一种基于规则的软件体系结构层性能演化优化方法

倪友聪[1,2],叶 鹏[3],杜 欣[1],陈 明[4],肖如良[1]
(1. 福建师范大学软件学院,福建福州 350117;2. 伦敦大学学院计算机科学学院,英国伦敦 WC1E 6BT;
3. 武汉纺织大学数学与计算机学院,湖北武汉 430200;4. 湖南师范大学数学与计算机科学学院,湖南长沙,410081)

摘 要: 目前基于规则的软件体系结构(Software Architecture,简记为 SA)层性能优化方法大多未充分考虑优化过程中规则的使用次数和使用顺序的不确定性,导致了搜索空间受限而难以获取更优的性能改进方案.针对这一问题并以最小化系统响应时间为优化目标,文中首先定义一种基于规则的 SA 层性能优化模型 RPOM,以将 SA 层性能优化抽象为求解最优规则序列的数学问题;然后设计一种支持 SA 层性能改进规则序列执行的框架 RSEF;进一步提出一种采用约束检查、修复及统计学习机制的演化求解算法 EA4PO;最后以 Web 应用为案例与已有方法进行实验对比.结果表明:(1)本文方法较已有方法可以获取更短的系统响应时间;(2)EA4PO 所引入的统计学习机制可显著提高演化求解算法的收敛速度和解质量.

关键词: 性能评估;性能优化;软件体系结构;基于搜索的软件工程
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2016)11-2688-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2016.11.018

An Approach for Rule-Based Performance Evolutionary Optimization at Software Architecture Level

NI You-cong[1,2], YE Peng[3], DU Xin[1], CHEN Ming[4], XIAO Ru-liang[1]
(1. *Faculty of Software, Fujian Normal University, Fuzhou, Fujian* 350117, *China*;
2. *Department of Computer Science, University College London, London, WC1E 6BT, UK*;
3. *College of Mathematics and Computer, Wuhan Textile University, Wuhan, Hubei* 430200, *China*;
4. *College of Mathematics and Computer Science, Hunan Normal University, Changsha, Hunan* 410081, *China*)

Abstract: In the existing rule-based performance optimization approaches at software architecture (SA) level, it has not been fully concerned that the count and the order of each rule usage are uncertain in the optimization process. As a result, the search space for performance improvement is limited and the better solutions are hard to find. Aiming to this problem and taking the system response time minimum as the optimization objective, firstly, a model called RPOM is defined to abstract rule-based software performance optimization at SA level as the mathematical problem for solving the optimal rule sequence. Secondly, a framework named RSEF is designed to support the execution of a rule sequence. Furthermore, an evolutionary algorithm named EA4PO is proposed to find the optimal performance improvement solution by introducing statistical learning, constraint checking and repairing. Finally, a web application is taken as a case in the experiments for comparing with the existing methods. The experimental results indicate that the shorter system response time can be obtained and the statistical learning can obviously improve the convergence rate and the solution quality in our approach.

Key words: performance evaluation; performance optimization; software architecture; search-based software engineering

收稿日期:2016-02-02;修回日期:2016-04-27;责任编辑:马兰英
基金项目:国家自然科学基金(No. 61305079, No. 61370078, No. 61402481);武汉大学软件工程国家重点实验室开放基金(No. SKLSE 2014-10-02);福建省自然科学基金(No. 2015J01235);福建省教育厅 JK 类项目(No. JK2015006);湖南省教育厅资助科研项目(No. 14C0680);河北省自然科学基金(No. F2015403046)

1 引言

软件的性能是衡量软件系统质量的一个重要属性,性能的优劣已经成为系统成败的关键因素. 在 SA 设计阶段进行性能的评估和预测,可以尽早地发现资源使用率过高、响应时间过长和吞吐量过小等性能问题,并通过相应的设计改进来缓解或消除这些问题,进而可在软件生命周期的早期达到性能优化的目的.

经过多年的研究和实践,人们已提出了排队网 QN、分层排队网 LQN、随机 Petri 网、随机进程代数和马尔可夫链等性能模型及其分析工具[1,2],在此基础上已涌现出了一些 SA 层性能评估和预测方法[3,4]. 这些方法能够将 SA 变换成某种性能模型,经过进一步的解析和分析后,可获取 SA 中相关软件元素和硬件元素的各项性能指标,进而可以评估和预测最终软件系统的性能. 在 SA 层性能评估和预测技术的基础上,如何对 SA 进行自动改进以获得满足性能需求的 SA 设计方案已成为一个受到高度关注的研究主题[5]. 然而随着软件系统的规模越来越大、复杂性越来越高,影响性能的因素也随之增多,使得基于 SA 的性能改进空间也在不断增大[6]. 此外,在性能改进过程中常常需要满足各种不同的设计约束和限制,又致使性能改进空间呈现高度的非连续形态. 在庞大且离散的性能改进空间中,如何自动找出 SA 层最优的性能改进方案,一直是软件工程工业界和学术界亟待解决的难题之一.

