基于蚁群算法的面向服务软件的部署优化方法

李 琳,应 时,赵 翀,董 波
(1. 武汉大学软件工程国家重点实验室,湖北武汉 430072;2. 武汉大学计算机学院,湖北武汉 430072)

摘 要: 面向服务软件的部署优化问题是典型的 NP 难题. 本文构建了基于性能改善的软件部署优化模型,设计了一种蚁群优化算法 ACO-DO 进行近似最优解的快速求解. 该算法通过设计基于部署优化问题的启发式、改进部署方案的构建顺序、增加局部搜索过程实现蚁群算法求解效率的提升. 通过不同规模的实例实验,验证了 ACO-DO 算法能够取得比现有的混合整数线性规划算法、蚁群算法和遗传算法更好的性能.

关键词: 面向服务的软件; 部署优化; 蚁群算法; 性能
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2016)01-0123-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.01.018

Deployment Optimization of Service-Oriented Software Based on Ant Colony Algorithm

LI Lin, YING Shi, ZHAO Chong, DONG Bo
(1. *State Key Lab of Software Engineering, Wuhan University, Wuhan, Hubei* 430072, *China*;
2. *Computer School, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: The deployment optimization of service-oriented software is well known to be NP hard. In this paper, a software deployment optimization model is built for improving the performance of service-oriented software, and an Ant Colony Algorithm for Deployment Optimization (ACO-DO) is designed to solve it so that the near-optimal solutions can be obtained quickly. The algorithm improves ant colony algorithm by designing a heuristic based on the considered problem, optimizing the orders of constructing deployment solutions and adding a local search procedure. A series of instances with different sizes are tested and analyzed. The experimental results show that the designed ACO-DO algorithm performs better than the existing Mixed Integer Linear Programming, ant colony and genetic algorithms.

Key words: service-oriented software; deployment optimization; ant colony algorithm; performance

1 引言

应用和系统环境的变化将会使面向服务的软件在长时间运行后出现性能降级问题,对软件部署方案进行调整与优化是解决该问题的常用方法[1]. 由于可能的部署方案总数通常非常巨大,探索可行的部署空间需耗费大量的计算成本;且评估性能的指标之间以及不同用户对同一指标的评价之间常常存在冲突[2],搜寻最优的软件部署方案常常需要衡量这些冲突因素. 因此,设计和开发能够表现优异性能的部署方案成为一项具有挑战性的任务.

目前,国内外专家学者在软件部署方案的优化与评估方面开展了大量研究,提出了许多创新性的解决方法,包括利用多目标遗传算法 E^3-R[3] 解决面向服务等级协议的服务部署优化问题,利用 CDOXplore 算法[4] 处理云环境下的软件部署优化问题,基于结构化约束敏感的软件部署方法[5] 实现大规模的软件部署与优化过程,基于用户偏好与多维 QoS 指标相结合的可扩展的部署优化框架[6],基于启发式装箱算法和粒子群演化算法相结合的混合型空间部署优化算法 ScatterD[7],基于进化优化的移动感知节点部署算法[8] 从网络覆盖和能量消耗两个方面优化移动传感器网络中节点部署,以满足用户个性化需求为目标的基于体系结构驱动和网络化移动应用的客户端软件部署方案优化方法 ADCA[9],基于 P-ACO 蚁群算法自动搜索嵌入式软件系统的部署优化策略[10]. 这些方法虽然都可以实

收稿日期:2014-09-25;修回日期:2015-03-12;责任编辑:孙瑶
基金项目:国家自然科学基金(No. 61373038, No. 61070012);国家 863 高技术研究发展计划(No. 2012AA011204-01)

现软件部署优化过程,但针对软件性能优化的部署框架与模型的研究还很少,且优化算法主要采用传统的演化算法,而构造性算法作为一种最快的近似方法[11],目前还没在软件部署优化领域得到同等的重视.

ACO 算法是构造性算法的典型代表,是专门用来求解复杂组合优化难题的,目前已在诸多领域取得了很好的求解效果[12]. ACO 算法能够使用基于问题导向的启发式,指导蚂蚁快速朝向全局最优解的方向前进[13]. 因此,本文将 ACO 算法引入面向服务的软件部署优化问题之中,研究和设计了基于 ACO 算法和单目标优化策略的面向服务软件部署优化方法 ACO-DO (Ant Colony Optimization for Deployment Optimization, ACO-DO). 该方法通过设计问题的启发式,提高算法的收敛速度;通过考虑方案组件的选择顺序,提高部署方案的构建质量;通过增加局部搜索过程,避免过早收敛于局部最优解.

