基于PDG图的分布式动态可执行服务组合方法

赵文评,葛 玮
(西北大学 信息学院,陕西 西安 710127)

摘 要:组合服务是将一系列Web服务协调地组合在一起,从而完成预期目标。组合服务按照其引擎分布可分为:集中式服务组合和分布式服务组合。分布式组合服务可以有效地解决集中式组合服务在系统可伸缩性、消息传输效率、自治性和有效负载均衡等方面的问题,将可执行全局流程等价分解成可执行本地流程,以及将服务组合和负载均衡结合集群概念,研究服务组合的QOS。这些将成为新的挑战,文中通过PDG图进行等价分解,并结合QOS,通过实验测试引擎负载,提出将负载均衡融入服务组合QOS研究思想。

关键词:组合服务;BPEL4WS;流程分解;QOS;负载均衡
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2007)07-0040-05

Approach for Decentralizing Dynamic Web Services Composition Based on PDG

ZHAO Wen-ping, GE Wei
(Department of Information Science, Northwest University, Xi'an 710127, China)

Abstract: Web service composition consists of orchestrated Web services. Web service composition can be divided into two classes, namely centralized architecture and decentralized architecture, by the location of the Web services composition engine. Decentralized Web services composition can solve the problem of scalability, message exchange efficiency, autonomy and load. But it's big challenge to partition a composite Web service written as a single BPEL program into an equivalent set of decentralized processes, to study the composite Web service QOS by the host set conception, Web services composition method and load balance of Web services engine.

Key words: Web services composition; BPEL4WS; process partition; QOS; load balance

0 引 言

在基于Web services的分布式Web应用框架中,为了完成特定的功能,必须先把企业组件包装成Web service,然后把这些Web服务按照特定顺序进行编排,从而跨Internet来完成指定的功能模块[1]。而描述这些Web服务的编排顺序方法通常采用基于工作流的描述方式来描述,即业务流程执行描述语言BPEL(Business Process Execution Language)[2]。

在BPEL描述的服务组合中,参与服务编排的Web服务的目的是为了一个集中式的总体目标,称这种服务编排方式为集中式服务组合。如果将集中式BPEL程序拆分成几个独立的BPEL程序,虽然对于每个独立的BPEL程序并没于特定编排目标,但是如果将这些独立的BPEL程序部署到不同BPEL引擎上,通过这些独立BPEL程序协同工作,则可以来完成集中式BPEL程序功能。因此,称这种编排方式为分布式或非集中式服务组合。集中与分布式服务组合相比,集中式BPEL程序部署到一台组合服务引擎上,在引擎负载为中载或重载时,使得相应时间变长和服务正确执行变坏。而分布式正好可以克服集中式服务服务组合的缺陷。

在分布式组合服务的分解算法的研究上,IBM的Mangala G. Nanda[3]等人提出按照不同Web服务,将集中式BPEL程序分解成分布式可执行服务组合。而刘必欣[4]等人通过将集中式BPEL程序进行建模,然后根据参与服务编排的Web服务的角色不同进行分解,从而形成分布式服务组合模型,而不是直接部署到组合服务引擎上的BPEL程序。文中通过将参与编排的Web服务按照角色不同,参照Mangala提出的基于PDG(Program Dependence Graph)分解的思想,然后融入动态服务组合概念和通过最短路径选择服务副本,提出虚拟服务组合概念和组合服务的QOS,即:PDG

收稿日期:2006-09-15

作者简介:赵文评(1979-),男,陕西礼泉人,硕士研究生,研究方向为中间件、工作流、Web服务;葛 玮,副教授,硕士生导师,研究方向为中间件、工作流等。

结构下,基于角色的 BPEL 程序分解算法。从而解决 Mangala 等人提出分解算法必须为每个参与服务编排的 Web 服务提供一个组合服务引擎条件限制;以及解决刘必欣等人提出按照角色分解的不可执行性,同时对分解后的服务组合的 QOS 进行实验和分析。

1　PDG 图相关概念和属性分析

分解先决条件:将集中式组合服务分解成分布式组合服务,就要对集中式组合服务参照表 1[3],将 BPEL 程序翻译成由 notation 表示的程序,并基于此建立程序依赖图 PDG(Program Dependence Graph)。

