一种基于预测控制的 SaaS 系统自适应方法

熊 伟[1),3)] 李 兵[2)] 陈 军[1)] 周华昱[1)]

[1)](武汉大学软件工程国家重点实验室 & 计算机学院 武汉 430072)

[2)](武汉大学国际软件学院 武汉 430072)

[3)](湖北文理学院 湖北 襄阳 441000)

摘 要 对于部署在虚拟服务器集群上的多层云端应用系统而言,可以通过调整运行时架构以适应需求和上下文环境的变化,并保证其在动态负载下的性能.然而,由于缺乏通用的方法,如何从问题空间上的需求自适应地映射到解空间上的架构就成为了一个关键的问题.为解决这个问题,目前存在 3 种驱动方法:需求驱动模型、架构驱动模型和综合需求和架构模型的驱动方法.然而,这些方法的性能是有限的,并且忽视了时间变化规律,对横切系统多个层次的需求演化支持也不足.文中提出了一个基于预测控制的自适应方法(SAPC),它采用综合需求和架构的模型来驱动系统的自适应.该方法根据监控获取的运行时状态信息日志得到服务组件 QoS 值,学习基于小波变换的模型以准确预测服务组件的 QoS,并通过预测控制诱导需求进化或实现运行时架构的模型转换来达到系统的自适应.更进一步,其根据当前的服务组件的 QoS 以及目标 QoS 做出优化决策.在控制操作部分中,判别当前情况下是做框架调整还是需求调整.如果存在可行的运行时模型,则通过预测控制产生操作向量,继而自动生成面向方面的脚本;该脚本被执行后会重构运行时模型并生成相应实例,该实例被传递给并行计算架构(比如 MapReduce、Yarn 或 Spark 等)来执行.否则,运用训练好的模型标识出进化点,然后将进化点与初始的需求混合起来以诱导需求进化.为验证上述方法,文中以一个名为 CloudCRM 在线 SaaS 平台为基准进行了大规模的实验,该平台基于 RGPS 元模型框架,通过对开源软件 SuperCRM 进行服务化改造来获得,其支持多租户.为观察平台在不同模型驱动方法的性能表现,文中对于该平台进行了剪裁,构造了 4 种由不同模型驱动的系统(包括 Static、Req、Arch 与 SAPC),实验结果验证了文中方法的有效性.其中,在用户数为 20 时文中方法相对于其他 3 种方法在响应时间上分别降低 54%,26%与 21%,在吞吐率上提高 313%,288%与 12%,在可靠性上提高 0.40%,0.26%与 0.42%;在用户数为 200 时文中方法相对于其他 3 种方法在响应时间上分别降低 99.5%,99.2%与 20.7%,在吞吐率上提高 320%,298%与 10%,在可靠性上提高 500%,495%与 1.5%.进而文中方法使用小波变换分析相较于傅里叶变换在响应时间上降低 7.2%,在吞吐率上提高 2.4%,在可靠性上提高 0.08%.在参数对于系统的影响方面,随着目标集合的大小与插件扩展点的数目的增长,SAPC 方法都符合响应时间和吞吐量增加而可靠性下降的现象.

关键词 自适应系统;需求;预测控制;服务质量;软件即服务;云计算

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.00364

A Self-Adaptation Approach Based on Predictive Control for SaaS

XIONG Wei[1),3)] LI Bing[2)] CHEN Jun[1)] ZHOU Hua-Yu[1)]

[1)](*State Key Laboratory of Software Engineering & School of Computer, Wuhan University, Wuhan* 430072)

[2)](*International School of Software, Wuhan University, Wuhan* 430072)

[3)](*Hubei University of Arts and Science, Xiangyang, Hubei* 441000)

Abstract A self-adaptive system adapts its architecture at runtime to the changes of requirements and contexts, which assures the performance of multi-tier cloud applications deployed in virtualized server clusters according to system complexity and dynamic workloads. However, determining how to map from requirements in problem space to the architectural elements in

收稿日期:2014-07-28;在线出版日期:2015-06-14.本课题得到国家"九七三"重点基础研究发展规划项目基金(2014CB340400)、国家自然科学基金(61273216,61202032,61272111,61202031,61202048)、湖北省重大科技创新计划(2013AAA020)、中国国家科技支柱项目(2012BAH07B01)、武汉市青年科技晨光计划(2014070404010232)和软件工程国家重点实验室开放基金(SKLSE-2012-09-21)资助.**熊 伟**,男,1973 年生,博士研究生,讲师,中国计算机学会(CCF)会员,研究方向为云计算、服务计算和软件工程.E-mail:xwei9093@126.com.**李 兵**(通信作者),男,1969 年生,博士,教授,研究领域为云计算、服务计算和软件工程.E-mail:bingli@whu.edu.cn.**陈 军**,男,1967 年生,博士,教授,中国计算机学会(CCF)高级会员,研究领域为多媒体网络通信、安防应急信息处理和软件工程.**周华昱**,男,1992 年生,本科生,研究方向为云计算、数据挖掘和软件工程.

solution space is a critical research problem. There existed several approaches: requirements-driven model, architecture-based model and model which combine them. However, the performance of those approaches is limited, which ignored the impact of temporal variation and did not support requirements evolution crosscutting multiple layers of system fully. This paper proposes a SAPC (Self-adaptation Approach based on Predictive Control) approach which combines requirements and architectural model. This approach acquires QoS values via history logs of state information produced by runtime monitoring, which are utilized to learn a wavelet-transform-based model to predict the QoS of services, and induce requirements evolution or conduct architecture-based model transformations at runtime to realize self-adaptability. Furthermore, optimal design decisions were made according to current and goal QoS: we can identify which policy is needed between requirements or architectures adjustment; if we can seek feasible architectural model via predictive control based optimizer according to context changes, an aspect-oriented script are automatically generated, which can be executed to reconfigure plugin-based architecture models and build plugin instances for model transformation at runtime, as a result, those instances generate a set of tasks, which will be transmitted to the parallel computation system such as MapReduce, Yarn or Spark; Otherwise, the specific improvement points are identified and synthesized into target requirement specification with initial requirement model specification, which are utilized to induce requirements evolution. To validate our approach, large-scale experiments are conducted based on an online SaaS benchmark named as CloudCRM, which is built by transforming an open source software SuperCRM into service-oriented one based on a RGPS framework and supports multi-tenant. In order to observe the performance of different driven models, this paper constructed the platform driven by four different models, including Static, Req, Arch and SAPC, through clipping in CloudCRM. The results show that our proposed approach achieves higher performance than other approaches, where the SPAC is decreased by 54%, 26% and 21% respectively on Response time, is increased by 313%,288% and 12% respectively on Throughput, and is increased by 0.40%, 0.26% and 0.42% respectively on Reliability, compared with the other three models when the number of users is 20; The SPAC is decreased by 99.5%, 99.2% and 20.7% respectively on Response time, is increased by 320%, 298% and 10% respectively on Throughput, and is increased by 500%, 495% and 1.5% respectively on Reliability, compared with the other three models when the number of users is 200. Furthermore, Wavelet transform could achieve higher performance compared with Fourier transform, where the SPAC is decreased by 7.2% on Response time, is increased by 2.4% on Throughput, and is increased by 0.08% on Reliability. Finally, the response-time and throughput increases, however, the reliability drops when the size of goals and extension points is increased.

Keywords　self-adaptive system; requirements; predictive control; QoS; SaaS; cloud computing

1　引　言

在软件工程中,需求工程关注的是准确描述问题以及如何加快系统从需求建模到实现的过程[1].在应用程序部署之后,由于软件需求的演化而产生的需求变更会导致耗时费力的开发活动[2],并同时影响客户的体验.因此,软件系统需要具有自适应调整的能力,以快速响应需求变更和感知上下文变化.

