DOI:10.7544/issn1000-1239.2015.20140313
52(7): - , 2015

Asyn-SimRank:一种可异步执行的大规模 SimRank 算法

王春磊[1] 张岩峰[2] 鲍玉斌[1] 赵长宽[2] 于 戈[1] 高立新[3]
[1](东北大学信息科学与工程学院计算机软件研究所 沈阳 110819)
[2](东北大学计算中心 沈阳 110819)
[3](美国麻州大学阿默斯特校区电子与计算机工程系 美国阿默斯特 01003)
(zhangyf@cc.neu.edu.cn)

Asyn-SimRank: An Asynchronous Large-Scale SimRank Algorithm

Wang Chunlei[1], Zhang Yanfeng[2], Bao Yubin[1], Zhao Changkuan[2], Yu Ge[1], and Gao Lixin[3]
[1](*Institute of Computer Software, College of Information Science and Engineering, Northeastern University, Shenyang* 110819)
[2](*Computer Center, Northeastern University, Shenyang* 110819)
[3](*Department of Electrical and Computer Engineering, University of Massachusetts Amherst, Amherst, U.S.* 01003)

Abstract The 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 the satisfactory performance. In this paper, 1) We propose Asyn-SimRank byemploying the iterate-cumulate approach, which asynchronously executes the core iterative computation to avoid the high-cost synchronization barriers in large-scale distributed environments, and effectively reduces the amount of computation and communication. 2) We further propose the keypoint priority scheduling mechanism to accelerate convergence. 3) We prove the accuracy and convergence property of Asyn-SimRank as well as the efficiency of the keypoint priority scheduling. 4) We then implement Asyn-SimRank on Maiter, which is a distributed framework supporting asynchronous iteration. Expremental results show that, compared with the SimRank and Delta-SimRank implementing on Hadoop and Spark, the large-scale Asyn-SimRank significantly promotes the computational efficiency and accelerates the convergence.

Key words asynchronous computation; iterative computation; Asyn-SimRank; similarity; big data; MapReduce; Maiter

摘 要 SimRank 算法利用网络结构来评估网络中任意 2 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

收稿日期:2014-04-09;**修回日期**:2014-08-12
基金项目:国家自然科学基金项目(61300023,61272179,61033007,61173028);中央高校基本科研业务费基金项目(N120416001,N120816001);中国移动基金项目(MCM20122051);辽宁省科技计划基金项目(2013217004)

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,SimRank 算法不能在分布式环境下高效地实现.1)提出 Asyn-SimRank 算法,该算法采用迭代-累积的方式完成迭代计算,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,同时有效降低计算量并减少通信开销;2)提出关键点优先调度计算,提升了 Asyn-SimRank 算法的全局收敛速度;3)证明了 Asyn-SimRank 算法的正确性和收敛性以及关键点优先调度计算的有效性;4)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,相比较于 Hadoop,Spark 上实现的 SimRank 算法和 Delta-SimRank 算法,Asyn-SimRank 算法大大提升了算法的计算效率,加速了算法收敛.

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

中图法分类号 TP301.6

随着数据采集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,人们采用机器学习和数据挖掘算法对这些庞大的数据进行处理和分析.由于所涉及的数据量巨大,人们通常利用分布式环境进行大数据的处理.如何有效利用分布式环境设计高效的大规模机器学习和数据挖掘算法,成为当今大数据领域的热点研究问题.

SimRank 算法[1]利用图中各顶点之间的相互关系评估任意 2 顶点的相似度.它被广泛应用在社交网络、引用关系网络、链接预测等诸多领域中.例如,SimRank 算法被用于评估购物网络中产品之间的相似度或社交网络中朋友之间的相似度,再根据相似度来进行产品推荐或好友推荐.近年来,随着推荐系统的流行,SimRank 算法的重要性也更加突显出来.

由于单个计算机的计算能力和磁盘存储空间有限,所以单机上实现的 SimRank 算法在计算效率上以及处理数据的规模上,都难以满足大数据的要求.分布式计算通过多台计算机协同计算、共享磁盘存储空间等方式,有效地解决了单机计算的限制,因此在实际应用 SimRank 算法时,往往需要设计高效的分布式 SimRank 算法来支持大数据处理.并且随着 SimRank 算法应用的愈加广泛,设计高性能的分布式 SinRank 算法也愈加重要.

目前已经有很多 SimRank 算法的分布式实现.MapReduce 模型[2]和 Hadoop 框架[3]是目前最流行的大规模数据处理分布式编程模型和框架.在 Hadoop 框架上实现分布式 SimRank 算法时,虽然实现比较简单,但是通信量大,空间复杂度高,并且产生的大量中间结果,影响 SimRank 算法的计算效率.另外,由于 SimRank 算法是同步迭代算法.同步算法的计算效率受到超级步的限制,每次计算何时开始依赖于上次迭代何时结束.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,框架的改进对 SimRank 算法的效率提升有限.

Cao 等人[4]提出的 Delta-SimRank 算法采用迭代-累积(迭代地计算 2 次迭代的差值,然后将差值累积起来,得到最终相似度值,3.1 节详述)计算形式计算相似度值.迭代-累积计算形式避免了计算过程中收敛顶点参加运算,从而减少了计算量和通信量.但是 Delta-SimRnak 算法是 SimRank 基于 MapReduce 编程模型的改进算法,它仍然同步地执行迭代,没有避免同步开销.另外,Delta-SimRank 算法需要额外的 MapReduce 作业完成差值累积,增加了部分计算开销.

针对于 SimRank 算法和 Delta-SimRnak 算法所存在的同步开销问题,本文提出了可异步执行的 Asyn-SimRank 算法,在 Asyn-SimRank 算法的计算过程中,各个计算节点相似度值的更新不依赖上次迭代的计算结果,更新操作可以随时进行,无需任何同步过程,节省了同步开销.本文在异步迭代的基础之上提出了关键点优先调度计算,进一步提升了 Asyn-SimRank 算法的收敛速度.本文的主要贡献总结如下:

1) 在 Delta-SimRank 算法的基础之上,提出可以异步计算的 Asyn-SimRank 算法.Asyn-SimRank 算法在保持 Delta-SimRank 算法优点的同时,进一步实现异步计算,节省了同步开销.本文同时证明了 Asyn-SimRank 算法的正确性和收敛性.

2) 在异步迭代的基础之上,提出关键点优先调度计算.优先地计算影响全局收敛的关键点,有效地

提升了 Asyn-SimRnak 算法的收敛速度.本文同时证明了关键点优先调度计算对于提升算法整体收敛速度的有效性.

3) 改进了分布式异步计算框架 Maiter[5] 并扩展了 Maiter 的 API,在 Maiter 框架上实现了 Asyn-SimRank 算法.

4) 在真实的数据集上对 Asyn-SimRnak 算法进行了性能评估,实验结果显示了 Asyn-SimRank 算法的优良性能.