概念 1　程序依赖图 PDG (Program Dependence Graph)。程序依赖图是控制流图 CFG (Control - Flow Graph)中的数据依赖和控制依赖用同一系列结点关系表示,即:将控制依赖用树表示,数据依赖用图表示。数据依赖图中的结点都对应控制树中某个结点。结点的操作是 CFG 中的预定义的表达式或操作。

概念 2　数据依赖(Data Dependence)。数据依赖[4]是结点 receivenode 从结点 sendnode 接受业务数据,只有业务数据正确到达 receivenode,receivenode 才可能开始运行。用 sendnode≺receivenode 表示数据依赖关系,即 receivenode 依赖 sendnode 业务数据。函数 dd(sendnode,receivenode)表示具有数据依赖结点间传输业务数据。

概念 3　控制依赖(control dependence)。控制依赖[4]是结点可以从多个结点中根据比较条件结果来决定某个或某几个结点运行。记为 cd(node)。

概念 4　前驱分支(pre branch)。结点 N_j 不是根结点,对所有 N_i,如果存在 $N_i \prec \cdots \prec N_j$,且 N_i 的父结点对应操作是比较操作,$N_i \in \text{preset}(N_j)$,则 N_i 是 N_j 前趋分支。函数 preset(N_j) 表示 N_j 前驱分支集合。

概念 5　结点(node)。PDG 中结点定义 node::=(op, parent, prenodes, postnodes, prebranch, type, role),其中:

op(操作)将一些相关操作组成一个线程,操作可以为表 1 中活动和比较操作(if 操作),但是结点中操作只容许有一个比较操作同时不可含有 invoke 操作。

parent 是在条件分支树中父结点,表示父结点通过比较操作来判断是否执 node 对应操作。对于控制依赖函数 cd(node)则返回 node.op 比较操作的结果。

prenodes 是结点 node 前驱结点集合,即:在数据依赖图中,node 从 prenodes 接受数据。如果 node.prenodes 为空,则 node.prenodes 为 parent.prenodes。

postnodes 则是结点 node 后驱结点集合,postnodes 中结点是从 node 结点接受数据。

prebranch 是 node 的前驱分支集合。

type 是结点的类型,结点 node 可分为固定结点(fixed node)、可合并结点(portable node)[3]和 if 型(node.op 为比较操作)可合并结点。固定结点操作一般是同步调用 Web 服务(invoke),或者响应组合服务对用户出口和入口,即 receive(req, var)或 reply(res. var);而可合并结点则是其它操作可以和固定结点合并组成新的线程;if 型可合并结点是特殊可合并结点的一种,它的操作中有且有一个比较操作。

概念 6　条件接收(condition receive)。条件接收是对于两个结点 N_j,N_i,如果 N_j 是在选择语句(如 switch_1)的某个条件语句(如 case_1)中的某个操作(如 invoke,assign,receive 操作),而 N_i 则是在该选择结构 switch_1 外的某个操作。并且,在 switch_1 该 case_1 条件成立时,N_i 才从 N_j 接收数据,否则,N_i 不用从 N_j 接收数

表 1　BPEL 结构、描述和含义(notation)对应关系

BPEL construct	Description	Notation
Control Flow Constructs		
`sequence`	sequential flow	`sequence ... end-sequence`
`switch`	conditional flow	`switch ... end-switch`
`while`	iterative flow	`while ... end-while`
`pick`	non-deterministic conditional flow	`pick ... end-pick`
`flow`	concurrent flow similar to cobegin-coend	`flow ... end-flow`
`link`	wait-notify type of synchronization	`source(linkId), target(linkId)`
Data Structures		
`variable`	variables include a set of parts analogous to fields	`variableName { part1, part2, ... partn }`
Activities		
`invoke`	synchronous (blocking) invocation on a partner *P*, sending data from an input variable *in* and receiving the response in the output variable *out*	`invoke(P, in, out)`
`send`[1]	asynchronous (oneway, nonblocking) invocation on a partner *P*, sending data using an input variable *in* (no response variable)	`send(P, in)`
`receive`	blocking receive of data from a partner *P* into a variable *var*	`receive(P, var)`
`reply`	send response to a partner *P* from a variable *var*	`reply(P, var)`
`assign`	assignment. Multiple assignments can be specified in a single `assign` statement, which executes atomically	`var1.p1.g1 = var2.p1.g3`
`compute`	arithmetic or logical operation	