目前,随着工业界自适应技术变得越来越流行,一些采用自适应技术的产品或原型已经开发出来.比如 IBM 已经获得了一些有意义的结果:AutoTune[3]是一个典型的自适应应用实例.当前有很多研究使

用模型驱动的方法来实现运行时的自适应[4],而不是依靠低级且易出错的手工脚本来实现.模型在运行时,系统可以收集系统状态信息,然后分析和预测系统的性能,进而依据它们实现动态适应性调整[5-6].在模型驱动中,需求模型[7-11]和架构模型[12-15]都已经被使用过.需求模型驱动的方法为简化需求元素(目标或功能)和架构元素(服务组件)之间的映射,通常会忽视架构设计部分的复杂性.而在架构模型驱动的方法中通常会假设在设计时确定需求,并且在运行时需求不再改变,因此其无法动态地适应变化的需求.

既然这两种自适应方法都有不足,那么可以考虑把它们结合起来.文献[16-17]尝试利用 Kramer 和 Magee 的架构[18]将它们组合在一起,其中架构元素被视为一组功能的接口.然而,软件架构不应该仅包含功能的集合,还应该包含一组设计决策,这是由于功能服务组件会被自适应系统组合起来并相互作用[14].这些设计决策能够根据变化调整需求,并可能横切系统的多个层次(目标或流程).

1.1 研究动机

本文使用一个名为 CloudCRM 的在线 SaaS 平台作为基准进行案例研究.该平台的业务部门负责营销策略,其通过多租户(multi-tendency)的方法来满足客户个性化定制的需求,该平台的技术部门则负责开发和维护系统.根据历史监测(如图 1),其业务负载通常表现趋势性、周期性和随机性.当系统在一年的某个特定的时间负载达到峰值,该系统由于遇到大量并发请求,性能会降低.这时若是没有适当地增加系统的负载能力,系统的响应时间会变长,业务的失败率也会增加,从而导致客户的不满.现在考虑以下两个场景.

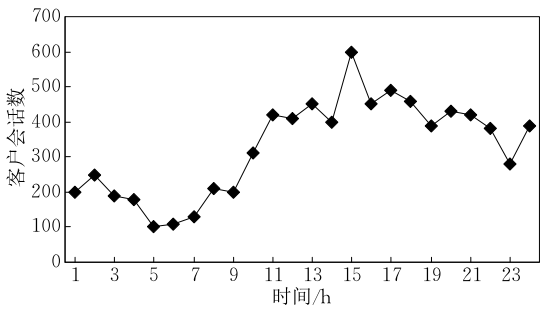

图 1 一日的客户会话数变化

(1)场景 A.技术部门发现系统的瓶颈在于订单验证过程.该过程由几个服务组件组成(包括订单信息检查、信用检查和欺诈检查).考虑到设计决策,他们决定将检查服务组件从顺序模式切换到并行模式以获取更好的性能.其通过采用支持水平扩展的并行计算架构(比如 MapReduce①、Yarn② 或 Spark③ 等)来实现运行时的自适应模型转换,以之解决性能瓶颈(其中切换过程对于业务部门是透明的).

(2)场景 B.若是技术部门不能找到可行的运行时模型来解决问题,则把问题提交给业务部门.而业务部门根据情况决定改变订单验证流程,使得该流程只涉及订单信息检查和信用检查,而跳过欺诈检查.虽然这样做可能会增加由于恶意订单所导致的风险,但是可以有效地缓解拥塞问题,这需要业务部门来权衡.上述操作将会产生新的需求(通过需求的自适应来实现),进而系统可以将其映射到相应架构模型来实现运行时的自适应.此外,系统通过为租户配置角色来实现客户的个性化定制的需求.而角色中则包括非功能属性 QoS(服务质量)[19],其主要包括可用性、响应时间、可靠性、吞吐量等.该场景同时也表明为满足用户的个性化需求,系统常常需要调整分散在不同的层次的多个元素.

1.2 贡 献

由于缺乏考虑时间变化规律且综合需求和架构的模型,目前把问题空间上的需求自适应地映射到解空间的架构上的方法还有不足之处.本文的主要贡献包括:

(1)提出一个通过预测控制的自适应方法 SAPC(Self-adaptation Approach based on Predictive Control).该方法可以通过学习基于小波变换的模型以预测系统服务组件的性能,并诱导需求进化或实现架构的模型转换以达到运行时系统的自适应.

(2)使用一个名为 CloudCRM 在线 SaaS 平台作为基准进行案例研究,其结果验证了本文方法的有效性.

本文第 2 节回顾与本文相关的研究工作;第 3 节给出问题的定义;第 4 节给出解决方案的整体构架;第 5 节给出解决方案的弹性机制;第 6 节给出支持需求模型自适应驱动的元模型及相关技术;第 7 节给出支持架构模型自适应驱动的元模型

① http://hadoop.apache.org/
② http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/
③ http://spark.apache.org/

及相关技术；第 8 节通过实验对本文提出的方法进行了评估；第 9 节总结全文并给出下一步的研究方向.

2　相关研究

本文的相关研究主要从 3 个方面展开：(1) 需求模型驱动的方法；(2) 架构模型驱动的方法；(3) 综合需求和架构模型的驱动方法.

2.1　需求模型驱动的方法

Elkhodary 等人[7]提出了一个面向特征自适应框架 FUSION，该框架能够学习自适应决策. Baresi 等人[8]提出了一个促进需求驱动的适应性框架 FLAGS. Peng 等人[9]提出了一个考虑质量动态变化，通过基于价值的反馈回路驱动需求的方法. Fu 等人[10]提出了一个通过有状态的需求监测来自我修正的社会技术系统. Souza 等人[11]提出了一个可以反映需求变化的需求进化方法.

这些方法都假定需求可以直接映射到架构元素上，但是这个假定是有问题的，因为它们忽略了架构模型的复杂性.

2.2　架构模型驱动的方法

Oreizy 等人[12]提出了一个架构驱动的运行时进化机制. Garlan 等人[13]提出一个架构驱动的自适应框架 Rainbow，该框架能对特定的需求进行定制. Floch 等人[14]提出一个支持移动自适应的框架 MADAM.

这些方法都假定当系统正在运行时，需求保持不变，且其中的大多数方法支持简单操作(如添加、删除或替换组件)，但是它们不支持横切系统多个层次的需求演化.

2.3　综合需求和架构模型的驱动方法

Kramer 等人[18]结合需求和架构提出一个三层参考模型，其中包括目标管理，变更管理及计划执行. Sykes 等人[16]使用基于模型检验的技术生成脚本，并根据脚本重构架构的模型 MBP. Tajalli 等人[17]提出一个基于计划模型驱动的自适应方法 PLASMA. 通过设计决策，动态配置架构模型来适应变化.

在综合需求和架构的模型方面，相关研究工作相对较少. 它们的性能有限，且忽视了时间变化规律，对横切系统多个层次的需求演化支持也不足.

3　问题定义

自适应是个很棘手的问题，它需要解决如何在运行时将需求映射到架构元素上并能适应需求演化. 问题的形式化描述如下：

给定 C 为一个三元组集合，其每个元素表示为 $(i,t,C_{i,t})$. 其中 i 是应用执行单元，t 是时刻，$C_{i,t}$ 是主控制节点在时间 t 对于服务组件 i 的控制操作. 控制操作包括为满足是期望的性能，而触发的运行时的模型转换或标识需求改进点操作.

给定 Y 为一个四元组集合，其每个元素表示为 $(u,i,t,Y_{u,i,t})$. 其中 u 是租户，i 是服务的第 i 个 QoS，t 是时刻，$Y_{u,i,t}$ 是租户 u 在时间 t 观察到的运行时定制服务第 i 个 QoS 值. 未来的 $Y_{u,i,(t+1)}$ 可以根据日志中作为上下文的当前时间点和前面 $k-1$ 个时间点 $Y_{u,i}=(Y_{u,i,t-k},Y_{u,i,t-k+1},\cdots,Y_{u,i,t-k+l},\cdots,Y_{u,i,t})$ 来预测.

