Journal of Computer Applications
计算机应用,2013,33(8):2167 -2170,2183
ISSN 1001-9081
CODEN JYIIDU
2013-08-01
http://www.joca.cn

文章编号:1001-9081(2013)08-2167-04
doi:10.11772/j.issn.1001-9081.2013.08.2167

基于服务簇和 QoS 的 Web 服务组合方法

邓式阳[1,2*],杜玉越[2]
(1.潍坊学院 计算机工程学院,山东 潍坊 261061; 2.山东科技大学 信息科学与工程学院,山东 青岛 266590)
(*通信作者电子邮箱 gmdsy@163.com)

摘 要:针对海量的语义 Web 服务组合中如何提高搜索速度和获得最优组合的问题,提出了一种基于服务簇和服务质量(QoS)的快速组合方法。利用预先建立的服务簇进行服务搜索,可以有效缩小搜索空间,降低语义比较的复杂度,快速得到候选服务集合。组合过程中基于服务的最优组合 QoS 值动态确定阈值进行服务过滤,可以获得多个最优组合。同时采用高效的冗余处理方法保证组合中冗余服务最少,并采用服务簇内部过滤的方法限制候选服务数量,解决了因组合过多造成的运行超时问题。基于海量服务库进行实验,结果表明,服务搜索效率比普通搜索方法可提高几十倍,服务过滤和冗余处理效果明显,可快速获得多个 QoS 最优的无冗余组合,并且在百万级服务库进行大层次深度的组合时仍可保证良好的运行性能。

关键词:Web 服务组合;服务簇;服务质量;冗余处理
中图分类号:TP311 **文献标志码**:A

Web service composition approach based on service cluster and QoS

DENG Shiyang[1,2*], DU Yuyue[2]
(1. *School of Computer Engineering, Weifang University, Weifang Shandong* 261061, *China*;
2. *College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao Shandong* 266590, *China*)

Abstract: To improve the searching speed and get optimal services compositions on large scale of semantic Web services, a quick composition approach based on service cluster and Quality of Service (QoS) was proposed. Using the pre-built service clusters, it could quickly get the candidate service set with the effectively reduced searching space and semantic comparison complexity. It could obtain more optimal compositions by filtering service with the dynamically determined threshold based on the best composition QoS in the process of composition. It adopted an effective redundancy processing method to ensure minimum redundant services were used in the composition, and a service cluster internal filtering method was used to limit the number of candidate services, that solved the operation overtime problems caused by too many compositions. The results of experiments performed on large scale service storage illustrate that the searching speed is improved dozens of times than common methods, and the effectiveness of service filtering and redundancy processing is remarkable, so that the approach can quickly get multiple QoS optimal and non-redundant service compositions, and perform well on deep hierarchy composition in service storage of millions level.

Key words: Web service composition; service cluster; Quality of Service (QoS); redundancy processing

0 引言

随着 Web 服务技术的发展与应用的普及,如何利用现有服务进行自动组合以满足用户的需求,成为当前的研究热点。Web 服务组合主要包含两个方面的问题:一是如何在海量服务库中快速搜索候选服务;二是如何构建最优服务组合,使冗余服务最少、服务质量(Quality of Service, QoS)最优。

由于很多服务具有功能和接口的相似性,在海量服务库中,利用 Web 服务聚类来缩小服务搜索空间,是提高服务发现效率的一种有效方法。但多数文献主要关注聚类方法的研究[1-3],很少给出具体的基于服务簇的服务发现与组合方法。文献[4]提出了一种基于服务簇的 QoS 全局最优服务动态选择算法,其服务簇要求服务的参数接口一致,聚类粒度较小,服务组合的适应性不高。文献[5-6]给出了在组合中去除冗余服务的一般方法:以用户请求的输出参数为起点逆向求解,将不能为下一层服务提供输入参数的服务去除。文献[7-8]在求解最优服务组合问题时考虑了服务 QoS 的影响,但其对组合服务的 QoS 计算只考虑了简单的求和方法,局限性较大。文献[9-10]提出了最优组合 QoS 的概念和算法,根据每个参数的最优源进行逆向搜索得到 QoS 最优的组合服务,该方法仅得到一个最优组合,用户的选择余地较小。

