基于策略的 Web 服务实时性能评价与验证

刘永利,白晓颖,陈 光,王立军
(清华大学计算机科学与技术系,北京 100084)

摘 要: 面向服务的架构(SOA)通过标准 Internet 协议实现了异构平台上的服务动态集成.然而,开放网络环境的不稳定性和不确定性,给 Web 服务在线质量评价带来一定的困难.Web 服务的测试只能给出测试环境下的服务评价,不能评价服务的真实运行情况,因此服务实时监测对服务运行状态追踪及异常检测具有重要的意义.本文基于服务系统的协同监测,采用原子服务和组合服务多种时间特性定义了 Web 服务的性能评价模型.在该模型的基础上,提出了一种基于策略的服务在线质量评价与验证方法,给出了基于 WS-Policy 的策略描述,实现了原型系统,并通过对实验服务的在线监测数据,分析了方法的有效性及性能代价.

关键词: Web 服务; 实时监测; 服务性能评价; 策略执行
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2010) 2A-182-06

Policy-Based Runtime Performance Evaluation and Validation of Web Services

LIU Yong-li, BAI Xiao-ying, CHEN Guang, WANG Li-jun
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

Abstract: Service-Oriented Architecture (SOA) enables the dynamic composition of services across heterogeneous platforms through standard Internet protocols. However, it is hard to evaluate service runtime qualities due to the instability and uncertainty of the open Internet environment. Testing is usually exercised in the dedicated testing environment and cannot analyze service behavior in the execution environment. Hence, runtime monitoring is necessary for runtime system status tracking and anomaly detection. The paper is an extension of our previous research on the collaborative monitoring of service-based systems. It defines a service performance evaluation model using the temporal properties of both atomic and composite service executions. Policies are defined and encoded following the WS-Policy framework. They are enforced at runtime for evaluating the service performance based on the evaluation model. A prototype system is implemented. Experiments are exercised to illustrate the proposed approach and analyze the monitoring overhead.

Key words: web service; runtime monitoring; service performance evaluation; policy enforcement

1 引言

面向服务架构(Service Oriented Architecture, SOA)提供了一种新的软件开发范型,Web 服务技术是 SOA 的典型实现方式.服务的质量评价是服务选择、服务各方建立信任关系的基础.但是,传统 SOA 架构中缺少对服务质量的持续验证和确认机制,因此,需要提供服务在线监测及服务质量评价机制,建立用户对服务质量的信任.

服务质量评价模型是进行服务评价的基础,评价指标主要包括价格、执行时间、可靠性、可用性、负载、信誉度等[4,7,10].文献[7]给出了基于 SOA 的应用的基本评价指标,包括最大响应时间、最小响应时间、平均响应时间等,并给出了测试结果;但该评价模型只给出了原子服务的性能评价方法,对组合服务的情况没有进行深入分析.

在评价模型的基础上,可以通过对服务执行引擎扩展事件机制[9],或通过服务执行引擎的 Log 机制[12]来实现服务监测.但是上述两种机制不能对服务进行实时监测.文献[5]提出了一种 Web 服务协同(WSLA)框架,定义了服务在线监测和验证 QoS 参数;提出了一种描述不同类型度量体系的描述语言,给出了一种基于 WSLA 的服务互操作架构.文献[6]针对 QoS 参数收集和服务的动态组合提出了一种基于策略的在线监测框架,其服务评价指标主要为服务组合的动态调整服务.

验证策略是服务评价的一种实现方式,多采用约束定义语言进行描述[2,6],缺乏统一的描述标准.WS-Policy 是针对服务策略定义的标准规范[6,11],在服务安全等领

收稿日期:2009-05-22;修回日期:2010-02-05
基金项目:国家自然科学基金(No.60603035);北京市自然科学基金(No.4072014)

域得到了广泛应用.

随着研究的深入,已经出现了实时在线监测工具.针对 BEPL 描述的服务的监测工具已经出现,如 Oracle 业务活动监视(Oracle BAM)工具提供了基于 BPEL 的 Web 服务实时监测能力[14],提供了个性化的实时信息板,通过信息缓存来保证系统的实时性,提出了信息捕获、过滤、分析、警告的监测方案.但是,针对 OWL-S 描述的服务的监测工具还很少见.

