QSynth-TopK:一个支持 Top K 查询的质量敏感的自动服务组合系统

马环宇[1,2],姜 伟[1,2],虎嵩林[1]
(1.中国科学院计算技术研究所,北京 100190;2.中国科学院研究生院,北京 100049)

摘 要: 质量敏感的自动服务组合旨在根据用户的输入输出请求和服务质量约束,从海量的 Web 服务中快速搜索满足要求的最优组合方案,能够有效应用于服务组合辅助建模、正确性校验等领域.为了向用户提供更多的组合方案,以满足多样化的应用需求,并消除由于集中选择最优方案所带来的性能瓶颈隐患,我们在 QSynth 系统的基础上开发了一个支持 top k 查询的系统——QSynth-TopK.该系统通过对最优组合方案中的服务进行迭代地替换,实现全局服务质量的有序递减,以得到组合质量排名前 k 个的服务组合方案.理论证明该算法具有可靠的精确性,实验证明系统在大规模服务集上有着良好的、稳定的运行效率.

关键词: 自动服务组合; 服务质量; top k; 图搜索
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2012)10-1933-05
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.10.004

QSynth-TopK: A Top K Query Supporting System for QoS-Aware Automatic Service Composition

MA Huan-yu[1,2], JIANG Wei[1,2], HU Song-lin[1]
(1. *Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *Graduate University of Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract: QoS-aware automatic service composition is committed to synthesize the correct work plans from huge amounts of web services based on the functional quality requirements of the users. It is highly effective in fields such as service composition modeling, correctness checking, etc. Noticing that to give users more choices can meet the diversified demands and eliminate the possible performance bottle necks, we extend our tool, QSynth, to a new system QSynth-TopK which supports the query of top k work plans. In this system, top k work plans are generated orderly by iteratively replacing the web services in the optimal composition. In theory this way is proved accurate. Evaluations show that QSynth-TopK achieves stable and outstanding efficiency with respect to large composition scenarios.

Key words: automatic service composition; quality of service; top k; graph search

1 引言

随着面向服务的计算(Service Oriented Computing, SOC)应用的逐步展开,Web 服务数量的增长速度也在不断加快[1].在海量的服务资源中,手工选择服务并构造服务组合流程的难度越来越大,正确性校验工作也日趋繁琐.因此,自动服务组合技术应运而生,近年来日益引起学术界和工业界的重视,被 Amazon 和 SAP 等企业广泛应用在辅助建模[2]、组合正确性校验[3]等实际应用中.现有的自动服务组合研究致力于根据用户的需求求取最优的组合方案,在满足用户对组合流程的功能需求的基础上,提供最优的服务质量(包括响应时间,吞吐量,价格等).

然而,仅仅求取质量最优的组合服务往往无法满足实际应用场景的需求.用户对服务的选择往往还会受到许多其它因素的影响,并出于某种偏好而选择质量稍逊

收稿日期:2010-12-15;修回日期:2011-11-10
基金项目:国家基础研究项目(No.2007CB310805);国网信息通信有限公司科技项目(No.SGIT[2010]449);国家自然科学基金(No. 61070027);北京自然科学基金(No.4092043);广东省科技计划项目(No.2010B050100009)

但满足某些特殊需求的服务组合方案.因此,从用户的角度来看,生成服务组合质量较好的前 k 个服务组合结果供用户选择,可以满足多样化的应用需求,更具有吸引力;从服务提供者的角度来看,提供前 k 个服务组合结果也可避免由于集中选择最优方案所带来的负载不均等性能隐患.

虽然 top k 查询问题在关系数据库[4]、信息检索[5]、图论[6]等特定领域已经得到深入研究,但在服务组合领域,由于问题背景和抽象方式的特殊性和多样性,如 DAG 的不同组合模式、QoS 的不同聚合方式等等,这些工作难以得到直接应用.文献[7]提出了一种基于启发式规则的方法 WSCBT*,能够依次输出前 k 个服务组合,但不能达到 100% 的精确性,且在大服务集上的性能欠佳.

本文基于 QSynth[8]——一个提供最优质量敏感的自动服务组合的系统,设计并实现了一个支持 top k 查询的系统——QSynth-TopK.该系统能够精确地、快速地得到组合质量排名前 k 个的组合方案.