笔者在文献[11]中提出一种基于逻辑 Petri 网[12-15]的 Web 服务簇模型,以逻辑向量表示服务簇与服务的关系,使得基于语义相似度的参数匹配仅在服务簇层进行,在服务簇内通过逻辑比较判断服务的匹配,计算复杂度大大降低,提高了服务发现效率。利用这个服务簇模型,本文提出了一种基于服务簇和 QoS 的快速组合方法。该方法首先根据用户的请求,利用服务簇之间的参数依赖关系进行广度优先搜索,得到具有并行层次调用结构的服务组合原型;之后,利用候选服务的参数依赖关系计算每个服务的最优组合 QoS 值,并以之为

收稿日期:2013-02-21;**修回日期**:2013-03-29。 **基金项目**:国家973 计划项目(2010CB328101);国家自然科学基金资助项目(61170078,60773034);山东省科技发展计划项目(2011GGX10114);山东省高等学校科技计划项目(J12LN11);潍坊市科技发展计划项目(2010RKX011)。
作者简介:邓式阳(1971-),男,山东高密人,讲师,博士研究生,主要研究方向:分布式系统、Web 服务发现、Petri 网; 杜玉越(1960-),男,山东聊城人,教授,博士生导师,博士,CCF 高级会员,主要研究方向:计算机支持协同工作、软件工程、形式化技术、Petri 网。

依据在服务簇内部进行服务过滤;然后以用户请求的输出参数为起点逆向求解,逐层动态确定 QoS 阈值进行服务过滤和冗余处理,构造所有可能的最优服务组合。动态确定 QoS 阈值进行过滤,可以保留更多符合最优条件的服务,获得多个最优服务组合;服务簇内部过滤的方法可以限制候选服务的数量,高效的冗余处理方法可以大大减少组合数量,从而解决了因组合过多造成的运行超时问题,在百万级海量服务库中保证执行速度。

1 基本概念

1.1 Web 服务、服务簇和服务组合

定义 1 Web 服务。服务可以描述为一个四元组 $s = (N, I, O, QoS)$。其中:N 表示服务的名称,I 是服务的输入参数集合,O 是服务的输出参数集合,QoS 代表服务的一类 QoS 属性。

定义 2 服务请求。服务请求可以描述为三元组 $Q = (I, O, QoS)$。其中:I 表示用户可以提供的输入参数集合,O 表示用户所要求的输出参数集合,QoS 表示用户的 QoS 要求。

定义 3 服务簇。服务簇可以描述为五元组 $C = (N, S, I, O, F)$。其中:N 为服务簇的名称;S 是服务簇中 Web 服务的集合;I 表示服务簇的输入参数集,是 S 中各服务的输入参数的基于语义的并集;O 表示服务簇的输出参数集,是 S 中各服务的输出参数的基于语义的并集;F 表示服务簇与 S 中服务的映射关系,每个服务的输入/输出参数分别对应服务簇的输入/输出参数有唯一的位置向量。详细的定义说明请参见文献[11]。

定义 4 服务组合。对于一个服务请求 Q,服务组合可以描述为一个有向无环图 $G = (V, E)$,节点集 V 代表组合中的各个服务,边集 E 表示服务间的参数依赖关系,服务的输入参数由其前驱服务的输出参数共同提供。服务组合的示意图见图 1,其中 start 和 end 是虚拟服务,start 以 $Q.I$ 为输出参数,但没有输入参数,end 以 $Q.O$ 为输入参数,没有输出参数。

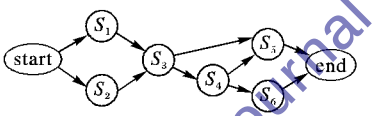

图 1 服务组合示意图

1.2 服务和组合服务的 QoS

定义 5 服务的 QoS。服务的 QoS 是指服务的非功能属性,如平均响应时间、吞吐量、价格花费及服务可信度等。

定义 6 组合服务的 QoS。组合服务的 QoS 是指组合中各个服务的 QoS 指标的综合评价。给定一个组合服务 G,根据服务 QoS 的性质和组合结构的不同,计算组合服务 QoS 的方法也不相同[4,9]。例如,对于响应时间,如果 G 中的服务可以并行执行,则 G 的整体响应时间为其中服务的最大响应时间;如果 G 中的服务为串行执行关系,则 G 的整体响应时间为各服务的响应时间之和。而价格花费只能是所有服务的价格之和,吞吐量只能是各服务吞吐量的最小值等。