为了解决这一问题,目前已涌现出基于元启发[7,8]和基于规则[9~12]两类 SA 层性能自动优化方法. 基于元启发的方法将 SA 层性能优化抽象为参数优化问题,并通过元启发搜索技术[13~15]进行求解. 在预定义的优化参数上,元启发方法能在较大空间中搜索最优性能改进方案. 然而元启发方法往往只能对组件部署、硬件配置和组件选择等少数几种 SA 参数进行优化[8],且大多数元启发方法尚未充分考虑将 SA 层性能改进知识应用到优化过程中,以提高算法的收敛速度或增加优化结果的可解释性. 而基于规则的 SA 层性能优化方法则能以规则形式显式、精确地描述性能反模式[16]. 利用这些规则并借助相应的执行引擎可对 SA 的结构、行为和部署视图中的多种参数进行自动优化. 基于规则的方法能按照一定的策略在所有规则组合所确定的性能改进空间中,搜索最优改进方案. 然而由于规则的使用次数和使用顺序的不确定,往往导致规则的组合空间较为庞大. 例如:Cortellessa 和 Trubiani 等人针对 12 种 SA 层性能反模式,给出了对应的 12 条性能改进规则[9]. 即使在不考虑规则可以重复使用的情况下,由这 12 条规则不同的排列组合所确定的性能改进空间也将包含多于 12^{12}(近 9000 万亿)个候选的改进方案. 已有的基于规则的方法大多未充分考虑规则的使用次数和使用顺序的不确定性,从而导致搜索空间受限,难以获取更优的性能改进方案.

2 相关工作

2.1 SA 层性能改进规则及其执行引擎

SA 层性能改进规则由条件和动作两部分组成. 规则的条件部分用于诊断存在的性能问题,而动作部分则给出相应的改进方案. Cortellessa 和 Trubiani 等人[9,10]基于 SA 元模型和一阶谓词,并运用逻辑规则描述了一组性能反模式. Mcgregor 等人[12]通过综合考虑 SA 的可修改性,定义了一组性能改进规则. Xu 等人[11]基于 LQN 模型并运用 Jess 框架定义了一组不特定于 SA 建模语言的性能改进规则. 这些规则给出了资源瓶颈和响应时间长等性能问题的诊断条件以及对应的改进动作. 为了支持 SA 层性能改进规则的自动执行,相应的规则执行引擎[9,11,12]也已被研发.

2.2 基于规则的 SA 层性能优化算法

在 SA 层性能改进规则及其执行引擎的支持下,优化算法运用一定策略在规则组合所构成的改进空间中,搜索具有最短系统响应时间的 SA. 这些优化算法主要采用步进式[9,10,12]、按轮迭代的深度优先[11]两种搜索策略,如图 1 和图 2 所示. 图中弧上标记的 $r_1, r_2, \cdots, r_n$ 表示 n 条性能改进规则;圆形结点表示 SA;有向弧表示应用弧上标记的规则,将弧尾结点的 SA 改进为弧头结点的 SA;搜索过程从根结点对应的初始 SA 开始.

步进式搜索策略总是将 n 条规则 r_1 至 r_n 分别应用到当前搜索结点上,具有最短系统响应时间的 SA 被选作下一次搜索的当前结点. 如此反复,直至没有任何规则可以改进系统响应时间. 按轮迭代的深度优先搜索策略则将 n 条规则划分成配置改进和设计改进两类规则. 其中规则 $r_1, r_2, \cdots, r_m$ 为 m 条配置改进规则,而规则 $r_{m+1}, r_{m+2}, \cdots, r_n$ 为余下的 $n-m$ 条设计改进规则. 搜索首先进行初始的配置改进,然后再按轮进行迭代改进. 在每轮改进中,总是先进行设计改进,然后进行配置改进.

从以上两种策略的搜索过程可以看出两点:一是从根结点至最终结点的路径上使用的规则所构成序列可以视为求解出的性能改进方案;另一是这两种搜索策略都没有充分考虑规则的不同使用次数和不同使用顺序可能形成的各种组合情况,因而只能搜索相对较小的空间,更优的改进方案可能被排除.

3 基于规则的 SA 层性能优化模型 RPOM

RPOM 模型可描述优化过程中规则的使用次数、使用顺序与最优性能改进方案之间的数学关系. 下面给出它的具体定义.

