# 基于服务关联模型的服务排序算法——ServiceRank

朱怡安[1]，雷万保[2]

(1. 西北工业大学软件与微电子学院 西安 710072; 2. 西北工业大学计算机学院 西安 710072)

【摘要】提出一种以Web service关联结构为基础的权重计算和排序算法——ServiceRank。在ServiceRank中，每个Web service借助后继可组合关联关系，将自身的权重以连接度为尺度平均地分配给所有后继可组合Web service，并以类似的方法从所有前驱可组合Web service中获取权重。与现有方法相比，ServiceRank能提供全局一致的权重函数，且能离线工作，算法稳定性好且效率高。通过实验对算法的收敛性和稳定性进行了深入分析。

**关 键 词** 服务组合; 服务排序; 服务关联结构; 面向服务的计算; Web服务

**中图分类号** TP306.1 **文献标识码** A doi:10.3969/j.issn.1001-0548.2011.04.027

# Service Ranking Algorithm Based on Service's Relational Model—ServiceRank

ZHU Yi-an[1] and LEI Wan-bao[2]

(1. School of Software and Micro-electronics, Northwest Polytechnical University Xi'an 710072;

2. School of Computer Science and Technology, Northwestern Polytechnical University Xi'an 710072)

**Abstract** On the basis of Web service relational structure, this paper proposes an algorithm called ServiceRank to compute the service weight and rank the services In ServiceRank, by analyzing the relationship among the subsequent composable Web services, each Web service averagely distributes its weight by connectivity to all the subsequent composable Web services, and captures weight from all the predecessor composable Web services through the similar method. Compared with the current methods, ServiceRank has the ability to compute the unified weight function and to work offline. This paper thoroughly analyzes the convergence and stability of SercviceRank through experiments, which proves that the proposed algorithm has a high stability and efficiency.

**Key words** service composition; service ranking; service relational structure; SOC; Web service

作为一种新的计算方式，面向服务的计算(service-oriented computing，SOC)凭借在平台互操作性、网络互访问性以及松散耦合等[1]方面所表现的优良特性，越来越受到国内外学术界和工业界(特别是在大规模企业级应用中)的关注。截止2007年，大约72%的应用软件支持Web service(WS)，45%的软件支持WS访问[2]。近几年，随着WS种类和数量的迅速增长，服务发现和服务组合算法返回的WS和WS组合路径的数量不断增加，如何有效地对这些WS和WS路径进行排序成为一个急待解决的问题。现有的研究内容主要集中在服务发现框架、服务组合算法[3-5]和语义级关联关系描述[6]等方面，而对服务权重计算和排序的研究处于初始阶段(如文献[3])。在具体实现方式上，现有服务权重计算和排序算法多以WS间的语法或语义相似度为基础，并在每次服务发现和服务组合过程中采用在线计算方法进行实时WS权重计算和排序。通常，该类算法设计简单且易于实现，但缺点也是显然的，如：1) 每个WS在不同服务发现和服务组合过程中的权重与上下文有关，缺乏一个统一且一致的权重计算函数，易发生权重抖动；2) 每次服务发现和服务组合过程都必须进行权重重新计算，存在大量重复计算，影响算法执行效率。

文献[7]通过挖掘Web网页之间的链接关系，构建链接关联矩阵，并以此为基础实现了对大规模Web网页的权重计算，成为第二代搜索引擎研究中具有里程碑意义的算法。本文在充分分析WS特点并借鉴文献[7]思想的基础上，提出一种有效的WS权重计算和排序算法—ServiceRank。与现有WS权重计算和排序算法不同，在ServiceRank中，每个WS借助后继可

收稿日期：2009 – 11 – 13; 修回日期：2010 – 12 – 18
基金项目：部级基础科研项目；部级预研基金
作者简介：朱怡安(1961 – )，男，教授，主要从事软件工程、计算机网络和服务计算方面的研究.

