面向动态环境的复合服务自适应方法

吕 晨[1),2)] 姜 伟[3)] 虎嵩林[1)]

[1)](中国科学院计算技术研究所前瞻实验室 北京 100190)

[2)](中国科学院大学 北京 100149)

[3)](中国石油长城钻探工程有限公司测井技术研究院 北京 100101)

摘 要 质量敏感的自动服务组合技术旨在从海量Web服务中生成满足用户功能性和非功能性需求的复合服务.这类技术通常假设服务环境静态不变.然而,对真实网络环境下服务的调研结果表明,实际情况与此不符:互联网中每天都会新增或消失大量的Web服务,出现因网络故障、延迟等原因而失效或QoS发生变化的Web服务.该文针对动态服务导致复合服务失效或QoS变差的问题,提出了一种基于事件驱动机制的复合服务自适应方法,可实时处理多种不同类型的动态服务,自动地对复合服务进行检查和更新.最后,文中对自适应方法进行了理论分析及实验验证:与WS-Challenge 2009和2010的冠军系统QSynth相比,文中方法在人工服务数据集和真实QoS数据集上均获得更优的性能.

关键词 动态服务;服务组合;自适应;服务计算;云计算

中图法分类号 TP391 **DOI号** 10.11897/SP.J.1016.2016.00305

Dynamic Environment-Oriented Self-Adaptation of Service Composition

LV Chen[1),2)] JIANG Wei[3)] HU Song-Lin[1)]

[1)](*Advanced Research Laboratory, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*University of Chinese Academy of Sciences, Beijing* 100149)

[3)](*Greatwall Drilling Company R&D Academy of Well Logging, CNPC, Beijing* 100101)

Abstract Quality of Service (QoS) aware automatic service composition (ASC) approaches aim at synthesizing the correct service compositions from thousands of Web services in order to satisfy both the functional requirements and the quality requirements of the users. Current approaches generally assume that the services are static. However, the investigation results of the services in real settings demonstrate that such a rigid assumption often fails: amounts of Web services in the internet are newly registered or deleted every day, a number of Web services become invalid or their QoSs change dynamically due to network failure, network latency, etc. Targeting the problem that the service compositions whose QoSs deteriorate or become invalid affected by dynamic services, this paper proposes a novel approach to adapt service compositions automatically in an event driven way. It can cope with different types of dynamic services, and meanwhile check the composition results and update them as necessary. Finally, we analyze our proposal and conduct comparison experiments with the state-of-the-art solution which won the performance championship of Web Service Challenge in 2009 and 2010. Experimental results show that our proposal achieves the superior performance on both synthetic Web service data and real QoS data.

Keywords dynamic services; service composition; self-adaptation; service computing; cloud computing

收稿日期:2015-02-08;**在线出版日期**:2015-07-23.**本课题得到国家自然科学基金**(61070027)**和软件工程国家重点实验室**(SKLSE2012-09-02)**资助.****吕 晨**,男,1983年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为服务计算、软件工程、API可用性. E-mail: lvchen@ict.ac.cn. **姜 伟**,男,1985年生,博士,主要研究方向为软件工程、API可用性、服务计算. **虎嵩林**(通信作者),男,1973年生,博士,研究员,主要研究方向为大数据处理、大规模分布式系统、服务计算. E-mail: husonglin@iie.ac.cn.

1 引 言

随着服务计算在人类生产、生活领域的不断普及与推广，Web 服务的种类和数量也在急剧增多[1]. 为满足用户快速、准确获取相关服务的需要，诸多网站，如 xmethods.net、seekda.com 提供了 Web 服务的查询与注册功能. 然而，单个 Web 服务的功能有限，多数情况下需要将分布在互联网上松散耦合的多个 Web 服务组合成功能更加丰富的复合服务(Service Composition，SC). 由于 Web 服务网络的规模和复杂度持续提升，Web 服务组合已成为一个典型的 NP(Non-Deterministic Polynomial)难问题[2]. 为此，相关研究者提出自动服务组合(Automatic Service Composition，ASC)技术来重用已存在的 Web 服务，通过对其进行自动组合来满足用户的功能性需求. 目前，ASC 已受到学术界和企业界的广泛重视，并涌现出大量成熟的应用. 例如，Amazon[3] 和 SAP[4] 公司均在真实场景下利用自动服务组合进行建模. 同时，为保证复合服务 SC 的服务 QoS(Quality of Service，QoS)，近年来，质量敏感的自动服务组合受到众多研究人员的关注. 但是大部分工作往往假设服务特性静态不变.

当前，已有研究者指出，基于服务的系统日益受到动态变化的服务环境带来的压力，需要持续增强系统的可用性、可靠性以及可扩展性[5]. 此外，文献[6]对服务功能性演化(如接口的变化)做了研究. 为了对真实环境下服务的功能性和非功能性演化进行更为广泛和全面的调研，在 2013 年 11 月至 2014 年 2 月期间，我们对网络上公开的 Web 服务进行了总计 11 个星期的监控. 这些 Web 服务分别抓取自 seekda.com，webservicelist.com 和 xmethods.net. 表 1 给出连续两周抓取数据的对比结果，其中约有 4%至 8%的 Web 服务发生了变化. 调研结果显示，互联网中每天都会新增或消失大量的 Web 服务，出现因网络故障、延迟等原因而失效或 QoS 发生变化的 Web 服务. 与我们的结论类似的是，一些更早期的研究工作[1,7] 揭示出在 2006 年 10 月至 2007 年 10 月期间，约有 21%的被监控的 Web 服务失效. 这些动态变化的 Web 服务会对已存在的复合服务产生负面影响：(1) 功能失效；(2) 非功能性需求难以满足. 为了消除上述负面影响，保证复合服务相关应用的可用性和可靠性，需要处理 Web 服务发生的各种变化，以便复合服务能够正确的反映当前的服务状态并实际可用.

表 1 Web 服务监控结果

连续两周对比	相同 WSDL	变化 WSDL/%	失效服务	新增服务	接口变化服务
1 vs. 2	12435	5.56	58	35	145
2 vs. 3	12823	4.52	33	2	42
3 vs. 4	12518	5.92	210	17	47
4 vs. 5	12635	5.75	320	453	88
5 vs. 6	12786	6.56	46	171	85
6 vs. 7	11355	4.24	115	88	75
7 vs. 8	11240	7.11	24	134	156
8 vs. 9	12306	4.35	62	13	289
9 vs. 10	12225	4.04	71	27	32
10 vs. 11	12370	5.33	16	267	54

本文以表 2 所示 Web 服务为例，解释并说明动态服务对复合服务的影响. 表 1 中给出 8 个 Web 服务及其相关信息. 其中，服务的 QoS 表示该服务的响应时间. 假设某用户输入的查询请求为：根据所在地地址、饮食偏好(如火锅)和行程要求(如目的地距离当前地址小于 20 km)等信息，获取相关酒店的行车路线. 为满足用户的这一查询请求(功能性需求)，程序需要自动选择合适的服务构建复合服务. 此外，为了快速响应用户请求并返回结果(非功能性需求)，在此场景下，我们还要求复合服务的总体响应时间尽量缩小. 根据表 1 所示服务，可构建复合服务 SC_1：$\{w_2 \rightarrow w_4 \rightarrow w_8 \rightarrow w_7\}$ 以满足上述要求. 然而，当 w_8 失效时，我们无法找出一个具备相同功能的服务对 w_8 进行替换. 此时，为了仍然满足用户的要求，需要对复合服务内部的组合逻辑进行重新调整. 例如，我们可构建出满足需求的复合服务 SC_2：$\{w_2 \rightarrow w_3 \rightarrow w_7\}$，然而其逻辑结构与 SC_1 不同. 由此可见，动态调整复合服务，不仅需要 1-1 替换，而且需要 n-m 替换.

表 2 Web 服务示例

Web 服务	输入参数	输出参数	功能	QoS/ms
W_1	a,b,c	d	返回酒店行车路线	800
W_2	a,b	e,f	返回城市名称和邮编	100
W_3	c,e	h	返回酒店地址	600
W_4	c,f	g	返回酒店邮编	100
W_5	k	h	返回酒店地址	600
W_6	i,j	d	返回酒店行车路线	500
W_7	h	d	返回酒店行车路线	300
W_8	g	h	返回酒店地址	100

参数含义：
a：所在地地址；b：行程要求；c：饮食偏好；d：酒店行车路线；
e：城市名称；f：城市邮编；g：酒店邮编；h：酒店地址；
i：酒店电话；j：酒店名称；k：酒店级别.

针对动态变化的服务环境，在自适应服务选择(Adaptive Service Selection)领域，诸多研究者使用整数规划[8-9]、启发性算法[10]、马尔可夫决策[11]、基于区域的重配置[12]、排队论[13]、遗传算法[14]、部分筛选(partial selection)[15] 和服务关联分析[16] 等策

略对运行时失效的服务进行替换. 通常，这些自适应服务选择的方法首先预定义组合模板来描述复合服务的组合逻辑. 然后，利用模板中抽象的服务类在运行时绑定具体的服务实例. 这类方法的不足之处在于：(1) 依赖预定义模板，无法自动搜索出复合服务；(2) 受限于模板中定义的组合逻辑，难以发现新的具有不同逻辑结构的复合服务作为替代方案；(3) 仅能感知和处理模板内部的服务类别，不能利用模板所包含的服务类别之外的服务进一步改善复合服务的服务 QoS；(4) 采用运行时被动处理的方式，难以对失效的复合服务进行主动发现和更新；(5) 每次只能处理一个动态服务，效率较为低下.

为了克服上述不足并解决动态环境下自动服务组合带来的挑战性问题，基于我们之前的工作 QSynth[17-18]，本文提出了一种复合服务自适应方法来应对动态变化的服务环境. 基于该方法，我们扩展了原有系统并将其称为 QSynth+. QSynth+基于服务依赖图(图 1)工作. 该图用反向索引表表示，能够准确表达 Web 服务之间的依赖关系. 基于该图，QSynth+首先检索出满足查询请求且服务 QoS 最优的复合服务；其次，从发布/订阅网络中接收动态服务事件；再次，根据动态变化的服务更新服务依赖图；最后，当原有最优复合服务失效或 QoS 变差时，基于更新后的服务依赖图，利用复合服务自适应算法生成新的复合服务.

需要指出，QSynth 可通过重新查询获取最新的复合服务. 然而，利用重新查询无法直接获取哪些 Web 服务发生了变化或受到了影响. 因此，在动态环境下，为确保复合服务的正确性，QSynth 需对所有的请求进行重新查询，这种频繁的查询操作会严重影响系统的效率. 与之相比，QSynth+能够识别出受影响的复合服务，同时只需更新少量受影响的服务状态，无需针对同一请求进行反复或重新查询，因此可极大地提升系统的性能.

本文的主要贡献包括：

(1) 引入了一种新颖的复合服务自适应机制进行动态环境下的自动服务组合，并通过事件驱动的方法主动发现和处理动态变化的服务.

(2) 提出了一种高效的复合服务自适应算法. 该算法无需预定义组合模板，能够感知模板内部和外部的所有服务，不仅支持服务的一对一置换，而且支持更为复杂的逻辑重构. 同时，该算法可采用批处理的方式一次性处理多个动态变化的服务.

(3) 实现了 QSynth+系统，并在人工和真实数据集上对系统的性能和准确性进行了有效评估.