据。

从以上概念可以得到 PDG 以下属性：

性质 1　并行属性。$N_i, N_j \in$ PDG，如果 N_i.parent $= N_j$.parent，$cd(N_i) = cd(N_j)$，不存在 $N_i \prec \cdots \prec N_j$，则 N_i 和 N_j 关系为并行。

证明：如果这两个结点不是并行关系，则这两个结点数据依赖(矛盾)，或分支树中属于两个不同分支或相同分支中但存在先后的顺序关系，又因为它们控制依赖函数相同，且它们之间没有数据依赖，故与假设相矛盾。

性质 2　分支路径。$N_i \in$ PDG 如果存在一对结点 N_k 和 N_j，$N_k \in N_i$.prenodes，$N_j \in N_i$.prebranch，并且 N_k 在 N_j 的子树中，那么在 PDG 中控制依赖树中，有且仅有一条路径从 N_j 到 N_k。

证明：根据树性质可以很显然得到 N_j 到其子孙 N_k 有且只有一条路径存在。

2　分解算法

算法 partition：

输入：PDG 图，固定结点与角色的影射。

输出：为总共的角色数 R_n 个本地 BPEL 流程。

步骤：

(1) 找出 PDG 中所有结点 N_i，N_i 的所有孩子结点为叶子结点，将其孩子结点 N_j 中的含有相同的控制依赖，即：$cd(N_j)$ 返回值相同。

(2) 随机找出与结点 N_i 相邻结点 N_j，且 N_i, N_j 不都是固定结点，如果 $\forall N_k \in N_j$.prenode，$N_k \in N_i$.postnodes 和 $N_i \in N_j$.prenode，则存在依赖环，通过复制 N_i 结点为 N_i'，将 N_i' 与 N_k 合并，N_i 和 N_j 合并[3]。

(3) 如果可合并结点与固定结点合并，将可合并结点的角色赋值为固定结点对应的角色。

(4) 对于剩下没有合并的结点，则将其和其父结点合并。

(5) 逐个考察每个结点的数据流接收问题：

(5.1) 确定接收。

对于 N_i, N_j, N_i.role $\neq N_j$.role，$N_j \in N_i$.prenode 且 $N_i \notin N_j$.prebranch，则在 N_j.op 代码前部加入 receive(N_j.role，dd(N_j, N_i))。

(5.2) 条件接收。

$\forall N_j \in N_i$.prebranch，$N_k = N_j$.parent，且 N_k.role $= N_i$.role，则在 N_k 的 N_j.CP 条件下加入 receive(N_j.role，dd(N_j, N_i))，否则，复制 N_k 结点，N_k' 表示 N_k 复制结点且 N_k'.role $= N_i$.role 在 N_k' 的 N_j.CP 条件下加入 receive(N_j.role，dd(N_j, N_i))。

(6) 逐个考虑相同角色结点间的并行或顺序关系，即：按照 PDG 图从左到右，从上向下，逐个考虑角色相同结点，如果两结点之间满足性质 1 则这两个结点为并行，否则为顺序。

(7) 参照表 1，将所有角色对应的结点描述成 R_n 个本地 BPEL 程序。

算法时间复杂度分析：

步骤(1) ～ (4) 将在控制依赖树中，按照规则将叶子结点间相邻兄弟结点合并，没有孩子的叶子结点可以和其父结点合并，而原来非叶子结点变成叶子结点，然后重复合并，直至没有可合并的叶子结点。因此等于考虑所有结点(n 表示结点个数)。步骤(2) 结点的前驱和后驱，采用复制消除依赖环，结点的前驱和后驱个数不可能很多，则步骤(1) ～ (4) 时间复杂度为 $O(n)$。步骤(5) 也是考虑结点的前驱分支复杂度为 $O(1)$，步骤(6) 的结点操作关系的并行和顺序考察，只要将结点前驱的结点集合，将集合中结点的前驱加入集合，直到集合中结点数量不再增加，这可以在 $O(n)$ 时间复杂度下完成。因此整个算法时间复杂度为 $O(n)$。

