基于 MapReduce 的并行 Web 服务自动组合

黄龙涛,邓水光,戴 康,李 莹,尹建伟
(浙江大学计算机科学与技术学院,浙江杭州 310027)

摘 要: 如何在大规模的 Web 服务集合中进行快速、高效的自动组合是当前 Web 服务组合研究与应用的难点.传统的 Web 服务自动组合方法大多建立在单机计算基础上,服务数量一旦过多,规划或搜索空间随之膨胀,组合效率低下.本文提出了一种分步分治、深度优先搜索的 Top-k Qos 服务组合算法,并采用 MapReduce 实现了分布式、并行的服务自动组合过程.实验结果表明,该方法在应对大规模的服务集合时,能快速、高效的提供满足用户需求的组合服务.

关键词: Web 服务; 服务组合; MapReduce; 回溯树
中图分类号: TN311 **文献标识码:** A **文章编号:** 0372-2112 (2012) 07-1397-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.07.018

Automatic Service Composition in Parallel with MapReduce

HUANG Long-tao, DENG Shui-guang, DAI Kang, LI Ying, YIN Jian-wei
(*College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang* 310027, *China*)

Abstract: How to compose services automatically and efficiently is a difficult issue, especially for a large number of services. Traditional methods based on single-computation usually come to inefficiency due to the explosion of the planning and searching space when the number of services grows too much. Based on the MapReduce framework, this paper proposes an automatic service composition method based on depth-first searching for the Top-k Qos service composition issue. The result from a serial of experiments indicates that the method can satisfy composition requirements quickly and efficiently even with a large-scale service repository.

Key words: web service; service composition; MapReduce; backward tree

1 引言

近年来,Web 服务以其良好的重用性和平台无关性得到了广泛应用.由于自动服务组合能充分重用已有服务,具有即时、快速、灵活组建分布式松耦合的新应用和服务的特点,能满足用户多样化、个性化和多变的需求,可提高已有服务的利用率和 IT 资源的使用价值,因此,服务组合技术已成为实现云计算按需服务的关键技术与研究热点,并作为建设、推广和应用"云"的基础技术而备受关注.

目前,自动服务组合得到了众多学者的高度关注,已有大量工作见诸于各类文献,也有学者对其进行综述[1,2].纵观已有的自动组合方法,大多借鉴情态演算、规划语言、图或树等形式化方法进行建模[3~6],将组合问题转化为规划或搜索问题.而随着服务数量的增加,规划或搜索空间也随之膨胀,造成组合效率低下.为此,本文作者曾提出了一种基于回溯树的服务自动组合方法,该方法采用分步分治思想,减小搜索空间,降低复杂度,在一定程度上提高了组合效率[7].但该方法还是建立在单机计算的基础上,即服务组合过程都在单一计算节点上完成,而当服务数量过于庞大时,其性能还是存在瓶颈.因此,如何提高已有方法的可扩展性和性能成为当前自动服务组合研究与应用的一个重要问题.目前,已有学者开始关注该问题并提出多线程方式提高组合效率[8],但并未脱离单机计算的模式,因而性能提升极为有限.而近年来出现的多核计算、并行计算技术为解决这一问题提供了一条有效途径.

MapReduce 是 Google 公司提出的一种用于处理大规模数据集并行运算的技术框架[9].由于该框架隐藏了并行化、容错、数据分布、负载均衡等诸多细节,极大地

收稿日期:2011-06-21;修回日期:2011-07-15
基金项目:国家自然科学基金(No.61170033);国家科技支撑计划课题(No.2011BAD21B02,No.2011BAH16B04)

方便了编程人员实现分布式并行化的应用开发.本文针对首届全国 WEB 服务竞赛提出的"Top-k QoS 自动服务组合"的要求,在对文献[7]提出的回溯树构造方法进行了重新设计和优化之后,继承了其分步分治思想,利用 MapReduce 框架实现了分布式的并行化服务自动组合过程.利用竞赛提供的测试集进行实验,结果表明该方法在应对大规模的服务集合时,能快速、高效的提供满足用户需求的组合服务.

