doi:10.11959/j.issn.1000-436x.2016026

# 基于语义扩展类型论的云服务替换性判定研究

王先清[1,2]，黄昌勤[1,3]，罗旋[1]，聂瑞华[1]，汤庸[1]，梅晓勇[1]

（1. 华南师范大学教育信息技术学院，广东 广州 510631；
2. 广东科学技术职业学院艺术设计学院，广东 广州 510640；3. 浙江大学电子服务研究中心，浙江 杭州 310027）

**摘 要**：云计算环境下服务的动态性和易失效性是云应用的重要挑战，服务替换是其主要对策和关键研究问题。在类型论的支持下提出了一种新的云服务替换判定方法，该方法首先对会话类型论进行语义扩展以建模云服务行为，设计了典型云服务 QoS 类型实现服务质量判断，然后构造了语义会话类型和 QoS 类型的各子类型规则，最后以此完成了服务一致性和上下文兼容性命题判定与实施。通过应用判定实例展示和实验效果分析，表明该判定方法可行，并能为组合服务应用带来更高的执行成功率。

**关键词**：云服务；类型理论；会话类型；服务替换

**中图分类号**：TP393 **文献标识码**：A

# Determining substitutability of cloud services supported by semantically extended type theory

WANG Xian-qing[1,2], HUANG Chang-qin[1,3], LUO Xuan[1], NIE Rui-hua[1],TANG Yong[1], MEI Xiao-yong[1]

(1. School of Information & Technology in Education, South China Normal University, Guangzhou 510631, China;
2. School of Art & Design, Guangdong Institute of Science & Technology, Guangzhou 510640, China;
3. E-Service Research Center, Zhejiang University, Hangzhou 310027, China)

**Abstract:** In cloud environments, the high dynamics and more service failures were great obstacles to cloud applications, service substitution was a key research issue and also was a main solution to these challenges. A method of determining substitutability of cloud services was proposed using type theory, in which session types were semantically extended for modeling the behaviors of cloud service, QoS such as price, reliability were introduced as QoS type, and a series of subtyping rules were constructed for SST and QoST. After that, determining consistency and context compatibility of services were put into practice. The method was proved feasibly by a case determining, and the experimental results show that it brings higher success rate of execution.

**Key words:** cloud service, type theory, session types, service substitution

## 1 引言

在云计算环境下，服务作为一种基本的计算实体在云应用中起着至关重要的作用[1]。在实际的业务应用中，由于云平台及应用的可伸缩性、可移动云服务本身的不确定性，导致服务呈现高度动态

**收稿日期**：2015-03-09；**修回日期**：2015-12-15
**通信作者**：黄昌勤，cqhuang@zju.edu.cn
**基金项目**：国家自然科学基金资助项目(No.61370229, No.61370178)；国家科技支撑计划基金资助项目(No.2013BAH72B01)；教育部-中国移动基金资助项目（No.MCM20130651）；广东省自然科学基金资助项目（No.S2013010015178）；广东省科技计划基金资助项目（No.2014B010103004, No.2014B010117007, No.2015A030401087, No.2015B010110002）；广东省教育厅科技创新基金资助项目（No.2012KJCX0037）；广州市科技基金资助项目（No.2014Y2-00006）

**Foundation Items:** The National Natural Science Foundation of China (No.61370229, No.61370178), The National Key Technology R&D Program of China(No.2013BAH72B01), The MOE-CMCC Research Fund(No.MCM20130651), The Natural Science Foundation of Guangdong Province (No.S2013010015178), The S&T Projects of Guangdong Province (No.2014B010103004, No.2014B010117007, No.2015A030401087, No.2015B010110002), The S&T Project of DEGP (No.2012KJCX0037), The S&T Project of Guangzhou Municipality (No.2014Y2-00006)

性，服务 QoS 变化甚至服务失效成为云应用面临的严峻挑战[2]；同时，在市场机制激励下云服务数量不断增加、功能演进，服务存在版本更新需求。为了提高云服务应用（尤其是组合云服务）的正确性、及时性和可靠性，服务动态替换成为领域关注的焦点问题之一。

服务替换的核心在于服务选择与语义验证，主要包括服务描述、服务选择和替换验证。在服务描述方面，传统方法基于 UDDI 技术，通过输入输出参数格式与关键字来描述服务的语法信息，后续研究在关键词匹配的基础上引入本体论进行描述，以弥补了语义描述方面的缺陷，但对服务行为的考虑仍存不足[3]。一些研究将形式化的方法运用于服务建模，使用 Petri 网、扩展的动态描述逻辑 EDDL(X)、π 演算等对服务进行形式化建模，并借助动作过程断言等来刻画动作的执行过程[4~6]。该类形式化方法能够较好刻画服务的行为，但对服务 QoS 约束的考虑存在不足，不能较好地支持服务在动态与全局环境下的替换。在服务选择方面，较早的服务选择多以静态的服务功能需求为目标，通过参数格式匹配进行语法级别的服务选择，后续研究尝试将语义信息匹配引入选择过程，如基于联接本体、事务级别、着色 Petri 网模型等，它们或设计选择机制或提出选择算法，并多支持语义的推理与匹配[7~9]。上述研究虽提高了服务选择的准确率，但在语义信息不足的情况下，会出现较大的不确定性。在服务的替换性验证方面，目前研究主要利用进程代数、π 演算等理论，验证内容涵盖上下文兼容性、行为的一致性等方面，相关研究有的基于形式化工具，有的借助自定义的消解规则或自动机[10~13]。这些方法因使用了完备的理论方法，可以较好地保障服务替换的准确性，然而验证的复杂度很高，在服务较多的应用场景下可用性明显受限，且较难实现验证的自动化。云计算环境下，云服务数量巨大且高度动态，需要寻找新方法支持高效、准确和自动化的服务替换。

类型论的形式化系统完备，能有效支持服务的语法和语义描述，且具备相对较低的计算复杂度。在现有研究中，典型情况是扩展马丁洛夫类型理论，补充新类型，从而对云服务的交互行为进行有效表征[14~16]。部分研究基于扩展会话类型实现服务建模，以此为基础完成服务等价性验证[17,18]。由此可见，现有相关研究主要考虑服务的语法部分，但描述精确度有待提高，同时，同一形式化体系中尚未见对云服务多维 QoS 关注的服务替换研究成果。

综上，传统形式化方法在描述和判定方面，要么因异常复杂而失效，要么存在不能兼顾服务行为功能和非功能需求。鉴于类型论对服务相关语法和语义等具有良好的描述能力，本文对其进行语义拓展和规则设计以有效解决云服务替换性判定。

## 2　语义会话类型

### 2.1　会话类型及其语义扩展

会话类型是一种采用类型论通用方法构造的类型，常用于描述服务的动态行为，因其对动态行为描述的优势，可用于高度动态的云服务的建模，且通过类型间的子类型判定，来推导会话间的子类型关系，进而基于子类型关系实现服务一致性和兼容性命题的推导，并最终完成可替换性判定。