1 SimRank 算法原理与 Hadoop 实现

1.1 SimRank 算法原理

SimRank 算法的基本思想是:如果 2 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 1 条边,就可以得到 1 张对象图 $G=(V,E)$,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

$$S^{k+1}(a,b)=\begin{cases}1, & a=b;\\ \dfrac{C}{|I(a)||I(b)|}\displaystyle\sum_{\substack{c\in I(a)\\ d\in I(b)}}S^k(c,d), & a\neq b;\end{cases} \tag{1}$$

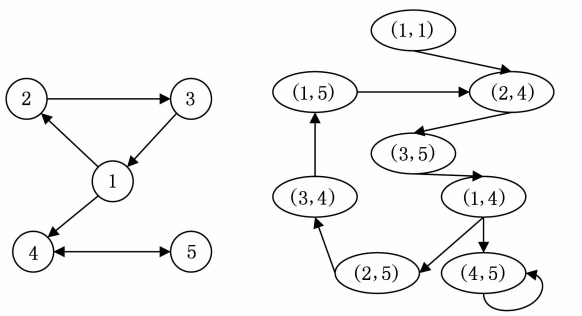
其中,$I(a)$和 $I(b)$分别为顶点 a 和顶点 b 的指入顶点集合;$S(a,b)$的取值范围为$[0,1]$;C 是一个阻尼系数,其取值范围为$(0,1)$.当 $a=b$ 时,$S^0(a,b)=1$;当 $a\neq b$ 时,$S^0(a,b)=0$.由式(1)经多次迭代,就可以得到最终的顶点间的相似度.

为了形象地描述 SimRank 算法的计算过程,可以构造 1 张顶点对图 $G^2(V^2,E^2)$[1] 来直观地观察 SimRank 算法在计算过程中消息的传递过程.图 G^2 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$(a,b)\to(c,d)\in E^2$.

图 1(a)为原始图 G,图 1(b)为 G 转化而来的顶点对图 G^2.以顶点 2,3,4,5 为例,顶点 2,3,4,5 出现在图 G 中,那么由图 G^2 的构造规则,顶点对(2,4),(3,5)出现在图 G^2 中.又由于图 G 中有边 2→3,4→5,那么在图 G^2 中就有 1 条由顶点对(2,4)指向(3,5)的边.其他顶点的转化过程类似.

在构造图 G^2 过程中,有一些边和顶点对在转化过程中被删除了,主要有 2 种情况:1)由于 $S(a,a)$始终为 1,(a,a)接收到的值并不在(a,a)上参加运算,因此向(a,a)传值没有必要,可以删去指向顶点对(a,a)的边;2) 入度为 0 的顶点对没有初值也没有非 0 值传入,始终为 0,没有必要参加运算,可以删除入度为 0 的顶点对(a,b)及其所有出边,其中 $a\neq b$.在删除顶点对(a,b)及其所有出边后,会导致某些顶点对(c,d)入度为 0,其中 $c\neq d$,级联地删除这些顶点对及其所有出边.删除这些点可以节省计算开销.

以图 1(b)中的部分顶点对为例,在 SimRank 算法计算开始时的第 1 次迭代过程中,由上文的初值设定规则,只有顶点对(1,1)的相似度不为 0.顶点对(1,1)将自己的初值 1 传给顶点对(2,4),该顶点对接收到该值后,利用式(1)计算出自己的相似度值,在第 2 次迭代时,(2,4)再将自己的相似度值传给顶点对(3,5),第 3 次迭代时,(3,5)再将自己的相似度值传给(1,4),依次类推,直至算法收敛.



(a) Original G (b) G^2

Fig. 1 An example of G and G^2.

图 1 G 图到 G^2 图的对应实例

由上例可以看出,SimRank 算法在计算的过程中消息是从一个顶点对传给另一个顶点对,即 2 顶点的相似度依赖于各自相邻顶点之间的相似度,其传递过程确切地反映了 SimRank 算法的计算原理.

图 G^2 可以将计算顶点间关系的 SimRank 算法转变成顶点对值的计算,以便于 SimRank 算法在以顶点为中心的分布式框架上实现,这在 5.1 节将详细介绍.

1.2 Hadoop 框架上的实现

SimRank 算法能够应用到大规模数据集,原因是它可以在分布式编程模型上,以消息传递的方式实现,这样就可以有效地利用分布式资源,进行大规模数据集上的相似度计算.SimRank 算法在 MapReduce 编程模型上的实现如算法 1 所示.

算法 1. MapReduce 上实现的 SimRank 算法.

输入:图 G,初始值集合 S^0;

输出:相似度值结果集合 S^*.

① for $t=0$ to $T-1$

② mapphase 输入:$\langle key=(a,b), value=S^t(a,b)\rangle$;

③ for each $(c,d)|c\in O(a), d\in O(b)$;

④ 发送$\langle key=(c,d), value=S^t(a,b)\rangle$;

⑤ end for

⑥ reducephase 输入:$\langle key=(c,d), value=vs[\ ]\rangle$;

/ * $vs[\ ]$为相同 key 值的所有 $value$ 值构成的数组 * /

⑦ if $c=d$

⑧ $S^{t+1}(c,d)=1$;

⑨ else

⑩ $S^{t+1}(c,d)=\dfrac{C}{len(vs)}sum(vs)$;

⑪ end if

⑫ 输出$\langle key=(c,d), value=S^{t+1}(c,d)\rangle$;

⑬ $S^t=S^{t+1}$;

⑭ end for

⑮ $S^*=S^{T-1}$;

⑯ 输出 S^*.

MapReduce 编程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计算出自己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 算法存在如下问题:

1) 存储复杂度高. 在 Map 阶段,由于分配的顶点对是任意的,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 任务只能存储整张原始图 G.

2) 计算复杂度高. 由于 SimRank 算法计算的是图中任意 2 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 为原始图中顶点的个数. 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 1 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 2 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 2 次全局同步,即 Shuffle 过程的同步和 Job 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 MapReduce 编程模型上的实现,虽然实现了相似度的计算,却并不能保证计算的高效性,消耗大量的计算资源和时间成本. 接下来,本文将提出一种高效的 SimRank 改进算法,它可以避免 MapReduce 编程模型上实现的 SimRank 算法的若干问题.

2 Asyn-SimRank 算法

本节首先简要地介绍 Delta-SimRnak 算法,然后在 Delta-SimRnak 算法的基础之上提出 Asyn-SimRank 算法,并证明 Asyn-SimRank 算法的收敛性和正确性.

2.1 同步迭代-累积形式:Delta-SimRank

由 1.2 节 SimRank 算法的分布式实现可以看出,传统的 SimRank 算法在迭代计算过程中,所有顶点对都会参加后续的运算. 但是实际上,有很多的顶点对的相似度值已经收敛,在后续的计算过程中不会再变化,继续参加运算会无谓的消耗计算资源.

为了避免收敛的顶点对继续参加运算,Dlta-SimRank 算法[4]对式(1)进行了如下变换.

对于任意的顶点对(a,b),当 $a\neq b$ 时,令:

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

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

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

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

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

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. 这样就可以先迭代计算差值,然后再将这些差值累积起来:

当 $a\neq b$ 时,

$$\begin{cases}\Delta S^{k+1}(a,b)=\dfrac{C}{|I(a)||I(b)|}\sum\limits_{\substack{c\in I(a)\\ d\in I(b)}}\Delta S^{k}(c,d),\\ S^{k+1}(a,b)=\Delta S^{k+1}(a,b)+S^{k}(a,b).\end{cases}\tag{2}$$

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

由式(2)可以看出,Delta-SimRank 的计算过程被分为 2 步:1)迭代. 迭代地计算 2 次迭代的差值;2)累积. 将迭代差值累积起来. 本文将这种迭代过程总结为迭代-累积计算方式. 由于收敛顶点对的相似度值在计算过程中不再变化,因此这些顶点对迭代差值一定为 0,在后续计算中迭代差值为 0 的顶点对不再参加运算,进而减少了计算量和通信量.

