基于JBPM的服务动态编排及迁移方案

王 勇[1] 常静波[1] 强保华[1] 王玉峰[2] 张学庆[2]
(桂林电子科技大学计算机科学与工程学院 桂林 541004)[1]
(中国电子科技集团公司第五十四研究所 石家庄 050081)[2]

摘 要 为了适应复杂环境下动态多变的业务需求,对以服务为核心的流程动态编排和再造提供支持,提出了基于JBPM工作流的流程动态编排与迁移方案,该方案分析了目前工作流在流程编排方面的局限性;结合柔性工作流的特点,给出了流程动态编排的模型,并对流程变更所产生的4类操作做了形式化的描述,通过对流程变更引发的流程进行迁移继而实现接续执行这一问题的研究,提出一种流程迁移算法。最后通过实例和性能测试工具验证了算法的可行性和高效性。

关键词 动态流程编排,流程迁移,柔性工作流,流程再造
中图法分类号 TP391.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.7.033

Dynamic Service Scheduling and Migration Scenarios Based on JBPM

WANG Yong[1] CHANG Jing-bo[1] QIANG Bao-hua[1] WANG Yu-feng[2] ZHANG Xue-qing[2]
(College of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)[1]
(The 54th Research Institute, China Electronics Technology Group Corporation, Shijiazhuang 050081, China)[2]

Abstract In order to adapt to dynamically change business needs in complex environment and to provide support for dynamical process scheduling and recycling that centralize service, a dynamical process scheduling and migration schema based on JBPM workflow was proposed. This schema analyzes the limitations of workflow in terms of the current process scheduling, displays the dynamical process orchestration model combining with the characteristics of flexible workflow and formally describes the four operations produced by process changes. Through the study on the issues triggered by the migration process and continual performing, a process migration algorithm was proposed. Finally, instances and performance testing tools were employed to verify the feasibility and efficiency of the algorithm.

Keywords Dynamic process orchestration, Process migration, Flexible workflow, Business process reengineering

在大规模跨组织的网络环境下,如何针对动态多变的业务需求,实现资源的透明访问、流程的柔性编排和动态重组,是实现系统动态集成的关键。传统的面向构件技术能够创建可复用的软件构件模块,并通过构件装配解决业务模式比较固定的流程编排,有较高的效率[1]。面向构件技术由于主要专注于创建可复用的特定系统的构件模块,可扩展性受限,不能很好地支撑复杂系统的业务流程的动态重组,面对跨组织的业务需求,更是无能为力。现代企业的高速发展对业务流程管理(Bussiness Process Managerment, BPM)提出了敏捷、实时、动态性等需求,其中一个重要特点就是能够将各种Web服务编排成可执行的业务流程,由于面向服务在动态更新和交互性等方面的优势,以服务为实体来构建面向服务的功能模块并根据业务需求进行动态编排和重组,成为解决大规模跨组织的网络计算环境下系统动态集成的理想选择[2]。

在跨组织的网络化复杂业务应用背景下,针对服务编排具有动态变化且迅速增长的特点,目前的工作流模型的可编程性及柔性还有较大局限,对流程的编排大部分仅研究实现了可预期流程编排问题[3]。不可预期服务编排涉及的随机流程干预造作(插入业务服务、删除业务服务、跳转业务服务、次序调换、替换等)等方面还需要进一步深入研究。本文利用轻量级的开源工作流引擎JBPM(Java Bussiness Process Managerment),根据动态多变的业务流程的编排、执行和迁移等需求,在面向服务的基础上,提出了一种服务动态编排及迁移方案,解决了流程在面对不可预期服务编排时的接续执行问题,提高了流程的执行效率。最后,通过基于JBPM的流程柔性编排集成系统和测试工具LoadRunner验证了其有效性和高效性。

1 相关工作介绍

目前,针对Web服务的流程动态编排和迁移问题,国内外开展了一系列的研究工作,文献[4]提出一种面向用户的工作流模型——工作流描述网,通过提供单实例任务、多实例任

到稿日期:2014-09-28 返修日期:2014-11-16 本文受国家自然科学基金(61163057,61163058),广西自然科学基金(2012jjAAG0063),广西可信软件重点实验室开放基金(KX201308)资助。

王 勇(1965—),男,博士,教授,主要研究方向为服务计算、云计算;常静波(1990—),男,硕士生,主要研究方向为服务计算、工作流;强保华(1972—),男,博士,教授,主要研究方向为Web数据管理、智能搜索,E-mail:qiangbh@guet.edu.cn(通信作者);王玉峰(1965—),男,研究员,主要研究方向为服务计算、信息集成;张学庆(1968—),男,研究员,主要研究方向为云计算、大数据分析与处理。

