Asyn-SimRank:一种异步执行的大规模 SimRank 算法

王春磊[1)], 张岩峰[2)], 赵长宽[2)] 于戈[1)], Lixin Gao[3)]

[1)](东北大学计算机软件与理论研究所，沈阳 中国 100819)

[2)](东北大学计算中心，沈阳 中国 100819)

[3)](University of Massachusetts，Amherst 美国 01002)

摘要 SimRank 算法利用网络结构来评估网络中任意两点的相似性，它被广泛应用于社交网络和链接预测等诸多领域中。近年来，随着大数据技术的发展，SimRank 算法处理的数据不断增大，人们利用 MapReduce 等分布式计算模型设计实现分布式的大规模 SimRank 算法来适应大数据处理的需求。但是，由于 SimRank 算法包含开销较大的迭代过程，每次迭代之后都需要一个全局同步，且每次迭代的计算复杂度高、通信量大，使得 SimRank 算法的分布式实现效率很低。本文提出 Asyn-SimRank 算法，该算法异步执行 SimRank 的核心迭代过程，避免了大规模分布式计算中的大量同步开销。并且 Asyn-SimRank 算法采用累积-迭代的方式完成迭代计算，可以有效降低计算量并减少通信开销。本文严格证明了 Asyn-SimRank 算法的正确性和收敛性，分析了其优点，并且在支持异步迭代的分布式框架 Maiter 上实现了 Asyn-SimRank 算法。实验结果显示 Asyn-SimRank 算法大大加速了算法收敛，相比较于传统的 Hadoop 上实现的 SimRank 算法，Asyn-SimRank 最高可加速 130 倍。

关键词 异步计算；迭代计算；Asyn-SimRank；相似度；大数据；MapReduce；

中图法分类号 TP391

Asyn-SimRank: An Asynchronous Large-Scale SimRank Algorithm

Wang Chunlei[1)] , Zhang Yanfeng[2)] , Zhao Changkuan[2)],Yu Ge[1)] , GAO Lixin[3)]

[1)] (Northeastern University Institute of Computer Software and Theory, Shenyang China 100819)

[2)] (Northeastern University Computer Center, Shenyang China 100819)

[3)](University of Massachusetts，Amherst USA 01002)

Abstract SimRank algorithm, which exploits network structure to measure the similarity between node pairs, has been widely used in many areas, such as online social networks and link prediction. In recent years, with the development of big data, the input data set of the SimRank algorithm is constantly increasing. People are utilizing distributed computing models, such as MapReduce, to design large-scale SimRank algorithm for solving the big data problems. However, since SimRank algorithm contains a high-cost iterative process with synchronization barriers between iterations and the computational complexity is high in each iteration, the large-scale SimRank computation does not result in satisfactory performance. In this paper, we propose

收稿日期：2013 年 7 月 31 日；最终修改稿收到日期：2013 年 9 月 30 日. 本课题得到国家自然科学基金（61300023）；中央高校基本科研业务费(N120416001, N120816001)；教育部一中国移动科研基金 (MCM20122051)；教育部-英特尔信息技术专项科研基金 (MOE-INTEL-2012-06)资助.王春磊，男，1989 年生，硕士，主要研究领域为大数据处理，E-mail: wangchunlei.neu@foxmail.com. 张岩峰，男，1982 年生，博士，副教授，主要研究领域为大数据处理，E-mail: zhangyf@cc.neu.edu.cn. 赵长宽，男，硕士，讲师，主要研究领域为云计算,E-mail:zck@cc.neu.edu.cn.于戈，男，1962 年生，博士，教授，主要研究领域为数据库，云计算，E-mail: yuge@mail.neu.edu.cn. Lixin Gao，女，1968 年生，博士，教授，主要研究领域为网络路由，云计算，E-mail: lgao@ecs.umass.edu.
第一作者手机号码：15840097038，E-mail: wangchunlei.neu@foxmail.com.

Asyn-SimRank, which asynchronously executes the core iterative process in order to avoid the high-cost synchronization barriers in large-scale distributed environments. Moreover, Asyn-SimRank adopts the cumulate-iterate method, which can effectively reduce the amount of computation and communication. We prove the accuracy and the convergence of the Asyn-SimRank algorithm and analyze its advantages. We then implement Asyn-SimRank on Maiter, which is a distributed framework supporting asynchronous iteration. Our results show that the large-scale Asyn-SimRank significantly accelerates the convergence. Specially, Asyn-SimRank on Maiter is 130x faster than the traditional SimRank implementation on Hadoop.

Keywords asynchronous computation; iterative computation; Asyn-SimRank; similarity; big data; MapReduce;

1. 引言

随着数据采集技术和数据存储等技术的发展，各类应用所处理的数据量不断增大，人们应用数据挖掘和机器学习等算法对这些庞大的数据进行处理和分析，取得有意义的分析结果。由于所涉及的数据量巨大，人们通常利用云计算环境来进行大数据的处理。如何有效利用云环境下的分布式资源设计高效的大规模数据挖掘和机器学习算法，成为当今大数据领域的热点研究问题。

SimRank[1]算法利用网络结构评估网络中两结点的相似度。它广泛应用在社交网络、引用关系网络、链接预测等诸多主要领域中。例如，SimRank 算法常被用于评估商品购买关系中产品之间的相似度或朋友关系中朋友之间的相似度，根据相似度来进行产品推荐和好友推荐。近年来随着推荐系统的流行，SimRank 算法的重要性也更加凸显出来。

当采用 SimRank 算法处理大数据时，需要设计大规模的 SimRank 算法来支持分布式处理。Hadoop[2]框架和 MapReduce[3]模型是目前最流行的大规模数据处理模型和平台。在 Hadoop 框架上实现 SimRank 算法时，虽然实现比较简单，但每个 Mapper 任务都要存储整张图，空间复杂度过高，并且产生的大量中间结果也会影响 SimRank 算法的运算效率。另外，由于传统的 SimRank 算法是一个同步迭代算法，多次迭代的多个同步过程在大规模异构分布式环境下将造成大量的同步开销，导致 SimRank 算法在分布式环境下或云环境下不能高效地实现。虽然目前有很多新型分布式处理框架支持高效的迭代计算，但是 SimRank 算法只具备同步计算形式，框架本身的改进对 SimRank 算法的效率提升不大。

本文提出的 Asyn-SimRank 算法摒弃传统 SimRank 算法的同步迭代方法，采用累积迭代差值方式。它仅计算各次迭代中的差值，且仅累积这些差值信息，取得最终的收敛结果。该方法避免计算没有变化的结点信息，可以有效减小迭代计算过程中的计算量和通信量。另外，根据累积的性质可以异步执行 Asyn-SimRank 算法。本文从理论上证明了该异步算法的收敛性和正确性，并且分析了其效率优势。

为了支持大规模分布式计算，本文实现了异步执行的大规模 Asyn-SimRank 算法，并部署到分布式环境上进行性能测试，并且与传统的 Hadoop 上实现的 SimRank 算法和近期的其它相关工作进行了比较。实验结果显示了 Asyn-SimRank 算法优异的性能。

2. 相关工作

SimRank 算法不针对特定的图数据，也不局限于特定的应用范围，具有通用性强的特点，因而该算法被广泛的应用于各个领域。随着对 SimRank 算法应用需求的不断提升，很多的工作都集中于相似度的计算性能。这些工作主要有三个方向：改进分布式框架来提高迭代速度，进而提高 SimRank 运行速度、提出优化策略来降低 SimRank 算法时间复杂度、提出新的算法来近似计算相似度。

随着分布式计算技术的不断发展，很多可以高效的支持迭代算法的分布式框架被提出。Spark[4]通过将迭代中反复用到的数据存在内存，避免反复的从磁盘中读取数据来加快迭代的速度。Haloop[5]通过循环调度和缓存机制，节省每次迭代开始时串行任务部署开销来高效支持迭代。还有如 Pregel[6]、Twister [7]、iMapReduce[8]、Priter[9]等通过支持优先级迭代、异步迭代等措施来提高效率。在

