软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(9):2136−2148 [doi: 10.13328/j.cnki.jos.004637]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

大数据下基于异步累积更新的高效 P-Rank 计算方法*

王旭丛[1], 李翠平[2], 陈 红[2]

[1](中国人民大学 信息学院 计算机系,北京 100872)

[2](中国人民大学 信息学院 数据仓库与商务智能实验室,北京 100872)

通讯作者: 王旭丛, E-mail: wangpu9003@126.com

摘 要: P-Rank 是 SimRank 的扩展形式,也是一种相似度度量方法,被用来计算网络中任意两个结点的相似性.不同于 SimRank 只考虑结点的入度信息,P-Rank 还加入了结点的出度信息,从而更加客观准确地评价结点间的相似程度.随着大数据时代的到来,P-Rank 需要处理的数据日益增大.使用 MapReduce 等分布式模型实现大规模 P-Rank 迭代计算的方法,本质上是一种同步迭代方法,不可避免地具有同步迭代方法的缺点:迭代时间(尤其是迭代过程中处理器等待的时间)长,计算速度慢,因此效率低下.为了解决这一问题,采用了一种迭代计算方法——异步累积更新算法.这个算法实现了异步计算,减少了计算过程处理器结点的等待时间,提高了计算速度,节省了时间开销.从异步的角度实现了 P-Rank 算法,将异步累积更新算法应用在了 P-Rank 上,并进行了对比实验.实验结果表明该算法有效地提高了计算收敛速度.

关键词: 异步累积更新;大数据;相似度;P-Rank;大规模计算

中图法分类号: TP311

中文引用格式: 王旭丛,李翠平,陈红.大数据下基于异步累积更新的高效 P-Rank 计算方法.软件学报,2014,25(9):2136−2148. http://www.jos.org.cn/1000-9825/4637.htm

英文引用格式: Wang XC, Li CP, Chen H. High-Efficiency P-Rank computation through asynchronous accumulative updates in big data environment. Ruan Jian Xue Bao/Journal of Software, 2014,25(9):2136−2148 (in Chinese). http://www.jos.org.cn/1000-9825/4637.htm

High-Efficiency P-Rank Computation Through Asynchronous Accumulative Updates in Big Data Environment

WANG Xu-Cong[1], LI Cui-Ping[2], CHEN Hong[2]

[1](Department of Computer Science, School of Information, Renmin University of China, Beijing 100872, China)

[2](Data Warehouse and Business Intelligence Laboratory, School of Information, Renmin University of China, Beijing 100872, China)

Corresponding author: WANG Xu-Cong, E-mail: wangpu9003@126.com

Abstract: P-Rank enriches the traditional similarity measure, SimRank. It is also a method to measure the similarity between two objects in graph model. Different from SimRank which only considers the in-link information, P-Rank also takes the out-link information into consideration. Consequently, P-Rank could effectively and comprehensively measure "how similar two nodes are". P-Rank is applied widely in graph mining. With the arrival of big-data era, the data scale which P-Rank processes is increasing. The existing methods which implement P-Rank, such as the MapReduce model, are essentially synchronous iterative methods. These methods have some shortcomings in common: the iterative time, especially the waiting time of processors during iterative computing, is long, thus leading to very low efficiency. To solve this problem, this paper uses a new iterative method—the Asynchronous Accumulative Update method. Different from the traditional synchronous methods, this method successfully implementes asynchronous computations and as a result reduces the waiting

* 基金项目: 国家自然科学基金(61272137, 61033010, 61202114); 国家高技术研究发展计划(863)(2014AA015204); 国家基础研究发展计划(973)(2012CB316205); 国家社会科学基金(12&ZD220); 中国人民大学科学研究基金(中央高校基本科研业务费专项资金资助)(10XNI018)

收稿时间: 2014-01-24; 修改时间: 2014-04-30; 定稿时间: 2014-06-09

time of processors during computing. This paper implements P-Rank using the asynchronous accumulative update method, and the experiment results indicate that this method can effectively improve the computation speed.

Key words: asynchronous accumulative update; big data; similarity; P-Rank; large-scale computation

随着大数据时代来临,很多数据挖掘算法中需要处理的数据量越来越大,于是,用于处理大规模数据的分布式系统如 Hadoop[1],Giraph[2]等应运而生.Hadoop 作为 MapReduce 计算模型的开源实现,已经被广泛应用于各种大规模数据计算;Giraph 是基于 hadoop 平台的大规模图处理框架,它提出并实现了 SuperStep(超级步)的概念.

但尽管如此,大规模数据计算所需求的计算速度仍受到限制,这在很多传统的迭代算法中尤其突出.因为很多迭代算法在使用 Hadoop 或者 Giraph 实现的时候,都受到了同步计算的限制.Hadoop 系统中的 Map 过程和 Reduce 过程以及 Giraph 中的 SuperStep 过程,其本质是以一次全局磁盘读写为一次迭代,而在处理海量数据时,要完成一次全局磁盘读写是相当耗时的.在整个计算期间,总有一些处理器会先于其他处理器完成分配给它的计算任务,但是为了等待全局计算,这些早已完成任务的处理器处于赋闲的状态,这就导致了大量不必要的时间开销.因此,很多迭代算法面临这样一个问题:迭代时间(尤其是迭代过程中处理器等待的时间)越来越长,计算速度越来越慢.

这些迭代算法有一个共同点,即需要根据上一次迭代计算的结果才能开始下一次迭代.在分布式系统中,每个处理器结点只处理部分数据,由于不同数据的复杂性不同,会导致不同处理器结点的计算速度不同.因此,这些传统的迭代算法需要做到同步计算:必须等到所有处理器结点都把第 k–1 迭代过程计算完毕后,才能开始第 k 次迭代.这就导致处理器在某些时候处于一种等待状态,延长了计算时间,降低了计算速度.

为了解决这一问题,本文采用了一种迭代计算方法——异步累积更新算法(asynchronous accumulative update,简称 AAU 算法)[3].这个方法保证了处理器结点的高效率运行,实现了异步计算机制,即不需等到所有 k–1 次计算都结束后才开始第 k 次计算,只要有新的数据被接收,处理器就一直计算,一直处于忙碌状态,减少了处理器结点的等待时间.如此,大大提高了计算速度,节省了时间开销.