务、可迁移的任务等组件，直接支持多实例、取消、高级同步等大多数工作流模型不直接支持的模式。该模型虽然实现了对各种复杂的业务流程进行建模和分析，但无法适应流程的动态重组。文献[5]通过描述业务流程模型变更管理的特点以及业务模型生命周期，提出了一个支持业务流程动态更新的工作流管理模型和基于该模型的业务流程动态迁移算法，算法中引入区域划分法，在迁移之前进行相关数据一致性检查和影响区域比较，但只是实现了一定程度的动态调整，无法达到在流程整个生命周期的自适应。文献[6]使用上下文感知技术，提出了一种自适应构件，采用事件机制感知流程在运行过程中发生的变化，将流程结构进行调整，针对流程变更的4种方式实现了流程动态变更的算法。但是该算法仅支持简单的面向任务的工作流的相关变更，对动态流程尤其是以服务为核心的流程缺乏有效支持。文献[7]研究了基于 flex 和 JBPM 实现自定义工作流来解决业务流程需求多变的问题，实现流程模型到 xml 数据的转换，通过动态路由实现了工作流多步回退。文献[8]基于 JBPM 工作流提出了一种动态生成流程定义文件的自动生成算法，通过对工作流设计阶段的流程任务节点数据进行分析，根据设计节点数据自动生成流程定义文件和流程图，改变了使用 eclipse 图形化工具手工进行绘制的方法。

目前的研究工作能够比较有效地解决相对独立的系统和比较固定的业务流程编排需求，而面对复杂业务流程的不可预期变化，不能很好地解决跨域业务流程的动态重组和迁移。传统工作流对流程变更和迁移的支持并不完善，工作流的运行机制局限于相对固定的流程模板，模板的变更会引起流程定义文件的再次部署和执行，新旧流程的迁移存在很大困难，使不可预期的流程变更操作缺乏连续性，无法实现接续执行。目前 JBPM 工作流提供的迁移方法仅能够支持流程的简单迁移，大量冗余节点依然需要重复执行，这在一方面增加了流程重复运行的开销，另一方面增加了面向服务流程编排的复杂度，使流程在监控、服务数据的接收方面难以控制。本文在借鉴参考文献的基础上，对流程动态编排行为进行建模，给出了一个面向服务的工作流迁移算法，从方法上解决了业务流程变更、动态迁移、接续执行等问题。

2 流程编排与迁移模型

2.1 模型

服务动态编排需要根据用户提交的任务，由系统作全局规划，建立任务与资源之间的匹配，根据 QoS 进行服务优选，确定流程模板来对流程组件进行编制和组合，通过服务间的相互协作，完成核心业务，再将编排的流程向外公布，形成更加粗粒度的服务供客户端调用[9]。建立业务流程模型并部署完成后，就应建立起业务流程和业务流程执行语言的映射关系，然后把业务流程转化成业务流程可执行语言[10,11]。在流程开始时提供输入参数，在流程结束时返回结果至该接口。在流程执行过程中，对流程进行实时监控，如果流程不需要变更，则控制流程让其执行完成，释放资源返回数据；如果流程在进行过程中需要变更，则中断整个流程，将对应的流程实例挂起，对流程文件进行调整，调整的内容主要包括流程节点位置的变化、增加一个流程节点、删除一个流程节点、替换一个流程节点上绑定的服务、修改一个流程节点。调整完成后进行流程的正确迁移后再定位断点，继续执行。流程的运行需要流程引擎的支持才能完成它被赋予的业务功能。JBPM 工作流引擎提供了部署业务流程模板的方法，将与业务流程模板相关的文件(定义流程的 JPDL 文件、根据图形化流程定义同步生成的流程图片 PNG 格式文件、事件监听器等用户自定义代码的 Java 类文件、其他流程资源文件)整合成归档文件，统一部署在流程服务器的模板库中。流程动态编排及迁移模型如图1所示。

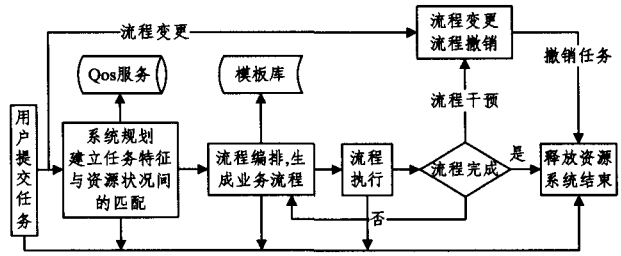

图1 流程动态编排及迁移模型

2.2 流程变更

工作流流程的动态编排体现为流程节点的调整，主要包括动态加入流程节点、动态删除流程节点、动态变更流程节点次序、动态替换流程节点[12]。将节点之间的连接线 transition 作为一个构件，在调整节点时只需要更改连接关系就可以实现对流程的修改。N_{comp} 代表组件节点，$C(n,r)$ 代表节点 n 与 r 的连接线，初始状态流程图如图2所示。