2 系统的总体设计

QSynth-TopK 的总体体系结构如图 1 所示.按照功能划分,可以划分成以下主要部分:

QoS 信息器 从 WSLA 文件中读取并分析 Web 服务的 QoS 信息.

初始化模块 响应客户端的初始化请求,从 OWL-S 文件中构建参数的语义本体树;结合 QoS 信息器的返回信息,通过解析 WSDL 文件来构建服务集.

服务组合触发器 响应并解析客户端的查询请求,调用服务组合算法来访问服务集,从而得到满足用户查询需求的、服务质量排名前 k 个的抽象的服务组合结果.

组合结果生成器 把服务组合触发器生成的服务组合结果,整合成客户端需要的形式,如 WS-BPEL 格式[9]或 DAG 图邻接表格式,返回给客户端,从而完成一次完整的服务组合过程.

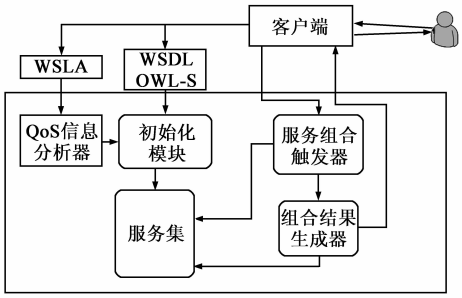

图1 QSynth-TopK的体系结构

3 质量敏感的自动服务组合问题模型

在首届全国 Web 服务竞赛(CWSC 2011)* 中,给出以下定义和描述:

Web 服务 (W_i): $W_i = \{I_{W_i}, O_{W_i}, Q_i\}$,其中 I_{W_i}是 W_i 的输入参数集合,O_{W_i}是 W_i 的输出参数集合,Q_i 是 W_i 的 QoS 值.在 CWSC 2011 中,只考虑单一 QoS 的情况,即 Web 服务的响应时间(Response Time).

参数匹配 两个 Web 服务参数 P_1 和 P_2 的匹配,不仅要根据其类型定义,也要依据其本体语义关系.令 Concept(P)为参数 P 所属的概念,则定义 P_1 可以匹配 P_2,当且仅当 Concept(P_a)等同于 Concept(P_b)或是 Concept(P_b)的子类.

Web 服务的匹配 定义 W_a 可以匹配 W_b,当且仅当 W_a 的部分输出参数可以匹配 W_b 的部分输入参数.其中,如果 $O_{W_a} \supseteq I_{W_b}$,则称 W_a 完全匹配 W_b;如果 $O_{W_a} \cap I_{W_b} \neq \varnothing$ 且 $O_{W_a} \cup I_{W_b}$,则称 W_a 部分匹配 W_b.

根据以上定义,在一组 Web 服务集之上,对于查询请求 $R = \{I_R, O_R\}$,定义 WP_{All}为可以满足 R 的所有服务组合;WP_{All} 中的每个服务组合定义了一组 Web 服务 $(W_1, \cdots, W_N)$的执行顺序,满足以下条件:

(1) $\{I_R \cup O_{W_1} \cup \cdots \cup O_{W_i}\} \supseteq I_{W_{i+1}}$ $(1 \leqslant i \leqslant N-1)$;

(2) $\{O_{W_1} \cup O_{W_2} \cup \cdots \cup O_{W_N}\} \supseteq O_R$.

4 基本算法描述

本节首先介绍 QSynth 生成 DAG(Directed Acyclic Graph,有向无环图)形式的最优服务组合的算法,接下来介绍基于最优 DAG 的关键路径生成 DAG 形式的 top k 服务组合的算法.

4.1 最优服务组合查询

QSynth 的最优服务组合生成算法分为两个步骤.第一步,使用一个前向过滤算法,从查询输入参数 I_R 开始,前向地遍历整个服务集,直至得到所有能够被触发的服务,形成了一个服务依赖子图,如图 2 所示.前向过滤算法同时能够实现两种级别的最优化:在服务级别,能够得到所有被触发服务的全局最优 QoS 值;在参数级别,所有被触发参数的全局最优 QoS 值也被记录.第二步,使用一个后向搜索算法来生成代表最优服务组合结果的 DAG.