P-Rank(penetrating rank)[4]是图数据管理领域中新提出的一种相似度度量方法,被用来计算网络中任意两个结点的相似性,它是 SimRank[5]的扩展形式.相比于 SimRank,P-Rank 可以更加准确客观地衡量结点间的相似程度,因为 SimRank 只考虑了结点的入度信息,而 P-Rank 同时考虑了结点的出度和入度信息.SimRank 在实际当中有很多应用,如社会网络链接预测[6]、推荐系统[7]等.同样地,P-Rank 也有非常大的潜力被应用到这类系统中.因此,本文将 P-Rank 作为了优化研究的对象.随着大数据技术的发展,图挖掘需要处理的数据日益增大.使用 MapReduce[8]等分布式模型实现大规模 P-Rank 迭代计算的方法,本质上是一种同步算法,其计算时间长,收敛速度慢,因此效率低下.本文从异步的角度实现了 P-Rank 算法,即将异步累积更新算法应用在了 P-Rank 上,有效地提高了计算收敛速度.

本文第 2 节详细介绍 P-Rank 定义和异步累积更新方法 AAU.第 3 节介绍如何将 P-Rank 原始形式转换成 AAU 异步形式,并介绍两个实现 P-Rank 的平台——Hadoop 平台和 Maiter[9]平台:前者为同步计算;后者实现了 AAU 算法,为异步计算.第 4 节是实验部分,通过与传统的 MapReduce 方法对比,着重介绍 AAU 算法的精确性和快速性.第 5 节对全文进行总结.

1 相关工作

为满足各类图数据日益增长的需求,很多基于图的分布式模型被开发出来,如 Pregel[10],GraphLab[11]等.这些系统采用以顶点为中心(vertex-centric)的方式进行图数据迭代计算.Apache Giraph[2]正是 Pregel 的开源实现.这些以顶点为中心的系统将原始图数据划分成很多部分,然后使用 vertex-centric 算法进行迭代计算,易于实现,且在很多图算法中都有应用价值.但这些算法也有缺陷,如它将 Data Partition(数据划分)过程隐藏起来,使得程序员在应用不同的图算法时不能进行这方面的优化,错失了很多进行算法优化的机会.

Giraph++[9]正是为解决这一问题被提出的.Giraph++也是一个分布式图数据处理模型,在 Giraph 的基础上

实现,从以顶点为中心(vertex-centric)的计算模型转变为以图为中心(graph-centric)的计算模型,将 Partition 过程对用户开放,使得用户可以自定义数据划分过程,从而为算法优化提供了另一个着眼点.并且,文献[9]证明了这种计算模型比以顶点为中心的计算模型速度更快.

iMapReduce[23]是 MapReduce 的优化形式,MapReduce 的每次迭代都要进行一次全局数据读写,而 iMapReduce 只需要计算开始前从 DFS 读取一次,计算结束后向 DFS 写一次,其他只需在本地进行数据更新即可,因此在数据流方面节省了运算时间.PrIter[24]从优先级迭代的角度进行 MapReduce 的优化,每次进行 Map 前要先检查数据的优先级,只有优先级高的才会优先计算,合理设置优先条件可以极大地节省计算时间.

以上这些分布式框架,从系统实现的角度给了用户进行算法优化的机会.然而,这些系统的实现机制仍然停留在同步计算的角度,不可避免具有上文提到过的同步迭代计算的缺点.

SimRank 作为图数据处理领域的热点之一,在算法应用和优化层面,有基于 SimRank 相似连接(similarity join)查询[12,13]和 SimRank 优化计算[14,15].从计算数据规模方面,有人提出 Single- Pair SimRank[16]计算,以区别于 All-Pair SimRank 计算.

相似度度量可以划分为两大类[17]——基于内容的度量和基于链接的度量.SimRank,PPR,Hitting time 都是基于链接的相似度度量.除了上述度量之外,P-Rank[18]是 SimRank 的扩展,用来处理图数据类型多样的情况,同时考虑了入度和出度(SimRank 只考虑了出度链接).MatchSim[19]将两个结点的最大匹配相似邻居对的平均相似度作为它们的相似度度量.SimFusion[20]和 PathSim[21]用来处理异构网络上的相似度(SimRank 应用在同构网络上).RoudTripRank[22]把 specificity 和 importance 结合在一起作为一种度量方式.

这些相似度优化研究一方面着眼于改进 SimRank 算法以提高计算速度,另一方面则着重寻找新的相似度度量方法,很少有人从实现机制上考虑进行优化,仍然通过传统的同步迭代机制进行相关研究.

而文献[3]提出的异步累积更新方法(AAU 方法)正是从实现机制考虑了算法优化,提出并实现了异步计算.作者详细阐明了异步累积更新方法的内涵、使用条件、实现模型等,很多传统迭代算法都可以用这种异步方法进行异步计算,只要满足 AAU 方法的使用条件,都可以使用这个模型进行优化计算,本文的 P-Rank 便是其中之一.Maiter 系统是 AAU 方法的实现框架,它在文献[8]里有非常详细的介绍,并且它是一个开源系统,只要用户拥有 1 台或者多台 PC 机,就可以安装本系统进行相关研究.

2 基本定义和方法介绍

2.1 P-Rank定义

P-Rank 是新提出的用来计算对象间相似度的度量方法.P-Rank 算法的核心思想包括两方面:

(1) 如果两个对象 a,b 被相似的对象所指向,那么 a,b 是相似的;

(2) 如果两个对象 a,b 指向了相似的对象,那么 a,b 是相似的.

显然,第 1 条就是 SimRank 的定义,它根据结点的入度结点计算其相似性.相比于 SimRank,P-Rank 增加了第 2 条,将结点的出度信息也考虑进来.因此,P-Rank 将同时根据结点的出度信息和入度信息计算其相似性.

P-Rank 通常被用于处理有向图.如果将对象看做图的顶点,对象间的关系看做一条有向边,那么这个有向图可以表示成 $G=\langle V,E\rangle$,其中,V 是顶点的集合,E 是有向边的集合.在传统的迭代方法中,P-Rank 的计算式如下:

$$S_{ab}^{k}=\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}S_{mn}^{k-1} \tag{1}$$

其中,S_{ab}^{k} 表示结点 a 和 b 在第 k 次迭代时的相似度;$|I(a)|$和$|I(b)|$分别是 a 和 b 的入度数;$|O(a)|$和$|O(b)|$分别是 a 和 b 的出度数;i 和 j 分别是 a 和 b 的入度结点;m 和 n 分别是 a 和 b 的出度结点;c 是常量,λ是用来设置出度信息和入度信息权重的参数,c 和λ取值都在区间[0,1]中.如果 $a=b$,那么 $S_{ab}=1$.