这样的平台上实现 SimRank 算法，可以一定程度的提高运算速度。但是这些框架采用的仍然是传统的迭代方式，不能对 SimRank 算法的计算量和通信量进行大幅度优化。另外 SimRank 算法在这些框架上实现比较复杂，有时甚至不能实现。所以对 SimRank 算法效率的提升是有限的。

在 SimRank 算法的改进方面，Delta-SimRank[10]算法采用了的累积差值的迭代计算方式。这一计算方式可以利用收敛的图顶点迭代差值为零的特性，有效的避免已收敛图顶点再度参加运算，成功的减小了计算开销、降低了通信量，但是该算法每次迭代计算节点仍然要存储整张图和所有顶点的入度才能完成连续的迭代，导致迭代过程中计算节点的 IO 过大，影响计算效率。并且该算法采用的仍然是同步的迭代方式，严格的规定累积两次迭代的差值，不能支持异步计算。同时也不能支持优先级迭代来优先地计算慢收敛的图顶点。

有相关工作提出的一些技术、策略来优化 SimRank 算法，降低其计算复杂度。文献[11,12]提出了局部和函数，将 SimRank 算法的最坏情况计算复杂度由 O(n^4)降为 O(n^3)。文献[13]给出了衡量相似度值精度的方法，并提出了三种对 SimRank 算法的优化策略。还有的工作如[14]通过利用图形处理器 GPU 以及多核 CPU 等硬件设备来并行的运算 SimRank 算法，并给出了并行算法。

在其他的相似度计算算法方面，SimFusion 算法[15]是另一种计算相似度值的算法，它的基本原理与 SimRank 算法类似，与 SimRank 不同的是，它能够计算异构数据源之间的关系，并支持异构数据间的相似度值的计算。相对于 SimRank 算法，SimFusion 算法的计算复杂度较低。但是仍然属于高计算复杂度的算法。Fingerprint-SimRank [16]随机的游走图中的路径，并提前计算这些路径上的值，一定程度上的加快了计算速度，但是也增大了空间复杂度。

与上文一些优化策略不同的是，本文采用异步累积迭代以及优先级迭代的策略来改进 SimRank 算法，有效的加快 SimRank 算法的计算速度，并降低了计算量和通信量，显著地解决 SimRank 算法计算量高、通信量大、运行速度慢等诸多问题。为高效的计算 SimRank 算法提供了有力地途径。

3. SimRank 算法原理与 Hadoop 实现

3.1 原理及计算公式

SimRank 算法的计算思想来源于 PageRank[16]算法，主要根据对象出现的结构环境来衡量对象间的相似性。算法的基本思想是：如果两个对象与相同的对象或相似的对象有关系，那么他们是相似的。如果将对象作为一个顶点，对象间的关系当作一条边，就可以得到一张图 G=(V，E)，V 为对象的集合，E 为关系的集合。SimRank 算法基于图 G 进行相似度计算。

如果将对象 a 与对象 b 之间的相似度记为 $S(a,b)$，SimRank 算法的迭代计算公式为：

$$S^{k+1}(a,b)=\begin{cases}1 & a=b\\ \frac{C}{|I(a)||I(b)|}\sum_{c\in I(a),d\in I(b)}S^{k}(c,d) & a\neq b\end{cases} \quad (1)$$

其中$I(a)$和$I(b)$分别为顶点a和顶点b的指入顶点集合；$S(a,b)$的取值范围为[0,1]；系数C是一个阻尼系数，其取值范围为(0,1)。$S^0(a,b)$的初始值为，当$a=b$时，$S^0(a,b)=1$；当$a\neq b$时，$S^0(a,b)=0$。由(1)式经多次迭代，就可以得到最终的顶点间的相似度。

为了明确 SimRank 算法的计算过程，可以构造一张顶点对图 $G^2(V^2, E^2)$ [1] 来直观地观察 SimRank 算法在计算时数据是如何流动的。G^2 图的构造方法为：如果 $a\in V$ 且 $b\in V$，则顶点对 $(a,b)\in V^2$。如果边 $a\to c\in$ E 且 $b\to d\in E$，则边 $(a,b)\to(c,d)\in E^2$。以图 1 中的(a)图为例，由(a)图构造出来的G^2图如(b)所示。

一些顶点对并没有在G^2图中出现，就说明该顶点对是孤立的，并不对其他顶点对的值造成影响。从G^2图中可以观察到，SimRank 算法在计算的过程中数据是从一个顶点对流向另一个顶点对，即两顶点的相似度依赖于各自相邻顶点之间的相似度，这与 SimRank 算法的计算原理是一致的。

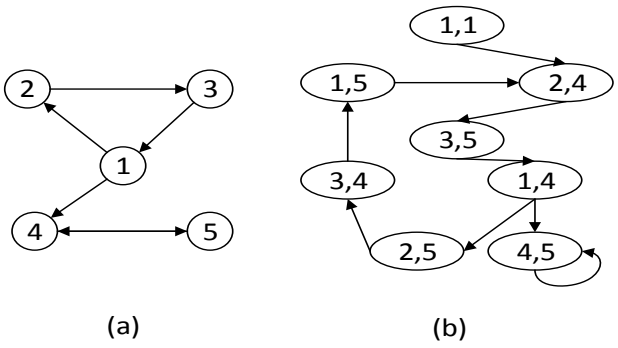

图 1 G 图到G^2图的对应示例

G^2图成功地将 G 图中两顶点之间的相似度的计算转变为G^2图中顶点对状态值的计算。它可以将

以边为中心的 SimRank 算法转变为以点为中心，以便于其在以点为中心的框架上实现，这在后续章节将详细介绍。

3.2 Hadoop 框架上的实现

SimRank 算法在分布式框架 Hadoop 上的实现形式如算法 1 所示。

算法 1 Hadoop 上实现的 SimRank 算法

输入：图 G，初始值 S^0

输出：顶点对(*a*,*b*) 相似度值

1. 迭代开始：
2. FOR t = 0 TO T-1
3. *Map* 输入：{key=(*a*,*b*) ,value= $S^t(a,b)$}
4. 分别找出 *a*，*b* 的指出邻接点集合 O(a)O(b)
5. 对每一对 c∈O(a)，d∈O(b)
6. 发送{key=(*c*,*d*)，value=$S^t(a,b)$}
7. END *Map*
8. *Reduce* 输入：{key = (*c*,*d*)，value = vs[ ]}
9. IF c = d
10. $S^{t+1}(c,d)$ = 1
11. ELSE
12. $S^{t+1}(c,d) = \frac{C}{len(vs)} sum(\text{vs})$
13. 输出 (c,d) $S^{t+1}(c,d)$
14. END *Reduce*
15. END FOR
16. 迭代结束。

MapReduce 模型将每次迭代作为一个 Job 来完成，每个 Job 中 *Map* 阶段负责将顶点对的相似度值发给邻接顶点对，*Reduce* 阶段则负责将其他顶点对发来的相似度值累加起来作为本顶点对的相似度值并将结果输出，下一个 Job 用这些输出结果作为输入信息进行下一次迭代。

基于 MapReduce 模型实现的 SimRank 算法存在如下若干问题：

（1）存储复杂度高。在 *Map* 阶段，由于收到顶点对是任意的，*Map* 要获得每个顶点对中顶点的邻接表，并且保证下一次迭代时仍然可以用到这些信息，每个 *Map* 任务只能存储整张图。

（2）计算复杂度高。由于 SimRank 算法计算的是图中任意两个顶点的相似度。所以该算法的实际计算的组合顶点对(a,b)的数量级会达到n^2，其中 n 为图中顶点的个数。假设每个顶点的平均邻接顶点数为 p，每个顶点对会给邻接顶点对发送信息，平均发送次数为 p^2，那么就可得出，SimRank 算法一次迭代的时间复杂度为 $O(n^2p^2)$。由于处理的数据是海量数据，n 通常很大，所以实际的计算量非常大。