本文针对 OWL-S 描述的服务,从原子服务和组合服务的角度,采用多种时间特性定义 Web 服务的性能评价模型,采用基于策略的服务在线质量评价与验证方法,通过 WS-Policy 进行策略描述,初步探讨了策略冲突检测与消解问题.服务实时监测与验证框架如图 1 所示,探测器插装部件完成探测器的部署,探测器负责收集服务执行信息;收集到的数据由性能评价部件依据定义好的服务性能评价模型进行处理,给出服务性能评价结果;策略定义描述了服务约束,用来验证对象是否符合需求;数据、性能评价结果交由策略执行部件进行数据或服务性能验证,以确定服务是否满足用户需求,以便于用户对服务调用进行动态调整,提高服务质量.

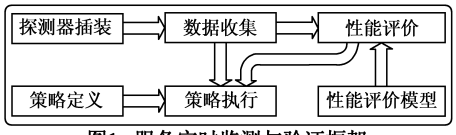

图1　服务实时监测与验证框架

2　服务性能评价模型

服务性能是影响用户体验的重要因素,服务性能评价是服务评价的重要方面,下面从原子服务、组合服务、客户端、服务端等不同维度给出服务性能评价各项指标的定义及度量方法.

2.1　原子服务性能评价

原子服务性能可从服务端和客户端两个角度,通过时间相关指标来度量,如响应时间、传输延迟时间等.

响应时间 T 分为客户端响应时间 T_c 和服务端响应时间 T_s. T_c 指从应用发起请求开始到接收到服务端完整的响应消息所耗费的时间. T_s 是指从服务接收到请求消息,到产生执行结果所耗费的时间. T_c、T_s 不仅与服务本身有关,还同服务的运行环境、网络状态、服务器负载等因素有关,是一个由多种因素决定的动态变量,是服务评价体制中的重要参数.

最大响应时间 $T_{\max}$ 包括服务端最大响应时间 $T_{\max\_s}$ 和客户端最大响应时间 $T_{\max\_c}$. $T_{\max\_s}$ 指服务历次调用中服务端响应时间的最大值; $T_{\max\_c}$ 指服务历次调用中客户端响应时间的最大值.

最小响应时间 $T_{\min}$ 包括服务端最小响应时间 $T_{\min\_s}$ 和客户端最小响应时间 $T_{\min\_c}$. $T_{\min\_s}$ 指服务历次调用中服务端响应时间的最小值; $T_{\min\_c}$ 指服务历次调用中客户端响应时间的最小值.

平均响应时间 T_{avr} 是一个统计量,是服务历次执行所监视到的响应时间的平均值,包括客户端平均响应时间 T_{avr_c} 和服务端平均响应时间 T_{avr_s},分别由以往监视到的 T_c 和 T_s 经统计计算得出.

传输延迟时间 T_{delay} 指服务请求消息和服务响应消息在网络上的传输时间之和.在组合服务的执行引擎中可监视到各个原子服务的 T_c; T_s 可在 SOAP 引擎的服务端监视到. T_{delay} 由原子服务的 T_c 和 T_s 计算得到.

$$T_{delay} = T_c - T_s$$

平均传输延迟时间 T_{avr_delay} 由 T_{avr_c} 和 T_{avr_s} 计算得出.

$$T_{avr\_delay} = T_{avr\_c} - T_{avr\_s}$$

2.2　组合服务性能评价

组合服务通过服务组合描述语言(如 BPEL、OWL-S)来描述,由原子服务采用不同的控制结构组成,控制结构分为顺序执行、并行执行、条件选择执行和循环执行四种[4].其中,循环执行结构可以展开成顺序执行序列,所以可以仅考虑顺序执行、条件选择执行和并行执行三种情况.由于服务组合模型及输入参数的不同,组合服务每次执行中调用的原子服务是不同的,每次执行所经历的路径也可能不同.假定一个组合服务 cs 由 n 个原子服务 s_1、s_2,…, s_n 组成,给每个原子服务 s_i 赋予一个执行概率 p_{s_i},对于顺序执行的原子服务 p_{s_i} 取 1;对于条件分支中的原子服务 s_i,通过历史监测数据计算出 p_{s_i};对于并行执行的原子服务,将每个原子服务的执行概率设为 1.