传统的会话类型在服务调用参数的构造上，多基于语法而未考虑语义，因而在 Reception 类型（接收类型，表示服务接收其他服务消息的接口）和 Sending 类型（发送类型，表示服务向其他服务发送消息的接口）的子类型判定上存在不足，易导致替换性方案查全率的降低；同时，在 Branching 类型（分支类型，表示服务的内部流程分支，接受其他服务对分支的选择）和 Selection 类型（选择类型，表示服务对外的选择操作，该选择对应其他服务相应的分支）的操作语义标签上语义描述较为模糊。本文构造领域本体，利用本体的语义推理和匹配优势完成服务消息的语义级支持。不失一般性，本研究将操作语义归为“动作”和“对象”2 个部分，利用本体支持下的服务操作语义进行类型论扩展，以改变传统类型论中的简单语义标签状况，其基本策略是：将 Selection/Branching 中的语义标签替换成构造类型 Op，将 Reception/Sending 中的消息类型 Message 进行构造，形成语义会话类型（简称 SST），从而基于精确语义促使服务替换性得以更加可靠的验证。

SST 对传统会话类型的扩展内容如下

$$T_{\text{selection}} ::= +\left\{\left\langle op_k : T_k \right\rangle_{k \in K}\right\}$$

$$T_{\text{branching}} ::= \&\left\{\left\langle op_k : T_k \right\rangle_{k \in K}\right\}$$

$$T_{\text{sending}} ::= !\left\langle Message \right\rangle ; T$$

$$T_{\text{reception}} ::= ?\left\langle Message \right\rangle ; T$$

上述形式中，各扩展后的子类型表示为 $T_{类型名}$。对于某个具体的服务实例而言，扩展后的会话类型亦可基于上述各子类型来描述和判断服务的动态行为，其中，$T_{selection}$ 类型表示该服务可对外发起 $k$ 种调用，具体调用行为由对应类型 $T_k$ 表示；$T_{branching}$ 类型表示该服务可以接受其他服务的 $k$ 种调用，具体被调用的行为由对应 $T_k$ 表示；$T_{sending}$ 类型表示该服务向外发送消息，具体消息细节由 Message 表示，发送完毕后服务行为由 $T$ 表示；$T_{reception}$ 类型表示该服务对外接收消息，具体消息细节由 Message 表示，接收完毕后服务行为由 $T$ 表示。对于 Message、Op 的构造，扩展部分需要利用到本体。由于本体是一个与领域紧密的概念，本文以服务组合研究中常见的旅行服务领域为例，进行领域本体构造以支持后续的建模与构造。

### 2.2 领域本体的构造

领域本体由领域内的共享概念及概念间关系组成，因此，将领域本体定义为二元组 Ontology={Concepts×Relationships}set。为了有效构造 Message 类型，拟将旅行服务常见的消息概念抽取出来，将其完整建立在旅行服务本体中（如图 1 所示）；对于 Op 类型构造，将领域本体中概念的常用的操作动作进行抽取，形成旅行服务操作动作的枚举集合 Action={Cancel,Book,Refund,⋯}set。

根据已有研究，引入依赖记录类型[19] (DRT, dependent record type)对本体进行描述。DRT 的表现形式为<$l_1$:$T_1$,$l_2$:$T_2$,⋯,$l_n$:$T_n$>，$r.l$ 用于选取标签为 $l$ 的记录。

以旅行服务本体中的 Ticket 概念为例，利用 DRT 对其描述如下

Ticket=<fare:Integer,arrivalPlace:Address,departurePlace: Address, arrivalTime: Time, departureTime: Time, vehicle:Vehicle>

Address=<country: String, provence: String, city: String,district:String,street:String,number:Integer>

Vehicle=<kind:String,owner:String, capacity: Integer>

Time=<year:Integer,month:Integer,date:Integer,hour:Integer,minute:Integer,second:Integer>

通过选取操作可获取 Ticket 中的某个属性或子概念，例如 Ticket.fare=Integer。通过递归使用 DRT 的选取操作，可得 Ticket.vehicle.kind =String。为后续消息间子类型的判定，需要构造概念间子类型关系。分析可知，服务消息中的参数可能属于某个概念的范畴，但不一定包含该概念在本体树中所有的子节点。以 Vehicle 范畴为例，若 $A$、$B$ 都属于 Vehicle 范畴，$A$=<kind:String, owner:String>，$B$=<owner:String>，可知 $A$ 具备 $B$ 的所有语义信息，且更多，则可判定 $A<:B$。利用形式化方法表述该概念类型的子类型规则为

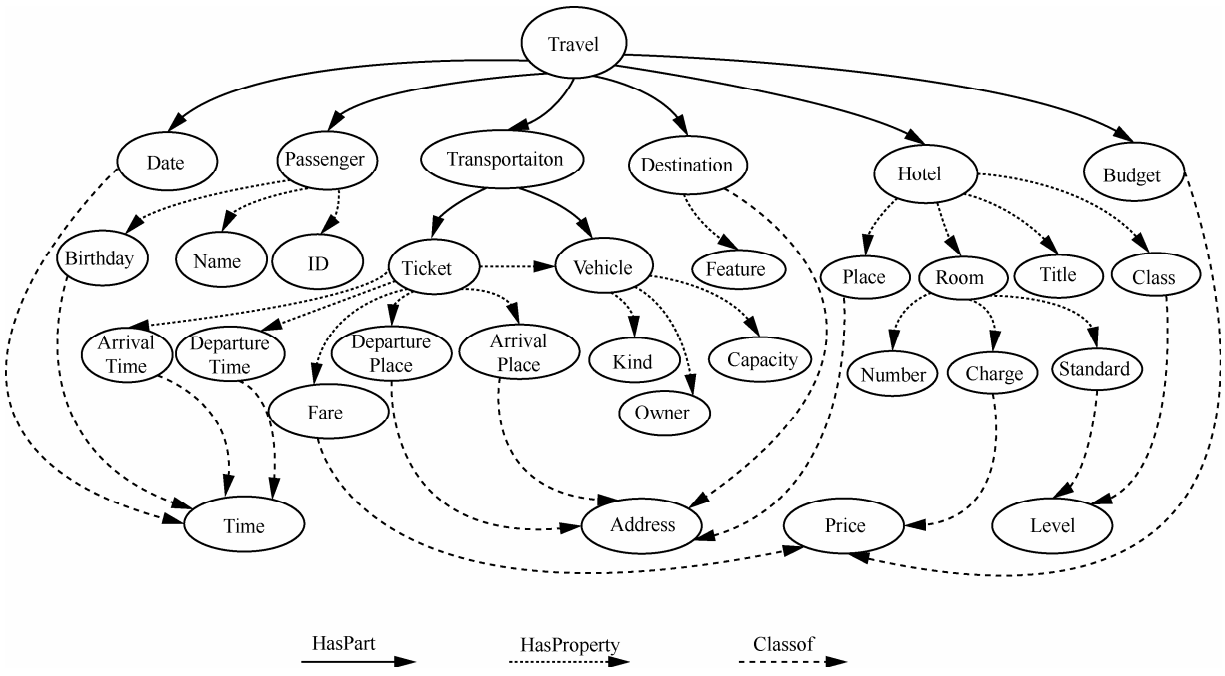


图 1 旅行服务本体

$$C_1 =< a_1 : S_1, a_2 : S_2, \cdots, a_n : S_n >$$
$$C_2 =< b_1 : T_1, b_2 : T_2, \cdots, b_m : T_m >$$
$$\exists a_x \, \forall b_y \, ( \, (b_y = a_x) \wedge ((T_y <: S_x) \vee (T_y = S_x)))$$
$$\frac{[x = 1, 2, \cdots, n][y = 1, 2, \cdots, m]}{C_1 <: C_2}$$