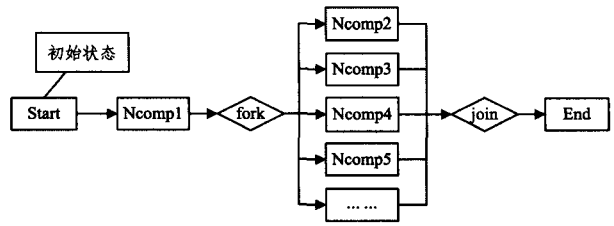

图2 流程初始状态

定义 1(动态插入流程节点 increase)

$S:\{\exists t1,t2,t3\in T(t1<t2<t3),$

$\exists N_{comp1},C_{(comp1,fork)},C_{(start,comp1)}\in F^{(t1)},$

$N_{comp},C_{(comp,null)}\notin F^{(t1)}\}$

$function(increase(N_{comp},C_{(start,comp1)})^{(t2)})\xrightarrow{S}$

$\exists N_{comp},C_{(comp,comp1)},C_{(start,comp)}\in F^{(t3)},$

$C_{(start,comp1)}\notin F^{(t3)}$

increase 操作可以加入一个流程节点，在 $t1$ 时刻 $Ncomp$ 和连接 $C(comp,null)$ 不在流程 F 中，$t2$ 时刻执行 *increase* 操作后，$t3$ 时刻流程 F 包含节点 $Ncomp$ 和连接 $C(comp,comp1)$、$C(start,comp)$。插入流程节点操作如图3所示。

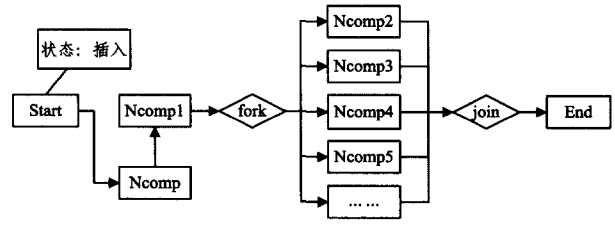

图3 插入流程节点示意图

定义 2(动态替换流程节点 replace)

$S:\{\exists t1,t2,t3\in T(t1<t2<t3),$

$\exists N_{comp1} \in F^{(t1)}, \exists N_{comp} \notin F^{(t1)}$

$\exists C_{(comp1,fork)}, C_{(start,comp1)} \in F^{(t1)}\}$

$function(replace(N_{comp1}, N_{comp})^{(t2)}) \xrightarrow{S}$

$\exists N_{comp}, C_{(comp,fork)}, C_{(start,comp)} \in F^{(t3)}$,

$\exists N_{comp1}, C_{(comp1,fork)}, C_{(start,comp1)} \notin F^{(t3)}$

replace 操作用一个新的流程节点替换流程中已存在的流程节点，在 $t1$ 时刻，流程 F 中存在节点 $Ncomp1$、节点 $Ncomp1$ 与后续节点 *fork* 的连接 $C(comp1, fork)$ 和节点 $Ncomp1$ 与前驱节点 *start* 的连接 $C(comp1, start)$，$t2$ 时刻执行 *replace* 方法，$t3$ 时刻流程 F 中流程节点 $Ncomp$ 替换了 $Ncomp1$，前驱节点和后续节点仍然为 *start* 和 *fork*，替换流程节点操作如图 4 所示。

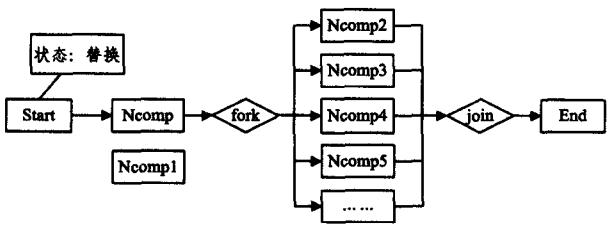

图 4　替换流程节点示意图

定义 3(动态删除流程节点 delete)

$S:\{\exists t1, t2, t3 \in T(t1 < t2 < t3)$,

$\exists N_{comp1}, C_{(comp1,fork)}, C_{(start,comp1)} \in F^{(t1)}\}$

$function(delete(N_{comp1})^{(t2)}) \xrightarrow{S}$

$\exists C_{(start,start)} \in F^{(t3)}$,

$\exists N_{comp1}, C_{(comp1,fork)}, C_{(start,comp1)} \notin F^{(t3)}$

delete 操作删除流程中已经存在的一个节点，如图 5 所示，$t1$ 时刻节点 $Ncomp1$ 存在于流程中，与节点 $Ncomp1$ 相关的两条连接 $C(comp1, fork)$，$C(start, comp1)$ 也存在于流程图之中，$t2$ 时刻执行操作 *delete* 后，节点 $Ncomp1$、$C(comp1, fork)$、$C(start, comp1)$ 不在流程之中，节点 *start* 与节点 *fork* 之间的连接 $C(start, fork)$ 包含在流程之中。删除流程节点的操作如图 5 所示。