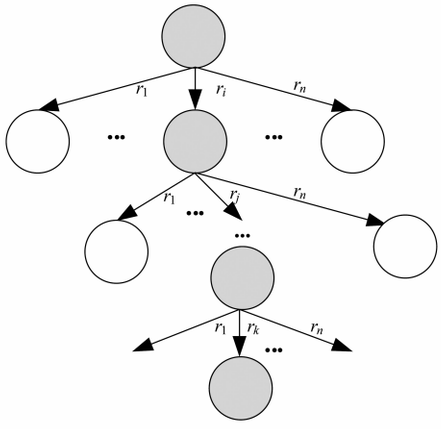

图1 步进式搜索过程

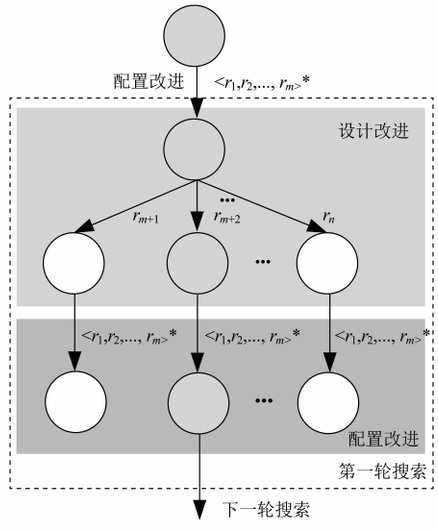

图2 按轮迭代的深度优先搜索过程

定义 1 定义函数 $r(SA)$ 为根据 SA 求解并返回系统的响应时间.

定义 2 从 1 至 n 依次对 n 条 SA 层性能改进规则进行编号,定义变换函数 $t(i,SA)$ 表示将 $i(1\leqslant i\leqslant n)$ 号规则应用到软件体系结构 SA 上,返回 i 号规则执行后的软件体系结构.

定义 3 定义函数 $imp(i,SA)$ 用于判定将 i 号规则应用到软件体系结构 SA 后,是否有性能改进效果. imp 函数的定义如式(1)所示.

$$imp(i,SA)=\begin{cases}0, & r(t(i,SA))\geqslant r(SA)\\ 1, & r(t(i,SA))<r(SA)\end{cases} \tag{1}$$

定义 4 定义规则号序列 $X=<x_1,x_2,\cdots,x_k,\cdots,x_l>$ 表示 SA 层性能改进方案. 并用 u_i 和 $h_i(X)$ 分别表示规则号 i 在 X 中最多可能出现的次数和实际出现的次数. X 的长度 l、X 中每个元素 x_k 的取值范围、$h_i(X)$ 的约束分别由式(2)、(3)和(4)定义. 其中:i,k,x_k,l 和 n 均为自然数.

$$l\leqslant\sum_{i=1}^{n}u_i \tag{2}$$

$$1\leqslant x_k\leqslant n,(1\leqslant k\leqslant l) \tag{3}$$

$$h_i(X)\leqslant u_i,(1\leqslant i\leqslant n) \tag{4}$$

定义 5 在 SA 层性能改进方案 X 中,x_k 号规则的执行前软件体系结构 BSA 由式(5)定义.

$$BSA(x_k)=\begin{cases}SA_0, & k=1\\ t(x_1,SA_0), & k=2\\ t(x_{k-1},BSA(x_{k-1})), & 3\leqslant k\leqslant l\end{cases} \tag{5}$$

定义 6 定义规则序列变换函数 $\vec{t}(X,SA_0)$ 表示依次应用 SA 层性能改进方案 X 所指示的一组规则到初始软件体系结构 SA_0 上,最后得到的结果软件体系结构.

$$\vec{t}(X,SA_0)=t(x_l,BSA(x_l)) \tag{6}$$

定义 7 基于规则的 SA 层性能优化问题可建模为:在满足式(2)、式(3)和式(4)的条件下,求解 $\bar{X}$ 使其满足式(7),获得的最优改进方案 X^* 由式(8)定义. 其中:Ω 表示 SA 层性能改进方案的空间,而 X^* 为去除 $\bar{X}$ 中没有改进效果的规则号而形成的序列.

$$\min_{\bar{X}\in\Omega}\{r(\vec{t}(\bar{X},SA_0))\} \tag{7}$$

$$X^*=\bar{X}/\{x_k\},(imp(x_k,BSA(x_k))=0\wedge 1\leqslant k\leqslant l) \tag{8}$$

4 SA 层性能改进规则序列的执行框架 RSEF

基于已有规则执行引擎,下面给出 RSEF 框架的总体结构和执行过程.