2 问题描述及定义

本届 CWSC2011 竞赛提供了大规模的测试集,所有 Web 服务由服务描述文件(WSDL)描述,并对这些 Web 服务的输入输出对象进行了语义标注,语义标注信息存储在 OWL 文件中,此外 WSLA 文件描述了服务的 QoS 信息.参赛者以这三个文件作为输入通过自己设计的服务组合方法从这大规模的测试集中组合出 QoS 最佳的前 k 个组合服务.为便于描述问题,本节对基本概念作如下定义.

定义 1 组件服务 组件服务是构成服务组合的基本单元,表示成三元组 $s=(n,P,Q)$,其中:

(1)n 为服务名称;

(2)P 表示为该服务提供的操作集合,一个操作可表示为二元组 $p=(C_I,C_O)$,其中 C_I 为操作所有的输入对象所对应的语义概念集合,C_O 为所有输出对象所对应的语义概念集合;

(3)Q 为服务 QoS 属性集合 $Q=\{q_1(s)\cdots q_n(s)\}$,$q_1(s)$表示该服务的第 i 项 QoS 属性值.

QoS 属性一般包括服务响应时间、吞吐量、可用性和可靠性等.本次竞赛只考虑响应时间,记作 rt,且所有的服务只包含一个操作,因此一个组件服务可简化为 $s=(n,C_I,C_O,rt)$.

定义 2 组合服务 组合服务是由一系列组件服务按照一定的业务逻辑组合而成,表示为一个二元组 $sc=(S,A)$,其中:

(1)S 为组合服务包含的所有组件服务的集合;

(2)A 为各组件服务参数间的依赖关系集合 $A=\{a(s_i,s_j)\mid s_i,s_j\in S\}$,其中 $a(s_i,s_j)$表示 s_j 的输入依赖于 s_i 的输出.

根据 Web 服务的相容性[10],当一个语义概念 c_a 是 c_b 的子类,则 c_a 可赋值给 c_b,即 c_b 可依赖于 c_a;反之则不成立.

Top-k QoS 自动服务组合问题即针对用户请求 $R=(I,O)$<其中 I 为用户提供的输入集合,O 为期望得到的输出集合>,从组件服务集合中构造出全局 QoS 最佳的前 k 个组合服务,使得每一个组合服务接收用户输入的参数,通过调用其组件服务并进行参数传递,最后输出用户期望的输出集合.根据竞赛要求,组合服务须表示成有向无环图(DAG),并采用邻接表描述.

3 服务组合算法

本文提出的服务自动组合算法借鉴了文献[7]的分步分治思想,即分别为用户请求的每一个输出概念建立回溯树,并在建立的过程中求出该概念的 Top-k 条服务链,最后将服务链进行合并形成组合服务.由于该方法是为求解一个组合服务而设计的,直接用于 Top-k 问题求解时,一旦服务集合过大,加之此次竞赛提供的服务的输入/输出参数较多,回溯树的中间状态数量极易膨胀,造成内存溢出,求解效率低下,为解决这一问题,本文对产生式规则和回溯树的构造进行了改进和优化.

3.1 产生式规则

文献[7]在将服务转化为产生式规则时,是分别针对服务的每一个输出各生成一条从该服务的输入集合到该输出的规则.一方面造成规则数量过大,远远超出了服务数量,这给回溯树的构造带来了较大的时间开销;另一方面,由于每条规则仅产生一个输出,因此在回溯树构造时,节点只能针对其包含的每一个输出单独进行回溯,而不能对多个输出采用一条规则进行回溯,极易造成节点扇出过大,这给回溯树的构造带来了巨大的空间开销.为此,我们将服务转化为一条从输入集合到输出集合的规则,即一个服务对应一条规则,规则数与服务数相同.以此次竞赛提供的测试服务集合为例,规则数可减少至原来的 15% ~20%.

