}{\ln(p_0)}$,其中,$p_0$为初始接受概率,$\overline{\Delta f}$为初始搜索所得到的适应度平均增量.我们设定$p_0$=0.8,通过反复实验分析,最终确定$T_{high}$=250.

(3) 退火系数α

退火系数α控制温度下降的方式,对于退火系数,通常在0.5~0.95之间取值.通过实验分析发现:在本问题中,退火系数对于解的影响远小于其他参数.因此,根据经验值可设置α=0.9.

(4) 内循环次数l

内循环是指在一个给定的温度下,通过 Metropolis 准则进行随机搜索,最终达到一种平衡状态的过程.为了保证在每一个温度下算法都能够达到平衡状态,内循环的次数 l 一般要足够大,但是如果 l 过大,则会导致邻域搜索时间过长,从而影响算法的性能.为了确定一个较为合适的内循环次数,我们在一系列终止温度条件下,通过不断地增加 l 取值来观测当前种群全局最优解的变化趋势(执行算法 100 次,取所有次数的平均值).通过实验分析发现:当构件数量较少时(≤40),内循环次数 l 对解的质量影响较小.这是因为,当构件数量较少时,虚拟机的使用成本占总成本的比例较高,而 HGSA 可以快速地搜索到成本最低的虚拟机.图 6 给出在构件数量 n 分别为 50,60,70,80,90 和 100、种群规模 s=100、初始温度 T_{high}=250、退火系数 α=0.9、交叉概率 p_c=0.9、终止温度 T_{low} 分别为 200,100,10,1,0.1 和 0.01 时的种群全局最优适应度(成本)变化趋势曲线图.从这些曲线可以看出:当 l 增加至 6 以后,每种终止温度下种群最优适应度(成本)的下降趋势不大,因此可以设置 $l \geq 6$.l 的具体取值可以根据实际问题的规模来设定.