本文第 2 节综述与本文相关的研究工作；第 3 节介绍预备知识并给出问题定义；第 4 节阐述系统的工作机理和相关算法细节；第 5 节对核心算法的复杂度、性质和开销进行分析；第 6 节讨论实验结果；第 7 节总结全文.

2　相关工作

服务选择和服务组合是与本文研究内容较为相关的工作. 本节从两方面分别阐述相关工作进展并进行讨论.

2.1　服务选择

服务选择是服务计算领域中的一个重要研究方向，已经吸引了众多研究者的关注，并取得了许多显著的创新成果[6-16]. 服务选择以某一预定义组合模板作为输入，并定义模板中的结点为抽象服务类别. 其目的是从众多功能属性相同但 QoS 不同的候选服务中选择满足用户要求的 Web 服务，将其作为服务实例与模板结点对应的服务类别进行绑定，使组合后的复合服务满足局部和全局 QoS(Global QoS, GQ)的约束要求. 针对动态变化的服务环境(如当服务失效或 QoS 变化时)，一些服务选择的研究工作提出了相应的服务替换策略，如混合整数规划[8]、区域重配置[12]、整数规划[9]、启发式算法[10]、马尔可夫决策[11]、排队论[13]、遗传算法[14]、部分筛选(partial selection)[15]和服务关联分析[16]等方法重新绑定符合要求的新服务. 但是，服务选择方法受限于预定义模板，其包含的服务类别和组合逻辑必须事先给出，因此无法根据动态变化的服务环境进行合理更改. 与之相比，本文提出的方法能够处理模板中已定义服务类别之外的新增服务，并可根据当前服务网络的状态对复合服务的组合逻辑进行调整. 此外，我们对动态服务进行实时监控，一旦动态服务发生，就主动对其进行处理，实现满足用户需求的复合服务的主动发现和更新.

2.2　自动服务组合

自动服务组合通过重用互联网上来自第 3 方提供者的 Web 服务，对多个原子服务进行组合来构建功能丰富、多样化以及个性化的复合服务. 每个复合服务定义了若干原子服务的组合逻辑和调用次序，并满足用户的功能性和非功能性需求. 面对日益增长的服务数量，相关研究者提出了众多自动服务组

合方法[17-22]以解决人工服务组合所带来的费时、低效和易错等问题. 上述自动服务组合方法均假定在服务静态不变的环境下,如何求取满足功能或服务QoS最优或近似最优的复合服务. 例如,QSynth[17]旨在解决QoS最优的海量语义服务快速组合问题,通过输入查询请求,构建相应的复合服务. 然而,每当服务动态变化时,这类方法必须对所有已构建的复合服务进行重新求取,效率十分低下(见第6节实验部分),因此并不可取. 文献[23-24]提出利用人工智能规划的方法处理动态服务并对复合服务进行相应更新. 然而,由于未考虑服务的QoS,这些方法通常不能保证更新后的复合服务依然满足非功能性需求. 文献[25]提出了一种面向动态服务环境的服务组合方法,基于最短路径的图搜索算法搜索出链状的复合服务. 但是,这种方法无法搜索出实际中存在的非链状复合服务,如可建模为有向无环图(Directed Acyclic Graph,DAG)形式的复合服务(见第3.2节). 文献[26-27]分别给出了面向动态环境的自动服务组合方法,然而,这些方法只关注服务功能性因素的变化(如服务接口的变化),而未考虑服务QoS这类非功能性因素变化的影响,难以推荐出满足QoS最优约束的复合服务. 与上述两个工作相反,文献[28-29]专注于根据原子服务QoS的变化改善复合服务的服务质量,而未对服务功能性因素变化的情形给出解决方案.

综合以上分析可知,面向动态服务环境且质量敏感的自动服务组合的相关研究较少,目前尚缺乏成熟完整的解决方案,需要有针对性的进行专门的研究. 下节将介绍必要的预备知识,并据此定义本文要解决的问题.

3 预备知识与问题定义

3.1 相关术语介绍

本文使用的相关术语及其定义见表3.

表3 相关术语

术语	定义
服务QoS	QoS是指w_i的非功能性属性,例如响应时间和吞吐量等.
Web服务(w_i)	w_i是一个三元组$\{I,O,Q\}$. I是w_i的输入参数集合;O是w_i的输出参数集合;$Q=\{Q_j\|_{j=1}^{n}\}$,其中Q_j是指w_i的第j维服务QoS.
查询请求R	R包含输入参数信息$R.I$以及输出参数信息$R.O$. $R.I$定义了用户提供的信息,如饭店名称. $R.O$声明了用户需要的信息,如饭店星级. 该请求可通过"饭店信息查询"服务满足. 然而,在多数情形下,查询请求需要多个服务的组合才能满足.
参数匹配	两个Web服务的参数,P_a和P_b匹配当且仅当两者类型相同或P_a所属本体概念是P_b所属的本体概念的子类. 例如,给定3个概念,"食品"、"水果"和"橘子",若某参数匹配"橘子",则一定匹配"食品"和"水果",反之未必成立.
Web服务匹配	两个Web服务,w_a和w_b匹配当且仅当$w_a.O\cap w_b.I\neq\varnothing$.

3.2 服务依赖图

为准确地表达Web服务集合中包含的依赖关系以及QoS等信息,我们使用服务依赖图$G=(W,E)$对表2所示Web服务进行建模,如图1所示. 本文将基于此图进行质量敏感的自适应服务组合. 在G中,节点集W表示Web服务集合,$\forall w_k\in W$,$w_k=(k,I,O,Q)$. 其中,k是节点w_k的标识,I和O分别表示w_k对应的Web服务的输入参数集合和输出参数集合,Q为w_k的QoS. 有向边集E表示Web服务匹配集合,该集合满足:$\forall e_k\in E$,$e_k=(w_u,w_v,tag_{e_k})$. 其中,w_u和w_v分别是边的头结点(head

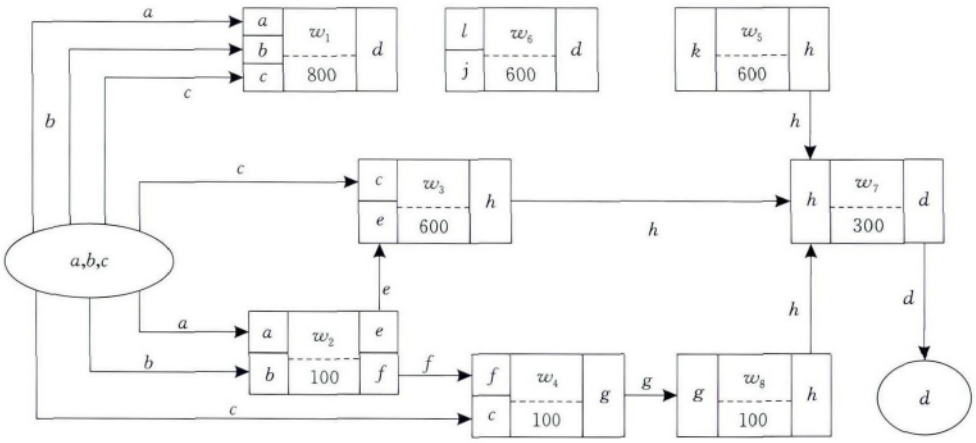


图1 服务依赖图

node)和尾节点(tail node)，w_u是w_v的直接前继，w_v是w_u的直接后继，w_u对应的方法匹配w_v对应的方法. e_k的标记tag_{e_k}满足：

(1) $tag_{e_k} \in w_u.O$；

(2) $tag_{e_k} \in w_v.I$.

需要特别指出，当查询请求R发起时，图中将动态生成临时的起始节点$Start$和终节点End，如图1中两个椭圆形节点所示，它们满足：

(1) $(Start.I=\varnothing) \wedge (Start.O=R.I)$；

(2) $(End.I=R.O) \wedge (End.O=\varnothing)$.

我们利用反向索引表存储和构建服务依赖图. 本文将在4.1节将对此进行详述.

与经典图论的节点可达性不同，G的节点可达性如下.

定义1.　节点的可达性. 给定服务依赖图$G=(W,E)$，在一次从起始节点$Start$开始的搜索过程中，$\forall w_i \in W$，w_i可达当且仅当$\forall input \in w_i.I$，$\exists p_i=\langle Start,\cdots,w_{i-1},w_i\rangle$，使得路径$p_i$中的边$e=(w_{i-1},w_i,tag_e)$满足$tag_e=input$.

在定义1中，一个节点w_i可达，不仅要求从$Start$到w_i存在路径，而且要求存在从$Start$到w_i的路径集合，使得该路径集合必须包含w_i的全部输入参数作为Tag的入边. 这不同于经典图论的节点可达性. 在经典图论中，判断一个节点w_i是否可达，只需判断从$Start$到w_i是否存在路径即可.

基于节点的可达性定义，本文提出的搜索算法在访问节点时遵循如下原则：若一个节点w_i可达，则可访问w_i的后继节点，否则，不能访问. 因此，当w_i输入参数还未被全部获取时，定义1中的可达性能够避免本文算法因调用该服务而错误地获取其输出参数. 例如，在图1中，当从$Start$开始搜索时，首先找到路径$p=\langle Start,w_4\rangle$. 然而，此时$w_4$并不可达，原因在于$w_4$的另外一个输入参数$f$作为Tag的入边未被$p$包含.

3.3　问题定义

基于服务依赖图，文献[17]提出质量敏感的自动服务组合问题.

定义2.　最优质量敏感的自动服务组合问题. 给定查询请求R和服务依赖图$G=(W,E)$，加权函数$\omega: M \to \mathrm{R}$表示从节点集合到实数集合的映射. 从G中找出一个子图SG表示满足R的复合服务对应的子图. SG定义了其蕴含的服务$(w_1,w_2,\cdots,w_n)$的调用次序，这些服务满足下述条件：

(1) $w_i.I \subseteq \bigcup_{j=1}^{i-1} w_j.O \cup R.I$；

(2) $R.O \subseteq \bigcup_{j=1}^{n} w_j.O$；

子图SG的权重值是指SG包含节点的权重值的聚合值：$\omega(SG)=SG.GQoS$. 若令SG_{All}表示所有满足R的候选复合服务集合，则最优复合服务的GQ(Global QoS)定义为$\delta=\mathrm{Min}\{\omega(SG_i) \mid SG_i \in SG_{\mathrm{All}}\}$. SG满足下述条件：

(3) $\omega(SG)=\delta$.

需要指出，定义1中的条件(1)、(2)定义了复合服务需满足的功能性需求，即满足查询请求，而条件(3)则定义了复合服务需满足的非功能性需求，即其GQ为最优.

为解决上述问题，QSynth基于t_1时刻的服务依赖图搜索出复合服务以满足查询请求. 当服务动态变化时，我们需对QSynth系统进行扩展，根据图中服务的最新状态持续更新已有的复合服务，保证其可用性. 为此，本文提出了质量敏感的复合服务自适应问题.

定义3.　最优质量敏感的复合服务自适应问题. 给定查询请求R，服务依赖图$G=(W,E)$和满足定义2中条件的复合服务SG. 当G被动态更新为$G'=(W',E')$时，自动更新SG为SG'，使SG'蕴含的服务依然满足定义1中的条件(1)、(2)，同时保证SG'的GQ最优性.

3.4　全局QoS计算规则

本小节将进行说明服务依赖图中的GQ计算规则. 该计算规则与QoS类型和图的组合模式相关.

