基于随机搜索规则的软件体系结构层性能演化优化方法

倪友聪[1,2] 李 松[1] 叶 鹏[3] 杜 欣[1]
(福建师范大学数学与信息学院 福州 350117)[1] (伦敦大学学院计算机科学学院 伦敦 WC1E 6BT)[2]
(武汉纺织大学数学与计算机学院 武汉 430200)[3]

摘 要 已有的基于规则的软件体系结构(Software Architecture,SA)层性能优化方法大多未充分考虑优化过程中各规则的改进幅度、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。针对该问题,基于SA层性能改进策略定义一组随机搜索规则,以增大各规则的性能改进空间;在此基础上考虑这些规则的不同使用顺序和不同使用次数的组合情况,构建SA层性能优化模型RRPOM,并设计演化求解算法,进而形成一种SA层性能优化方法RRMO4PO。与已有方法在WebApp应用案例上的实验对比表明,该方法在使用更少的规则、更少次修改SA元素而获取更好可解释性的同时,有效减少了系统响应时间和改进代价。在最好的情况下,平均使用有改进效果的规则的次数和平均修改SA元素的次数较已有方法分别降低了33.3%和52.9%,与此同时将系统响应时间和改进代价分别降低了30.5%和73.6%。

关键词 软件体系结构,性能优化,系统响应时间,随机搜索规则
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.023

Random Search Rule Based Performance Evolutionary Optimization Method at Software Architecture Level

NI You-cong[1,2] LI Song[1] YE Peng[3] DU Xin[1]
(College of Mathematics and Informatics,Fujian Normal University,Fuzhou 350117,China)[1]
(Department of Computer Science,University College London,London WC1E 6BT,UK)[2]
(College of Mathematics and Computer,Wuhan Textile University,Wuhan 430200,China)[3]

Abstract The existing rule-based performance optimization approaches at software architecture (SA) level don't fully concern the improvement range of the rule and the uncertainty of the count and the order of each rule usage in the optimization process. As a result,the search space for performance improvement is limited and the better solutions are hard to find. Aiming at this problem,firstly,a group of the random search rules (RSRs) were designed based on performance improvement tactics so that each RSR can explore the larger performance improvement space. Then the combination of the order and the count of each rule usage was involved in the definition of the random search rule based performance optimization model named RRPOM and the evolutionary algorithm was designed to solve RRPOM. Further,the random search rule based method for performance optimization at software architecture level (RRMO4PO) was proposed. Finally,a WebApp application was taken as a case in the experiments for comparing RRMO4PO with the existing methods. The experimental results show that RRMO4PO can obtain the solutions with better interpretability by using fewer rules and fewer times to modify the SA elements. In addition,the results also prove that both system response time and cost for performance improvement can be decreased more efficiently in our approach. In the best results in our experiments, the average number of rules with improvement effect and the times to modify the SA elements are reduced by 33.3% and 52.9% respectively,and the system response time and the cost for performance improvement are decreased by 30.5% and 73.6% respectively.

Keywords Software architecture,Performance optimization,System response time,Random search rule

---

到稿日期:2016-10-27 返修日期:2016-12-12 本文受国家自然科学基金(61305079,61370078),福建省自然科学基金(2015J01235,2017501498),福建省教育厅JK类项目(JK2015006),武汉大学软件工程国家重点实验室开放基金(SKLSE 2014-10-02)资助。

倪友聪(1976-),男,博士,副教授,主要研究方向为基于搜索的软件设计、移动云计算,E-mail:youcongni@foxmail.com;李 松(1992-),男,硕士生,主要研究方向为基于搜索的软件设计,E-mail:lisong_fjnu@foxmail.com;叶 鹏(1976-),男,博士,讲师,主要研究方向为基于搜索的软件设计、软件可靠性;杜 欣(1979-),女,博士,副教授,主要研究方向为智能计算、基于搜索的软件工程。

1　引言

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。在软件体系结构(SA)设计阶段进行性能的评估[1]和诊断[2]可以尽早发现资源利用率过高、系统响应时间过长和吞吐量过低等性能问题，从而通过相应的改进设计[3]加以缓解或消除，进而达到在软件生命周期的早期进行性能优化的目的。近年来，已涌现出基于元启发和基于规则的 SA 层自动性能优化方法。

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进行求解。针对 UML[4-5]，PCM[6-8]和 East-AADL[9]等不同的 SA 建模语言，同时通过设计爬山[6]、GA[10]、粒子群[7]、NSGA-II[8]、SMS-EMOA[9,11]等不同的元启发算法，人们提出了一些基于元启发的 SA 层性能优化方法。元启发方法能在预定义的参数上搜索较大的空间，发现具有较好性能的 SA。但元启发方法的优化参数多集中于组件部署、硬件配置和组件选择等，较少涉及组件内部的结构和行为参数。此外，元启发方法对优化结果还缺乏合理解释的能力，使得 SA 的涉众难以深入地洞察具体的优化过程，进而影响最终 SA 方案的权衡和选择。基于规则的方法运用规则来描述 SA 层的性能反模式[12]和性能改进策略[13]等性能优化知识。每个规则的条件部分被用于诊断性能问题，而动作部分则按预定义幅度(或比例)对引发问题的 SA 元素进行改进。另外，基于规则的方法还设计了优化算法并按照一定的策略组合使用预定义的规则，以搜索最佳的性能改进方案。基于规则的方法可获取具有较好解释性的性能改进方案，这些方案能够展示初始 SA 按照何种顺序使用各条规则，并且这些规则以多大的幅度改进引发性能问题的元素，最后得到具有最佳性能指标(如系统响应时间)的结果 SA。

在基于规则的方法中，每个规则的改进幅度、使用次数和使用顺序的不同组合可构成较为庞大的 SA 层的性能改进方案空间。例如：PB 方法[14]设计了 7 条性能改进规则。即使每个规则最多只可使用 2 次，且各规则的改进幅度也只有 2 种可选值，在此情景下其对应的性能改进空间也将高达14!(近 872 亿)。然而，现有的基于规则的方法大多未充分考虑优化过程中各规则的改进幅度、使用次数和使用顺序的不确定性，导致搜索空间受限而难以获取更优的性能改进方案。针对上述问题，本文在我们前期工作[15-16]的基础上，以最小化系统响应时间和改进代价为目标，提出了一种基于随机搜索规则的 SA 层优化性能方法 RRMO4PO。本文的主要贡献如下：

(1)基于部署、行为和结构 3 种视图定义了一组随机搜索规则来描述 SA 层性能改进策略，为增大每个规则的性能改进空间提供了有效支持。

(2)构建了一种基于随机搜索规则的 SA 层性能优化模型 RRPOM。该模型可精确刻画各规则的使用次数、使用顺序和修改元素与系统响应时间和改进代价两个优化目标之间的关系。