若 $C_1$、$C_2$ 均为本体中的某个概念，$C_2$ 中的每个标签都能在 $C_1$ 中找到对应标签，且利用相同标签取出的类型中，$C_2$ 取出的类型是 $C_1$ 取出类型的子类型或相等类型。更具体地，即 $C_1$ 包含 $C_2$ 的所有语义，要么 $C_1C_2$ 完全一样，要么 $C_1$ 更宽泛 $C_2$ 更具体。例如“火车票”是“票”的子类型，因为“火车票”相比“票”至少多 Ticket.vehicle.kind 语义。显然，当用户请求的是无种类限制的票时，火车票也能满足用户的需求。对于旅行业务中的其他概念，也可以建立类似的子类型规则，最终形成本体内的子类型规则库，以支持概念实体间的子类型判定。为了一致性需要，将概念实体间的子类型规则统称为 concept-subtyping rules，简称 Concept-S。

### 2.3　消息类型的构造

服务消息即服务的输入与输出（也称 IO），即 WSDL 中的 Message。在最典型应用场境——服务组合中，原子服务之间的交互，以及组合服务与用户间的交互，都可以看做消息的传递。在类型论的服务支持中，对服务消息进行类型论建模，是服务组合方案搜索、服务替换等相关操作的基础。基于 MLTT(马丁洛夫类型理论)中无序列表类型、迪氏积类型、枚举类型的构造方法，消息类型的构造如下

$$Message = List(Concept \times MsgBody \ ) \text{set}$$
$$MsgBody = List\left(ComplexType\right) \text{ set}$$
$$ComplexType = List\left(Concept \times SimpleType\right) \text{ set}$$
$$SimpleType = \left\{Integer, Long, String \cdots\right\} \text{ set}$$

在实际使用中，MsgBody 类型的实例中可能存在概念相同的元素。为避免在比较中混淆，可将其父节点加入命名。例如将 Hotel 中的 Place 标记为 Hotel-Place 以避免与其他 Place 混淆。旅行业务中的一个有关 Hotel 的消息类型可能表达如下

{<Hotel,{<Hotel-Title,String>,<Room-Number, String>}>,<Hotel,{<Hotel-Title,String>,<Room-Number, String>}>}

引入笛氏积集合的选取子 *fst* 和 *snd*，方便在推导中取出笛氏积集合典则元的各个部分，*fst*(<*a*, *b*>)=*a*，*snd*(<*a*,*b*>)=*b*。为支持后续的子类型命题的证明，定义笛氏积子类型规则（Decare-S）为

$$A =< x_a, y_a >$$
$$B =< x_b, y_b >$$
$$\frac{x_a < \ : x_b \wedge y_a < \ : y_b}{A < \ : B}$$

并在前文 Concept-S 基础上构造服务消息的子类型规则，子类型规则构造如下。

**规则 1**　SimpleType-S 简单数据类型的子类型规则

$$simpleType_1, simpleType_2 \in \left\{Integer, Long, String \cdots\right\}$$
$$simpType_1 \mapsto simpleType_2$$
$$\frac{/* \text{运用XSD-C} */}{simpleType_1 < \ : simpleType_2}$$

**规则 2**　ComplexType-S 复杂数据类型的子类型规则

$$complexType_1, complexType_2 \in List(Concept \times SimpleType)$$
$$ct_1 = l_1.a \ [l_1 \in List(Concept \times SimpleType),$$
$$a \in Concept \times SimpleType]$$
$$ct_2 = l_2.b \ [l_2 \in List(Concept \times SimpleType),$$
$$b \in Concept \times SimpleType]$$
$$\frac{\begin{array}{l} a < \ : b \ \ /* \text{运用Decare-S,Concept-S,SimpleType-S}*/ \\ listrec(l_1, c, e) <: listrec(l_2, c, e) \ /* \\ \text{运用Decare-S,Concept-S,SimpleType-S}*/ \end{array}}{complexType_1 <: complexType_2}$$

**规则 3**　MsgBody-S 消息体类型的子类型规则

$$msgBody_1, msgBody_2 \in List(ComplexType)$$
$$msgBody_1 = l_1.a \ [l_1 \in List(ComplexType), a \in ComplexType]$$
$$msgBody_2 = l_2.b \ [l_2 \in List(ComplexType), b \in ComplexType]$$
$$\frac{\begin{array}{l} a < \ : b \ \ /* \text{运用ComplexType-S}*/ \\ listrec(l_1, c, e) < \ : listrec(l_2, c, e) \ \ /* \text{运用ComplexType-S}*/ \end{array}}{msgBody_1 < \ : msgBody_2}$$

**规则 4**　Message-S 服务类型的子类型规则

$$message_1, message_2 \in List(Concept \times MsgBody)$$
$$message_1 = l_1.a[l_1 \in List(Concept \times MsgBody),$$
$$a \in Concept \times MsgBody]$$
$$message_2 = l_2.b[l_2 \in List(Concept \times MsgBody),$$
$$b \in Concept \times MsgBody]$$
$$\frac{\begin{array}{l} a < \ : b \ \ /* \text{运用Decare-S,Concept-S,MsgBody-S}*/ \\ listrec(l_1, c, e) < \ : listrec(l_2, c, e) \ \ /* \text{运用Decare-S,} \\ \text{Concept-S,MsgBody-S}*/ \end{array}}{message_1 < \ : message_2}$$

规则 1 中，所使用的 XSD-C 为基本数据类型

间的强制转换规则。根据 XSD 标准数据类型的定义，如果数据类型 $A$ 可以强制转换为数据类型 $B$，且不丢失精度不改变意义，则记为 $A \mapsto B$。例如 Integer 可以转化为 Float 类型，XML 可以转化为 String 类型，且不丢失其精确度，则 Integer $\mapsto$ Float，XML $\mapsto$ String。由上述规则可知，只要有充足的概念子类型规则支持，即可通过规则推导得出消息之间的准确子类型关系。

### 2.4 操作类型的构造

操作类型 Op 表达原子服务对外发起调用、或受到外界调用时的操作语义，通常作为成员出现在 Selection 类型和 Branching 类型中。一个操作的语义通常包括{发起者，动作，对象}3 部分，发起者对本研究 SST 的操作语义没有影响，在构造中省略。因此，本研究将 Op 构造为动作与对象的笛氏积，其中，动作由动作集合中元素描述，而对象则利用本体中的概念描述。构造 $Op=Action\times Concept$ set，则 Op 类型的子类型规则 Op-S 定义如下

$$\frac{op_1 =< action_1, concept_1 > \quad op_2 =< action_2, concept_2 > \quad action_1 = action_2 \quad concept_1 < : concept_2}{op_1 < : op_2}$$

## 3 云服务 QoS 类型

### 3.1 云服务 QoS 类型的构造

云服务 QoS 是云服务的非功能性因素，亦即服务质量，是服务能否满足云应用需求的重要标准，在高度动态的云计算环境下尤显重要。与第 2 节类似，建立 QoS 本体，以便利用类型论建模 QoS，支持服务替换命题的证明。云服务具有资源绑定高度动态、宿居环境多样、按需付费等特性，需选取相应的特征性指标来建模云服务 QoS。本文参照已有研究，仅选取价格、可靠性和云服务等级 3 项指标做针对性地分析与建模。