3.4.1　QoS类型

文献[30-31]将QoS分为两类：(1)否定型，即QoS值越大，服务QoS越差，比如响应时间和价格；(2)肯定型，即QoS值越大，服务QoS越好，比如吞吐量和声誉. 同时，为了对多个QoS进行统一度量，按照度量方式的不同又可把QoS分为以下4种类型：(1)累加型，如响应时间. 对于两个顺序调用的服务，其全局响应时间可由各服务的响应时间累加获得；(2)最小值型，如吞吐量，两个顺序调用的服务的全局吞吐量，由具有最小吞吐量的服务决定；(3)乘积型，如声誉、可靠性；(4)最大值型. 文献[32-33]给出了更多可供参考的QoS类型及分类策略，本文不再赘述.

3.4.2　组合模式

本文的复合服务自适应算法将返回相应子图表示满足需求的复合服务. 多数情况下，这些子图

呈现为有向无环图 DAG 的形式. DAG 中主要包含 3 种组合模式:顺序(Sequence)、合并(Joint)与分叉(Split),如图 2 所示. 对于少量含有环的复合服务,本文将使用 Unfolding 方法[21]对环进行剔除操作.

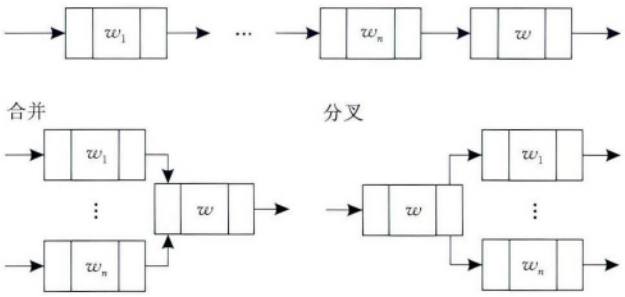


图 2 DAG 的组合模式

3.4.3 GQ 计算规则

服务的 GQ 计算规则由图的组合模式(见图 3)和 QoS 类型共同决定. 以响应时间为例①,其作为服务的第 j 维 QoS 时的全局计算规则如表 4 所示②.

表 4 GQ 计算规则

模式	计算规则
顺序	$w.GQ_j = w.Q_j + \sum_{i=1}^{n} w_i.Q_j$
合并	$w.GQ_j = w.Q_j + \max\{w_i.GQ_j\}|_{i=1}^{n}$
分叉	$w_k.GQ_j = w.GQ_j + w_k.Q_j (1 \le k \le n)$

首先,在顺序模式中,因为服务依次被顺序调用,所以采用累加型函数计算服务 w 的第 j 维 QoS 的全局聚合值,即最后一个被调用服务 w 的全局响应时间由其所有前驱的响应时间和 w 自身的响应时间累加获得.

其次,在合并模式中,当服务 w_1-w_n 均已能够被调用时,才可调用服务 w. 因此,w 的第 j 维 GQ 由 w_1-w_n 中第 j 维 GQ 最差的节点决定. 同时又因为响应时间属于否定型,所以采用最大值函数和累加型函数计算服务响应时间的全局聚合值,即合并节点 w 的全局响应时间是所有前驱响应时间的最大值与其自身响应时间之和.

最后,在分叉模式中,当 w 能被调用时,则可调用其分支节点 w_1-w_n 中的任一节点. 因此可采用累加型函数计算 GQ 聚合值,即分支节点的全局响应时间由其分叉节点 m 的全局响应时间和其自身的响应时间累加得到.

例 1. 以图 1 为例,利用表 4 中的 GQ 计算规则可得($Start$ 节点的 QoS——响应时间设置为 0):

$$\begin{aligned} w_4.GQoS &= \sum(\max\{Start.GQoS, w_2.GQoS\}, w_4.QoS) \\ &= \sum(\max\{Start.QoS, (Start.QoS + w_4.QoS)\}, w_4.QoS) \\ &= \sum(\max\{0, (0+100)\}, 100) \\ &= 100+100=200. \end{aligned}$$

需要特别指出,若 w 的同一输入参数由两个以上的服务提供,则本文选择其最优提供者作为其前驱节点. 任一可达节点的输入参数的最优提供者定义如下.

定义 4. 节点输入参数的最优提供者. 即该节点所有可达前驱中,提供该输入参数且 GQ 最小的前驱结点. 其形式化定义如下:给定服务依赖图 $G=(W,E)$, $\forall w_i \in W$, $\forall input_i \in w_i.I$, $input_i$ 的最优提供者为 p_{opt}, $p_{opt} \in W$,当且仅当 p_{opt} 满足:

(1) $p_{opt}.status = enabled$;

(2) $p_{opt}.o = input_i$;

(3) $p_{opt}.GQoS = \mathrm{Min}\{w.GQoS \mid w \in W \wedge w.o = input_i \wedge w.status = enabled\}$.

例 2. 以图 1 为例,节点 w_7 的输入参数 h 可由 w_3($status=enabled$, $GQ=700$), w_5($status=disabled$, $GQ=+\infty$)和 w_8($status=enbbled$, $GQ=300$)提供,根据定义 4, w_7 的输入参数 h 的最优提供者为 w_8. 因此,

$$w_7.GQ = w_8.GQ + w_7.QoS = 300 + 300 = 600.$$

4 质量敏感的复合服务自适应

为处理动态变化的服务,我们将复合服务自适应算法集成到 QSynth 系统中,设计并实现质量敏感的自适应服务组合系统——QSynth+. 如图 3 所示,该系统主要包含 4 个主要的工作步骤:

(1) 最优复合服务构建. 解决质量敏感的自动服务组合问题(定义 1),生成 t_1 时刻满足用户查询请求且服务 QoS 最优的复合服务. 同时,保存服务的中间状态信息,例如,在当前查询中,服务是否可达以及服务的 GQ 等信息.

(2) 动态服务监控与处理. 监控发生变化的动态服务. 同时,对动态服务进行分析,判断动态服务是否对当前已构建的服务依赖图和 t_1 时刻的最优复

① 本文中的 QoS 并不特指响应时间,也可以是其他指标,如服务价格等,此处仅以服务响应时间为例描述问题.

② 对于多维 QoS,可考虑采用多维决策的方法,如权重和文献[34],将多维 QoS 的值转化为一个聚合值.

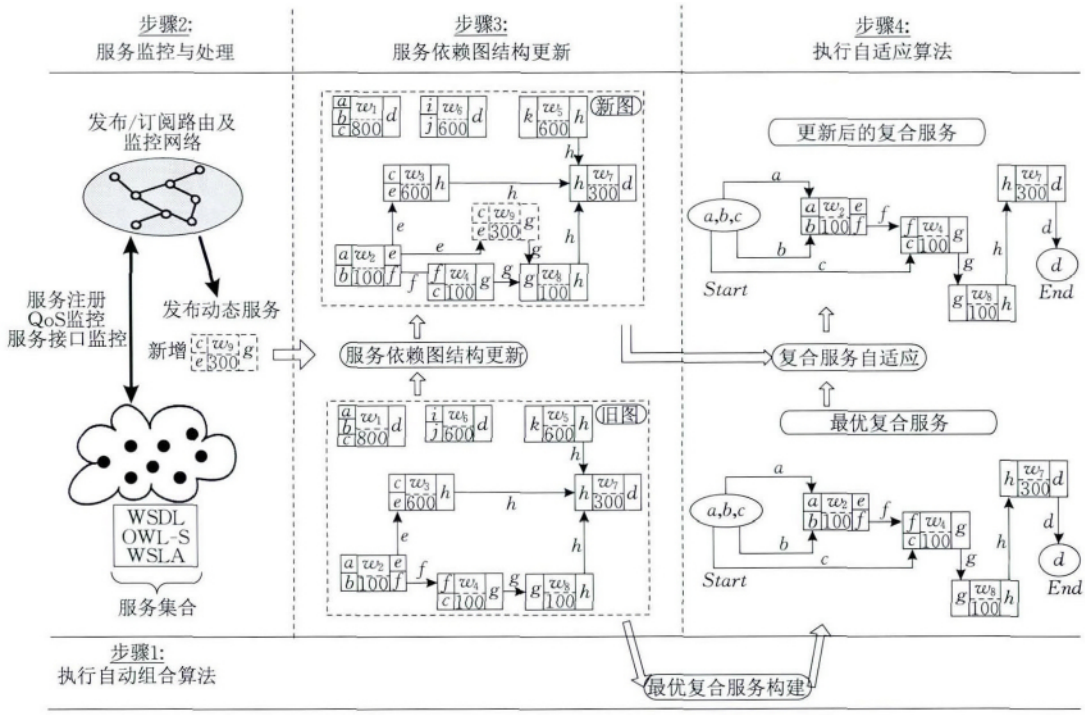


图 3　系统工作流程

合服务产生影响.

(3) 服务依赖图结构更新. 更新因动态服务的影响而发生结构变化的服务依赖图. 例如,根据步骤 2 的监控结果,须将新增服务 w_9 添加到图 3(步骤 3)所示的服务依赖图中.

(4) 复合服务自适应. 解决质量敏感的复合服务自适应问题,根据需要自动更新复合服务. 其基本处理流程是:首先,依据更新后的服务依赖图,找出图中受动态服务影响的服务并按照次序更新受影响服务的状态,例如 GQ 等;其次,判断复合服务是否受到动态服务环境的影响,即是否还满足用户查询请求和服务 QoS 最优;最后,利用反向搜索仅更新受影响的复合服务,而保留未受影响的复合服务. 以图 3(步骤 4)所示的最优复合服务为例,由于新增服务 w_9 亦提供其包含原子服务 w_8 的输入参数 g 且 w_9 在更新后的服务依赖图中可达,w_8 的 GQ 可能因 w_9 的加入而发生改变. 根据表 4 中的规则可知 $w_9.GQ > w_4.GQ$(400 > 200). 因此,w_9 为非参数 g 的最优提供者,最终不影响 w_8 的 GQ. 综上,最优复合服务无需更新.

下文将首先介绍 QSynth+依赖的数据结构,然后分别对上述 3 个步骤进行详细介绍.

4.1　数据结构

本文使用的数据结构包括反向索引表(Inverted Index Table, IIT),服务节点(Service Node),可达前置表(Reachable Precondition Table, RPT),如图 4 所示.

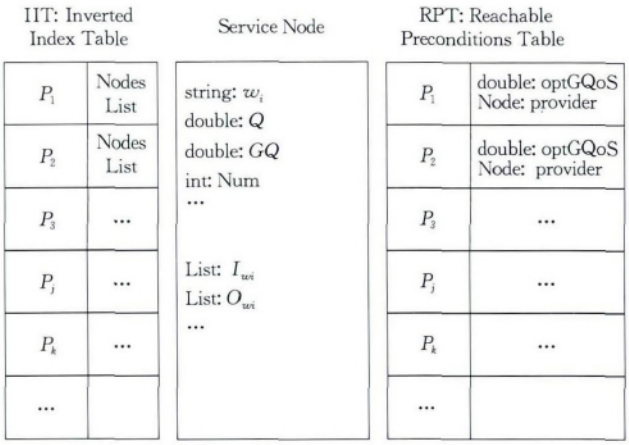


图 4　数据结构

反向索引表. 表示和存储服务依赖图. 表中的项为键值对(输入参数,节点列表). 其中,键为参数,其值为需要该参数作为输入的节点列表. 利用反向索引表,可避免传统的图存储结构(如邻接表和邻接矩阵)带来的计算复杂度较高的问题,从而能有效提升最优复合服务的搜索效率[17].