例子描述：

本例子是贷款利率评估的服务组合。流程描述是客户输入客户姓名(firstName，lastName)，贷款额度(amount)。如果贷款额度小于 1000，则风险 risk = 0；否则调用风险评估 Web 服务，该服务返回风险 risk 等级。将贷款额度 amount 和风险等级 risk 进行 SOAP 包装，通过 HTTP 分别发送到两个利率评估 Web 服务，每个利率评估 Web 服务返回一个利率，如 rate1，rate2。然后比较 rate1 和 rate2，取最小的利率返回给客户，流程结束。该例子的 PDG 见图 1[3]。

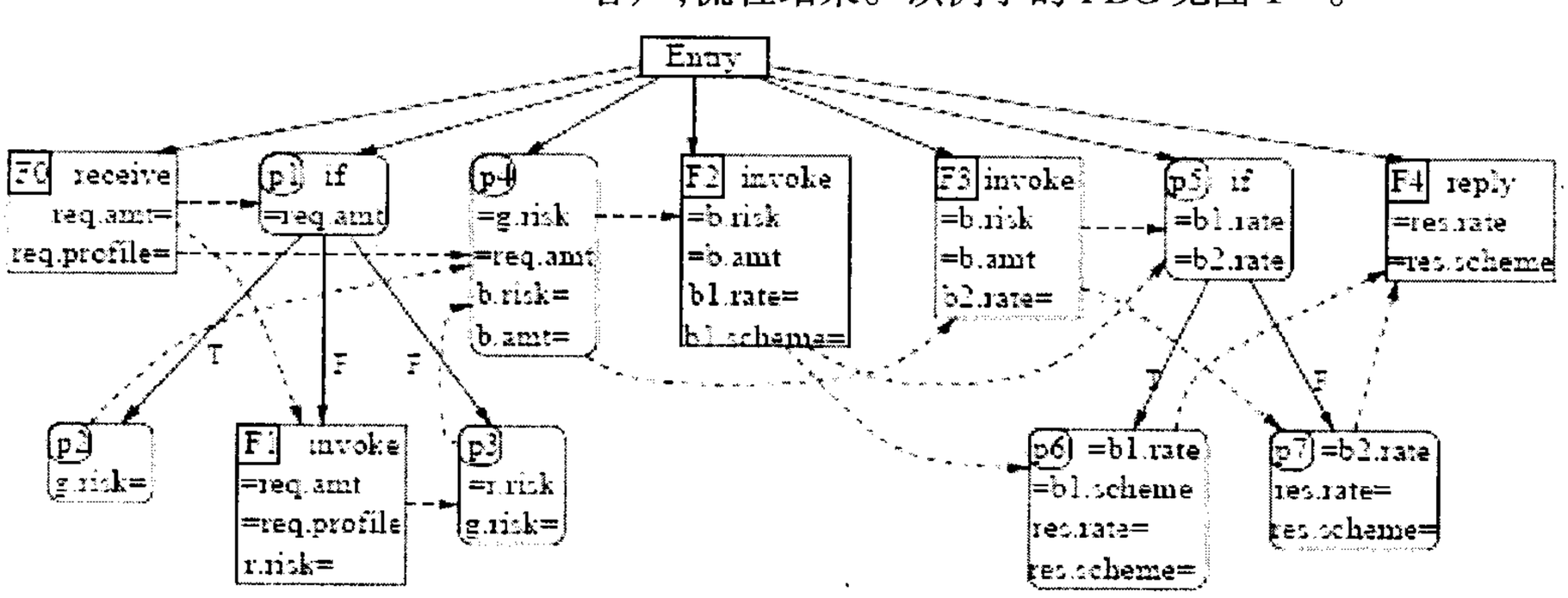


图 1　贷款利率评估的服务组合 PDG 图

证明：要证明应用该分解算法分解的 R_n 个本地 BPEL 程序在功能上与分解前集中式的 BPEL 程序功

能等价,首先,应证明 R_n 个本地 BPEL 程序对应的 PDG 与集中式的 BPEL 程序在结构上是等价的。然后,将 R_n 个本地 BPEL 程序分别部署在不同的 BPEL 引擎上,然后通过实验测试该分布式部署的服务组合完成的功能与集中式 BPEL 程序部署到一个 BPEL 引擎上完成功能相同。