4.1 RSEF 框架的总体结构

图 3 给出了 RSEF 的总体结构. RSEF 以初始软件体系结构 SA_0、性能改进方案 X 为输入,通过控制引擎和规则执行引擎协同交互完成规则序列的执行,并输出序列中规则使用情况表 *T_RuleUseInSeq*、X 对应规则序列的执行前系统响应时间 *BRT* 和执行后系统响应时间 *ART*. *T_RuleUseInSeq* 表用于记录 RPOM 定义中 *imp* 函数的输出结果,其设计如表 1 所示. *loc* 和 *rulNum* 两个字段构成了 *T_RuleUseInSeq* 表的主码. *BRT* 和 *ART* 对应于 RPOM 模型中 $r(SA_0)$ 和 $r(\vec{t}(X,SA_0))$ 的值.

表 1 *T_RuleUseInSeq* 表的设计

序号	字段名称	含义
1	*loc*	规则号序列 X 的下标位置
2	*rulNum*	X 中 *loc* 位置的规则号 x_{loc}
3	*isImp*	根据 *loc* 位置上的编号为 *rulNum* 的规则执行前后的系统响应时间,判定该规则的应用是否有性能改进效果. 有改进效果为 1,否则为 0.

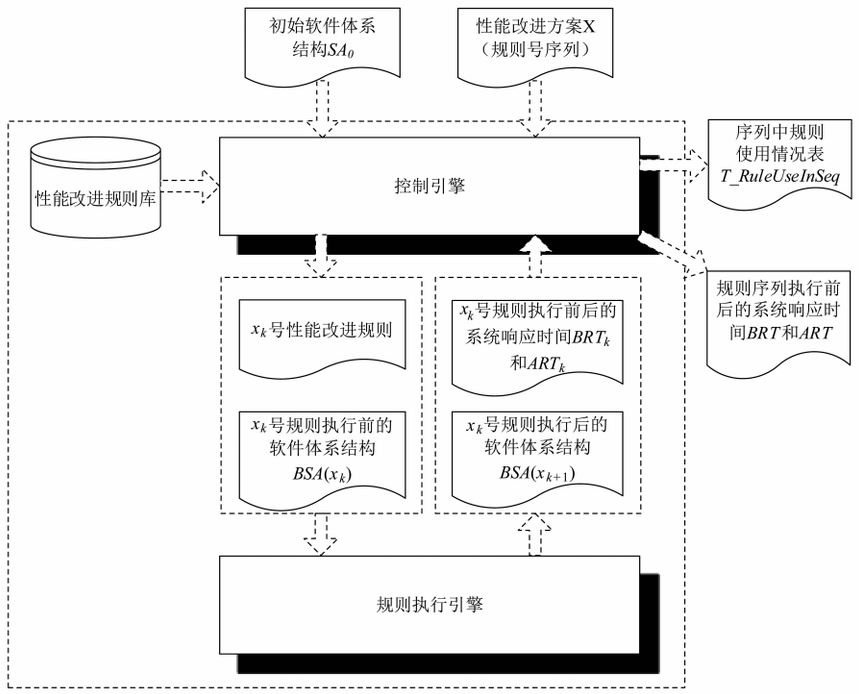

图3　SA层性能改进规则序列的执行框架RSEF的总体结构

4.2　RSEF 框架的执行过程

RSEF 框架的执行过程可由表 2 定义的 ARSE 算法进行描述.

表 2　ARSE 算法

	输入:X,SA_0 输出:$T\_RuleUseInSeq$ 表、BRT 和 ART
1	执行前软件体系结构 $TSA \leftarrow SA_0$,$k \leftarrow 1$,$BRT \leftarrow -1$,$ART \leftarrow -1$,$T\_RuleUseInSeq$ 置空;
2:	while $k \leqslant \lvert X \rvert$ do
3:	从规则库取出 x_k 号规则,并与 TSA 一并交由规则执行引擎执行;
4:	根据规则执行引擎返回的 BRT_k、ART_k 和 $BSA(k+1)$,令 $\Delta_{RT} = BRT_k - ART_k$,$TSA = BSA(k+1)$;
5:	if $BRT = -1$ then
6:	$BRT \leftarrow BRT_k$;
7:	end if;
8:	$ART \leftarrow ART_k$;
9:	if $\Delta_{RT} > 0$ then
10:	将记录$(k, x_k, 1)$插入 $T\_RuleUseInSeq$ 表;
11:	else
12:	将记录$(k, x_k, 0)$插入 $T\_RuleUseInSeq$ 表;
13:	end if;
14:	$k \leftarrow k+1$;
15:	end while;
16:	输出 $T\_RuleUseInSeq$ 表、BRT 和 ART.

5　基于规则的 SA 层性能演化优化算法 EA4PO

EA4PO 算法流程与传统遗传算法的类似,下面仅给出其个体编码、适应度计算、交叉和自适应变异算子.