组合关联关系将自身的权重根据连接度平均地分配给所有后继可组合WS，并以类似的方法从所有前驱可组合WS中获取权重。与现有算法相比，ServiceRank不仅能够提供一个全局统一且一致的权重计算函数和排序算法，还具备离线工作和分析能力，可有效地避免权重抖动和重复计算操作，提高服务发现和服务组合的效率，以及返回权重向量的稳定性。

# 1 ServiceRank算法描述

## 1.1 基本定义

**定义 1** 一个WS集关联关系可以用二元组表示为$D(S,L)$，其中，$S$表示所有WS的集合，即$S=(s_1, s_2,\cdots, s_n)$，$n$为WS的总数；$L$表示WS间的关联关系，$L=(L_{ij})_{n\times n}$，$1\leqslant i, j\leqslant n$。本文中的关联关系主要指以输入(INPUT)、输出(OUTPUT)、前置条件(PRECONDITION)以及影响(EFFECT)等为基础的可组合关系，且有：

$$L_{ij}=\begin{cases}\left|s_i\rightarrow s_j\right| & \text{存在}s_i\rightarrow s_j\\ 0 & \text{否则}\end{cases} \tag{1}$$

**定义 2** 一个节点的出度函数是指以该节点为源节点的所有关联关系的函数，记为DegOut；一个WS集关联矩阵是指用于描述所有WS之间关联关系的矩阵，记为$\boldsymbol{T}$。通常有：

$$\text{DegOut}(i)=\sum_j L_{ij}\qquad 1\leqslant i,\ j\leqslant n \tag{2}$$

$$T_{ij}=\begin{cases}L_{ij}\big/\text{DegOut}(i) & \text{DegOut}(i)>0,\ 1\leqslant i,\ j\leqslant n\\ 0 & \text{否则}\end{cases} \tag{3}$$

**定义 3** 一个WS集的权重向量是指以每个WS服务的权重作为元素值的向量，记为$\boldsymbol{w}(s)=(\boldsymbol{w}(s_1),\boldsymbol{w}(s_2),\cdots,\boldsymbol{w}(s_n))$，$\boldsymbol{w}(s_i)\geqslant 0$，$1\leqslant i\leqslant n$，$n$为Web服务数量。

**定义 4** 对于WS集中的任意节点$s_i$和$s_j$而言，如果存在一条以$s_i$为源节点及$s_j$为目的节点的关联关系，则称$s_i$是$s_j$的前驱可组合服务，称$s_j$是$s_i$的后继可组合服务。

## 1.2 ServiceRank算法改进

与传统服务权重计算和排序算法不同，ServiceRank更强调WS之间以关联关系为基础建立的相互“信任”关系，并以此作为权重计算的基础。在ServiceRank中，每个WS通过后继可组合关联关系将自身的权重以连接度为尺度平均地分配给所有后继可组合WS，并以类似的方法从所有前驱可组合WS中获取权重。通常，一个节点的前驱可组合节点越多，则权重越大；同时，对于某些节点而言，如果前驱可组合节点的数量并不多，但存在一些权重比较大的前驱可组合节点，则其权重也会很大。换言之，一个WS的权重大小取决于前驱可组合服务的权重及该前驱可组合服务的后继可组合WS的数量，前驱可组合WS的权重越大，且其后继可组合WS的数量越小，则当前WS的权重越大；反之亦然。

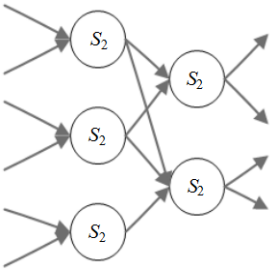


图1 服务关联关系图

如图1所示，如果$s_4$和$s_5$是$s_1$、$s_2$、$s_3$仅有的两个后继可组合WS，且$s_1$、$s_2$、$s_3$也是$s_4$和$s_5$仅有的3个前驱可组合WS，则利用ServiceRank算法有：

$$\boldsymbol{w}(s_4)=\frac{1}{2}\boldsymbol{w}(s_1)+\frac{1}{2}\boldsymbol{w}(s_2)$$

$$\boldsymbol{w}(s_5)=\frac{1}{2}\boldsymbol{w}(s_1)+\frac{1}{2}\boldsymbol{w}(s_2)+\boldsymbol{w}(s_3)$$