结构等价证明:要证明生成的本地流程与全局流程结构等价,则应保证分解前和分解后数据依赖和控制依赖是等价的。在分解前后没有改变 PDG 的控制依赖。而对于数据依赖是在 PDG 图增加了结点。因此只要证明在条件接受的数据依赖在分解前后是等价的。在分解算法中将 $N_k(N_k = N_j.\text{parent})$ 复制为 N_k',然后在 N_k' 中 case_1 条件下加入从 N_j 接收数据 receive 操作,而这正表示条件接受的定义。由于 N_k 和 N_k' 对应前驱和后驱一样,且控制依赖又另一样,因此在加入 N_k' 不会影响 PDG 结构的正确性。同时通过分解步骤(2) 经过复制来消除在合并过程中出现依赖环路,从而保证分解前后的 PDG 图等价。

实验证明 R_n 个本地 BPEL 程序通过协同交互完成的服务组合在功能上与集中式 BPEL 程序完成的服务编排的功能等价。

分解 1:将接收客户输入和响应客户的 F0,F4 角色为 Role_0。风险评估 Web 服务角色为 Role_1,两个利率评估 Web 服务角色为 Role_2。

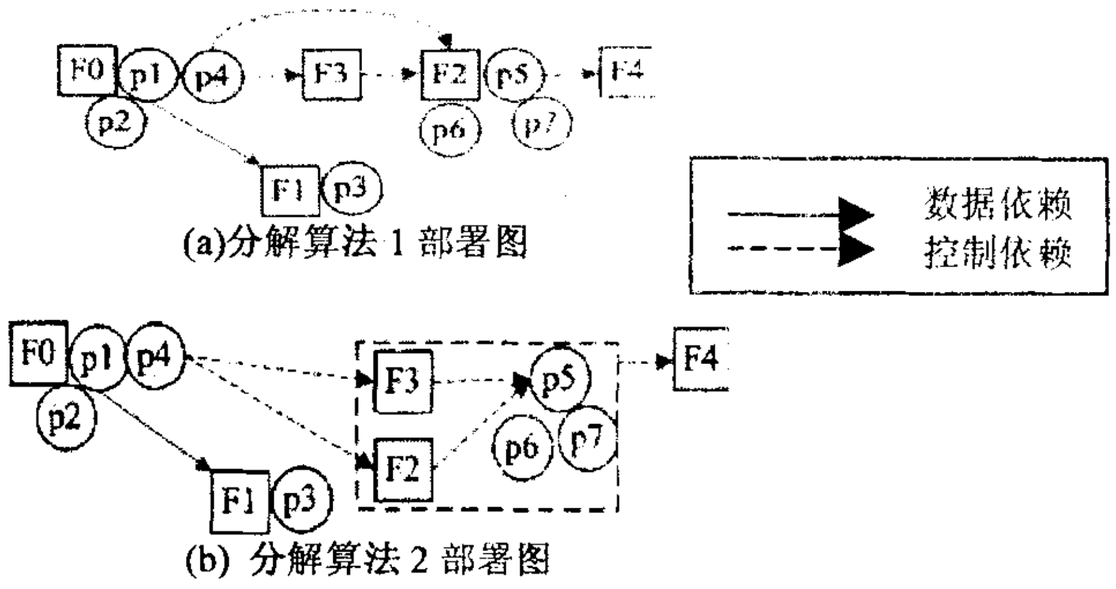
分解 2:将接收客户输入和响应客户的 F0,F4 角色为 Role_0。风险评估 Web 服务角色为 Role_1,一个利率评估 Web 服务角色为 Role,另一个利率评估 Web 服务角色为 Role_2。

按照分解算法分解后如图 2(a)和图 2(b)所示。通过按照图 2 分别部署分解 1,2 的分布式服务组合,使用相同的测试用例,得到测试结果与集中式服务组合的测试结果相同。则功能等价性得以证明。

3　分布式组合服务研究及实验结果

鉴于 BPEL 是流程的描述和执行语言,因此 BEPL 程序中同步或异步调用 Web 服务在设计阶段不予以指定,同时在按照文中提出的分解算法进行分解时,仍然不指定调用的具体 Web 服务,在运行中动态绑定。将没有绑定的服务组合称为虚拟流程。