4.1.1 前向过滤算法

前向过滤算法从查询输入参数 I_R 能够直接触发的服务开始,逐层在服务依赖图中搜索出能够被触发的

* 首届全国 Web 服务竞赛(CWSC 2011):http://debs.ict.ac.cn/cwsc2011/

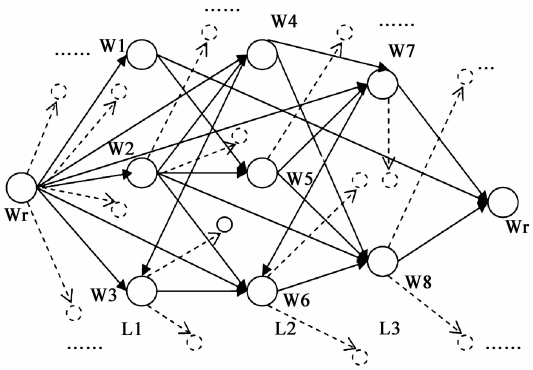

图2　前向过滤算法建立的服务依赖子图

服务.这样一方面有效地显著缩减了搜索空间,另一方面也为最优结果的生成提供了充分的条件.

前向过滤算法使用一个五元的数据结构来表示 Web 服务 W_i: *selfQoS* 记录 W_i 自身的 QoS 值; *allQoS* 记录自 W_i 被标记为可触发后其全局最优 QoS 值; *inputs* 和 *outputs* 两个列表分别存储 I_{W_i} 和 O_{W_i}; *counts* 记录 *inputs* 中未被触发的参数数目,初始化为 *inputs* 中的参数个数,当 *counts* = 0 时我们称 W_i 被触发.

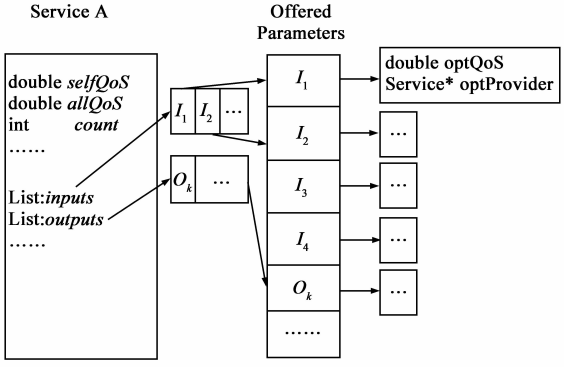

图3　前向搜索算法的数据结构

算法同时定义一些辅助数据结构.集合 *OPars*(Offered Parameters)用来记录任一参数的最优 QoS 值及其提供者的索引.随着算法的进行, *OPars* 会频繁更新.算法定义两个队列, *enabledServices* 和 *reProServices*,分别维护新进触发的服务和需要重新处理的服务.

如算法 1 和算法 2 所示,前向过滤算法开始于对 *enabledServices* 进行初始化,即把所有能够被 I_R 直接触发的服务添加到 *enabledServices* 中.接下来依次处理该队列中的服务,每次取出一个服务 W_i 之后,对于 O_{W_i} 中的每个参数,依据其是否在 *OPars* 中存在表项,分成两种处理方式.

如果该参数不存在于 *OPars*,则与 W_i 及其 *allQoS* 一起记录到 *OPars* 中.同时,把所有 *inputs* 中包含该参数的服务的 *counts* 值减 1.如果服务 W 的 *counts* 值因此等于 0,则称 W 被触发,添加到 *enabledServices* 中(算法 2 第 9 行-第 20 行).

如果 *OPars* 中已经有 O_{W_i} 的记录,则比较新近计算出的 *allQoS* 值与 *OPars* 中记录的值,若前者更优则进行更新.注意到更新操作会连锁地引起一部分服务的 *allQoS* 值需要更新,则把这部分服务放入 *reProServices* 队列中(算法 2 第 2 行-第 7 行).

在上述过程结束、*enabledServices* 为空时,需要处理 *reProServices* 队列,将其中的每个服务依照上述方式进行处理(算法 1 第 12 行-第 17 行).