由式(1)、(2)和(3)知，对于任意转换矩阵$\boldsymbol{T}$，每一行向量表示该行所对应的入度权重向量，每一列向量表示该列所对应的出度权重向量。对于任一WS集，如果设$\boldsymbol{w}(s)=(\boldsymbol{w}(s_1),\boldsymbol{w}(s_2),\cdots,\boldsymbol{w}(s_n))^{\mathrm{T}}$，第$k$步服务权重向量为$\boldsymbol{w}(s)^{(k)}$，则有：

$$\boldsymbol{w}(s_j)^{(k+1)}=\sum_{(s_i\rightarrow s_j)}w(s_i)^{(k)*}\,|\,s_i\rightarrow s_j\,|\,/\,\text{DegOut}(i)=\sum_i(T_{ij}*w(s_i)^{(k)}) \tag{4}$$

即：

$$\boldsymbol{w}(s)^{(k+1)}=\boldsymbol{T}^{\mathrm{T}}*\boldsymbol{w}(s)^{(k)} \tag{5}$$

由上述可知，在具体权重计算过程中，可设不同时为0的任意常量为初始特征向量$\boldsymbol{w}(s)^{(0)}$，如总关联关系数的倒数。从理论上讲，如果不发生权重“丢失”，初始向量的选择不会影响最终稳定状态的权重特征向量值，但可能影响迭代次数。在经过若干次重复计算后，利用式(6)作为终止条件，从而得到WS集中的权重向量：

$$\left|\boldsymbol{w}(s)^{(k+1)}-\boldsymbol{w}(s)^{(k)}\right|<\zeta \tag{6}$$

式中，$\zeta$为某一常数。

## 2　ServiceRank算法改进

ServiceRank算法的基础是每个服务都能从前驱可组合WS继承权重，并通过后继关联关系将自身的权重平均分配给所有后继可组合WS。换言之，如果要保证ServiceRank有效地运行，必须保证任意两个WS节点在有限步内是可达的。但在实际分析过程中，这通常是不现实的。如图2所示，在EEE05测试集中，在不同服务数情况下连接比都很低，很难保证所有节点同时具备前驱可组合和后继可组合节点，即易出现权重“丢失”和权重“黑洞”现象。所谓权重“丢失”是指因某些WS没有前驱可组合或后继可组合节点，导致某些节点始终无法获取其他节点的权重或无法将自身的权重顺利地传递给后继可组合节点。对于没有后继可组合节点而言，经过若干次迭代后，其他WS的权重将自动地向其积蓄，从而形成权重“黑洞”。

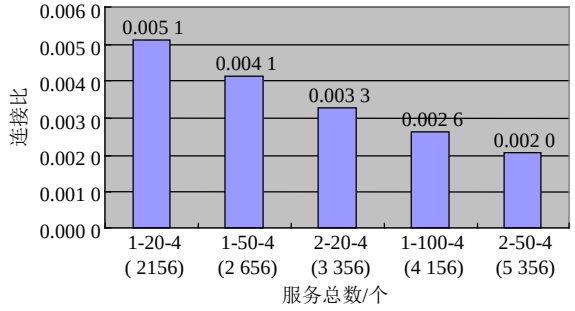


图2　服务连接比分析

出现权重“丢失”和权重“黑洞”在的根本原因在于一些节点缺少前驱或后继可组合节点，从而导致权重无法在各WS之间顺利流动，造成某个节点权重“只进不出”或“只出不进”。为了解决该问题，本文提出“虚关联”概念。虚关联是相对于实际的关联关系而言的，即该关联关系本身不存在，仅作为保证权重顺利流动的一种方式。“虚关联”构造过程如下：