2 问题模型

本文构建的部署优化模型主要包含四部分:应用软件模型(Application Software Model, ASM)、运行平台模型(Runtime Platform Model, RPM)、应用场景(Application Scenarios, AS)、约束(Constraint, CON). 其中,ASM 提供部署的软件信息,RPM 提供用于部署的硬件节点信息,AS 提供软件性能评估的其它相关因素信息,CON 提供软件部署过程中所需满足的约束条件.

定义 1 应用软件模型 ASM 是一个五元组 ASM = (S, TR, SR, CW, ES):

① $S = \{s_1, s_2, \cdots, s_{SN}\}$, $SN \in \mathbf{N}$,为一个非空有限服务集,表示组成软件的所有服务.

② $TR: S \rightarrow \mathbf{R}$ 为一个服务时间需求函数,指定每个服务请求对单位计算能力的平均服务时间需求.

③ $SR: S \rightarrow \mathbf{R}$ 为一个空间需求函数,指定每个服务运行时占用的内存空间.

④ $CW = \{w_1, w_2, \cdots, w_{WN}\}$, $WN \in \mathbf{N}$,为一个非空有限控制流集,其中 w_i 为一个控制流,描述了实现一个组合服务时服务之间的交互关系.

⑤ $ES: S \times S \rightarrow \boldsymbol{R}$ 为一个事件规模函数,指定两个服务间交互的事件的平均规模.

定义 2 运行平台模型 RPM 是一个四元组 RPM = (N, CP, AS, CC):

① $N = \{n_1, n_2, \cdots, n_{HN}\}$, $HN \in \mathbf{N}$,为一个非空有限硬件节点集,表示可供软件部署的所有硬件节点.

② $CP: N \rightarrow \mathbf{N}$ 为一个计算能力函数,指定每个硬件节点的计算能力.

③ $AS: N \rightarrow \mathbf{R}$ 为一个可用空间函数,指定每个硬件节点上可用的内存空间.

④ $CC: N \times N \rightarrow \boldsymbol{R}$ 为一个通信能力函数,指定任意两个硬件节点之间的通信能力.

定义 3 应用场景 AS 是一个五元组 $AS = (RA, IF, UC, PM, UF)$:

① $RA: S \rightarrow \mathbf{R}$ 为一个请求到达函数,指定服务集 S 中的每个服务接收的请求的平均到达率,即服务在单位时间内接收的平均请求数.

② $IF: S \times S \rightarrow \mathbf{R}$ 为一个交互频率函数,指定任意两个服务之间的交互频率. 如果 $s_i = s_j$ 或 s_i 与 s_j 之间不存在交互,则 $IF(s_i, s_j) = 0$.

③ $UC = (uc_1, uc_2, \cdots, uc_{UN})$, $UN \in \mathbf{N}$,是一个非空有限用户集,表示使用该软件的所有的用户.

④ $EM = \{m_1, m_2, \cdots, m_{MN}\}$, $MN \in \mathbf{N}$,是一个非空有限指标集,表示用于评估性能的所有指标.

⑤ $UF = \{UF_1, UF_2, \cdots, UF_{MN}\}$ 为一个非空有限效用函数集,UF_i 指定第 i 个性能指标对应的效用函数.

定义 4 约束 CON 是一个二元组 CON = (LC, CLC):

① $LC: S \times N \rightarrow \{0, 1\}$ 为位置约束函数,用于约束服务到硬件节点的分配. 如果服务 s_i 可以部署在硬件节点 n_j 上,则 $LC(s_i, n_j) = 1$,否则 $LC(s_i, n_j) = 0$.

② $CLC: S \times S \rightarrow \{-1, 0, 1\}$ 为同位约束函数,用于约束两个服务之间的位置关系. 如果服务 s_i 和 s_j 必须被部署在同一个硬件节点上,则 $CLC(s_i, s_j) = 1$;如果服务 s_i 和 s_j 不能被部署在同一个硬件节点上,则 $CLC(s_i, s_j) = -1$,如果 s_i 和 s_j 之间不存在这一约束,则 $CLC(s_i, s_j) = 0$.

本文主要考虑三种重要的性能评价指标:组合服务的平均延迟时间(用 l 表示)、组合服务的平均吞吐量(用 t 表示)以及硬件资源的平均利用率(用 u 表示). 因此,本文的目标函数可以定义为:

$$\max f(d) = \sum_{uc_i \in UC} \Big(\sum_{n_j \in N} UF_u(uc_i, u_j) + \sum_{w_k \in CW} \big(UF_l(uc_i, l_k) + UF_t(uc_i, t_k) \big) \Big) \tag{1}$$

其中,UF_u、UF_l 和 UF_t 分别表示硬件节点的利用率、组合服务的平均延迟时间、组合服务的平均吞吐量所对应的效用函数;l_k 和 t_k 分别表示在部署方案 d 下第 k 类组合服务的平均延迟时间和平均吞吐量. 由于组合服务由控制流定义,因此可以使用控制流 w_i 表示第 i 个组合服务;u_j 表示在部署方案 d 下第 j 个硬件节点的利用率.