2.2 异步迭代-累积形式-Asyn-SimRank

第 2.1 节推导出的式(2)是同步执行的迭代-累积计算方式. 所谓同步形式,就是每次累积的迭代差值 $\Delta S^{k+1}(a,b)$ 恰好是第 $k+1$ 次和第 k 次迭代结果之差. 但是为了获得更高的运行效率,可行的计算方式是让每个顶点对的迭代-累积操作异步地进行. 任意顶点对的迭代-累积操作可以随时执行,不必等待 2 次迭代的差值全部算出来,才开始执行后续操作. 为了达到异步执行的目的,就必须打破同步迭代对每次累积差值大小的限制. 若用 $\Delta P^{k+1}(a,b)$ 表示顶点对(a,b)在第 $k+1$ 次迭代中接收到的差值之和,则有:

$$\Delta P^{k+1}(a,b)=\Delta S_1^{k}+\Delta S_2^{k}+\Delta S_3^{k}+\cdots+\Delta S_p^{k},$$

其中,$\Delta S_i^{k}=\Delta S_i^{k}(c,d)$,$c\in I(a)$,$d\in I(b)$,下标 p 为顶点对(a,b)的指入顶点对的个数,即顶点对(a,b)入度. 在同步方式下可以保证$\{\Delta S_1^{k},\Delta S_2^{k},\cdots,\Delta S_p^{k}\}$全部到达之后才开始后续的操作. 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,在某时刻$\{\Delta S_1^{k},\Delta S_2^{k},\cdots,\Delta S_i^{k}\}$首先到达,而$\{\Delta S_{i+1}^{k},\Delta S_{i+2}^{k},\cdots,\Delta S_p^{k}\}$随后到达,其中 $1\leqslant i\leqslant p$,此时令:

$$\Delta P_1^{k+1}(a,b)=\Delta S_1^{k}+\Delta S_2^{k}+\Delta S_3^{k}+\cdots+\Delta S_i^{k},$$

$$\Delta P_2^{k+1}(a,b)=\Delta S_{i+1}^{k}+\Delta S_{i+2}^{k}+\Delta S_{i+3}^{k}+\cdots+\Delta S_p^{k}.$$

那么就有:

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

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

$$\Delta P_1^{k+1}(a,b)=\Delta S_1^{k}+\Delta S_2^{k}+\Delta S_3^{k}+\cdots+\Delta S_i^{k},$$

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

当 $\Delta P_2^{k+1}(a,b)$到达时:

$$\Delta P_2^{k+1}(a,b)=\Delta S_{i+1}^{k}+\Delta S_{i+2}^{k}+\Delta S_{i+3}^{k}+\cdots+\Delta S_p^{k},$$

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

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

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

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

由此可以发现,在假设的情况下,即使是顶点对(a,b)在计算时所需的差值并没有全部到达就开始接下来的操作,最终仍然可得到正确的结果 $S^{k+1}(a,b)$. 当然这只是在异步计算的实际情况中最简单的一种情况.

为了能够涵盖异步计算过程中所有的情况,下面不再像上例中那样将顶点对(a,b)接收到的差值 ΔS_i 的个数限定在 1 次迭代的 p 个,而是将从计算开始到计算结束顶点对(a,b)接收到的所有值均算在内,假设为 q,q 在理论上趋于无穷. 那么顶点对(a,b)的相似度值异步计算的过程可表示为

$$\begin{cases}\Delta\dot{P}(a,b)\leftarrow\Delta\dot{S}_i+\Delta\dot{S}_{i+1}+\Delta\dot{S}_{i+2}+\cdots+\Delta\dot{S}_j,\\ \Delta\dot{S}(a,b)\leftarrow\dfrac{C}{|I(a)||I(b)|}\Delta\dot{P}(a,b),\\ S(a,b)=S(a,b)+\Delta\dot{S}(a,b),\\ \Delta\dot{P}(a,b)\leftarrow 0,\end{cases}\tag{3}$$

其中,$0\leqslant i\leqslant j\leqslant q$,$\{\Delta\dot{S}_i,\Delta\dot{S}_{i+1},\cdots,\Delta\dot{S}_j\}$表示当前时刻$(a,b)$已经接收到的差值集合. 式(3)就是本文提出的 Asyn-SimRank 算法的计算公式. 该公式所表示的计算方式就是异步计算方式:在任意时刻,每个顶点对都可以利用当前时刻所接收到的差值进行

接下来的运算,而完全不用等待某个差值是否接收到.这就是说整个计算过程不再像 Delta-SimRank 那样,等待 1 次迭代的 p 个差值全部接收到之后才开始后续操作,而是只要接收到了差值就可以开始计算.

但是在本节开始部分只是简单地验证了在 1 次同步迭代中,(a,b) 接收的差值分 2 次到达时,仍然可以得到正确的结果.但是在式(3)中,(a,b) 当前时刻接收到的差值集合为 $\{\Delta\dot{S}_i,\Delta\dot{S}_{i+1},\cdots,\Delta\dot{S}_j\}$,由于在异步计算中完全不限制任何顶点对的计算,各个顶点对的计算异步执行,可能有快有慢,那么就可能出现以下情况:某个顶点对计算得快,已经达到同步的情况下迭代很多次的程度,此时它向 (a,b) 发来 $\Delta\dot{S}_i$.而另一个顶点对可能只达到在同步情况下迭代较少次的程度,此时它向 (a,b) 发来了 $\Delta\dot{S}_{i+1}$.除此之外,发来 $\{\Delta\dot{S}_{i+2},\cdots,\Delta\dot{S}_j\}$ 的顶点对都处于不同的同步迭代程度,而在 (a,b) 上这些差值却同时参加了运算.这种情况下,异步计算最终结果的收敛性和正确性必须有严格的理论保证.这正是 2.3 节所解决的问题.

2.3 收敛性与正确性证明

在证明之前,先简要描述证明的主要思路.实际上无论是 Delta-SimRank 还是 Asyn-SimRank 算法,其主要的计算过程都是将初始差值 ΔS^0 沿所有可走的路径 1 跳 1 跳地传播出去,每经过一个点,就在该点进行计算并被累积.然后再将计算出的新的差值继续向前传.下面通过一个类比实例来形象地描述证明思路.

该计算过程十分类似于加工物品的流水线.Delta-SimRank 是所有顶点对同步地将接收到的差值计算并累积后,将计算后的差值传给邻接点,就好像加工流水线上的每个人每次都加工特定数量的物品之后,等待最慢的人将自己特定数量的物品加工完成,大家再同时将处理好的物品交给下批人进行后续处理.而 Asyn-SimRank 是接收到差值后就马上计算并累积,并将新差值向前传.就好像流水线上的每个人每拿到一个物品就马上加工,完成后马上交给下批人进行后续处理.但是比较这 2 种工作方式,虽然可能有快慢之分,但是 2 种工作方式却是等效的,不管哪种工作方式,每个物品都会一道工序不差地完成处理.

对应到 2 个算法,无论是哪种计算方式,每个顶点对的初始差值 ΔS^0 都会经过路径上的每个点,被计算并累积,并最终都会到达目的顶点对.因此只要 Delta-SimRank 算法收敛,则 Asyn-SimRank 算法收敛,并收敛到相同的值.