(3)设计了带约束检查和修复机制的 RRPOM 模型演化求解算法，可高效地在较大空间中搜索 Pareto 最优性能改进方案。

(4)针对 WebApp 案例，将本文 RRMO4PO 方法与典型的基于规则的方法 PB 进行了实验对比。实验结果表明，较 PB 方法，本文 RRMO4PO 方法在使用更少的规则、更少次修改 SA 元素而获取更好可解释性的同时有效地降低了系统响应时间和改进代价。在最好的情况下，平均使用有改进效果的规则的次数和平均修改 SA 元素的次数较 PB 方法分别降低了 33.3%和 52.9%，与此同时，将系统响应时间和改进代价分别降低了 30.5%和 73.6%。

本文第 2 节阐述了相关工作；第 3 节设计了一组用于 SA 层性能改进的随机搜索规则；第 4 节定义了基于随机搜索规则的性能优化模型 RRPOM；第 5 节提出了 RRPOM 模型的演化求解算法；第 6 节给出了案例研究及实验结果；最后总结全文并展望未来。

2　相关工作

下面简要介绍 SA 层的性能评估、SA 层的性能改进知识和基于规则的 SA 层性能方法等相关工作。

2.1　SA 层的性能评估

为了能够在设计阶段基于 SA 进行软件性能评估，人们已提出了排队网 QN[17]、分层排队网 LQN[18]、随机 Petri 网[19]、随机进程代数[20]和马尔可夫链[21]等性能模型及其分析工具，并在此基础上提出了一些 SA 层的性能评估方法[1,22-23]。

不同 SA 层的性能评估方法可能采用不同的性能模型，但是它们的评估过程基本相同，都是先将待建系统的 SA 变换成某种性能模型的一个实例；然后再对该性能模型实例进行解析和分析，并输出相应的性能评估报告；最后从评估报告中获取待建系统的各项性能指标(如系统响应时间、吞吐量和资源使用率等)，本文仅讨论系统响应时间。

2.2　SA 层的性能改进知识

经过多年的研究和实践，人们已经积累了一些 SA 层的性能改进知识，并将它们进行了系统的总结及文档化[24-26]。文献[13]将性能反模式[12]和 SA 设计策略[13]进行了有机整合，较为全面地提出了独立于具体 SA 描述语言和特定应用的 17 种 SA 层的性能改进策略。

2.3　基于规则的 SA 层性能优化方法

基于规则的 SA 层性能优化方法将性能改进知识运用机器可处理的规则进行定义和描述。基于 SA 元模型和一阶逻辑谓词，同时在运用逻辑规则的基础上，Cortellessa 和 Trubiani等人[27-30]描述了 12 种性能反模式。Xu 等人[14]基于 LQN 模型运用 Jess 框架定义了一组不特定于 SA 建模语言的性能改进规则。Bachmann 等人[31]通过综合考虑 SA 的可修改性，定义了一组性能改进规则。Cadoret 等人[32]面向嵌入式应用领域的 SA 描述语言定义了一组性能改进规则。

在 SA 层性能改进规则的支持下，基于规则的优化方法还设计了相应的优化算法。这些算法运用一定策略在各种规

则组合所构成的改进空间中，搜索具有最优性能指标的 SA。这些优化算法主要采用步进式[27-32]和按轮迭代的深度优先[14]两种搜索策略。

在基于规则的 SA 层性能优化方法中，仅 PB 方法[14]所定义的规则面向 LQN 模型[18]而不特定于具体的 SA 建模语言。鉴于 PB 方法具有较好的通用性，将其作为本文的对比方法。下面详细介绍 PB 方法的性能改进规则和性能优化算法。

(1)PB 方法定义的性能改进规则

在 PB 方法中，性能改进规则被划分为配置改进规则和设计改进规则。配置改进规则共 3 条，用于发现因资源配置不当而引起的性能瓶颈问题，并通过变更硬件和软件资源使用数目、重新配置将其予以消除。而设计改进规则共 4 条，用于发现因设计不当而引起的系统响应时间过长的问题。

根据性能改进规则在编码阶段的实现难度，PB 方法将每条规则的单位改进代价值定义为使用一次该规则并将一个元素改进一个单位幅度时所对应的改进代价。由于配置改进规则不需要考虑代码实现，因此 PB 方法定义各配置改进规则的单位改进代价值为 0，而各设计改进规则的单位改进代价值请参见文献[14]。

(2)PB 方法设计的性能优化算法

PB 方法的优化算法采用按轮迭代的深度优先搜索策略。搜索始于 LQN_0 模型(从初始软件体系结构 SA_0 抽取获得)的根结点，经初始的配置改进后，以轮为单位进行迭代式搜索。搜索过程可表示为如图 1 所示的一棵搜索树。

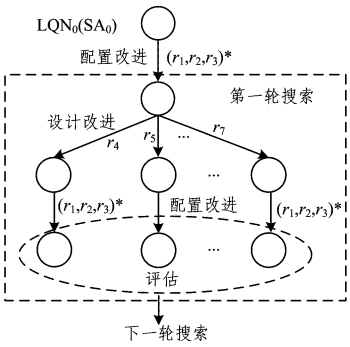


图 1 PB 方法的搜索过程

在初始配置改进时，3 条配置改进规则构成的序列〈r_1，r_2，r_3〉被反复应用直至无任何性能改进效果，进而得到一个配置改进后的 SA，并将其作为第一轮搜索的起始 SA。在图 1 虚矩形框标出的第一轮搜索中，首先将 4 条设计改进规则 r_4，r_5，r_6 和 r_7 分别应用于第一轮的起始 SA 上，并获取一组设计改进后的 SA；然后分别对这组 SA 中的每个 SA 分别进行配置改进，进而得到一组候选的 SA；最后对这组 SA 进行评估，选择最好的一个 SA 作为下一轮搜索的起始 SA。如此以轮为单位反复进行迭代，直至不能发现任何改进后停止搜索。

文献[14]将系统响应时间和累计改进代价作为搜索过程的两种评价指标，提出了改进代价优先和系统响应时间优先两种不同的优化方法。累计改进代价被定义为从根结点出发到当前结点的路径 l 上所有结点的改进代价的累计值，如式(1)中的$cost(l)$所示。在式(1)中，根结点 $root$ 的改进代价值为 0；而对非根结点 $node_i$ 而言，若从父结点通过使用规则 j 修改 N 个元素后到达它，且 w_j 和 $ampl_k$ 分别是规则 j 的单位改进代价和第 k 个元素的改进幅度，则 $node_i$ 的改进代价由式(1)中的 $cost(node_i)$定义。

$$\begin{aligned} &cost(root)=0\\ &cost(node_i)=\sum_{k=1}^{N} w_j\times ampl_k \\ &cost(l)=\sum_{node_i\in l} cost(node_i) \end{aligned} \tag{1}$$

为了便于阐述本文方法与 PB 方法的实验对比，将采用改进代价优先和系统响应时间优先两种 PB 方法，分别称为 PB1 和 PB2 方法。在 PB1 和 PB2 方法中，从根结点到优化结果结点的路径 l^* 上所使用的规则及其改进幅度可以视为对初始 LQN_0 模型的一个最佳性能改进方案 sln^*，并且将 l^* 路径上各结点改进代价的累计值作为方案 sln^* 的改进代价值。