5.1　个体编码

在满足 RPOM 中式(2)和式(4)定义的同时,尽可能减少规则的使用次数,我们引入了一条特殊的、编号为 0 的空规则. 约定当 0 号规则应用到任何软件体系结构将不做任何修改,即式(9)恒成立. 规则号 0 的最多可能出现次数由式(10)定义,它是所有非 0 规则号的最多可能出现次数之和.

$$t(0, SA) = SA \tag{9}$$

$$u_0 = \sum_{k=1}^{n} u_k \tag{10}$$

EA4PO 中个体被编码为固定长度的规则号序列 $X' = <x_1', x_2', \cdots, x_k', \cdots, x_l'>$. 其中:$X'$的长度 l'、X'中每个元素 x_k' 的取值范围、规则号 i 在 X'中实际出现次数 $h_i(X')$的约束分别由式(11)、式(12)和式(13)定义.

$$l' = u_0 \tag{11}$$

$$0 \leqslant x_k' \leqslant n, (1 \leqslant k \leqslant l) \tag{12}$$

$$h_i(X') \leqslant u_i, (0 \leqslant i \leqslant n) \tag{13}$$

5.2　适应度计算

EA4PO 中个体适应度值可用式(14)定义的 *fitness* 函数计算. 适应度值越小,表明个体越优.

$fitness(X') = r(\vec{t}(X, SA_0))$，其中：$X = X' / \{0\}$ (14)

为了收集各个规则使用的启发式信息，在函数 *fitness* 的计算过程中建立了规则使用历史表 *T_RuleUseInHis*，其设计如表 3 所示. 其中，*loc*、*rulNum* 和 *preRulNum* 三个字段构成 *T_RuleUseInHis* 表的主码.

表 3 规则使用历史表 *T_RuleUseInHis*

字段名称	含义
loc	序列 *X* 的下标位置
rulNum	*X* 中 *loc* 位置的规则号
preRulNum	*X* 中 *loc* − 1 位置的规则号，若 *loc* = 1，取值为 − 1
impNum	在 *loc* 和 *loc* − 1 位置的规则号分别是 *rulNum* 和 *preRulNum* 的情景下，*rulNum* 号规则有改进效果的使用次数
totNum	在 *loc* 和 *loc* − 1 位置的规则号分别是 *rulNum* 和 *preRulNum* 的情景下，*rulNum* 号规则的使用总数

5.3 带约束检查和修复机制的交叉算子

EA4PO 的交叉算子包括交叉、约束检查和修复三个计算步骤. 交叉步骤与一般的单点交叉操作相同. 约束检查步骤对产生的两个中间个体，从交叉位开始至最后一位依次检查每位上的规则号是否违反式(13)定义的最多出现次数的约束，并记录所有违反约束的位. 修复步骤将这些位上的规则号全部设置为 0.

5.4 带约束检查和修复机制的自适应变异算子

EA4PO 的变异算子采用带约束检查和修复机制的自适应单点变异算子. 其包括基于统计学习的条件变异、约束检查和修复三个计算步骤. 后两个步骤类似于 5.3 节中定义的约束检查和修复步骤. 基于统计学习的条件变异定义如下.

定义 8 函数 $nxtRulNum(i, m)$ 表示在 *T_RuleUseInHis* 表中按第 i 位置上为规则号 m 的条件进行查找，并返回第 $i+1$ 位置上所有出现的规则号构成的集合.

定义 9 函数 $avgImpRate(j, k, q)$ 表示在 *T_RuleUseInHis* 表中按第 j 和 $j-1$ 位置上规则号分别是 k 和 q（当 $j=1$ 时，$q=-1$）的条件进行查找，计算结果记录 *impNum* 和 *totNum* 两字段值的商，称为 k 号规则的平均改进率.

定义 10 $p(x_j' = k \mid x_{j-1}' = q)$ 表示个体 X' 在第 $j-1$ 位置上规则号为 q（当 $j=1$ 时，$q=-1$）的条件下，将第 j 位上的规则号变异为 k 的概率，由式(15)、式(16)和式(17)所定义.

$$S = nxtRulNum(j-1, q) / x_j' \tag{15}$$

$$W = \{ i \mid i \in N \wedge 1 \leqslant i \leqslant n \wedge i \neq x_j' \} \tag{16}$$

$$p(x_j' = k \mid x_{j-1}' = q) = \begin{cases} 0, & k = x_j' \\ \dfrac{avgImpRate(j,k,q)}{\sum_{i \in s} avgImpRate(j,i,q)}, & S \neq \Phi \\ u_k / \sum_{i \in W} u_i, & S = \Phi \end{cases} \tag{17}$$

6 案例研究

在基于规则的 SA 层性能优化方法中，仅 Xu 方法[11]所定义的规则是面向 LQN 模型[17]而不特定于具体的 SA 建模语言. 考虑到 Xu 方法具有较好的通用性，故将其作为本文的对比方法. 为了实证我们方法的有效性，以文献[11]中的 Web 应用（WebApp）为案例将我们的 EA4PO 与 Xu 按轮迭代的深度优先搜索算法（简称 DFS）进行实验对比，并进一步将不带统计学习机制的 EA4PO$^-$ 与 EA4PO 进行比较，以实证采用统计学习机制的自适应变异算子可以提高演化求解算法的收敛速度和解质量.