3.2 回溯树的构造与 Top-k 问题求解

回溯树的构造方法是问题求解的关键.文献[7]在为用户请求的每个输出构造回溯树时,采用广度优先的策略,即对当前节点的对象集合所包含的每一个输出对象,分别利用规则穷举所有可能回溯情况,每一种可能回溯则新增一个儿子节点.这样虽然能确保找到所有的回溯分支,但产生了巨大的空间开销.在对此次竞赛提供的一个 4000 个服务集进行实验时,当回溯树构造到第三层时就达到上百万个节点,空间开销急剧增长,算法难以继续运行.究其原因,除了 3.1 所述的规则数过多和规则单一输出所引起节点扇出过大之外,一个重要原因在于广度优先须保留所有的中间节点信息.为此,本文采用深度优先策略构造回溯树,并针对 Top-k 问题,可在构造过程中利用服务的 Qos 信息进行启发式的选取,只对 QoS 最佳的节点进行回溯,从而确保在短时间内找到近似 Top-k 的最佳生成路径或服务链.

对用户请求 $R=(I,O)$,我们首先将 R 进行语义转化,即把用户请求中的对象,转化成其所对应的语义概

念,转化后的请求为 $R'=(C_I,C_O)$.之后对每个 $c_i\in R$. C_O 构造回溯树 $t_{ci}=(V,\chi,E)$,其中 V 为回溯树中的节点集合,χ 为节点的标识函数,E 为有向边的集合.由于此次竞赛提供的每个服务只有一个操作,因而每条有向边就代表一个服务,生成路径即为服务链.在构造的过程中,每新生成一个节点时,判断当前节点是否为 I 的子集,若是,则表示找到了一条服务链,进而继续寻找下一条.基于深度优先的回溯树构造和 Top-k 服务链求解算法 CBTC-DFS 如表 1 所示.

表 1　深度优先的回溯树构造与 Top-k 问题求解算法

ALGORITHM: CBTC-DFS

Input: $R_S=(C_I,C_O,R)$(服务规则库),$c_i\in R$. C_O,k(Top-k 的 k 值),$m(R,C_O$ 中的元素个数),*maxdepth*(回溯树的最大深度).
Output: P(服务链集合)

```
01: set tStack as a stack of node;
02: set tNode as a node;
03: set the root of t = (V, χ, E) as o;
04: set t = 0;
05: V.add(o); //add the root node into the set V
06: tStack.push(o);
07: while (! tStack.empty() && t ≤ N {
08:    tNode = tStack.top(); //get the top node;
09:    set l = {e1, e2, ···, en} as the path from tNodeto the root;
10:    if(χ(tNode) ⊆ CI ∪ SuperClass(R, CI)){
11:    |   P.add(1);
12:    |   t + +;
13:    |   tStack.pop(tNode);
14:    |   continue;
15:    }
16:    if(tStack.size() < maxdepth){
17:    |   set O^tNode = χ^(tNode) ∩ RS, CO;
18:    |   set DNodes = r, Od ∩ O^tNode;
19:    |   set CRules as a list of rules
20:    |   for each r ∈ RS{
21:    |   |if (DNodes! = ∅ && r, OS ∩ (U_{ei∈l} ei, Od) = ∅){
22:    |   |   |   CRules.add(r);
23:    |   |   }
24:    |   }
25:    |   set r as the rule that has the i^th shortest response time in CRules
           //Iis the backward times of tNode
26:    |   create a new node cNode;
27:    |   setχ(cNode) = {χ(tNode) - DNodes} ∪ r, OS;
28:    |   tStack.push(cNode);
29:    |   V.add(cNode);
30:    |   create an edge e = (cNode, tNode, r);
31:    |   E.add(e);
32:    }
33:    else{
34:    |   tStack.pop(tNode);
35:    |   continue;
36:    }
37:}
38:return P
```

首先构造一个堆栈 *tStack*,从根节点 c_i 开始依次将回溯的中间节点 push 到堆栈中.

采用深度优先策略构建回溯树,在对每一个节点回溯时,首先对能够回溯该节点的所有规则(服务)的 QoS 进行排序,同时记录每个节点被回溯的次数,如果是第 i 次,则选择 QoS 排名第 i 的规则进行回溯,这样基本能够保证 QoS 表现最佳的服务能优先被选取进行节点扩展.然后将新扩展出的节点 push 进 *tStack* 中.