3 用于 SA 层性能改进的随机搜索规则

通过对每个策略的改进方案进行随机化处理，将已有的 17 种 SA 层性能改进策略[13]描述成 17 条对应的随机搜索规则，并基于部署、结构和行为 3 个视图来组织这些规则。

表 1 列出了部署视图中的 6 条随机搜索规则，这些规则通过随机重新部署资源、随机增加资源的多重性和实例、随机提高资源处理速度等方法，来缓解出现的资源瓶颈问题。表 2 列出了结构视图中的 6 条随机搜索规则的定义，这些规则通过随机改变组件的接口、连接以及职责分配等结构元素来改进资源使用率、吞吐量和系统响应时间等。表 3 列出了行为视图中的 5 条随机搜索规则。这些规则通过随机修改行为元素而引入更多的异步通信、并发和并行化，从而解决系统响应时间过长的性能问题。

表 1 部署视图中的随机搜索规则

规则号	策略名	随机搜索规则	
		条件	动作
r_1	组件重部署	P 为最高使用率的处理器，且 P 上部署了一组组件 Coms	从 Coms 中随机选择一个组件并将其部署到使用率最低的处理器上
r_2	组件复制	C 既是使用率最高的组件，也是被访问次数最多的组件	随机增加 C 的实例数，但要低于其上限
r_3	更快硬件	P 是使用率最高的处理器	随机增加 P 的处理速度，但要低于上限
r_4	更多硬件	P 是使用率最高的处理器	随机增加 P 的多重性，但要低于上限
r_5	局部化	C1 是使用率最高的组件，且有一组组件 Coms 与之交互	从 Coms 中随机选择组件 C2，将 C1 重部署到 C2 所在的处理器上
r_6	资源池	C1 是使用率最高的组件	随机增加 C1 的多重性，但要低于其上限

表 2　结构视图中的随机搜索规则

规则号	策略名	随机搜索规则	
		条件	动作
r_7	批处理	Cal 是调用次数最多且大于 1 的跨网络调用	随机减少 Cal 的数量并增加每次数据传输的大小
r_8	缓存	Cal 是调用次数最多且大于 1 的调用	创建随机数量的缓存组件
r_9	划分	C1 是使用率最高的组件	创建组件 C2，并随机从 C1 上移动一些操作放入 C2
r_{10}	耦合和内聚	Cals 是调用数目大于 1 的一组调用	从 Cals 中随机选择一个 Cal，并将 Cal 的调用方组件与被用方组件进行合并
r_{11}	数据结构和算法	C 是使用率最高的组件，且 C 有一组可替换组件 Coms	从 Coms 中随机选择一个组件 C2，用 C2 替换 C
r_{12}	远程数据交换	Cal 是跨网络交换数据最多的调用	创建压缩组件 C1 和解压缩组件 C2，并随机设定压缩比，并将 C1 和 C2 用于 Cal 的调用方和被调用方

表 3　行为视图中的随机搜索规则

规则号	策略名	随机搜索规则	
		条件	动作
r_{13}	异步通信	服务 Srv 的响应时间大于期望值，且 Srvs 的请求路径上有一组可被异步化的同步调用 Cals	从 Cals 中随机选择一个或多个调用，并将其转变成异步调用
r_{14}	并发或并行化	C 是使用率最高的组件，且 C 有一个线程池	随机增大 C 的线程池，但低于上限
r_{15}	快速路径	服务 Srv 的响应时间大于期望值，且 Srvs 是 Srv 请求路径上的一组服务	创建组件 C，并随机从 Srvs 中选择若干个服务放入 C 中
r_{16}	锁定粒度	C 是使用率最高的组件，且在 C 的内部行为中有一组可以减少获得和释放锁时间的动作 Acs	从 Acs 中随机选择一些动作，并修改它获得和释放锁的行为
r_{17}	状态管理	状态组件 C1 有一组可替换组件 Coms	从 Coms 中随机选择组件 C2，并用 C2 替换 C1

表 1－表 3 定义的 17 条随机搜索规则均具有较好的通用性，但由于各种 SA 建模语言在描述软件系统的结构、行为和部署元素的详略程度和语法成份上有所差异，因此在随机搜索规则的实现上也会存在一定的差异。另外，受制于 SA 建模语言的描述内容和描述能力，往往只能实现上述 17 条随机搜索规则中的一部分。为了与 PB 方法（参见相关工作部分）进行实验对比，我们实现了一组基于 LQN 模型的随机搜索规则[1)]。对于特定的用 SA 建模语言描述的一个具体应用的 SA 而言，因常常只能修改某些特定的 SA 元素，故在优化性能时仅需在这种 SA 建模语言下实现的随机搜索规则中选用合适的规则即可。

4　基于随机搜索规则的 SA 层性能优化模型（RRPOM）

在随机搜索规则的基础上，将 SA 层性能优化抽象为求解最优规则序列的数学模型 RRPOM。下面给出 RRPOM 模型的详细定义。后文均用 SA_0 表示初始软件体系结构。

定义 1　从 1 到 n 依次对随机搜索规则进行编号，并用 u_i 表示第 $i(1\leqslant i\leqslant n)$ 条规则最多允许使用的次数。

定义 2　定义函数 exe 表示将规则 r_i 应用到软件体系结构 SA_j 上，从而获得变更元素集 E_i。其中，E_i 中的每个元素由被修改元素的名字和修改后的值构成。若 r_i 的条件不满足，则 E_i 为空集 $\emptyset$。

定义 3　定义函数 $modi$ 表示按变更元素集 E_i 中的每个元素对软件体系结构 SA_j 进行修改，从而获得修改后的软件体系结构 SA_k。若 E_i 为空集 $\emptyset$，则 SA_k 与 SA_j 相同。

定义 4　定义变换函数 $t(i,SA_j)=modi(exe(i,SA_j),SA_j)$ 表示将规则 r_i 应用到软件体系结构 SA_i 上，从而获得结果软件体系结构。

定义 5　基于随机搜索规则的 SA 层的性能优化问题的解 X 的定义如式(2)所示。

在式(2)中，x_i 表示规则编号，$E_i=exe(x_i,SA_{i-1})$，且 $SA_i=t(x_i,SA_{i-1})(1\leqslant i<l)$。