以往在比较分布式组合服务与集中式组合服务性能时,着重对响应时间进行分析,而这在组合服务引擎轻载时是没有意义的。而 Mangala 等人在对组合服务引擎的负载分析时,着重分析在单位时间随机发出请求,依据在下一个单位时间收到响应计算引擎的有效负载,而无法测量在较短时间内(100ms)到达引擎 SOAP 请求的数量和其反映时间和正确响应率。也就是说,组合服务引擎最大范围内能处理或正确处理同时到达引擎的 SOAP 数量。而这才能正确反映在中载或重载时引擎的有效负载或引擎服务质量。



(a)分解算法 1 部署图

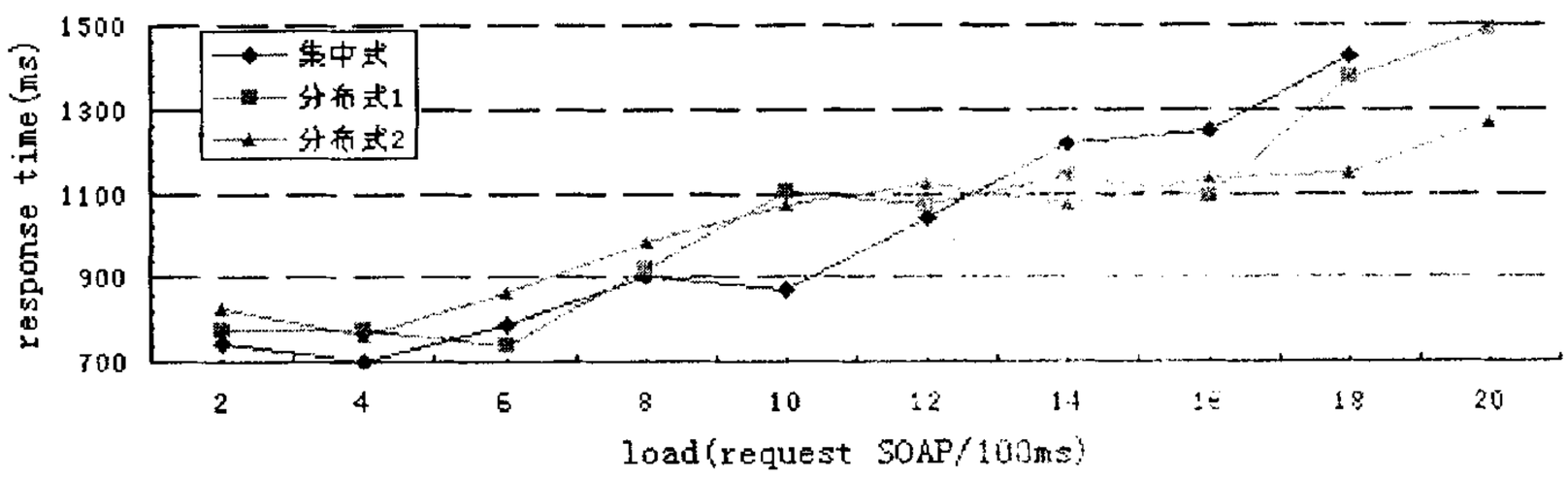
(b) 分解算法 2 部署图

图 2　分解算法生成部署图

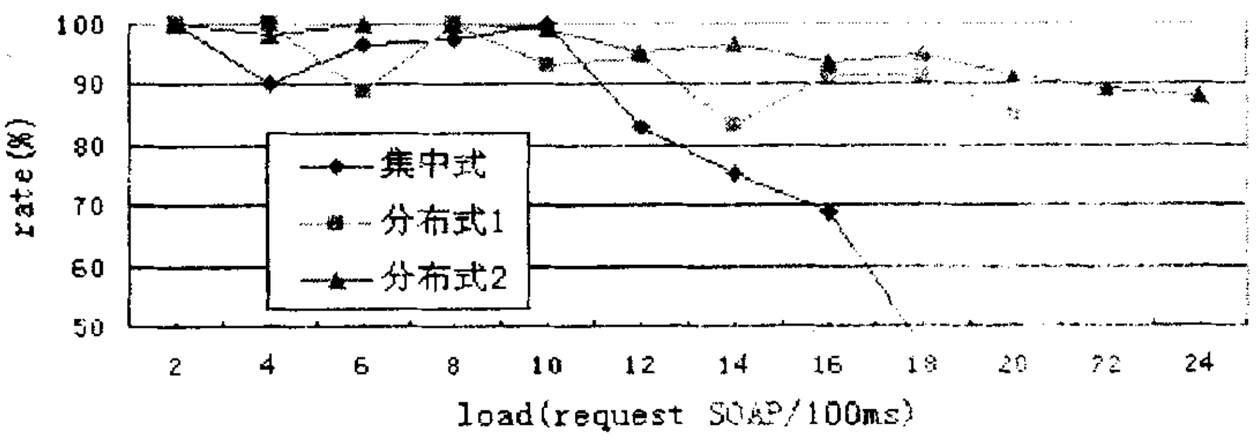
本次实验通过最短路径进行 Web 服务副本的选择。具体副本设定如下:风险评估 Web 服务没有副本,利率评估 Web 服务(F2 结点)1 个副本,另外一个利率评估 Web 服务(F3 结点)2 个副本。

