软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1090−1106 [doi: 10.13328/j.cnki.jos.007072]

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

GPPR: 跨域分布式个性化 PageRank 算法*

陈子俊[1], 马德龙[1], 王一舒[1], 袁 野[2]

[1](东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

[2](北京理工大学 计算机学院, 北京 100081)

通信作者: 袁野, E-mail: yuanye@mail.neu.edu.cn

摘 要: 个性化 PageRank 作为大图分析中的基本算法, 在搜索引擎、社交推荐、社区检测等领域具有广泛的应用, 一直是研究者们关注的热点问题. 现有的分布式个性化 PageRank 算法均假设所有数据位于同一地理位置, 且数据所在的计算节点之间具有相同的网络环境. 然而在现实世界中, 这些数据可能分布在跨洲的多个数据中心中, 这些跨域分布(cross-geo-distributed)的数据中心之间通过广域网连接, 存在网络带宽异构、硬件差异巨大、通信费用高昂等特点. 分布式个性化 PageRank 算法需要多轮迭代, 并在全局图上进行随机游走. 因此, 现有的分布式个性化 PageRank 算法不适用于跨域环境. 针对此问题, 提出了 GPPR (cross-geo-distributed personalized PageRank) 算法. 该算法首先对跨域环境中的大图数据进行预处理, 采用启发式算法映射图数据, 以降低网络带宽异构对算法迭代速度的影响; 其次, GPPR 改进了随机游走方式, 提出了基于概率的 Push 算法, 通过减少工作节点之间传输数据的带宽负载, 进一步减少算法所需的迭代次数. 基于 Spark 框架实现了 GPPR 算法, 并在阿里云中构建真实的跨域环境, 在 8 个开源大图数据上, 与现有的多个代表性分布式个性化 PageRank 算法进行了对比实验. 结果显示, GPPR 的通信数据量在跨域环境中比其他算法平均减少 30%. 在算法运行效率方面, GPPR 比其他算法平均提升了 2.5 倍.

关键词: 跨域分布式; 个性化 PageRank; 近似计算

中图法分类号: TP301

中文引用格式: 陈子俊, 马德龙, 王一舒, 袁野. GPPR: 跨域分布式个性化 PageRank 算法. 软件学报, 2024, 35(3): 1090–1106. http://www.jos.org.cn/1000-9825/7072.htm

英文引用格式: Chen ZJ, Ma DL, Wang YS, Yuan Y. GPPR: Cross-geo-distributed Personalized PageRank Algorithm. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1090−1106 (in Chinese). http://www.jos.org.cn/1000-9825/7072.htm

GPPR: Cross-geo-distributed Personalized PageRank Algorithm

CHEN Zi-Jun[1], MA De-Long[1], WANG Yi-Shu[1], YUAN Ye[2]

[1](School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

[2](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Personalized PageRank, as a basic algorithm in large graph analysis, has a wide range of applications in search engines, social recommendation, community detection, and other fields, and has been a hot problem of interest to researchers. The existing distributed personalized PageRank algorithms assume that all data are located in the same geographic location and the network environment is the same among the computing nodes where the data are located. However, in the real world, these data may be distributed in multiple data centers across continents, and these cross-geo-distributed data centers are connected to each other through WANs, which are characterized

* 基金项目: 国家重点研发计划(2022YFB2702100); 国家自然科学基金(61932004, 62225203, U21A20516); 中央高校基本科研业务专项资金(N232405-16)

本文由“面向多模态数据的新型数据库技术”专题特约编辑彭智勇教授、高云君教授、李国良教授、许建秋教授推荐.

收稿时间: 2023-07-17; 修改时间: 2023-09-05; 采用时间: 2023-10-24; jos 在线出版时间: 2023-11-08

CNKI 网络首发时间: 2023-12-25

by heterogeneous network bandwidth, huge hardware differences, and high communication costs. The distributed personalized PageRank algorithm requires multiple iterations and random wandering on the global graph. Therefore, the existing distributed personalized PageRank algorithms are not applicable to the cross-geo-distributed environment. To address this problem, the GPPR (cross-geo-distributed personalized PageRank) algorithm is proposed in this study. The algorithm first preprocesses the big graph data in the cross-geo-distributed environment and maps the graph data by using a heuristic algorithm to reduce the impact of network bandwidth heterogeneity on the iteration speed of the algorithm. Secondly, GPPR improves the random wandering approach and proposes a probability-based push algorithm to further reduce the number of iterations required by the algorithm by reducing the bandwidth load of transmitting data between working nodes. The GPPR algorithm is implemented based on the Spark framework and a real cross-geo-distributed environment in AliCloud is built to conduct experiments on eight open-source big graph data compared with several existing representative distributed personalized PageRank algorithms. The results show that the communication data volume of GPPR is reduced by 30% on average in the cross-geo-distributed environment compared with other algorithms. In terms of algorithm running efficiency, GPPR improves by an average of 2.5 times compared to other algorithms.