1) 预处理阶段。该阶段主要对没有前驱可组合或没有后继可组合WS进行“次级关联”关系处理，该关联关系通常比实关联关系要“弱”，但比第二阶段“虚关联”关系要“强”。该处理阶段的主要目的是防止部分节点因缺少实关联关系的支持，导致迭代次数太多，影响算法性能。设次级关联权重向量$\boldsymbol{r}=(r_1,r_2,\cdots,r_n)^{\mathrm{T}}$；调节向量$\boldsymbol{p}=(p_1,p_2,\cdots,p_n)^{\mathrm{T}}$和$\boldsymbol{q}=(q_1,q_2,\cdots,q_n)^{\mathrm{T}}$，$n$为Web服务数量，且有：

$$p_i=\begin{cases}1 & \sum_i L_{ij}=0\\ 0 & \text{否则}\end{cases} \tag{7}$$

$$q_i=\begin{cases}1 & \sum_j L_{ij}=0\\ 0 & \text{否则}\end{cases} \tag{8}$$

则式(3)中的转换矩阵$\boldsymbol{T}$可变更为：

$$\boldsymbol{T}^{(1)}=\boldsymbol{T}+\boldsymbol{p}\boldsymbol{r}^{\mathrm{T}}+(\boldsymbol{q}\boldsymbol{r}^{\mathrm{T}})^{\mathrm{T}} \tag{9}$$

2) 构建“虚关联”。建立“虚关联”后，每个WS都会与其他WS存在“虚关系”，从而保证任意两个WS在有限步都是可达的，即权重可以在各WS之间顺畅流动。另外，对每个WS而言，其前驱可组合和后继可组合WS都是整个WS集，故避免了某些“只进不出”或“只出不进”节点因权重累赘而造成后继可组合WS的权重偏大，并最终影响权重向量的质量。设$\boldsymbol{u}=(u_1,u_2,\cdots,u_n)$，$\boldsymbol{e}=(1)_{1\times n}$，则式(9)中的$\boldsymbol{T}^{(1)}$变更为：

$$\boldsymbol{T}^{(2)}=c\boldsymbol{T}^{(1)}+(1-c)\boldsymbol{u}^{\mathrm{T}}\boldsymbol{e}\qquad 0<c\leqslant 1 \tag{10}$$

式中，$\boldsymbol{u}^{\mathrm{T}}\boldsymbol{e}$是“虚关联”矩阵，用于构建“虚关联”关系；$c$是调节常数，主要用于调整“虚关联”权重值。在PageRank[7]算法中，$c$的建议值为0.85～0.9。

本文以实例分析ServiceRank的运行过程。如图3所示，$s_1$～$s_6$为WS集中的所有WS，$p_1$～$p_{12}$为服务库中所有WS包含的输入和输出参数名。图中某些节点(如$s_6$)同时缺少前驱和后继可组合WS，即存在权重“丢失”现象。如上述分析，为了有效地完成对WS集的权重向量计算，本文引入“虚关联”，并以式(7)计算权重。

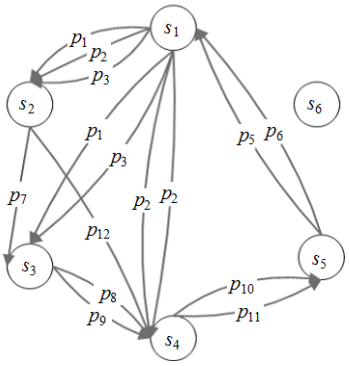


图3　服务关联实例

设$\boldsymbol{p}^{(0)}=[1,1,1,1,1,1]^{\mathrm{T}}$，$\zeta=10^{-9}$，$c=0.85$，$\boldsymbol{r}=(1/N)_{n\times 1}$，其中，$N$为关联关系数。根据图2，以及式(1)～(3)、(9)和(10)等，可得：