为支持 SaaS 系统的“多租户”特征，满足多租户的个性化定制的需求. 给定需求约束 Y_{ref} 为一个三元组集合，其中每个元素表示为 $(u,i,Y_{u,i,ref})$. $Y_{u,i,ref}$ 是单个租户个性化定制的需求(租户 u 定制服务的第 i 个 QoS 的约束值).

然后根据预测的 $\{Y_{u,i,(t+1)}\}$、给定的 Y_{ref} 与三元组集合 C 生成时间 $t+1$ 时的控制操作 $\{C_{i,(t+1)}\}$. 其目的是实现系统运行时的自适应，达到需求约束下的最优稳定性能.

4　整体构架

如图 2 所示，本文提出的框架是基于 MAPE-K[20]控制回路模型的，它由监控器、分析引擎、设计决策管理器、需求演化管理器和执行引擎构成. 首先，实时监控器获取服务组件的 QoS 值并用日志记录下来，然后将其传到分析引擎. 分析引擎根据日志记录使用基于小波变换的模型预测 QoS 值. 设计决策管理器则根据预测 QoS 值和目标 QoS 值寻求满足需求约束的最佳设计决策：如果能找到可行的运行时模型，则自动生成面向方面的脚本，进而执行引擎通过执行脚本来完成运行时的模型转换；否则，设计决策管理器则标识出需求演进的具体改进点，以诱导需求进化.

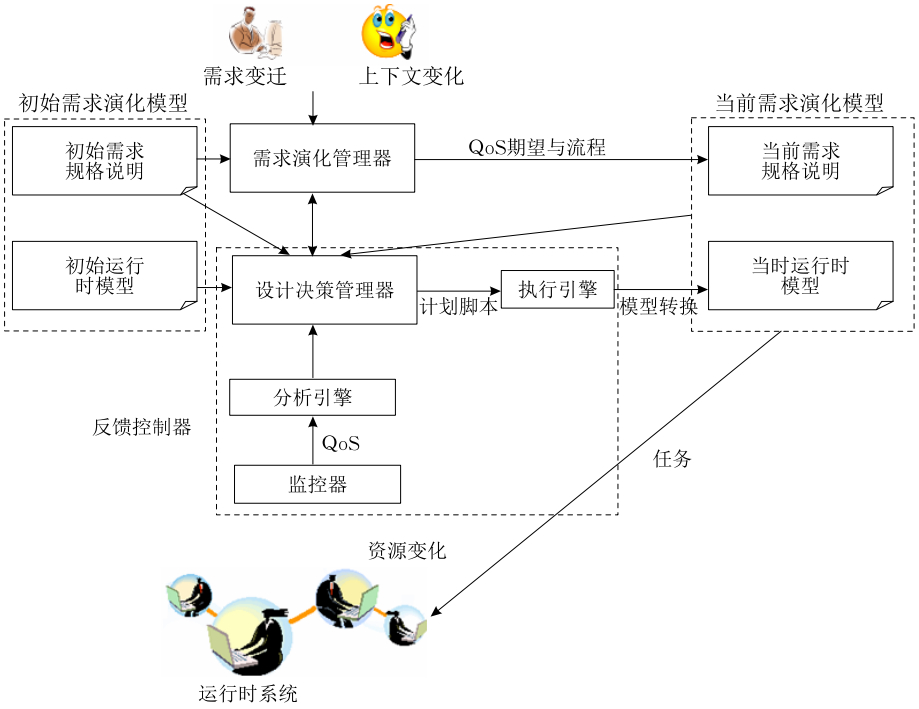

图 2 整体框架

5 弹性机制

5.1 运行时服务单元的 QoS 值的预测

傅里叶变换在声学、电信、电力系统、信号处理等诸多领域都有广泛应用,它可以将时间序列数据转换为频率序列数据以抽取时间序列的特征.虽然其能够将信号的时域特征和频域特征联系起来,但却不能把二者有机地结合起来.这是因为信号的时域波形中不包含任何频域信息,其 Fourier 谱(如式(1)所示)表现的是信号的统计特性:它是整个时间域内的积分,没有局部化分析信号的功能,存在着时域和频域的局部化矛盾.

$$F_n=\frac{1}{T}\int_{-T/2}^{T/2}f(t)\mathrm{e}^{-\mathrm{i}2\pi nt/T}\mathrm{d}t \tag{1}$$

其中,T 为函数的周期.

Milidiu 等人[21]将小波变换和混合专家模型(MEM)相结合,用来进行时间序列的预测.其可以更深刻的把握非线性问题的本质.

因此,本文选择使用小波变换分析和描述应用在各个单位时间内运行时服务单元的 QoS 值随时间的变化规律.该变换能提高系统的性能,我们将在 8.4.1 节给出具体的实验验证.

回顾在问题定义中的内容,未知的时间序列元素 $Y_{u,i,(t+1)}$ 可以根据日志信息来预测,日志信息是 $Y_{u,i}=(Y_{u,i,t-k},Y_{u,i,t-k+1},\cdots,Y_{u,i,t-k+l},\cdots,Y_{u,i,t})$.

具体地,可将 $Y_{u,i,(t+1)}$ 展开为

$$Y_{u,i,(t+1)}=\sum_{j=1}^{l}\omega_{ji}h(j),\ j=1,2,\cdots,m \tag{2}$$

其中,ω_{ji} 是隐含层到输出层权值,$h(j)$ 是第 j 个隐含层节点的输出,l 是隐含层节点数,m 是输出层层节点数.更进一步,$h(j)$ 定义为

$$h(j)=h_j\left(\frac{\sum_{l=1}^{k}\omega_{lj}Y_{u,i,t-k+l}-b_j}{a_j}\right),\ j=1,2,\cdots,l \tag{3}$$

其中,ω_{lj} 是输入层和隐含层的连接权值,b_j 是小波基函数 h_j 的平移因子,a_j 是小波基函数 h_j 的伸缩因子,h_j 是小波基函数,本文采用了 Morlet 母小波基函数,其定义为

$$y=\cos(1.75x)\mathrm{e}^{x^2/2} \tag{4}$$

小波网络权值参数修正公式如下:

(1)计算预测误差

$$e=Y_{ref}-Y \tag{5}$$

其中,Y_{ref} 是期望输出,Y 是预测输出.

(2)根据误差修正相关权值和系数

$$\omega_{lj}^{(t+1)}=\omega_{lj}^{t}+\Delta\omega_{lj}^{(t+1)} \tag{6}$$

$$a_j^{(t+1)}=a_j^{t}+\Delta a_j^{(t+1)} \tag{7}$$

$$b_j^{(t+1)}=b_j^{t}+\Delta b_j^{(t+1)} \tag{8}$$

其中,梯度公式如下:

$$\Delta\omega_{lj}^{(t+1)}=-\eta\frac{\partial e}{\partial\omega_{lj}^{t}} \tag{9}$$

$$\Delta a_{j}^{(t+1)}=-\eta\frac{\partial e}{\partial a_{j}^{t}} \tag{10}$$

$$\Delta b_{j}^{(t+1)}=-\eta\frac{\partial e}{\partial b_{j}^{t}} \tag{11}$$

其中，η 是学习速率.

算法的时间复杂度为 $O(nkm)$，其中 n 是迭代次数，k 是输入层节点的数目，m 是隐含层节点的数目.