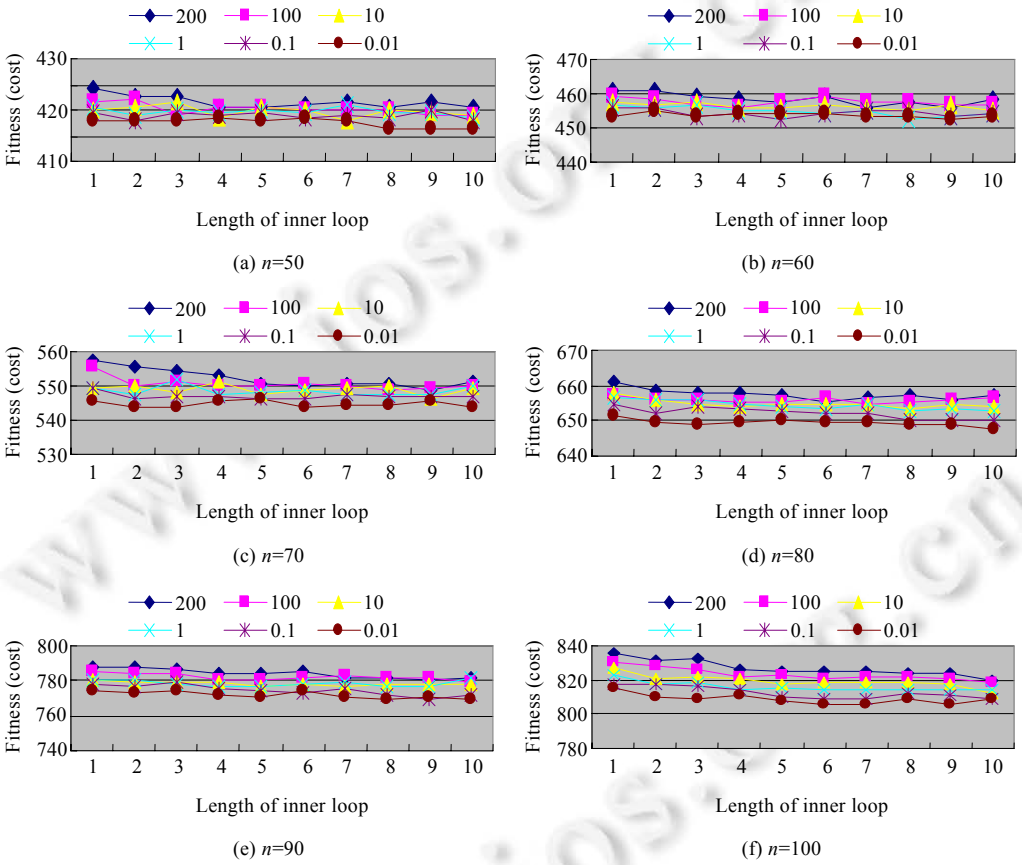

Fig.6 Tendency of fitness vary with length of inner loop

图 6 适应度随内循环次数变化趋势

(5) 终止温度 T_{low}

当初始温度 T_{high} 和退火系数 α 确定以后,可以通过终止温度 T_{low} 来控制外循环的执行次数,终止温度可以设定为一个很小的正数.一般来说,终止温度要足够小,以保证算法有足够的时间来获取全局最优/好解.然而,如果终止温度过小则会增加中间温度的数量,进而导致过多的邻域搜索,增加了算法的执行时间.为了确定一个较为合理的终止温度,我们通过不断降低 T_{low} 的取值来观测当前种群全局最优解的变化趋势(执行算法 100 次,取所有次数的平均值).图 7 给出了在构件数量 n 分别为 50,60,70,80,90 和 100、种群规模 s=100、初始温度 T_{high}=250、

退火系数α=0.9、内循环次数 l=6、交叉概率 p_c=0.9 时的种群全局最优适应度(成本)随终止温度变化趋势曲线图.从这些曲线图可以看出:当 T_{low} 降低至 0.01 以后,种群最优适应度(成本)的下降趋势不大,因此可以设置 $T_{low}\leqslant 0.01$,T_{low} 的具体取值可以根据实际问题的规模来设定.