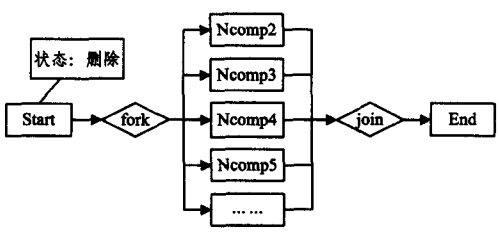

图 5　删除流程节点示意图

定义 4(动态更新节点次序 change)

$S:\{\exists t1, t2, t3 \in T(t1 < t2 < t3)$,

$\exists N_{comp1}, C_{(comp1,fork)}, C_{(start,comp1)} \in F^{(t1)}$,

$\exists N_{comp2}, C_{(comp2,join)}, C_{(fork,comp2)} \in F^{(t1)}\}$

$function(change(N_{comp1}, N_{comp2})^{(t2)}) \xrightarrow{S}$

$\exists N_{comp1}, C_{(comp1,join)}, C_{(fork,comp1)} \in F^{(t3)}$,

$\exists N_{comp2}, C_{(comp2,fork)}, C_{(start,comp2)} \in F^{(t3)}$

change 操作将流程中已经存在的节点变换次序。在 $t1$ 时刻，流程中存在节点 $Ncomp1$、$Ncomp2$ 以及两个节点的前后连接；在 $t2$ 时刻，执行了 *change* 操作；在 $t3$ 时刻，两个节点仍然存在于流程中，而两个节点的前后连接顺序发生了改变，节点之间的位置发生了互换。流程节点次序更替的操作如图 6 所示。

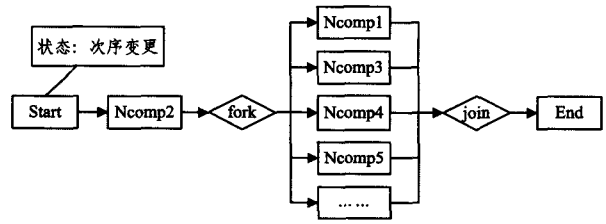

图 6　更替流程节点次序示意图

3　迁移算法

迁移算法针对动态流程编排及迁移的业务需求，根据流程变更反馈的信息，以及流程的变化对流程文件进行扫描和修改，使用两类映射分别标记用户与流程之间的关系以及流程和流程之间的继承关系，修改完成后运行新流程寻找断点位置并接续执行，在流程结束后根据新旧流程的继承关系完成数据的一致性合并。

算法 1

要求：初始化 JBPM 工作流模板库

初始化用户映射表 UserMap

初始化流程实例映射表 ProcessMap

初始化流程实时运行节点 NodeMap

将流程定义文件部署在数据库中

```
1. 函数 deployment
2. if("流程文件". exist)
3. do String userId;/ * 生成唯一用户标识符 userId * /
4. do inputProcessInfo / * inputProcessInfo 传入流程所需参数 * /
5. do startProcessInstance / * startProcessInstance 开始流程实例 * /
6. 返回 processID / * 流程实例 Id * /
7. do put(userId,processID) / * 将用户唯一标示符与流程实例 Id 利
   用键值对的关系存入用户映射表 UserMap * /
8.    decide=true;
9. if (decide) / * true 表示无需修改，流程继续运行 * /
   do NodeMethod/ * 执行 java 节点中定义的方法 * /
10. else if(!decide)/ * false 表示需要修改 * /
11.    do modifyProcess/ * 修改流程并支持接续传输 * /
12.    return receiveMsg
13. 函数 NodeMethod
14.    do Connection() / * 连接服务总线 * /
15. sendMessage(userId, message) / * 将用户唯一标识符放入消息
    头，发送调用 WebService 的请求 * /
16.    while(true)
17.    if(queue. has(userId)) then
18.    do receiveMsg / * 循环等待队列中是否有 userId 为消息头的信
    息，有则接受 * /
19.       do list. put(node)
20. do nodeMap. put(processId,list) / * 将当前执行完成的节点与流
    程实例 id 对应存入 NodeMap * /
21. 函数 modifyProcess
22. do 修改流程模板(根据上面提出的迁移方案就行流程灵活变更)
23.       save
24. / * 根据 processId 取出旧流程已经运行过的节点 List * /
```

```
25. for i=0;i<list. size;i++
26.    do xpath(node);/ * 取出节点 xpath 在新流程中进行定位 * /
27.    do jdom. modify();/ * jdom 操作新流程文件,将运行过的节点
       内容删除 * /
28. 发布新流程
29. 开始流程实例返回新流程实例 id
30. put 旧流程实例 id:新流程实例 id ProcessMap
    / * 将新旧流程实例的 id 存入 ProcessMap * /
31.    get list(list. size);/ * 获得旧流程停止节点信息 * /
32. while(node! =停止节点)
33. do 向下继续执行
34. 函数 receiveMsg
35. receive(userid,message)/ * 接收队列中消息头有 userId 的消息
36. UserMap. get(userId)/ * 根据用户 id 获得旧流程 id * /
37. ProcessMap. get(ProcessId);/ * 根据新旧流程 id 取到新流程实
    例 id * /
38. if(message belong to 新流程 id) or (message belong to 旧流程 id)
39. combine(messages)
40. return message
```