5.2　基于 QoS 的设计决策

回顾问题定义中的内容，我们将根据预测的 $Y_{(t+1)}$、给定的 Y_{ref} 与三元组集合 C 生成时间 $t+1$ 时的控制操作向量 $\boldsymbol{C}_{(t+1)}$. 控制操作包括为满足是期望性能 Y_{ref} 而触发的运行时的模型转换或标识需求改进点操作，具体实现见算法 1. 其中，$Y_t=\{Y_{u,i,t}\}$ 是所有租户在时间 t 观察到的运行时运行时定制服务 QoS 值的集合，$Y_{(t+1)}$ 是对所有租户预测到的在时间 $t+1$ 的定制服务 QoS 值的集合，$Y_{ref}=\{Y_{u,i,ref}\}$ 是期望性能(租户定制服务的 QoS 集合)，$\boldsymbol{C}_{(t+1)}=\{C_{i,(t+1)}\}$ 是时间 $t+1$ 时对服务组件 i 的控制.

算法 1.　设计决策算法.

输入：Y_t，$Y_{(t+1)}$，Y_{ref}

输出：$\boldsymbol{C}_{(t+1)}$

初始化：

train classification prediction model

train labeling improvement points of requirements model

IF $classification\text{-}prediction(Y_t,Y_{(t+1)},Y_{ref})=$ requirements THEN

　$\boldsymbol{C}_{(t+1)}=label\text{-}improvement\text{-}points(Y_t,Y_{(t+1)},Y_{ref})$

ELSE

　$\boldsymbol{C}_{(t+1)}=architecture\text{-}transformations(Y_t,Y_{(t+1)},Y_{ref})$

END

RETURN $\boldsymbol{C}_{(t+1)}$

操作集合. 算法主要包括两大部分：初始化部分和控制操作部分.

在初始化部分中，我们利用历史记录来学习 SVM(支持向量机)模型[22]，来判别当前情况下是否存在可行的运行时模型. 更进一步，如果不存在可行的运行时模型，我们同样利用历史记录来学习 SVM 模型，只不过模型的输出变成了对改进点的标注，用以诱导需求进化，而该标注是基于概念和本体的.

在控制操作部分中，在当前需求的约束下，如果不存在可行的运行时模型，则运用训练好的模型诱导需求进化. 否则，通过预测控制产生操作向量 $\boldsymbol{C}_{(t+1)}$. 而为寻求控制操作向量 $\boldsymbol{C}_{(t+1)}$，需要控制器用最小幅度的调控让系统满足多租户个性化定制需求. 为此，我们定义目标函数

$$J(t)=q\left\|\sum_{u,i}(Y_{u,i,(t+1)}-Y_{u,i,ref})\right\|^2\times\left\|W\cdot(\boldsymbol{C}_{(t+1)}-\boldsymbol{C}_t)\right\|^2 \tag{12}$$

并使得其值为最小. 其中，q 是稳定因子，$\boldsymbol{C}_t$ 是当前控制操作向量，Y_{ref} 是期望输出，$W=\{w_i\}$ 是服务组件的权重，$\boldsymbol{C}_{(t+1)}=\{C_{i,(t+1)}\}$.

我们采用梯度下降法来预测优化的控制操作向量 $\boldsymbol{C}_{(t+1)}$，梯度公式为

$$\frac{\partial J}{\partial\boldsymbol{C}_{(t+1)}}=2q\times\left\|\sum_{u}(Y_{u,(t+1)}-Y_{u,ref})\right\|^2\times\left\|W(\boldsymbol{C}_{(t+1)}-\boldsymbol{C}_t)\right\| \tag{13}$$

算法的时间复杂度为 $O(nl)$，其中 n 是迭代次数，l 是服务组件单元的数目.

更进一步，由式(13)得到的控制操作向量 $\{C_{i,(t+1)}\}$ 只是反映了操作的强弱，并没有物理实际意义. 因此，需要将控制操作向量 $\boldsymbol{C}_{(t+1)}$ 映射到系统的实际运作的弹性机制上，我们定义映射函数 $f(x)$ 如下：

$$f(x)=\begin{cases}1, & x<w_1'\\ 2, & w_1'\le x\end{cases} \tag{14}$$

其中，数字{1,2}分别表示不同的计算模型(串行，并行)，w_1' 是通过一系列实验得到的经验值. 最后，我们得到

$$\boldsymbol{C}_{(t+1)}=f(\boldsymbol{C}_{(t+1)}) \tag{15}$$

然后返回.

6　需求的自适应

根据 5.2 节的设计决策产生的输出(对改进点的标注)，我们可以定位变更需求对初始需求的演化生长点，更进一步捕获需求变更的影响，继而采用面向方面描述变更需求，完成需求的自适应.

6.1　支持演化的需求元模型

为支持需求的演化，本文扩展了 RGPS 需求元模型框架[15]以构建需求模型库(如图 3). 通过添加标签、概念、本体、方面和词汇等元素，以提高标签的语义互操作性. 通过标签的方式来识别需求的改进点，以诱导需求进化.

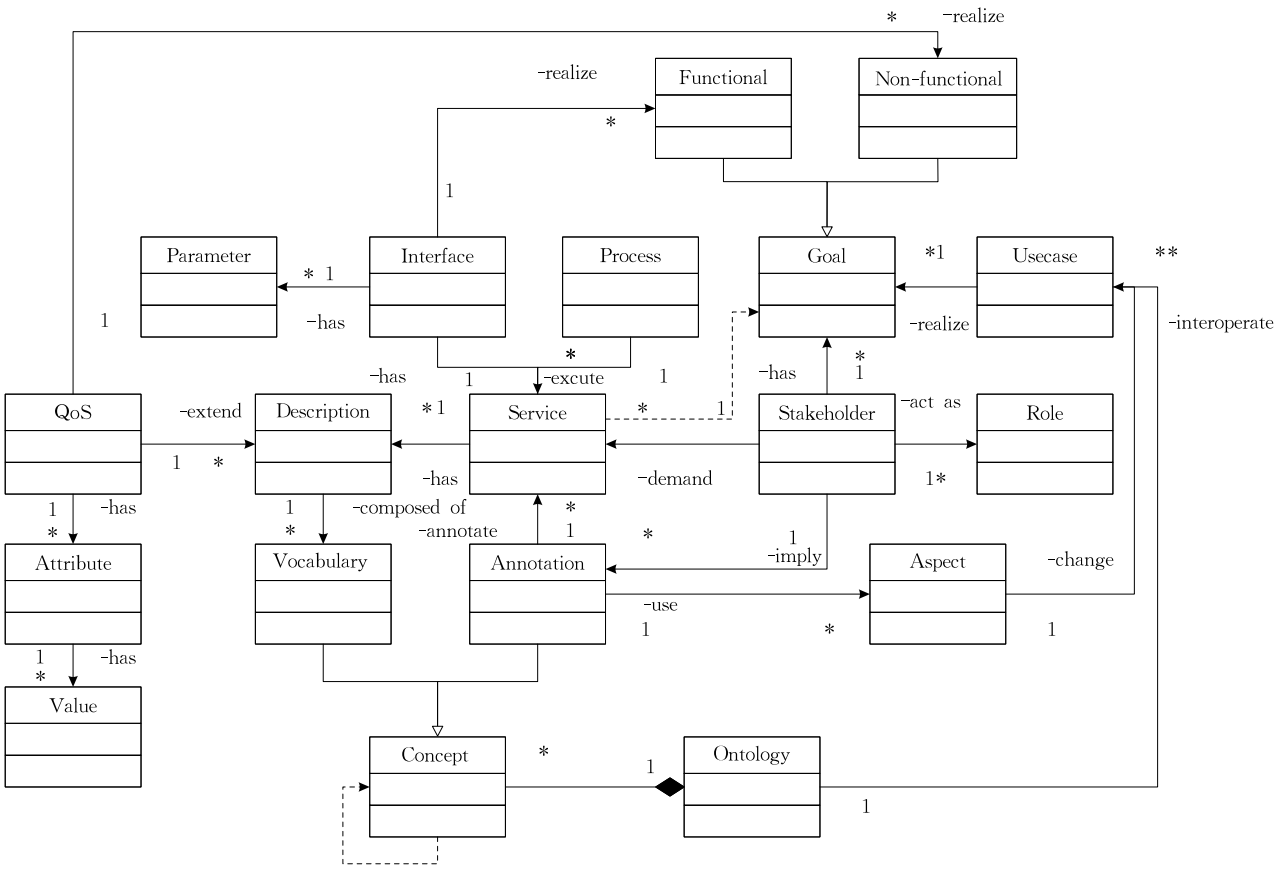

图 3 基于 RGPS 支持演化的需求元模型

6.2 面向方面的需求演化

需求演化建模要解决的根本问题是:如何根据变更的需求将原始需求模型演变为满足新的需求的需求模型.