与第 2 节中类似，对 QoS 类型（后称 QoST）使用 DRT 进行描述，如图 2 所示。

QoS = <price: Price, reliability: Reliability, service Level:ServiceLevel>

Price=<cellingPrice:Integer,bottomPrice:Integer, quotePrice:Integer>

Reliability = <cellingReliability: Integer, bottomReliability: Integer,quoteReliability:Integer>

ServiceLevel=<currentLevel:Integer,peakLevel: Integer,bottomLevel:Integer>

其中，Price 为服务调用需要支付的费用，ServiceLevel 为云提供商的评级，两者以数值方式参与后续 QoS 计算。价格由服务提供商提供，包括调用服务的最低价格和最高价格，对调用者来说，通常只有报价可见。云服务的评级则由用户的评分累积计算形成。价格与服务评级为云服务的特征性因素，获取较为简单，此处不做详细探讨，将重点分析云服务可靠性的计算方法。

云服务环境中，原子服务在可靠性方面具有不确定性，它由服务宿居硬件可靠性、自身软体可靠性以及涉外通信链路的可靠性共同决定。在一般假设前提下（即原子服务宿居节点的各硬件、宿居的软件体、基于链路的对外数据交换的故障、失效等事件相互独立，宿居节点各硬件和通信链路上的故障或失效过程服从齐次泊松分布等），用 $P_{sr}$(cpu)、$P_{sr}$(mem)、$P_{sr}(C)$和 $P_{sr}(E)$,分别表示服务宿居节点的 CPU、内存、网络通信和软体本身的可靠性，则原子服务的可靠性计算为 $P_{sr}(\text{cpu})P_{sr}(\text{mem})P_{sr}(C)P_{sr}(E)$。在可移动环境中，$P_{sr}(C)$较为复杂，如果用 $Int_L$ 表示原子服务宿居设备节点与对外通信节点间的通信链路的失效强度，用 $A$ 表示通信平均时长，$P_{sr}(C)=\exp(-Int_L.A)$。

### 3.2 QoST 的子类型规则

考虑应用需求，在服务替换中存在服务选择与计算过程，其中，服务 QoS 各分量存在大小关系，

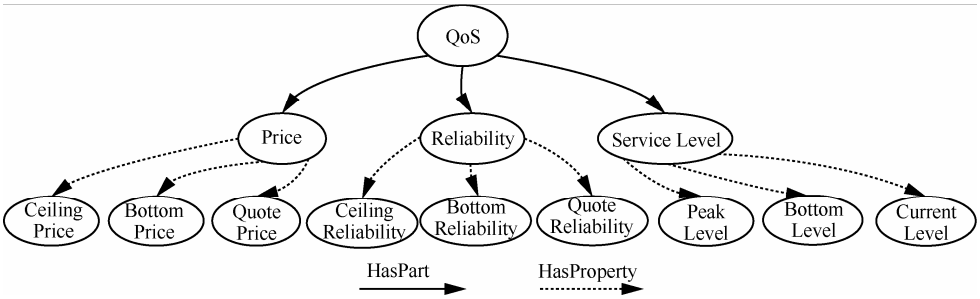


图 2 云服务 QoS 本体

将用于数量比较。根据可靠性、价格、服务评级等因素对服务替换的影响，定义如下子类型基本判定规则——QoST 子类型规则(QoST-S)。

$$\frac{\begin{array}{l}A,B\in QoST\\ A.price.quotePrice\leqslant B.price.quotePrice\\ A.reliability.quoteReliability\geqslant B.reliability.quoteReliability\\ A.serviceLevel.currentLevel\geqslant B.serviceLevel.currentLevel\end{array}}{A<:B}$$

在服务 QoS 比较的应用场景中，经常出现某个服务的某些 QoS 指标较高，而其他 QoS 指标较低，但综合 QoS 仍旧较高的情况。因此 QoS 类型的子类型应具备指标间协商的功能。实际应用场景中，如果根据用户需求可以进行某些 QoS 指标的协商，则可将相应协商规则加入 QoST-S 规则中，参与服务子类型的推导。本文考虑到云服务按需付费的特征性因素，针对价格可协商的特性，定义了协商表达式，并构造了如下 QoS 协商规则。其中，协商表达式 $A\downarrow(X,\Delta)^n$ 含义为，对 $A$ 进行 $n$ 次协商，每次协商使 $A$ 的属性 $X$ 下降（若箭头为↑则为上升）$\Delta$（步长），并最终返回协商完成（各项指标发生相应变化）之后的 $A$。

**规则 5**　QoS-(Reliability-lowerPrice)-N-S 服务 QoS 协商规则

$$\frac{\begin{array}{l}A,B\in QoST\\ A.price.quotePrice\leqslant B.price.quotePrice\\ A.reliability.quoteReliability<B.reliability.quoteReliability\\ A.serviceLevel.currentLevel\geqslant B.serviceLevel.currentLevel\\ \exists n(A\downarrow(price.quotePrice,\Delta)^n.reliability.quoteReliability\\ \quad>B.reliability.bottomReliability)[n=1,2,\cdots]\end{array}}{A<:B}$$

**规则 6**　QoS-(ServiceLevel-lowerPrice)-N-S 服务 QoS 协商规则

$$\frac{\begin{array}{l}A,B\in QoST\\ A.price.quotePrice\leqslant B.price.quotePrice\\ A.reliability.quoteReliability\geqslant B.reliability.quoteReliability\\ A.serviceLevel.currentLevel<B.serviceLevel.currentLevel\\ \exists n(A\downarrow(price.quotePrice,\Delta)^n.serviceLevel.currentLevel\\ \quad>B.serviceLevel.bottomLevel)[n=1,2,\cdots]\end{array}}{A<:B}$$

**规则 7**　QoS-(Reliability-higherPrice)-N-S 服务 QoS 协商规则

$$\frac{\begin{array}{l}A,B\in QoST\\ A.price.quotePrice\leqslant B.price.quotePrice\\ A.reliability.quoteReliability<B.reliability.quoteReliability\\ A.serviceLevel.currentLevel\geqslant B.serviceLevel.currentLevel\\ \exists n(A\uparrow(price.quotePrice)^n.reliability.quoteReliability\\ >B.reliability.quoteReliability)[n=1,2,\cdots]\\ [A.price.quotePrice\leqslant B.price.ceilingPrice]\end{array}}{A<:B}$$

**规则 8**　QoS-(ServiceLevel-higherPrice)-N-S 服务 QoS 协商规则

$$\frac{\begin{array}{l}A,B\in QoST\\ A.price.quotePrice\leqslant B.price.quotePrice\\ A.reliability.quoteReliability\geqslant B.reliability.quoteReliability\\ A.serviceLevel.currentLevel<B.serviceLevel.currentLevel\\ \exists n(A\uparrow(price.quotePrice,\Delta)^n.serviceLevel.currentLevel\\ \quad>B.serviceLevel.currentLevel)[n=1,2,\cdots]\\ [A.price.quotePrice\leqslant B.price.ceilingPrice]\end{array}}{A<:B}$$

**规则 9**　QoS-all-N-S 服务 QoS 协商规则

$$\frac{\begin{array}{l}A,B\in QoST\\ \alpha\dfrac{A.price.quotePrice}{B.price.quotePrice}+\\ \beta\dfrac{B.reliability.quoteReliability}{A.reliability.quoteReliability}+\\ \gamma\dfrac{B.serviceLevel.currentLevel}{A.serviceLevel.currentLevel}>1\\ [\alpha+\beta+\gamma=1]\end{array}}{A<:B}$$