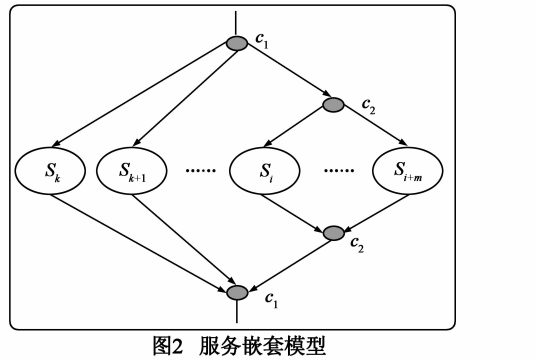

图2　服务嵌套模型

如果服务组合模型中存在嵌套结构,如图 2 所示,并行结构 c_1 中嵌套了 c_2, c_2 可能是选择结构,也可能是并行结构.假设嵌套结构 c_2 的执行概率为 p_{c_2}, p_{c_2} 由历史监测数据统计得出.对 c_2 中的服务 s_i,当 c_2 是选择执行结构时,假定 s_i 在 c_2 中的历史统计执行概率为 $p_{c_2\_s_i}$,则服务 s_i 的执行概率可以通过如下公式计算.

$$\begin{cases} p_{s_i} = p_{c_2}, & c_2 \text{ 为并行执行结构} \\ p_{s_i} = p_{c_2\_s_i} p_{c_2}, & c_2 \text{ 为选择执行结构} \end{cases}$$

引擎执行时间 T_{engine} 指执行引擎完成一次原子服务调用,引擎本身所带来的平均时间耗费.其由引擎工作原理、引擎设计的合理性、引擎运行环境等因素共同决定,其值可以通过监视信息统计得出.

将引擎平均完成一次服务调用所用的完整时间耗费定义为 $T_{complete}$,包括 T_{engine} 和 T_{avr_c} 两部分.

$$T_{complete}(s_i) = T_{engine} + T_{avr\_c}(s_i)$$

组合服务响应时间 $T(cs)$ 可以通过监测得到.当组合服务不满足用户需求时,如响应时间过大,需要对组合服务中的原子服务进行替换.重构后,对新的组合服务的 $T(cs)$ 依据服务组合模型及历史监测结果进行预测,以确定是否满足需求.$T(cs)$ 的预测要选择一条耗时最长的路径进行,以图 2 为例,在控制结构 c_2 中,如果 c_2 是并行执行结构,就要选择 c_2 中响应时间最长的服务参与计算,如果 c_2 为选择执行结构,c_2 中的所有服务都要参与计算.假设选出的服务集合为 $s_1, s_2, \cdots, s_n$,依据概率计算模型有:

$$T(cs) = \sum_{i=1}^{n} T_{complete}(s_i) p_{s_i}$$

最大响应时间 $T_{\max}$ 指组合服务历次调用中响应时间的最大值.

最小响应时间 $T_{\min}$ 指组合服务历次调用中响应时间的最小值.

3 基于策略的服务验证

3.1 策略模型

服务监测策略描述了服务性能需求,提出了服务执行过程中必须满足的约束条件.本文仅针对服务性能进行验证.服务策略定义包含策略对象及策略断言,具体定义如下.

定义 1 服务监测策略 p 表示为 $< subject, assertions >$,其中 *subject* 是策略对象,指策略附加的具体对象;*assertions* 是策略断言集合,描述了策略的约束条件,一个策略可以由多条策略断言组成.