6.2.1 基本过程

具体而言,分为 5 个阶段完成:

(1)根据 5.2 节的工作,分析与获取变更需求(主要是通过标注来完成的).

(2)本体匹配.根据第一阶段得到的标注,在基于 RGPS 的演化需求模型库中搜索目标或过程元素,作为初始需求模型.

(3)利用面向方面的需求演化建模分析方法,针对初始需求模型,找到合适的演化改进点.

(4)基于演化改进点,捕获需求变更的影响(在 6.2.2 节中详述).

(5)业务部门以 OWL-S^{A} 作为描述语言(在 6.2.3 节详述),根据上述分析,建立满足演化需要的新的需求规格脚本,为系统的即时演化提供支持.

6.2.2 需求演化的传播

根据基于RGPS支持演化的需求元模型(如图 3),需求项之间以及需求项与其他系统元素间存在着关联关系.由前面分析获取的改进点,其可以作为演化生长点将需求变更传播出去.

为了捕获被影响到的系统元素,假设我们从 RGPS 需求元模型中获取了需求项间的关联矩阵 $\boldsymbol{M}_1$,需求项与目标项间的关联矩阵 $\boldsymbol{M}_2$,目标项与过程项间的关联矩阵 $\boldsymbol{M}_3$.由于需求项间的传递依赖,我们需计算矩阵 $\boldsymbol{M}_1$ 的闭包 M_1^*.

当变更需求向量为 $\boldsymbol{r}$ 时,我们可以通过

$$\boldsymbol{o}=\boldsymbol{r}\cdot M_1^*\cdot \boldsymbol{M}_2 \tag{16}$$

计算受影响的目标元素向量 $\boldsymbol{o}$,通过

$$\boldsymbol{p}=\boldsymbol{r}\cdot M_1^*\cdot \boldsymbol{M}_2\cdot \boldsymbol{M}_3 \tag{17}$$

计算受影响的过程元素向量 $\boldsymbol{p}$,最终我们获取了变更需求的集合.

6.2.3 OWL-S^{A} 描述语言

描述语言 OWL-S^{A} 基于 OWL-S[23] 的,其采用切入点-通知模型(pointcut-advice model).其中,A 表示方面(Aspect),其代表的是演化的需求规格,它包括一个或多个切入点-通知对(pointcut-advice pairs).OWL-S^{A} 的元模型如图 4 所示.

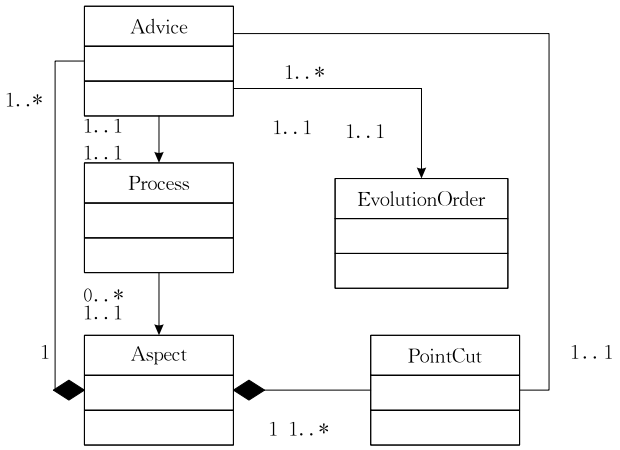

图 4　面向方面的演化元模型

切入点-通知对中的切入点用于描述演化的位置. 其中定义两种类型的切入点：

(1)控制制导切入点，即原过程模型中的某节点或其附近的位置. 其描述形式如下：

//process:CompositeProcess[@rdf:ID="CP1"]//process:perform[@rdf:ID="Step1"].

(2)数据制导切入点，即切入的过程将改变原过程模型的数据，比如输入、输出参数，其描述形式如下：

//process:CompositeProcess[@rdf:ID="CP1"]//process:hasInput[@rdf:ID="I1"].

切入点-通知对中的通知用于描述演化的内容. 其包括：切入点附近插入、删除和修改过程，或是在过程中增加的输入输出参数，通常情况下是由控制结构 Perform 触发.

通知与切入点之间的关系用 advice 的一个属性 evolutionOrder 来表示，其值有 4 种：Before(前面)、after(后面)、parallelTo(并行)和 Around(替换).

回到 1.1 节的场景 B 中，假设负载达到顶峰后开始下降. 受需求演化的诱导，被跳过的欺诈检查将被业务部门恢复. 该需求演化脚本如图 5 所示.

```
〈Aspect name="FraudChecking"〉
 〈pointcut name="CreditChecking"〉
  process:CompositeProcess[@rdf:ID="OrderVerificationProcess"]
  process:perform[@rdf:ID="CreditChecking"]
 〈/pointcut〉
 〈advice evolutionOrder="after"〉
  〈Perform rdf:ID="FraudCheckingPerformed"〉
   〈process rdf:resource="&aux;#FraudChecking"/〉
   〈hasDataFrom〉
    〈InputBinding〉
     〈theParam rdf:resource="&aux,#Order"/〉
     〈ValueSource〉
      〈ValueOf〉
       〈theVar rdf:resource="#Order"/〉
       〈fromProcess "rdf:resource=#http://www.daml.org/
            services/owl-s/1.1/Process.owl#TheParentPerform" 〉
      〈/VauleOf〉
     〈/ValueSource〉
    〈/InputBinding〉
   〈/hasDataFrom〉
   〈Produce〉
    〈ProducedBinding〉
     〈OutputBinding〉
      〈theParam rdf:resource="#VerificationResults"/〉
      〈valueSource〉
       〈ValueOf〉
        〈theVar rdf:resource="#VerificationResults"/〉
        〈fromProcess rdf:resource="rdf:resource=#http://www.daml.org/services/
                                   owl-s/1.1/Process.owl#TheParentPerform"〉
       〈/ValueOf〉
      〈/valueSource〉
     〈/OutputBinding〉
    〈/ProducedBinding〉
   〈/Produce〉
  〈/Perform〉
 〈/advice〉
〈/Aspect〉
```

图 5　需求演化脚本

7　架构模型的自适应

根据 5.2 节的设计决策的输出(待触发的运行时的模型转换)，系统可以自动产生计划脚本. 脚本紧接着被传到执行引擎，而引擎则使用反射的方法，将变更对应的插件织入到系统中，从而完成架构模型的自适应.

7.1 基于插件架构的元模型

系统的弹性机制是通过切换在插件扩展点上的支持不同计算模型的插件来完成的.其计算模型包括串行和并行.

本文利用插件架构形成支持不同计算模型统一框架.下面给出了基于插件架构的元模型,如图 6 所示.