## 4　基于 SST 和 QoST 的云服务建模

### 4.1　基于类型论的云服务建模

从替换的视角来看，云服务由服务的 QoS 及服务的功能构成，基于语义会话类型 SST 与 QoST，将云服务建模为 QoS 与服务功能的笛氏积 $Service=QoST\times ServiceProtocol$ set。其典则元（云服务实例）包含 2 部分，分别为 QoST 的实例与 SST 描述的 *ServiceProtocol* 实例。

### 4.2　云服务类型的子类型规则

构造服务的子类型规则，需在 QoST-S 的基础上，继续构造 SST 所描述的服务功能类型的子类型规则。服务功能由多种类型构造而成，因此服务功能的子类型判定规则也需要多种子类型规则同时

支持。引入会话类型的标准环境(well-formed environments)$\Gamma$，参照传统会话类型的子类型规则，构造如下 SST 中的子类型规则，如表 1 所示。

**表 1 SST 的子类型规则**

| 规则名 | 规则体 |
|---|---|
| End-S | $\Gamma \triangleright \text{end} <: \text{end}$ |
| Reception-S | $\dfrac{\Gamma \triangleright T <: S}{\Gamma \triangleright ?\langle msg_t \rangle; T <: ?\langle msg_s \rangle; S}$ |
| Sending-S | $\dfrac{\Gamma \triangleright T <: S}{\Gamma \triangleright !\langle msg_t \rangle; T <: !\langle msg_s \rangle; S}$ |
| Branching-S | $\dfrac{\forall k \in K \subseteq J \quad \Gamma \triangleright op_k <: op_j, T_k <: S_j}{\Gamma \triangleright \&\left\{\langle op_k : T_k \rangle_{k \in K}\right\} <: \&\left\{\langle op_j : S_j \rangle_{j \in J}\right\}}$ |
| Selection-S | $\dfrac{\forall j \in J \subseteq K, \Gamma \triangleright op_j <: op_k, T_k <: S_j}{\Gamma \triangleright +\left\{\langle op_k : T_k \rangle_{k \in K}\right\} <: +\left\{\langle op_j : S_j \rangle_{j \in J}\right\}}$ |
| Recursion-S | $\dfrac{\Gamma \triangleright T\left\{\dfrac{\mu x.T}{x}\right\} <: S}{\Gamma \triangleright \mu x.T <: S} \quad \dfrac{\Gamma \triangleright S <: T\left\{\dfrac{\mu x.T}{x}\right\}}{\Gamma \triangleright S <: \mu x.T}$ |

分析上述子类型规则可发现，子类型在接收消息时，需要的消息在语义范围上更小；在发送消息时，发出的消息在语义涵盖范围上更大；在向外发出选择时，选择更多且该选择的语义涵盖更大；自己提供的分支更少，且分支的语义涵盖更精确。当会话子类型在替换原会话时，接受消息和分支操作的接口语义涵盖更广，意味着在被调用时能够保证原功能的实现；发送消息和选择操作则语义涵盖范围更窄，意味着在调用其他服务时，也能保证其他服务功能的实现。因此子类型在功能上可以安全的替换父类型。

在服务 QoST-S 子类型命题证明的基础上，构造服务的子类型规则（Service-S）。

$$\frac{\begin{array}{l} A = \langle QoS_a, ServiceProtocol_a \rangle \in Service \\ B = \langle QoS_b, ServiceProtocol_b \rangle \in Service \\ fst(A) <: fst(B) \\ snd(A) \in ServiceProtocol <: snd(B) \in ServiceProtocol \end{array}}{A <: B}$$

## 5 云服务替换性判定

在动态化的云计算环境中，服务替换是一个涉及行为功能语义、服务 QoS 及其上下文等因素的复杂过程，这也导致了服务替换性判定具有挑战性。由于类型论对服务相关语法和语义等具有良好的描述能力，可成为服务替换性判定的支持工具。本研究将基于上述扩展类型论支持下的云服务模型及相关规则，解决其云服务替换判定。

### 5.1 替换性命题构造与服务一致性判定

根据会话类型的相关定义，云服务替换性命题可以等价为云服务在类型理论体系中的子类型命题，由此依据类型论，服务替换性命题构造如下

$$\frac{A \text{ can replace } B \Leftrightarrow A <: B}{\begin{array}{l} fst(A) <: fst(B) \\ snd(A) <: snd(B) \end{array}}$$

服务一致性和上下文兼容性是云服务替换的可靠保障。服务一致性判定指从行为功能和 QoS 这 2 个方面进行比较，以判定原服务能否被新服务替换进行业务操作。根据前述云服务描述，显然可将服务一致性判定分解为行为功能和QoS这2部分的子类型判定命题。基于会话类型论，本研究将利用各子类型规则对命题进行判定（或称证明）。对于服务 QoS 部分命题（即 $fst(A)$与 $fst(B)$间子类型命题），由于云环境下具有波动的可靠性问题、价格等多重因素，采用基本判定和协商判定并存的策略。基本判定利用 QoST-S 规则进行，协商判定则可将相应协商规则加入判定过程。服务行为功能的子类型命题（即 $snd(A)$与 $snd(B)$间子类型命题）可据 SST 子类型判定规则，将原命题分解为各个不同成员类型之间的子类型命题进行判定。命题分解证明（判定）方法如下

$$\frac{\begin{array}{l} snd(A) = ?\langle message_1 \rangle; +\{op_a : !\langle Message_2 \rangle; \text{end}\} \\ snd(B) = ?\langle message_3 \rangle; +\{op_b : !\langle Message_4 \rangle; \text{end}\} \\ snd(A) <: snd(B) \end{array}}{\begin{array}{l} \text{由Reception}-\text{S规则分解为} \\ message_1 <: message_2 \cdots \\ +\{op_a : !\langle Message_2 \rangle; \text{end}\} <: +\{op_b : !\langle Message_4 \rangle; \text{end}\} \\ \text{由Message-S规则,Selection-S规则分解为} \\ \cdots \end{array}}$$

综上所述，基于类型论完成替换性命题构造，可为替换性判定带来形式化支持，其中，服务一致性命题判定完全可利用本研究定义的子类型规则、按照上述方法得到递归式证明（判定）。

### 5.2　替换性命题的功能兼容性判定

由于云服务依据服务上下文而动态变化，在云服务替换中服务的上下文兼容尤显重要。因此，替换性命题需要其兼容性的判定支持。由于服务 QoS 不具有兼容性特性，因此在替换性命题的兼容性判定中，仅需要判定服务行为功能的兼容性即可。

根据类型论，其类型间的兼容可构造为命题 $T_1 \to T_2$，则基于子类型规则证明兼容性可转化为

$$\frac{T_1 \to T_2 \quad T_3 <: T_2}{T_1 \to T_3} \text{ 或者 } \frac{T_1 \to T_2 \quad T_3 <: T_1}{T_3 \to T_1}$$

因此，服务功能兼容性判定需藉助各子类型规则支持，最终证明上式成立。根据会话类型中镜像类型可知，判定过程中必须处理兼容性成员类型为 Sending、Reception、Selection、Branching，前两者包含消息收发的兼容性，后两者包含流程分支的兼容性。根据会话类型的语法和语义，可分别构造兼容性的判定规则，然后实施规则推导，最终完成命题判定。限于篇幅，仅以 2 个例子说明判定过程。