在流程执行迁移的过程中,我们需要更新如下数据结构:

(1)UserMap:这是一个哈希表,可以返回用户的唯一标示符与流程实例 Id 的对应关系。哈希表的键是流程实例 Id,在多用户多任务的执行环境下,一个用户对应多个流程实例,使用 UserMap 来区分同一个用户下多个流程。

(2)ProcessMap:用于显示新旧流程对应关系的哈希表,哈希表的键是旧流程的实例 ID,在流程发生动态变更时,由于流程文件的更改产生的新流程实例依据算法所述可以实现接续执行,但在执行结束后仍应根据哈希表中新旧流程的对应关系来将两个流程的返回数据进行合理地组合。

(3)NodeMap:随着流程的向前推进将每个流程的信息存入一个链表中,NodeMap 用于存放实例 Id 与流程节点链表的对应关系,在流程发生动态变更时,根据流程实例 Id 获得当前实例的节点链表 list,list 中存有当前执行的断点信息,流程迁移过程中需要根据断点信息进行流程文件的重写,并查找到断点处接续执行。

4 实验

本章以基于 SOA 的复杂系统中的旅游动态规划流程为例,来阐述流程动态编排及迁移方案的有效性。简化的流程如图 7 所示。在用户需要旅游服务时启动该流程:(1)用户填写基本信息,如出发地、目的地、时间及关注的重点(如价位优先或速度优先);(2)系统根据用户信息查询 Web 服务库中的 QoS 服务质量信息来动态规划服务,并反馈给用户;(3)根据反馈的服务系统自动生成流程定义文件,将优选的服务与流程节点绑定,生成流程实例;(4)运行流程实例,若用户对服务无异议可继续执行直至流程结束,否则转到(5);(5)用户暂停流程,并根据需求更改流程定义文件,重新绑定服务,根据流程迁移算法对流程进行重写,并返回到流程断点处接续执行,若用户对服务无异议可继续执行直至流程结束,否则转到(5)。生成具体的流程图如图 8 所示,流程图中服务包括机票、酒店、天气等服务,随着业务的变化,需要对原有的流程进行变更,如增加服务、删除服务、替换服务及修改服务等,变更后根据第 3 节提供的流程迁移算法对流程进行重写,并定位到执行断点接续执行。

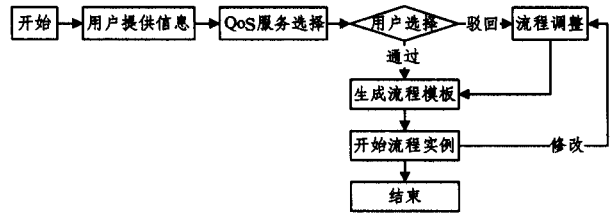

图 7 流程动态迁移与更替示意图

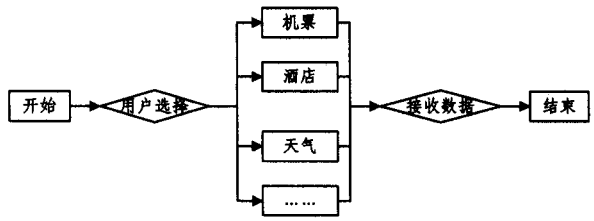

图 8 旅游流程模板

LoadRunner 是一种预测系统行为和性能的工业标准级测试工具。LoadRunner 通过模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题,能够对整个企业架构进行测试,是一种适用于各种体系架构的自动负载测试工具,能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个系统,它通过模拟实际用户的操作行为和实时性能监测,来更快地查找和发现问题。本文使用 LoadRunner 测试工具,在基于 SOA 的集成系统中以旅游为例,通过对比采用迁移算法和不采用迁移算法的流程接续执行的时间和并发度,验证流程接续传输算法的高效性。