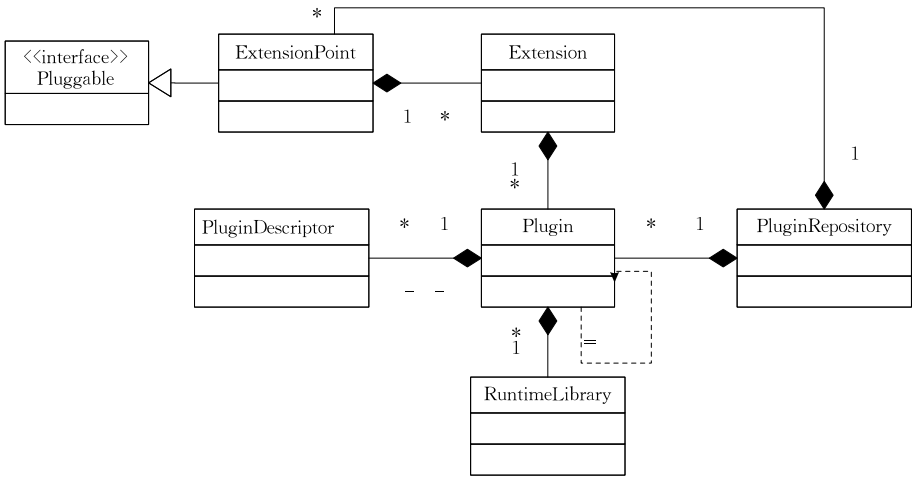

图 6 基于插件架构的元模型

其中,扩展点接口(Pluggable)用于定义扩展点.

扩展点(Extension point)是系统中可以被再次扩展的接口,它可以使得执行过程变得可插入,可以任意变化.扩展点实现扩展点接口.

扩展(Extension)是针对扩展点的实现,每个扩展都可以拥有自己的额外属性,用以区分同一个扩展点不同的实现.

插件(Plugin)包含了插件依赖关系,运行时库和插件描述符.

插件描述符(PluginDescriptor)包含了插件的名称 id 版本等信息.

插件工厂(PluginRepository)用来产生插件实例.

下面给出一个基于插件架构的实例(如图 7).

```
〈plugin id="order-verification-basic" version="1.0.0"
 provider-name="CloudCRM.org"〉
  〈runtime〉
   〈library name="order-verification.jar"〉
    〈export name="*"/〉
   〈/library〉
  〈/runtime〉
  〈requires〉
   〈import plugin="order-extensionpoints"/〉
  〈/requires〉
  〈extension id="org.CloudCRM.order.verification.basic"
              name="CloudCRM Basic order verification"
              point="org.CloudCRM.order.verification.basic"〉
   〈implementation id="BasicOrderVerification"
       class="org.CloudCRM.order.verification.basic.parallel"/〉
  〈/extension〉
〈/plugin〉
```

图 7 基于插件架构的实例

7.2 计划脚本的生成

描述语言 OWL-S[A] 不仅能用于支持需求演化,还能够用到运行时的模型转换中.我们将其扩充以支持变更操作:增加或删除插件扩展点、增加或删除插件、关联插件扩展点与插件,替换插件等.根据 4.2 节的设计决策的输出(待触发的运行时的模型转换),系统使用替换插件操作:使用 Xpath 来定位文档树中的插件扩展点,修改其下的插件节点,从而产生新的计划脚本.

回到 1.1 节的场景 A 中,系统能够自动生成计划脚本(如图 8).

```
〈Aspect name="FraudChecking"〉
 〈pointcut name="CreditChecking"〉
  process:CompositeProcess[@rdf:ID="OrderVerificationProcess"]
  process:perform[@rdf:ID="CreditChecking"]
  process:ExtensionPoint[@rdf:ID="CreditCheckingArchitecture"]
 〈/pointcut〉
 〈advice evolutionOrder="after"〉
  〈Perform rdf:ID="FraudCheckingPerformed"〉
   〈process rdf:resource="&aux;#FraudChecking"/〉
    〈Extension point rdf:ID="FraudCheckingPerformed"〉
                〈Architecture Model〉
                〈theVar rdf:resource="#parallel"/〉
                〈/Architecture Model〉
   〈/Extension point〉
  〈/Perform〉
〈/Aspect〉
```

图 8 生成的计划脚本

该脚本由执行引擎执行,切换在插件扩展点上的支持不同计算模型的插件.

7.3 计划脚本的执行

执行引擎通过执行计划脚本重新配置运行系统来实现运行时的模型转换.详细过程如下:

系统利用载入内存并已经初始化过的插件工厂

生成相关插件实例：通过插件的 id 定位并读取计划脚本中相应的插件描述符配置项，进而使用反射机制生成插件实例.

紧接着我们通过插件实例调用实现支持不同计算模型的方法，进而生成一组任务，并将其传到并行计算系统上去执行.

计划脚本的执行的代码如图 9 所示.

```
public class parallel extends verification. basic{
Configuration conf=null;
PluginRepository repo=null;
PluginDescriptor d=null;
ClassLoader cl=null;
Static Class clazz=null;
Static Method m=null;
public parallel(){
conf=Configration. create();
repo=new PluginRepository(conf);
d=repo. getPluginDescriptor("BasicOrderVerification");
cl=d. getClassLoader;
try{
clazz=Class. forName("org. CloudCRM. order. verification. basic.
parallel",true,cl);
}catch(Exception e){
System. err. println("can not load the class'parallel'"+e. get-
Message());
return;}
}
try{
m = clazz. getMethod ("verify", new Class[]{Class. forName
("org. CloudCRM. order. verification. basic. parallel")});
}catch(Exception e){
System. err. println("can not load the class'parallel'"+e. get-
Message());
return;}
}
try{
result=m. invoke(clazz. newInstance());
}catch(Exception e){
System. err. println("can not load the class'parallel'"+e. get-
Message());
return;}
}
}
```

图 9　计划脚本的执行代码

8　实验评估

8.1　实验设计

本文使用了综合需求模型和架构模型的驱动方法，扩展了课题组现有的 SaaS 平台（CloudCRM 客户关系管理云服务平台）. 该平台基于 RGPS 元模型框架，通过对开源软件 SuperCRM 进行服务化改造来构建，其支持多租户. 该平台的配置环境如下：

（1）该集群包括 18 个节点，每个节点有 64 位 8 核CPU 和 16 GB RAM.

（2）所有节点都用 CentOS 6. 2，Java 1. 6. 022.

（3）数据库采用 mongodb2. 0. 8.

为观察平台在不同模型驱动方法的性能表现，本文对于该平台进行了剪裁，构造了 4 种由不同模型驱动的系统，具体见 8. 2 节.

为观察平台在不同状况下的性能表现，本文可以设置或调整如下参数：模拟并发用户的数目、需求目标集合的大小（goal size）、插件扩展点的数目（extension points size）. 当前实验参数设置如下：模拟并发用户的数目设置为 20 或 200，目标集合的大小设置为 21，插件扩展点的数目设置为 8.

8.2　比较方法

本文对比了以下 4 种不同模型驱动方法：

（1）不支持自适应的方法 Static.

（2）基于需求模型的自适应方法 Req.

（3）基于架构模型的自适应方法 Arch.

（4）综合需求和架构模型的自适应方法 SAPC.

8.3　实验结果分析

为了避免实验结果的随机性，我们重复该实验 50 次，对获得的性能指标求均值. 实验结果如 表 1，其中性能指标包括全局响应时间（Response-time）、吞吐量（Throughput）和可靠性（Reliability）. 详细分析如下：

（1）从表中可知，我们的方法与其他方法相比获得了更好的性能.

（2）当模拟并发用户的数目从 20 增加到 200 时，变得响应时间变长，吞吐量更高，可靠性则降低了.

（3）当模拟并发用户的数目从 20 增加到 200 时，不同的方法会有不同的表现. 例如，Static 方法的响应时间从 1473. 92 ms 变化到 157644. 33 ms，而 SAPC 方法的响应时间则从 676. 35 ms 变化到 731. 43 ms. 这表明，综合了需求和架构自适应的方法起了作用，达到更好的性能.