接下来,本文将通过定理 1 证明 Asyn-SimRank 算法的收敛性.通过引理 1,引理 2 以及定理 2 来证明 Asyn-SimRank 算法的正确性.

定理 1. 如果在进行无穷次迭代之后 Delta-SimRank 算法收敛,那么异步地进行无穷次迭代-累积操作之后,Asyn-SimRank 算法也收敛.

证明. 文献[4]中给出的 Delta-SimRank 算法是 SimRank 算法的等价推导,等价推导可保证如果 SimRank 算法收敛则 Delta-SimRank 算法一定收敛,并且收敛到相同的值.其等价推导过程在本文 2.1 节已给出.SimRank 算法的收敛性在文献[1]中已经给出,在此不做赘述,设其收敛值为 S^*,那么 Delta-SimRank 算法也收敛,其收敛值也为 S^*.

首先由式(2)可以推导出,经过 k 次迭代之后:

$$\begin{aligned}S^k(a,b)=&(S^0(a,b)+\Delta S^1(a,b))+\\&\sum_{\substack{c_1\in I(a)\\d_1\in I(b)}}f_{(a,b)}(\Delta S^1(c_1,d_1))+\\&\sum_{\substack{c_1\in I(a)\\d_1\in I(b)}}f_{(a,b)}\Big(\sum_{\substack{c_2\in I(c_1)\\d_2\in I(d_1)}}f_{(c_1,d_1)}(\Delta S^1(c_1,d_1))\Big)+\cdots+\\&\sum_{\substack{c_1\in I(a)\\d_1\in I(b)}}f_{(a,b)}\Big(\cdots\sum_{\substack{c_k\in I(c_{k-1})\\d_k\in I(d_{k-1})}}f_{(c_{k-1},d_{k-1})}(\Delta S^1(c_1,d_1))\Big),\end{aligned}\tag{4}$$

其中,$f_{(x,y)}(z)=\dfrac{z}{|I(x)||I(y)|}\times C$.由式(4)可以看出,Delta-SimRank 算法进行 k 次迭代计算后,由于所有的顶点对都会同步完成迭代-累积操作,所以 $S^k(a,b)$ 可以收到所有的以 (a,b) 为终点、距离它跳数小于或等于 k 的顶点对传来的差值.式(4)中的第 m 项就对应以 (a,b) 为终点、距离它为 $m-1$ 跳的顶点对传来的差值.

对于 Asyn-SimRank 算法,为了表示点 (a,b) 接收到的差值,定义时间序列 $t_0,t_1,t_2,t_3,\cdots,t_k$.并定义 $S=\{S_1,S_2,S_3,\cdots,S_k\}$,其中 S_k 表示一个顶点对集,$S_i\subset V^2$,$1\leqslant i\leqslant k$,并且 S_k 中的顶点对发出去的所有差值在时刻 t_{k-1} 到 t_k 时间段内全部被其邻接顶点对接收.

定义时刻 t_k 时任意顶点对 (a,b) 的相似度值为 $S^{t_k}(a,b)$,由式(3)可以得到时刻 t_k 时:

$$S^{t_k}(a,b)=(S^0(a,b)+\Delta S^1(a,b))+\sum_{\substack{c_1\in I'(a)\\ d_1\in I'(b)}} f_{(a,b)}(\Delta S^1(c_1,d_1))+\sum_{\substack{c_1\in I'(a)\\ d_1\in I'(b)}} f_{(a,b)}\Big(\sum_{\substack{c_2\in I'(c_1)\\ d_2\in I'(d_1)}} f_{(c_1,d_1)}(\Delta S^1(c_1,d_1))\Big)+\cdots+\sum_{\substack{c_1\in I'(a)\\ d_1\in I'(b)}} f_{(a,b)}\Big(\cdots\sum_{\substack{c_k\in I'(c_{k-1})\\ d_k\in I'(d_{k-1})}} f_{(c_{k-1},d_{k-1})}(\Delta S^1(c_1,d_1))\Big),\tag{5}$$

其中,$I'(x)\subset I(x)$,时刻 t_k 时,(a,b)最远能够收到距离它 k 跳的顶点对传来的差值.如果(a,b)收到(c_k,d_k) 传来的差值,那么对于路径上的所有 k 个点$(a,b),(a'_1,b'_1),(a'_2,b'_2),\cdots,(c_k,d_k)$一定有$(a,b)\in S_k,(a'_1,b'_1)\in S_{k-1},(a'_2,b'_2)\in S_{k-2},\cdots,(c_k,d_k)\in S_0$.

将式(5)与式(4)相对比,由于 $I'(x)\subset I(x)$,当取 Delta-SimRank 的迭代次数为 k 时,式(5)与式(4)具有相同的项数,并且式(5)中的每项都小于或等于式(4)中对应的项,对于任意顶点对(a,b),显然有 $S^{t_k}(a,b)\leqslant S^k(a,b)$,又由于当 $k\to\infty$时,$S^k(a,b)$收敛,因此有:当 $k\to\infty$时,$S^{t_k}(a,b)$收敛.设其收敛值为 $\dot{S}^*$. 证毕.

引理 1. 对于$\forall k\geqslant 1$,总有 $S^{t_k}(a,b)\leqslant S^k(a,b)$成立.

证明. 在定理 1 的证明中已经得出该结论.

证毕.

引理 2. 当 $k\to\infty$,一定 $\exists k'\geqslant k$,使得 $S^k(a,b)\leqslant S^{t_{k'}}(a,b)$成立.

证明. 对于任意的顶点对(a,b),当 $k\to\infty$,总能够找到这样一个 k',$k\leqslant k'$,使得 $S=\{S_1,S_2,S_3,\cdots,S_{k'}\}$包含所有可走的以$(a,b)$为终点跳数小于或等于 k 的路径,即:(a,b)可以接收到所有距离其跳数小于或等于 k 的顶点对发来的差值.又由于 $k\leqslant k'$,此时(a,b)可能接收到距离它跳数大于 k 的顶点对发来的差值,所以 $S^k(a,b)\leqslant S^{t_{k'}}(a,b)$. 证毕.

定理 2. 如果在进行无穷次迭代之后 Delta-SimRank 收敛,其收敛值为 S^*,那么异步地进行无穷次迭代-累积操作之后,Asyn-SimRank 也收敛,其收敛值为 $\dot{S}^*$,并且有 $S^*=\dot{S}^*$.

证明. 由引理 1 和引理 2 可得:$S^{t_k}(a,b)\leqslant S^k(a,b)\leqslant S^{t_{k'}}(a,b)$,并且当 $k\to\infty$时,$k'\to\infty$.又由定理 1,$S^{t_k}(a,b)$收敛于 $\dot{S}^*$,$S^{t_{k'}}(a,b)$也收敛于 $\dot{S}^*$.由夹逼准则可得:$S^{t_k}(a,b)$与 $S^k(a,b)$收敛到相同的值,即 $S^*=\dot{S}^*$. 证毕.

3 关键点优先调度

本节在 Asyn-SimRank 算法异步计算的基础之上提出关键点优先调度计算,来提升 Asyn-SimRank 算法的全局收敛速度,并证明关键点优先调度计算在提升 Asyn-SimRank 算法全局收敛速度方面的有效性.