（3）通信复杂度高。SimRank 是高通信的算法，每次迭代计算任意两点的相似度，在 Map 阶段将发送 p^2 个消息，经过 Shuffle 阶段传递给对应的 Reduce，所以每次迭代的通信量是 $O(n^2p^2)$。

（4）同步代价高。每次迭代对应的每个 MapReduce 作业存在两次全局同步，即 Shuffle 过程的同步和作业之间的同步。全局同步要求所有任务结束后才可以进行下一步的操作，这样就使速度较快的计算节点等待计算较慢的计算节点，造成不必要的全局等待开销。

总之，SimRank 算法在 Hadoop 上的实现，虽然实现了相似度的计算，却并不能保证计算的高效性，需要很长的运行时间并消耗大量的计算资源。接下来，本文将提出一种高效的 SimRank 改进算法，它可以避免 Hadoop 实现 SimRank 算法的若干问题。

4. Asyn-SimRank 算法

本章提出可以异步执行的 Asyn-SimRank 算法来加速大规模的 SimRank 相似度计算。首先，从理论上推导出 Asyn-SimRank 算法的同步迭代形式；然后，给出可以异步执行的 Asyn-SimRank 算法；最后，证明 Asyn-SimRank 算法的正确性和收敛性。

4.1 同步累积-迭代形式

传统的 SimRank 算法不能支持异步计算，它的迭代形式要求第 k 次迭代的计算完全依赖于第 *k-1* 次迭代的结果，这种依赖性导致每次迭代必须进行同步来保证结果的正确性。

可以对 SimRank 算法做一些变换，计算两次迭代的差值，然后将这些差值累积起来，来摆脱两次迭代的完全依赖。对(1)式的变换如下：当$a \neq b$时

令：$\Delta S^{k+1}(a,b) = S^{k+1}(a,b) - S^k(a,b)$

$$= \frac{C}{|I(c)||I(b)|}\sum_{c\in I(c),d\in I(b)} S^k(c,d) \; - \; \frac{C}{|I(c)|I(b)|}\sum_{c\in I(c),d\in I(b)} S^k(c,d)$$

$$= \frac{C}{|I(c)|I(b)|}\sum_{c\in I(c),d\in I(b)} [S^k(c,d) - S^{k-1}(c,d)]$$

$$= \frac{C}{|I(c)||I(b)|}\sum_{C\in I(c),d\in I(b)}\Delta S^{k}(c,d)$$

当a = b时，则有：

$$\Delta S^{k+1}(a,b) = S^{k+1}(a,b) - S^{k}(a,b) = 1 - 1 = 0$$

从前文推导过程可以看出，迭代的差值$\Delta S^{k+1}(a,b)$也是可以迭代地进行计算的。那么如果先迭代计算差值，然后再将这些差值累积起来：

$$\begin{cases} S^{k}(a,b) = \Delta S^{k}(a,b) + S^{k-1}(a,b) \\ \Delta S^{k+1}(a,b) = \dfrac{C}{|I(a)||I(b)|}\displaystyle\sum_{C\in I(a),d\in I(b)}\Delta S^{k}(c,d) \end{cases} \quad (2)$$

当a = b时$\Delta S^{k}(a,b) = 0$。

公式(2)[10]与 Delta-SimRak 算法的计算公式是一样的，这是推导 Asyn-SimRank 必经的过程。但是在后续的推导过程中，在累积差值的处理方式上会完全的不同。本文会在 4.2 节后半部分详细的说明 Asyn-SimRank 算法与 Delta-SimRnak 算法的不同。

由(2)可以看出，SimRank 的计算过程被分为两步，（1）累积：将迭代差值累积起来；（2）迭代：迭代的计算两次迭代的差值。这种迭代过程就是本文所说的累积-迭代方式。

可以用这两步操作来计算相似度，代替传统的迭代方式。两步操作在分布式环境下的实现如下所示：

累积：$\begin{cases} \text{接收所有从其他顶点发来的}\Delta S^{k}(a,b) \\ S^{k}(a,b) = \Delta S^{k}(a,b) + S^{k-1}(a,b) \end{cases}$

迭代：$\begin{cases} \Delta S^{k+1}(\mathrm{c},d) = \dfrac{C}{|I(c)||I(d)}\Delta S^{k}(a,b) \\ \text{将}\Delta S^{k+1}(c,d)\text{发给顶点}(c,d) \\ \text{其中}c\in I(a), d\in I(b) \end{cases}$

但是在按以上过程实现的时候就会发现一个问题，在迭代操作中，当前顶点对(*a*,*b*)在运算的时候却需要用到*|I(c)||I(d)|*，其中$c\in I(a), d\in I(b)$，这就意味着当前顶点对(*a*,*b*)，需要 *a* 的邻接表和 *b* 的邻接表中顶点的入度。获得这样的信息，就需要数据按着逆邻接表的方向流动，由顶点对(*c*,*d*)传送给(*a*,*b*)，每个顶点只知道自己邻接表的前提下是不可能的。虽然分布式环境下可以将这些信息保存在计算节点的本地磁盘上，在任务开始时装进内存，但是每个计算节点的空间开销会是 *O(n)*，*n* 为输入图的顶点数。

为了解决此问题，可以让顶点对(*a*,*b*)只完成部分能够完成的运算，然后将没有计算完全的值发给(*c*,*d*)来完成后续运算。这种方法需要对公式(2)做以下变换：首先，将：

$$\Delta S^{k+1}(a,b) = \frac{C}{|I(c)||I(b)|}\sum_{C\in I(c),d\in I(b)}\Delta S^{k}(c,d)$$

两边同时乘以 $\frac{|I(a)||I(b)|}{C}$ 则得到下式：

$$\frac{|I(a)||I(b)|}{C}\ \Delta S^{k+1}(a,b) = \sum_{C\in I(a),d\in I(b)}\Delta S^{k}(c,d)$$

令：

$$\Delta P^{k+1}(a,b) = \frac{|I(a)||I(b)|}{C}\ \Delta S^{k+1}(a,b)$$

将$\Delta P^{k+1}(a,b)$带入到公式(2)就有：

$$\Delta P^{k+1}(a,b) = \sum_{c\in I(a),d\in I(b)}\frac{\Delta P^{k}(c,d)}{|I(c)||I(d)} * C$$

综上，可得公式(3)：

$$\begin{cases} S^{k}(a,b) = S^{k-1}(a,b) + \dfrac{\Delta P^{k}(a,b)}{|I(a)||I(b)} * C \\ \Delta P^{k+1}(a,b) = \displaystyle\sum_{c\in I(a),d\in I(b)}\frac{\Delta P^{k}(c,d)}{|I(c)||I(d)} * C \end{cases} \quad (3)$$

在分布式环境下，公式(3)的累积-迭代操作的实现形式如下所示：

累积：$\begin{cases} \text{接收所有从其他节点发来的}\Delta P^{k}(a,b) \\ S^{k}(a,b) = \Delta S^{k-1}(a,b) + \dfrac{\Delta P^{k}(a,b)}{|I(a)||I(b)} * C \end{cases}$

迭代：$\begin{cases} \Delta P^{k+1}(\mathrm{c},d) = \dfrac{\Delta P^{k}(a,b)}{|I(a)||I(b)} * C \\ \text{将}\Delta P^{k+1}(c,d)\text{发给顶点}(c,d) \\ \text{其中}c\in I(a), d\in I(b) \end{cases}$

经过了以上的变换之后可以发现，在顶点对(*a*，*b*)进行累积-迭代操作时，用到的数据只是与顶点对(*a*,*b*)有关，而与其他的顶点对无关。在操作中用到的*|I(a)||I(b)|*，可以将它放在计算节点的输入数据里，这样每个计算节点只需存储自己所需顶点的入度，而不需存储所有顶点的入度。

4.2 异步累积-迭代形式

上文推导出的公式(3)是同步执行的累积-迭代算法。所谓同步形式，就是每一次累积的迭代差值 $\frac{\Delta P^{k}(c,d)}{|I(c)||I(d)} * C$恰好是第 *k* 次和第 *k-1* 次迭结果之差。但是为了获得更高的运行效率，必须让每个顶点对的累积-迭代操作异步地进行。任意顶点对的累积-迭