2 服务搜索与优化组合算法

本文的 Web 服务组合方法包括服务搜索和优化组合两个阶段。

2.1 服务搜索

服务搜索阶段是根据用户提供的输入参数,在服务库中查找所有输入可被满足的服务,如果这些服务的输出不能满足用户的输出请求,则将这些服务的输出加入到输入参数集,再次在服务库中进行搜索,直到满足用户的输出请求或者找不到后继服务或达到强制终止条件。如果在海量的服务库中直接进行这样的搜索,其执行效率将非常低。本文基于语义相似度对服务库进行了聚类预处理,服务簇与服务的映射关系以向量的形式表示,在服务簇内通过逻辑比较判断服务的参数匹配,可以缩小搜索空间,减少语义比较,降低计算复杂度;建立了服务簇之间的参数依赖关系,在重复搜索时可以直接检查后继服务簇,提高搜索效率。

服务搜索过程:首先基于语义匹配找出必要输入参数可被用户输入参数满足的服务簇,并将用户输入参数转化为位置向量,如果簇内某个服务的输入参数位置向量可被用户的位置向量所包含,记入候选服务集,并进行输出判断和输入参数扩展,同时记录每个候选服务的 QoS 值、每个参数的源服务及候选服务簇等相关信息,供优化组合算法使用。如果检查完全部服务簇后用户的输出仍未满足,则检查当前层次候选服务簇的后继,直到满足用户输出或达到结束条件。搜索过程是一个广度优先算法,如算法 1 所示。

算法 1 基于服务簇的广度优先搜索算法(CBFS)。

输入:用户可提供的输入 I,用户要求得到的输出 O;

输出:候选服务及其输入参数集的哈希表 $mapSrvs$。

```
1)  listC = {All clusters};
2)  mergePara(mapParaIn, "start", I);
                                    //用户输入参数存入哈希表
3)  while (pathLimit < 15 and listC ≠ ∅) {
4)    for each lsc in listC{
5)      flag = false; setSrv = ∅; setOut = ∅;
                                    //匹配服务簇的必要输入参数,
                                    //得到用户输入参数位置向量
6)      if (matchParaInCluster (lsc, mapParaIn, listVectorIn)) {
7)        for each s in lsc {
8)          if (s.VI ⊆ listVectorIn) {  //匹配输入参数位置向量
9)            flag = true;                    //标记可用簇
10)           mapSrvs.put(s, s.I);     //记录服务及输入参数
11)           mapSrvOut.put(s, s.VO);
                                    //服务的输出参数位置向量
12)           mapQoS.put(s, s.QoS);          //服务的 QoS
13)           setSrv.add(s);                 //簇内候选服务
14)           setOut.add(s.O);    //簇内候选服务的输出并集
15)           mergePara (mapParaIn, s, s.O); //扩展输入参数
16)         }
17)       }
18)     }
19)     if (flag) {
20)       mapClsSrv.put(lsc, setSrv);
                                    //记录候选服务簇中的候选服务
21)       mapClsOut.put(lsc, setOut);
                                    //记录候选服务簇的输出
                                    //如果用户输出参数已满足,结束搜索
22)       if (findAllOut (O, mapParaIn)) return true;
23)       } else listC = listC - {lsc}       //去掉无用的服务簇
24)     }
25)     listC = getNextList (listC);          //取得后继服务簇
26)     pathLimit++;                          //记录搜索层次
27)   }
28) return false;
```

2.2 优化组合

采用逆向组合的方法,可以很容易地去除对后继服务无用的服务,但候选服务中仍可能存在大量功能相近的服务,必

须进行优化以获得最优组合。优化组合有两个方面:1)组合服务的 QoS 最优;2)组合中的服务个数最少。下面将就这两个方面分别进行讨论。为简化描述,后文的 QoS 仅指服务的响应时间。