3.1 基本思想

在分布式环境下,每个工作节点负责一部分顶点对相似度值的计算,在任意单核工作节点上,每次只能有一个顶点对参加计算,那么就必须按一定的顺序调度不同的顶点对参加计算,最简单的方法就是轮询式的调度.假设在同步计算过程中,顶点对(a,b)在第 k 次迭代时接收到的差值集合为$\{\Delta S_1^k,\Delta S_2^k,\Delta S_3^k,\cdots,\Delta S_p^k\}$,那么同步计算时,这些差值会在$(a,b)$上迭代 1 次得到 ΔS^{k+1},然后将 ΔS^{k+1}累积后再发送出去,显然 1 次迭代-累积操作就可以完成这些运算.

但是在异步计算的过程中,如果集合中$\{\Delta S_1^k,\Delta S_2^k,\Delta S_3^k,\cdots,\Delta S_p^k\}$中的元素分若干次到达,而$(a,b)$恰好被轮询调度了若干次,因为异步计算接到差值就进行计算,这样就有可能在同步时,1 次迭代-累积操作可以完成的计算,在异步计算过程中却要多次,这样无疑增加了计算量,这些增加的计算量会抵消掉异步计算所带来的收益.

简单的解决方法是设置缓存,将接收到的差值缓存到一定数目再参加运算,避免差值到达就直接参加运算,导致迭代-累积操作过于频繁,但是单独地使用缓存有一定的难度,比如如何设定缓存的大小.如果设置得过大,会造成大量的点没有差值计算而等待,增加计算的时间开销.如果设置得过小,显然又不能有效地避免迭代-累积操作过于频繁.

本文提出的方法是关键点优先调度.主要的思想是:异步计算过程中顶点对何时参加运算都是任意的,那么就将计算的机会尽量地分给那些对全局收敛影响较大的点,本文将这样的点称为关键点.这样就可以合理地分配计算资源,使得关键点及时地被调度计算,减少无关紧要的点的计算机会,从而减小计算频率,减少计算量.那么主要的问题是哪些点优先地计算会得到更快的收敛速度,即:如何准确地找到关键点.如果在计算中能够动态地确定关键点,并给这些关键点赋较高的优先级,在异步计算中优先地被调度计算,那么就可以实现关键点优先调度,节省计算量.

本文发现优先计算接收到的差值之和$|\Delta\dot{P}(a,b)|$较大的点可以得到较快的收敛速度,接下来,本文将证明基于$|\Delta\dot{P}(a,b)|$的关键点优先调度计算的有效性.

3.2 理论证明

令向量$\boldsymbol{s}=(\dot{S}_1,\dot{S}_2,\dot{S}_3,\cdots,\dot{S}_m)$为所有的顶点对的相似度值构成的相似度值向量,其中$m$为顶点对的总个数.$\|\boldsymbol{s}\|_1=\sum_{i=1}^{m}|\dot{S}_i|$为向量的1-范数,$\boldsymbol{s}^*=(\dot{S}_1^*,\dot{S}_2^*,\dot{S}_3^*,\cdots,\dot{S}_m^*)$为全局收敛时所有顶点对的相似度值构成的收敛相似度向量.由式(5)可知,采用迭代-累积计算形式,由于每次累计的差值大于0,因此每个顶点对的相似度值是递增的,进而$\|\boldsymbol{s}\|_1$也是递增的,由于收敛,$\|\boldsymbol{s}\|_1$会趋于定值$\|\boldsymbol{s}^*\|_1$.

定理3. 在异步计算过程中,进行相同次数的迭代-累积操作,优先地调度顶点对$(a,b)=\arg\max_{(a,b)}|\Delta\dot{P}(a,b)|$参加计算得到的相似度向量为$\boldsymbol{s}_1$,轮询调度计算得到的相似度值向量为$\boldsymbol{s}_2$,$\|\boldsymbol{s}_1\|_1$更接近收敛点$\|\boldsymbol{s}^*\|_1$即:$\|\boldsymbol{s}^*\|_1-\|\boldsymbol{s}_1\|_1\leqslant\|\boldsymbol{s}^*\|_1-\|\boldsymbol{s}_2\|_1$.

证明. 在异步计算的过程中,将$|\Delta\dot{P}(a,b)|$作为该顶点对的优先级,即:$|\Delta\dot{P}(a,b)|$越大的点,越优先地进行迭代-累积计算.假设关键点优先调度和轮询调度从计算开始到当前时刻都进行了k次迭代-累积操作.关键点优先调度产生的新差值向量为$\Delta\boldsymbol{s}=(\Delta\dot{S}_1,\Delta\dot{S}_2,\Delta\dot{S}_3,\cdots,\Delta\dot{S}_m)$,如果顶点对$i$没有进行迭代-累积操作,则$\Delta\dot{S}_i=0$,如果进行了多次,则$\Delta\dot{S}_i$为多次累积的差值之和.轮询调度计算产生的新差值向量为$\Delta\boldsymbol{s}'=(\Delta\dot{S}'_1,\Delta\dot{S}'_2,\Delta\dot{S}'_3,\cdots,\Delta\dot{S}'_m)$.由于在关键点优先调度计算过程中,总是选择顶点对$(a,b)=\arg\max_{(a,b)}|\Delta\dot{P}(a,b)|$优先调度,显然有$\|\Delta\boldsymbol{s}\|_1\geqslant\|\Delta\boldsymbol{s}'\|_1$,因为向量$\Delta\boldsymbol{s}$中的元素总是选择该时刻最大的差值,$\Delta\boldsymbol{s}'$中的元素则是轮询式选择的.那么进行累积之后,$\|\boldsymbol{s}_1\|_1=\|\boldsymbol{s}\|_1+\|\Delta\boldsymbol{s}\|_1$,$\|\boldsymbol{s}_2\|_1=\|\boldsymbol{s}\|_1+\|\Delta\boldsymbol{s}'\|_1$,显然有$\|\boldsymbol{s}_1\|_1\geqslant\|\boldsymbol{s}_2\|_1$,进而得到$\|\boldsymbol{s}^*\|_1-\|\boldsymbol{s}_1\|_1\leqslant\|\boldsymbol{s}^*\|_1-\|\boldsymbol{s}_2\|_1$. 证毕.

4 Asyn-SimRank算法的分布式实现

本节将结合分布式环境的特点详细地阐述Asyn-SimRank算法如何在分布式环境下实现.

4.1 输入图的预处理

SimRank算法计算的是2顶点的相似程度,但很多以顶点为中心的分布式框架,由于只支持在图顶点上运行更新操作,所以不能直接进行顶点关系的计算.如果利用2.1节提到的将原始图G构造成图G^2的方式,将顶点间关系的计算转化为顶点对上相似度值的计算,就可以在以顶点为中心的分布式框架上实现SimRank算法.

这种方式有诸多的优点:1)消息恰好是按图G^2的结构传递的,在进行数据发送的时候只需用到图G^2中顶点对的邻接表,那么集群中的每个计算节点只需存储所处理的顶点对的邻接表,不再需要存储整张原始图;2)可以利用1.1节所提到的删减方式,对图G^2中无用的边和点进行删减,可以有效地减少计算量.

4.2 分布式环境上的实现

为了能够发挥出Asyn-SimRank算法异步计算和关键点优先调度计算的优势,必须选取合适的异步分布式框架来实现Asyn-SimRank算法.Maiter框架[5]支持异步计算,并且可以在计算的过程中动态地设定顶点的优先级,并进行优先级调度,这为实现Asyn-SimRank算法的异步计算和关键点优先调度计算提供了条件.