从公式(1)可以看出,$\langle a,b\rangle$的相似度是由 a、b 的入度结点对$\langle i,j\rangle$的相似度和 a、b 的出度结点对$\langle m,n\rangle$共同决定的.如果 $i\to a$ 且 $j\to b$,那么$\langle i,j\rangle$就会对$\langle a,b\rangle$产生影响;如果 $a\to m$ 且 $b\to n$,那么$\langle m,n\rangle$也会对$\langle a,b\rangle$产生影响.因此,可以把 S_{ab} 看做是$\langle i,j\rangle$和$\langle m,n\rangle$对$\langle a,b\rangle$的影响值之和.

2.2 异步累积更新方法

异步累积更新算法(asynchronous accumulative updates,简称 AAU)由张岩峰等人[3]提出,本文在此只作简单介绍.

迭代算法的通式为 $v^k=F(v^{k-1})$,其中,v 是需要计算的数据集,F 函数是需要反复执行的操作.通常,v 是一个多维向量,可以写成 $v^k=(v_1^k,v_2^k,...,v_n^k)$,代表第 k 次迭代计算后的结果.在很多情况下,F 函数其实是一系列函数 F_j 的集合,每个 F_j 函数只负责计算向量 v 的第 j 个元素 v_j,即:

$$v_j^k=F_j(v_1^{k-1},v_2^{k-1},...,v_n^{k-1}) \tag{2}$$

在分布式系统中,多个处理器并行进行计算.向量 v 的 n 个维度 v_1~v_n 被分配成几组,每组的 v_j 由相应的处理器负责计算.从这个公式可以明显看出,只有当所有第 k–1 次迭代 v_j^{k-1} 计算结束后,才能开始第 k 次计算.由于不同处理器处理的数据复杂性不同,会导致某些处理器经常处于赋闲状态.为解决这一问题,AAU 算法被提出.

AAU 算法首先将传统迭代公式进行变型,然后再改写成可以进行异步计算的形式.

2.2.1 AAU 公式

在某些情况下,公式(2)可以改写为

$$v_j^k=F_j(v_1^{k-1},v_2^{k-1},...,v_n^{k-1})=G_{\{1,j\}}(v_1^{k-1})\oplus G_{\{2,j\}}(v_2^{k-1})\oplus...\oplus G_{\{n,j\}}(v_n^{k-1}),$$

其中,$G_{\{i,j\}}(v_i^{k-1})$ 表示 v_i 在第 k–1 迭代中的值对 v_j 在第 k 次迭代中值的影响值;⊕是某种运算,满足交换律、结合律和分配率.

令 $\Delta v_j^k=v_j^k-v_j^{k-1}$,则通过一系列变形后,公式(2)最终转化为公式(3)的形式:

$$\begin{cases} v_j^k=v_j^{k-1}\oplus\Delta v_j^k \\ \Delta v_j^k=\sum\limits_{i\to j}\oplus G_{\{i,j\}}(\Delta v_i^{k-1}) \end{cases} \tag{3}$$

其中,v_j^k 表示 v_j 在第 k 次迭代后的值;v_j^{k-1} 表示 v_j 在第 k–1 次迭代后的值;Δv_j^k 表示从第 k–1 次迭代到第 k 次迭代的变化量;$G_{\{i,j\}}(\Delta v_i^{k-1})$ 表示 i 的变化量Δv_i 对 j 的变化量Δv_j 的影响值;⊕是某种运算,具有结合律、交换律和分配率.

公式(3)可以理解为:v_j 每一次迭代的值,等于上一次迭代的值加上它的变化值;而 v_j 的变化值,等于所有对 j 有影响的 i 的变化值的累加.只有 $i\to j$ 存在时,i 才会对 j 产生影响.

在分布式系统中,处理器 j 先收集其他处理器送来的 $G_{\{i,j\}}(\Delta v_i^{k-1})$,然后根据公式(3)更新 Δv_j^k,进而更新 v_j^k.

2.2.2 AAU 公式的异步形式

公式(3)本质上还是同步形式,因为 v_j^k 总是需要等待 Δv_j^k 更新完毕后才能更新,而 Δv_j^k 又需要等待其他处理器都完成计算并且把数据发送过来后才更新.因此要作进一步改变,变为异步形式:

$$\begin{aligned} &\text{Receive}:\Delta v_j^k=\Delta v_j^k\oplus G_{\{i,j\}}(\Delta v_j^{k-1}) \\ &\text{Update}:\begin{cases} 1.\ v_j^k=v_j^{k-1}\oplus\Delta v_j^k \\ 2.\ \text{把}G_{\{j,h\}}(\Delta v_j^k)\text{发送给处理器}h,\text{如果}j\to h\text{存在} \\ 3.\ \Delta v_j^k=0 \end{cases} \end{aligned} \tag{4}$$

公式(4)可以这样理解:处理器 j 维护两个线程——Receive 线程和 Update 线程:

- Receive 负责接收数据,累积 Δv_j^k;
- Update 负责更新 v_j^k,并向其他处理器发送相应的影响值 g.每经过一段时间,处理器就执行步骤 1,及时将 Δv_j^k 累计到 v_j^k 上;当步骤 2 计算完毕后,将 Δv_j^k 归零,再重新开始新一轮 receive-update 过程.

从这个公式可以看到:每个处理器上的 Receive 和 Update 操作都是彼此独立的,任意一个处理器可以在任何时间进行这两种操作.因此,处理器不需等到所有 k–1 次计算都结束后才开始第 k 次计算,只要有新数据被送

来,就开始计算,一直处于忙碌状态,且彼此独立,减少了等待时间.这样便实现了异步计算.

3 P-Rank 异步方法和实现

3.1 P-Rank的异步形式

AAU 算法公式的使用条件有两个:

(1) 公式(2)中的迭代公式 F_j 可以拆分成一系列 $G_{\{i,j\}}(x)$的形式;

(2) ⊕运算满足结合律、交换律和分配率.

我们先来证明 P-Rank 公式如何满足这两个条件.