2.2.1　基于 QoS 优化组合

文献[10]提出了最优组合 QoS 值的概念:服务的前驱服务组合的 QoS 最小值与服务本身的 QoS 值之和。该值可利用服务的参数依赖关系进行递归计算获得。该文在构建组合时利用服务的最优组合 QoS(以下简称 QoS)值进行动态过滤,对每个参数取其源服务中 QoS 值最小的一个服务,最终可获得唯一的最优组合。

事实上,根据定义 6,如果用各参数源服务的 QoS 值的最小值中的最大值为阈值,仅过滤掉大于阈值的服务,仍可保证任意组合的 QoS 值最优,这样就可以获得多个最优组合,为用户提供更多的选择。但是,如果作为阈值的某个参数的源服务 QoS 最小值很大,其他参数的源服务就可能保留很多,造成组合太多而运行超时的问题。对此可以采用如下方法解决:如果一个服务的输出包含另一个服务的输出,且 QoS 值较小或 QoS 值相等但二者的输出是真包含关系,则过滤掉另一个。在服务簇中,每个服务的参数对应一个位置向量,因此在服务簇内部用位置向量来判断服务参数的包含关系,不需进行语义计算,有利于提高执行效率。另外,对服务簇中候选服务的输出参数的并集,取各参数源服务 QoS 最小值中的最大值为阈值进行过滤,由于缩小了参数的范围,也可以过滤掉部分服务。这两种方法合并在一起称为服务簇内部过滤。

2.2.2　过滤冗余服务

过滤冗余服务的目的是使组合中的服务个数最少。较简单的方法是判断各组合的服务集是否存在包含关系,将较大的组合去掉,其缺点是计算量较大。本文根据逻辑表达式的化简公式提出了一种高效的冗余过滤方法。

$$(A \vee B) \wedge A = A \tag{1}$$

1)由于服务的同一参数的各源服务是逻辑或关系,而不同参数的源服务集之间是逻辑与关系,所以,根据式(1),对服务的前驱进行组合前,判断各输入参数的源服务集是否存在包含关系,将较大的源服务集去掉。例如:某服务 3 个输入参数 p_1、p_2 和 p_3 的源服务集分别为 $\{s_1\}$、$\{s_1,s_2\}$ 和 $\{s_3,s_4\}$,因为任意组合必含有 s_1,它可以为 p_2 提供输入,s_2 成为冗余服务,将 p_2 的源服务集去掉,得到的组合情况不变,都是 $\{s_1,s_3\}$ 和 $\{s_1,s_4\}$。这样就可以去除大量冗余服务,大大减少组合的计算量。

2)对服务的前驱进行组合时,如果一个临时组合与某个参数的源服务集有交集,则二者不必组合。假设此时的临时组合为 $(A \wedge B \wedge C)$,下一个参数的源服务集为 $(A \vee D)$,根据式(1)可得

$$(A \vee D) \wedge (A \wedge B \wedge C) = ((A \vee D) \wedge A) \wedge B \wedge C = A \wedge B \wedge C \tag{2}$$

组合后的结果与原组合一致,因此不必组合这个参数的源服务。

通过这两种方法,可以大大提高冗余服务过滤的效率,快速得到所有无冗余组合。

2.2.3　组合构造算法

逆向构造最优服务组合的过程如下:

1)计算所有服务的最优组合 QoS 值;

2)依据服务的最优组合 QoS 值在各服务簇内进行服务过滤;

3)计算每个用户输出参数的源服务的 QoS 最小值,以其中的最大值为阈值,过滤用户输出参数的源服务,然后对剩余的源服务进行无冗余分组,以每个分组作为一个服务组合图的首节点;

4)对各个服务组合图的首节点的输入参数的源服务进行同样的过滤和分组,以每个分组作为新的首节点,将原组合图扩展为多个组合图;

5)重复步骤 4),直到每个图的首节点的输入参数的源服务均为虚拟服务 start,则每个图即为一个最优组合服务。

组合构造的伪算法如下:

算法 2　基于 QoS 的最优服务组合构造算法。

输入:用户请求的输出 O;

输出:最优服务组合列表 *listDaGOptimal*。

```
1)  mapBestQoS = getBestQoS(mapParaIn, mapQoS, mapSrvs);
                                              //最优组合 QoS
2)  filteSrcClsSrv(mapBestQoS, mapClsSrv, mapClsOut,
      mapSrvOut);                             //簇内过滤
3)  listSrcSrv = getSrcSrv(O, mapParaIn)
                                  //用户各输出参数的源服务集列表
4)  filteSrcSrv(listSrcSrv, mapBestQoS);      //源服务 QoS 过滤
5)  listHeads = getHeads(listSrcSrv);
                                  //无冗余分组获得头节点列表
6)  for(int i = 0;i < listHeads.size();i++){
7)    listDaG = new ArrayList<Set<String>>();
                                              //新建组合图
8)    listDaG.add(listHeads.get(i));
9)    listDaGs.add(listDaG);                  //组合图列表
10) }
11) buildCompositionDags(listHeads, listDaGs, listDaGOptimal);
                          //算法 2-1(服务组合递归构造算法)
```

算法 2-1　服务组合递归构造算法。

输入:组合图的头节点列表 *listHeads*,临时组合图列表 *listDaGs*;

输出:最优服务组合列表 *listDaGOptimal*。

```
1)  int count = listHeads.size();
2)  if(count == 0) return;
3)  for(int i = 0; i < count; i++){
4)    if(listHeads.get(0).size = 1 &&
        listHeads.get(0).contains("start")){
5)      listDaGOptimal.add(listDaGs.get(0));
6)    }else{
7)      setParaIN = getSrcPara(listHeads.get(0), mapSrvs);
                                  //头节点的输入参数集
8)      listSrcSrv = getSrcSrv(setParaIN, mapParaIn);
                                  //源服务集列表
9)      filteSrcSrv(listSrcSrv);                //服务过滤;
10)     listHeadNew = getHeads(listSrcSrv);
                                  //源服务分组,新的头节点
11)     for(int j = 0; j < listHeadNew.size(); j++){
                                  //扩展组合路径
12)       listDaG = new ArrayList<Set<String>>();
13)       listDaG.add(listHeadNew.get(j));
14)       listDaG.addAll(listDaGs.get(0));
15)       listDaGs.add(listDaG);
16)       listHeads.add(listHeadNew.get(j));
17)     }
18)   }
19)   listHeads.remove(0);                  // 删除当前头节点
20)   listDaGs.remove(0);                   // 删除当前路径
21) }
22) buildCompositionDags(listHeads, listDaGs, listDaGOptimal);
                                  //递归调用
```

2.3 复杂度分析

在候选服务搜索阶段,假设服务库中有 n 个服务,每个服务有 k 个参数,需要进行 m 次搜索查找候选服务,那么,如果直接在库中搜索,由于每次搜索出的服务个数远小于 n,每个轮次的服务个数都可视为 n,则 m 次查找需要进行 $n*k*m$ 次参数语义比较。若服务库可以分为 d 个服务簇,第一轮次需要进行 d 次比较,找到 p 个服务簇,由于服务簇的依赖关系已经建立,后继服务簇可以直接得到,后续每次找到 p 个服务簇,在服务簇内,每个服务只需进行一次逻辑比较,可视为一个参数的语义比较,一个服务簇只需进行 n/d 次比较,则整个搜索过程大约需要进行 $(d+m*p)*k+m*p*n/d$ 次语义比较。一般情况下,d 和 k 都远小于 n,p 远小于 d,假设 $n=10\,000$, $d=100$, $m=10$, $k=10$, $p=10$,基于服务簇的语义比较次数仅为直接搜索的 1.2%,比较次数大大减少。