Maiter是Master-Slave结构的分布式框架.包含一个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.

在Maiter框架上实现Asyn-SimRank算法时,每个Slave节点需要完成的工作就是迭代-累积操作.在任务开始时,Master节点会将相应的数据分片partition分配给Slave节点,其中包含顶点对、顶点对的邻接表、相似度初值、迭代差值初值等信息.Slave节点为partition中的每个顶点对保存3个变量:唯一标识K、相似度累积值V、迭代差值ΔV.如果该Slave节点的partition中有m个顶点对,那么就会创建一个长度为m的3元组数组来存储所有顶点对的K,V和ΔV.在构建的同时,Slave节点会用partition中的信息,初始化这些变量.

图2所示的是Maiter上实现的Asyn-SimRank算法的运算过程.Slave1节点和Slave2节点用各自ΔV中的值($\Delta\dot{P}$)计算出要发送的差值$\Delta\dot{S}_1$,$\Delta\dot{S}_2$并发送.Slave3节点则接收Slave1和Slave2发来的$\Delta\dot{S}_1$,$\Delta\dot{S}_2$,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$\Delta\dot{S}_3$,发送到其他的Slave节点,然后将ΔV变量清0.

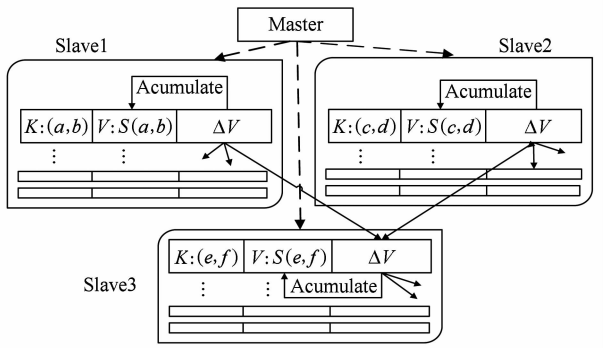


Fig. 2 The implementation of Asyn-SimRank on Maiter.

图 2 Maiter 上 Asyn-SimRank 的实现

下面给出算法 2,详细地阐述 Asyn-SimRank 算法的分布式实现方法.必须指出的是,每个 Slave 节点都会执行算法 2,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.

算法 2. 分布式 Asyn-SimRank 算法.

输入:图 G^2;

输出:顶点对相似度值.

① while not converge

② for each (a,b) in partition / * partition 为数据分片 * /

③ if ($\Delta V==0 \| !checkpri()$)

④ continue;

⑤ end if

⑥ 接受其他节点法来的 $\Delta\dot{S}_i$ 累积到 ΔV;

⑦ for each $(c,d)\in O(a,b)$

⑧ 发送 $C\times\Delta V/|I(a)||I(b)|$ 给(c,d);

⑨ end for

⑩ $V=V+C\times\Delta V/|I(a)||I(b)|$;
/ * 累积操作 * /

⑪ $\Delta V=0$;

⑫ end for

⑬ end while

⑭ 将所有 V 值输出.

算法 2 的行①判断算法是否收敛.行③的函数 *checkpri*()用于检测记录的优先级是否满足要求,满足要求返回 1.行②~⑫负责对 partition 中的每个顶点对进行迭代-累积操作.在进行操作之前,会先判断 ΔV 是否为 0,如果为 0 则说明该顶点已经收敛或还没有其他的顶点对传来的差值,此时迭代-累积操作无意义.除此之外还会判断该顶点对的优先级,优先级达不到特定值则不进行迭代-累积操作,这就有效的利用了关键点优先调度计算来加快算法的收敛速度.

本文改进了分布式异步计算框架 Maiter 并对 Maiter 的 API 进行了扩充,可以在 Maiter 上高效地实现 Asyn-SimRank 算法.

5 实 验

为了验证本文提出的 Asyn-SimRank 算法在通信量、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,Spark 框架上实现 SimRank 算法、delta-SimRank 算法,在异步分布式框架 Maiter 上实现 Asyn-SimRank 算法,比较 3 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,D-SR 分别代表 SimRank 算法和 Delta-SimRank 算法, A-SR-UP 代表轮询调度计算的 Asyn-SimRank 算法,A-SR-P 代表关键点优先调度计算的 Asyn-SimRank 算法. Asyn-SimRank 算法均在 Maiter 框架上实现.

5.1 实验数据集及实验环境

本实验所选用的所有数据均出自斯坦福大学的大规模网络数据集[①](Stanford large network dataset collection).该数据集中的所有数据均是实际应用中产生的数据.选取实际的数据集运行算法可以有力说明算法在实际应用中运行的真实情况.实验中选择的 5 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路 ca-CondMat、文件共享网络 P2P.数据集的相关信息如表 1 所示:

Table 1 Summary of the Experimental Datasets

表 1 实验数据集相关信息

Dataset	Vertices	Edges	Diameter	Node-Pairs
ca-AstroPh	2 403	9 852	10	2 888 406
ca-GrQc	5 242	28 980	17	13 741 903
ca-HepTh	9 877	51 971	17	48 782 503
ca-CondMat	15 014	61 557	14	112 717 605
P2P	22 687	54 705	10	257 361 328

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

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、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:

Table 2 Configuration of Computation Node

表 2 节点软硬件配置

Component	Configureation
CPU	Intel Core i3-2100 Lga-1155 4 Slots
Memory	Apacer 4 GB-DDR3×2
Disk	Hitachi 500 GB/7200RPM
Network	1 000 Mbps Ethernet
Opertor	Redhat 6.1 64 b
Hadoop	Release 1.0.4
Maiter	Release 0.1

5.2 总体运行时间对比

本实验在不同的同步框架上实现了同步的 SimRank 算法和 Delta-DimRank 算法,与异步框架 Maiter 上实现的 Asyn-SimRank 算法在运行时间上进行了比较.

图 3 所示的是 3 个算法在不同的数据集上运行时间的对比. 可以看出 Asyn-SimRank 算法总体计算时间明显缩短. Asyn-SimRank 算法最快能比 Hadoop 上实现的 SimRank 算法快 75.3 倍,比 Hadoop 上实现的 Delta-SimRank 算法最快可快 64.7 倍,运行速度提升明显,说明了 Asyn-SimRank 算法的在计算速度方面的优势.

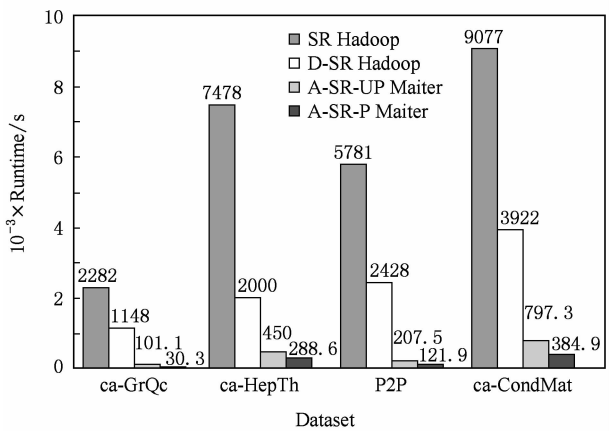


Fig. 3 Runtime of algorithms implemented on Maiter and Hadoop.