服务节点. 存储节点及其相关信息. 本文使用七元组表示节点,具体形式为 $\{w_i, I_{wi}, O_{wi}, Q, GQ, Count, Status\}$. 其中,$w_i$ 是节点对应 Web 服务的标识. I_{wi} 和 O_{wi} 分别表示节点的输入和输出参数集. Q 表示节点自身的 QoS. GQ 表示从起始节点到当前

节点的$GQ.Count$用来记录节点需要的输入参数的个数，初始值设定为节点输入参数集的大小. $Status$表示节点的状态，默认状态为不可达，记为 disabled. 在算法执行过程中，如果$Count$值减为 0，则令$Status$为可达态，记为 enabled. 需要特别指出，在执行过程中，算法将根据实际情况对相应节点的GQ，$Count$以及$Status$进行动态更新.

可触发参数表. 存储算法执行过程中可达节点的输入参数及其最优提供者的相关信息.

需要指出，由于我们采用反向索引表构建服务依赖图. 若记 c 为服务输入参数的平均个数，p 为单个输入参数服务提供者的平均个数，n 为服务依赖图中的节点总数，则构建服务依赖图的时间复杂度为 $O(n\times c\times p)$. 利用反向索引表中记录的键-值对信息(输入参数-服务列表)，可知与某服务接口参数匹配的前继服务和后继服务. 由于服务的语义不是本文研究的重点，所以在本文的服务模型中，按照通常自动服务组合领域的惯例，我们对服务参数的语义进行了简化处理，服务接口之间的匹配是按照参数类型的匹配进行判断. 例如，当一个服务 A 的输出参数是另一服务 B 的输入参数时(参见本文 3.1 节“参数匹配”和“服务匹配”)，A 和 B 就被视为匹配. 此外，因为服务依赖图由数据结构反向索引表唯一定义和表达，所以服务依赖图也唯一.

4.2 最优复合服务构建

为解决质量敏感的自动服务组合问题，我们利用基于前向搜索的 Sim-Dijkstra 算法生成最优复合服务的中间信息，然后利用后向搜索生成最优复合服务.

4.2.1 Sim-Dijkstra 算法

为求取最优复合服务，我们在之前的工作[18]中提出了 Sim-Dijkstra 算法. 该算法的基本思想是根据查询请求，执行一次从始发端点 $Start$ 到终止端点 End 的前向搜索，找出位于两端点之间所有状态可达的节点，并利用可达前置表存储和记录所有可达节点输入的最优提供者.

例 3. 当不考虑动态服务时，以图 3(步骤 1)为例说明 Sim-Dijkstra 算法的执行过程，见表 5.

4.2.2 后向搜索

Sim-Dijkstra 算法结束后，将获得存储在可达前置表中关于可达节点的相关信息. 利用这些信息，可求取满足查询请求的最优复合服务. 具体过程如下：从 End 到 $Start$ 发起一次后向搜索. 在搜索过程中，对于当前的节点，选择可达前置表中记录的该节点输入参数的最优提供者，并将其作为当前节点的前驱节点，直至反向回溯至 $Start$ 节点.

表 5 Sim-Dijkstra 算法执行步骤

步骤	$enabledNodes(GQ)$	RPT 项 参数(GQ, $provider$)
1	$Start(0)$	$\varnothing$
2	$w_2(100)$, $w_1(800)$	a (0, $Start$); b (0, $Start$); c (0, $Start$)
3	$w_4(200)$; $w_3(700)$; $w_1(800)$	e (100, w_2); f (100, w_2)
4	$w_8(300)$; $w_3(700)$; $w_1(800)$	g (200, w_4)
5	$w_7(600)$; $w_3(700)$; $w_1(800)$	h (300, w_8)
6	$End(600)$; $w_3(700)$; $w_1(800)$	d (600, w_7)

例 4. 利用表 5 中的 RPT 项，可构建图 3 所示的最优复合服务，具体过程见表 6.

表 6 最优复合服务构建过程

步骤	节点	参数(前驱)
1	End	$d(w_7)$
2	w_7	$h(w_8)$
3	w_8	$g(w_1)$
4	w_4	$c(Start)$; f (100, w_2)
5	w_2	a ($Start$); b($Start$);
6	复合服务：$Start\rightarrow w_2\rightarrow w_4\rightarrow w_8\rightarrow w_7\rightarrow End(GQ=600)$	

4.3 动态服务监控与处理

在 QSynth+系统中，我们采用 soapUI① 对 Web 服务及其 QoS 进行监控和获取，然后按照文献[35]中阐述的方法从发布/订阅的网络中接收关于动态服务的事件. 例如，当某个服务 w_i 的响应时间超出指定阈值(如大于 4 s)时，相应的事件将动态生成并路由至系统中.

对于接收到的动态服务，QSynth+按照不同的处理策略将其可分为 4 类，分别为：(1) 新增服务(C1)：服务依赖图中新加入的服务；(2) 失效服务(C2)：服务依赖图中已存在但当前不可用的服务；(3) 接口变化服务(C3)：服务依赖图中已存在但接口信息(如输入、输出参数)发生变化的服务；(4) QoS 发生变化的服务(C4).

根据分析可知，这些动态服务将会产生如下影响：

(1) 动态服务将引起服务依赖图结构的改变. 因此，本文将相应地对存储服务依赖图的反向索引表和服务节点进行更新操作. 例如，图 3 中的新增服务 w_9，其输入参数包括 c 和 e. 对于该服务，我们首先在服务节点中添加新表项，用于注册 w_9 的相关信息，然后，根据参数 c 和 e 查找反向索引表中相应的表项，将 w_9 添加到 c 和 e 对应的节点列表中.

① http://www.soapui.org/

(2)动态服务将引发某些 Web 服务的状态变化,如 GQ 或可达状态的变化.例如,新增服务 w_9 在更新后的服务依赖图中的状态为可达,因此 w_9 可能影响其可达后继节点的 GQ 或可达状态.

(3)动态服务将造成某些复合服务的失效或服务 QoS 下降.此时,必须对受影响的复合服务进行更新,以满足用户的功能性和非功能性需求.尽管利用 QSynth 重新查询一次可获取当前状态下最新的复合服务,然而每当出现新的动态服务时,QSynth 就需要对所有存在的复合服务进行重新查询,效率极为低下,故不能采用.本文 4.5 节将给出效率更高的方式判断缓存中(已存在)的复合服务是否受到影响,从而决定是否对其进行更新.

4.4　服务依赖图结构更新算法

由 4.3 节可知,4 类动态服务会对服务依赖图产生影响.按照动态服务的类型,本文分别使用如下策略进行服务依赖图的更新:首先,对于新增服务,在服务依赖图中添加相应的节点;其次,对于失效服务,从服务依赖图中删除相应的节点;再次,对于 QoS 发生变化的服务,更新服务依赖图中相应节点的 QoS 值;最后,对于接口变化的服务,分为两步处理:(1)在服务依赖图中删除原有服务;(2)将服务作为新增服务处理.因此,按照上述方式,我们已将 4 类动态服务转换为 3 类动态服务.

服务依赖图结构更新算法见算法 1,其具体过程为:首先,处理接口变化服务,即将原有服务归类为失效服务,同时将接口变化后的服务归类为新增服务(第 1～4 行).其次,处理新增服务.算法将在反向索引表和服务节点中添加其对应的表项(第 5～15 行).再次,处理失效服务.算法将从反向索引表和服务节点中移除相应表项(第 16～23 行).最后,处理 QoS 发生变化的服务.算法将更新服务节点中相应节点的 QoS(第 24～26 行).

算法 1.　服务依赖图结构更新算法.

输入:原始服务依赖图　IIT&Nodes
　　动态服务　$C1,C2,C3,C4$
输出:新服务依赖图　IIT&Nodes

```
    //处理接口变化服务集合 C3
1.  FOREACH Service u∈C3
2.    C1.add(original u);
3.    C2.add(changed u);
4.  END FOR
    //处理新增服务集合 C1
5.  FOREACH Service v∈C1 do
6.    Nodes.add(v);
7.    FOREACH par∈v.I do
8.      entry⇐IIT.findByKey(par);
9.      IF entry≠∅ THEN
10.       list⇐entry.nodeList;
11.       list.add(v);
12.     ELSE
13.       IIT.addNewEntry(par,v);
14.   ENDFOR
15. ENDFOR
    //处理失效服务集合 C2
16. FOREACH Service w∈C2 do
17.   Nodes.remove(w);
18.   FOREACH par∈w.I do
19.     entry⇐IIT.findByKey(par);
20.     list⇐entry.nodeList;
21.     list.remove(w);
22.   ENDFOR
23. ENDFOR
    //处理 QoS 变化服务集合 C4
24. FOREACH Service x∈C4 do
25.   x.updateQoS();
26. END FOR
```

4.5　复合服务自适应算法

当接收到动态服务事件后,QSynth+利用复合服务自适应算法根据实际情况生成新的最优复合服务.该算法的基本思想是:首先,判断服务依赖图中的动态服务是否影响其他服务的状态.如果动态服务的加入或删除不影响其他服务的状态,则无需更新其他服务的状态,否则,需要从动态服务开始,在服务依赖图中依次更新受影响服务的状态.其次,若原有复合服务包含受影响的服务,则需要对此复合服务进行更新,否则,可维持原有复合服务不变.

复合服务自适应算法见算法 2,其具体步骤如下:

(1)分析动态服务的影响,识别出可能引起其他服务状态变化的动态服务(第 1～12 行).具体而言,一方面,对于新增服务,当服务依赖图添加该服务后,重新获取其 GQ.新的 GQ 将存储到 $newGQ$ 变量中,$newGQ$ 初始值为 $+\infty$.若新增服务的 $newGQ$ 不等于 $+\infty$,则该服务加入服务依赖图后,将可能会影响其后继节点的状态.例如,该服务不可达的后继节点可能因新增服务节点变为可达.另一方面,对于失效服务与 QoS 变化的服务,若其原始的 GQ 不等于 $+\infty$,则该服务节点的删除或 QoS 更新会影响其可达后继节点的状态.例如,当失效服务节点被删除后,其原本处于可达状态的后继节点可能变为不可

达.当服务 QoS 变化后,其可达后继节点的 GQ 将可能会受到影响.

(2)更新受影响服务的状态(第 13～26 行).在该步中,算法首先利用优先队列 PQ 存储第一步中识别出的可能影响其他服务状态的动态服务.然后,循环弹出 PQ 中的服务并进行相应处理.定义 $pqQoS=\min(newGQ,GQ)$.受 Dijkstra 算法启发①,算法 2 按照 $pqQoS$ 从小到大顺序依次更新服务状态,即 PQ 优先弹出 $pqQoS$ 最小的节点.这种更新顺序可避免服务状态的反复更新.以图 3 为例,假设 w_2 和 w_4 是动态服务,其 QoS 发生了变化.若先更新 w_4,接着处理 w_8,w_7 和 End,则当更新 w_2 时,会引起 w_4,w_8,w_7 和 End 的再次更新.一种较好的更新次序应为:先更新 w_2,然后依次处理 w_4,w_8,w_7 和 End.利用优先队列 PQ,算法 2 可有效避免上述示例中服务状态的冗余更新.本文第 5 节将对此进行详细讨论.对于每个弹出的服务,在当前服务依赖图中重新获取其输入的最优提供者.若其输入的最优提供者发生变化,则更新相关 RPT 表项.此时,由于该服务的直接后继节点可能受到上述更新的影响,算法 2 将把这些后继节点放入 PQ 中等待下次处理.