算法 1　前向过滤算法

Require: *Service Repository*
1: **for all** Service $W_i \in$ *Service Database* **do**
2: $W_i.count \Leftarrow |I_{W_i}|$
3: **end for**
4: $enabledservices.add(I_R.trigger())$
/*The services that I_R can match*/
5: **while** $enabledservices \neq \varnothing$ **do**
6: $W \Leftarrow enabledservices.remove()$
7: $ForwardEnable(W)$
8: **end while**
9: **if** R is not enabled **then**
10: return no result.
11: **end if**
12: **while** $reProServices \neq \varnothing$ **do**
13: Service $W_{update} \Leftarrow reProServices.remove()$
14: **if** $W_{update}.calculateQoS()$ is better than $O_R.allQoS$ and $W_{update}.allQoS$ **then**
15: updateQoS(W_{update}.successors)
/*update the QoS value of the successors of W_{update}*/
16: **end if**
17: **end while**

算法 2　Forward Enable(W)函数

Require: Service: W, $OPars$, $reProServices$, Input inverted index table: inTable
1: **for all** parameter $out \in W.outputs$ **do**
2: **if** $\exists item \in OPars$ where $item = out$ **then**
3: **if** $item.optQoS$ is worse than $W.allQoS$ **then**
4: $item.optQoS \Leftarrow W.allQoS$
5: $item.optProvider \Leftarrow W$
6: $reProServices$.addUnique(inTable.lookup(out))
/*No duplicate services in $reProServices$.*/
7: **end if**
8: **else**
9: $OPars.add(out)$; $out.optProvider \Leftarrow W$;
$out.optQoS \Leftarrow W.allQoS$
10: A services set ($setNeed$) $\Leftarrow$ inTable.lookup(out)
11: **for all** Service $W' \in setNeed$ **do**
12: $W'.count$--
13: **if** $W'.count = 0$ **then**
14: $W'.allQoS \Leftarrow W'.calculateQoS()$
15: **if** $W' \neq O_R \wedge W'.allQoS$ better than $O_R.allQoS$ **then**
16: $enabledservices.add(W')$
17: **end if**
18: **end if**
19: **end for**
20: **end if**
21: **end for**

4.1.2　最优服务组合生成算法

在前向过滤算法结束后,我们可以通过一次从 O_R

出发、到 I_R 结束的后向搜索,来生成最优的服务组合结果.对于每个服务,通过其 *inputs* 和 *OPars* 中记录的参数最优提供者,找到其最优的前继.这样,我们生成了一个拥有最优 QoS 值的服务组合.

4.2 Top K 服务组合生成

在 4.1 中,通过前向过滤算法和后向搜索算法,能够得到 QoS 值最优的服务组合的 DAG 图.注意到 DAG 的 QoS 是由其关键路径唯一决定的——若更改 DAG 中的服务,只要关键路径不变,则 DAG 的 QoS 值维持不变.基于这个重要性质,我们可以通过更改最优 DAGs 的关键路径,来获得 QoS 值不同的新关键路径,进而扩充出新的 DAG.总之,通过上述基于关键路径的松弛操作,可以严格地、依次地、完全精确地得到 top 2 至 top k 的服务组合方案.

在图论里,寻找 DAG 的关键路径是通过拓扑排序进行的.在本文中,我们反其道而行之,首先通过返向迭代地寻找服务的关键前驱来得到关键路径,然后再由该关键路径获取.定义服务的关键前驱为其所有输入参数的最优 QoS 提供者中 QoS 值最差的服务.从 O_R 出发,反向地迭代寻找每个服务的关键前驱直至 I_R,便得到了 DAG 的一条关键路径.如果更改关键路径上某个服务的前驱、进而引起整个关键路径 QoS 的变差,则称其为一次松弛操作.

QoS 最优的 DAG 可能有多个,每个 DAG 的关键路径也可能有多条.取所有最优 DAG 的所有关键路径为算法的输入.对于每条关键路径,遍历其上的各个服务,找到所有的松弛方案,记录其松弛值.则最小的松弛值对应 top 2 服务组合的关键路径.依此类推,可以依次求出 QoS 值更差的关键路径.

每得到一条新的关键路径后,可以将其扩充为若干个完整的 DAG,加入结果集中.从关键路径到 DAG 的扩充,只需逆向遍历关键路径、依次添加未满足参数的提供者即可.

5 系统性能分析

为了评估 QSynth-TopK 的性能,进行了如下两组实验,目的是比较该系统在不同规模的测试集上的运行时间.实验运行环境为一台配置 Intel P8600(双核 2.4GHz)、4GB 内存和 Windows 7 64 位操作系统的电脑.