定义 2 服务性能验证对应关系定义为 $< S, P, C >$,其中 S 表示服务集合,服务既可以是组合服务,也可以是原子服务;P 表示策略集合;$\boldsymbol{C}$ 表示 S 与 P 之间的对应关系矩阵,矩阵 $\boldsymbol{C}$ 中的行列值 c_{ij} 定义如下:

$$\begin{cases} c_{ij} = 1, \text{如果 } p_j \text{ 作用于服务 } s_i \text{ 的策略对象上}, \\ \qquad p_j \in P, s_i \in S \\ c_{ij} = 0, \text{否则} \end{cases}$$

定义 3 策略动作对应关系定义为 $< P, A, R >$,其中 P 表示策略集合;A 表示策略所产生的动作集合;$\boldsymbol{R}$ 表示 P 与 A 的对应关系矩阵,矩阵 $\boldsymbol{R}$ 中的行列值 r_{ij} 定义如下:

$$\begin{cases} r_{ij} = 1, \text{如果策略 } p_i \text{ 导致动作 } a_j, p_i \in P, a_j \in A \\ r_{ij} = 0, \text{否则} \end{cases}$$

对 WSDL 描述的原子服务而言,策略对象为操作或者原子服务本身;对 OWL-S 描述的组合服务而言,策略对象可以为原子服务(过程),也可以为组合服务本身.

针对原子服务或组合服务的性能验证需求,选定不同的策略对象,定义策略.一个服务可能存在多个策略对象,一个策略对象可以定义多个策略,构成策略对象的一个完整的策略集合;一个策略也可以同时作用在多个服务的策略对象上.一个策略可以产生多个策略动作,一个策略动作也可以由不同的策略产生.将服务的策略看成对服务的性能、功能、数据的验证条件,当条件满足时表明服务满足用户需求;反之亦然.对原子服务可以定义与原子服务性能相关的策略;对组合服务可以针对各个原子服务及组合服务自身的性能参数定义策略.

3.2 基于 WS-Policy 的策略定义

WS-Policy 提供了一种灵活、可扩展的语法,用于表示基于 XML 的 Web 服务系统中实体的能力、要求和一般特性,把系统中实体特性的约束表示为策略.策略表示既支持简单的声明式断言,也支持复杂的条件式断言.策略断言表示行为的个体要求、能力或其他特性.

WS-Policy 只是一个框架,实现服务在线验证,需要根据领域需求对 WS-Policy 进行扩展,定义 Web 服务监测领域断言体系.首先对 WS-Policy 名字空间进行扩展,扩展前缀 wsMon,表示该策略为服务监测验证断言,并扩展断言类型 wsPerfomanceAssertion,对服务策略对象性能进行约束.然后,将断言与 Web 服务关联.对于服务监测而言,策略可以在 WSDL、UDDI 中进行定义;本文将策略定义扩展到 OWL-S 定义中,策略采用独立的 XML 文件描述,服务调用过程中将策略读入执行.