表 1　不同模型驱动方法的性能比较

方法	用户数=20			用户数=200		
	响应时间/ms	吞吐率/Mbps	可靠性/%	响应时间/ms	吞吐率/Mbps	可靠性/%
Static	1473. 92	8. 32	89. 95	157644. 33	844	13. 95
Req	925. 66	8. 86	90. 07	91183. 71	891	14. 07
Arch	857. 64	30. 72	89. 93	922. 51	3213	82. 55
SAPC	676. 35	34. 44	90. 31	731. 43	3551	83. 76

8.4 进一步讨论

8.4.1 时间序列预测采用不同变换对系统性能的影响

在这个实验中,参数设置为:模拟并发用户的数目设置为 200,其他同 8.1 节.从表 2 可知,由于小波(Wavelet)变换分析相较于傅里叶(Fourier)变换有较好的预测精度,在系统中采用小波变换分析获得了较好的性能.

表 2 不同变换的性能比较

方法	用户数=200		
	响应时间/ms	吞吐率/Mbps	可靠性/%
Fourier	788.25	3465	83.69
Wavelet	731.43	3551	83.76

8.4.2 综合需求和架构模型不同驱动方法的性能比较

目前在综合需求和架构模型方面具有代表性的模型包括 Kramer 等人[18]的模型、Sykes 的模型 MBP[16]、Tajalli 等人[17]的模型 PLASMA.下面我们通过实验来验证同上述几种方法相比,我们的方法 SAPC 的有效性.

在这个实验中,参数设置为:模拟并发用户的数目设置为 200,其他同 8.1 节.从表 3 可知,由于考虑了时间序列规律,我们的方法与其他综合需求和架构模型的驱动方法相比获得了最好的性能.

表 3 综合需求和架构模型不同驱动方法性能比较

方法	用户数=200		
	响应时间/ms	吞吐率/Mbps	可靠性/%
Kramer's model	920.66	3204	81.95
MBP	897.37	3216	82.07
PLASMA	821.23	3375	82.54
SAPC	731.43	3551	83.76

8.4.3 目标集合的大小对性能的影响

在这个实验中,我们逐步改变目标集合的大小从 3 到 24,每步为 3.其他参数设置为:模拟并发用户的数目设置为 20 或 200,插件扩展点的数目设置为 8.

图 10(a)~(c)表明的是当并发用户的数量是 20 时的影响,而图 10(d)~(f)表明的是当并发用户的数量是 200 时的影响.图 10 表明:在不同大小的目标集合下,SAPC 方法都符合响应时间和吞吐量增加而可靠性下降现象.当模拟并发用户的数目扩大时,对于不同性能指标影响幅度不同(吞吐量幅度最大而可靠性幅度最小).

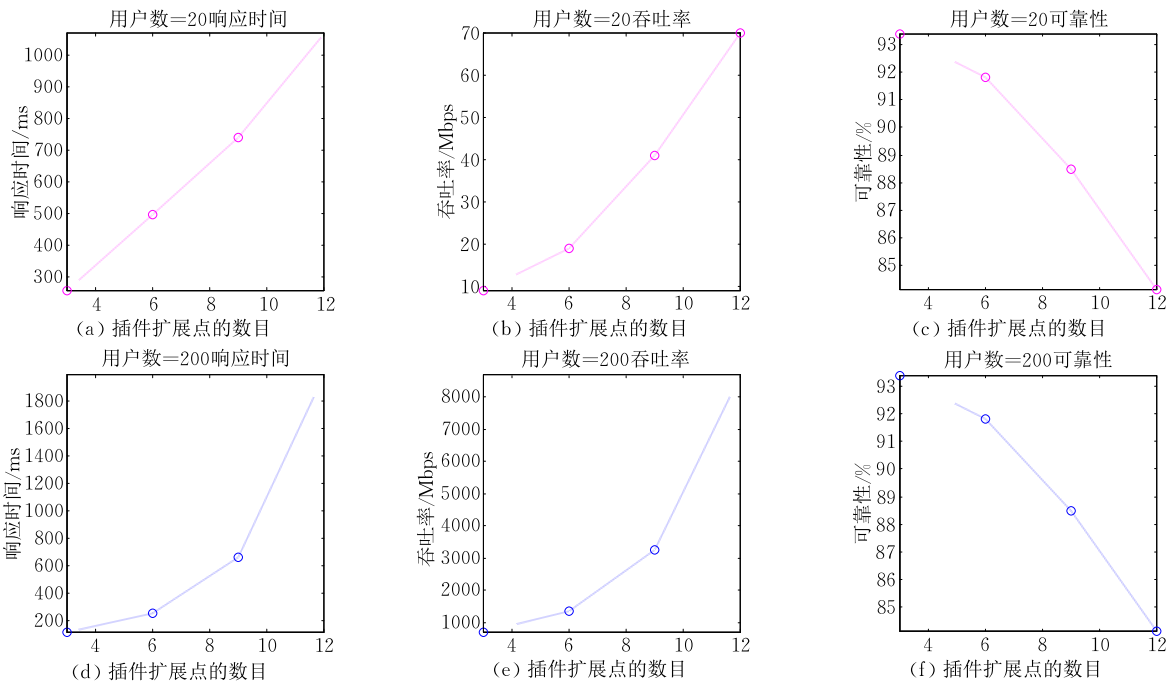

图 10 目标集合的大小对性能的影响

8.4.4 插件扩展点的数目对性能的影响

在这个实验中,我们逐步改变插件扩展点的数目从 3 到 12,每步为 3.其他参数设置为:模拟并发用户的数目设置为 20 或 200,目标集合的大小设置为 21.

图 11 表明:在不同插件扩展点的数目下,SAPC 方法对于性能的影响同 8.4.3 节.