6.1 WebApp 案例

WebApp 的初始 LQN 模型如图 4 所示，其对应的初始系统响应时间为 179.71ms. 文献[11]给出了 WebApp 案例所使用的 6 条性能改进规则 r_1 至 r_6 及其相关的参数值. Xu 的 DFS 算法获取的 WebApp 优化后的 LQN 模型如图 5 所示，其对应的系统响应时间 29.88ms，红色部分为优化后不同于优化前的模型参数值. DFS 获取的性能改进方案是 $< r_2, r_2, r_2, r_2, r_2, r_3, r_4, r_5 >$.

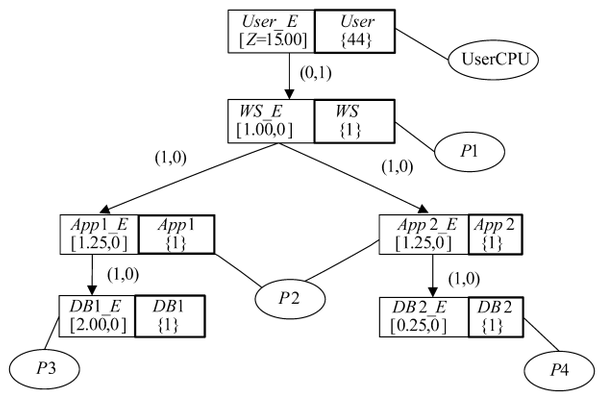

图4 初始LQN 模型

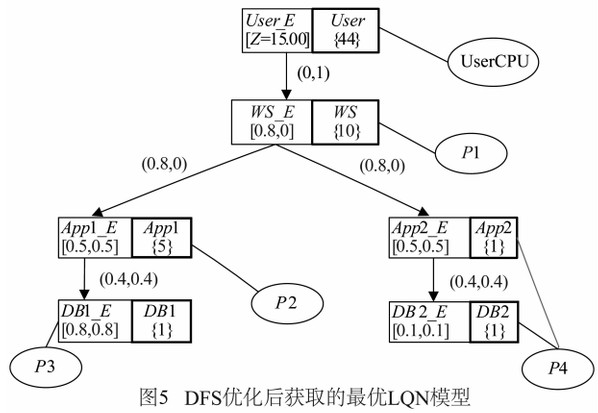

图5 DFS优化后获取的最优LQN模型

6.2 实验结果

(1) EA4PO 与 Xu 的算法比较 为了公平地比较，

设置规则 r_1 至 r_6 相关的参数值与 Xu 方法相同. EA4PO 中的种群大小、运行次数、演化代数、交叉概率和变异概率分别为 15,20,30,0.6 和 0.3. EA4PO 运行 20 次的结果如表 4 所示,其中第 3、4、9、12、15、16、18、20 次运行获取的结果响应时间是 22.6ms,它们对应的 LQN 模型都是相同的,如图 6 所示. 与图 5 中 LQN 模型的不同

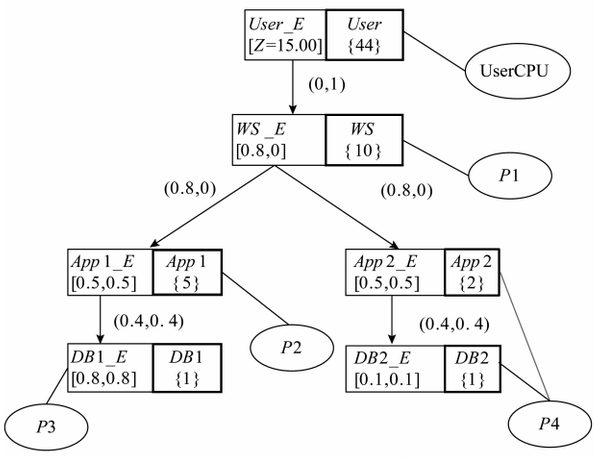

图6　EA4PO优化WebApp后获取的最优LQN模型

之处仅在于 App2 的多重性不同. 从表 4 可看出,除了第 10 次运行的结果外,EA4PO 获取的结果响应时间均优于 DFS. 我们使用 Wilcoxon 秩和检验对 EA4PO 和 DFS 获取的结果响应时间进行了统计检验. 在置信水平 $\alpha = 0.05$ 下,$p\text{-}values = 4.1696e-009 < 0.05$,统计检验结果表明 EA4PO 显著优于 DFS.