(3)当优先队列中所有服务处理完毕时,根据需要生成新的复合服务(第 27～32 行):① 若 End 节点的 GQ 变为 $+\infty$,则表明查询请求已不能满足.此时,算法 2 无法生成新的、可供替换的复合服务;② 若 End 节点的 GQ 不等于 $+\infty$,则检查原有复合服务是否包含受影响的服务(即 GQ 发生变化的服务).如果包含,就需要反向生成新的复合服务,否则保留原有复合服务不变.

算法 2. 复合服务自适应算法.

输入:新服务依赖图 IIT&Nodes
动态服务 $C1,C2,C3,C4$
节点可达信息 RPT
最优复合服务 SC
输出:新最优复合服务 SC'

```
//识别可能影响其他服务的动态服务
1.  FOREACH Service u∈C1 do
2.    u.GQ⇐+∞;
3.    u.newGQ⇐u.getGQ();
4.    IF u.newGQ≠+∞ THEN
5.      PQ.add(u);
6.    END IF
7.  END FOR
8.  FOREACH Service v∈C2 ∪ C4 do
9.    IF v.GQ≠+∞ THEN
10.     PQ.add(v);
11.   END IF
12. END FOR
    //更新受影响服务状态
13. WHILE PQ≠∅ do
14.   w⇐PQ.popOpt();
15.   FOREACH par∈w.O
16.     newProvider⇐getOptP(par);
17.     IF par.provider≠newProvider THEN
18.       entry⇐RPT.findByKey(par);
19.       IF entry≠∅ THEN
20.         entry.optGQ⇐newProvider.GQ;
21.         entry.provider⇐newProvider;
22.       END IF
23.       PQ.add(w.directSuccessors);
24.     END IF
25.   ENDFOR
26. ENDWHILE
    //复合服务更新
27. IF End.GQ=+∞ THEN
28.   Return no results;
29. END IF
30. IF Service x∈SC & x.GQ≠X.newGQ THEN
31.   Return SC′ by backward search;
32. END IF
```

例 5. 如图 3(步骤 4)所示,当新增服务 w_9 加入到服务依赖图中时,表 7 给出了复合服务自适应算法的具体过程.从表 7 中可以看出,尽管 w_9 也提供了最优复合服务,包含原子服务 w_8 的输出参数 g,但是该最优复合服务未受到影响,因此无需更新.

表 7 复合服务自适应算法执行步骤

步骤	操作	优先队列 PQ
1	由于 $w_9\in C1$,$w_9.newGQ(=400)$ $w_9=+\infty$,$w_9.pqQoS=400$,$PQ.add(w_9)$	$\varnothing$
2	$w=w_9$,输出参数 g 对应 RPT 表项为 $g(200,w_4)$.由于 $w_9.newGQ>w_4.GQ(=300)$,无需更新 RPT.因此,需要参数 g 作为输入的 w_8 的 GQ 未发生变化.	w_9
3	由于原有最优复合服务不包含受影响的服务,可维持原有结果不变,无需进行更新	$\varnothing$

5 自适应算法分析

本节首先分析了复合服务自适应算法(算法 2)

① 令 v 为始发节点,为解决单源最短路径问题,Dijkstra 算法发起一次从 v 开始的前向搜索.在搜索过程中,使用优先队列存储当前所有可达节点.然后,按照与 v 距离从小到大的顺序,依次弹出节点,处理其后继节点.算法反复执行这一过程,直至队列为空.

的时间复杂度，然后给出算法重要性质的定义及证明，最后讨论了算法需要的额外开销.

5.1　时间复杂度

记 c 为服务输入参数的平均个数，m_1、m_2、m_3 和 m_4 分别为新增服务数、失效服务数、接口变化服务数和 QoS 变化服务数，k 为优先队列 PQ 中服务节点的最大数目. 若记 n 为服务依赖图中的节点总数，则推断出 nc 为服务依赖图中边的数量.

算法 2 第 1 行到第 12 行的时间复杂度是 $O(m_1\times c+m_2\times c+2m_3\times c+m_4\times c)=O(m\times c)$，其中 m 为动态服务总数. 算法 2 第 13 行到第 26 行对 PQ 中服务的处理操作的时间复杂度为 $O(k\log k+kkc)$. 其中 $O(k\log k)$ 为维护优先队列[①]的开销，$O(kkc)$ 为处理每个节点的开销. 算法 2 中第 27 行到 32 行调用后向搜索的时间复杂度为 $O(nc+n)$. 综上，算法 2 的时间复杂度为 $O(m\times c+k\log k+kkc+nc+n)$.

5.2　算法性质

定理 1.　可终止性. 算法经过有限步后停止.

证明.　在算法 2 中，当优先队列为空时，算法将终止. 因为每个服务节点最多放入优先队列中一次，所以该队列最大值为 n. 每次循环算法均会弹出队列中的元素，故最多经过 n 步后，算法将会终止.　　证毕.

定理 2.　完整性. 在动态服务环境下，只要存在满足查询请求的复合服务，复合服务自适应算法就能够将其找出.

证明.　若查询请求可被满足，则 End 节点可达，即 End 节点的 GQ 不等于 $+\infty$. 算法 2 依据不同类型的动态服务对其受影响服务的状态进行相应更新，仅按照 $pqQoS$ 调整可达服务节点的更新顺序，而没有剪枝或剔除任一可达服务节点. 因此，只要 End 节点可达，算法 2 就可找出能够抵达 End 节点的复合服务.　　证毕.

定理 3.　最优性. 复合服务自适应算法得到的复合服务 SC 是 GQ 最优的.

证明.　假设存在一复合服务 SC'，$SC'.GQ<SC.GQ$，即 SC' 优于 SC，等价于至少存在参数 par，其提供者为服务 w' 和 w，其中，$w'\in SC'$，$w\in SC$，且 $w'.GQ<w.GQ$. 由题设可知算法 2 返回 SC 而非 SC'. 同时，根据算法 2 第 10 行，算法总是取 par 的最优提供者，因此算法 2 返回 SC 表明 $w.GQ<w'.GQ$，与假设矛盾.　　证毕.

定理 4.　单次更新性：复合服务自适应算法对受影响的服务只进行一次状态更新.

证明.　设存在两个动态服务 w_1 和 w_2，且 $w_1.pqQoS<w_2.pqQoS$. 根据算法 2 第 8 行，算法先更新 w_1 的 GQ，再更新 w_2 的 GQ. 假设更新 w_2 后，w_1 的 GQ 受到影响，需要再次更新，则表明 w_2 位于某一从 $Start$ 节点到 w_1 的路径中. 因为在这种情况下，w_2 的 GQ 更新才会传播影响到 w_1 的 GQ. 根据算法 2 可知：首先，对于所有新增服务，根据算法 2 第 2 行，其对应的 GQ 原值——$GQ=+\infty$；其次，对于所有失效服务，由于这些服务将从服务依赖图中删除，其对应的 GQ 新值——$newGQ=+\infty$.

综上，可得

(1) 若 $w_1\in C1$，$w_2\in C1$，则
$w_1.GQoS=w_2.GQoS=+\infty$，
$w_1.newGQoS>w_2.newGQoS$；

(2) 若 $w_1\in C1$，$w_2\in C2$，则：
$w_1.GQoS=w_2.newGQoS=+\infty$，
$w_1.newGQoS>w_2.GQoS$；

(3) 若 $w_1\in C1$，$w_2\in C3$，则：
$w_1.GQoS=+\infty$，
$w_1.newGQoS>w_2.newGQoS$；

(4) 若 $w_1\in C2$，$w_2\in C1$，则：
$w_1.newGQoS=w_2.GQoS=+\infty$，
$w_1.GQoS>w_2.newGQoS$；

(5) 若 $w_1\in C2$，$w_2\in C2$，则：
$w_1.newGQoS=w_2.newGQoS=+\infty$，
$w_1.newGQoS>w_2.newGQoS$；

(6) 若 $w_1\in C2$，$w_2\in C3$，则：
$w_1.newGQoS=+\infty$，
$w_1.GQoS>w_2.GQoS$；

(7) 若 $w_1\in C3$，$w_2\in C1$，则：
$w_1.newGQoS>w_2.newGQoS$，
$w_2.GQoS=+\infty$；

(8) 若 $w_1\in C3$，$w_2\in C2$，则：
$w_1.GQoS>w_2.GQoS$，
$w_2.newGQoS=+\infty$；

(9) 若 $w_1\in C3$，$w_2\in C3$，则：
$w_1.GQoS>w_2.GQoS$，
$w_1.GQoS>w_2.newGQoS=+\infty$.

上述 9 种情形可囊括所有动态服务类型. 以第 1 种情形为例，当 w_1，w_2 均被加入到服务依赖图中后，由于 w_2 位于某一从 $Start$ 节点到 w_1 的路径中，根据 3.4 节中 QoS 的计算规则易得 $w_1.newGQ>w_2.newGQ$. 因为

① 本文采用 Relaxed heap 实现优先队列.

$w_1.pqGQ=\min\{+\infty, w_1.newGQ\}=w_1.newGQ$;
$w_2.pqGQ=\min\{+\infty, w_2.newGQ\}=w_2.newGQ$.

所以有 $w_1.pqGQ>w_2.pqGQ$,与假设矛盾.以此类推,易得所有 9 种情形均与假设矛盾. 证毕.

5.3 额外开销

算法 2 的额外开销包括:(1)不仅要存储 Sim-Dijkstra 算法返回的最优提供者信息,还需要存储其他非最优提供者信息.例如,当最优提供者失效时,自适应算法可利用其他提供者生成当前状态下的最优结果.总之,算法 2 需要存储额外的服务节点的状态信息、RPT 表项等;(2)需在数据结构中添加 *newGQ* 和 *pqQoS* 两个新变量.尽管上述开销使得算法 2 需占用更多的存储空间,但是无需从头开始搜索和求取满足用户需求的复合服务.在多数情况下,算法 2 只需更新少量受影响服务的状态,与重新查询相比,效率更高,从而能够满足实时响应的需要.本文第 6 节对此进行了实验验证.

6 实 验

6.1 实验设置

为应对动态服务,基于复合服务自适应算法,本文实现了 QSynth+系统来进行自适应服务组合.实验设置介绍如下.

对比系统:(1)QSynth,每当动态服务发生时,重新执行查询;(2)QSynth+,扩展后的 QSynth 系统,使用复合服务自适应算法动态更新复合服务;(3)QSynth+*,使用无优先队列版本的复合服务自适应算法的系统,以不考虑服务更新次序的方式执行复合服务自适应算法.

测试数据集:(1)人工数据集:利用公开的 Web Service Challege 测试集生成器①生成测试数据集.利用该测试集生成器的可调参数,即服务数目,服务输入参数类型数目和复合服务层数(一般而言,层数越多,则复合服务越复杂),我们生成了包含不同参数类型数量、不同层数、不同服务数和不同动态服务数的测试集,见表 8;(2)真实 QoS 数据集,采用文献[36]提供的数据源,我们搜集并整理了 2000 个真实服务在不同时刻的 QoS,并提取其中的响应时间赋值给测试集生成器生成的服务集合②.