3　基于蚁群优化的 ACO-DO 算法

在软件部署优化的问题中,蚂蚁构建的每个解对应一种部署方案,解组件对应服务到硬件节点的一种分配,蚂蚁一步步将每个服务分配到一个硬件节点来构建部署方案,当所有服务都被分配到一个相应的硬件节点时,一个部署方案就构建完成.

为提高蚁群算法在部署优化问题中的求解效率,本文设计了基于问题的启发式,考虑了部署方案的构建顺序,同时还设计了局部搜索过程.

3.1　信息素与启发式的设计

本文分别使用 $\tau(i,j)$ 和 $\eta(i,j)$ 表示将服务 s_i 分配到硬件节点 n_j(可以用公式 $d(s_i)=n_j$ 表示)所对应的信息素和启发式. 在部署软件时,为减少通讯延迟,通常希望将相互交互的服务部署在同一硬件节点上,但这会引起服务间的资源竞争,从而产生计算延迟. 为均衡这两种开销,本文定义启发式为:

$$\eta(i,j)=\begin{cases}\dfrac{CP(n_j)\sum_{s_k\in S_j}IF(s_k,s_i)}{\sum_{s_k\in S_j}RA(s_k)TR(s_k)}, & S_j\supseteq\{s_i\}\\ \dfrac{CP(n_j)}{DN}, & S_j=\{s_i\}\end{cases} \tag{2}$$

其中 S_j 为 s_i 以及当前部署在硬件节点 n_j 上的所有服务的集合,$IF(s_k,s_i)$ 为服务 s_k 与 s_i 之间的交互频率,DN 为已经部署服务的硬件节点数.

上述启发式表明,先分配的服务将影响后续服务的分配. 为保证部署方案的质量,本文使用服务列表表示方案构建过程中服务的分配顺序,并对服务列表进行优化. 服务列表的优化使用绝对位置模型表示信息素模型,即信息素 $\tau_s(i,k)$ 表示服务 s_i 位于服务列表中第 k 个位置的期望值.

在软件运行过程中,具有较大请求量的服务的性能指标的变化通常会对软件的整体性能产生较大影响,如果能够先满足这些服务的性能需求,则软件的整体性能将得到较大的提升. 因此,本文为服务列表优化问题设计的启发式为:

$$\eta_s(i)=RA(s_i) \tag{3}$$

3.2　方案的构建

根据 3.1 节的分析可知,方案的构建主要分两步:服务列表的构建和部署方案的构建. 在构建服务列表时,本文结合伪随机比例规则[14]和 Merkle 等[15]提出的求和规则进行解组件的选择,解决简单使用 $\tau_s(i,k)$ 进行解组件选择时算法容易收敛于局部最优解的问题[16];在构建部署方案时,约束 CON 将被用来阻止蚂蚁将服务分配到不满足约束的硬件节点上. 方案的构建步骤如下:

(1)将所有服务都放置于列表 S_{un} 中,即 $S_{un}=\{s_1,s_2,\cdots,s_{SN}\}$,且设 sList = { },$k=1$.

(2)从列表 S_{un} 中选择一个服务 s_i 放置到服务列表 sList 的第 k 个位置. 选择规则如下:首先,产生[0,1]之间的一个随机数 q,并将其与用户定义的参数 q_s 进行比较,如果 $q<q_s$,则为服务列表中第 k 个位置选择的服务 s_i 为:

$$i=\arg\max_{s_i\in S_{un}}\{\sum_{x=1}^{k}\tau_s(t,x)\cdot\eta_s(t)\} \tag{4}$$

否则,采用轮盘选择规则完成服务列表第 k 个位置的服务选择,即选择在服务列表的第 k 个位置放置服务 s_i 的概率 p_{ik} 被定义为:

$$p_{ik}=\begin{cases}\dfrac{\sum_{x=1}^{k}\tau_s(i,x)\cdot\eta_s(i)}{\sum_{s_j\in S_{un}}\sum_{x=1}^{k}\tau_s(j,x)\cdot\eta_s(j)}, & \text{如果 } s_j\in S_{un}\\ 0, & \text{否则}\end{cases} \tag{5}$$

(3)将服务 s_i 从列表 S_{un} 中移除,且 $k=k+1$. 如果 $k<SN$,则跳转到步(2).