图 3 给出了策略定义的例子:策略对象为各个原子

```
<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsMon="http://schemas.xmlsoap.org/ws/2004/09/wsMon"
wsu:Id = "PerformancePolicy_xxxService">
  <wsp:ExactlyOne>
    <wsp:All>
        <wsMon:wsPerformanceAssertion ServiceTimeOut="30"/>
        <wsMon:wsPerformanceAssertion ClientTimeOut="400"/>
        <wsMon:wsPerformanceAssertion DelayTimeOut="200"/>
        <wsMon:wsPerformanceAssertion ServiceMaxTimeOut="100"/>
        <wsMon:wsPerformanceAssertion ClientMaxTimeOut="500"/>
        <wsMon:wsPerformanceAssertion ServiceMinTimeOut="50"/>
        <wsMon:wsPerformanceAssertion ClientMinTimeOut="400"/>
        <wsMon:wsPerformanceAssertion ComServiceTimeOut="800"/>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

图3 策略定义

服务(过程)及组合服务本身,定义了针对服务端平均响应时间、客户端平均响应时间、网络平均延迟时间、服务端最大响应时间、客户端最大响应时间、服务端最小响应时间、客户端最小响应时间等性能特性的策略断言.

3.3 策略冲突检测与消解

策略 p_i 由一个断言集合构成,用 $assertions(p_i)$ 表示.对于服务验证而言,断言为条件式断言,策略 p_i 的断言集就是一个条件集合,因此策略 p_i 及策略动作 a,可以转换为一阶谓词逻辑 $assertions(p_i) \rightarrow a$.策略冲突检测可以看成一阶谓词冲突检测.当同一个策略对象定义多个策略时,策略之间可能存在冲突.针对一个策略对象定义的策略,必须保证策略的完整性和一致性,所以必须对策略进行冲突检测与消解.如 $assertions(p_i) \rightarrow a$ 与 $assertions(p_i) \rightarrow \neg a$ 存在策略冲突.可以定义以下几种策略冲突情况.

(1) $assertions(p_i) \rightarrow a \wedge assertions(p_i) \rightarrow \neg a$

(2) $assertions(p_i) \rightarrow a \wedge \neg assertions(p_i) \rightarrow a$

(3) $assertions(p_i) \rightarrow a \wedge assertions(p_j) \rightarrow \neg a \wedge assertions(p_i) \cap assertions(p_j) \neq \phi$

策略冲突的消解,可以是去掉一条策略,或是改变策略中的策略断言.策略由定义到执行要经过策略定义、策略生成、冲突检测、冲突处理等步骤,并由策略执行引擎进行执行.

例如,一个组合服务 cs 由两个原子服务 s_1、s_2 组成,定义如下策略:

- $p_1 = <cs, \{T_{cs}(cs) > 800\}>$
- $p_2 = <cs, \{T_{cs}(cs) < 1000\}>$
- $p_3 = <s_1, \{T_s(s_1) < 300\}>$
- $p_4 = <s_2, \{T_s(s_2) < 400\}>$

其中 p_1、p_2 以组合服务 cs 本身作为策略对象,对组合服务响应时间进行约束;p_3、p_4 以两个原子服务 s_1、s_2 作为策略对象,定义原子服务响应时间的约束条件,p_3、p_4 即可以作为原子服务的验证策略,也可以作为组合服务的验证策略.

策略断言 $p_1 = <cs, \{T_{cs}(cs) > 800\}>$ 和 $p_2 = <cs, \{T_{cs}(cs) < 1000\}>$ 构成策略对象 cs 的一个策略集合,假定两条策略产生的动作均为发出告警(动作 a),且 $assertions(p_1) \rightarrow a$, $assertions(p_2) \rightarrow \neg a$,即满足 p_1 时,发出告警;当满足 p_2 时,不发出告警信息.显然 $assertions(p_1) \cap assertions(p_2) \neq \phi$,存在策略冲突.