$$\boldsymbol{T}^{(2)}=\begin{bmatrix}
0.003\,0 & 0.414\,0 & 0.277\,0 & 0.277\,0 & 0.003\,0 & 0.025\,9\\
0.009\,6 & 0.009\,6 & 0.444\,6 & 0.444\,6 & 0.009\,6 & 0.082\,1\\
0.009\,6 & 0.009\,6 & 0.009\,6 & 0.879\,5 & 0.009\,6 & 0.082\,1\\
0.009\,6 & 0.009\,6 & 0.009\,6 & 0.009\,6 & 0.879\,5 & 0.082\,1\\
0.879\,5 & 0.009\,6 & 0.009\,6 & 0.009\,6 & 0.009\,6 & 0.082\,1\\
0.166\,7 & 0.166\,7 & 0.166\,7 & 0.166\,7 & 0.166\,7 & 0.166\,7
\end{bmatrix}$$

故得：

$$\boldsymbol{p}^{(68)}=$$

$$[1.342\ 2\ 0.675\ 3\ 0.785\ 1\ 1.468\ 1\ 1.400\ 8\ 0.468\ 1]^{\mathrm{T}}$$

## 3 实验分析

本文以公共测试集EEE05[8]为分析对象，对本文所提出的ServiceRank算法进行收敛性和稳定性分析。实验环境设置如下：CPU Intel Pentium(R) 43.00 GHz；内存512 MB；操作系统Windows XP；虚拟内存3 GB；Matlab7.0；$\zeta$值为$10^{-9}$；虚向量$\boldsymbol{r}=(1/N)_{n\times 1}$，$n$为Web服务数量，$N$为关联关系数；$c$=0.85。

如图4所示，在不同WS数量下，ServiceRank都能够在有限次迭代后到达一个相对稳定状态。值得注意的是，式(9)和式(10)之间的根本区别在于，式(10)能够将权重通过“虚关联”更顺利地传递给其他WS节点，因此，无论如何调节$c$值(通常要求$c$大于0.5，否则预处理阶段的“次级关联”的权重向量将小于“虚关联”向量)，相同测试对象到达稳定状态时的迭代次数都是不变的。本文以compostion1-20-4测试集为例(EEE05的一个子集)，在0.5～0.9之间，以0.05为递增量，所得到的迭代次数都是26次，其他测试子集也存在同样的情况。