在构建过程中,当堆栈顶节点的概念集合是 $R,C_I\cup SuperClassOf(R,C_I)$ 的子集时,表明用户所提供的输入能够经过一系列的服务产生概念 c_i,此时连接栈内的所有节点的规则构成一条能产生概念 c_i 的服务链.随后将该节点 pop 出栈,新的栈顶节点则继续选取 QoS 排名第 $i+1$ 的规则进行回溯.若栈顶的节点没有任何规则可用于扩展,同时也没有产生服务链,则也将该节点 pop 出栈,再重新对新的栈顶节点回溯,如此循环直到堆栈为空时构造结束.

理论上讲,要完全准确的找出 Top-k 个结果就需要得出所有可能的服务链,再根据其 QoS 表现进行排序.但是计算出所有可能的服务链会产生大量的时间开销,且考虑到本次竞赛只需要找出 Top-k 个组合结果,因此可设定一个阈值 N,当计算出 N 条服务链时,即停止回溯树构造并返回服务链.N 设定的越大则最终计算结果越准确,相应的时间开销也越多,因此需要进行折中选择,既保证较高的准确率,又能在尽量短的时间内完成计算.由于在进行服务链构造时,当前节点都是优先按照响应时间最短的规则(服务)进行回溯,因此先找出的服务链通常是要比后找出的总响应时间短.此外,为了减少构造回溯树的时间开销,本文对回溯树的最大深度通过变量 *maxdepth* 进行限定,当回溯树的深度(或 *tStack* 栈的大小)超过 *maxdepth* 时,则停止构建. *maxdepth* 设定的过小会导致无法找到解,设的过大会降低处理效率.因此,在具体实现时,我们可以先将 *maxdepth* 设为一个较小的值,从而可在尽量短的时间内得出 N 条 QoS 表现最优的服务链;若没有找到解,则可加大 *maxdepth*,直到找到 N 个解.分别为每个用户请求输出对象计算出 N 条按 QoS 排序的服务链之后,再将每个不同输出的服务链进行组合,从而得到满足用户请求的组合服务,再将这些服务组合根据 QoS 表现筛选出最终的 Top-k.

4　分布式并行服务组合框架

传统的自动服务组合在技术实现上,通常依赖于集中式的服务组合引擎或规划器.一旦服务规模过大时,导致搜索或规划空间膨胀,易引起组合响应时间过长或无法求解的问题.由于本文的服务组合算法利用了分步分治思想,便于并行化处理,因此基于 MapReduce 提出了如图 1 所示的分布式并行自动服务组合框

架.该框架由一台主机(Master)和若干工作机(Worker)协同完成服务的自动组合,其过程依次为:

4.1 Map 过程

Master 接收到服务组合请求 R 之后,在考虑负载均衡基础上,将任务分发给各 Worker 机,任务的数据结构可表示为四元组 $t=(c_i,R,C_I,N)$,其中 $c_i\in R$,C_O,为分配给当前 Worker 处理的一个用户请求的输出,N 表示要求返回给 Master 的服务链数量.

4.2 并行处理过程

Worker 接收到任务 $t=(c_i,R,C_I,N)$后,根据算法 CBTC-DFS 为 c_i 构造回溯树并返回按 QoS 排序的 N 个服务链.最后结果组织成 *Key-Value* 映射对返回给 Master,*Key* 为用户请求的某个输入对象 c_i,Value 为计算出的按 QoS 排序的 N 个服务链,此外每个服务链的 QoS(总响应时间)也一并存入 Value 中返回给 Master.

4.3 Reduce 过程

Master 负责将来自不同 Worker 的服务链进行合成产生组合服务,进一步判断每个组合服务的不同服务链上是否有相同组件服务,如果有,则 Master 对这些服务链进行合并形成最终的组合服务.最后根据组合服务的 QoS,排序选出最佳的 k 个返回给用户.具体的 Reduce 流程如下:

Master 首先从每个 *Key-Value* 映射对中都选出 QoS 表现最佳的服务链,将这些服务链进行合并得到一个服务组合,该服务组合即为计算出的 QoS 最优(总响应时间最短)的服务组合,记作 sc_o.在最优服务组合 sc_o 中定义一条关键服务链 p_k,该服务链为服务组合中所有服务链响应时间最长的,也就是说服务组合的总响应时间即为该服务链的响应时间.