设 u_i 和 $h_i(X)$ 分别表示第 i 条规则在 X 中最多允许出现的次数和实际出现的次数，则 X 的长度 l、X 中每个元素 x_k 和 $h_i(X)$ 的取值范围需满足式(3)－式(5)。其中，i,x_k,k,l 和 n 均为自然数。

$$X=\langle(x_1,SA_0,E_1),\cdots,(x_i,SA_{i-1},E_i),\cdots,(x_l,SA_{l-1},E_l)\rangle \tag{2}$$

$$l\leqslant\sum_{i=1}^{n}u_i \tag{3}$$

$$1\leqslant x_k\leqslant n,1\leqslant k\leqslant l \tag{4}$$

$$0\leqslant h_i(X)\leqslant u_i,1\leqslant i\leqslant n \tag{5}$$

定义 6　定义函数 $\vec{t}(X,SA_0)$ 表示将 X 中的规则依次作用到 SA_0 上，从而获取结果软件体系结构。函数 $\vec{t}(X,SA_0)$ 的定义如式(6)所示。

$$\vec{t}(X,SA_0)=t(x_l,SA_{l-1}) \tag{6}$$

定义 7　称解 $\overline{X}$ 是可解释的，当且仅当式(7)中的条件 $\forall i(E_i\neq\emptyset)$ 被满足。通过式(8)定义的函数 $inpt$ 可将解 X 转换成对应的可解释的解 $\overline{X}$。

$$\overline{X}=\langle(x_1,SA_0,E_1),\cdots,(x_i,SA_{i-1},E_i),\cdots,(x_l,SA_{l-1},E_l)\rangle \tag{7}$$

$$inpt(X)=\{(x_i,SA_{i-1},E_i)\mid(x_i,SA_{i-1},E_i)\in X\wedge E_i\neq\emptyset\} \tag{8}$$

定义 8　若 f_1 和 f_2 分别表示计算系统响应时间和改进代价的目标函数，则基于随机搜索规则的性能优化问题可以描述为式(9)定义的两目标优化问题，对应的 Pareto 最优解集 Opt 可由式(10)定义。式(10)中的 Ω 代表解空间。

1) https://git.oschina.net/login，登录的用户名和密码分别为：reviewer 和 reviewer2016

式(11)给出了式(10)所对应的可解释的 Pareto 最优解集 Opt^*。

$$SA_l=\vec{t}(X,SA_0) \tag{9}$$

$$Minimize\ F(X)=(f_1(SA_l),f_2(SA_l))$$

$$Opt=\{X\mid \exists X'\in\Omega\wedge\forall i(f_i(X')\leqslant f_i(X))\wedge\exists j(f_j(X')\leqslant f_j(X))\wedge i,j\in\{1,2\}\} \tag{10}$$

$$Opt^*=\{X^*\mid X^*=inpt(X)\wedge X\in Opt\} \tag{11}$$

5 RRPOM 模型的演化求解算法

RRPOM 模型的演化求解算法的总体框架与 NSGA-II[33]类似。下面仅对该算法的个体编码、遗传操作和适应度求解等步骤做进一步阐述。

5.1 个体编码

图 2 给出了个体 X'的编码，它由 3 段构成。浅灰色背景指示的第一段 $X'(R)$是规则号序列，记第 i 个位置上的规则号为 $X'(R_i)$。白色背景指示的第二段 $X'(P_E)$是指针序列，记第 i 个位置上的指针为 $X'(P_{Ei})$。$X'(P_{Ei})$指向的变更元素集是 $X'(R_i)$规则号被执行后获得的变更元素集。最后一段是黑色背景指示的 strtPos 位，strtPos 位用于定义从规则序列$X'(R)$的哪一位开始执行。初始个体 X'的 strtPos 置为 0，当 $X'(R)$中的规则被依次执行后，X'的 strtPos 也将依次增加 1。

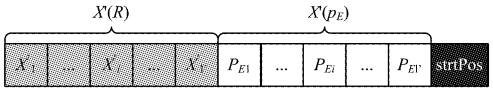


图 2 个体 X'的编码

为了保证 $X'(R)$中规则的使用次数能够遵从 RRPOM 模型中式(3)和式(5)定义的约束条件，并得到尽可能短的规则序列，引入了 0 号规则。0 号规则应用到任何 SA 时都不做修改，即式(12)恒成立。规定规则 0 最多允许使用的次数等于所有非 0 规则最多允许使用的次数之和，如式(13)所示。

$$t(0,SA)=SA \tag{12}$$

$$u_0=\sum_{k=1}^{n}u_k \tag{13}$$

在 $X'(R)$中，长度 l'、每个元素 x_k'及第 i 条规则实际使用次数的取值范围由式(14)－式(16)定义。

$$l'=u_0 \tag{14}$$

$$0\leqslant x_k'\leqslant n,1\leqslant k\leqslant l' \tag{15}$$

$$h_i(X'(R_i))\leqslant u_i,0\leqslant i\leqslant n \tag{16}$$

5.2 遗传操作

单点变异和单点交叉被采用并应用于编码中的 $X'(R)$规则号序列段。当变异和交叉完成后，参与操作个体的 strtPos 将被设定为遗传操作的位置和当前 strtPos 中的最小值。与此同时，修复机制被用于保证遗传操作后的个体满足式(16)定义的约束。具体地，从遗传操作位置到 $X'(R)$的最后 1 位被依次检查，以验证是否违反式(16)的约束。对于违反的位，将其置为 0。例如：$X'(R)_1$ 和 $X'(R)_2$ 都是由规则号 1，2，3 和 0 所构成的序列段，并且这 4 条规则的最大次数分别为 $u_1=3,u_2=2,u_3=3,u_0=8$，假定交叉位置为 4。图 3 给出了 RRMO4PO 算法对 X_1'和 X_2'进行交叉的过程。

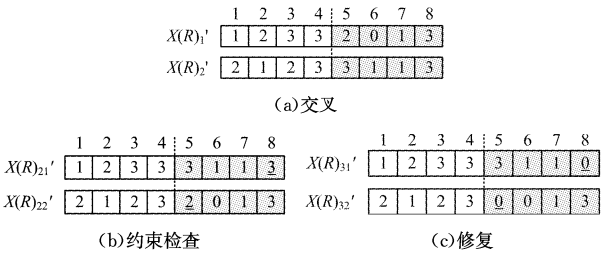


图 3 RRMO4PO 算法交叉算子计算步骤的例子

5.3 适应度求解

图 4 给出了个体 X'的适应度求解过程。首先将 $X'(P_E)$的第 0 位到第 strtPos 位所指向的变更元素集依次应用到 SA_0 中，可生成起始 SA；然后从 $X'(R)$的 strtPos 位到第 l'位依次取出规则号并执行对应的规则，同时记录变更元素集、设定 $X'(P_{Ei})$并修改当前 SA；最后根据结果 SA，通过性能和改进代价评估来获取系统的响应时间和改进代价，进而得到个体 X'的两个目标值。

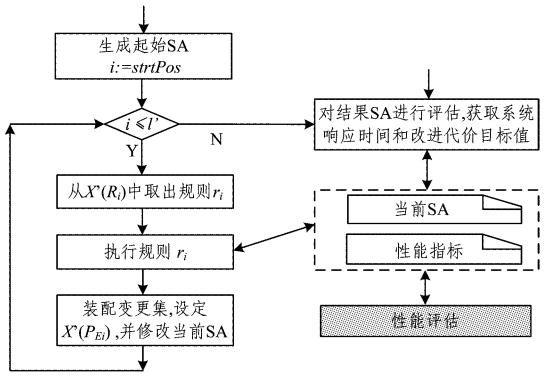


图 4 适应度求解过程

6 案例研究

通过案例将本文方法与 PB 方法进行对比，以验证本文方法的有效性。下面先描述验证的问题及对应的度量指标，然后介绍 WebApp 案例，再给出实验的设定，最后展示实验结果并作简要分析。