代操作可以随时执行，不必等待两次迭代的差值算出来，才开始执行操作

为了达到异步执行的目的，就必须打破迭代次数 k 对算法计算粒度的束缚（计算粒度是指每次累积迭代差值的大小）。在(3)式中，$\Delta P^k(a,b)$是由一系列的值累加而得到的，可以将其表示为以下形式：

$$\Delta P^k(a,b) = \Delta P_1^{k-1} + \Delta P_2^{k-1} + \Delta P_3^{k-1} + \cdots + \Delta P_p^{k-1}$$

其中下标 p 为顶点对(a,b)的邻接顶点对的个数。在同步方式下可以保证$\{\Delta P_1^k, \Delta P_2^{k-1} \ldots \Delta P_p^k\}$全部到达之后才开始累积-迭代操作将$\Delta P^k(a,b)$累积起来。但是在异步执行的过程中，它们未必能够同时到达，那么假设一种简单的情况，在某一时刻 $\{\Delta P_1^{k-1}, \Delta P_2^{k-1} \ldots, \Delta P_i^{k-1}\}$ 首先到达，而$\{\Delta P_{1+1}^{k-1}, \Delta P_{1+2}^{k-1} \ldots, \Delta P_p^{k-1}\}$随后到达，其中$1 \le i \le p$，此时令：

$$\Delta P_1^{k-1}(a,b) = \Delta P_1^{k-1} + \Delta P_2^{k-1} + \cdots + \Delta P_i^{k-1}$$

$$\Delta P_2^{k-1}(a,b) = \Delta P_{i+1}^{k-1} + \Delta P_{i+2}^{k-1} + \cdots + \Delta P_p^{k-1}$$

那么就有 $\Delta P^k(a,b) = \Delta P_1^{k-1}(a,b) + \Delta P_2^{k-1}(a,b)$，在$\Delta P_1^{k-1}(a,b)$到达而$\Delta P_2^{k-1}(a,b)$未到达时就开始累积-迭代操作，那么就可以得到如下计算过程：

$$\Delta P_1^{k-1}(a,b) = \Delta P_1^{k-1} + \Delta P_2^{k-1} + \cdots + \Delta P_i^{k-1}$$

$$S_1^k(a,b) = S^{k-1}(a,b) + \frac{\Delta P_1^{k-1}(a,b)}{|I(a)||I(b)} * C$$

当$\Delta P_2^{k-1}(a,b)$到达的时候：

$$\Delta P_2^{k-1}(a,b) = \Delta P_{i+1}^{k-1} + \Delta P_{i+2}^{k-1} + \cdots + \Delta P_p^{k-1}$$

$$S_2^k(a,b) = S_1^k(a,b) + \frac{\Delta P_2^{k-1}(a,b)}{|I(a)||I(b)} * C$$

$$= \frac{\Delta P_1^{k-1}(a,b)}{|I(a)||I(b)} * C + \frac{\Delta P_2^{k-1}(a,b)}{|I(a)||I(b)} * C + S^{k-1}(a,b)$$

$$= \frac{\Delta P_2^{k-1}(a,b) + \Delta P_2^{k-1}(a,b)}{|I(a)||I(b)} * C + S^{k-1}(a,b)$$

$$= S^{k-1}(a,b) + \frac{\Delta P^k(a,b)}{|I(a)||I(b)} * C = S^k(a,b)$$

由以上过程可以发现，在假设的情况下，即使是顶点对(a,b)在计算时所需的值并没有全部到达就可以开始接下来的操作，并且最终仍然可得到正确的结果$S^k(a,b)$。当然这只是在异步时实际情况中最简单的一种情况，只是为了便于理解而举出的简单例子，实际情况要复杂的多。

为了能够涵盖所有的情况，本文采用时间序列来表征算法的计算进度，下面给出 Asyn-SimRank 算法的计算形式如公式(4)。

$$\begin{cases} S^{t_k}(a,b) = S^{t_{k-1}}(a,b) + \dfrac{\Delta P^{t_k}(a,b)}{|I(a)||I(b)|} * C \\ \Delta P^{t_{k+1}}(a,b) = \displaystyle\sum_{c \in I(a), d \in I(b)} \frac{\Delta P^{t_k}(c,d)}{|I(c)||I(d)} * C \end{cases} \tag{4}$$

其中$t_0, t_1, t_2, t_3 \cdots\cdots$是一个时间序列。$t_{k+1}$- t_k与 t_k- t_k,可以不相等。每个顶点对所对应的时间序列均不同。由 Asyn-SimRank 算法可以看出，相似度的计算可按时间变化，而这个时间序列完全可以由顶点对的累积-迭代操作何时结束来决定。那么显而易见，Asyn-SimRank 算法的计算粒度是由操作来决定的，而不是由迭代次数 k 来决定的，这样就保证了累积-迭代操作可以随时进行而不需等待。

当然，这样的计算形式算法能否收敛并得到正确的结果，必须给出严格的证明。在下一节将证明 Asyn-SimRank 算法的正确性和收敛性。

由于 Asyn-SimRank 算法的计算公式与 Delta-SimRank 算法有很多类似之处，在此有必要说明一下两算法的不同。

（1）Delta-SimRank 算法计算粒度固定。Delta-Simrnak 算法严格地规定每次计算的粒度是两次迭代之差。固定的计算粒度使得计算节点不得不等待所有必要的数据全部到达后才能开始下一步操作，造成很大的时间开销。而计算粒度任意的 Asyn-SimRank 算法可以利用现有的数据立即开始下一步的操作，无需等待，这也是异步运行的前提。

（2）Delta-SimRnak 算法存在对邻接顶点入度的依赖。在前文已经提到这个问题。对邻接顶点入度的依赖造成运行 Delta-SimRank 算法的每个计算节点在存储整张图的同时还必须存储整张图每个顶点的入度，造成很大的存储开销。Asyn-SimRank 算法通过构造G^2图和变换计算公式的方法解决了计算节点需存储整张图和所有顶点入度的问题，并剔除了孤立的顶点对，有效的节省了存储空间。

（3）Delta-SimRank 无法应用优先级迭代。Delta-SimRank 算法的同步计算方式决定了它只能优先的计算迭代差值非零的顶点，而无法优先的计算部分优先级高的顶点，而优先的计算优先级高的顶点又是应用优先级迭代的前提条件。Asyn-SimRank 可以在优先级比较高的顶点上进行更多次的累积-迭代操作，使该顶点的影响迅速的传

播出去。应用优先级迭代可以有效地加速算法的收敛速度，进而得到更快的整体运算速度。

综上 Delta-SimRank 算法仍然是同步的算法，收敛速度以及计算速度均不如Asyn-SimRank算法。

4.3 收敛性证明

SimRank 的累积-迭代形式的收敛性是 Asyn-SimRank 算法的收敛性的充分条件。要证明 Asyn-SimRank 算法的收敛性必须首先证明 SimRank 累积-迭代形式的收敛性。

SimRank 累积-迭代形式的收敛性：若能够证明$\Delta P^{k+1}(a,b)$在$\mathrm{k}\to\infty$时趋于 0，那么就可以证明 SimRank 的累积-迭代形式是收敛的。假设$k\to\infty$时，$M=max_{(a,b)}|\Delta P^{k+1}(a,b)|$，由公式(3)有：

$$M = |\sum_{c\in I(a),d\in I(b)} \frac{\Delta P^k(c,d)}{|I(c)||I(d)|} * C|$$

$$M \le \sum_{c\in I(a),d\in I(b)} |\frac{\Delta P^k(c,d)}{|I(c)||I(d)|} * C|$$

由于$|I(c)|I(d)| > 0$，$\mathrm{C} > 0$所以有：

$$M \le \sum_{c\in I(a),d\in I(b)} \frac{|\Delta P^k(c,d)|}{|I(c)||I(d)|} * C$$

$$M \le \sum_{c\in I(a),d\in I(b)} \frac{M}{|I(c)||I(d)|} * C$$

$$M \le |I(c)|I(d)|\frac{M}{|I(c)||I(d)|} * C$$

$$M \le M * C$$

由于$0 < C < 1$，若要$M \le M * C$成立，当且仅当$M = 0$，由此可得，在$\mathrm{k}\to\infty$时，$\Delta P^{k+1}(a,b) = 0$，那么$\frac{\Delta P^k(c,d)}{|I(c)||I(d)} * C = 0$，就可以得出以下结论：SimRank 累积-迭代形式在$k\to\infty$时，$S^k(a,b) = S^{k-1}(a,b)$，即相似度值趋于定值，所以 SimRank 累积-迭代形式收敛。