实验中,我们将在每个测试集中按照随机方式生成动态服务.在真实 QoS 数据集中,我们根据真实服务的 QoS 变化设置动态服务的 QoS.每个测试集包含如下 3 个文件:WSDL(描述服务及接口信息),OWL-S(描述概念(参数类型)信息),WSLA(描述服务的 QoS 信息).上述 3 个文件是描述 Web 服务的国际标准,进行实验对比的 3 个系统将分别读取这些文件以及描述查询请求的查询文件,处理动态服务并求取新的满足查询请求且服务 QoS 最优的复合服务.

表 8 测试数据集

测试数据集	参数类型总数	复合服务层数(每层平均服务数)	服务总数	动态服务总数
第 1 组:不同参数类型总数	1000,5000,10000,15000,20000,25000	8(4.4)	6000	100
第 2 组:不同复合服务层数	15000	4(5),8(4.4),12(4.3),16(3.8),20(2.1),24(2.8)	6000	100
第 3 组:不同服务总数	15000	8(4.4)	1000,2000,4000,6000,8000,10000	100
第 4 组:不同动态服务总数	15000	8(4.4)	6000	50,100,200,300,400,500

评估指标:为了对相关系统进行评估,本文采用如下 3 个的评价指标、效率、敏感性和准确性来对系统的优劣进行统一衡量.具体而言,效率用来评估服务依赖图的构建时间和系统的响应时间,即系统从接收到动态服务到生成新的复合服务所消耗的时间.敏感性:衡量系统对 QoS 变化程度是否敏感,即 QoS 变化是否对系统效率造成剧烈影响.准确性用来测试系统返回的复合服务是否满足查询请求和服务 QoS 最优的要求.

实验环境为 2.4 GHz CPU,4 GB RAM,Windows 7 操作系统.

实验方法在表 8 给出的 4 组测试数据集中,由于数据集中的动态服务是按照随机方式生成的,实验中存在随机因素.为了消除最终实验结果的随机性,我们在不同数据集上进行了多次独立地实验.具体而言,针对任意一组数据集,我们分别对不同系统进行了 10 次测试.每次测试中均按照随机方式生成

① Web Service Challenge 2009 [Online]. Available: http://wschallenge.georgetown.edu/wsc09/

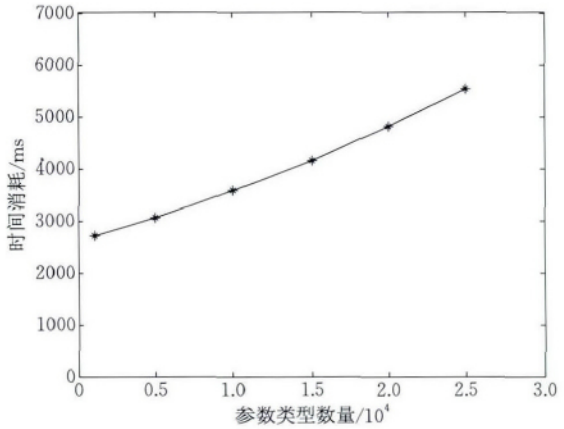
② 真实 Web 服务缺乏语义标注信息,同时多个真实的 Web 服务常常使用不同词汇标注同一概念(参数类型).因此,在缺乏语义标注信息的情况下,我们很难进行精确地服务匹配和组合.综上,本文未采用真实 Web 服务直接进行服务组合,而是利用真实服务的 QoS 进行实验对比.

动态服务. 最终，实验将选取 10 次测试结果的平均值作为最终实验结果.

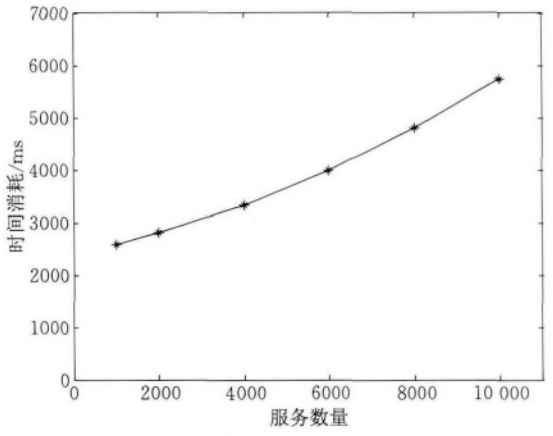
6.2　效率评估

6.2.1　服务依赖图构建时间评估

本文算法基于服务依赖图搜索最优的复合服务. 服务依赖图的创建是算法高效运行的前提. 图 5(a)和图 5(b)分别给出了在不同参数类型总数和不同服务总数下的服务依赖图的创建时间. 从图中可以看出，随着参数类型总数和服务总数的增加，服务依赖图的创建时间也在不断增大. 这是因为图的复杂性与服务节点和参数类型的数目相关. 从图 5 中可以看出，当参数类型总数或服务总数达到万级规模时，图的创建较为高效，其时间消耗均在 6000 ms 之内.



(a) 不同参数类型总数(服务总数固定为6000)



(b) 不同服务总数(参数类型总数固定为15 000)

图 5　服务依赖图的创建时间

6.2.2　系统查询响应时间的评估

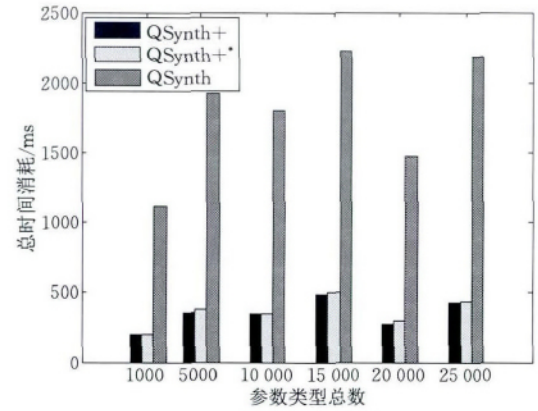
本文实验采用人工生成服务的 QoS 和真实服务的 QoS，分别在表 8 所示的 4 组测试数据集上对 QSynth、QSynth+和 QSynth+* 进行了效率评估. 由图 6 可知，总体上，QSynth+性能最优，其查询响应时间平均仅为 QSynth 查询响应时间的 10%～40%. 这是因为每当动态服务产生时，QSynth 均需重新执行查询，从头开始搜索整个服务空间，而 QSynth+只需要更新部分受动态服务影响的服务状态，无需搜索整个服务空间. 因此节省了时间消耗. 同时，相比于 QSynth+*，QSynth 的查询响应时间平均提升了 4%～30%. 这是因为 QSynth+* 使用的复合服务自适应算法未考虑服务的更新次序问题，因此会消耗额外的时间对受影响的服务进行多次、反复地更新.

此外，我们综合图 6 所示的 4 组测试数据集上的实验结果对系统效率依赖的 4 个重要因素，即参数类型总数、复合服务层数、服务总数以及动态服务总数的影响分别进行了分析. 结果如下：

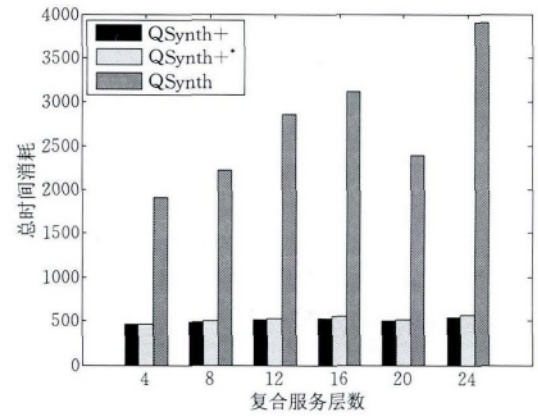
如图 6(a)所示，当参数类型总数由 1000 增长到 25 000，复合服务层数、服务总数和动态服务总数分别固定为 8、6000 和 100 时，3 个系统的查询时间消耗不随参数类型总数增长而呈现线性增长的趋势，这表明系统相关的算法复杂度并不直接依赖于参数类型的数量. 根据本文第 5 节以及文献[18]中关于算法复杂度的分析，文本算法的时间复杂度主要取决于动态服务的数量，而 QSynth 的算法复杂度则与图中服务节点的总数以及边的数目相关.

如图 6(b)所示，当复合服务层数由 4 增长到 24，参数类型总数、服务总数和动态服务总数分别固定为 15 000、6000 和 100 时，3 个系统的查询时间消耗在总体上呈现增长趋势. 因为一般而言，层数越多，则复合服务越复杂，系统耗费的查询时间也应该越多. 然而，当复合服务层数为 20 时，各系统查询的时间消耗反而比复合服务层数为 12 和 16 时更低. 对此我们进行了分析，并在表 8 第 3 列给出了各复合服务每层的平均服务数(四舍五入). 由此可得，测试集中 20 层的复合服务尽管层数较多，但相比 12 层和 16 层的复合服务，其包含的服务节点总数较少. 根据本文第 5 节以及文献[18]中关于算法复杂度的分析，3 个对比系统的时间复杂度分别取决于动态服务的数量和图中服务节点的总数以及边的数目，而非直接取决于复合服务层数. 因此，在实验中，尽管复合服务层数为 20，但是其包含的节点总数相对较少，因此各系统查询的时间消耗相对较低.

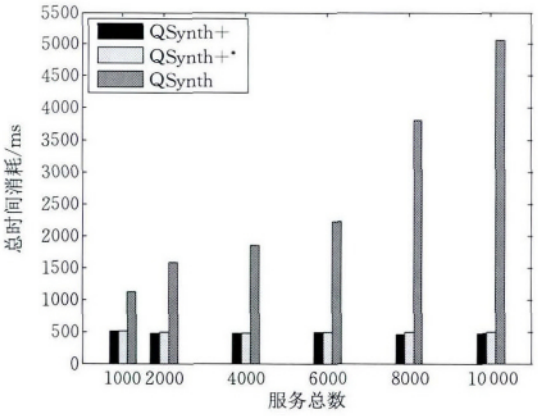
如图 6(c)所示，当服务总数由 1000 增长到 10 000，参数类型总数、复合服务层数和动态服务总数分别固定为 15 000、8、100 时，QSynth 的查询响应时间明显依赖于 Web 服务集的规模——服务数量越大则查询响应时间越长. 这是因为每当动态服



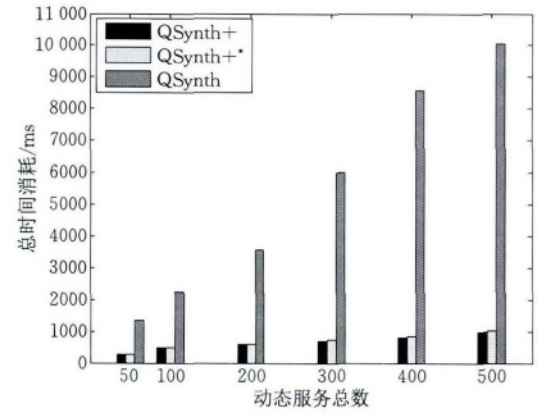
(a) 不同参数类型数目



(b) 不同层数



(c) 不同服务数目



(d) 不同动态服务数目

图 6 效率对比结果(人工服务 QoS)

务产生时,QSynth 均需重新执行查询,从头开始搜索整个服务空间,因此查询的响应时间主要取决于服务数量的大小. 而对于 QSynth+和 QSynth+*,其查询响应时间不明显依赖于 Web 服务集的规模. 这是因为复合服务自适应算法只需要更新部分受动态服务影响的服务状态,无需搜索整个服务空间. 因此,当服务数量呈现增长趋势时,其响应时间未随之明显增大.