首先,设计一类有 50 个流程节点的流程 1 和流程 2,随机产生断点并进行迁移接续执行,在 10 用户、50 用户和 100 个用户并发执行的过程中进行时间对比,流程 1 表示未使用接续传输迁移算法的流程模板,流程 2 表示使用接续传输算法后流程接续传输的流程模板。流程 1 与流程 2 的执行时间对比如表 1 所列。

表 1 流程节点执行时间对比

	用户数	总时长	单个最大时间	单个最小时间	平均时间
流程 1	100	5min40s	52.961	13.376	35.214
流程 2	100	5min26s	54.213	1.585	24.649
流程 1	50	1min20s	62.945	54.045	59.685
流程 2	50	1min4s	53.151	20.666	43.26
流程 1	10	1min4s	45.17	19.928	31.062
流程 2	10	46s	15.864	19.928	16.606

通过分析表 1 的实验数据可以明确知道:流程随机断点的位置越接近开始节点,流程 1 与流程 2 的时间差距越小,而当随机断点位置越靠近结束节点时,由于流程 1 的迁移需要执行大量的重复节点,单个流程的执行时间将远大于采用本文迁移算法的流程 2。表 1 中流程 2 在 100 用户下的单个最小时间 1.585s 相比于流程 1 的 13.376s 有明显的优势,由此显示了本文的迁移算法的高效性。

其次,为了验证流程迁移算法在支持高并发方面的优势,在集成系统中设计一类有 50 个流程节点的模板,使用 LoadRunner 测试工具进行一组测试,其中流程 1 为不使用迁移算法的流程模板,流程 2 采用了本文中介绍的流程迁移算法进行断点接续执行。在实验中,流程 1 和流程 2 在相同的位置

产生断点并采用不同的迁移方式进行接续执行，同一实验条件下，模拟多用户分别并发调用流程 1 和流程 2 的过程，通过执行时间和系统的吞吐量以及平均响应时间的交叉对比来说明本文提出的迁移算法对高并发性的有效支持。实验条件如下。

(1)实验 1：流程模板 1 与模板 2 各设置有 50 个节点，模拟 100 个用户，在持续每秒增加 1 个用户的情况下调用模板 1 和模板 2，在相同的断点处执行流程迁移并查找断点继续执行。

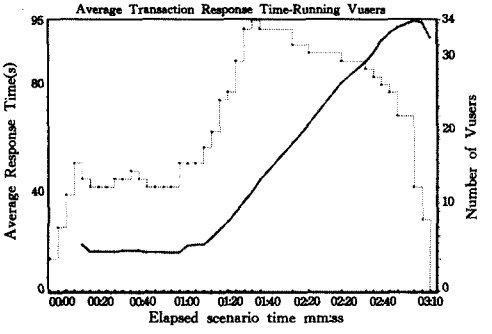

图 9　流程 1 用户并发数与平均响应时间对比

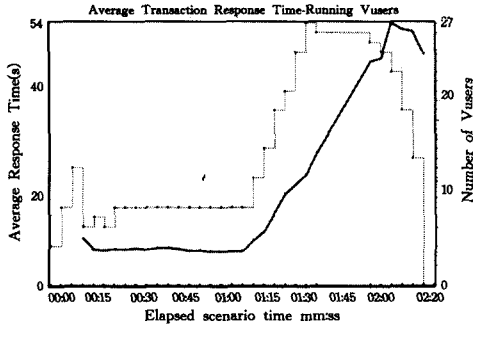

图 10　流程 2 用户并发数与平均响应时间对比

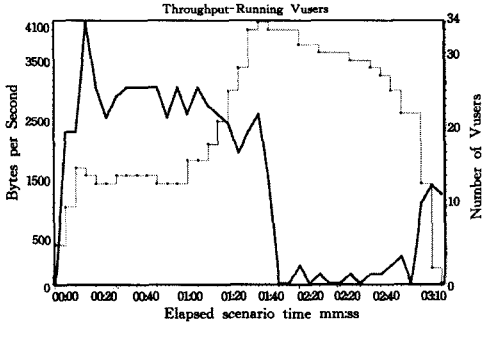

图 11　流程 1 并发用户数与系统吞吐量对比

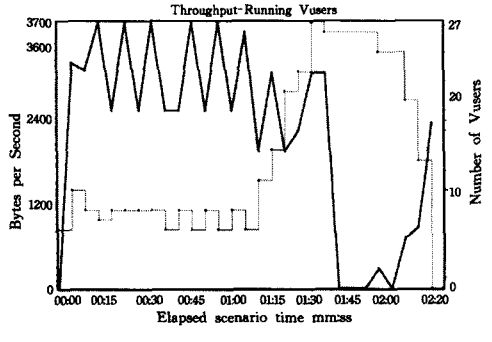

图 12　流程 2 并发用户数与系统吞吐量对比

实验结果如图 9－图 12 所示，图 9 和图 10 分别表示流程 1 和流程 2 的用户数与平均响应时间的对比。图 11 与图 12 分别表示流程 1 和流程 2 的用户数与系统吞吐量的对比。表 2 统计了本次实验中的一些关键数据的对比。