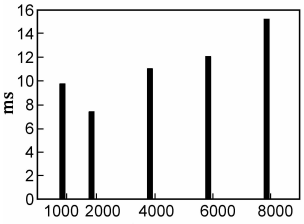

图4 服务数变化时的运行时间

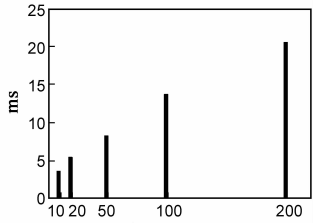

图5 k值变化时的运行时间

实验数据采用 CWSC 2011 提供的测试集生成器,通过改变服务个数来生成不同测试集.

第一组实验令 k 值为 50,分别取服务数目为 1000、2000、4000、6000、8000 生成 5 组测试集;第二组实验分别取 k 值为 10、20、50、100、200,在服务数为 4000 这组测试集上进行了实验.实验结果如图 4 及图 5 所示.可见,随着服务集规模的扩大和 k 值的增加,系统的运行时间没有显著地增长,保持在 ms 级别.实验证明,系统有着良好的、稳定的运行效率.

6 结束语

随着 Web 服务增长速度的加快和服务组合应用的扩展,学术界和工业界对自动服务组合技术日益提出更高的要求,在满足功能需求的同时也要考虑服务质量.Top k 查询能够为用户提供更多的组合方案,满足多样化的应用需求,并消除由于集中选择最优方案所带来的性能瓶颈隐患.本文设计并实现了一个支持 top k 查询的系统——QSynth-TopK,通过对最优组合方案中的服务进行迭代地替换,递减地得到组合质量排名前 k 个的服务组合方案.在理论上能够证明该算法具有可靠的精确性;实验证明系统有着良好的、稳定的运行效率.

参考文献

[1] E Al-Masri, Q H Mahmoud. Investigating web services on the world wide web[A]. Proceedings of the 17th International Conference on World Wide Web[C]. New York, NY, USA: ACM, 2008. 795 – 804.

[2] A Marconi, M Pistore, et al. Automated web service composition at work: The amazon/mps case study[A]. ICWS 2007[C]. Salt Lake City, UT, USA, 2007. 767 – 774.

[3] J Rao, D Dimitrov, P Hofmann, et al. A mixed initiative approach to semantic web service discovery and composition: Sap's guided procedures framework[A]. ICWS 2006[C]. Chicago, IL, USA, 2006. 401 – 410.

[4] Ihab F Ilyas, Walid G Aref, Ahmed K Elmagarmid. Supporting top-k join queries in relational databases[J]. The VLDB Journal, 2004, 13(3): 207 – 221.

[5] Buckley, Chris, Ellen M Voorhees. Evaluating evaluation measure stability[A]. SIGIR 2000[C]. New York, NY, USA: ACM, 2000. 33 – 40.

[6] D Eppstein. Finding the k shortest paths[J]. SIAM J Computing, 1998, 28(2): 652 – 673.

[7] X Wang, S Huang, A Zhou. QoS-aware composite services retrieval[J]. J Comput Sci Technol, 2006, 21(4): 547 558.

[8] W Jiang, C Zhang, Z Huang, M Chen, S Hu, Z Liu. Qsynth: A tool for qos-aware automatic service composition[A]. ICWS

2010[C]. Miami, FL, USA, 2010. 42 – 49.
[9] OASIS. Web Services Business Process Execution Language Version 2.0[EB/OL]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2009 – 04 – 01.

作者简介

马环宇　男,1986 年出生于河北.2009 年毕业于南开大学,同年进入中国科学院计算技术研究所,现为硕士研究生,主要研究方向为服务计算和分布式事件系统.
E-mail:mahuanyu@ict.ac.cn

姜　伟　男,1985 年出生于江西.2006 年毕业于中南大学,同年进入中国科学院计算技术研究所.现为博士研究生,研究方向为服务计算和分布式事件系统.
E-mail:jiangwei@ict.ac.cn

虎嵩林　男,副研究员,计算机学会高级会员,开放系统专业委员会委员.1973 年出生于山西.2001 年在北京航空航天大学获博士学位.2002 年进入中国科学院计算技术研究所工作.研究方向为分布式系统与中间件、服务计算、事件计算.
E-mail:husonglin@ict.ac.cn