6.1 验证的问题及评估指标

第 2 节给出了 PB 方法的简介，其中包括 PB1 和 PB2 两种子方法，分别对应优化过程中改进代价优先和系统响应时间优先两种评价策略。本文的验证问题主要针对 PB1 和 PB2 两种方法展开。

问题 1(RQ1) 与 PB1 和 PB2 方法相比，本文方法是否可以获取更短的系统响应时间及更低代价的解？通过回答该问题，验证本文方法是否能够增大搜索空间、获取更优的解。由于本文方法输出的是 Pareto 最优解集，而 PB1 和 PB2 方法输出在不同评价策略下的最佳解，因此需要分两个步骤进行回答。首先要判断本文方法获取的解集 S 是否优于 PB1 和 PB2 获取的解，再观测 S 中这些解对应的系统响应时间和改进代价。因此，提出 N1 和 N2 两个指标分别度量本文方法获取的解集中有多少个解优于 PB1 和 PB2。下面给出它们的具体定义。

设 s 是本文方法、PB1 方法或 PB2 方法求得的一个解。将 s 对应的目标值表示为序偶$\langle r,c\rangle$。其中，r 和 c 分别表示

系统的响应时间和改进代价。设 $pr1$ 和 $pr2$ 分别表示从解 s 中获取 r 和 c 的投影函数。若 s_1 和 s_2 分别是 PB1 和 PB2 求得的最优解，而 S 是本文方法得到的 Pareto 解集，则 $N1$ 和 $N2$ 分别表示本文方法获取的优于 PB1 和 PB2 方法的解的个数，其定义如式(17)和式(18)所示：

$$\eta_1=\{s|s\in S\wedge pr_1(s)\leqslant pr_1(s_1)\wedge pr_2(s)<pr_2(s_1)\}$$
$$N1=|\eta_1| \tag{17}$$

$$\eta_2=\{s|s\in S\wedge pr_1(s)<pr_1(s_2)\wedge pr_2(s)\leqslant pr_2(s_2)\}$$
$$N2=|\eta_2| \tag{18}$$

问题 2(RQ2)　在优于 PB1 和 PB2 的解中，本文方法能否获取更好的解释？通过对该问题的回答，验证本文方法在获取高质量解的同时，能否获得更加简洁的解释。

在获取最优解的过程中，平均使用有改进效果的规则的次数 $avgImpRlNum$ 和平均修改 SA 元素的次数 $avgMdfNum$ 成为解释性优劣的指标，且它们的值越小则表明解释性越好。由于 PB1 和 PB2 方法求出的是单解，因此 $avgImpRlNum$ 和 $avgMdfNum$ 是它们在求取最佳解的过程中记录的使用有改进效果的规则的次数和修改 SA 元素的次数。本文方法与 PB1 方法对比时，$avgImpRlNum$ 和 $avgMdfNum$ 的计算分别如式(19)和式(20)所示，而与 PB2 对比时 $avgImpRlNum$ 和 $avgMdfNum$ 的计算分别如式(21)和式(22)所示。式(19)－式(22)中，$imprNum(s)$ 和 $E(s)$ 分别表示获取的解 s 中使用有改进效果的规则的次数和修改 SA 元素的次数，而 η_1 和 η_2 则由式(17)和式(18)定义。

$$\frac{\sum_{s\in\eta_1} imprNum(s)}{|\eta_1|} \tag{19}$$

$$\frac{\sum_{s\in\eta_1} E(s)}{|\eta_1|} \tag{20}$$

$$\frac{\sum_{s\in\eta_2} imprNum(s)}{|\eta_2|} \tag{21}$$

$$\frac{\sum_{s\in\eta_2} E(s)}{|\eta_2|} \tag{22}$$

6.2　案例介绍

在 WebApp 案例[14]中，用户通过浏览器访问部署在 Web 服务器 WS 上的页面，Web 服务器加载和处理这些页面时，需要分别调用部署在两个应用程序服务器 App1 和 App2 上的业务服务，而这些业务服务又需要分别访问部署在两个数据库服务器 DB1 和 DB2 上的数据管理服务。WebApp 的初始 LQN 模型如图 5 所示。

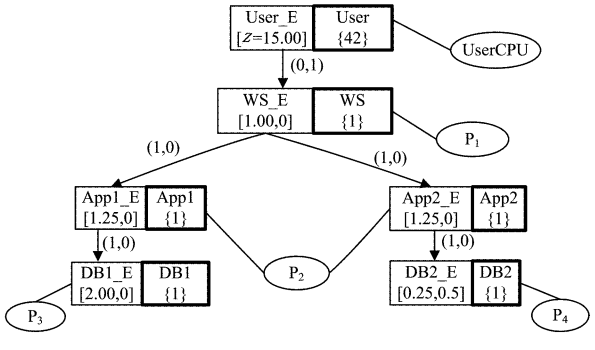


图 5　WebApp 案例的初始 LQN 模型

图 5 中加粗矩形 User，WS，App1，App2，DB1 和 DB2 定义了 6 种软件资源，其中花括号中的数字设定了这些资源的当前可用数目。这 6 种软件资源被分别部署在由椭圆形表示的各个处理器上，如 App1 和 App2 资源被部署在 P2 处理器上。每种软件资源向外部提供的服务/入口由加粗矩形表示。顶层入口 User_E 中的 $Z=15.00$，用于模拟用户通过浏览器发出请求之间的时间间隔；而非顶层入口中中括号中的两个数字分别用于表示入口服务的第一阶段和第二阶的处理时间，当入口提供同步服务时，第二阶段的处理时间为 0。入口之间的调用关系用有向箭头表示，箭头旁边的圆括号中的数字分别用于设定调用目标入口的两个阶段服务的平均次数。

6.3　实验设定

PB1 和 PB2 方法中使用的规则、各规则的改进幅度和改进代价的设定请参见文献[14]。下面给出本文方法中随机搜索规则、演化优化算法参数的设定，以及所采用的统计方法。

(1)随机搜索规则的设定

为了公平，按照文献[14]确定的 7 种可用的性能改进方案，选用表 1－表 3 中的 r_1，r_4，r_6，r_7，r_9，r_{11} 和 r_{13} 共 7 条随机规则。这些规则对应于 PB1 和 PB2 方法使用的 7 条规则 r^3_{PB}，r^1_{PB}，r^2_{PB}，r^7_{PB}，r^6_{PB}，r^4_{PB} 和 r^5_{PB}，并且设定它们的单位改进代价与 PB1 和 PB2 方法相同。根据各条规则和 WebApp 案例中关联元素的个数、初始值和最大值，设定了 7 条随机规则的最多使用次数，如表 4 所列。