令 $G_{\{(i,j),(a,b)\}}(S_{ij}^{k-1})=\dfrac{\lambda C}{|I(a)||I(b)|}S_{ij}^{k-1}, G'_{\{(m,n),(a,b)\}}(S_{mn}^{k-1})=\dfrac{(1-\lambda)C}{|O(a)||O(b)|}S_{mn}^{k-1}$,则公式(1)可改写为

$$S_{ab}^{k}=\sum_{i\to a\in E}\sum_{j\to b\in E}G_{\{(i,j),(a,b)\}}(S_{ij}^{k-1})+\sum_{a\to m\in E}\sum_{b\to n\in E}G'_{\{(m,n),(a,b)\}}(S_{mn}^{k-1}) \tag{1'}$$

显然, S_{ab}^{k} 写成了一系列 $G_{\{i,j\}}(x)$的形式;又 S_{ab}^{k} 是加和运算,"+"法显然满足结合律、交换律和分配率.

至此,证明了 P-Rank 公式满足了 AAU 的两个条件,即 P-Rank 可以使用 AAU 算法.下面给出如何将 P-Rank 公式写成可以进行异步计算的形式.

令 $\Delta S_{ab}^{k}=S_{ab}^{k}-S_{ab}^{k-1}$,则

$$\begin{aligned}\Delta S_{ab}^{k}&=\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}S_{mn}^{k-1}-\\&\left(\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}S_{ij}^{k-2}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}S_{mn}^{k-2}\right)\\&=\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}(S_{ij}^{k-1}-S_{ij}^{k-2})+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}(S_{mn}^{k-1}-S_{mn}^{k-2})\\&=\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}\Delta S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}\Delta S_{mn}^{k-1}.\end{aligned}$$

整合后,可以写成:

$$\begin{cases}S_{ab}^{k}=S_{ab}^{k-1}+\Delta S_{ab}^{k}\\ \Delta S_{ab}^{k}=\displaystyle\sum_{i\to a\in E}\sum_{j\to b\in E}\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}\Delta S_{mn}^{k-1}\end{cases} \tag{5}$$

公式(5)和公式(3)是对应的.进一步地,为了实现异步,再将公式(5)带入公式(4),改写为异步形式:

$$\text{Receive}:\begin{cases}\Delta S_{ab}^{k}=\Delta S_{ab}^{k}+\dfrac{\lambda c}{|I(a)||I(b)|}\Delta S_{ij}^{k-1}\\ \Delta S_{ab}^{k}=\Delta S_{ab}^{k}+\dfrac{(1-\lambda)c}{|O(a)||O(b)|}\Delta S_{mn}^{k-1}\end{cases}$$

$$\text{Update}:\begin{cases}1.\ S_{ab}k=S_{ab}k-1+\Delta S_{ab}^{k}\\ 2.\ \text{把}G_{\{(a,b),(h,k)\}}(\Delta S_{ab}^{k})=\dfrac{\lambda c}{|I(h)||I(k)|}\Delta S_{ab}^{k}\text{发送给}\langle h,k\rangle\text{所在的处理器,如果}a\to h\in E,b\to k\in E;\\ \quad\text{把}G'_{\{(a,b),(x,y)\}}(\Delta S_{ab}^{k})=\dfrac{(1-\lambda)c}{|O(x)||O(y)|}\Delta S_{ab}^{k}\text{发送给}\langle x,y\rangle\text{所在的处理器,如果}x\to a\in E,y\to b\in E;\\ 3.\ \Delta S_{ab}^{k}=0\end{cases} \tag{6}$$

公式(6)即为 P-Rank 实现异步累积更新计算所需的形式.在本文第 3.3.2 节,用分布式平台 Maiter 实现了这种计算.

假设处理器 A 负责计算结点对$\langle a,b\rangle$的相似度.处理器 A 维护两个线程——Receive 线程和 Update 线程:

- Receive 负责接收数据,累积 ΔS_{ab}^k;
- Update 负责更新 S_{ab}^k,并向其他处理器发送$\langle a,b\rangle$对其他结点对的影响值 G.每经过一段时间,处理器 A 就执行步骤 1,及时将 ΔS_{ab}^k 累计到 S_{ab}^k 上;当步骤 2 计算完毕后,将 ΔS_{ab}^k 归零,再重新开始新一轮 receive-update 过程.

3.2 P-Rank的收敛性

由于在实验部分需要讨论 P-Rank 的计算时间,所以有必要确定什么时候计算终止.对于迭代公式来说,计算终止的条件就是迭代结果足够收敛,前提是迭代公式本身是收敛的.因此在本节中,我们将证明 P-Rank 原始公式的收敛性和 P-Rank 异步形式的收敛性.

3.2.1 P-Rank 公式的收敛性

首先引入一个数学中的收敛性定理.

定理 1. 对于$\forall x_1,x_2$,若$\exists 0<L<1$,使得 $f(x)$满足$\|f(x_1)-f(x_2)\|\leqslant L\|x_1-x_2\|$,则 $x^k=f(x^{k-1})$收敛.

由公式(1′)可知:

$$\begin{aligned}S_{ab}^k &= \frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}S_{mn}^{k-1}\\ &= \sum_{i\to a\in E}\sum_{j\to b\in E}G_{\{(i,j),(a,b)\}}(S_{ij}^{k-1})+\sum_{a\to m\in E}\sum_{b\to n\in E}G'_{\{(m,n),(a,b)\}}(S_{mn}^{k-1}).\end{aligned}$$

要证明 S_{ab}^k 收敛,需证明每个 $G_{\{(i,j),(a,b)\}}(S_{ij}^{k-1})$ 和 $G'_{\{(m,n),(a,b)\}}(S_{mn}^{k-1})$ 都收敛.

令 $G(x)=\dfrac{\lambda C}{|I(a)||I(b)|}x$,其中,$x$ 即为 S_{ij}.则对于任意的 x_1,x_2,有:

$$\begin{aligned}\|G(x_1)-G(x_2)\| &= \left\|\frac{\lambda C}{|I(a)||I(b)|}x_1-\frac{\lambda C}{|I(a)||I(b)|}x_2\right\|\\ &= \frac{\lambda C}{|I(a)||I(b)|}\|x_1-x_2\|.\end{aligned}$$

由于 $0<\lambda,c<1$,$|I(a)|\geqslant 1$,$|I(b)|\geqslant 1$,则 $\dfrac{\lambda C}{|I(a)||I(b)|}\leqslant \lambda C$.而显然,$0<\lambda C<1$.令 $L=\lambda C$,则有:

$$\|G(x_1)-G(x_2)\|\leqslant L\|x_1-x_2\|.$$

根据定理 1 可知 $G(x)$收敛,即每个 $G_{\{(i,j),(a,b)\}}(S_{ij}^{k-1})$ 都收敛.