(4)选择服务列表 sList 中第一个服务 s_p,并利用伪随机比例规则为服务 s_p 选择一个部署节点. 选择规则如下:首先产生[0,1]之间的一个随机数 q,并将其与用户定义的参数 q_d 进行比较,如果 $q<q_d$,则服务 s_p 分配的硬件节点 n_j 为:

$$j=\arg\max_{n_y\in FH}\{\tau(i,y)^{\alpha}\cdot\eta(i,y)^{\beta}\} \tag{6}$$

否则,采用轮盘选择规则完成服务到硬件节点的分配. 在轮盘选择规则中,将服务 s_p 分配给硬件节点 n_j 的概率 p_{pj} 为:

$$p_{pj}=\begin{cases}\dfrac{\tau(p,j)^{\alpha}\cdot\eta(p,j)^{\beta}}{\sum_{n_k\in FH}\tau(p,k)^{\alpha}\cdot\eta(p,k)^{\beta}}, & \text{如果 } n_j\in FH\\ 0, & \text{否则}\end{cases} \tag{7}$$

其中,α、β 为参数.

(5)如果将服务 s_p 分配给硬件节点 n_j 能够满足约束 CON,则将服务 s_p 从列表 sList 中移除. 如果 sList $\neq\varnothing$,则跳转到步(4).

(6)输出构建的部署方案 d.

3.3　局部搜索过程

局部搜索过程应用于蚂蚁的一次迭代完成之后,并基于 m 个当前最优的部署方案和蚂蚁本次迭代所构建的部署方案实现,在蚂蚁还没有找到 m 个最优部署方案之前,局部搜索过程将不会执行. 局部搜索过程的基本步骤如下:

(1)设当前 m 个最优部署方案和本次迭代所构建的部署方案组成的方案列表为 dList,在该列表中,第 i 个元素的目标函数值为 OBV_i,并设 $k=1$.

(2)根据部署方案的目标函数值,采用轮盘选择规则从列表 dList 中选择一个部署方案 d_i. 其中选择列表 dList 中的部署方案 d_i 的概率为:

$$gp_i = \frac{OBV_i}{\sum_{d_j \in \text{dList}} OBV_j} \tag{8}$$

(3)选取部署方案 d_i 的第 k 个解组件,并判断如果将其作为新部署方案的解组件时是否能够满足约束 CON,如果满足,则将该解组件添加到新部署方案中,否则跳转到步(2).

(4)$k=k+1$,如果 $k<SN$,则跳转到步(2),否则计算构建的部署方案的目标函数值,并将其与当前最优的部署方案进行比较,保留最优的部署方案和目标函数值.

4 实验设计与结果分析

4.1 实验数据与环境

本文在 8 种不同规模的模拟实例上,将所提出的 ACO-DO 与混合整数线性规划(Mixed Integer Linear Programming, MILP)、蚁群系统(Ant Colony System, ACS)、遗传算法(Genetic Algorithm, GA)进行对比,以评估其有效性. 实例规模如表 1 所示.

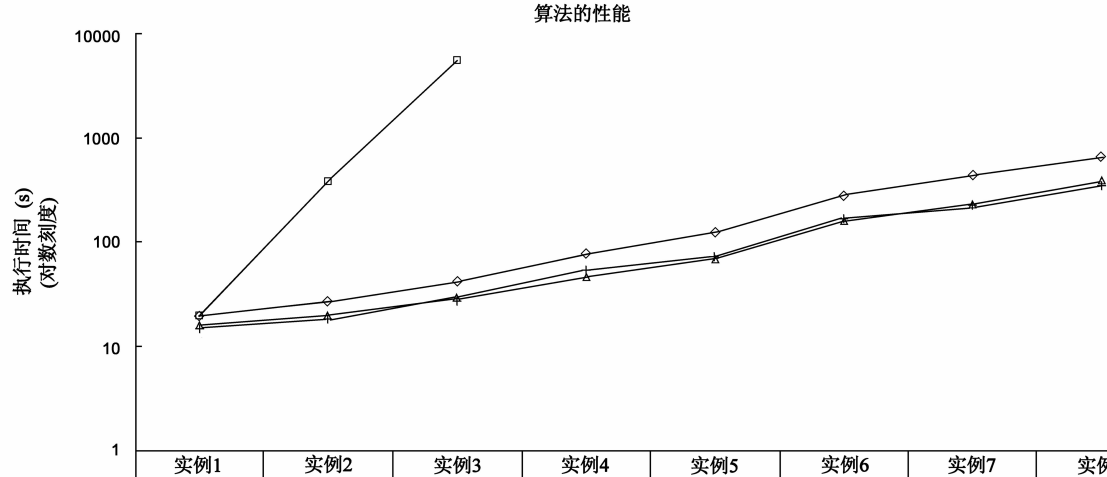
本文中各算法所使用的参数是通过为每个参数设置变化范围和变化步长,并进行反复试验得出的. 得出的最优参数组合为:蚁群算法(即 ACO-DO 和 ACS)的种群数 $AN=10$,参数 $\alpha=1$,局部搜索过程中的参数 $m=10$,信息素更新率 $\rho=0.1$,参数 $\beta=2$,伪随机比例参数 $q=q_s=q_d=0.5$;遗传算法种群数 $P=100$,杂交率 $p_c=0.9$,变异率 $p_m=0.1$.