Asyn-SimRank 算法收敛性：首先由 Asyn-SimRank 同步累积-迭代形式可以推导出，经过 k 次累积-迭代操作之后：

$$S^k(a,b) = \Delta P^1(a,b) + \sum_{\substack{c_1\in I(a)\\ d_1\in I(b)}} f_{(a,b)}\left(\Delta P^1(c_1,d_1)\right) +$$

$$\sum_{\substack{c_1\in I(a)\\ d_1\in I(b)}} f_{(a,b)}\left(\sum_{\substack{c_2\in I(c_1)\\ ,d_2\in I(d_1)}} f_{(c_1,d_1)}\left(\Delta P^1(c_2,d_2)\right)\right) + \cdots +$$

$$\sum_{\substack{c_1\in I(a),\\ d_1\in I(b)}} f_{(a,b)}\left(\cdots\sum_{\substack{c_k\in I(c_{k-1})\\ d_k\in I(d_{k-1})}} f_{(c_{k-1},d_{k-1})}\left(\Delta P^1(c_k,d_k)\right)\right) \quad (5)$$

其中 $\mathrm{f_{(x,y)}(z)} = \frac{\mathrm{z}}{|\mathrm{I(x)||I(y)|}} * \mathrm{C}$，由公式(5)可以看出，Syn-SimRank 算法进行 k 次累积-迭代操作后，由于所有的顶点对都会同步完成两操作，所以$S^k(a,b)$可以收到所有的以(a,b)为终点，长度为 k-1 的路径上传来的值。公式(5)中的第 m 项就对应以(a,b)为终点，长度为 m-1 的路径上传来的值。

对于 Asyn-SimRank 算法的异步累积-迭代形式，为了表示它接收数值的过程， 定义每个顶点对都对应一个不同的时间序列t_1，t_2，t_3，……，该时间序列完全由累积-迭代操作完成时刻来决定。如t_1，t_2，t_3，……是顶点对(a,b)对应的时间序列，那么 t_k时刻就代表顶点对(a,b)刚好完成 k 次累积-迭代操作。定义t_k时刻(a,b)的相似度值为$S^{t_k}(a,b)$，由式(5)可以得到$\mathrm{t_k}$时刻，

$$S^{t_k}(a,b) = \Delta P^{t_1}(a,b) + \sum_{\substack{c_1\in I(a)\\ d_1\in I(b)}} f_{(a,b)}\left(\Delta P^{t_1}(c_1,d_1)\right) +$$

$$\sum_{\substack{c_1\in I(a)\\ d_1\in I(b)}} f_{(a,b)}\left(\sum_{\substack{c_2\in I(c_1)\\ ,d_2\in I(d_1)}} f_{(c_1,d_1)}\left(\Delta P^{\mathrm{t}_1}(c_2,d_2)\right)\right) + \cdots +$$

$$\sum_{\substack{c_1\in I(a),\\ d_1\in I(b)}} f_{(a,b)}\left(\cdots\sum_{\substack{c_k\in I(c_{k-1})\\ d_k\in I(d_{k-1})}} f_{(c_{k-1},d_{k-1})}\left(\Delta P^{\mathrm{t}_1}(c_k,d_k)\right)\right) \quad (6)$$

其中 $\mathrm{f_{(x,y)}(z)} = \frac{\mathrm{z}}{|\mathrm{I(x)||I(y)|}} * \mathrm{C}$，公式(6)式与公式(5)形式十分类似，但实质不同，t_k时刻，$S^{t_k}(a,b)$可以收到以(a,b)为终点，长度为 k-1 的路径上传来的值。但是由于异步执行，不能保证完全收到，如当路径上的某一点还没有完成第 k 次累积-迭代操作，那么$S^{t_k}(a,b)$就收不到该路径上传来的值。显然有当所有的相似度值都刚好完成 k 次累积-迭代操作时，$S^k(a,b) = S^{t_k}(a,b)$。

由公式(5)(6)可以看出$S^k(a,b)$和$S^{t_k}(a,b)$都是单

调增加的。那么取 k 为异步执行时所有顶点对完成累积-迭代操作的最大次数，由于$S^{t_k}(a,b)$可能收不到长度为 k-1 的路径上传来的所有值，就有：

$$S^{t_k}(a,b) \le S^k(a,b)$$

进而，一定会找到一个常数 c，使得 k-c 为异步执行时所有顶点对完成累积-迭代的最小次数，就有：

$$S^{k-c}(a,b) \le S^{t_k}(a,b)$$

综上，就可以得到：

$$S^{k-c}(a,b) \le S^{t_k}(a,b) \le S^k(a,b)$$

上文已经证明$S^k(a,b)$当$k \to \infty$时收敛于一个定值，那么由夹逼准则就推出当$k \to \infty$时$S^{t_k}(a,b)$收敛，且与$S^k(a,b)$收敛于同一定值。至此，就证明了 Asyn-SimRank 算法的正确性以及收敛性。

4.4 Asyn-SimRank 算法的优势

Asyn-SimRank 避免了传统 SimRank 的问题，特别适合大规模的分布式实现。具体来说，它的优点可以总结如下：

支持异步计算：Asyn-SimRank 算法运行时每个顶点对的累积-迭代操作可以完全异步的进行，使得 Asyn-SimRank 算法可以在异步式分布式框架上实现，进而利用异步式分布式框架的速度优势，快速的计算相似度。

支持优先级迭代：在图上计算相似度值时，总是大部分顶点先收敛，少部分顶点后收敛。如果可以优先的计算这那些较晚收敛的点，就可以得到较快的整体收敛速度。实验表明，优先的计算迭代差值大的顶点，可以得到较快的整体收敛速度。这样就可以结合优先级迭代进一步提高计算速度。

计算量相对较小：Asyn-SimRank 采用的是累积-迭代的方式进行计算，收敛的点迭代差值一定为零，这样就可以不去计算这些顶点。进而节省已收敛顶点的计算开销。假设当前没有收敛的点的个数为 m，那么部分顶点快速收敛后，Asyn-SimRank 算法的时间复杂度会降为 $O(p^2m)$，其中 p 为顶点的平均邻接点数。由于图中两顶点的相似度总是由部分少数临近的顶点决定的，而不是由所有顶点决定的，所以 m 相对于n^2还是比较小的。

通信量相对较小：由于节省了已收敛顶点的计算，那么已收敛顶点自然也不会再有通信，所以当部分顶点快速收敛之后，通信量也会降为 $O(p^2m)$。

Asyn-SimRank 算法的这些优点，决定了该算法在计算量、通信量、整体计算速度等方面的优势，在后面的实验中也证实了这一点。

5. Asyn-SimRank 算法的分布式实现

4.1 节已经简要的给出了 Asyn-SimRank 算法对应的累积-迭代操作的分布式实现方法，但是仍然有很多（如对输入数据的处理，计算节点执行流程等）实现细节没有详细的说明。接下来本章将结合分布式环境的特点以及一些图例来进一步详细地说明 Asyn-SimRank 算法的分布式实现方法。

5.1 输入图的预处理

如果某个算法处理的数据是图的话，以边为中心的算法是指计算两图顶点关系的算法，即计算图顶点之间边的权值。SimRank 算法就是典型的以边为中心的算法。以顶点为中心的算法是指计算图中单个顶点上的值。典型的以顶点为中心的算法如 PageRank 算法。