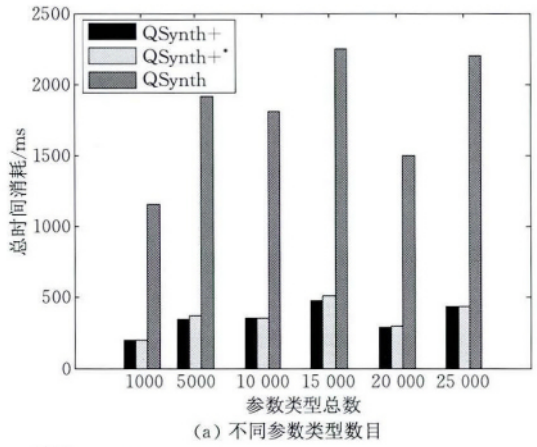
如图 6(d)所示,当动态服务总数由 50 增长到 500,参数类型总数、复合服务层数和服务总数分别固定为 15 000、8、6000 时,3 个系统的查询的总体时间消耗均呈现上升的趋势. 其中,QSynth 的增长幅度较大,而 QSynth+和 QSynth+* 的增长幅度较小. 这是因为对于 QSynth 而言,每新增一个动态服务,意味着增加一次搜索整个服务空间的时间,因此时间消耗的增长幅度较高. 与之相比,当动态服务发生时,由于 QSynth+和 QSynth+* 可避免重新搜索,消耗时间较少,因此增长幅度也较低.

此外,从图 7 可以看出,各系统在真实 QoS 服务数据集上的表现与上述实验结果类似.

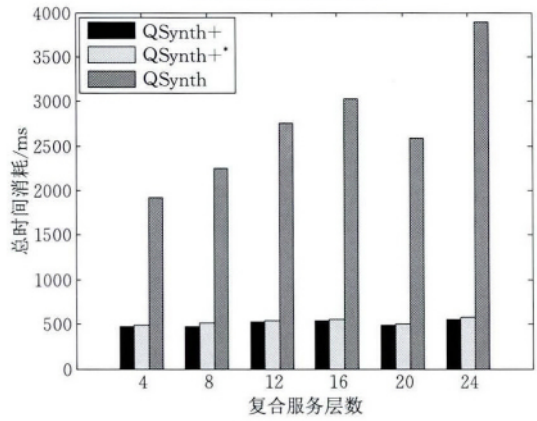
图 6 和图 7 给出的 QSynth+和 QSynth+* 的查询时间消耗是服务依赖图更新与自适应算法执行时间的总和. 实验中,我们还跟踪和记录了不同数据集上服务依赖图更新的性能,其时间消耗稳定在 1 ms～5 ms 之间. 相比总体时间消耗,所占比例很低. 图更新的低耗时与其采用的存储结构——反向索引表密切相关:本文的图更新算法的执行时间主要取决于动态服务包含的参数个数,而访问反向索引表中参数对应数据项的时间复杂度仅为 $O(1)$. 因此,图的更新较为高效.

为了分析动态服务对其他服务的影响,当层数为 8、参数类型数为 15 000、服务数为 6000、动态服务数为 100 时,我们在图 8 中给出各系统处理单个动态服务的查询响应时间. 从图 7 可以看出,QSynth 处理每个动态服务的时间较多而总体方差较小,而 QSynth+和 QSynth+* 处理每个动态服务的时间较少但总体波动较大. 这一结果表明:(1)每个动态服务的影响区域不同,因此使用复合服务自适应算法系统的处理时间变化较大;(2)每个动态服务的影响区域有限,因此使用复合服务自适应算法只需更新相对更少的受影响服务的状态,从而获得比重新查询更高的效率.

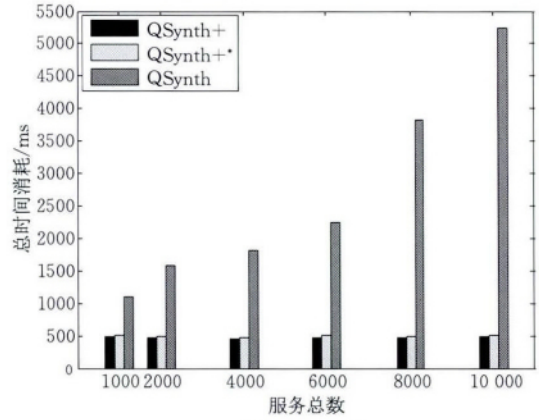
我们还考察了 QSynth+在批处理方面的性能. 当层数为 8、参数类型数为 15 000、服务数为 6000



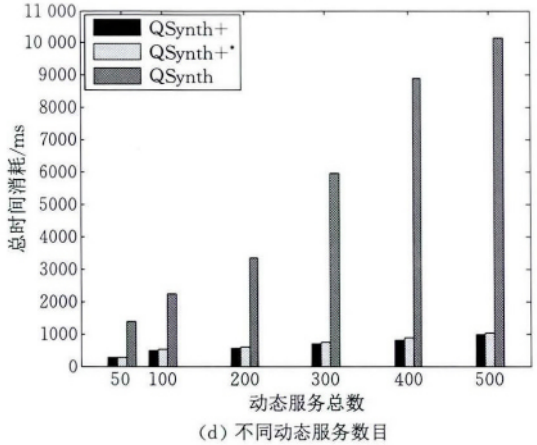
(a) 不同参数类型数目



(b) 不同层数



(c) 不同服务数目



(d) 不同动态服务数目

图 7　效率对比结果(真实服务 QoS)

时，如图 9 所示，QSynth＋一次性处理 50～600 个动态服务时间消耗为 14 ms～28 ms. 这一结果远远低于每次仅处理一个动态服务的处理方式所耗费的时间(2880 ms①). 然而，当一次处理的动态服务数量超过 500 时，使用 QSynth 进行重新查询反而效率更高. 这一结果表明，当一次性处理的动态服务数量超过某一阈值(如 500)时，可选择使用重新查询的方式. 然而，在实际中，同一时刻产生大量动态服务的情况并不多见，因此多数情况下选择 QSynth＋会获得更高的效率.

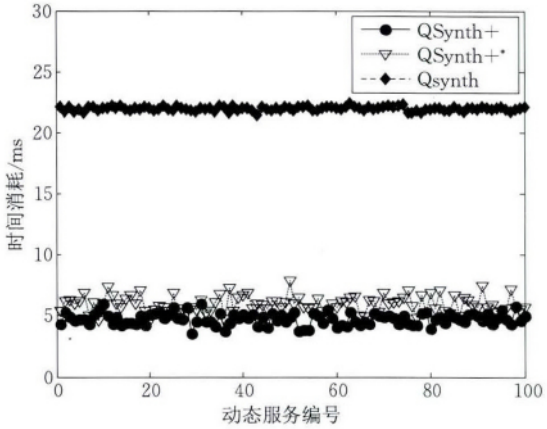


图 8　单个动态服务处理时间

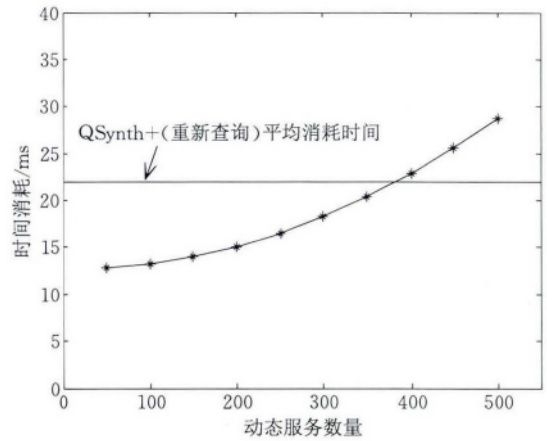


图 9　QSynth＋批处理时间消耗

6.3　敏感性评估

本节我们重点针对 QoS 变化是否对系统的查询响应时间造成影响进行了分析和比较. 从图 10 中可以看出，当层数为 8、参数类型数为 15000、服务数为 6000、动态服务数为 100 时，动态服务的 QoS 分别增加 10 ms、50 ms、100 ms、200 ms 和 300 ms 时，QSynth＋响应时间的波动范围为 402 ms～513 ms，

① 由图 7 可知，Qsynth＋处理单个动态服务的平均时间为 4.8 ms. 因此，按照每次处理一个的方式，处理 600 个动态服务需耗费 4.8 ms×600＝2880 ms 的时间.

QSynth+* 响应时间的波动范围为 484 ms～591 ms，QSynth 响应时间的波动范围为 2159 ms～2297 ms. 这一结果表明，当 QoS 发生变化时，被评估的 3 个系统的效率均未产生较大波动. 因此，它们对 QoS 的变化并不敏感.

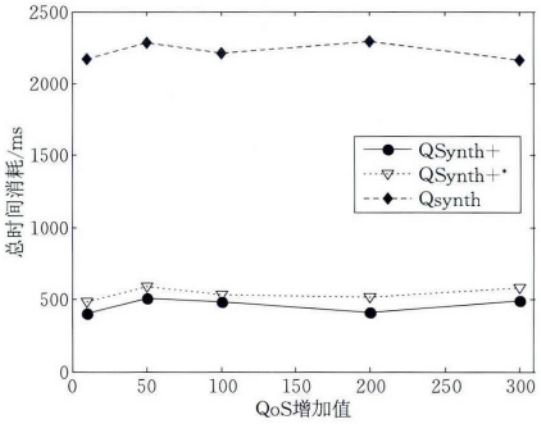


图 10 QoS 变化的影响

6.4 准确性评估

本节评估各系统的准确性，即其返回的复合服务是否满足查询请求且保证全局 QoS 最优. 一方面，我们认真检查各系统返回的复合服务，核对复合服务中包含的原子服务是否均可触发，即复合服务是否可执行以满足查询请求. 另一方面，QSynth 系统已被理论和实验证明其返回的复合服务是全局 QoS 最优的[18]，并且在动态更新后的图中搜索不影响其最优性. 因此，通过比较 QSynth 返回结果的全局 QoS 与其他系统返回结果的全局 QoS 是否相等，即可判断当前系统返回的复合服务是否是全局 QoS 最优的结果. 我们对 6.3 节实验中相关系统返回的所有结果都进行了上述两方面的检验. 结果显示，QSynth+ 和 QSynth+* 返回的复合服务与 QSynth 的返回结果一致且均可正确执行，同时全局 QoS 相等. 这表明基于复合服务自适应算法的 QSynth+ 能够应对动态变化的服务环境，根据需要自动地更新复合服务，保证更新后的结果仍然满足查询请求且全局 QoS 最优，系统准确率为 100%，进一步验证了第 5 节中给出的算法性质.

7 结 论

针对动态变化的服务环境，本文对质量敏感的自动服务组合系统 QSynth 进行了扩展，设计并实现了一个新的复合服务自适应系统——QSynth+. 在该系统中，我们引入了事件驱动机制并提出了一种新颖的复合服务自适应算法，使其能够主动地处理各种不同类型的动态服务，同时避免搜索整个服务集合空间，仅更新部分受影响服务的状态，有效地提升了系统的效率. 此外，QSynth+ 不仅支持服务的 1-1 替换，而且能够根据需要变更复合服务的组合逻辑，以支持服务的 n-m 替换. 与已有方法相比，QSynth+ 采用的数据结构更加复杂，并需存储中间的搜索结果（参见 5.3 节），这是它的代价. 然而，本文提出的算法具有最优性和完整性，并且通过重用已有的中间搜索结果，仅对局部受影响服务进行更新，无需重新搜索整个服务空间. 实验结果表明，在服务总数达到万级规模时，其依然能够保证算法的执行时间在 500 ms～600 ms 之间，并且在不同的测试集上均能准确反映用户的功能性和非功能性需求，具有较高的系统响应速率和较好的可扩展性.