表 1 测试实例的规模

Instance	*WN*	*SN*	*HN*	*UN*
1	3	8	4	4
2	4	12	5	5
3	5	14	6	6
4	6	16	7	7
5	7	25	8	8
6	9	36	10	10
7	10	40	12	10
8	11	50	13	11

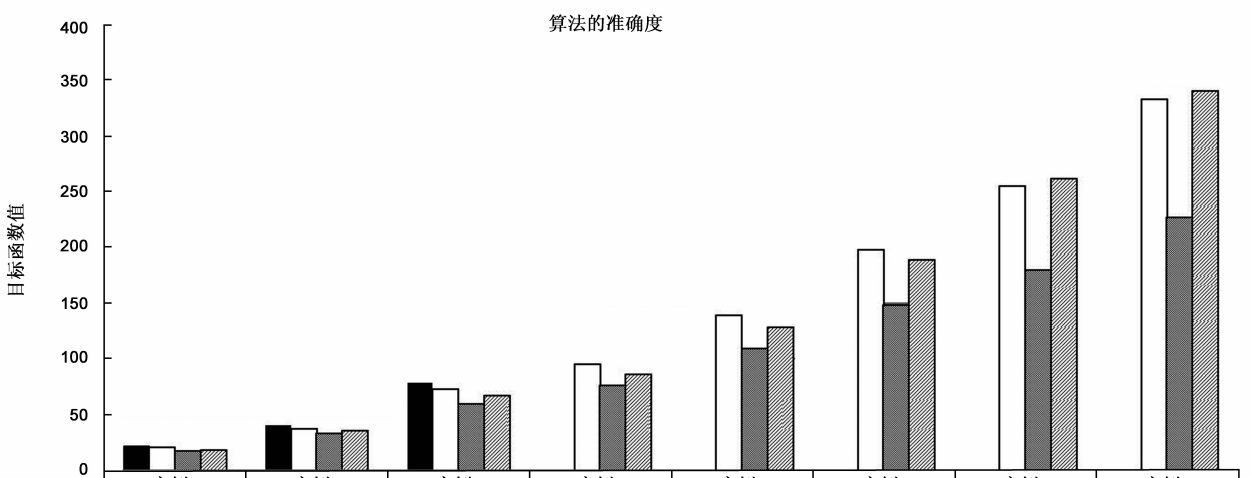
4.2 ACO-DO 算法的有效性分析

本文主要从两个方面评价 ACO-DO 算法的有效性:(1)性能,即算法的执行时间;(2)准确度,即目标函数值. 三种近似算法(ACO-DO、ACS 和 GA)的终止条件是,如果在连续的 5000 次评估(即 ACO-DO 算法的连续 250 次迭代或 ACS 算法的连续 500 次迭代或遗传算法的连续 50 次进化)过程中没有找到更好的解,则算法终止. 图 1 和图 2 分别显示了这四种算法在本文生成的模拟实例上运行所花费的执行时间和得到的目标函数值的平均值. MILP 算法的复杂性使得它不能求解较大规模的问题,因此,在图 1 和图 2 中,MILP 只给出了它能够求解的问题的结果.