很多的分布式框架不支持以边为中心的算法，如 Hadoop 框架，在算法 1 中，*Map* 函数输入的不是图顶点而是顶点对，因为直接输入单个图顶点是无法计算的。但是以输入顶点对的形式来支持以边为中心的算法，又会带来新的问题。比如每个顶点对的组合是任意的，在计算过程中为了获取两顶点的邻接表就不得不让每个 *Map* 节点存储整张图。

但是如果利用前文提到的将输入图 G 构造成G^2图的方式来解决这个问题，就可以成功的将 SimRank 算法由以边为中心转化为以顶点为中心。这种方式有诸多的优点。一方面数据恰好是按着G^2图的结构流动的，在进行数据发送的时候只需用到G^2图中顶点的邻接表，每个计算节点不在需要存储整张输入图。另一方面，G^2图中孤立的顶点在计算过程中是不影响其他顶点的值，自身的值也不会改变，它对整个计算是无意义的。因此可以直接去除G^2图中出度入度均为零的顶点来减少计算量。

5.2 分布式环境上的实现

目前的分布式环境结构主要要是 Master-Slave 结构。在这种结构下的分布式环境，都会有一个 Master 控制节点和多个 Slave 计算节点。Master 节点负责任务的提交、任务的分配和部署以及负载均

衡。Slave 节点则负责完成相应的任务分片，并且每个 Slave 节点运行的程序是完全一样的。在运行 Asyn-SimRank 算法时每个 Slave 节点完成什么样的工作以及各个 Slave 之间传递什么样的消息，将在本节详细的介绍。

不难想出每个 Slave 节点需要完成的工作就是累积操作和迭代操作。在任务开始时，Master 节点会将相应的数据分片分配给 Slave 节点，Slave 节点负责在相应的数据分片中的每行记录上完成迭代操作和累积操作。图 2 所示的就是各个 Slave 节点的处理过程。每个 Slave 节点为相应数据分片中的每行记录保存 3 个变量：唯一标识 K、相似度累积值 V、迭代差值值 ΔV。如果该 Slave 节点的数据分片有 n 行记录，那么就会创建一个长度为 n 的数组来存储 K，V 和 ΔV。图 2 中 Slave1 节点和 Slave2 节点用 ΔV 中的值计算出要发送的数据并发送。Slave3 节点则接受 Slave1 和 Slave2 发来的数据，并将这些数据累积到变量 ΔV，并适时的用 ΔV 计算出要发送的值发向其他的 Slave 节点，然后将 ΔV 变量清零。

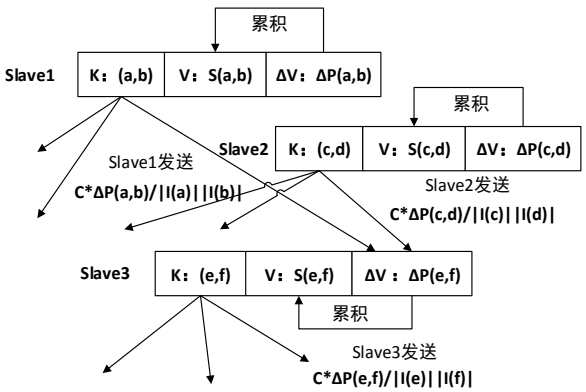

图 2 各个 Slave 节点的处理过程图

图 2 简要的给出了 Slave 节点的处理过程，便于直观的理解，算法 2 详细地说明 Asyn-SimRank 算法的分布式实现。

算法 2 分布式 Asyn-SimRank 算法

输入：G^2图

输出：顶点对 相似度值

1. WHILE(*check()*) DO：
   // *chech()*用于判断是否收敛或满足一定条件
2. FOR L= 0 TO size //size 为数据分片中的记录行数
3. IF(ΔV==0|| *checkpri()*) // *checkpri()*检测记录的优先级
4. CONTINUE;
5. 累积操作开始：
6. 接受其他节点法来的数据累积到 ΔV
7. $V=V+C*\Delta V/|I(a)||I(b)|$
8. 累积操作结束
9. 迭代操作开始：
10. 对每个$(c,d)\in O(a,b)$ //假设当前顶点是(a,b)
11. sent=$C*\Delta V/|I(a)||I(b)|$
12. 将 sent 发送给(c,d)
13. ΔV=0
14. 迭代操作结束
15. END FOR
16. END WHILE

算法 2 的最外层是一个循环，在算法不收敛或结果没有到达要求的精度时，重复的执行循环中的 FOR 语句。FOR 语句负责完成对数据分片中的每行记录进行累积-迭代操作。在进行操作之前，会先行的判断 ΔV 是否为零，如果为零则说明该顶点已经收敛或还没有其他的顶点影响到提的值，此时累积-迭代操作无意义。除此之外还会先行判断该行记录的优先级，达不到特定值则不进行累积-迭代操作，这就有效的利用了优先级迭代来优先的计算慢收敛的顶点来加快收敛速度。

必须提出的是，每个 Slave 节点都会执行算法 2 的循环过程，但是每个 Slave 节点完全是独立的，有数据的时候可以马上进行计算，无需同步，这就达到了异步计算的目的。分布式异步计算框架 Maiter[18]提供了充足的 API[18]可以方便的实现算法 2 ,因此在本文的实验中也是用 Maiter 框架来实现的 Asyn-SimRnak 算法。

6. 实验

为了验证本文提出的 Asyn-SimRank 算法的在通信量、收敛速度、整体计算速度等方面的优越性，本实验选取不同的实际应用系统产生的数据集作为输入数据，分别在 Hadoop 框架上实现 SimRank 算法、delta-SimRank 算法，以及异步分布式框架 Maiter 上实现 Asyn-SimRank 算法，比较三者运行过程中的通信量、收敛速度、以及总体的运行时间。

在本文的对比图中，SR 代表 SimRank 算法，D-SR 代表 Delta-SimRank 算法，A-SR-UP 代表无迭代优先级的 Asyn-SimRank 算法，A-SR-P 代表有迭代优先级的 Asyn-SimRank 算法。

6.1 实验数据集及实验环境

实验数据集：本实验所选用的数据集规模从小

到大，实际计算顶点数从 288 万到 2.6 亿，并且是实际应用中产生的数据集。选取实际的数据集运行算法可以有力的说明算法在实际应用中运行的真实情况。这些数据集[①]均可下载。实验中选择的五个数据集包括：天文物理论文合作网络 ca-AstroPh、相对论及量子宇宙论文合作网络 ca-GrQc、高能物理论文合作网络 ca-HepTh、凝态物理论文合作网路 ca-CondMat、文件共享网络 P2P。数据集的相关信息如表 1 所示.

表 1 实验数据集相关信息[①]

名称	顶点数	边数	直径	平均邻接点数	实际计算顶点数
ca-AstroPh	2403	9852	10	4.1	2888406
ca-GrQc	5242	28980	17	5.3	13741903
ca-HepTh	9877	51971	17	5.3	48782503
ca-CondMat	15014	61557	14	4.1	112717605
P2P	22687	54705	10	2.4	257361328

实验环境：本实验中所用到的分布式环境包括 33 台计算机。大部分实验如运行时间、收敛速度、通信量实验采用 17 台计算机，一个 Master 节点，

表 2 节点软硬件配置

CPU	INTEL CORE i3-2100 LGA-1155 4 核
内存	apacer 4G-DDR3 *2
硬盘	hitachi 500G/7200RPM
网络	1000M 以太网
系统	Redhat 6.1 64 位
Hadoop	1.0.4 版本
Maiter	0.1 版本