4 实验验证

4.1 系统总体架构

如图 4 所示,系统总体架构由原子服务监测和组合服务监测两部分组成,主要包含服务注册中心、监测代理、探测器插装、策略执行等功能部件.服务注册中心完成探测器插装策略生成、服务性能验证策略生成、监测数据收集存储分析及服务监测报告生成等功能,监测数据及处理结果存储在 XML 数据库中.监测代理负责收集本地监测数据并向服务注册中心发送,同时负责接收服务注册中心送来的探测器插装策略及服务性能验证策略,传给探测器插装部件及策略执行部件.探测器具体完成数据捕获任务,对于组合服务在 OWL-S 引擎中进行插装,对原子服务在 SOAP 引擎中进行插装.在 OWL-S 引擎及 SOAP 引擎中扩展实现服务监测数据及服务性能验证策略执行引擎,完成监测数据及服务性能的验证功能.

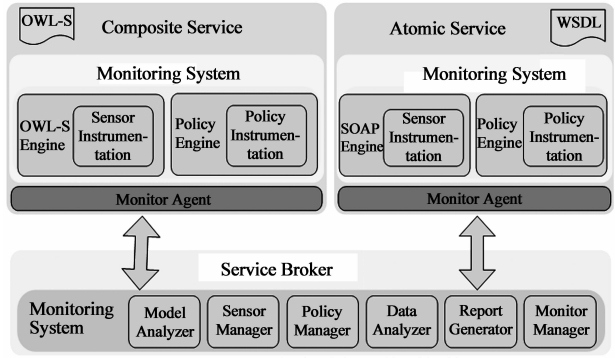

图4 Web服务实时监测体系架构

4.2 试验设计

为了验证平台的有效性,选取了实现旅程管理功能的组合服务 JourneyService 作为监测目标,该组合服务由 5 个原子服务组成:s_1:UserAuthenticationService,其方法 authenticateUser()完成用户验证功能;s_2:FlightService,其方法 reserveFlight()完成机票预订功能;s_3:HotelService,其方法 reserveHotel()完成宾馆预订功能;s_4:CarService,其方法 reserveCar()完成汽车预订功能;s_5:ConfirmService,其方法 confirmJourney()完成旅程确认功能,具体流程如图 5 所示.在该组合服务中,s_2、s_3、s_4 并行执行,构成了一个"Split-Join"结构,该"Split-Join"结构与 s_1、s_5 在整体上组成一个"Sequence"串行结构.

对该旅程管理服务进行实时监测及性能验证时,针对不同功能的原子服务及组合服务响应时间定义不

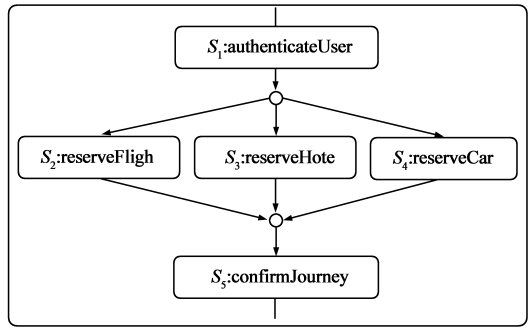

图5 旅程管理服务

同的限制策略,如表 1 所示.前 5 条策略分别对原子服务 s_1、s_2、s_3、s_4、s_5 进行验证,策略 6 对整个组合服务进行验证.各原子服务的 $T_c(s_i)$和 $T_s(s_i)$分别由组合服务执行引擎和 SOAP 引擎监测得出,$T_{delay}(s_i)$由 $T_c(s_i) - T_s(s_i)$得出;组合服务 $T_c(cs)$由 $T_c(s_1) + \max\{T_c(s_2), T_c(s_3), T_c(s_4)\} + T_c(s_5)$计算得出.实验中对组合服务进行了 2000 次调用,收集各原子服务及组合服务的监测数据和性能验证结果,进行综合分析处理.

表 1 服务性能验证策略

	Subject	Constrians	Assertion
1	authenticateUser	$T_{avr_c} < 600$ms	< wsMon:wsPerformanceAssertion ClientTimeOut = "600"/ >
2	reserveFlight	$T_{avr_c} < 450$ms	< wsMon:wsPerformanceAssertion ClientTimeOut = "450"/ >
3	reserveHotel	$T_{avr_c} < 400$ms	< wsMon:wsPerformanceAssertion ClientTimeOut = "400"/ >
4	reserveCar	$T_{avr_c} < 500$ms	< wsMon:wsPerformanceAssertion ClientTimeOut = "500"/ >
5	confirmJourney	$T_{avr_c} < 400$ms	< wsMon:wsPerformanceAssertion ClientTimeOut = "400"/ >
6	JourneyService	$T_{avr_cs} < 1$s	< wsMon:wsPerformanceAssertion ClientTimeOut = "1000"/ >

4.3 监测结果

表 2 给出了具体监测结果,包括调用操作、服务性能验证失效数、失效原因、T_{avr_s}、T_{avr_c}和 T_{avr_delay}.图 6 给出了各原子服务的 T_{max_c}、T_{min_c}、T_{avr_c}、T_{avr_s}的比对结果,从而找出组合服务中 T_{max_c}最长的原子服务;通过 T_{avr_s}及 T_{avr_c}分析出 T_{delay},以确定 T_c 是由原子服务造成的,还是由传输环节造成的,从而找出服务的薄弱环节,对服务进行改进.从结果看,(1)服务延迟时间在 T_s 中占了很大比重,是影响服务性能的关键因素.(2)服务本身的响应时间都能够满足用户要求,服务 CarService 的方法 reserveCar 在服务组装模型的并行执行结构中响应时间最长,是该组合服务中的薄弱环节,应该对该服务进行优化处理,以提升组合服务的整体性能.(3) CarService 的 reserveCar 方法的响应时间是由 T_{delay}造成的,应该从改善服务运行网络环境的角度来提升服务性能.