	实例1	实例2	实例3	实例4	实例5	实例6	实例7	实例8
MILP	20	386	5574					
ACO-DO	16	20	29	46	70	162	232	386
ACS	15	18	30	54	74	171	217	344
GA	19	27	42	77	127	284	439	663

图1 MILP、ACO-DO、ACS和GA的执行时间



	实例1	实例2	实例3	实例4	实例5	实例6	实例7	实例8
■MILP	21.5	40.4	78.7					
□ACO-DO	20.6	37.7	73.3	95.6	139.3	197.7	255.4	332.6
▨ACS	17.5	33.2	59.8	76.7	109.4	148.7	179.6	227.2
▨GA	18.7	36.1	67.3	86.8	128.6	188.9	261.5	340.1

图2　MILP、ACO-DO、ACS和GA得到的目标函数值

从图1和图2中可以看出,ACS算法与ACO-DO算法都具有较短的执行时间,但ACS算法的准确度明显低于ACO-DO算法,且在规模较大的问题中尤为明显;GA算法与ACO-DO算法都具有较高的准确度,但GA算法的执行时间较长,且问题规模越大,它们之间的时间差距越大.因此,ACO-DO相比其它算法,能够花费较短的时间找到较好的解.

4.3　启发式的影响

为了评估本文引入的启发式对算法性能的影响,本文设置ACO-DO算法中的启发式参数$\beta=0$,将得到的算法记为ACO-DO/H(ACO-DO without Heuristic),将基于ACS算法且使用本文定义的启发式的算法记为ACS-H(ACS with Heuristic),然后分别对比ACO-DO和ACO-DO/H、ACS-H和ACS.

本实验在前面生成的各实例上分别对ACO-DO、ACO-DO/H、ACS-H和ACS四种算法独立执行30次,每次执行30000次评估,记录每次实验迭代过程中得到的目标函数值,画出这四种算法的演化曲线.图3(a)和图3(b)分别显示了这两组算法在实例6上运行的演化曲线图(演化曲线上的值为30次独立实验得到的平均值).

从图3中可以看出,当执行相同次数的评估时,ACO-DO总是能够比ACO-DO/H找到拥有更优目标函数值的部署方案,且ACS-H总是能够比ACS找到拥有更优目标函数值的部署方案,这说明本文定义的启发式能够有效地帮助算法找到更好的解.此外,从图3中还可以看出,ACO-DO和ACS-H的收敛速度分别快于ACO-DO/H和ACS,且ACO-DO和ACS-H最终找到的解分别优于ACO-DO/H和ACS,这说明本文的启发式能够指导蚂蚁快速找到较好的部署方案.

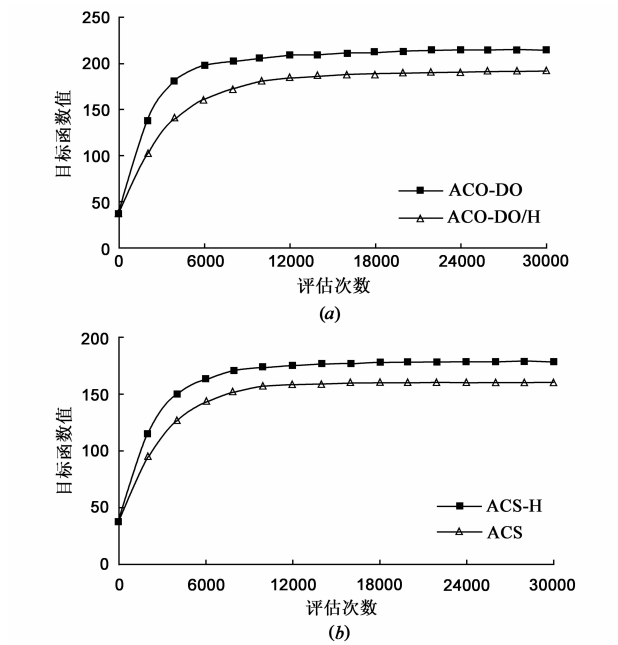


图3　算法ACO-DO、ACO-DO/H、ACS-H和ACS在实例6上运行的演化曲线图

4.4　方案构建顺序的影响

为了评估ACO-DO算法中引入的方案构建顺序对算法性能的影响,本文将基于ACO-DO算法、但不考虑方案构建顺序(即仅根据伪随机比例规则和信息素、启发式的值进行部署方案的构建)的算法记为ACO-DO/BO(ACO-DO without Build Order),将基于ACO-DO算

法且使用本文定义的构建顺序的算法记为 ACS-BO(ACS with Build Order),并在前面生成的各实例上分别对 ACO-DO、ACO-DO/BO、ACS-BO 和 ACS 四种算法各独立执行 30 次,每次执行 30000 次评估,记录每次实验迭代过程中得到的目标函数值,画出这四种算法的演化曲线. 图 4(*a*)和图 4(*b*)分别显示了这两组算法在实例 7 上运行的演化曲线图(演化曲线上的值为 30 次实验的平均值).