表 2　实验 1 关键数据对比

流程名称	最大用户数	平均用户数	最大吞吐量	平均吞吐量	平均响应时间
流程 1	38	19.70	3866.5	1513.77	49.38
流程 2	15	6.08	3692.4	2477.06	8.75

(2)实验 2：流程模板 1 与模板 2 各设置有 50 个节点，模拟 50 个用户，在同时并发的条件下分别调用模板 1 和模板 2，在相同的断点处执行流程迁移并查找断点继续执行。

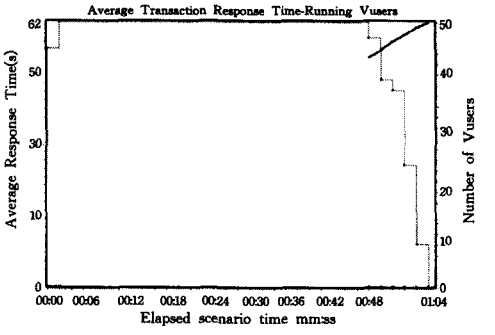

图 13　流程 1 用户并发数与平均响应时间对比

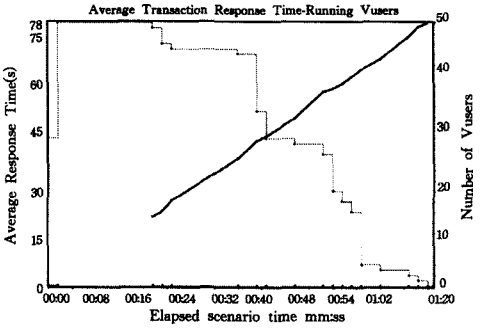

图 14　流程 2 用户并发数与平均响应时间对比

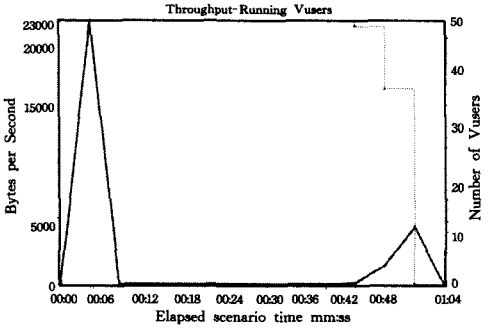

图 15　流程 1 并发用户数与系统吞吐量对比

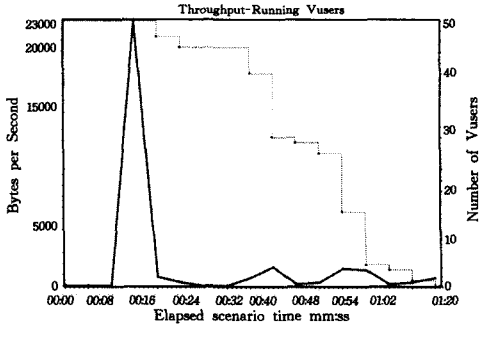

图 16　流程 2 并发用户数与系统吞吐量对比

实验结果如图13—图16所示，图13和图14分别表示流程1和流程2的用户数与平均响应时间的对比。图15与图16分别表示流程1和流程2的用户数与系统吞吐量的对比。表3统计了本次实验中的一些关键数据的对比。

表3 实验2关键数据对比

流程名称	最大用户数	平均用户数	最大吞吐量	平均吞吐量	平均响应时间
流程1	50	23.1	23719	4689.85	33.998
流程2	50	21.2	23740	5269.23	19.025

通过观察上述两种实验的结果，包括在不同并发情况下的时间、系统吞吐量以及平均事务响应时间的对比，可以明显地发现，流程2在迁移算法的支持下，相比流程1在执行时间上有所减少，在支持高并发上的表现也更加稳定和高效。在实验1中，为持续增加用户的测试，流程2始终保持着较少用户运行数量，峰值用户数为15，而没采用流程迁移算法的流程1峰值为38。在吞吐量的比较中，系统在采用流程2作为模板的实验全过程中始终保持高吞吐量，而采用流程1作为模板的实验全过程中，随着负载的增加，系统呈现出一个持续走低的吞吐量变化。在平均响应时间上，采用流程2为模板的系统平均响应时间更少，系统完成任务的时间也更少。在实验2中，系统中50个用户同时并发，分别调用流程1与流程2在相同用户并发数的前提下比较系统的吞吐量、平均响应时间，结果表明，系统采用流程1作为模板的实验全过程相比采用流程2作为模板的系统在吞吐量方面有一定的劣势，尤其在平均响应时间的对比上差距比较明显。