16 个 Slave 节点。在分布式环境规模与运行时间的关系实验中，最多用到了 33 台计算机。分布式环境中计算机的软件硬件配置如表 2 所示。

6.2 总体运行时间对比

图 3 所示（由于数据之间差值过大，为不失去柱形图的对比效果，有部分结果没有画入图中，这些数据会放在表 3 中）的是三个算法在不同的数据集上总体计算速度的对比，纵坐标的单位为秒。可以看出 Asyn-SimRank 算法具有明显的优势，总体计算时间明显缩短。从表 4 中看到，Asyn-SimRank 算法最快能比 SimRank 算法快 131.9 倍，比 Delta-SimRank 算法最快可快 64.7 倍，运行速度提升如此之大说明了 Asyn-SimRank 算法的高效性。并且数据的规模从实际计算顶点 288 万到 2.6 亿，

表 3 ca-CondMat 数据集运行时间

SR	D-SR	A-SR-UP	A-SR-P
50777s	3922s	797.3s	384.9s

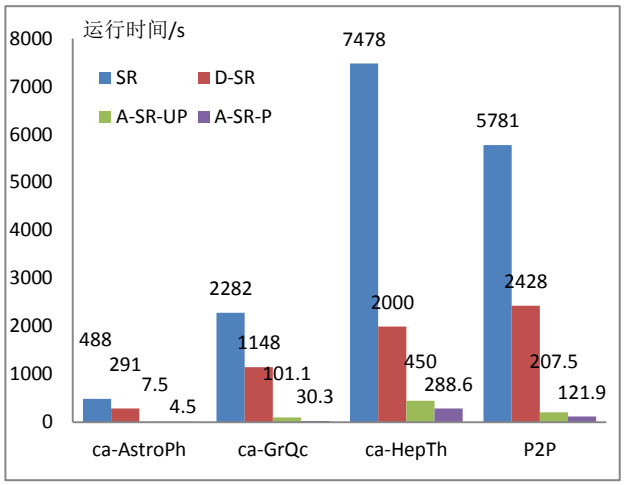

图 3 运行时间对比

随着数据量的增大，Asyn-SimRank 算法仍保持着很快的运行速度。可见 Asyn-SimRank 算法的扩展性非常好。

表 4 Asyn-SimRank 算法加速比

	相对 SimRank	相对 Delta-SimRank
ca-AstroPh	99.5	64.7
ca-GrQc	75.3	37.9
ca-HepTh	32.7	6.9
ca-CondMat	131.9	10.2
P2P	44.5	19.9

6.3 收敛速度对比

算法收敛的速度是评价算法好坏的重要指标。为了能够公平的衡量各个算法的收敛速度并剔除系统的影响，本实验通过监测相似值的总量增长的速度随顶点更新总次数的变化来衡量算法的收敛速度。这个监测的实质意义是在几个算法中，对图做相同的工作量，看哪个收敛的更快。所谓顶点更新总次数是指从算法开始到某一时刻，所有顶点的相似度值被更新的次数之和。相似度值总量是指所有顶点的相似度之和。因为在 SimRank 算法的计算过程中，可以证明相似度之和是递增的。那么随着更新次数的增加，相似度值的总量也会增加，并且增加的越快说明收敛的速度越快。

图 4 与图 5 所示的曲线为三个算法的收敛速度

① http://snap.stanford.edu/data/index.html

的对比。纵坐标为相似度总量，横坐标为更新次数。从图中可以看出，没有迭代优先级的 Asyn-SimRank 算法的收敛速度比 SimRank 好，但是并不比

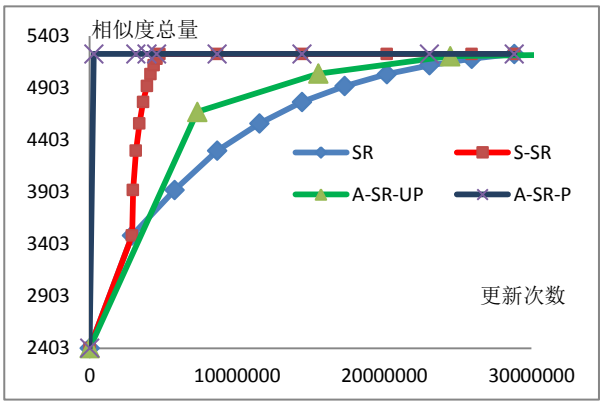

图 4 ca-AstroPh 数据集上收敛速度对比

Delta-SimRank 好，主要原因是因为在没有迭代优先级的情况下，Asyn-SimRank 算法会和 Delta-SimRank 算法一样，会对所有迭代差值非零的顶点更新，但是 Asyn-SimRank 是异步更新，可以是任意的更新粒度，所以更新的粒度要比

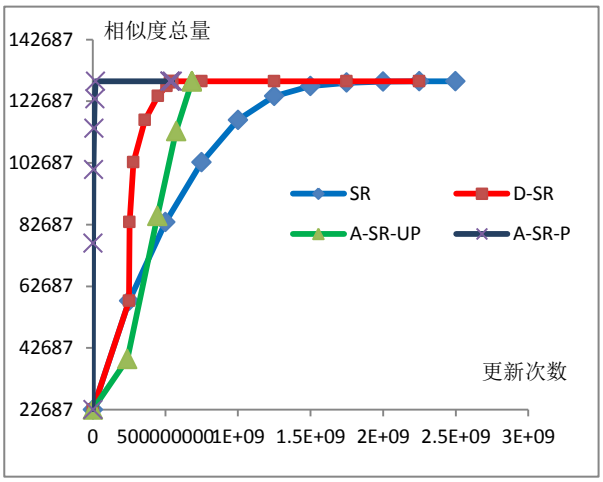

图 5 P2P 数据集上收敛速度对比

Delta-SimRank 算法小，达到相同的相似度总量时更新次数比较大。虽然如此，在运行速度上无迭代优先级的 Asyn-SimRank 算法还是远远快于 Delta-SimRank，主要得益于异步计算。

有迭代优先级的 Asyn-SimRank 算法的收敛速度远远比 SimRank 和 Delta-SimRank 算法快。在有优先级的情况下，Asyn-SimRank 将更新全部作用于那些会对全局影响明显的顶点上，让影响尽快的传播出去。并且让重要的或收敛慢的顶点先进行运算，加快了算法收敛的特性。因此可以支持优先级迭代是 Asyn-SimRank 算法的有力优势。

6.4 算法与分布式环境规模的关系

为了充分的验证 Asyn-SimRank 算法能否适应大规模的分布式环境，处理大规模的数据，本实验对 Asyn-SimRank 算法的运行时间与分布式环境规模的关系进行了实验分析。在本次实验中，分别在 17 台、25 台、33 台计算机上使用相同的数据集 (ca-CondMat，1.1 亿实际计算顶点) 运行了 Asyn-SimRank 算法，并监测算法运行的时间，实验结果如图 6 所示。

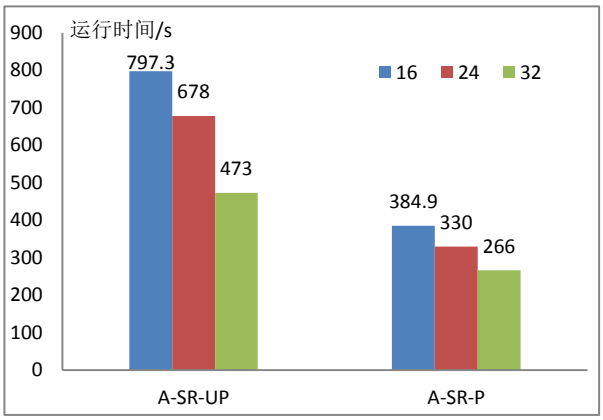

图 6 算法运行时间与分布式环境规模的关系

由图 6 中可以观察到，随着计算节点的增多，算法的运行时间也在不断的降低。可见随着分布式环境规模的增长，Asyn-SimRank 算法处理相同数据所用的时间越来越短，这就说明了 Asyn-SimRank 算法在分布式环境规模方面具有良好的扩展性，可以部署在大规模的分布式环境上处理大规模的数据。