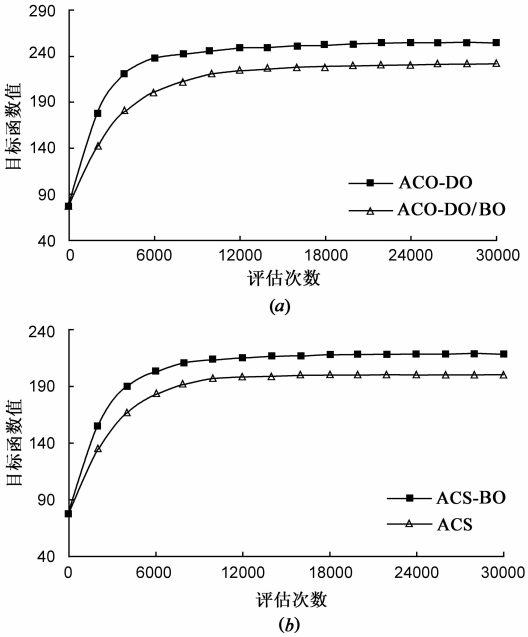


图4 算法ACO-DO、ACO-DO/BO、ACS-BO和ACS在实例7上运行的演化曲线图

从图 4 可以看出,算法开始执行的较短时间内,两组算法中的两类算法得到的目标函数值都相差不大;这是由于在算法执行的初始阶段,构建顺序还处于寻优阶段,优势不能完全得以体现. 而随着算法的演化,两组算法中的两类算法得到的目标函数值的差值越来越大,直到算法最终收敛;这是由于随着算法的演化,构建顺序越来越优,在后续的寻找最优部署方案的过程中发挥的作用越来越大.

对比图 4(*a*)和图 4(*b*)可以看出,图 4(*b*)中构建顺序的优势明显低于图 4(*a*),这是因为图 4(*b*)中对比的两种算法没有使用本文定义的启发式,构建顺序对最终构建的部署方案的影响仅依赖于约束,这使得构建顺序的优势无法得以完全发挥. 上述对比可以说明,本文提出的方案构建顺序能在一定程度上提高软件部署方案的构建质量.

4.5 局部搜索过程的影响

为了评估 ACO-DO 算法中引入的局部搜索过程对算法性能的影响,本文将不带本文定义的局部搜索方法的 ACO-DO 算法记为 ACO-DO/LS(ACO-DO without Local Search),将基于 ACS 算法且使用本文定义的局部搜索方法的算法记为 ACS-LS(ACS with Local Search),并在前面生成的各实例上分别对 ACO-DO、ACO-DO/LS、ACS-LS 和 ACS 四种算法各独立执行 30 次,每次执行 40000 次评估,记录每次实验迭代过程中得到的目标函数值,画出这四种算法的演化曲线(如图 5 所示).

从图 5 中可以看出,使用局部搜索过程的算法(ACO-DO 和 ACS-LS)比没有使用局部搜索过程的算法(ACO-DO/LS 和 ACS)能够更快地找到较好的解,且能够最终得到较优的目标函数值,这说明局部搜索方法能够进一步提高 ACO-DO 算法的性能.

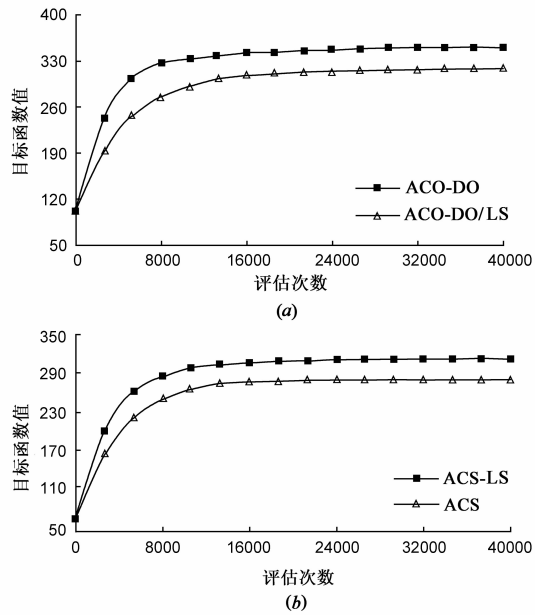


图5 算法ACO-DO、ACO-DO/LS、ACS-LS和ACS在实例8上运行的演化曲线图

5 结论

本文研究了蚁群算法在软件部署优化领域的应用,构建了面向服务软件的部署优化模型,并基于经典 ACS 算法,提出了基于蚁群优化的 ACO-DO 算法. 该算法结合软件工程领域的具体知识,设计了基于问题的启发式和部署方案的构建顺序,指导蚂蚁快速朝着最优解的方向前进. 此外,本文还将 ACO-DO 算法与局部搜索过程相结合,避免算法过早收敛于局部最优解. 通过在不同规模的部署优化实例上进行的一系列试验,以及与 ACS 和 GA 算法的性能比较,验证了 ACO-DO 算法在求解软件部署优化问题中具有较好的求解性能.