表 2 服务监测结果

	Service	Operation	Validation Fails	Fail Reason	T_{avr_c}	T_{avr_s}	T_{avr_delay}
1	UserAuthenticationService	authenticateUser	4	Time out	195.46ms	106.24ms	89.21ms
2	FlightService	reserveFlight	1	Time out	293.82ms	123.77ms	170.05ms
3	HotelService	reserveHotel	2	Time out	328.82ms	109.35ms	219.46ms
4	CarService	reserveCar	1	Time out	359.49ms	121.38ms	238.11ms
5	ConfirmService	confirmJourney	2	Time out	163.10ms	100.21ms	62.89ms
6	JourneyService	– –	0	– –	740.38ms	– –	– –

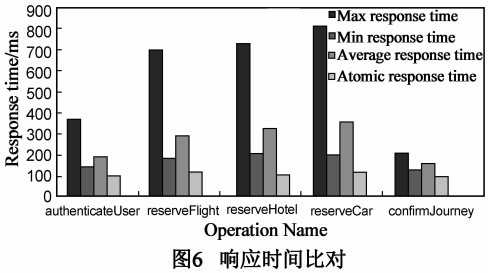

图6 响应时间比对

4.4 监测代价分析

为了评价方法的有效性,对执行监测及性能验证本身带来的代价进行评估.对服务在进行监测和不进行监测的情况下的响应时间进行比对,得出方法本身的执行代价.服务的响应时间受网络环境、服务本身的质量、服务运行环境等因素的影响较大,为了确保评估结果的准确性,我们给出了服务 500 次到 2000 次执行的代价分析结果.

在服务在线监测中,数据捕获是由探测器完成的,因此在进行服务在线监测时要进行探测器插装工作,一般探测器插装在监测前执行,或者在服务监测策略改变的时候,重新进行探测器插装.由图 7 可以看出,对一个组合服务进行监测,其监测代价主要在第一次探测器插装的过程中,其后如果不改变探测器插装方案,

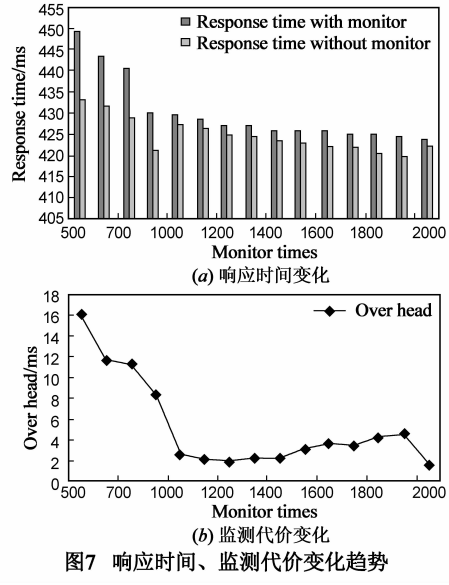

图7 响应时间、监测代价变化趋势

即进行探测器的重新插装,监测代价会趋于稳定,此时仅为数据收集及性能验证的时间耗费.由图 7(*a*)可以看出,执行监测的服务响应时间稳定在 428ms 左右,不执行监测的服务响应时间稳定在 423ms 左右;由图 7(*b*)可以看出稳定后,监测代价在 2 ~ 5ms 范围内,占服务响应时间的 0.9% 以下,对服务的运行性能的影响比较小,该方法对服务的在线监测及服务质量验证是有效的.

5　结论

服务性能评价和服务质量验证是服务在线监测的两个重要内容,本文主要贡献体现在以下几个方面:(1)针对网络环境的动态性、不确定性及服务运行效率的不稳定性,提出了一种服务在线评价与性能验证方法,引入了从原子服务、组合服务、客户端、服务端等不同维度定义的 Web 服务性能评价模型;(2)提出了基于策略的服务性能验证方法,给出了服务性能验证策略的定义及处理方法,采用 WS-Policy 规范对策略进行定义,初步给出了策略冲突检测与消解方法,对组合服务及原子服务的性能验证具有重要意义;(3)服务监测策略可以在线动态调整;(4)提出了对原子服务及组合服务进行统一监测的服务在线监测体系架构,实现了原形系统.

本文利用对 Web 服务实例的监测数据,验证了方法的有效性,分析了方法本身的执行代价.论文工作表明,方法本身的执行代价较小,能够同时适应原子服务及组合服务的在线监测任务,监测策略可以动态调整,监测结果可以共享,满足服务在线监测、评价、验证需求.

参考文献:

[1] Bai X, Lee S, Tsai W, Chen Y. Collaborative Web Services Monitoring with Active Service Broker[A]. In Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference[C]. Washington: IEEE Computer Society, 2008. 84 – 91.

[2] Baresi L, Guinea S, Plebani P. WS-Policy for Service Monitoring[J]. Technologies for E-Services. 2006, LNCS 3811: 72 – 83.