表 4　EA4PO 算法 20 次运行结果

序号	结果响应时间(ms)	改进方案	
		序列	长度
1	28.80	$<r_2,r_2,r_4,r_2,r_5,r_2,r_3,r_2>$	8
2	26.48	$<r_4,r_2,r_2,r_2,r_2,r_2,r_5,r_3,r_2>$	9
3	22.60	$<r_2,r_2,r_2,r_5,r_4,r_3,r_2,r_2,r_2>$	9
4	22.60	$<r_2,r_2,r_2,r_5,r_4,r_3,r_2,r_2,r_2>$	9
5	27.31	$<r_2,r_2,r_2,r_5,r_2,r_4,r_3>$	7
6	26.48	$<r_4,r_2,r_2,r_2,r_5,r_2,r_3,r_2,r_2>$	9
7	26.48	$<r_4,r_2,r_2,r_2,r_5,r_2,r_3,r_2,r_2>$	9
8	**26.48**	$<r_2,r_4,r_2,r_2,r_2,r_2,r_3,r_5,r_2>$	9
9	**22.60**	$<r_2,r_2,r_2,r_2,r_5,r_3,r_2,r_4,r_2>$	9
10	29.88	$<r_5,r_4,r_2,r_2,r_3,r_2,r_2>$	7
11	26.48	$<r_4,r_2,r_2,r_2,r_5,r_2,r_3,r_2,r_2>$	9
12	22.60	$<r_2,r_2,r_2,r_5,r_4,r_3,r_2,r_2,r_2>$	9
13	26.48	$<r_2,r_2,r_4,r_2,r_2,r_2,r_5,r_3,r_2>$	9
14	28.80	$<r_2,r_4,r_2,r_2,r_2,r_2,r_3,r_5>$	8
15	22.60	$<r_2,r_2,r_2,r_5,r_2,r_4,r_3,r_2,r_2>$	9
16	22.60	$<r_2,r_2,r_2,r_5,r_2,r_4,r_3,r_2,r_2>$	9
17	26.48	$<r_4,r_2,r_2,r_2,r_2,r_2,r_5,r_3,r_2>$	9
18	22.60	$<r_2,r_2,r_2,r_5,r_4,r_3,r_2,r_2,r_2>$	9
19	**25.28**	$<r_2,r_2,r_2,r_5,r_2,r_4,r_3,r_2>$	8
20	**22.60**	$<r_2,r_2,r_2,r_5,r_2,r_4,r_3,r_2,r_2>$	9

此外,表 4 中第 8 行和第 9 行对应的改进方案都使用了规则 r_2,r_3,r_4 和 r_5,且每个规则使用次数都相同,但这些规则的使用顺序不同导致了两次运行获得的结果响应时间明显不同. 又如表 4 中第 20 行较第 19 行对应的序列仅在最后 1 位多出规则 r_2,其余部分全部相同. 但就是仅多使用的这 1 次规则 r_2,却导致了获得的结果响应时间有着显著的不同. DFS 因未充分考虑优化过程中各规则使用顺序和使用次数的问题而导致其搜索空间受限,难以获取更优的改进方案.

(2) EA4PO⁻ 与 EA4PO 比较　EA4PO⁻ 与 EA4PO 设置相同的种群大小、交叉概率和变异概率,分别为 15,0.6 和 0.3. 两个算法都独立运行 20 次,每次运行以连续 20 代结果响应时间不发生变化作为停机条件. 采用文献[18]提出的收敛曲线图来表示所有 20 次运行中每隔两代最优解平均值的变化曲线.

从图 7 中可以看出 EA4PO 每次评估获取的平均结果系统响应时间均优于 EA4PO⁻,其收敛曲线的下降速度要快于 EA4PO⁻. 使用 Wilcoxon 秩和检验对 EA4PO 和 EA4PO⁻ 所获取的结果响应时间进行了统计检验. 在置信水平 $\alpha = 0.05$ 下,$p - values = 1.8803e-006 < 0.05$,统计检验结果表明 EA4PO 显著优于 EA4PO⁻. 实验结果表明采用统计学习机制的自适应变异算子不仅可以加快算法的收敛速度,而且能够提高解质量.

7　结束语

文中提出了一种基于规则的 SA 层性能演化优化方法. 首先定义了一种 SA 层性能优化模型 RPOM,通过该模型精确刻画规则的使用次数、使用顺序与最优性能改进方案之间的关系,并将 SA 层性能优化抽象为求解最优规则序列问题;然后设计了一种用于支持 SA 层性能改进规则序列的执行框架 RSEF,以支持 RPOM 的求解;进一步通过引入约束检查、修复及统计学习机制提出了一种高效的演化求解算法 EA4PO,以搜索 SA 层最优性能改进方案. 最后通过 WebApp 案例开展实验对比研究,实证了本文方法的有效性. 下一步工作考虑将改进代价作为一个优化目标,研究基于规则的 SA 层性能多目标演化优化方法.