同理,每个 $G'_{\{(m,n),(a,b)\}}(S_{mn}^{k-1})$ 也都收敛.

因此,S_{ab}^k 收敛.

3.2.2 P-Rank 异步形式的收敛性

根据公式(5)和公式(1′),要证明 S_{ab}^k 收敛,需证明:当 $k\to\infty$时,$\Delta S_{ab}^k\to 0$.

令:

$$\begin{aligned}\Delta S_{ab}^k &= \frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}\Delta S_{ij}^{k-1}+\frac{(1-\lambda)c}{|O(a)||O(b)|}\sum_{a\to m\in E}\sum_{b\to n\in E}\Delta S_{mn}^{k-1}\\ &= S1_{ab}^k+S2_{ab}^k,\end{aligned}$$

则只需证明 $\Delta S1_{ab}^k\to 0,\Delta S2_{ab}^k\to 0$.

先证明 $\Delta S1_{ab}^k$ 收敛于 0.

$$\Delta S1_{ab}^{k}=\frac{\lambda c}{|I(a)||I(b)|}\sum_{i\to a\in E}\sum_{j\to b\in E}\Delta S_{ij}^{k-1}$$

$$=\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{i1j1}^{k-1}+\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{i2j2}^{k-1}+\ldots$$

$$=\frac{\lambda c}{|I(a)||I(b)|}\left(\frac{\lambda c}{|I(i1)||I(j1)|}\sum_{h\to i1\in E}\sum_{k\to j1\in E}\Delta S_{hk}^{k-2}+\frac{(1-\lambda)c}{|O(i1)||O(j1)|}\sum_{i1\to x\in E}\sum_{j1\to y\in E}\Delta S_{xy}^{k-2}\right)+\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{i2j2}^{k-1}+\ldots$$

$$=\frac{(\lambda c)^2}{|I(a)||I(b)||I(i1)||I(j1)|}\sum_{h\to i1\in E}\sum_{k\to j1\in E}\Delta S_{hk}^{k-2}+\frac{\lambda(1-\lambda)c^2}{|I(a)||I(b)||O(i1)||O(j1)|}\sum_{i1\to x\in E}\sum_{j1\to y\in E}\Delta S_{xy}^{k-2}+$$

$$\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{i2j2}^{k-1}+\ldots$$

$$=\ldots$$

$$=\frac{(\lambda c)^k}{|I(a)||I(b)|\ldots|I(o)||I(p)|}\sum_{w\to o\in E}\sum_{v\to p\in E}\Delta S_{wv}^{0}+\ldots+\frac{\lambda(1-\lambda)^{k-1}c^k}{|I(a)||I(b)|\ldots|O(q)||O(v)|}\sum_{q\to c\in E}\sum_{v\to d\in E}\Delta S_{cd}^{0}+$$

$$\frac{\lambda c}{|I(a)||I(b)|}\Delta S_{i2j2}^{k-1}+\ldots$$

由于 $0<\lambda,c<1,|I(i)|\geqslant 1,i\in V$,显然地,当 $k\to\infty$时,$(\lambda c)^k\to 0,\lambda^{k-1}(1-\lambda)c^k\to 0,\ldots,\lambda(1-\lambda)^{k-1}c^k\to 0$.

于是,$\Delta S1_{ab}^{k}\to 0$.同理,$\Delta S2_{ab}^{k}\to 0$. □

3.3 分布式环境中的实现

为了证明AAU算法的精确性和高效性,本文采用了两个平台来实现P-Rank算法——Hadoop平台和Maiter平台.Hadoop 和 Maiter 系统都是开源项目,其中,Hadoop 系统用来实现传统迭代,Maiter 系统用来实现异步累积更新.在 Hadoop 平台上,P-Rank 被以同步方式进行计算,即按照公式(1)的逻辑,先进行 k–1 次迭代,等所有 k–1 迭代中的计算全部结束后,再开始第 k 次迭代.在 Maiter 平台中,P-Rank 以异步方式进行计算,即按照公式(6)的逻辑进行累积计算.

3.3.1 Hadoop 实现

在 Hadoop 系统中,主要通过设计 Map 过程和 Reduce 过程实现 P-Rank 算法.根据公式(1),Map 过程用来计算每个加数,Reduce 过程负责求和运算.

在进行 MapReduce 计算前,需要对初始数据进行处理,生成结点的出入度信息文件,以便迭代过程使用.初始数据格式为

a b

含义为存在有向边 $a\to b$.经过处理后,数据变为更为详细的出入度形式,文件格式为

$\langle a\rangle$; in: $\lvert I(a)\rvert$: $o, p, q,\ldots$; out: $\lvert O(a)\rvert$: $x, y, z,\ldots$

其中,$I(a)$表示结点 a 的入度数,$O(a)$表示结点 a 的出度数,o,p,q 表示 a 的入度结点列表, x,y,z 表示 a 的出度结点列表.

此外,在进行 MapReduce 计算前还要进行数据初始化,目的是为 Map 过程提供初始计算值.根据公式(1),每个结点和其自身的相似度为 1,因此,初始化文件的格式为

a a 1

Map 过程和 Reduce 过程的核心见算法 1.

算法 1. MapReduce 实现 P-Rank 同步计算.

1. Iteration BEGIN
2. t=0;
3. if (t<*Iteration_time*) then
4. Map:
5. 输入:*key*=⟨a,b⟩, *value*=S_{ab};
6. for (a 的每个出度结点 h)
7. for (b 的每个出度结点 k){
8. 计算 $D_{hk}=S_{ab}\times\lambda C/[I(a)\times I(b)]$
9. }
10. for (a 的每个入度结点 x)
11. for (b 的每个入度结点 y){
12. 计算 $D'_{xy}=S_{ab}\times(1-\lambda)C/[O(a)\times O(b)]$
13. }
14. 输出:*key*=⟨h,k⟩,*value*=D_{hk}(或 *key*=⟨x,y⟩,*value*= D'_{xy}).
15. Reduce:
16. 输入:*key*=⟨i,j⟩,*value*=D_{ij};
17. 计算 $S_{ij}=S_{ij}+D_{ij}$
18. 输出:*key*=⟨i,j⟩,*value*=S_{ij}.
19. t++;
20. End of if
21. END of Iteration

3.3.2 Maiter 实现

Maiter 和 Hadoop 类似,也是一个分布式集群,包括一个 master 结点和多个 worker 结点,结点之间通过 MPI 机制进行通信.每个 worker 负责计算一组数据,维护公式(4)中的 Receive 线程和 Update 线程.Maiter 作为 Asynchronous Accumulative Update 算法的实现平台被开发出来,更为详细的介绍见文献[8].