参考文献

[1] Vittorio C, Antinisca D M, Paola I. Model-Based Software

Performance Analysis[M]. Berlin Heidelberg: Springer, 2011.

[2] 胡剑军,官荷卿,魏峻,等. 一种基于性能模型的中间件自配置框架[J]. 软件学报,2007,18(9):2117 – 2129.
Hu Jian-jun, Guan He-qing, Wei Jun, et al. A performance model-based self-configuration framework for middleware[J]. Journal of Software, 2007, 18(9): 2117 – 2129. (in Chinese)

[3] Wada H, Suzuki J, Yamano Y, et al. Evolutionary deployment optimization for service-oriented clouds[J]. Software Practice and Experience, 2011, 41(5): 469 – 493.

[4] Frey S, Fittkau F, Hasselbring W. Search-based genetic optimization for deployment and reconfiguration of software in the cloud[A]. Proceedings of the 2013 International Conference on Software Engineering[C]. San Francisco: IEEE, 2013. 512 – 521.

[5] Jayasinghe D, Pu C, Eilam T. Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement[A]. Proceedings of the 2011 IEEE International Conference on Services Computing[C]. Washington DC: IEEE, 2011. 72 – 79.

[6] Malek S, Medvidovic N, Mikic-Rakic M. An extensible framework for improving a distributed software system's deployment architecture[J]. IEEE Transactions on Software Engineering, 2012, 38(1): 73 – 100.

[7] White J, Dougherty B, Thompson C, et al. ScatterD: Spatial deployment optimization with hybrid heuristic/evolutionary algorithms[J]. ACM Transactions on Autonomous and Adaptive Systems, 2011, 6(3): 123 – 154.

[8] 南国芳,陈忠楠. 基于进化优化的移动感知节点部署算法[J]. 电子学报,2012,40(5):1017 – 1022
NAN Guo-fang, CHEN Zhong-nan. Deployment algorithm of mobile sensing nodes based on evolutionary optimization[J]. Acta Electronica Sinica, 2012, 40(5): 1017 – 1022. (in Chinese)

[9] 张晓薇,曹东刚,陈向群,等. 一种网络化移动应用部署方案优化方法[J]. 软件学报, 2011, 22(12): 2866 – 2878.
Zhang X W, Cao D G, Chen X Q, et al. Deployment solution optimization for mobile network application[J]. Journal of Software, 2011, 22(12): 2866 – 2878. (in Chinese)

[10] Aleti A, Grunske L, Meedeniya I, et al. Let the ants deploy your software-an ACO based deployment optimization strategy[A]. Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering[C]. Auckland: IEEE, 2009. 505 – 509.

[11] Dorigo M, Stützle T. Ant colony optimization: Overview and recent advances[A]. International Series in Operations Research & Management Science (Handbook of Metaheuristics, Volume 146)[M]. US: Springer, 2010. 227 – 263.

[12] 柳长安,鄢小虎,刘春阳,等. 基于改进蚁群算法的移动机器人动态路径规划方法[J]. 电子学报,2011,39(5):1220 – 1224.
LIU Chang-an, YAN Xiao-hu, LIU Chun-yang, et al. Dynamic path planning for mobile robot based on improved ant colony optimization algorithm[J]. Acta Electronica Sinica, 2011, 39(5): 1220 – 1224. (in Chinese)

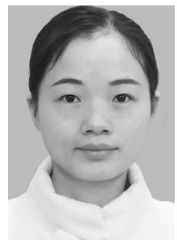
[13] Chen W N, Zhang J. Ant colony optimization for software project scheduling and staffing with an event-based scheduler[J]. IEEE Transactions on Software Engineering, 2013, 39(1): 1 – 17.

[14] 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.

[15] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(4): 333 – 346.

[16] Dorigo M, Birattari M. Ant colony optimization[A]. Encyclopedia of Machine Learning[M]. US: Springer, 2010. 36-39.

作者简介



李　琳　女,1988 年出生于湖北随州,武汉大学博士生,主要研究方向为面向服务的软件开发、云计算、智能算法.
E-mail: linl2012@whu.edu.cn

应　时(通讯作者)　男,1965 年出生,武汉大学教授、博士生导师,主要研究方向为软件工程中的智能分析与优化、软件工程的形式化理论与方法、云计算与软件服务、大数据的处理与智能分析.