实验环境是 Pentium4 3.0G,512M 内存,组合服务引擎 BPWS4J 2.1,Web 服务器 Tomcat,10M 局域网,每个组合服务引擎部署到不同主机上。

图 3(a)是在 100ms 到达引擎的 SOAP 数量和响应



(a) 负载与相应时间对比图



(b)负载与服务相应正确率

图 3　负载与提供服务 QOS 对比图

时间。图 3(b)是请求数量与服务正确响应率。图 4 是按照分解 1 得到的角色为 Role 本地流程 Process,该虚拟服务组合包含调用风险评估 Web 服务,类型为“分布式 1”的风险评估 Web 服务没有副本,而“分布式 1 *”有 1 个副本。

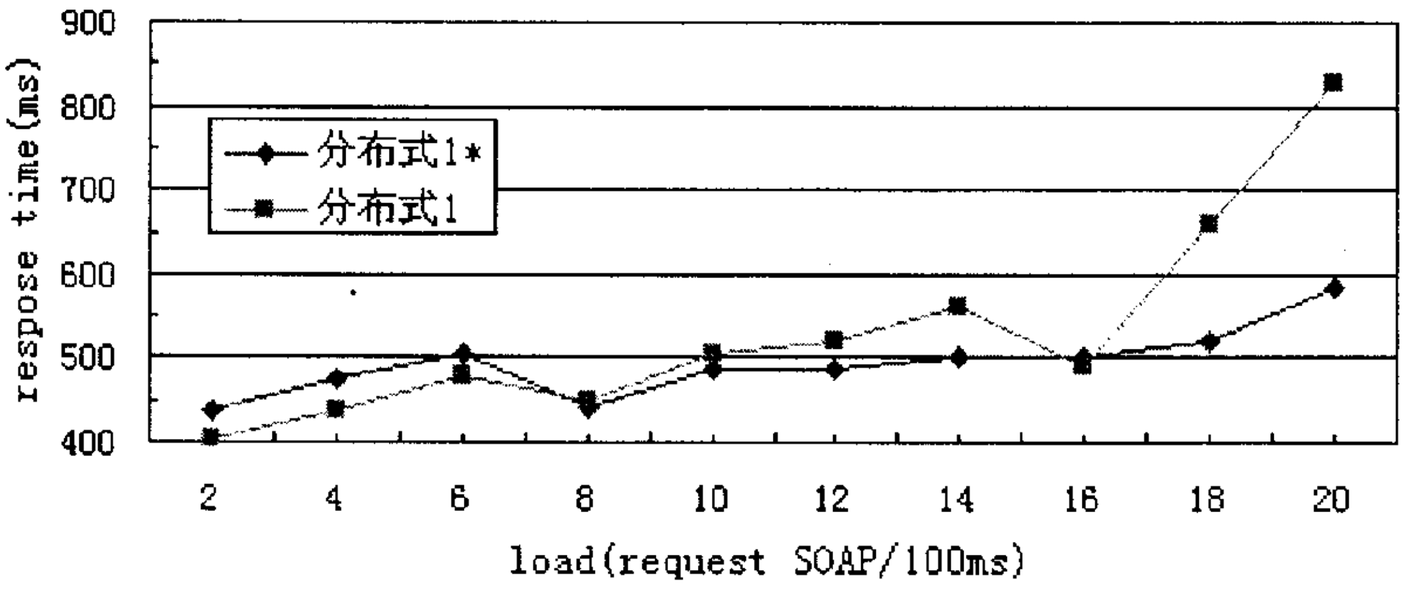


图 4 负载与响应时间在不同服务副本对比图

图 3 比较分布式与集中式服务组合,在 10～16(request SOAP/100ms)中载和 16～18(request SOAP/100ms)重载时,提供的服务组合优于集中式服务组合,而分解 2 比分解 1 在重载时能提供 1100ms 响应时间和 90%以上的正确执行的服务组合。图 4 测试了在随着服务副本增加使得在中载和重载时,响应时间保持在 0.5s 左右。