表 4　WebApp 案例选用的随机搜索规则及其最大使用次数

规则	r_1	r_4	r_6	r_7	r_9	r_{11}	r_{13}
使用次数	3	3	12	1	3	6	6

(2)演化算法的设定

设定本文演化算法中改进代价目标值的计算与 PB1 和 PB2 相同，都采用第 2 节相关工作中的式(1)。对于本文演化算法，式(1)中的 l 对应个体长度；$root$ 根结点对应于码长为 0 时的情景；而 $node_i$ 对应于个体编码中的第 i 个元素 (x_i, SA_{i-1}, E_i)，$ampl_k$ 则表示 E_i 中元素 k 的改进幅度，其可通过两步计算。首先在 $E_j(1\leqslant j<i)$ 中查找元素 k，记录它第一次出现时的值；然后将 E_i 中元素 k 的值与该值作差可求得 $ampl_k$。另外，本文演化算法的种群大小、交叉概率、变异概率和迭代次数被分别设为 30，0.8，0.6 和 200。

(3)采用的统计方法

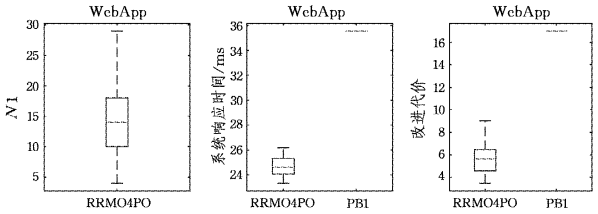
独立运行本文演化算法 30 次，并按照设定的度量指标收集结果数据，同时在置信水平为 0.05 的情况下进行秩和统计实验。在秩和实验中，当 $p\text{-}value$ 值小于 0.05 时，说明本文方法优于对比方法。

6.4　实验结果

下面分别给出 WebApp 案例中 RQ1 和 RQ2 两个问题的实验结果。

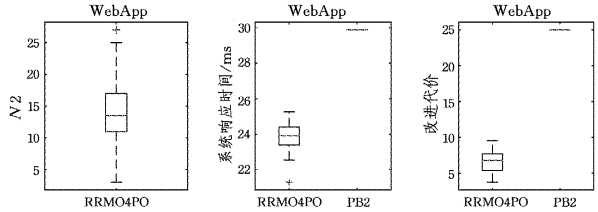
(1)RQ1 的实验结果

图 6(a)和图 7(a)给出了本文 RRMO4PO 方法可以获取比 PB1 和 PB2 方法更优的解，并且从表 5 的秩和实验的 $p\text{-}value$ 值可以看出，这些解在系统响应时间和改进代价上显著优于 PB1 和 PB2 方法。该结论与图 6(b)、图 6(c)及图 7(b)、图 7(c)的观测结果一致。



(a)优于 PB1 的解的个数 $N1$ (b)系统响应时间 (c)改进代价

图 6 本文 RRMO4PO 方法与 PB1 方法的比较



(a)优于 PB2 的解的个数 $N2$ (b)系统响应时间 (c)改进代价

图 7 本文 RRMO4PO 方法与 PB2 方法的比较

表 5 RRMO4PO 与 PB1 和 PB2 方法在各对比指标下进行秩和实验的 *p-value* 值

指标	PB1	PB2
系统响应时间	1.212e−12	1.212e−12
改进代价	1.212e−12	1.212e−12
平均使用有改进效果规则的次数 *avgImpRlNum*	1.211e−12	1.212e−12
平均修改 SA 元素的次数 *avgMdfNum*	1.210e−12	1.211e−12

表 6 和表 7 所列内容说明了本文方法获取的 Pareto 解集中平均有 14 个解优于 PB1 和 PB2 方法。从表 6 可进一步看出:本文方法获取的系统响应时间和改进代价的均值为 24.71ms和 5.62,优于 PB1 方法 30.5%和 66.9%。由表 7 可以看出,本文方法获取的系统响应时间和改进代价的均值为 23.86ms 和6.61,优于 PB2 方法 20.1%和 73.6%。

表 6 RRMO4PO 与 PB1 方法解质量的均值比较

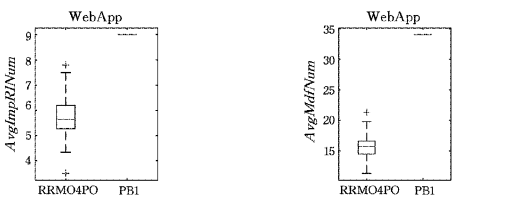
	更优解的个数 $N1$	系统响应时间/ms	改进代价
RRMO4PO	14	24.71	5.62
PB1	0	35.5518	17

表 7 RRMO4PO 与 PB2 方法解质量的均值比较

	更优解的个数 $N2$	系统响应时间/ms	改进代价
RRMO4PO	14	23.86	6.61
PB2	0	29.88	25

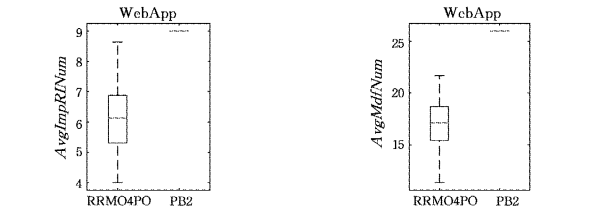
(2)RQ2 的结果

对于本文方法优于 PB1 和 PB2 方法的解,表 5 中秩和实验的 *p-value* 值表明了本文方法在平均使用有改进效果的规则的次数和平均修改 SA 元素的次数上显著优于 PB1 和 PB2 方法。该结论与图 8 和图 9 的观测结果一致。



(a)平均使用有改进效果的规则的次数 *avgImpRlNum* (b)平均修改 SA 元素的次数 *avgImpRlNum*

图 8 本文 RRMO4PO 方法与 PB1 方法的比较



(a)平均使用有改进效果的规则的次数 *avgImpRlNum* (b)平均修改 SA 元素的次数 *avgImpRlNum*

图 9 本文 RRMO4PO 方法与 PB2 方法的比较

表 8 表明在优于 PB1 方法的解中,本文方法使用的有改进效果的规则的次数平均仅为 6,且平均仅需修改 16 次元素。这较 PB1 方法降低了 33.3%和 52.9%。类似地,由表 9 可以看出本文方法平均仅需使用有改进效果的规则 6 次,且平均仅需修改 17 次元素,这较 PB2 方法降低了 33.3%和 34.6%。

表 8 RRMO4PO 与 PB1 方法的解的可解释性比较

	平均使用有改进效果的规则的次数 *avgImpRlNum*	平均修改 SA 元素的次数 *avgMdfNum*
RRMO4PO	6	16
PB1	9	34

表 9 RRMO4PO 与 PB2 方法的解的可解释性比较

	平均使用有改进效果的规则的次数 *avgImpRlNum*	平均修改 SA 元素的次数 *avgMdfNum*
RRMO4PO	6	17
PB2	9	26

图 10 和图 11 分别给出了 PB1 和 PB2 方法优化后的结果 LQN 模型,而图 12 是本文方法获取的 Pareto 解集中既优于 PB1 又优于 PB2,且具有最短系统响应时间的解所对应的结果 LQN 模型。