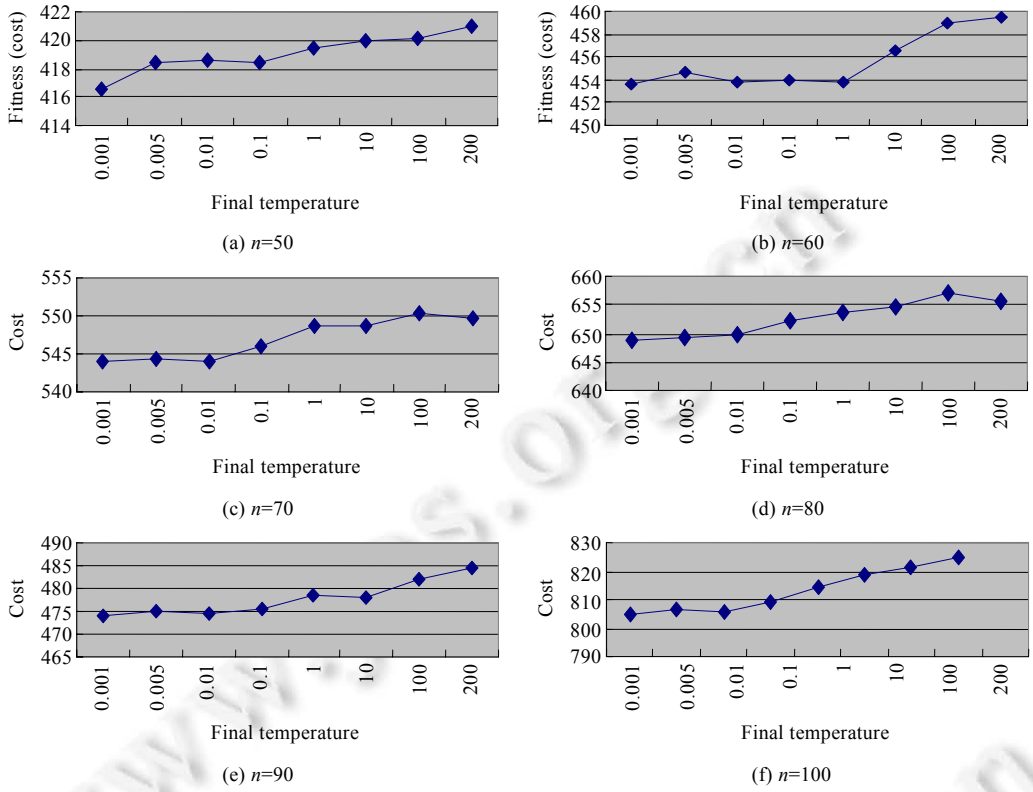

Fig.7 Tendency of fitness vary with end temperatures

图 7 适应度随终止温度变化趋势

(6) 交叉概率 p_c

交叉概率 P_c 为各代中交叉产生的后代数与种群中的个体数的比值.一般来说,较高的交叉概率将会达到更大的解空间,从而减少停止在非最优解上的机会.但是,如果交叉概率太高,则会因过多地搜索不必要的解空间而耗费大量的计算时间,从而降低了算法的性能.在 HGSA 中,交叉概率的设置与终止温度 T_{high} 密切相关:如果 T_{high} 的值较高,则 p_c 的值应该设置得低一些,这样可以加快收敛的速度;反之,p_c 的值可以设置得高一些,因为这样有更多的机会进行解空间的搜索.图 8 给出了在构件数量为 100、种群规模 s=100、初始温度 T_{high}=250、退火系数α=0.9、内循环次数 l=6、终止温度 T_{high}=0.01 时的种群全局最优适应度(成本)变化趋势曲线图.

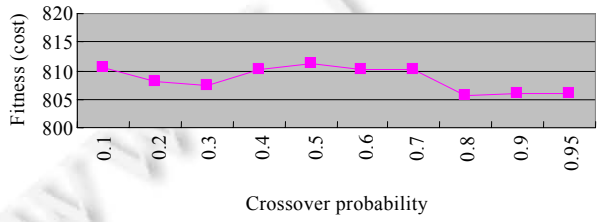

Fig.8 Tendency of fitness vary with crossover probabilities

图 8 适应度随交叉概率变化趋势

根据该曲线可以看出:当交叉概率较小和较大时,解的质量较好;当交叉概率大于 0.8 以后,种群最优适应度的下降趋势不大.因此,可以设定 $p_c \geqslant 0.8$.

4.3 对比实验

为了验证本文所提出的混合遗传模拟退火算法(HGSA)在求解 SaaS 构件优化放置问题方面的有效性,我们又单独采用了遗传算法(GA)和模拟退火算法(SA)对该问题进行了求解.

在 GA 中,由于虚拟机的数量是不确定的,因此,通过交叉操作所产生的新染色体的质量不但没有改进反而在不断地下降.这是因为,交叉操作会增加虚拟机的数量,导致构件的放置更加稀疏,从而增加了放置方案的成本.为了提高 GA 的求解质量,我们采用本文所提出的新解产生规则(3)和规则(4)对交叉操作后所产生的新解进行了优化,通过实验分析发现其求解质量已显著提高.

传统的 SA 是从一个初始解开始不断进行邻域搜索来产生新解,因此,初始解的质量对算法的影响比较大.为了提高 SA 的求解质量,我们从初始种群中选择一个最优解作为 SA 的初始解.在传统的 SA 中,邻域搜索是通过改变染色体中某个基因的取值来实现的,该操作相当于本文所提出的邻域搜索规则(1),即,改变构件的放置位置.这一操作的收敛速度比较慢,而且很容易陷入局部最优/好解.为此,我们又将其他 3 个邻域搜索规则都加入到 SA 中,从而极大地提高了 SA 的求解质量.

另外,本文所提出的解初始化方法采用的是一种基于贪心策略的启发式算法(HA),即,每次放置构件都是尽量选择已经存在的虚拟机,这样可以减少虚拟机的使用数量.单独采用 HA 进行问题求解的运行效率比较高,但是解的质量较差.为了提高 HA 的求解质量,我们首先采用 HA 来生成一组解(种群),然后,从中选择一个最优解(相当于 SA 的初始解)作为 HA 的最终解.

本文通过实验对上述 4 种算法(HGSA,GA,SA 和 HA)的求解质量进行了比较.我们采用模拟的方式随机生成了 10 个构件模型,其构件数量分别是 10,20,…,100,每个构件的资源需求数量以及构件之间通信量按照第 4.1 节所给出的范围进行随机均匀生成.针对每一构件模型,我们分别采用上述 4 种算法对其进行求解 30 次取平均值.为了体现实验的公平性,每次求解它们都是从同一初始种群开始,其中,SA 的初始解是该种群中的最优解,HGSA 和 GA 的进化代数和交叉概率是一致的(s=100,p_c=0.9),GA 的变异概率设置为 0.1.图 9 给出了 4 种算法所计算的总成本(虚拟机使用成本+网络通信成本)的对比分析图.