与 Hadoop 一样,在使用 Maiter 进行计算之前,需要对数据进行预处理,也需要根据初始数据生成结点的出入度信息文件.不同的是:Hadoop 处理的是单结点图,采用的是 vertex-centric(以顶点为中心)的计算方式;而 Maiter 处理的数据是结点对图,采用的是 Graph-centric(以图为中心)[9]的计算方式.因此,Maiter 需要的是结点对的出入度信息.

初始数据文件格式为

a b

含义为存在有向边 $a\rightarrow b$.经过预处理后,初始数据变为结点对的出入度形式,文件格式为

⟨a,b⟩; out_a: x:|$I(x)$|,…; out_b: m:|$I(m)$|,…; In_a: y:|$O(y)$|,…; In_b: n:|$O(n)$|,…

文件记录包括 5 部分:结点对⟨a,b⟩,结点 a 的出度信息(*out_a*:后的内容),结点 b 的出度信息(*out_b*:后的内容),结点 a 的入度信息(*In_a*:后的内容),结点 b 的入度信息(*In_b*:后的内容).

其中,x 表示 a 的一个出度结点,$|I(x)|$表示结点 x 的入度数;m 表示 b 的一个出度结点,$|I(m)|$表示结点 m 的入度数;y 表示 a 的一个入度结点,$|O(y)|$表示结点 y 的出度数;n 表示 b 的一个入度结点,$|O(n)|$表示结点 n 的出度数.

相比于 Hadoop 只有每个结点的出入度信息,Maiter 还增加了每个出入度结点的出入度信息.从表面上看,

Maiter 输入信息更多更复杂了.事实上,其进行数据预处理的时间确实比 hadoop 略长,但是相比于整个计算过程,这些时间是非常小的,可以忽略不计.因此在接下来的实验部分中,数据预处理的时间被略去了.

Maiter 的数据初始化和 Hadoop 类似,将每个结点和其自身的相似度初始化为 1,其他结点对初始化为 0.这种初始化方式的原理很简单,结点对之间的相似度是通过有向边传递的,而最初的传递源头就是每个结点自身构成的结点对.因此,Maiter 的初始化文件格式为

a	*a*	1
a	*b*	0

算法 2 写出了使用 AAU 实现 P-Rank 算法的代码主体.需要说明的是:AAU 算法已经没有了迭代的概念,只在计算过程中定期检查计算终止条件,若条件满足,则停止计算.终止条件在这里采用的是计算ΔS 总和的方式,若ΔS 总和足够小,说明 S 已经不再变化,那么可以认为 S 已经足够收敛了,于是计算停止.

算法 2. AAU 实现 P-Rank 异步计算.

```
1.  Computation BEGIN
2.    read data,将结点对⟨a,b⟩的出入度信息存入 statetable;
3.    初始化 S_ab 和ΔS_ab;
4.    计算 delta 的总和 sum_of_delta;
5.    if (sum_of_delta>阈值) then
6.      计算ΔS_ab=ΔS_ab+ΔD_ab+ ΔD'_ab ;        //执行 receive
7.      if (time_interval==VALUE) then     //一定时间后执行 update
8.        计算 S_ab=S_ab+ΔS_ab;
9.          for (a 的每个出度结点 h)
10.           for (b 的每个出度结点 k){
11.             计算ΔD_hk=ΔS_ab×λC/[I(h)×I(k)];
12.             将ΔD_hk 发送给⟨h,k⟩所在的 worker;
13.           }
14.         for (a 的每个入度结点 x)
15.           for (b 的每个入度结点 y){
16.             计算 ΔD'_xy =ΔS_ab×(1−λ)C/[O(x)×O(y)];
17.             将 ΔD'_xy 发送给⟨x,y⟩所在的 worker;
18.           }
19.       ΔS_ab=0;
20.     End of if
21.   else
22.     迭代终止
23.   End of if
24. END of Computation
```

4 实验及结果分析

4.1 实验设置

本文的数据集采用真实数据集 p2p-Gnuttella08,来源于http://snap.stanford.edu/data/p2p-Gnutella08.html.这个数据集是一个对等网络文件共享数据集,数据集中的结点表示主机,有向边表示主机之间的共享关系.该数据

集包含 6 301 个结点,20 777 条边.文件大小为 1 000M,实际需要计算的数据量为 18 170 868 条 P-Rank 记录.

实验的运行环境为 Intel 酷睿 2 双核 CPU,2G 内存和 Ubuntu 12.04 操作系统.Hadoop 版本为 1.0.0,Maiter 版本为 0.1.Hadoop 集群和 Maiter 集群均采用 1 个 master 结点和 3 个 worker 结点.

本节设置了 2 组实验:第 1 组实验用来比较 P-Rank 同步计算方式和异步计算方式的总体运行时间;第 2 组实验用来研究异步计算方式的性能,即随着 Worker 的增加,异步计算时间的变化趋势.这两组实验都是讨论计算速度.

除了速度之外,精确度也是必须考量的因素.因此,本文在第 3.2 节中先讨论了 P-Rank 异步方法的精确度,然后在第 3.3 节中讨论了 P-Rank 异步方法的速度因素.

4.2 AAU方法精确度分析

本节从实验的角度证明了 AAU 算法的精确性.表 1 是传统迭代方法 MapReduce 和异步累积更新方法 AAU 对一个 P-Rank 例子的计算结果比较,均保留了小数点后 6 位.“MapReduce 结果”和“AAU 结果”两列表示结点对的相似度值.其中,MapReduce 结果是迭代 20 次后的结果.事实上,在进行 10 次迭代后计算结果已经足够收敛,但为了使结果更加精确,迭代了 20 次.

Table 1 Comparison of the P-Rank results computed by MapReduce and AAU

表 1 MapReduce 和 AAU 计算结果的比较

结点对	MapReduce 结果	AAU 结果	二者差值的绝对值
1,4	0.132 332	0.132 336	0.000 004
1,5	0.033 877	0.033 877	0
2,4	0.413 551	0.413 551	0
2,5	0.105 865	0.105 865	0
3,4	0.042 346	0.042 346	0
3,5	0.330 829	0.330 841	0.000 012
4,5	0.088 218	0.088 217	0.000 001

从表 1 可以看出:使用异步方法 AAU 对 P-Rank 的计算结果几乎和 MapReduce 结果完全一致,二者差值在小数点后第 6 位才出现,有超过 50%的计算结果是一样的.因此,我们有理由相信 P-Rank 异步方法的精确性.