1) Sending-Reception 中 reception 类型的替换

前提条件为

$T_1 = !\langle msg_1 \rangle \in T_{\text{sending}}$

$T_2 = ?\langle msg_2 \rangle \in T_{\text{reception}}$

$T_3 = ?\langle msg_3 \rangle \in T_{\text{reception}}$

兼容性判定规则为

$T_1 \to T_2 \Leftrightarrow msg_2 <: msg_1$

目标命题证明为

$$\frac{\dfrac{T_1 \to T_2}{msg_2 <: msg_1} \quad \dfrac{T_3 <: T_2}{msg_3 <: msg_2}\ \text{由Reception-S规则}}{msg_3 <: msg_2 <: msg_1}$$

$$T_1 \to T_3$$

如果 $T_3 <: T_1$，证明方法类似。

2) Selection-Branching 中 branching 类型的替换

前提条件为

$T_1 = +\{\langle op_l : T_l \rangle_{l \in L}\} \in T_{\text{selection}}$

$T_2 = \&\{\langle op_m : T_m \rangle_{m \in M}\} \in T_{\text{branching}}$

$T_3 = \&\{\langle op_n : T_n \rangle_{n \in N}\} \in T_{\text{branching}}$

兼容性判定规则为

$T_1 \to T_2 \Leftrightarrow \forall m \in M \subseteq L$

$(\Gamma \rhd op_m <: op_l \wedge T_m <: T_l)$

目标命题证明

$$\frac{\dfrac{T_1 \to T_2}{\forall m \in M \subseteq L\ (\Gamma \rhd op_m <: op_l \wedge T_m <: T_l)} \quad \dfrac{T_3 <: T_2}{\text{由Branching-S规则}}}{\dfrac{\forall n \in N \subseteq M\ (\Gamma \rhd op_n <: op_m \wedge T_n <: T_m)}{\forall n \in N \subseteq L\ (\Gamma \rhd op_n <: op_l \wedge T_n <: T_l)}}$$

$$T_1 \to T_3$$

如果 $T_3 <: T_1$，证明方法类似。

对于其他兼容性成员类型，可采用类似子类型规则予以判定。由此，在服务行为功能的兼容性可得保证。

### 5.3　面向云服务组合的服务替换

在云环境中，服务组合是服务完成业务逻辑流程编制的基本形式。受上下文影响，云服务失效、低 QoS 检出等成为常态；同时，基于市场竞争机制的云服务演进，使得服务版本的判定与置换也成必然。因此，为了有效实现组合服务的云应用，必须解决云服务替换这个核心问题。本研究以云服务组合为背景考察云服务替换的具体实施。

鉴于云计算环境的特殊性，在云服务组合中服务替换改变依据传统服务检查等常规触发方法，将以云服务上下文变化为触发事件；同时在替换判定中，应用前述基于类型论的行为功能语义、QoS 及功能兼容性判定，必要时启用 QoS 协商。候选置换服务获取算法如图 3 所示。

```
算法：候选置换服务集的获取算法 Replacement_Algorithm
输入：需要进行替换的服务 WS 和可用云服务 UDDI
输出：备选的新服务组合 List(WSr)
Replacement_Algorithm(WSa)
Begin
1) While 云服务上下文变化 Do
2) 触发到服务检测事件；
3) IF 检测到组合服务中某 WS 失效 OR 其 QoS 指标<预期阈值 Then//进入替换判定
4) Flag=0；
5) For WS in UDDI，利用 SST 规则系统和 QoST 基本规则系统，从一致性及其兼容性判定 WS<:WSa 命题；
6) 如果推导得出 WS<:WSa，则将 WS 作为 WSr 加入组合服务候选集 List(WSr)
7) End For;
8) IF |List(WSr)|=Φ AND Flag=0 Then     //进入 QoS 协商
9) Flag=1；
10) QoST 协商规则系统->QoST 基本规则系统
11) Goto 5)；
12) End IF；                  //结束 QoS 协商
13) 返回 List(WSr)；
14) End IF                    //进入替换判定
15) End While
End
```

图 3　候选置换服务获取算法

## 6　应用实例及其效果分析

### 6.1　服务替换性判定实例

本文以旅行服务中最常见的交通票务预订服务为例，分析 SST 与 QoST 支持的服务描述与替换性证明过程。现有 2 个预订服务 $\mathrm{BOOKING}_1$ 和 $\mathrm{BOOKING}_2$，通过上文所述方法，判定前者是后者的子类型，则前者可以安全地替换后者。限于篇幅，对部分服务描述不作详细展开。

$\mathrm{BOOKING}_1$ = <$\mathrm{QoS}_1$, $S_1$>　　$S_1 = \&\{OP_1{:}T_1;S_1\}$

$OP_1$={Book,{<Ticket, {<Ticket-departurePlace, Position>, <Ticket-arrivalPlace, Position>,<Ticket-arrivalTime, Time>, <Ticket-departureTime, Time>}>, <Ticket-Vehicle-kind,”FLIGHT”>}

$T_1$=?({<Ticket,{<Ticket-departurePlace,Position>,<Ticket-arrivalPlace,Position>, <Ticket-arrivalTime, Time>,<Ticket-departureTime,Time>}>})

$S_1$=+{{Cancel,$\mathrm{ticket}_1$}:end|{Pay,$\mathrm{ticket}_2$}:!($\mathrm{msg}_1$); &{{Pay,$\mathrm{ticket}_3$}:?($\mathrm{msg}_2$)|{Cancel,$\mathrm{ticket}_4$}:end}|{Change,$\mathrm{ticket}_5$}:ChangeTicket}

$\mathrm{BOOKING}_2$=<$\mathrm{QoS}_2$, $S_2$>　　$S_2$=&{$OP_2{:}T_2;\mathrm{S}_2$}

$OP_2$={Book,{<Ticket,{<Ticket-depaturePlace, Position>, <Ticket-arrivalPlace, Position>, <Ticket-arrivalTime,Time>,<Ticket-departureTime,Time>}>}}

$T_2$=?({<Ticket,{<Ticket-departurePlace,Position>,<Ticket-arrivalPlace,Position>, <Ticket-arrivalTime, Time>,<Ticket-departureTime,Time>,<Ticket-Vehicle-kind,String>}>})

$S_2$=+{{Cancel,$\mathrm{ticket}_6$}:end|{Pay,$\mathrm{ticket}_7$}:!($\mathrm{msg}_3$); &{{Pay,$\mathrm{ticket}_8$}:?($\mathrm{msg}_4$)|{Cancel,$\mathrm{ticket}_9$}:end}}

依照前文所述方法，可将目标命题拆分为 $QoS_1 <: QoS_2$ 与 $S_1 <: S_2$ 这 2 个子判定命题。其中，QoS 部分为数值比较，此处不作展开。$S_1 <: S_2$ 命题的证明思路如下。

1)$S_1$、$S_2$ 均为 Branching 类型实例，根据 Branching-S 将命题拆分为 $OP_1 <: OP_2$、$T_1;S_1 <: T_2;S_2$ 这 2 个命题。

2)利用 Op-S 证明 $OP_1 <: OP_2$。

3)$T_1;S_1$、$T_2;S_2$ 均为 Reception 类型实例，根据 Reception-S 将命题拆分为 $T_1 <: T_2$、$S_1 <: S_2$。