图 3 Maiter 和 Hadoop 实现的算法的运行时间对比

由于目前很多 Hadoop 的改进框架如 Spark[6] 等可以高效地支持迭代运算,在这些改进框架上实现 SimRank 和 Delta-SimRank 算法,也可以有效提升运算效率,为了说明 Asyn-SimRank 算法的在运行速度上的优势,本实验将 Maiter 上实现的 Asyn-SimRank 算法与 Spark 上实现的 SimRank 算法和 Delta-SimRank 算法进行了性能对比,实验中使用的数据集为 ca-AstroPh,实验结果如图 4 所示. 可见,Asyn-SimRank 算法仍具有明显的优势.

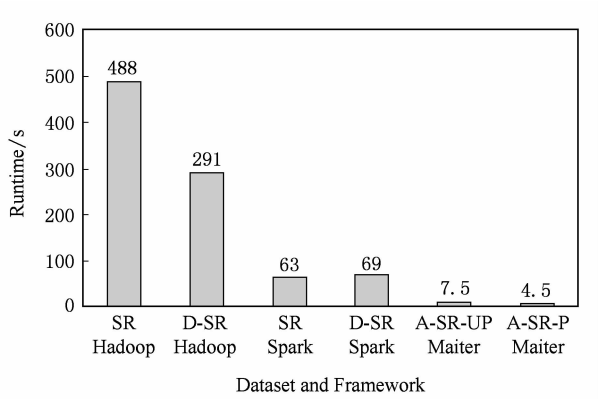


Fig. 4 Runtime of algorithms implemented on Maiter and Spark.

图 4 Maiter 和 Spark 实现的算法的运行时间对比

以上实验说明 Maiter 框架上实现的 Asyn-SimRank 算法比主流同步类型的框架 Hadoop, Spark 上实现的 SimRank 算法和 Delta-SimRank 算法的计算效率都要高. 但是不同算法在不同的框架上实现不能屏蔽框架的差异对实验结果的影响. 然而 Maiter 框架是异步类型的框架,同步类型的算法 SimRank 算法、Delta-SimRank 算法不能在 Maiter 上实现.

为了尽可能屏蔽框架差异对实验结果的影响,本文在 Maiter 框架的基础之上,人为增加了同步路障,以保证同步算法的正确性,实现了 Maiter-syn① 同步框架,可以实现 Delta-SimRank 算法. 本实验在 3 个不同的数据集上对 Maiter-syn 上实现的 Delta-SimRank 算法和 Maiter 上实现的 Asyn-SimRank 算法进行了性能对比,实验结果如图 5 所示. Asyn-SimRank 算法运算速度较快.

5.3 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. 为了能够公平地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速度

① http://code.google.com/Maiter/

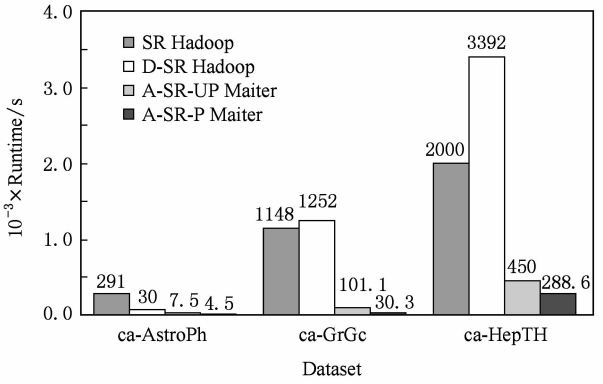


Fig. 5 Runtime of algorithms implemented on Maiter and Maiter-syn.

图 5 Maiter 和 Maiter-syn 实现的算法的运行时间对比

随顶点更新总次数的变化来衡量算法的收敛速度.所谓顶点更新总次数是指从算法开始到某时刻,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算法收敛越快.

图 6 与图 7 为 3 个算法的收敛速度对比.从图 6,7 中可以看出,没有采用关键点优先调度计算的 Asyn-SimRank 算法的收敛速度比 Sim-Rank 好,但是并不比 Delta-SimRank 好,主要原因是因为在没有采用关键点优先调度计算的情况下,Asyn-SimRank 算法会和 Delta-SimRank 算法一样,会对所有迭代差值非 0 的顶点更新,但是 Asyn-SimRank 是异步更新,累积差值的大小可以是任意的,所以累积的差值要比 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,在运行速度上没有使用关键点优先调度计算的 Asyn-SimRank 算法还是远远快于 Delta-SimRank,主要得益于异步计算.

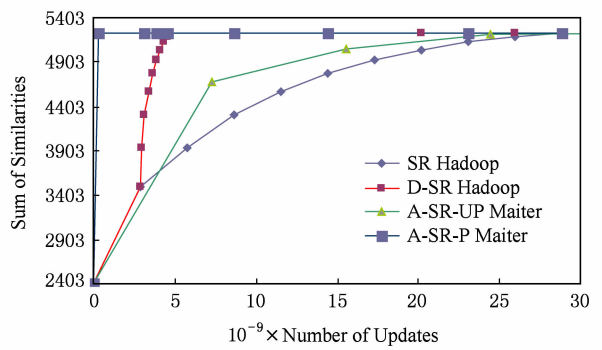


Fig. 6 Comparison of convergence speed on ca-AstroPh.

图 6 ca-AstroPh 数据集上收敛速度对比

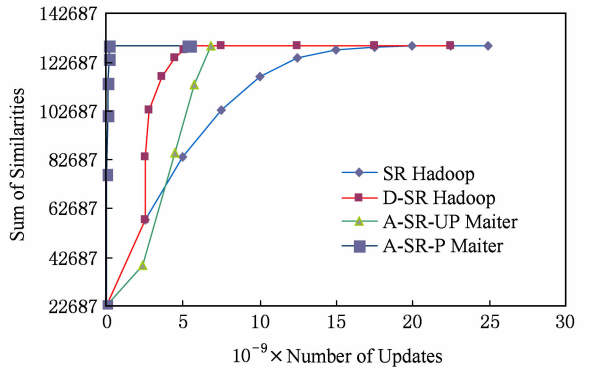


Fig. 7 Comparison of convergence speed on P2P.

图 7 P2P 数据集上收敛速度对比

关键点优先调度计算的 Asyn-SimRank 算法的收敛速度远远比 SimRank 和 Delta-SimRank 算法快.在关键点优先调度计算过程中,Asyn-SimRank 将更新全部作用于那些对算法全局收敛影响明显的顶点对上,让影响尽快地传播出去.并且让重要的或收敛慢的顶点先进行运算,有效提升了算法收敛的速度.

5.4 通信量对比

图 8 所示的是各算法在不同的数据集上通信量的对比.从图 8 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 算法的通信量最大差距可达到 51 倍,和 Delta-SimRank 算法的通信量差距最大可达 15 倍.Asyn-SimRank 算法的通信量小的原因主要是因为,通过收敛顶点和相似度值暂时不变顶点的迭代差值为 0 的原理,节省了这些顶点的计算.由于 SimRank 算法本身的性质,总是大部分顶点对提前收敛,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.

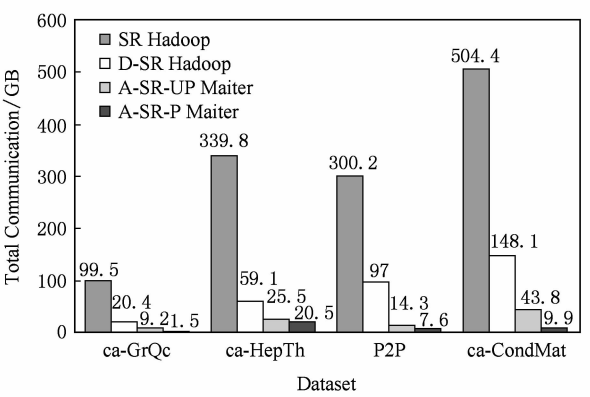


Fig. 8 Comparison of communication cost.