服务组合阶段的主要工作量在于服务的分组。假设用户请求由 m 个参数,每个参数有 n 个源服务,组合层数数为 1,则在最坏情况下,其时间复杂度为 $O(n^m)$。采用各参数源服务最小 QoS 值中的最大值为阈值进行过滤,最坏情况是除阈值所在的参数以外,其他参数的源服务的 QoS 值都小于阈值,则时间复杂度为 $O(n^{m-1})$,降低了 1 个指数级别。由于同服务簇内候选服务的相似性很大,在服务簇内进行基于参数包含关系和 QoS 值的过滤,可以过滤掉很多非最优服务。采用本文的冗余过滤算法,可以大大降低服务分组数量,降低计算复杂度。例如,有 10 个参数,每个参数的源服务集有 2 个服务,按照每个参数的源服务集取 1 个服务的分组方法,其组合数可达 2^{10};如果参数的源服务集有两种情况 $\{s_1,s_2\}$ 和 $\{s_3,s_4\}$,采用本文的冗余过滤算法,组合数仅为 2^2,计算复杂度显著降低。

3 算法实验

由于缺少实际的海量 Web 服务库,实验使用模拟数据进行测试。根据实验的不同,本体概念的数量为1 500 ~ 30 000,服务的数量为 5 万 ~ 100 万。模拟数据的生成方法为:首先建立一棵领域本体树,然后建立服务簇及服务簇间的参数依赖系,再根据服务簇参数随机生成一组相似的服务,并建立服务簇与服务的映射关系。簇内服务个数随机为10 ~ 500,服务的参数不超过 18 个,公共参数 6 ~ 12 个,服务的 QoS 取值为 5 ~ 2 500。本文算法采用 Java 语言编程,实验的硬件环境是 ThinkPad R61(CPU 双核2.10 GHz, 内存2.97 GB)。

3.1 服务搜索效率

由于直接在服务库内进行基于语义的服务搜索效率很低,实验对比基于服务簇的搜索方法(CBFS)和采用关键词与语义相结合的方法(KBFS)。KBFS 方法为逐层以输入参数作为关键词查找相关服务,然后进行服务参数的语义匹配来确定候选服务。进行 5 次实验,服务簇的个数为 200 ~ 1 000,服务的个数为 5 万 ~ 25 万。每次随机产生 100 个用户请求,30 个 3 ~ 6 个服务的并行组合,70 个 3 ~ 6 层的层级组合,请求的输出参数不超过 30 个。实验结果如图 2 所示。

由图 2 可知,随着服务簇和服务的增多,CBFS 的执行时间增长平缓,最高不超过于 0.5 s,KBFS 的执行时间为 CBFS 的几十倍,且增长迅速。实验结果证明了基于服务簇的搜索方法的高效性。

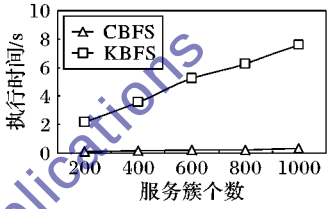

图 2 CBFS 和 KBFS 服务搜索效率比较

3.2 组合效率

组合效率实验采用基于服务簇的搜索方法,对比采用服务簇内部过滤与否的执行时间、过滤效果和获得的最优组合数。进行 5 次实验,每次实验执行 3 遍以保证数据的稳定性,服务库和请求的产生方法与第一个实验相同。实验结果如表 1 所示,其中 QCB 是仅采用动态 QoS 过滤的组合算法,CQCB 是采用服务簇内部过滤和动态 QoS 过滤的组合算法,执行超时时间为 3 min。由表 1 可以看出,两种算法的执行时间都是毫秒级别的,但 CQCB 算法的执行时间比 QCB 算法少,这是因为进行服务簇内过滤后,候选服务数量大幅减少,在进行动态过滤和参数的源服务分组时可以提高执行速度。虽然 CQCB 算法获得的最优组合数要少一些,但没有一次发生请求超时现象。

表 1 组合效率对比

服务簇数	执行时间/ms		候选服务数		最优组合数		超时请求比例/%	
	QCB	CQCB	QCB	CQCB	QCB	CQCB	QCB	CQCB
200	294	221	716	65	121	77	1.67	0
400	561	303	856	76	339	86	2.67	0
600	484	383	980	89	191	13	0.33	0
800	533	489	1 013	96	175	26	1.00	0
1 000	658	528	846	79	272	96	1.67	0