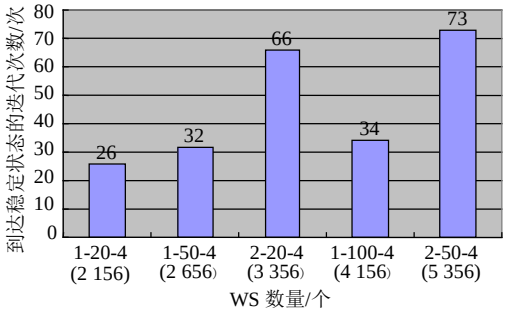


图4 Service Rank迭代次数分析

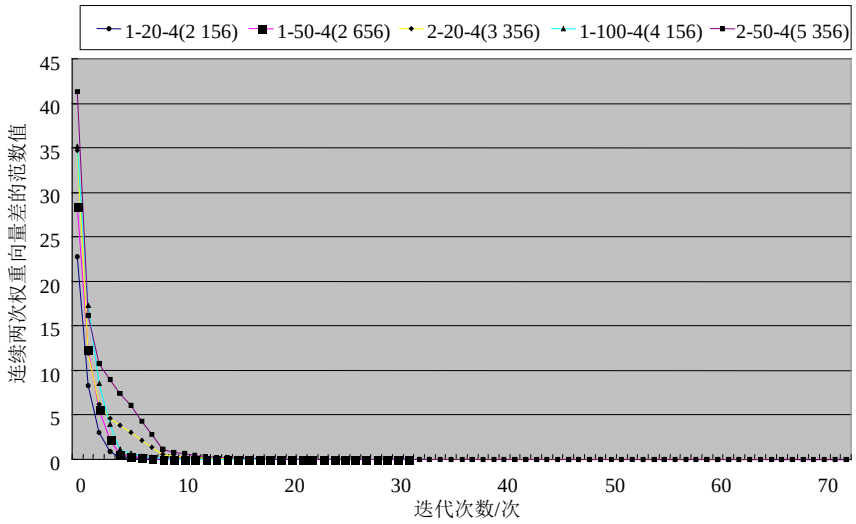


图5 算法的收敛性分析

不同WS数量情况下ServiceRank算法的收敛性如图5所示。由图可以看出，在不同WS数量情况下，ServiceRank收敛速度都比较快，且稳定性比较高。但值得注意的是，虽然调整因子$c$不能改变到达稳定状态的迭代次数，但对最终的权重值却能有一定的影响。

由式(5)可知，在整个计算过程中，转换矩阵$\boldsymbol{T}$主要在两次计算结果之间起桥梁作用，因此，计算过程开始后，整个算法所需要的存储空间主要是前后两次的计算结果，即需要$2n$个存储单元，故ServiceRank算法的空间复杂度为$O(n)$。

在传统服务发现和服务组合算法中，大多采用在线基于词法或语义的相似度计算方法，对于每次服务发现和组合请求必须实时进行权重计算和排序；对于不同的发现和组合过程，该类算法无法保证服务权重的一致性。从上述分析易知，ServiceRank与具体的服务组合请求内容无关，可离线工作，且可避免大量重复性权重计算操作，保证算法的执行效率和权重全局一致性。

## 4 结论和展望

本文在分析现有服务组合研究内容后，指出服务权重计算和排序算法研究的必要性，并以此为基础，提出基于服务关联结构的ServiceRank服务权重计算方法。与传统服务权重计算方法相比，该算法与服务组合请求内容无关，且可离线工作，在保证算法的公正性和公平性的同时，兼顾了算法的执行效率。本文以EEE05[8]测试集为分析对象，证明在不

同WS数量情况下，ServiceRank都能够通过有限次迭代获取稳定的WS权重值，且算法稳定性较高。

在后续工作中，将重点研究初始权重特征向量构造方法，以减少权重向量到达稳定状态的迭代次数，并探索高效的转换矩阵分割算法，以进一步减少算法空间复杂度和权重“丢失”，从而提升算法执行效率。

## 参 考 文 献

[1] 李曼, 王大治, 杜小勇, 等. 基于领域本体的Web服务动态组合[J]. 计算机学报, 2005, 28(4): 644-650.
LIMAN, WANG Da-Zhi, DU Xiao-yong, et al. Dynamic composition of Web services based on domain ontology[J]. Chinese Journal of Computers, 2005, 28(4): 644-650.

[2] CANTERA M. IT professional services forecast and trends for Web services[R]. Gartner Inc., 2004.

[3] GUERMOUCHE N, PERRIN O. RINGEISSEN C. A mediator based approach for services composition[C]//Sixth International Conference on Software Engineering Research, Management and Applications, Prague, Czech: Springer, 2008.

[4] ZENG Z L, NGU A H, BENATALLAH B, et al. Dynamic composition and optimization of Web services[J]. Distributed and Parallel Databases, 2008, 24(1-3): 45-72.

[5] IACOB S M, ALMEIDA J P A, IACOB M E. Optimized dynamic semantic composition of services[C]//Proceedings of the ACM Symposium on Applied Computing. Fortaleza, Ceara, Brazil: ACM, 2008.

[6] LECUE F, SILVA E, FERREIRA P L. A framework for dynamic Web services composition[C]//2nd ECOW Workshop on Emerging Web Services Technology, [S.l.]: [s.n.], 2007.

[7] PAGE L, BRIN S, MOTWANI R, et al. The pagerank citation ranking: Bringing order to the Web[R]. Stanford University, 1998.

编 辑 漆 蓉