4 结论和相关工作

应用分解算法分解产生 n(输入的角色数量)个本地流程,且本地流程没有绑定具体的 Web 服务,如果假设,某个本地流程接收客户 SOAP 请求,该本地流程通过与其他 $n-1$ 个本地流程交互,实现服务组合,将结果返回给用户,这种组合方式称为虚拟服务组合。因此在考虑分布式服务组合的 QOS 时,应从 n 个本地流程整体和局部方面考虑,以及考虑负载均衡[5],并结合对客户的 SOAP 请求到达的概率,考虑引擎在服务组合执行时的 QOS。现阶段在服务组合的 QOS 着重研究基于客户的 QOS,依据此刻的服务组合和参与编排的 *Web* 服务的负载(非该客户请求的服务组合运行时的负载),根据算法(如免疫算法[6])进行服务副本选择。而负载均衡则实时均衡主机间的负载[5]。在服务组合 QOS 的服务副本选择上考虑服务副本所在的主机的负载均衡问题。而这将成为服务组合的 QOS 研究的新热点。

参考文献:

[1] 基于服务的建模和架构[EB/OL]. 2004. http://www-128.ibm.com/developerworks/cn/webservices/ws-soa-design1.

[2] Business Process Execution Language for Web Services. Version1.1[EB/OL]. 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/bpel-ws.pdf.

[3] Nanda M G, Chandra S, Sarkar V. Decentralizing Execution of Composite Web Services[C]//Conference on Object Oriented Programming Systems Languages and Applications. New York: ACM Press, 2003:170-187.

[4] 刘必欣,王玉峰,贾 焰,等. 一种基于角色的分布式动态服务组合方法[J]. 软件学报,2005,16(10):1859-1867.

[5] 李文中,郭 胜,许 平,等. 服务组合中一种自适应的负载均衡算法[J]. 软件学报,2006,17(5):1859-1867.

[6] GAO Yan, NA Jun, ZHANG Bin, et al. Immune Algorithm for Selecting Optimum Services in Web Services Composition[J]. Wuhan University Journal of Natural Sciences, 2006, 11(1): 221-225.

(上接第 39 页)

通过实验结果可以看到采用了新的改进算法后可以将许多错误被分开的簇重新合并,并可以保证合并的准确率在 90%以上。事实证明此算法的改进是可行并有效的。同时由于采用的处理方法比较简单,而且没有附加的磁盘 I/O 操作,在算法的执行效率上比 OPTICS 要好得多。

7 总 结

基于密度的聚类算法 DBSCAN 是一种有效的算法,由于它可以发现任意形状的簇而且能够较好地减少噪声的干扰,因此能被广泛地应用。通过对此算法的改进,弥补了其本身的不足,因而可以更有效地利用此算法进行数据挖掘。

参考文献:

[1] Han Jiawei, Kamber M. 数据挖掘概念与技术[M]. 北京:机械工业出版社,2001.

[2] Ankerest M, Breuning M M. OPTICS: ordering, points to identify the clustering structure[C]//Proc. ACM SIGMOD Int Conf on Management of Data. Philadeiphia, PA: ACM Press, 1999:49-60.

[3] Jihong G, Shuigeng Z, Fuling B, et al. Scaling up the DBSCAN Algorithm for Clustering Large Spatial Databases Based on Sampling Technique[J]. Wuhan University Journal of Natural Sciences, 2001, 6(1-2):467-473.

[4] 蔡颖琨,谢昆青,马修军. 屏蔽了输入参数敏感性的 DBSCAN 改进算法[J]. 北京大学学报,2004,40(3):102-106.

[5] 黄永平,王丽珍. 考虑对象方向关系的密度聚类算法[J]. 云南大学学报,2004,26(3):216-219.