另外还对服务的冗余处理效果做了简单实验,如果不采用本文的冗余处理方法,绝大多数请求都会出现运行超时或内存溢出,从而验证了本文冗余处理方法的高效性。

3.3 性能测试

实验测试极端情况下本文算法的执行效率,服务总数分别为 50 万和 100 万,请求中组合的层次分别为 3 ~ 6、5 ~ 10 和8 ~ 15 层,测试结果如图 3 所示。由图 3 可以看出,在同一个服务库中,组合层次越多,执行时间越长;相同组合层次时,服务总数越大,执行时间越长。从总体上看,算法在组合层次数为 3 ~ 6 层时执行效果最好,在库中的服务数达到 100 万、组合层次 8 ~ 15 层的情况下,平均执行时间仍低于 4 s,说明本文算法能够较好地适应海量服务库和较多层次组合的情况。

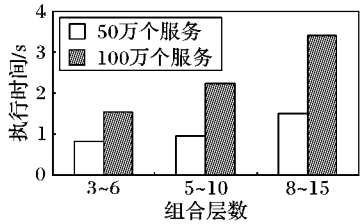

图 3 本文算法在海量服务库中的执行效率

(下转第 2183 页)

can Astronomical Society, 2008, 40: 500.
[2] VERGAUWEN M, POLLEFEYS M, van GOOL L J. A stereo-vision system for support of planetary surface exploration [J]. Machine Vision and Applications, 2003, 14(1): 5 – 14.
[3] CHANG G, LAW E, MALHOTRA S. Demonstration of LMMP workflow system using cloud computing architecture [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 70 – 71.
[4] BUI B, CHANG G, KIM R, *et al.* Demonstration of LMMP (Lunar Mapping and Modeling) using Amazon's elastic compute cloud [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 69 – 69.
[5] TRAN J J, CINQUINI L, MATTMANN C A, *et al.* Evaluating cloud computing in the NASA DESDynI ground data system [C]// SECLOUD'11: Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing. New York: ACM, 2011: 36 – 42.
[6] GREEN P J, SIBSON R. Computing dirichlet tessellafons in the plane [J]. The Computer Journal, 1978, 21(2): 168 – 173.
[7] LEWIS B A, ROBIRTSON J S. Triangulation of planar regions with applications [J]. The Computer Journal, 1978, 21(4): 324 – 332.
[8] AGGARWAL A, CHAZELLE B, GUIBAS L, *et al.* Parallel computational geometry[J]. Algorithmica, 1988, 3(1-4): 293 – 327.
[9] CHEN M-B, CHUANG T-R, WU J-J. Efficient parallel implementations of 2D Delaunay triangulation with high performance fortran [C]// PPSC'01: Proceedings of 10th SIAM Conference on Parallel Processing for Scientific Computing. Philadelphia: SIAM, 2001.
[10] OKUSANYA T, PERAIRE J. 3D parallel unstructured mesh generation [C]// Joint ASME/ASCE/SES Summer Meeting, Special Symposium on Trends in Unstructured Mesh Generation. [S. l.]: ASME, 1997: 109 – 115.
[11] CHRISOCHOIDES N, NAVE D. Simultaneous mesh generation and partitioning for Delaunay meshes [C]// Proceedings of the Eighth International Meshing Roundtable. South Lake Tahoe: [s. n.], 1999: 55 – 66.
[12] CHRISOCHOIDES N, NAVE D. Parallel Delaunay mesh generation kernel[J]. International Journal for Numerical Methods in Engineering, 2003, 58(2): 161 – 176.
[13] BLANDFORD D K, BLELLOCH G E, KADOW C. Engineering a compact parallel delaunay algorithm in 3D [C]// SCG '06: Proceedings of the Twenty-second Annual Symposium on Computational Geometry. New York: ASM, 2006: 292 – 300
[14] 罗斌. 基于约束数据域三角剖分的高精度 DEM 快速生成技术及实现[D]. 西安: 长安大学. 2006
[15] TAMMINEN M. Comment on quad- and octtrees[J]. Communications of the ACM, 1984, 27(3): 248 – 249.
[16] NGUYEN D, PINGALI K. Synthesizing concurrent schedulers for irregular algorithms [C]// ASPLOS'11: Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 333 – 344.
[17] KULKARNI M, PINGALI K, WALTER B, *et al.* Optimistic parallelism requires abstractions [C]// PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2007: 211 – 222.