本文只针对统一语义框架下服务的自动组合问题展开相关研究. 然而，现实中的服务匹配往往异常复杂. 对于同一个意义，不同服务可能存在不同的表述方式. 例如，在描述服务信息的 WSDL 文件中，关于“城市”这个概念，不同的服务提供者可能使用形式完全不同的词汇（例如“city”和“town”）进行描述. 为了避免现实问题因过于复杂而无解，我们采用科学问题常用的研究策略，在某些方面做了一定的简化处理，比如本文简化了语义和服务匹配，然后抽象出问题模型，并进一步提出解决方法. 自动服务组合是学术界较为关注且算法研究较多的领域. 在这个领域中，相关的研究工作通常都对现实问题进行了一定程度地简化，例如将其抽象为“最短路径”或“最优 DAG”等图论问题. 此外，这些抽象出的问题不仅可用于服务组合，也可应用于其他领域，例如供应链领域. 在供应链领域，一个相似的问题是如何根据不同的目标（例如最小费用、最短生产周期），选择合适的原材料和加工厂完成产品的制造. 因此，本文抽象出的问题模型不仅可以用于自动服务组合，也可以应用于其他领域.

在未来的工作中，我们将主要从以下两个方面进一步加强研究：(1) 根据实际需要，在算法中引入更多维度的 QoS，深入研究多维 QoS 场景下复合服务的自适应机制；(2) 目前 QSynth+ 仅针对最优 (Top 1) 复合服务进行自适应更新. 为了在动态服务环境下更好的满足用户的偏好（如中国大陆用户通常选择百度搜索服务而非谷歌搜索服务）以及解决负载均衡（避免过分使用热门复合服务或原子服务）等问题，我们将进一步对支持 Top-k 查询的

QSynth 系统[18]进行扩展，研究并解决前 k 个最优复合服务的自适应问题.

参 考 文 献

[1] Al-Masri E, Mahmoud Q H. Investigating Web services on the world wide web//Proceedings of the 17th International World Wide Web Conference (WWW'08). New York, USA, 2008: 795-804

[2] Wen Tao, Sheng Guo-Jun, Guo Quan, Li Ying-Qiu. QoS optimal automatic composition of semantic Web services. Chinese Journal of Computers, 2013, 36(5): 1031-1046(in Chinese)
(温涛，盛国军，郭权，李迎秋. 基于改进粒子群算法的 Web 服务组合. 计算机学报，2013，36(5)：1031-1046)

[3] Marconi A, Pistore M, Poccianti P. Automated Web service composition at work: The Amazon/MPS case study//Proceedings of the 14th International Conference on Web Services (ICWS'07). Salt Lake City, USA, 2007: 767-774

[4] Rao J, Dimitrov D, Hofmann P, Sadeh N. A mixed initiative approach to semantic Web service discovery and composition: Sap's guided procedures framework//Proceedings of the 13th International Conference on Web Services(ICWS'06). Chicago, USA, 2006: 401-410

[5] Broinizi M, Mutti D, Ferreira J. Application configuration repository for adaptive service-based systems: Overcoming challenges in an evolutionary online advertising environment //Proceedings of the 21th International Conference on Web Services(ICWS'14). Alaska, USA, 2014: 670-677

[6] Li J, Xiong Y, Liu X. How does Web service API evolution affect clients//Proceedings of the 21th International Conference on Web Services (ICWS'13). Santa Clara Marriott, USA, 2013: 300-307

[7] Lu J, Yu Y, Roy D, Saha D. Web service composition: A reality check//Proceedings of the Web Information Systems Engineering(WISE'07). New York, USA, 2007: 523-532

[8] Ardagna D, Pernici B. Adaptive service composition in flexible processes. IEEE Transactions on Software Engineering, 2007, 33(1): 369-384

[9] Chafle G, Dasgupta K, Kumar A, et al. Biplav Srivastava. Adaptation in Web service composition and execution//Proceedings of the ICWS'06. Los Alamitos, USA, 2006: 549-557

[10] Mabrouk N B, Beauche S, Kuznetsova E, et al. QoS-aware service composition in dynamic service oriented environments//Proceedings of the Middleware'09. New York, USA, 2009: 7:1-7:20

[11] Verma K, Doshi P, Gomadam K, et al. Optimal adaptation in Web processes with coordination constraints//Proceedings of the 13th International Conference on Web Services (ICWS'06). Alamitos, USA, 2006: 257-264

[12] Zhai Y, Zhang J, Lin K. SOA middleware support for service process reconfiguration with end-to-end QoS constraints//Proceedings of the 16th International Conference on Web Services (ICWS'09). Los Alamitos, USA, 2009: 815-822

[13] Yu L, Wang Z, Meng L, Qiu X. Towards multi-user and network-aware Web services composition//Proceedings of the 20th International Conference on Web Services(ICWS'13). Santa Clara Marriott, USA, 2013: 607-608

[14] Klein A, Ishikawa F, Honiden S. SanGA: A self-adaptive network-aware approach to service composition. IEEE Transactions on Services Computing, 2014(3): 452-464

[15] Chen Y, Huang J, Lin C. Partial selection: An efficient approach for QoS-aware Web service composition//Proceedings of the 21th International Conference on Web Services (ICWS'14). Alaska, USA, 2014: 1-8

[16] Zhang Y, Zhang B, Zhang C. Correlation-supported composite service reselection//Proceedings of the 21th International Conference on Web Services (ICWS'14). Alaska, USA, 2014: 510-517

[17] Jiang W, Zhang C, Huang Z, et al. Qsynth: A tool for QoS-aware automatic service composition//Proceedings of the ICWS'10. Alaska, USA, 2010: 42-49

[18] Jiang W, Hu S, Liu Z. Top K query for QoS-aware automatic service composition. IEEE Transactions on Services Computing, 2014, 4(7): 681-695

[19] Deng Shui-Guang, Huang Long-Tao, Wu Bin, et al. QoS optimal automatic composition of semantic Web services. Chinese Journal of Computers, 2013, 36(5): 1015-1030(in Chinese)
(邓水光，黄龙涛，吴斌等. 一种 QoS 最优的语义 Web 服务自动组合方法. 计算机学报，2013，36(5)：1015-1030)

[20] Bartalos P, Bielikova M. QoS aware semantic Web service composition approach considering pre/postconditions//Proceedings of the 17th International Conference on Web Services (ICWS'10). Los Alamitos, USA, 2010: 345-352

[21] Ma Y, Chen L, Hui J, Wu J. CBBCM: Clustering based automatic service composition//Proceedings of the 2011 International Conference on Services Computing. Washington DC, USA, 2011: 354-361

[22] Wagner F, Ishikawa F, Honiden S. QoS-aware automatic service composition by applying functional clustering//Proceedings of the 18th International Conference on Web Services (ICWS'11). Washington DC, USA, 2011: 89-96

[23] Yan Y, Poizat P, Zhao L. Repair vs. recomposition for broken service compositions//Proceedings of the 8th International Conference on Service Oriented Computing (ICSOC'10). San Francisco, USA, 2010: 152-166

[24] Yan Y, Poizat P, Zhao L. Self-adaptive service composition through graphplan repair//Proceedings of the 17th International Conference on Web Services (ICWS'10). Los Alamitos, USA, 2010: 624-627

[25] Kalasapur S, Kumar M, Shirazi B A. Dynamic service composition in pervasive computing. IEEE Transactions on Parallel Distributed System, 2007, 18(1): 907-918

[26] Wang H, Wu Q, Chen X, et al. Adaptive and dynamic service composition via multi-agent reinforcement learning// Proceedings of the 21th International Conference on Web Services (ICWS'14). Alaska, USA, 2014: 447-454

[27] Geyik S C, Szymanski B K, Zerfos P. Robust dynamic service composition in sensor networks. IEEE Transactions on Services Computing, 2013, 6(4): 560-572

[28] Feng Y, Ngan L D, Kanagasabai R. Dynamic service composition with service-dependent QoS attributes//Proceedings of the 21th International Conference on Web Services(ICWS'13). Santa Clara Marriott, USA, 2013: 10-17

[29] Groba C, Clarke S. Opportunistic service composition in dynamic ad hoc environments. IEEE Transactions on Services Computing, 2014, 7(4): 642-653

[30] Yu T, Zhang Y, Lin K. Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Transactions on Web, 2007, 1(1): 1-26

[31] Zeng L, Benatallah B, Ngu A, et al. QoS-aware middleware for Web services composition. IEEE Transactions on Software Engineering, 2004, 30(5): 311-327

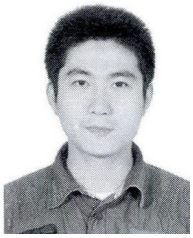
[32] Jaeger M C, Rojec-Goldmann G, Muhl G. QoS aggregation in Web service compositions//Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service(EEE'05). Hong Kong, China, 2005: 181-185

[33] Jaeger M C, Rojec-Goldmann Muhl G. QoS aggregation for Web service composition using workflow patterns//Proceedings of the 8th Enterprise Distributed Object Computing Conference (EDOC'04). Monterey, USA, 2004: 149-159

[34] Fulkerson D R, Ford L R. Multiple Attribute Decision Making: An Introduction. New York: Sage Publications, 1995

[35] Yan W, Hu S, Muthusamy V, et al. Efficient event-based resource discovery//Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS'09). Nashville, USA, 2009: 1-12

[36] Al-Masri E, Mahmoud Q H. QoS-based discovery and ranking of Web services//Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN'07). Honolulu, USA, 2007: 529-534



LV Chen, born in 1983, **Ph. D. candidate. His research interests include service computing, software engineering and API usability.**

JIANG Wei, born in 1985, **Ph. D. His research interests include software engineering, API usability and service computing.**

HU Song-Lin, born in 1973, **Ph. D., professor. His research interests include Big Data processing, large scaled distributed system, service computing.**

Background

Nowadays, increasing attention has been drawn towards the problem of automatic service composition (ASC) in dynamic services environment. However, conventional ASC approaches generally assume that the services are static. However, such an assumption is often baseless. New services come and go, service APIs change gradually, and QoS values fluctuate. In this paper, we propose an event driven sel-adaptation algorithm for QoS-aware automatic service composition problem to cope with different types of dynamic services. Moreover, we integrated this algorithm in our service composition system, QSynth. In this solution, only the region of affected services is updated instead of the whole Web service space. N-m substitution is supported instead of 1-1 substitution. Moreover, it avoids redundant updates of affected services. Therefore, our proposal can achieves much higher performance than related studies.

This work presented in this paper was supported by the National Natural Science Foundation of China under Grant No. 61070027, the State Key Laboratory of Software Engineering (SKLSE2012-09-02). These projects aim to perform some basic research on software engineering. Our group has working in the area of services computing, library reuse and software engineering for several years and published many papers. The papers most related to this work, which focus on the recommendation systems, such as API synthesis and service composition, have been published at international conferences, such as ICSE, ICWS, and in academic journals, such as IEEE Transaction on Services Computing, Journal of Computer Science and Technology, etc.