以 1)和 3)为例：

$$\frac{Op_1 <: \ : Op_2}{\text{由Op-S拆分}}$$

$$\frac{Book = Book}{\text{得证}} \qquad \frac{\{Ticket,\{\cdots\}\} <: \{Ticket,\{\cdots\}\}}{\text{由Concept-S逐个证明}}$$

$$\frac{T_1;S_1 <: \ : T_2;S_2}{\text{由Reception-S拆分}}$$

$$\frac{T_1 <: T_2}{\text{由Message-S拆分与证明}} \qquad \frac{S_1 <: S_2}{\text{由Selection-S拆分与证明}}$$

…　　…

观察可知，$T_1$ 与 $T_2$ 的 Reception 实例中的消息实例有如下关系：二者总体相似，但 $T_1$ 中的消息无 Vehicle.Name 语义，可推断 $T_1 <: T_2$。

4)类似 3)，照命题中需判定的对象所属的类型，选取对应规则进行拆分与证明。若全部拆解完成，且都能得到证明，则整体替换性命题得证。

### 6.2　面向服务组合的应用效果分析

为考察本研究提出的替换方法面向实际应用的能力，本文选取 Apache 软件基金会的 ODE 作为 BPEL 引擎以支持服务组合，选取 Cloudstack 作为 IaaS 框架以搭建云环境，构建了以下试验环境：1 台 IBM x3500M4 服务器（24 核 2 GHz 处理器、16 GB 内存和 Ubuntu12.04.4LTS 操作系统），1 台 IBM x3650M4 服务器（24 核 2 GHz 处理器、32 GB 内存和 Ubuntu12.04.4LTS 操作系统）。基于 Cloudstack4.2.0 的云环境，建立 30 台虚拟服务器（1 GHz 处理器、1 GB 内存和 Ubuntu12.04.4LTS 操作系统）以模拟云服务场景，建立 1 台虚拟服务器（4 GHz 处理器、4 GB 内存和 Ubuntu12.04.4LTS 操作系统）以运行 ODE 引擎，通过 6 台客户机（Intel Core i5 3.2 GHz，4 GB RAM 和 Windows7 64 bit Ultimate 操作系统）以模拟云服务请求。编写插件集成于 ODE，以实现规则库支持的推导演算。

鉴于旅游业务包含各类典型原子服务，可获得云平台的良好部署和应用支持，本文选择此业务进行理论验证。考虑到云服务替换中的现实复杂性（云服务类型和数量众多），本实验选取该业务中 10 个常用服务和大量服务数进行比较研究。所选常用服务涉及各项业务功能，皆用会话类型描述，具有验证代表性，它们包括行程规划、机票预订、火车票预订、酒店预订、租车、门票预订、在线第三方支付、银行结算、运营商短信、快递等，在此基础上对 10 个服务进行修改和扩展形成 500 个服务的集合，随机放入 30 台虚拟服务器中。通过模拟

服务组合中原子服务失效的场景，并选取传统的基于关键词的匹配与替换（本文简称 KSCM）、支持早期预测的服务替换算法（见文献[8]，本文简称为 EPSSM）、考虑组合上下文的服务替换（见文献[9]，本文简称 LDBSSM）和本文中基于类型论的服务替换方法（简称 STBM）进行对比实验。由于云服务的 QoS 高度动态，无法确定满足要求的服务数量，因此本文不选用查全率作为考察指标；在查准率的考察方面，考虑到在云服务组合中，判断单个服务查找是否准确存在困难。从应用视角出发，采用替换后组合服务的执行成功率作为指标具备可操作性，且能够反映替换方法的准确度；另将服务数量扩展至 3 500 个，以对比 4 种方案的查找耗时，考察 4 种方法的效率。情况如图 4 和图 5 所示。

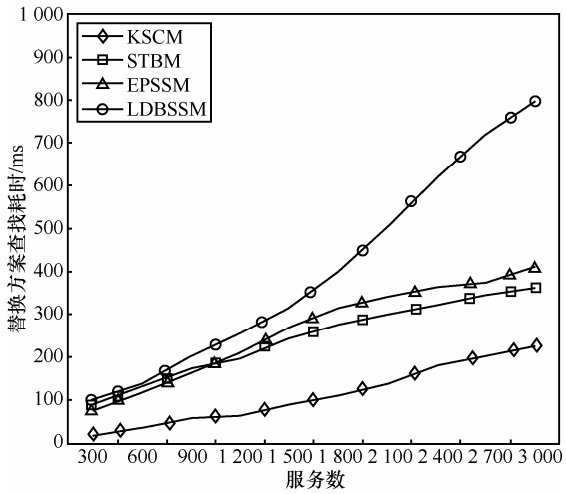


图 4　替换服务查找耗时对比

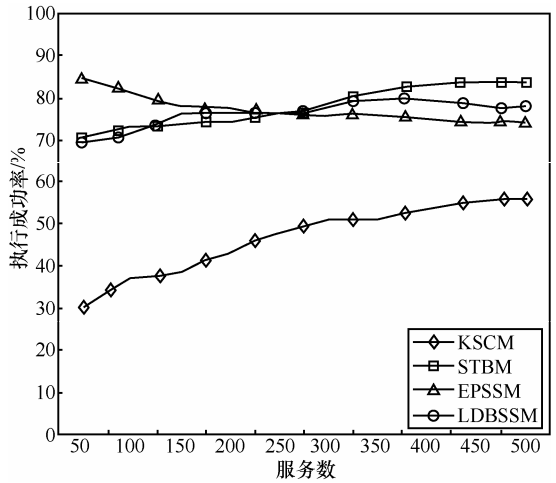


图 5　各替换方案下组合服务执行成功率对比

图 4 反映了不同服务数量情况下，4 种方法获取替换方案的耗时。可以看出，4 种方法均随候选服务集的增大耗时增多。EPSSM、STBM 与 LDBSSM 使用预测、上下文检测、形式化等方法，初始耗时相对较长。其中，STBM 由于初始规则较多，在开始阶段耗时较长，但随着服务增多，由于 STBM 的规则不会出现大幅增加，因此耗时增长速度放缓。LDBSSM 因上下文检测的对象快速增加，计算耗时快速增大。证明 STBM 方法可扩展性相对较强，在面对服务数量剧增时能兼顾效率。

图 5 反映了不同服务数量下，4 种方法在应用场景中的成功率。可以看出，STBM、EPSSM、LDBSSM 总体上明显优于 KSCM，尤其是在备选服务较少的情况下。其中，KSCM 在服务较少情况下替换成功率较低，随着服务增多成功率逐渐提高，但最终稳定在 60%左右。STBM 随服务数量增多成功率逐步提高，但提高过程存在小幅波动。EPSSM 在服务较少情况下成功率最高，但随着服务增多有小幅度下降。LDBSSM 则呈现先上升后小幅下降的趋势。分析可知，KSCM 在备选服务较少时可能无法通过关键字选取到合适的服务，因而替换的执行成功率较低。而 STBM、EPSSM、LDBSSM 由于具备上下文、语义信息采集能力，因此在备选服务较少时，有更大几率获取到关键字不匹配但功能相容的服务，因此成功率较高。EPSSM 因其未使用系统的形式化方法，服务表达存在一定模糊性，因而在服务数量较多的情况下成功率有所下降。LDBSSM 与 STBM 使用形式化方法，在服务数量增加时成功率上升，但 LDBSSM 使用的 Petri 网形式化方法存在验证上的复杂性，因而在服务数量较大的情况下可能给出不可靠的结果导致成功率有所下降。实验证明，STBM 在大规模服务场景下有较好的可适用性。