然后分别从 *Key-Value* 映射对的服务链中挑选 QoS 次优的服务链进行替换,得到次优的服务组合.替换的服务链遵循如下优选原则:

(1)若替换后服务链的响应时间仍小于关键服务链的响应时间,则不会改变服务组合的总响应时间,优先选取这类服务链进行替换.

(2)若替换后服务链的响应时间大于关键服务链的响应时间,则优先选取造成服务组合总响应时间增值最小的服务链进行替换,此时所选取的替换服务链成为新的服务组合中的关键服务链.

这样依次得到服务组合结果即为按 QoS 排序的结果.每当获得一个服务组合结果时,都进行一次冗余性检验,该检验包含两个方面:一方面是自身冗余,即该服务组合结果中是否含有重复服务,若有则进行服务链合并去除冗余服务.另一方面是服务组合间的冗余性检验,即新得出的服务组合与已有服务组合进行比较,若存在已有服务组合为该服务组合的子集,则视该服务组合为冗余服务,将其排除在最终结果之外.最后得到无冗余的 k 个服务组合即为满足用户请求的 Top-k 个服务组合.

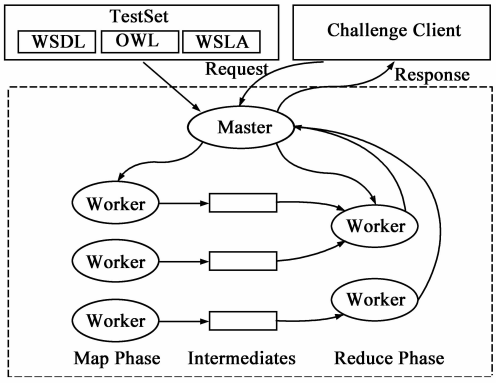


图1 基于MapReduce的并行服务组合框架

5 实验与验证

5.1 实验结果

为验证本文的服务组合方法的有效性,我们利用 WS-Challenge2009 提供的服务测试集生成器 Generator*,根据不同的参数设置产生一系列不同的测试集进行实验.事实上,该生成器正是本届 CWSC2011 竞赛所提供的测试集的生成器.由于测试集的生成受服务数量(SN)、概念数量(CN)、组合解深度(Solution-Depth)三个参数的影响,为考查本文服务组合方法的性能与这三个参数的关系,我们按照表 2 的参数设置生成三组测试集,每组固定两个参数而变动一个参数,如第一组服务集合的概念数、解的深度分别固定为 8000 和 8,服务数量则从 2000 增长到 20000,步长为 2000.

表 2 生成三组测试集的参数设置

	SN	CN	Solution-Depth
Group1	2000 ~ 20000	8000	8
Group2	8000	2000 ~ 20000	8
Group3	8000	8000	5 ~ 20

我们采用 JDK1.6 实现本文组合算法**,利用 Hadoop 通过一台 Master 主机和 3 台 Worker 主机搭建 MapReduce 并行环境,主机配置均为英特尔酷睿 i3-550 3.2G 处理器,2G RAM,基于上述在 3 组测试集求解 Top-10 问题(即 $k=10$),三组实验结果分别如图 2 ~ 4 所示.

图 2 给出了固定概念数量为 8000,解组合深度为 8,从服务数量 2000 开始,并以 2000 为步长,使用测试集生成工具生成的一组测试集,并对这组测试集求 Top-10 服务组合方案的计算时间开销.从实验结果可以

* http://ws-challenge.georgetown.edu/wsc09/

** 算法和测试集可从 http://mypage.zju.edu.cn/shuiguang 下载

看出,算法时间随着服务数量增加呈线性增长趋势.

图 3 给出了固定服务数量为 8000,解组合深度为 8,概念数量从 2000 到 20000,并以 2000 为步长,使用测试集生成工具生成的一组测试集,并对这组测试集求 Top-10 服务组合方案的计算时间开销.从实验结果可以看出,概念数的变化对算法无明显影响.