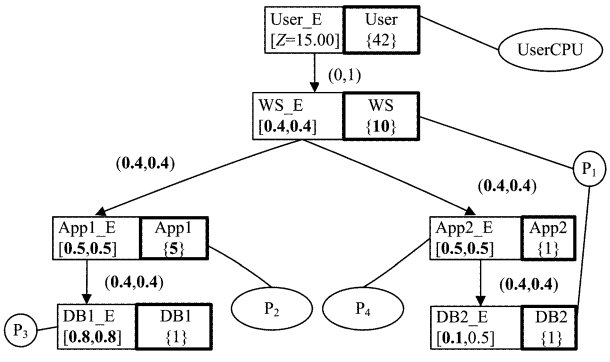


图 10 PB1 方法优化后的结果 LQN 模型

图 10−图 12 中被黑体标记元素的值是相对于图 5 所示的初始 LQN 模型经多次修改后,最终取得的值。图 10−图 12中分别有 21,21 和 17 个元素被修改。这说明 PB1,PB2 较本文方法多修改了 4 个元素。另外,从图 12 与图 10、图 11 的对比可以看出黑体标记元素被改进的幅度不同于 PB1 和 PB2 方法,这说明了每个规则的改进幅度将影响最终的优化结果。此外,在 PB1、PB2 和本文方法获取图 10−图 12 结果 LQN 模型的过程中,对有改进效果的规则进行记录。PB1 和 PB2 获取的规则序列为〈r_2,r_2,r_2,r_2,r_2,r_3,r_5,r_3,r_4〉和〈r_2,

r_2，r_2，r_2，r_2，r_3，r_4，r_2，r_5〉。为了便于对比，我们对 RRMO4PO 方法所获取序列的各规则进行编号，按照与 PB1 和 PB2 方法中规则的对应关系进行修改，最后获得的序列为〈r_5，r_3，r_2，r_4，r_2〉。对比这 3 个规则序列可以看出，各个规则的使用顺序和使用次数也会对优化结果产生直接影响。PB1 和 PB2 方法未充分考虑每个规则各种可能的改进幅度、使用顺序和使用次数，因而只能在受限的空间搜索最优性能改进方案，从而导致 PB1 和 PB2 方法未能获取较本文方法更优的性能改进方案。

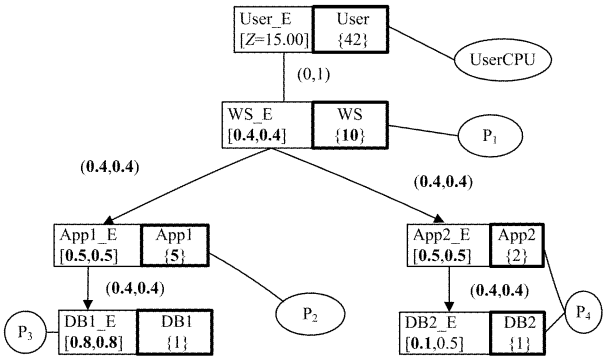


图 11　PB2 方法优化结果 LQN 模型

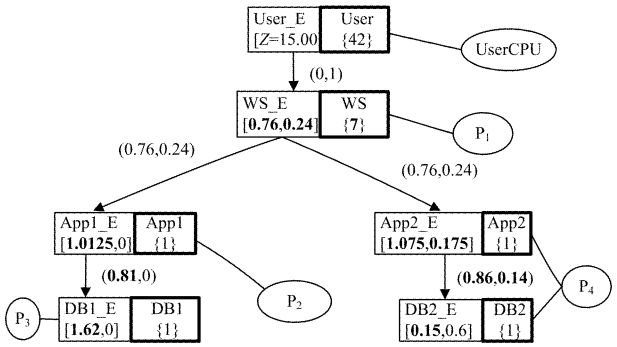


图 12　RRMO4PO 方法优化结果 LQN 模型

结束语　文中通过对 SA 层的性能改进策略进行随机化处理，定义了能增大性能改进空间的一组随机规则；在此基础上考虑这些规则的不同使用顺序和不同使用次数对优化结果的影响，构建了一种 SA 层的性能优化模型并设计对应的演化求解算法，以进一步增大规则组合的性能改进空间。Web-App案例的实验结果表明，通过单规则和规则组合两层改进空间的增大，本文方法在使用更少的规则、更少次修改 SA 元素而获取更好可解释性的同时，有效地降低了系统响应时间和改进代价。未来我们将系统响应时间的预测模型引入到本文算法的演化过程中以减少优化时间，并将本文方法应用到更多的 SA 建模语言中，以进一步扩展本文方法的适用范围。

参 考 文 献

[1] BROSIG F, MEIER P, BECKER S, et al. Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-Based Architectures[J]. IEEE Trans. on Softw. Eng., 2015, 41(2): 157-175.

[2] CORTELLESSA V. Performance antipatterns: State-of-art and future perspectives[C]// Computer Performance Engineering. Springer, 2013: 1-6.

[3] KOZIOLEK A. Automated improvement of software architecture models for performance and other quality attributes[OL]. http://www.gbv.de/dms/tib-ub-bannover/776654306.pdf.

[4] AMOOZEGAR M, NEZAMABADI-POUR H. Software performance optimization based on constrained GSA[C]// 2012 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP). 2012: 134-139.

[5] AMOOZEGAR M, NEZAMABADI-POUR H. A multi-objective approach to model-driven performance bottlenecks mitigation [J]. Scientia Iranica, 2015, 22(3): 1018-1030.

[6] MARTENS A, KOZIOLEK H. Automatic, model-based software performance improvement for component-based software designs[J]. Electron. Notes Theor. Comput. Sci., 2009, 253(1): 77-93.

[7] SAED A A, KADIR W, WAN M N. Applying particle swarm optimization to software performance prediction an introduction to the approach[C]// 2011 5th Malaysian Conference on Software Engineering (MySEC). 2011: 207-212.

[8] BUSCH A, KOZIOLEK A. Considering Not-Quantified Quality Attributes in an Automated Design Space Exploration[C]// 2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA). 2016: 50-59.

[9] LI R, ETEMAADI R, EMMERICH M T, et al. An evolutionary multiobjective optimization approach to component-based software architecture design[C]// 2011 IEEE Congress on Evolutionary Computation (CEC). 2011: 432-439.

[10] TRIBASTONE M. Efficient optimization of software performance models via parameter-space pruning[C]// Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. 2014: 63-73.

[11] LI H, CASALE G, ELLAHI T. SLA-driven planning and optimization of enterprise applications[C]// Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering. 2010: 117-128.

[12] WILLIAMS L G, SMITH C U. PASA (SM): An Architectural Approach to Fixing Software Performance Problems[C]// Int. CMG Conference. 2002: 307-320.