图 8 通信量对比

5.5 算法与分布式环境规模的关系

为了充分验证 Asyn-SimRank 算法能否适应大规模的分布式环境和处理大规模的数据,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 16 台、24 台、32 台计算节点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 9 所示:

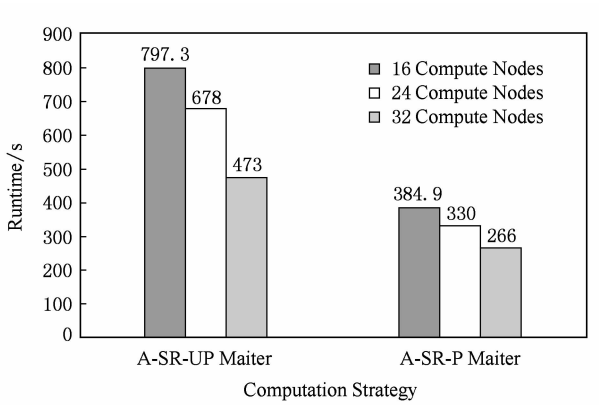


Fig. 9 Scaling performance of Asyn-SimRank algorithm.

图 9 Asyn-SimRank 算法的扩展性

由图 9 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,可以部署在分布式环境上处理大规模的数据.

6 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,因而该算法被广泛应用于各个领域. 随着对 SimRank 算法应用需求的不断提升,很多工作都集中于相似度的计算性能.

随着分布式计算技术的不断发展,很多可以高效支持迭代算法的分布式框架被提出. Spark[6] 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[7] 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地支持迭代. 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,可以一定程度上提高运算速度. 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 算法[4] 采用了迭代-累积计算方式. 这种计算方式可以利用收敛的图顶点迭代差值为 0 的特性,避免已收敛的顶点对参加运算,减小了计算开销、降低了通信量. 同时 Delta-SimRank 算法也为本文 Asyn-SimRank 算法的提出提供了基础. 但是 Delta-SimRank 算法是基于 MapReduce 编程模型的优化算法,仍然是一种同步形式的算法,每个计算节点要存储整张图和所有顶点的入度,每次迭代-累积操作要用 2 个 Job 完成等等.

还有相关工作如文献[10-11]也对 SimRank 算法进行了改进,降低了算法的复杂度,在计算速度和对大数据的支持方面都有了很大的提升. 文献[12] 提出 S-SimRank 算法,通过将文档的内容和链接信息结合在一起的方法,提高的相似度计算的效率和准确度. 这些改进都是针对于单机上的相似度值计算,不是在分布式环境下的改进. Asyn-SimRank 算法是一种分布式算法,主要针对分布式环境进行优化,在单机环境下,Asyn-SimRank 算法无法发挥出异步计算的优势. 因此,本文的实验中没有与这些算法进行对比.

与以上的相关工作不同的是,本文采用异步计算以及关键点优先调度策略来改进 SimRank 算法,有效地提升了 SimRank 算法的计算速度和全局收敛速度,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. 为高效地实现 SimRank 算法提供了有力的途径.

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. Asyn-SimRank 算法为计算图顶点间相似度提供了高效的方法.

Asyn-SimRank 算法的提出还进一步说明,迭代-累积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. 很多其他算法可以变换成这种形式. 所

以迭代-累积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] Jeh G, Widom J. Simrank: A measure of structural-sontext similarity [C] //Proc of the 8th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2002: 538-543

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

[3] Shvachko K, Kuang H, et al. The hadoop distributed file system [C] //Proc of the 2010 IEEE 26th Symp on Mass Storage Systems and Technologies. New York: ACM, 2010: 1-10

[4] Cao L, Cho B, Tsai M, et al. Delta-SimRank computing on mapreduce [C] //Proc of the 1st Int Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms Systems Programming Models and Applications. New York: ACM, 2012: 28-35

[5] Zhang Yanfeng, Gao Qixin, et al. Accelerate large-scale iterative computation through asynchronous accumulative updates [C] //Proc of the 3rd Workshop on Scientific Cloud Computing. New York: ACM, 2012: 13-22

[6] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets [C] //Proc of the 2nd USENIX Conf on Hot Topics in Cloud Computing. Berkeley: USENIX Association, 2010: 10-10

[7] Bu Y, Howe B, Balazinska M, et al. Haloop: Efficient iterative data processing on large clusters [J]. VLDB Endowment, 2010, 3(1/2): 285-296

[8] Zhang Yanfeng, Gao Qixin, et al. IMapReduce: A distributed computing framework for iterative computation [C] //Proc of the 2011 IEEE Int Symp on Parallel and Distributed Processing. Los Alamitos, CA: IEEE Computer Society, 2011: 1112-1121

[9] Zhang Yanfeng, Gao Qixin, et al. PrIter: A distributed framework for prioritized iterative computations [C] //Proc of the 2nd ACM Symp on Cloud Computing. New York: ACM, 2011: 13-14

[10] Zhang Yinglong, Li Cuiping, et al. Fast simrank computation over disk-resident graphs [C] //Proc of the 18th Int Conf on Database Systems for Advanced Applications. Berlin: Springer, 2013: 16-30

[11] Yu Wweiren, Lin Xuemin, et al. Towards efficient simrank computation on large networks [C] //Proc of the 29th Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2013: 230-241

[12] Cai Yuanzhe, Li Pei, Liu Hongyan, et al. S-SimRank: Combining content and link information to cluster papers effectively and efficiently [J]. Journal of Frontiers of Computer Science and Technology. 2009, 3(4): 378-391 (in Chinese)
(菜元哲, 李佩, 刘红岩, 等. S-SimRank: 结合内容和链接信息的文档相似度计算方法[J]. 计算机科学与探索, 2009, 3(4): 378-391)

Wang Chunlei, born in 1989. Master candidate in the College of Information Science and Engineering, Northeastern University. Student member of China Computer Federation. His main research interests include big data and cloud computing.

Zhang Yanfeng, born in 1982. Associate professor at Computing Center, Northeastern University. Member of China Computer Federation. His main research interests include big data and cloud computing (zhangyf@cc.neu.edu.cn).

Bao Yubin, born in 1968. Professor and PhD in the College of Information Science and Engineering, Northeastern University. Senior member of China Computer Federation. His main research interests include data warehouse, OLAP, cloud computing, data intensive computing (baoyubin@mail.neu.edu.cn).

Zhao Changkuan, born in 1976. Lecturer at Computing Center, Northeast University. Member of China Computer Federation. His main research interests include social networks, distributed system, cloud computing(zck@cc.neu.edu.cn).

Yu Ge, born in 1962. Professor and PhD supervisor in the College of Information Science and Engineering, Northeastern University. Senior member of China Computer Federation. His main research interests include databases, distributed systems and embedded systems(yuge @mail.neu.edu.cn).

Gao Lixin, born in 1968. Professor in the Department of Electrical and Computer Engineering, University of Massachusetts Amherst. ACM and IEEE Fellow. Her main research interests include Internet routing and cloud computing.