(上接第 2170 页)

4　结语

本文针对海量的语义 Web 服务的组合问题,提出了一种基于服务簇和 QoS 的组合方法。基于服务簇进行服务搜索,搜索空间小,语义比较的次数少,搜索速度快。基于服务的最优组合 QoS 值动态确定阈值进行服务过滤,可以获得多个最优组合。采用服务簇内部过滤和高效的冗余处理方法,可大量减少组合数量,保证组合中冗余服务最少,并且可在百万级海量服务库中保证执行速度。本文仅针对服务响应时间的 QoS 属性给出了最优组合算法,如何权衡多 QoS 属性获得最优组合,仍然需要进一步研究。

参考文献:

[1] 徐小良, 陈金奎, 吴优. 基于聚类优化的 Web 服务发现方法[J]. 计算机工程, 2011, 37(9): 68 – 70.
[2] NAYAK R, LEE B. Web service discovery with additional semantics and clustering [C]// ICWI'07: Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence. Washington, DC: IEEE Computer Society, 2007: 555 – 558.
[3] 孙萍, 蒋昌俊. 利用服务聚类优化面向过程模型的语义 Web 服务发现[J]. 计算机学报, 2008, 31(8): 1340 – 1353.
[4] 刘书雷, 刘云翔, 张帆, 等. 一种服务聚合中 QoS 全局最优服务动态选择算法[J]. 软件学报, 2007, 18(3): 646 – 656.
[5] KONA S, BANSAL A, GUPTA G. Automatic composition of semantic Web services [C]// ICWS'07: Proceedings of the 5th International Conference on Web Services. Washington, DC: IEEE Computer Society, 2007: 150 – 158.
[6] KWON J, LEE D. Non-redundant Web services composition based on a two-phase algorithm [J]. Data & Knowledge Engineering, 2012, 71(1): 69 – 91.
[7] NASERI M, TOWHIDI A. QoS-aware automatic composition of Web services using AI planners [C]// ICIW'07: Proceedings of Internet and Web Applications and Service. Washington, DC: IEEE Computer Society, 2007: 29 – 35
[8] SIRINA E, PARSIAB B, WU D, *et al.* HTN planning for Web service composition using SHOP2[J]. Journal of Web Semantics, 2004, 1(4): 377 – 396
[9] HUANG Z Q, WEI J, HU S L, *et al.* Effective pruning algorithm for QoS-aware service composition [C]// CEC'09: Proceedings of IEEE Conference on Commerce and Enterprise Computing. Washington, DC: IEEE Computer Society, 2009: 519 – 522
[10] WEI J, CHARLES Z, HUANG Z Q, *et al.* QSynth: A tool for QoS-aware automatic service composition [C]// ICWS'10: Proceedings of International Conference of Web Services. Washington, DC: IEEE Computer Society, 2010: 42 – 49
[11] 邓式阳, 杜玉越. 一种基于逻辑 Petri 网的 Web 服务簇模型[J]. 计算机应用, 2012, 32(8): 2328 – 2332.
[12] JIANG X, WU B, DU Y Y. Architecture of dynamic service composition using logic Petri nets [J]. Journal of Software Engineering & Applications, 2011, 4(10): 585 – 589.
[13] DU Y Y, JIANG C J, ZHOU M C. Modeling and analysis of real-time cooperative systems using Petri nets [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2007, 37(5): 643 – 654.
[14] MURATA T. Petri nets: properties, analysis and applications [J]. Proceedings of the IEEE, 1989, 77(4): 541 – 580.
[15] DU Y Y, QI L, ZHOU M C. A vector matching method for analyzing logic Petri nets [J]. Enterprise Information Systems, 2011, 5(4): 449 – 468.