[13] KOZIOLEK A, KOZIOLEK H, REUSSNER R. Peropteryx: automated application of tactics in multi-objective software architecture optimization[C]// Proceedings of the joint ACM SIGSOFT conference-QoSA and ACM SIGSOFT symposium-ISARCS on Quality of software architectures-QoSA and architecting critical systems-ISARCS. 2011: 33-42.

[14] XU J. Rule-based automatic software performance diagnosis and improvement[J]. Perform. Eval., 2012, 69(11): 525-550.

[15] DU X, YAO X, NI Y, et al. An evolutionary algorithm for performance optimization at software architecture level[C]// 2015 IEEE Congress on Evolutionary Computation (CEC). 2015: 2129-2136.

[16] DU X, NI Y, YE P. A Multi-objective Evolutionary Algorithm for Rule-based Performance Optimization at Software Architecture Level[C]// Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. 2015: 1385-1386.

（下转第 174 页）

算机应用研究,2010,27(11):4080-4086.

[12] HOLME P,KIM B J,YOON C N,et al. Attack vulnerability of complex networks [J]. Physical Review E Statistical Nonlinear & Soft Matter Physics,2002,65(5):634-649.

[13] WU J,TAN Y J. Study on measure of complex network invulnerability [J]. Journal of Systems Engineering,2005,20(2):128-131. (in Chinese)
吴俊,谭跃进. 复杂网络抗毁性测度研究[J]. 系统工程学报,2005,20(2):128-131.

[14] WU J,BARAHANA M,TAN Y J,et al. Natural connectivity of complex networks [J]. Chinese Physics Letters,2010,27(7):078902.

[15] WU J,BARAHANA M,TANY J,et al. Spectral measure of structural robustness in complex networks [J]. IEEE Transactions on Systems Man and Cybernetics Part A,2011,41(6):1244-1252.

[16] DEAN J,GHEMAWAT S. MapReduce:Simplified data processing on large clusters[J]. Communication of the ACM,2008,51(1):107-113.

[17] OUYANG M,FEI Q,YU M. Survey on Efficiency and Vulnerability of Complex Network [J]. Computer Science,2008,35(6):1-4. (in Chinese)
欧阳敏,费奇,余明. 复杂网络的功效性与脆弱性研究综述[J]. 计算机科学,2008,35(6):1-4.

[18] TAN Y J,WU J,DENG H Z. Progress in invulnerability of complex network [J],University of Shanghai for Science and Technology,2011,33(6):643-668. (in Chinese)
谭跃进,吴俊,邓宏钟. 复杂网络抗毁性研究进展[J]. 上海理工大学学报,2011,33(6):643-668.

[19] DU Y H. The research of graph algorithm based on cloud computing [D]. Beijing:Beijing University of Posts and Telecommunications,2011. (in Chinese)
杜雅红. 基于云计算平台的图算法研究[D]. 北京:北京邮电大学计算机学院,2011.

[20] CAIDA. Topology : Macroscopic Internet Topology Data Kit (Ark ITDK;restricted) [EB/OL]. (2015-10-01) [2016-03-15] https://topo-data.caida.org.

[21] 汪小帆,李翔,陈关荣. 复杂网络理论及其应用[M]. 北京:清华大学出版社,2006:10-11.

[22] XIE C,MAI L D,DU Z H,et al. Research and analysis of Parallel computing system speedup [J]. Computer Engineering and Applications,2003,39(26):66-68. (in Chinese)
谢超,麦联叨,都志辉,等. 关于并行计算系统中加速比的研究与分析[J]. 计算机工程与应用,2003,39(26):66-68.

(上接第 163 页)

[17] KOUNEV S. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets[J]. IEEE Trans. on Softw. Eng.,2006,32(7):486-502.

[18] TRIBASTONE M. A fluid model for layered queueing networks [J]. IEEE Trans. on Softw. Eng.,2013,39(6):744-756.

[19] DISTEFANO S,SCARPA M,PULIAFITO A. From UML to Petri Nets:The PCM-Based Methodology[J]. IEEE Trans. on Softw. Eng.,2011,37(1):65-79.

[20] TRIBASTONE M,GILMORE S,HILLSTON J. Scalable Differential Analysis of Process Algebra Models[J]. IEEE Trans. on Softw. Eng.,2012,38(1):205-219.

[21] KOZIOLEK A,ARDAGNA D,MIRANDOLA R. Hybrid multi-attribute QoS optimization in component based software systems [J]. J. Syst. Softw.,2013,86(10):2542-2558.

[22] HUANG X,WANG W,ZHANG W B,et al. Automatic performance modeling approach to performance profiling of Web applications[J]. Journal of Software,2012,23(4):786-801. (in Chinese)
黄翔,王伟,张文博,等. 面向性能剖析的 Web 应用自动性能建模方法[J]. 软件学报,2012,23(4):786-801.

[23] LI C H,WANG W M,SHI Y Y. Performance prediction method for UML software architecture and its automation[J]. Journal of Software,2013,24(7):1512-1528. (in Chinese)
李传煌,王伟明,施银燕. 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.

[24] NAVARRO E,CUESTA C E,PERRY D E,et al. Antipatterns for architectural knowledge management[J]. Int. J. Inf. Technol. Decis. Mak.,2013,12(3):547-589.

[25] SMITH C U,WILLIAMS L G. More new software performance antipatterns:Even more ways to shoot yourself in the foot[C]// Computer Measurement Group Conference. 2003:717-725.

[26] MEIER J D,VASIREDDY S,BABBARA,et al. Mackman,Improving. NET Application Performance and Scalability[OL]. http://www.bokus.com/bok/9780735618510/Improving-net-application-performance-and-scalability.

[27] CORTELLESSA V,MARTENS A,REUSSNER R,et al. A process to effectively identify 'guilty' performance antipatterns [C] // Fundamental Approaches to Software Engineering. Springer,2010:368-382.

[28] CORTELLESSA V,DI MARCO A,TRUBIANI C. An approach for modeling and detecting software performance antipatterns based on first-order logics[J]. Softw. Syst. Model.,2014,13(1):391-432.

[29] TRUBIANI C,KOZIOLEK A. Detection and solution of software performance antipatterns in palladio architectural models [J]. ACM SIGSOFT Software Engineering Notes,2011,95(9):19-30.

[30] CORTELLESSA V,FRITTELLA L. A Framework for Automated Generation of Architectural Feedback from Software Performance Analysis[C]//Formal Methods and Stochastic Models for Performance Evaluation. Springer Berlin Heidelberg,2007:171-185.

[31] BACHMANN F,BASS L,BIANCO P,et al. Using ArchE in the Classroom:One Experience[OL]. http://repsitory.cmu.edu/sei/3441.

[32] CADORET F,BORDE E,GARDOLL S,et al. Design Patterns for Rule-Based Refinement of Safety Critical Embedded Systems Models[C]// 17th International Conference on Engineering of Complex Computer Systems (ICECCS). 2012:67-76.

[33] DEB K,PRATAP A,AGARWAL S,et al. A fast and elitist multiobjective genetic algorithm:NSGA-II[J]. IEEE Trans. on Evol. Comput.,2002,6(2):182-197.