4.3 AAU方法计算速度分析

本节中,我们将详细讨论数据量和集群规模对 P-Rank 异步方法计算速度的影响.

4.3.1 总体计算时间对比

为了更加清晰地分析异步方法 AAU 和 MapReduce 计算 P-Rank 的时间,实验中把原始数据集按有向边数分成 500,1 000,3 000,5 000 共 4 个级别,这 4 个级别的相关数据可见表 2.事实上,在实验中,和计算时间相关的数据是表 4 中“实际计算数据条数”一列,这一列表示实际需要计算相似度的结点对的数量.

需要特别说明的是 MapReduce 计算时间的统计:MapReduce 迭代次数越多,结果越收敛;但与此同时,计算时间也逐渐延长.实验中,经过不同迭代次数的比较,我们得出结论:迭代 10 次后结果变化越来越小,即迭代 10 次的结果已经足够接近最终的收敛值.因此,为了客观评价 MapReduce 和 AAU 的计算时间,表 2 中的 MapReduce 计算时间以迭代 10 次的时间作为基准.

Table 2 Impacts of AAU computation time by different data scales

表 2 数据量对计算时间的影响

数据	边数	实际计算数据条数	实际数据规模(MB)	MR 时间(s)	AAU 时间(s)
1	500	8 069	0.2	360	120
2	1 000	29 697	0.8	425	230
3	3 000	228 715	6.9	840	443
4	5 000	1 266 876	40	1 500	929

从表 2 可以看出:无论数据量大小,异步方法 AAU 对 P-Rank 的计算时间均少于同步方法 MapReduce:对于数据集 2~数据集 4,异步方法的计算时间大约只有同步方法的一半;对于更小的数据集 1,异步计算节省了三分

之二的时间.

图 1 和图 2 可以更加直观地感受到 AAU 的速度优势.从图 2 我们可以得出另一个信息,即随着数据量的增长,异步方法的计算时间增长也略缓于同步方法.

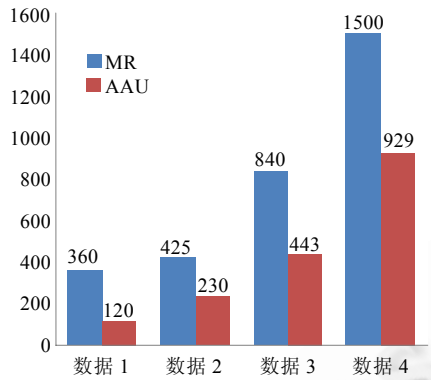

Fig.1 Comparison of computation time by MapReduce and AAU --column chart (s)

图 1 AAU 和 MapReduce 计算时间对比(s)

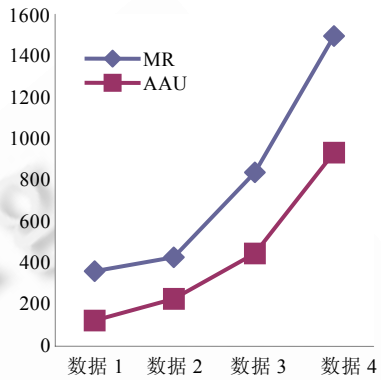

Fig.2 Comparison of computation time by MapReduce and AAU --line chart (s)

图 2 AAU 和 MapReduce 计算时间对比(s)

通过以上分析可以得出结论:P-Rank 异步方法的计算速度优于同步方法,且数据量越大,这种优势越明显.

4.3.2 集群规模对计算时间的影响

为了研究集群规模对 AAU 计算时间的影响,本实验将 Maiter 集群的 Worker 结点数分为 1,2,3 共 3 个类别.并分别用这 3 个类别运行了表 2 中的数据,统计了不同结点计算这 4 个数据集的时间.统计结果见表 3.

Table 3 Impacts of computation time by different cluster scales

表 3 集群规模对 AAU 计算时间的影响

数据	边数	实际计算数据条数	Worker 结点数量/AAU 时间(s)		
			1 台	2 台	3 台
1	500	8 069	202	151	120
2	1 000	29 697	366	284	230
3	3 000	228 715	643	529	443
4	5 000	1 266 876	1 283	1 123	929

图 3(a)和图 3(b)给出了计算时间随集群数量变化的柱形图和曲线图,从中可以更加直观地看出:无论数据集的大小,随着 worker 结点数量的增加,异步方法 AAU 处理 P-Rank 的计算时间均处于下降趋势,这意味着计算速度在逐步提高.从图 3(b)也可以看出:处理的数据量越大,计算时间的曲线越陡,说明下降趋势越明显.通过以上分析,我们可以得出结论:集群规模越大,计算速度越快;且数据量越大,这种速度优势越明显.

5 总结和未来工作

本文首先分析了传统迭代算法的缺点,即迭代过程中由于需要同步,处理器不可避免地需要等待时间.为了解决这一问题,本文引入了异步累积更新算法,实现了异步计算,节省了处理器的等待时间,提高了计算效率.阐述了如何将 P-Rank 应用到异步累积更新算法上,给出了 P-Rank 的异步形式,实现 P-Rank 异步形式的平台——Maiter 平台,并给出了在此平台上进行 P-Rank 异步计算的算法.同时,为了验证 AAU 算法的精确性和高效性,本文也简单介绍了 P-Rank 在 Hadoop 平台上的实现,用于和 AAU 进行比较.最后,用两组实验证明了异步累积更新算法 AAU 的计算速度优于传统同步迭代算法:集群规模越大,计算速度越快;且数据量越大,这种速度优势越明显.

由于实验环境限制,本文的分布式集群只使用了 3 台机器,由于机器性能有限,在处理更大规模数据方面仍

有所不足.因此,更大规模的数据处理.这也是我们未来研究工作的重点.