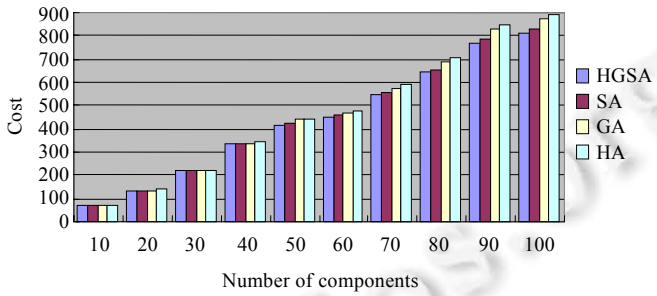

Fig.9 Comparison of total cost calculated by four algorithms

图 9 4 种算法所计算的总成本比较

通过实验分析发现:对于任意给定的初始种群,HGSA 求解质量最好,其次是 SA,再次是 GA,最后是 HA.由于 SA 是在 HA 的基础上继续进行邻域搜索,因此从理论上来说,SA 的求解质量至少不会比 HA 差.由于 HGSA 是采用多个解在其邻域内并行搜索的方式,并且利用交叉操作来提高全局搜索能力,因此其求解质量要高于 SA.与 HGSA 和 SA 相比,GA 对该问题的求解能力较弱,这主要是由于虚拟机数量的不确定性而导致搜索范围扩大,尽管我们希望通过相应的邻域搜索规则来改善新解的质量,但其效果仍不理想.

表 2 和表 3 分别给出了在每一构件数量下,4 种算法所计算的虚拟机使用成本和网络通信成本的具体数值.从实验结果可以看出:在每一个构件数量下,HGSA 所计算的虚拟机使用成本和网络通信成本都要低于其他 3

种算法;而且随着构件数量的增加,其求解质量显著提高.这说明HGSA在求解SaaS构件优化放置问题方面具有较好的效果.

Table 2 Comparison of virtual machine usage cost calculated by four algorithms

表 2 4 种算法所计算的虚拟机使用成本比较

	10	20	30	40	50	60	70	80	90	100
HGSA	67.53	135.48	214.84	322.27	399.38	423.93	501.14	587.31	695.33	713.64
SA	68.10	135.48	215.32	324.29	402.88	428.22	509.14	595.07	702.15	724.81
GA	67.53	135.48	215.32	325.34	415.60	434.91	521.45	622.67	741.77	757.01
HA	71.02	136.36	216.20	329.35	422.83	438.83	531.50	637.67	761.98	783.30

Table 3 Comparison of network communication cost calculated by four algorithms

表 3 4 种算法所计算的网络通信成本比较

	10	20	30	40	50	60	70	80	90	100
HGSA	0.35	0.97	4.41	10.33	18.92	29.45	43.09	56.55	75.76	94.49
SA	0.44	1.15	4.94	12.07	20.68	31.60	46.22	60.80	81.06	101.10
GA	0.35	1.22	5.71	13.13	22.41	33.98	49.12	64.80	86.11	106.62
HA	0.57	1.32	5.71	13.33	22.23	33.94	48.50	64.46	84.68	105.47

4.4 实例分析

本节通过图 1 所示的实例来说明 HGSA 在求解 SaaS 构件优化放置方面的能力.在该实例中,包含 10 个构件,每个构件的资源需求以及构件之间的通信如图10上半部分所示,虚拟机模板采用亚马逊的EC2(具体配置见表 1).利用本文所提出的 HGSA 算法(算法参数设置为 s=100,α=0.9,T_{high}=250,T_{low}=0.01,l=6,p_c=0.9),一次即可求得最优 SaaS 构件放置方案,如图 10 下半部分所示.在该方案中,我们选择了 2 台虚拟机,它们分别是 C3.xlarge 和 I2.2xlarge,其中,I2.2xlarge 上放置构件 c5,C3.xlarge 上放置其他所有构件,假设租期为 72h,网络通信价格为 0.01$/GB,则最终成本为 67.602$.其中,虚拟机使用成本为 67.536$,网络通信成本为 0.066$.这说明,本文所提出的算法在虚拟机数量不确定的情况下具有较好的效果.