图 4 给出了固定概念数量为 8000,服务数量为 8000,解组合深度从 5 开始,并以 1 为步长,使用测试集生成工具生成的一组测试集,并对这组测试集求 Top-10 服务组合方案的算法时间开销.从图中可以看出,整体上讲,解的组合深度变化对算法无明显影响.

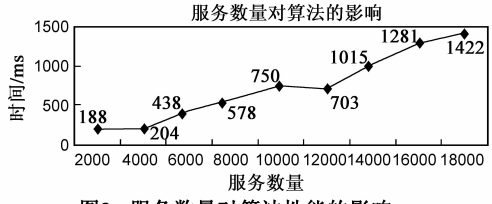


图2　服务数量对算法性能的影响

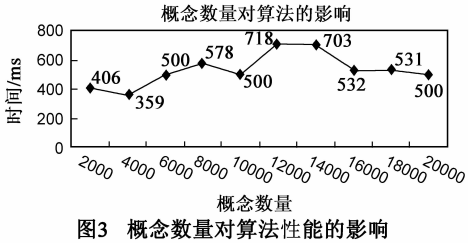


图3　概念数量对算法性能的影响

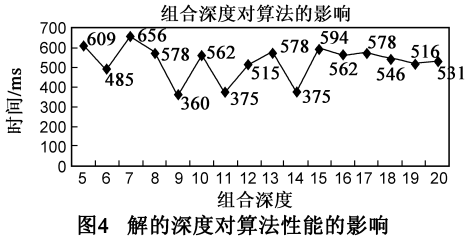


图4　解的深度对算法性能的影响

图 5 展现了对服务数 8000、概念数 8000 的测试集求解时,一台 Work 主机上依次产生的服务链的整体 QoS 值.横坐标表示按时间顺序找出的服务链序号,纵坐标为每条服务链的整体 QoS 值.结果表明,服务链的 QoS 随服务链序号呈整体正相关,即越先找出的服务链在 Qos 上整体上优于后找出的服务链,因此得到的组合服务也是越逼近 Top-k.

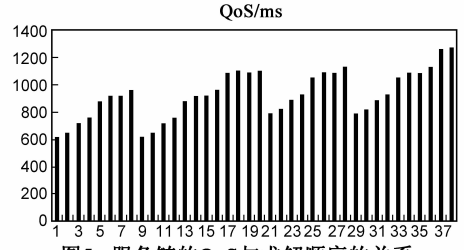


图5　服务链的QoS与求解顺序的关系

综合上述实验结果,我们认为本文方法能应对大规模服务集合的自动组合问题,并且能快速返回满足用户请求的 Top-k 服务组合解.

5.2　竞赛结果

本次竞赛所采用的计算环境为:CPU:Pentium(R) D CPU 3.40GHz 3.39GHz;内存:1GB;操作系统:Windows XP Professional.此次竞赛共生成了 5 组测试集,作为竞赛实验使用.5 组测试集的生成参数,如下:

表 3　竞赛提供的服务测试集

	服务数	概念数
测试集 1	8000	4000
测试集 2	10000	6000
测试集 3	4000	40000
测试集 4	15000	100000
测试集 5	40000	20000

应组委会要求,为了保证竞赛的公平性,我们将所提交的系统改成了单机单线程版本.该版本系统将原有的并行处理步骤改为串行处理,在一定程度上会影响运行效率,但并不会影响计算所得出的服务组合结果.表 4 展示了利用本文所提出的算法所得到的 Top-10 个服务组合结果与竞赛给出的实际 Top-10 个服务组合结果的对比,所计算出的 Top-10 服务组合结果有 54% 符合实际 Top-10 结果,有 80%符合实际 Top-20 结果.