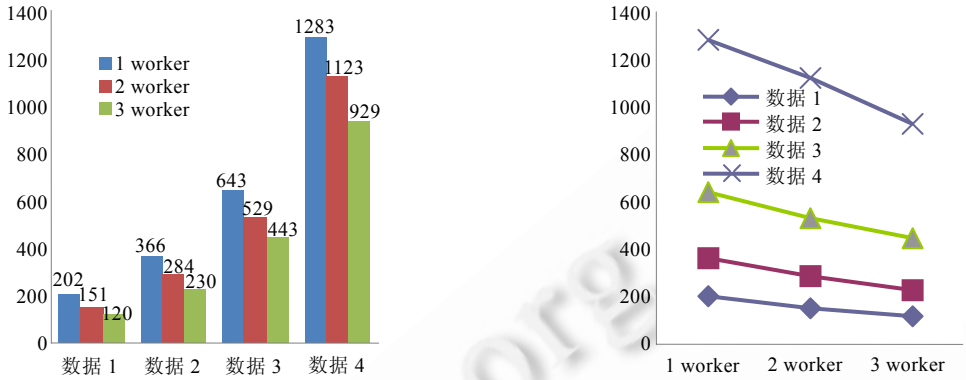

Fig.3 Comparison of AAU computation time with different worker numbers --column chart (s)

图 3 集群规模对 AAU 计算时间的影响(单位:s)

References:

[1] Bu Y, B. Balazinska HM, Ernst DM. Haloop: Efficient iterative data processing on large clusters. VLDB Endowment, 2010,3(1-2): 285−296. [doi: 10.14778/1920841.1920843]

[2] Ching A, Kunz C. Giraph: Large-scale graph processing infrastructure on Hadoop. Hadoop Summit, 2011,6(29):2011.

[3] Zhang Y, Gao Q, Gao L, Wang CR. Accelerate large-scale iterative computation through asynchronous accumulative updates. In: Proc. of the 3rd Workshop on Scientific Cloud Computing Date. ACM Press, 2012. 13−22. [doi: 10.1145/2287036.2287041]

[4] Jeh G, Widom J. SimRank: A measure of structural-context similarity. In: Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2002. 538−543. [doi: 10.1145/775047.775126]

[5] Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. Journal of the American Society for Information Science and Technology, 2007,58(7):1019−1031. [doi: 10.1002/asi.20591]

[6] Abbassi Z, Mirrokni VS. A recommender system based on local random walks and spectral methods. In: Proc. of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis. ACM Press, 2007. 102−108. [doi: 10.1145/1348549.1348561]

[7] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM Press, 2004,51(1): 107−113. [doi: 10.1145/1327452.1327492]

[8] Zhang Y, Gao Q, Gao L, Wang C. Maiter: A message-passing distributed framework for accumulative iterative computation. Technical Report, 2012. http://rio.ecs.umass.edu/~yzhang/maiter-full.pdf

[9] Tian YY, Balmin A, Corsten SA, Tatikonda S, Mcpherson J. From "think like a vertex" to "think like a graph". Proc. of the VLDB Endowment, 2013,7(3).

[10] Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: A system for large-scale graph processing. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2010. 135−146. [doi: 10.1145/1807167.1807184]

[11] Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein JM. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012,5(8):716−727. [doi: 10.14778/2212351.2212354]

[12] Sun LW, Cheng R, Li X, Cheung DW, Han JW. On link-based similarity join. Proc. of the VLDB Endowment, 2011,4(11).

[13] Zheng WG, Zou L, Feng YS, Chen L, Zhao DY. Efficient SimRank-based similarity join over large graphs. Proc. of the VLDB Endowment, 2013,6(7):493−504. [doi: 10.14778/2536349.2536350]

[14] Fujiwara Y, Nakatsuji M, Shiokawa H, Onizuka M. Efficient search algorithm for SimRank. In: Proc. of the 2013 IEEE 29th Int'l Conf. on Data Engineering (ICDE). IEEE, 2013. 589−600. [doi: 10.1109/ICDE.2013.6544858]

[15] Yu W, Lin X, Zhang W. Towards efficient SimRank computation on large networks. In: Proc. of the 2013 IEEE 29th Int'l Conf. on Data Engineering (ICDE). IEEE, 2013. 601−612. [doi: 10.1109/ICDE.2013.6544859]

[16] Li P, Liu H, Yu J X, He J, Du XY. Fast single-pair SimRank computation. In: Proc. of the SDM. 2010. 571−582.

[17] Lizorkin D, Velikhov P, Grinev M, Turdakov D. Accuracy estimate and optimization techniques forSimRank computation. Proc. of the VLDB Endowment, 2008,1(1):422−433. [doi: 10.14778/1453856.1453904]

[18] Zhao P, Han J, Sun Y. P-Rank: A comprehensive structural similarity measure over information networks. In: Proc. of the 18th ACM Conf. on Information and Knowledge Management. ACM Press, 2009. 553−562. [doi: 10.1145/1645953.1646025]

[19] Lin Z, Lyu MR, King I. Matchsim: A novel neighbor-based similarity measure with maximum neighborhood matching. In: Proc. of the 18th ACM Conf. on Information and Knowledge Management. ACM Press, 2009. 1613−1616. [doi: 10.1145/1645953.1646185]

[20] Xi W, Fox EA, Fan W, Zhang BY, Chen Z, Yan J, Zhuang D. Simfusion: Measuring similarity using unified relationship matrix. In: Proc. of the 28th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM Press, 2005. 130−137. [doi: 10.1145/1076034.1076059]

[21] Sun Y, Han J, Yan X, Yu PS, Wu TY. Pathsim: Meta path-based top-*k* similarity search in heterogeneous information networks. In: Proc. of the VLDB 2011. 2011.

[22] Yuan F, Chang K, Chen-Chuan W, Lauw H. RoundTripRank: Graph-Based proximity with importance and specificity. In: Proc. of the ICED. 2013. 613−624.

[23] Zhang YF, Gao QX, Gao LX, Wang CR. Imapreduce: A distributed computing framework for iterative computation. Journal of Grid Computing, 2012,10(1):47−68. [doi: 10.1007/s10723-012-9204-9]

[24] Zhang Y, Gao Q, Gao L, Wang CR. Priter: A distributed framework for prioritized iterative computations. In: Proc. of the 2nd ACM Symp. on Cloud Computing. ACM Press, 2011. 13. [doi: 10.1145/2038916.2038929]

王旭丛(1989－),女,河北石家庄人,硕士生,主要研究领域为大数据,图挖掘算法.
E-mail: wangpu9003@126.com

李翠平(1971－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据仓库和数据挖掘,信息网络分析,流数据管理.
E-mail: licuiping@ruc.edu.cn

陈红(1965－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能数据库系统,数据仓库与数据挖掘,数据流管理,无线传感器网络中的数据管理.
E-mail: chong@ruc.edu.cn