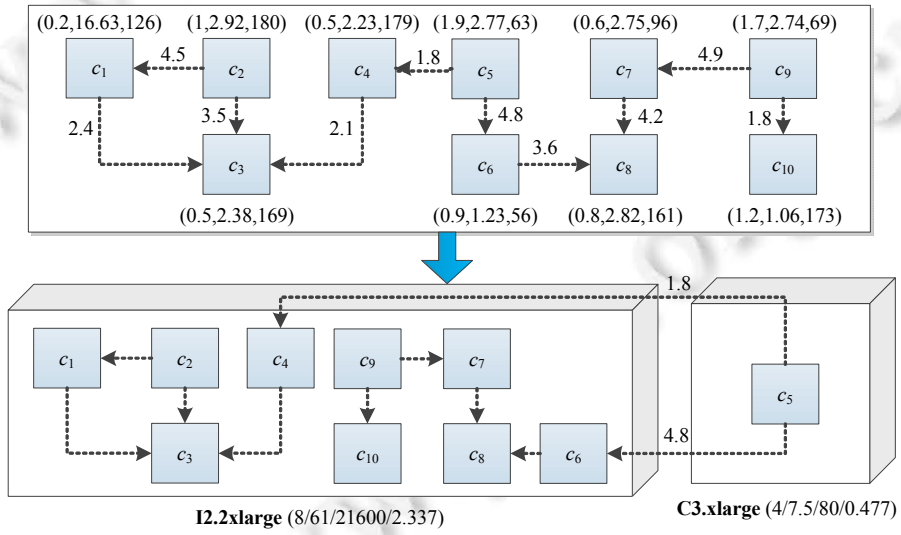

Fig.10 Txample of SaaS component optimal placement

图 10 SaaS 构件优化放置实例

5 结束语

针对传统软件提供商在向 SaaS 服务模式迁移时所遇到的 SaaS 构件放置问题,本文提出了一种基于混合遗

传模拟退火算法的 SaaS 构件优化放置方法,其优化目标是提高云资源利用率、降低云资源使用成本.尽管目前存在许多 SaaS 优化放置方法,但是这些方法都假定云环境中虚拟机的类型和数量是确定的,即,在限定的资源上进行优化放置;然而实际情况是:虚拟机的类型和数量一般都是不确定的,需要根据被部署的构件资源需求来确定.而本文所提出的 SaaS 构件优化放置模型弥补了这些方法的不足.本文采用混合遗传模拟退火算法 HGSA 对 SaaS 构件优化放置问题进行求解,HGSA 算法结合了遗传算法和模拟退火算法的优点,克服了 GA 收敛速度慢和 SA 容易陷入局部最优的缺点,与单独使用遗传算法和模拟退火算法相比,HGSA 算法具有更高的求解质量.从云架构角度来看,本文所提出的放置方法不仅适用于公有云基础设施的放置,同时也适用于私有云基础设施的放置.从软件生命周期阶段来看,本文所提出的放置方法不仅适用于 SaaS 的初始部署,而且也适用于 SaaS 运行阶段的动态部署.

References:

[1] Kang S, Kang S, Hur S. A design of the conceptual architecture for a multitenant SaaS application platform. In: Proc. of the Int'l Conf. on Computers, Networks, Systems, and Industrial Engineering. IEEE Computer Society Press, 2011. 462−467. [doi: 10.1109/CNSI.2011.56]

[2] Zhang Y, Wang ZH, Gao B, Guo CJ, Sun W, Li XP. An effective heuristic for on-line tenant placement problem in SaaS. In: Proc. of the 2010 IEEE Int'l Conf. on Web Services. IEEE Computer Society Press, 2010. 425−432. [doi: 10.1109/ICWS.2010.65]

[3] Yu HY, Wang DH. System resource allocation algorithm for multi-tenant SaaS application. In: Proc of the 2011 Int'l Conf. on Cloud and Service Computing. IEEE Computer Society Press, 2011. 207−211. [doi: 10.1109/CSC.2011.6138523]

[4] Wu LL, Garg SK, Buyya R. SLA-Based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences, 2012,78(5):1280−1299. [doi: 10.1016/j.jcss.2011.12.014]

[5] Yang EF, Zhang Y, Wu L, Liu YL, Liu SJ. A hybrid approach to placement of tenants for service-based multi-tenant SaaS application. In: Proc. of the 2011 IEEE Asia-Pacific Services Computing Conf. IEEE Computer Society Press, 2011. 124−130. [doi: 10.1109/APSCC.2011.35]

[6] Tian C, Jiang HB, Iyengar A, Liu X, Wu ZD, Chen JH, Liu WY, Wang CG. Improving application placement for cluster-based Web applications. IEEE Trans. on Network and Service Management, 2011,8(2):104−115. [doi: 10.1109/TNSM.2011.050311.100040]