[3] Bhargavan K, Fournet C, Gordon A D. Verifying Policy-Based Web Services Security[J]. ACM Transactions on Programming Languages and Systems, 2008, 30(6): 30 – 59.

[4] 范小芹,蒋昌俊,王俊丽,庞善臣.随机 QoS 感知的可靠 Web 服务组合[J].软件学报,2009,20(3):546 – 556.
Fan Xiao-Qin, Jiang Chang-Jun, Wang Jun-Li, Pang Shan-Chen. Random-QoS-Aware Reliable Web Service Composition [J]. Journal of Software, 2009, 20(3): 546 – 556. (in Chinese)

[5] Keller A, Ludwig H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services[J]. Journal of Network and Systems Management. 2003, 11(1): 57 – 81.

[6] Li F, Yang F C, Shuang K, Su S. A Policy-Driven Distributed Framework for Monitoring Quality of Web Services[A]. In Proceedings of the 2008 IEEE International Conference on Web Services[C]. Washington: IEEE Computer Society, 2008. 708 – 715.

[7] Stantchev V, Malek M. Architectural Translucency in Service-oriented Architectures[J]. IEE Proceedings-Software, 2006, 153(1): 31 – 37.

[8] Tsai W, Zhou X, Wei X. A Policy Enforcement Framework for Verification and Control of Service Collaboration[J]. Information Systems and E-Business Management. 2008, 6(1): 83 – 107.

[9] Vaculin R, Sycara K. Semantic Web Services Monitoring: An OWL-S Based Approach[A]. In Proceedings of the 41st Annual Hawaii International Conference on System Sciences[C]. Washington: IEEE Computer Society, IEEE Computer Society, 2008. 313.

[10] 张成文,苏森,陈俊亮.基于遗传算法的 QoS 感知的 Web 服务选择[J].计算机学报,2006,29(7):1029 – 1037.
Zhang Cheng-Wen, Su Sen, Chen Jun-Liang. Genetic Algorithm on Web Services Selection Supporting QoS[J]. Chinese Journal of Computers, 2006, 29(7): 1029 – 1037. (in Chinese)

[11] Web Services Policy 1. 5-Framework[EB/OL]. http://www.w3.org/TR/ws-policy/, 2007-09-04.

[12] Peter Brittenham. Understanding the WS-I Test Tools[EB/OL]. http://www.ibm.com/developerworks/webservices/library/ws-wsitest/, 2003-11-18.

[13] 蔡维德,白晓颖,陈以农.浅谈深析面向服务的软件工程[M].北京:清华大学出版社,2008.

[14] Oracle 业务活动监视[EB/OL]. http://www.oracle.com/technology/global/cn/products/integration/bam/index.html, 2008.

作者简介:

刘永利　男,1980 年 11 月出生,现为清华大学计算机科学与技术系硕士研究生.主要研究领域为软件工程.
E-mail: liuyongli07@mails.tsinghua.edu.cn

白晓颖　女,1973 年 11 月出生,博士,现为清华大学计算机科学与技术系副教授.主要研究领域为软件工程.
E-mail: baixy@tsinghua.edu.cn