表 4　系统运行得出的 Top-10 结果与实际结果的对比

	测试集 1		测试集 2		测试集 3		测试集 4		测试集 5	
	计算结果	实际结果	计算结果	实际结果	计算结果	实际结果	计算结果	实际结果	计算结果	实际结果
Top 1	790	790	1740	1740	2813	2813	2630	2560	3340	3340
Top 2	840	840	1740	1740	2813	2813	2630	2560	3340	3340
Top 3	860	850	1750	1750	2813	2813	2630	2560	3340	3340
Top 4	900	850	1750	1750	2813	2813	2630	2560	3340	3340
Top 5	910	860	1780	1750	2813	2813	2630	2560	3360	3340
Top 6	930	900	1780	1770	2813	2813	2630	2560	3360	3340
Top 7	950	910	1790	1770	2816	2816	2694	2590	3360	3340
Top 8	970	930	1790	1780	2816	2816	2694	2590	3360	3340
Top 9	980	950	1790	1780	2816	2816	2694	2590	3360	3340
Top 10	1000	950	1790	1780	2816	2816	2694	2590	3390	3340

该系统在竞赛中运行时间开销的结果如图 6 所示,基本能够在秒级以内完成大规模数据量的服务组合问

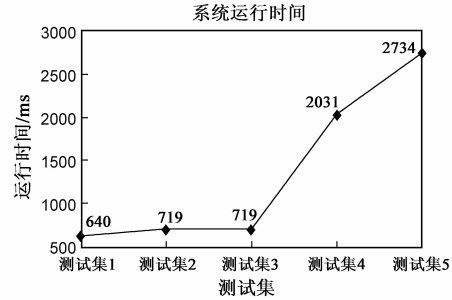


图6　系统运行5组测试集的时间开销

题,并能给出较准确的结果.

6 相关工作

近年来,作为 Web 服务研究领域的一个热点,自动服务组合受到了广泛关注,其研究成果也是层出不穷.

现有的大多数服务组合方法都是基于图搜索的方法.文献[11]论述了一种基于接口匹配的 Web 服务自动组合方法,该方法利用了服务本体关系图,其描述的场景与本次竞赛也十分接近,同样要解决图的规模过大所造成的搜索空间膨胀问题.文献[12]将服务组合问题转化为图搜索问题,再使用类似 A * 搜索算法查找组合方案.这类方法的不足在于,随着服务数量的增加,规划或搜索空间也随之膨胀,造成组合效率低下.

也有一些服务组合方法不仅考虑功能属性,同时也引入了非功能属性.文献[13]就对服务组合的功能性和非功能性属性同时做了考虑.该算法在功能性是否匹配的判断上,是基于建立模型和符号推演的过程;而在非功能性的判断上,是根据用户提供的一个临界值去对 Web 服务进行筛选的.由于其对非功能性属性 Qos 是以阈值的形式筛选,对求解 Top-k 问题有一定的局限.而文献[14] 提出了一个基于动态描述逻辑的 Web 服务自动组合框架,基于该框架给出了一个支持非线性 QoS 聚合和显式数据流声明的 QoS 模型.除了考虑服务组合的 QoS 属性,也有学者[15]将信任引入服务组合,通过把信任度作为组合服务实现调度和绑定的依据,进而提高组合服务的可信性.

7 结束语

Web 服务组合是服务计算研究的一个热点和难点问题.为提高服务自动组合效率,本文借鉴前期工作[7]的分步分治思想,提出了深度优先的组合搜索算法,并利用 MapReduce 框架实现了分布式、并行的自动服务组合过程.一系列实验表明该方法能够在大规模的服务集合中,进行快速准确的自动组合以满足用户 Top-k 的服务组合需求,但在时间开销上以及准确性方面仍存在改进空间,如预先过滤不可能出现在最终结果的服务的方法,可进一步减小搜索空间,提高组合效率.

参考文献

[1] Dustdar S, Schreiner W. A survey on web services composition [J]. International Journal of Web and Grid Services, 2005, 1 (1): 1 – 30.

[2] 邓水光,黄龙涛,尹建伟,等.Web 服务组合技术框架及其研究进展[J].计算机集成制造系统,2011,17(2):404 – 412.
Deng Shuiguang, Huang Longtao, Yin Jianwei, et al. Technical framework for web services composition and its progress[J]. Computer Integrated Manufacturing Systems, 2011, 17(2): 404 – 412. (in Chinese)

[3] Mohamad El Falou, Maroua Bouzid, Abdel-Illah Mouaddib, et al. Automated web service composition using extended representation of planning domain[A]. IEEE International Conference on Web Services 2008[C]. America: IEEE, 2008. 762 – 763.