[7] Yusoh ZIM, Tang M. Composite SaaS placement and resource optimization in cloud computing using evolutionary algorithms. In: Proc. of the 2012 IEEE 5th Int'l Conf. on the Cloud Computing. IEEE Computer Society Press, 2012. 590−597. [doi: 10.1109/CLOUD.2012.61]

[8] Lloyd W, Pallickara S, David O, LyonbAuthor J, ArabibAuthor M, Rojas K. Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: Towards performance modeling. Future Generation Computer Systems, 2013, 29(5):1254−1264. [doi: 10.1016/j.future.2012.12.007]

[9] Moens H, Truyen E, Walraven S, Joosen W, Dhoedt B, De Turck F. Cost-Effective feature placement of customizable multi-tenant application in the cloud. Journal of Network System Management, 2014,22(4):517−588. [doi: 10.1007/s10922-013-9265-5]

[10] Zhu XY, Santos C, Beyer D, Ward J, Singhal S. Automated application component placement in data centers using mathematical programming. Int'l Journal of Network Management, 2008,18:467−483. [doi: 10.1002/nem.707]

[11] Jin ZH, Cao J, Li ML. A distributed application component placement approach for cloud computing environment. In: Proc. of the 9th IEEE Int'l Conf. on Dependable, Automic and Secure Computing. IEEE Computer Society Press, 2011. 488−495. [doi: 10.1109/DASC.2011.94]

[12] Sailer A, Head MR, Kochut A, Shaikh H. Graph-Based cloud service placement. In: Proc. of the 2010 IEEE Int'l Conf. on Services Computing. IEEE Computer Society Press, 2010. 89−96. [doi: 10.1109/SCC.2010.67]

[13] Pisinger D. An exact algorithm for large multiple knapsack problem. European Journal of Operational Research, 1999,114(3):528−541. [doi: 10.1016/S0377-2217(98)00120-9]

[14] Kataoka S, Yamada T. Upper and lower bounding procedures for the multiple knapsack assignment problem. European Journal of Operational Research, 2014,237(2):440−447. [doi: 10.1016/j.ejor.2014.02.014]

[15] Sarac T, Sipahioglu A. Generalized quadratic multiple knapsack problem and two solution approaches. Computer & Operations Research, 2014,43:78−89. [doi: 10.1016/j.cor.2013.08.018]

[16] Ren ZG, Feng ZR, Zhang AM. Fusing ant colony optimization with Lagrangian relaxation for the multiple-choice multidimensional knapsack problem. Information Science, 2012,182(1):15−29. [doi: 10.1016/j.ins.2011.07.033]

[17] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002,6(2):182−197. [doi: 10.1109/4235.996017]

[18] McCall J. Genetic algorithms for modelling and optimization. Journal of Computational and Applied Mathematics, 2005,184(1): 205−222. [doi: 10.1016/j.cam.2004.07.034]

[19] Chen D, Lee CY, Park CH. Hybrid genetic algorithm and simulated annealing (HGASA) in global function optimization. In: Proc. of the 17th IEEE Int'l Conf. on Tools with Artificial Intelligence. IEEE Computer Society Press, 2005. 113−117. [doi: 10.1109/ICTAI.2005.72]

[20] Li WD, Ong SK, Nee AYC. Hybrid genetic algorithm and simulated annealing approach for the optimization of process plans for prismatic parts. Int'l Journal of Production Research, 2002,40(8):1899−1922. [doi:10.1080/00207540110119991]

[21] Loukil L, Mehdi M, Mebab N. A parallel hybrid genetic algorithm-simulated annealing for solving Q3AP on computational grid. In: Proc. of the IEEE Int'l Symp. on Parallel & Distributed Processing. IEEE Computer Society Press, 2009. 1−8. [doi: 10.1109/IPDPS.2009.5161126]

[22] Li YH, Guo H, Wang L, Fu J. A hybrid genetic-simulated annealing algorithm for the location-inventory-routing problem considering returns under E-supply chain environment. The Scientific World Journal, 2013. [doi: 10.1155/2013/125893]

孟凡超(1974－),男,黑龙江依兰人,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件体系结构,云计算,服务计算.

李克秋(1971－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为网络,云计算,数据中心.

初佃辉(1969－),男,博士,教授,CCF 高级会员,主要研究领域为服务计算与服务工程,面向服务的软件体系结构.

周学权(1979－),男,讲师,主要研究领域为云计算,SaaS.