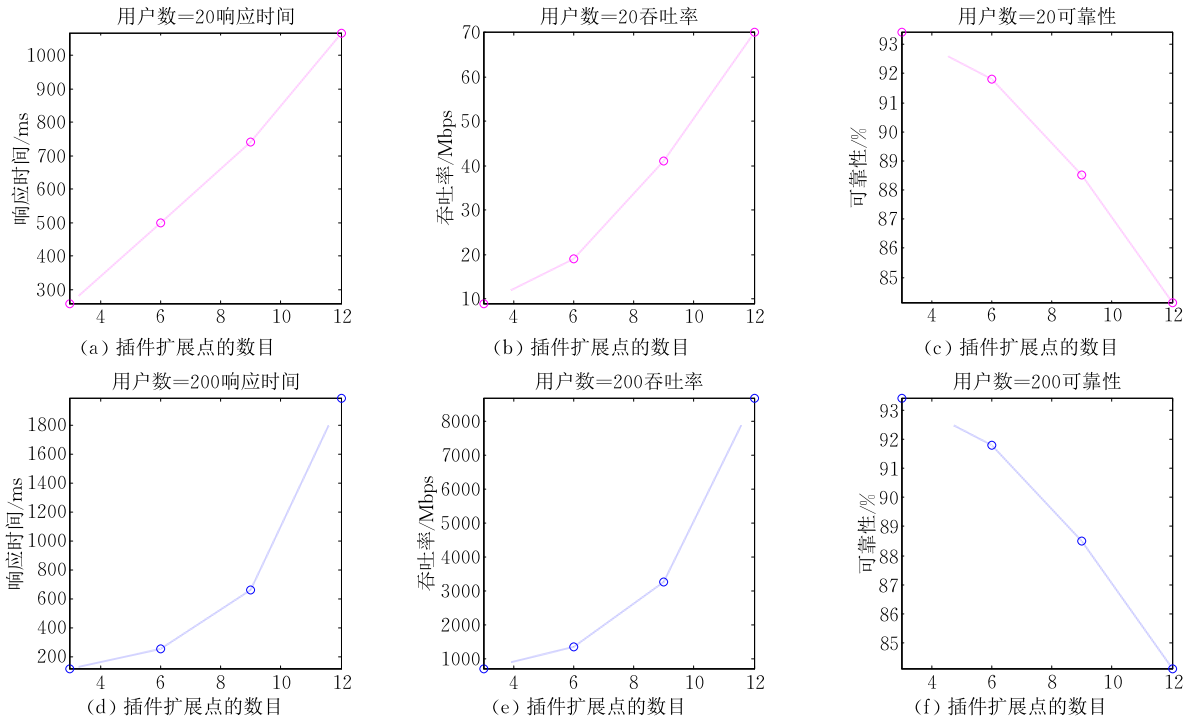

图 11　插件扩展点的数目对性能的影响

9　结　论

本文提出了一个通过预测控制的自适应方法(SAPC),其采用了综合需求和架构的模型来驱动系统的自适应.通过分析日志信息,学习基于小波变换的模型以预测未来时刻服务单元的 QoS;通过基于预测控制产生设计决策,诱导需求进化或进行运行时的模型转换.大量实验的结果显示我们的方法的有效性.在未来,我们计划更进一步研究基于预测控制模型,在支持需求演化的元模型中加入验证机制,提取更多的状态信息以改进方法,以及从用户满意度等更多的方面评估该方法.

参　考　文　献

[1] van Vliet H. Software Engineering: Principles and Practice. 3rd Edition. Hoboken, USA: Wiley, John Wiley & Sons Inc, 2008

[2] Nuseibeh B. Weaving together requirements and architectures. Computer, 2001, 34(3): 115-117

[3] Diao Y, Hellerstein J L, Parekh S, et al. Managing Web server performance with AutoTune agents. IBM Systems Journal, 2003, 42(1): 136-149

[4] France R, Rumpe B. Model-driven development of complex software: A research roadmap//Proceedings of the Future of Software Engineering. Washington, USA, 2007: 37-54

[5] Blair G S, Bencomo N, France R B. Models@run. time. Computer, 2009, 42(10): 22-27

[6] Morin B, Barais O, Jezequel J M, et al. Models@run. time to support dynamic adaptation. Computer, 2009, 42(10): 44-51

[7] Elkhodary A, Esfahani N, Malek S. FUSION: A framework for engineering self-tuning self-adaptive software systems//Proceedings of the 18th Foundations of Software Engineering. New York, USA, 2010: 7-16

[8] Baresi L, Pasquale L, Spoletini P. Fuzzy goals for requirements-driven adaptation//Proceedings of the 18th Requirements Engineering Conference. Sydney, Austrilia, 2010: 125-134

[9] Peng X, Chen B, Yu Y, Zhao W. Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. Journal of Systems and Software, 2012, 85(12): 2707-2719

[10] Fu L, Peng X, Yu Y, Mylopoulos J, Zhao W. Stateful requirements monitoring for self-repairing socio-technical systems//Proceedings of the 20th IEEE Conference on Requirements Engineering, Chicago, USA, 2012: 121-130

[11] Souza V, Lapouchnian A, Angelopoulos K, Mylopoulos J. Requirements-driven software evolution. Computer Science-Research and Development, 2013, 28(4): 311-329

[12] Oreizy P, Medvidovic N, Taylor R N. Architecture-based runtime software evolution//Proceedings of the 20th International Conference on Software Engineering. Washington, USA, 1998: 177-186

[13] Garlan D, Cheng S W, Huang A C, et al. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 2004, 37(10): 46-54

[14] Floch J, Hallsteinsen S, Stav E, et al. Using architecture models for runtime adaptability. IEEE Software, 2006, 23(2): 62-70

[15] Wang J, He K, Gong P, et al. RGPS: A unified require-

ments meta-modeling frame for networked software// Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames, New York, USA, 2008: 29-35

[16] Sykes D, Heaven W, Magee J, Kramer J. From goals to components: A combined approach to self-management// Proceedings of the 2008 SEAMS, New York, USA, 2008: 1-8

[17] Tajalli H, Garcia J, Edwards G, Medvidovic N. PLASMA: A plan-based layered architecture for software model-driven adaptation//Proceedings of the ASE. New York, USA, 2010: 467-476

[18] Kramer J, Magee J. Self-managed systems: An architectural challenge//Proceedings of the Future of Software Engineering. Minneapolis, MN, 2007: 259-268

[19] Papazoglou M P. Service-oriented computing: Concepts, characteristics and directions//Proceedings of the 4th International Conference Web Information Systems Engineering. Rome, Italy, 2003: 3-12

[20] Kephart J O, Chess D M. The vision of autonomic computing. IEEE Computer, 2003, 36(1): 41-50

[21] Milidiu R L, Machado R J, Renteria R P. Time series forecasting through wavelets transformation and a mixture of expert models. Neurocomputing, 1999, 28(1): 145-156

[22] Joachims T. Making large scale SVM learning practical. Universität Dortmund, DE, Report: 24, 1999

[23] Wang H, Payne T, Gibbins N, et al. Formal specification of OWL-S with Object-Z: The dynamic aspect//Proceedings of the Web Information Systems Engineering (WISE 2007). Nancy, France, 2007: 237-248

XIONG Wei, born in 1973, Ph. D. candidate. His research interests include cloud computing, service computing and software engineering.

Li Bing, born in 1969, Ph. D., professor. His research interests include cloud computing, service computing and software engineering.

CHEN Jun, born in 1967, Ph. D., professor. His research interests include multimedia network communications, security emergency information processing and software engineering.

ZHOU Hua-Yu, undergraduate student. His research interests include cloud computing, data mining and software engineering.

Background

In software engineering, requirement engineering pursues to describe the problem accurately and determine how to speed up from the requirements modeling to the system implementation process. After an application is deployed, requirement changes to the issue lead to a time-consuming development activity during the evolution of software requirements. However, software systems are required to respond quickly to requirements and contexts using self-adaptation according to the perception of customers' satisfaction.

Requirements-driven approaches usually simplify the mapping between the requirements elements (e. g., goals or features) and the architectural elements (e. g., components), which neglect the complexity of architectural design. Architecture-based approaches assume that requirements are well-understood at design time and unchanged at runtime. Thus, they are unable to support dynamic adaptations to changing requirements specifications. Since these two kinds of self-adaptation approaches may have a role to play, they should be combined.

This paper proposes a SAPC approach which combines requirements and architectural model. It can learn a wavelet-transform-based model to predict the QoS via the history logs of state information produced by runtime monitoring. Optimal design decisions were made to reconfigure plugin-based architecture models using self-adaptation of model transformations at runtime via predictive control based optimizer or analyze the specific improvement points, which are utilized to induce requirements evolution.

To validate our approach, large-scale experiments are conducted based on an SaaS benchmark named as Cloud-CRM. We built it by transforming an open source software named as SuperCRM into a service-oriented one and clipped it by four different models to observe the performance of them. The results show that our proposed approach achieves higher performance than other approaches. Furthermore, Wavelet transform could achieve higher performance compared with Fourier transform, and the response-time and throughput increases, however, the reliability drops when the size of goals and extension points is increased.

The work described in this study was fully supported by the National Program on Key Basic Research Project (973 Program) of China under Grant (No. 2014CB340400), the National Natural Science Foundation of China under Grant (Nos. 61273216, 61202032, 61202031 and 61202048), the Science and Technology Innovation Program of Hubei Province under Grant (No. 2013AAA020), the National Science & Technology Pillar Program of China under Grant (No. 2012BAH07B01), the Youth Chenguang Project of Science and Technology of Wuhan City in China (No. 2014070404010232) and Open-Fund in State Key Laboratory of Software Engineering, Wuhan University (No. SKLSE-2012-09-21).