结束语 本文根据目前动态多变的业务流程的编排、执行和迁移等需求，在面向服务的基础上，首先提出了一种服务动态编排及迁移的方案，构造基于JBPM流程的动态编排及迁移模型，通过任务与资源之间的匹配、QoS优选服务、柔性工作流引擎进行服务编排和组合完成核心业务。其次，形式化表述了流程变更的几种方式，并给出了一种流程迁移算法，该算法通过记录断点建立新旧流程的数据关联，提升迁移过程中的执行效率。最后，使用LoadRunner测试工具在集成系统中测试对比了采用本文介绍的迁移算法的流程与原流程在相同断点处执行迁移的时间效率及对高并发的支持，证明了本文提出的迁移算法在执行时间和支持高并发的方面的优势。下一步将在流程无等待节点的监控以及子流程动态绑定从而实现复杂流程编排方面进行深入研究。

参 考 文 献

[1] 王德俊. 面向服务的分布式系统动态更新研究[D]. 上海：上海交通大学，2010
Wang De-Jun. Research on dynamic updating of service oriented distributed Systems [D]. Shanghai: Shanghai Jiao Tong University, 2010

[2] 张辉栋，卢选民，杨杰，等. 一种基于SOA和JBPM的工作流引擎模型[J]. 微型机与应用，2013，15：12-14，17
Zhang Hui-dong, Lu Xuan-min, Yang jie, et al. A workflow engine model based on SOA and JBPM[J]. Microcomputer & Its Applications, 2013, 15: 12-14, 17

[3] 杨芙清，吕建，梅宏. 网构软件技术体系：一种以体系结构为中心的途径[J]. 中国科学(E辑：信息科学)，2008(6)：818-828
Yang Fu-qing, Lu Jian, Mei Hong. Technical frameworkfor internetware: An architecturecentric approach[J]. Sciencein China Series E: Information Sciences, 2008(6): 818-828

[4] 张亮，姚淑珍. 一种新的面向用户的工作流模型[J]. 计算机集成制造系统，2006，12(11)：1760-1765
Zhang Liang, Yao Shu-Zhen. A new user oriented workflow model [J]. Computer Integrated Manufacturing Systems, 2006, 12(11): 1760-1765

[5] 杨书新，王坚，马福民. 工作流管理系统的流程柔性动态变更研究[J]. 计算机应用，2006，11：2736-2738
Yang Shu-xin, Wang Jian, Ma Fu-min. Research on flexible and dynamic changing process of workflow management system [J]. Journal of Computer Applications, 2006, 11: 2736-2738

[6] 魏乐，李亚玲，赵秋云，等. 基于自适应构件的工作流流程动态变更模型[J]. 计算机集成制造系统，2010，12：2603-2610
Wei Le, Li Ya-Ling, Zhao Qiu-Yun, et al. A dynamic changing-model of workflow process based on adaptive component [J]. Computer Integrated Manufacturing Systems, 2010, 12: 2603-2610

[7] 王超，戴牡红. 基于Flex和Jbpm的自定义工作流研究与实现[J]. 计算机系统应用，2013，08：58-62
Wang Chao, Dai Mu-hong. Research and implementation on custom workflow based on Flex and Jbpm[J]. Computer Systems & Applications, 2013, 08: 58-62

[8] 王伟，徐文胜. JBPM流程定义文件自动生成算法[J]. 计算机应用，2012，S2：89-91
Wang wei, Xu Wen-sheng. Automatic algorithm for JBPM process definition file generation[J]. Journal of Computer Application, 2012, S2: 89-91

[9] Peltz C. Web Services Orchestration and Choreography [J]. IEEE Computer, 2003(36): 46-52

[10] Decker G, Ko pp O, Leymann F, et al. BPEL4Chor: Extending BPEL for Modeling Choreographies [C]//Proceeding s of IEEE International Conference onWeb Services(ICWS). 2007: 296-303

[11] 吴步丹，林荣恒，陈俊亮. 基于模板的工作流应用系统代码自动生成[J]. 华中科技大学学报(自然科学版)，2013，S2：18-21
Wu Bu-dan, Lin Rong-huan, Chen Jun-liang. The automatic code generation of workflow application system based on templates [J]. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2013, S2: 18-21

[12] 崔立真，王海洋，于庚. 跨组织工作流模型的形式化描述及分析[J]. 系统仿真学报，2005，4：782-785
Cui Li-zhen, Wang Hai-yang, Yu geng. Formal description and analyzation on cross-organizational workflow model [J]. Journal of System Simulation, 2005, 4: 782-785