[4] Stephan Reiff-Marganiec, Chen Kun, Xu Jinyu. Markov-HTN planning approach to enhance flexibility of automatic web services composition[A]. IEEE International Conference on Web Services 2009[C]. America: IEEE, 2009. 9 – 16.

[5] Wolfgang Mayer, Rajesh Thiagarajan, Markus Stumptner. Service composition as generative constraint satisfaction[A]. IEEE International Conference on Web Services 2009[C]. America: IEEE, 2009. 888 – 895.

[6] Zhao Haibo, Prashant Doshi. Towards automated RESTful web service composition [A]. IEEE International Conference on Web Services 2009[C]. America: IEEE, 2009. 189 – 196.

[7] 邓水光,吴健,李莹,等.基于回溯树的 Web 服务自动组合[J].软件学报,2007,18(8):1896 – 1910.
Deng Shuiguang, Wu Jian, Li Ying, et al. Automatic web service composition based on backward tree[J]. Journal of Software, 2007, 18(8): 1896 – 1910. (in Chinese)

[8] Wolf-Tilo Balke, Patrick Hennig. Highly scalable web service composition using binary tree-based parallelization[A]. IEEE International Conference on Web Services 2010[C]. America: IEEE, 2010. 123 – 130.

[9] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 1958 – 2008.

[10] 殷昱煜,李莹,邓水光,尹建伟.Web 服务行为一致性与相容性判定[J].电子学报,2009,37 (3):433 – 438.
YIN Yu-yu, LI Ying, DENG Shui-guang, YIN Jian-wei. Determining on consistency and compatibility of web services behavior[J]. Acta Electronica Sinica, 2009, 37 (3): 433 – 438. (in Chinese)

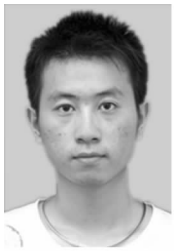
[11] RAO J. Semantic web service composition via logic-based program synthesis[D]. Norway: Department of Computer and Information Science, Norwegian University of Science and Technology. 2004.

[12] Seog-Chan Oh, Byung-Won On, Eric J Larson, et al. BF *: Web services discovery and composition as graph search problem[A]. IEEE International Conference on e-Technology, e-Commerce and e-Service 2005 (EEE'05)[C]. America: IEEE, 2005. 784 – 786.

[13] Pathak J, Basu S, Honavar V. Modeling web services by iterative reformulation of functional and non-functional requirements[A]. International Conference on Service Oriented Com-

puting 2006[C]. Berlin:Springer,2006.314 – 326.
[14] 万长林,韩旭,牛温佳,等.基于动态描述逻辑的服务组合及质量模型[J].电子学报,2010,38(8):1923 – 1928.
WAN Chang-lin, HAN Xu, NIU Wen-jia, et al. Dynamic description logic based web service composition and QoS model [J]. Acta Electronica Sinica, 2010, 38(8): 1923 – 1928. (in Chinese)
[15] 王勇,代桂平,姜正涛,等.信任增强的服务组合调度算法[J].电子学报,2009,37 (10):2234 – 2238.
WANG Yong, DAI Gui-ping, JIANG Zheng-tao, et al. A trust enhanced service composition scheduling algorithm[J]. Acta Electronica Sinica, 2009, 37 (10): 2234 – 2238. (in Chinese)

作者简介



黄龙涛　男,1988 年 2 月出生于河北廊坊.2010 年本科毕业于浙江大学软件工程专业.现为浙江大学计算机学院直博生,主要研究领域:服务计算、云计算.
E-mail:hlt218@zju.edu.cn

邓水光(通信作者)　男,1979 年 7 月出生于湖南衡山.2002 年、2007 年分别在浙江大学计算机学院获工学学士和工学博士学位.现为浙江大学计算机学院副教授、硕士生导师,主要研究领域:服务计算、云计算、中间件技术.
E-mail:dengsg@zju.edu.cn