综上，本文提出的方法有效提高了小范围内的服务替换成功率，且在服务集规模增大过程中，依旧保持了较高的替换成功率，具备较好的扩展性。在效率方面，本文方法避免了计算复杂度的爆炸式增长，提高替换成功率的同时，付出了较小的时间代价。如实验对象选择时所述，旅游业务包含了多种类多数量的云服务，具有一般组合服务业务的典型特征，同时鉴于会话类型在服务描述方面代价较低，因此，以会话类型为基础的该云服务替换性判断方法具有较好的通用性，尤其在涉及高可信性需求的云服务应用中将表现出良好的现实价值。

## 7 结束语

云服务选择与替换对云应用至关重要，其中，对云服务动态行为功能与 QoS 进行准确描述，是判断服务行为可替换性的前提条件，也是云服务快速自动和可靠组合的重要基础。本文提出扩展后的语义会话类型 SST，并对云服务 QoS 进行分析与建模，构造 QoST 类型，设计 SST 与 QoST 的子类型规则，以此实现云服务特征化描述，最终完成基于子类型规则的替换性命题和兼容性命题判定。实例分析和实验测试表明，该云服务替换性判定方法具有良好可行性，不仅能完成对服务替换性兼容性的有效验证，也能较好提高组合服务的执行成功率。后续将本文方法运用到服务组合方案的制定中，并对原子服务间一对多、多对多的替换方法进行探究，是后续研究的可能方向。

## 参考文献：


[1] TAO F, LAILI Y, XU L, et al. FC-PACO-RM: a parallel method for service composition optimal-selection in cloud manufacturing system[J].IEEE Tran on Industrial Informatics, 2013, 9(4): 2023-2033.

[2] AMIN J, ELANKOVAN S, ZALINDA O. Cloud computing service composition: a systematic literature review[J]. Expert System with Applications, 2014,41(8): 3809-3824.

[3] 吴健, 吴朝晖, 李莹, 等. 基于本体论和词汇语义相似度的 Web 服务发现[J]. 计算机学报, 2005, 28(4): 595-602.
WU J, WU Z H, LI Y, et al. Web service discovery based on ontology and similarity of words[J]. Chinese Journal of Computers, 2005, 28(4): 595-602.

[4] JUAN C V, MANUEL L, ALBERTO B. Toward the use of petri nets for the formalization of OWL-S choreographies [J]. Knowledge and Information Systems, 2012, 32(3): 629-665.

[5] 常亮, 史忠植, 陈立民, 等.一类扩展的动态描述逻辑[J].软件学报, 2010,21(1):1-13.
CHANG L, SHI Z Z, CHEN L M, et al. Family of extended dynamic description logics[J]. Journal of Software, 2010,21(1):1-13.

[6] 廖军, 谭浩, 刘锦德. 基于 Pi-演算的 Web 服务组合的描述和验证[J]. 计算机学报, 2005,28(4):635-643.
LIAO J, TAN H, LIU J D. Describing and verifying Web service using Pi-calculus[J]. Chinese Journal of Computers, 2005, 28(4): 635-643.

[7] HE K, WANG J, LIANG P. Semantic interoperability aggregation in service requirements refinement[J]. Journal of Computer Science and Technology, 2010,25(6):1103-1117.

[8] 印莹, 张斌, 张锡哲. 面向组合服务动态自适应的事务级主动伺机服务替换算法[J].计算机学报, 2010, 33(11): 2147-2162.
YIN Y, ZHANG B, ZHANG X Z. An active and opportunistic service replacement algorithm orienting transactional composite service dynamic adaptation[J]. Chinese Journal of Computers, 2010, 33(11): 2147-2162.

[9] 王海艳,李思瑞. 基于组合上下文的服务替换方法[J]. 通信学报,2014,35(9):57-67.
WANG H Y, LI S R. Service substitution method based on composition context[J]. Journal on Communications, 2014,35(9):57-67.

[10] KUANG L, XIA Y, DENG S, et al. Analyzing behavioral substitution of Web services based on π-Calculus[C]//2010 IEEE International Conference on Web Services. Florida, USA, c2010:441-448.

[11] BOUROUZ S, ZEGHIB N. Verifying Web services substitutability using open colored nets reduction techniques[C]//The 5th International Conference on Modeling, Simulation and Applied Optimization. Hammamet, Tunisia, c2013: 1-5.

[12] 刘方方, 史玉良, 张亮, 等.基于进程代数的 Web 服务合成的替换分析[J]. 计算机学报, 2007,30(11):2033-2039.
LIU F F, SHI Y L, ZHANG L, et al. Substitution analysis of web service composition via process algebra[J]. Chinese Journal of Computers, 2007,30(11):2033-2039.

[13] REN H, LIU J. Service substitutability analysis based on behavior automata[J]. Innovations in Systems and Software Engineering, 2012, 8(4): 301-308.

[14] YIN Y, YIN J, LI Y, et al. Verifying consistency of web services behavior using type theory[C]// 2008 IEEE Aisa-Pacific Services Computing Conference. Yilan, China, c2008:1560-1566.

[15] YIN Y, DENG S. Analysing and determining substitutability of different granularity Web services[J]. International Journal of Computer Mathematics, 2013,90(11):2201-2220.

[16] 殷昱煜, 李莹, 邓水光, 等. Web 服务行为一致性与相容性判定[J]. 电子学报, 2009,37(3):433-439.
YIN Y Y, LI Y, DENG S G, et al. Determining on consistency and compatibility of Web services behavior[J]. Acta Electronica Sinica, 2009,37(3):433-439.

[17] ANTONIO V, VASCO T. V, ANTONIO R. Typing the behavior of software components using session types[J]. Fundamenta Information, 2006, 73(4): 583-598.

[18] PIERRE-MALO D, NOBUKO Y, ANDI B, et al. Parameterised multiparty session types[J].Logic Method in Computer Science, 2012, 8(4:6):1-46.

[19] DAPOIGNY R, BARLATIER P. Towards a conceptual structure based on type theory[C]//The Int'l Conference on Computational Science. Krakow, Poland, c2008:1-8.


## 作者简介：


**王先清**（1966-），男，湖南临澧人，广东科学技术职业学院副教授，主要研究方向为云服务、计算机应用技术、多媒体技术等。

**黄昌勤**（1972-），男，湖南常德人，华南师范大学教授、博士生导师，主要研究方向为可信云服务、语义智能、大数据技术及其教育应用等。

**罗旋**（1990-），男，湖南常德人，华南师范大学硕士生，主要研究方向为服务计算、云计算等。

**聂瑞华**（1963-），男，江西樟树人，华南师范大学教授，主要研究方向为计算机网络及应用、云计算与大数据等。

**汤庸**（1964-），男，湖南张家界人，华南师范大学教授、博士生导师，主要研究方向为社交网络与大数据应用、时态数据与知识工程、协同计算等。

**梅晓勇**（1974-），男，湖南常德人，博士，主要研究方向为服务计算、Petri 网技术与可信计算等。