6.4 通信量对比

表 5 ca-AstroPh 数据集通信量

SR	D-SR	A-SR-UP	A-SR-P
3GB	1.1GB	0.48GB	0.4GB

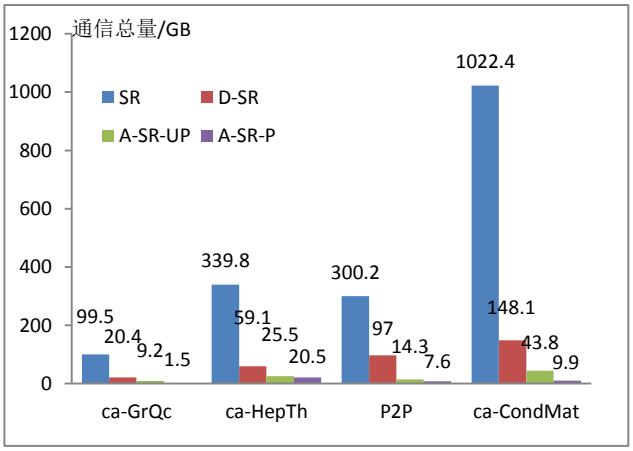

图 6 通信量对比

图 7 所示（由于数据之间差值过大，为不失去

柱形图的对比效果，有部分结果没有画入图中，这些数据会放在表 5 中）的是各算法在不同的数据集上通信量的对比。图中纵坐标的单位为 GB。从图中可以看出，Asyn-SimRank 算法的通信量相与其他的算法相比明显降低。Asyn-SimRank 算法和 SimRank 算法的最大差距可达到 103 倍，和 Delta-SimRank 算法的差距最大可达 15 倍。Asyn-SimRank 算法的通信量小的原因主要是因为节省了已经收敛顶点和相似度值暂时不变顶点的计算。由于 SimRank 算法的本身的性质，总是大部分值提前收敛，少部分的值后收敛，因此计算量的节省和通信量的节省是十分可观的。

7. 总结

本文提出的 Asyn-SimRank 算法通过改变 SimRank 算法的迭代计算方法，解决了传统 SimRank 算法计算量高、通信量大、计算效率低等的诸多问题。改进的 SimRank 算法在计算效率上有了很大的提升，并且特别适合大规模分布式计算。

Asyn-SimRank 算法的提出还进一步说明，累积-迭代方式是一种高效率的迭代方式，它不仅能够减小计算量、通信量，还为算法支持异步计算提供了有利的条件。很多的其他算法，如 PageRank 算法也可以变换成这种形式。所以可以说累积-迭代方式为解决大规模数据处理上的迭代计算问题提供了有效途径。

参 考 文 献

[1] Glen, J. Widom. Simrank: a measure of structural-sontext similarity//Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining(KDD'02). Strathcona,USA. 2002:538-543

[2] K. Shvachko, H. Kuang. The hadoop distributed file system//Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies. Salt Lake City, USA, 2010:1-10

[3] J. Dean, S. Ghemawat. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2004,51(1):107-113

[4] M. Zaharia, M. Chowdhury, M. J. Franklin. Spark: cluster computing with working sets //Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. Washington, DC, USA,2010:10-10

[5] Y. Bu, B. Howe, M. Balazinska, and D. M. Ernst. Haloop: efficient iterative data processing on large clusters. VLDB Endowment, 2010, 3(1-2):285-296

[6] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn,N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). Ashraf Aboulnaga ,2010:135–146

[7] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G.Fox. Twister: a runtime for iterative mapreduce // Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10). Chicago, Illinois, USA, 2010:810–818.

[8] Zhang Y, Gao Q, Gao L, Wang C. iMapReduce: a distributed computing framework for iterative computation// Proceedings of the 2nd ACM Symposium on Cloud Computing(SOCC'11). Cascais, Portugal, 2011: 1112-1121

[9] Zhang Y, Gao Q, Gao L, Wang C. PrIter: a distributed framework for prioritized iterative computations// Proceedings of the 2nd ACM Symposium on Cloud Computing(SOCC'12). San Jose, CA, USA, 2012: 13

[10] Cao L, H. Duk Kim, M. Tsai. Delta-SimRank computing on mapreduce//Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms Systems Programming Models and Applications (BigMine '12). Beijing, China, 2012:28-35

[11] Lizorkin, D., Velikhov, P., Grinev, M.N., Turdakov. Accuracy estimate and optimization techniques for simrank computation: VLDB ,2010, 19(1):45–66

[12] Mendelzon, A.O. Review—authoritative sources in a hyperlinked environment//Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, 2000: 604–632

[13] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation: VLDB Endowment, 2008,1(1):422–433

[14] G. He, Feng H, Li C, Chen H. Parallel simrank computation on large graphs with iterative aggregation//Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, DC, USA, 2010: 543-552

[15] W. Xi, E. A. Fox, W. Fan, B. Zhang, Z. Chen, J. Yan, and，D. Zhuang. Simfusion: measuring similarity using unified relationship matrix//Proceedings of the 28th，annual international ACM SIGIR conference on Research and development in information retrieval. Washington, USA,2005: 130–137

[16] D. Fogaras and B. Rácz. Scaling link-based similarity search //Proceedings of the 14th international conference on World Wide Web. Chiba, Japan,2005:641–650

[17] Distributed PageRank computation based on iterative aggregation-disaggregation methods// Proceedings of the 14th ACM international conference on Information and knowledge management. Bremen, Germany, 2005:578-585

[18] Zhang Y, Gao Q. Accelerate large-scale iterative computation through asynchronous accumulative updates //Proceedings of the 3rd workshop on Scientific Cloud Computing Date (ScienceCloud '12). Chicago, USA, 2012:13-22

.

WANG Chun-Lei, born in 1989. Graduate student in School of Information Science and Engineering, Northeastern University. His research interests include big data and cloud computing.

ZHANG Yan-Feng, born in 1982. Associate Professor at Computing Center, Northeastern University. His research interests include big data and cloud computing (zhangyf@cc.neu.edu.cn).

ZHAO Chang-Kuan, born in 1976. Lecture at Computing Center, Northeast University. His research interests include social networks, distributed system, cloud computing (zck@cc.neu.edu.cn).

YU Ge, born in 1962. Professor and PhD supervisor in School of Information Science and Engineering, Northeastern University. His research interests include databases, distributed systems and embedded systems (yuge@mail.neu.edu.cn)

GAO Li-Xin, born in 1968. Professor in ECE department, UMass Amherst. Her research interests include Internet routing and cloud computing (lgao@ecs.umass.edu)

Background

The advances in data acquisition, storage, and networking technology have created huge collections of high-volume, high-dimensional relational data. Making sense of these relational data is critical for companies and organizations to make better business decisions and even bring convenience to our daily life. People are using cloud platforms or distributed environment to perform large-scale data mining. The efficiency of large-scale data mining algorithms will affect the performance a lot. Designing efficient large-scale algorithms and improving the performance of large-scale applications are quite meaningful work in the era of “big data”.

SimRank algorithm is used to explore the network structure to measure object similarity. It has been successfully used for many applications in social networks, social annotation, information retrieval, and link prediction. SimRank algorithm is a typical high complexity algorithm and will result in high volume communication in a distributed environment. In recent years, improving the performance of large-scale SimRank algorithm has attracted many research efforts. These include some techniques for reducing the computation complexity and using new hardware (e.g. GPU) to compute SimRank. However, all these work do not notice that iterative process is the key to limit performance in a distributed environment. This paper exploits cumulate-iterate approach and priority iteration to resolve the problems of high-complexity and high-communication. The results show that the performance of large-scale SimRank computation is significantly improved.

This work is mainly supported by Fundamental Research Funds for the Central Universities (N120416001), named as “Research on Optimization Techniques for Fixpoint Iterative Algorithms based on MapReduce”. The project aims to improve the performance of large-scale iterative algorithms; this paper focuses on improving the performance of a typical iterative algorithm, SimRank. This paper is also supported by National Natural Science Foundation of China (No. 61300023), Fundamental Research Funds for the Central Universities (N120816001), MOE-Intel Special Fund of Information Technology (MOE-INTEL-2012-06), and MOE-China Mobile Labs Special Fund (MCM20122051). The group has working in the area of large-scale iterative computations for a few years and published many papers.