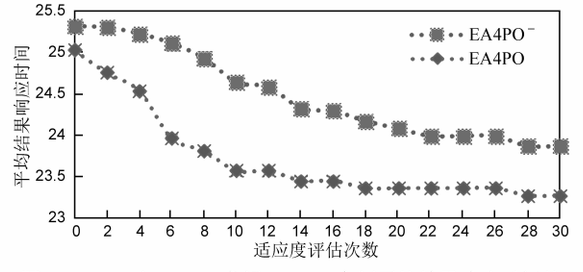

图7　EA4PO⁻与EA4PO求解WebApp案例平均结果响应时间的收敛曲线图

参考文献

[1] Koziolek H. Performance evaluation of component-based software systems: A survey[J]. Performance Evaluation, 2010,67(8):634 – 658.

[2] Brosig F, et al. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures[J]. IEEE Transactions on Software Engineering,2015,41(2):157 – 175.

[3] 李传煌,等. 一种 UML 软件架构性能预测方法及其自动化研究[J]. 软件学报,2013,24(7):1512 – 1528.
LI Chuan-Huang, et al. Performance prediction method for UML software architecture and its automation[J]. Journal of Software,2013,24(7):1512 – 1528. (in Chinese)

[4] Cortellessa V, et al. Model-Based Software Performance Analysis[M]. Berlin: Springer,2011.

[5] Aleti A, et al. Software architecture optimization methods: A systematic literature review[J]. Software Engineering, IEEE Transactions on,2013,39(5):658 – 683.

[6] Koziolek A. Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes [M]. Karlsrule: KIT Scientific Publishing,2014.

[7] Martens A, Koziolek H. Automatic, model-based software performance improvement for component-based software designs[J]. Electronic Notes in Theoretical Computer Science,2009,253(1):77 – 93.

[8] Etemaadi R, Chaudron M R. New degrees of freedom in metaheuristic optimization of component-based systems architecture: Architecture topology and load balancing[J]. Science of Computer Programming,2015,97:366 – 380.

[9] Cortellessa V, et al. An approach for modeling and detecting software performance antipatterns based on first-order logics [J]. Software & Systems Modeling,2014,13(1):391 – 432.

[10] Trubiani C, Koziolek A. Detection and solution of software performance antipatterns in palladio architectural models[J]. Acm Sigsoft Software Engineering Notes, 2011,36(5):19 – 30.

[11] Xu J. Rule-based automatic software performance diagnosis and improvement[J]. Performance Evaluation, 2012, 69(11):525 – 550.

[12] Mcgregor J D, et al. Using arche in the classroom: One experience[R]. Hanscom AFB, MA, USA: SEI, carnegie Mellon University,2007.

[13] Li Y, et al. An improved multiobjective estimation of distribution algorithm for environmental economic dispatch of hydrothermal power systems[J]. Applied Soft Computing,2015,28:559 – 568.

[14] Li Y, et al. Dynamic-context cooperative quantum-behaved particle swarm optimization based on multilevel thresholding applied to medical image segmentation[J]. Information Sciences,2015,294:408 – 422.

[15] Licheng J, et al. Quantum-inspired immune clonal algorithm for global optimization. [J]. IEEE Transactions on Systems Man & Cybernetics Part B Cybernetics A Publication of the IEEE Systems Man & Cybernetics Society, 2008,38(5):1234 – 1253.

[16] Smith C U, Williams L G. Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software [M]. Redwood City, CA, USA: Addison-Wesley Longman Publishing Co., Inc.,2002.

[17] Tribastone M. A fluid model for layered queueing networks[J]. IEEE Transactions on Software Engineering, 2013,39(6):744 – 756.

[18] Suganthan P N, Hansen N, et al. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization[R]. Singapore: Nanyang Technological University,2005.

作者简介

倪友聪 男,1976 年 8 月出生于安徽省肥西县. 现为福建师范大学软件学院副教授、硕士生导师. 主要研究领域为基于搜索的软件设计,软件体系结构等.
E-mail: youcongni@foxmail.com

叶 鹏 男,1976 年 12 月出生于湖北省武汉市. 现为武汉纺织大学数学与计算机学院讲师. 主要研究领域为基于搜索的软件设计,软件体系结构等.
E-mail: whuyp@126.com

杜 欣(通信作者) 女,1979 年 2 月出生于新疆河子市. 现为福建师范大学软件学院副教授、硕士生导师. 主要研究领域为基于搜索的软件工程,演化计算等.
E-mail: xindu79@126.com