Key words: cross-geo-distributed; personalized PageRank; approximate calculation

图是表达对象与对象之间关系的基本数据结构, 也是对现实世界中对象及其关系的一种抽象[1]. 一方面, 在大数据时代, 当人们意识到数据蕴含的巨大价值以后, 对图数据的分析需求变得越来越迫切, 如网页推荐、生物基因分析以及信息检索等[2]; 另一方面, 随着互联网及物联网的应用范围持续扩大, 图数据的规模也在不断扩张[3], 传统的图分析算法在这样蕴含大量信息的大图数据上产生了各方面的局限, 如节点间通信成本高、计算资源或存储容量不足. 因此, 如何高效地对真实大图数据进行分析, 并获取数据背后的价值信息, 一直是大图数据分析中的研究热点.

个性化 PageRank 算法(personalized PageRank)是图搜索和分析领域的基本算法, 是 PageRank 算法的一个重要变体. PageRank 算法最初由 Brin 和 Page 在 1998 年提出, 并引入了 Google 的 Web 搜索引擎[4], 作为其引擎内部用于网页推荐的基础算法. 与传统在数据提取领域用的模式匹配算法不同, PageRank 算法完全依赖于基础的图数据来确定图中节点的重要性程度[5]. 所以, PageRank 算法可以给出图上任意一个节点在全局环境下的重要性分数, 进而可以根据重要性分数的高低得到该节点相对全图的重要性程度, 在文献计量学、社交网络分析、道路交通网络分析等领域有广泛的实际应用[6], 是现代搜索服务的基础算法. 个性化 PageRank 算法是 PageRank 算法的一个重要变体, 个性化 PageRank 算法与 PageRank 算法不同体现在: 个性化 PageRank 算法给出的图上每一个节点的重要性分数是相对某一特定节点, 而不是针对图数据中所有节点的, 是一种个性化的重要性程度. 与 PageRank 算法类似, 个性化 PageRank 算法也在诸多领域有广泛的应用, 如信息提取、推荐系统、知识发现等[7].

现有的个性化 PageRank 算法主要包括两大类,分别是单机和分布式. 单机个性化 PageRank 算法进一步可以概括为两个研究方向: 矩阵迭代和本地推送[8]. 采用矩阵迭代的方式进行计算, 算法迭代时间随着数据量的增加成指数级增长, 不适用于现在的大图数据; 本地推送算法计算得到的结果的是个性化 PageRank 算法的无偏估计, 不是个性化 PageRank 算法的准确值, 但由于应用时需要的往往只是节点重要性程度间的比较, 因此, 本地推送算法可以在一定误差界限下极大地减少算法的运行时间. 分布式个性化 PageRank 算法通过在更改本地推送算法以适用分布式环境, 并进一步增加蒙特卡洛模拟来减少算法的运行时间, 可以通过增加计算节点的方式, 利用算法的可扩展性来加快算法迭代, 减少算法的运行时间. 但是现有的分布式个性化 PageRank 算法大都假设所有计算节点之间的带宽是相同的, 这样的假设在跨域环境下并不成立, 由此会产生诸多问题. 如某个数据中心节点传输效率低拖慢算法整体迭代速度, 从而导致算法出现单点瓶颈问题[9].

用户数据产生于世界各地的服务集群, 如 Amazon 现在在全球范围内有 15 个集群[10], Bing 提供的搜索来自 8 个不同地理分布的地区[11]. 由于服务集群间数据传输成本过高, 单机房难以处理大量数据, 因此很少会将数据传输到同一机房来计算, 所以为了低成本并高效地计算个性化 PageRank 算法, 跨国公司会将数据分散在地理分布的各个数据中心内.

Google 依托超过 20 个地理分布的数据中心, 通过内部优化过的 PageRank 算法来提供搜索服务[12]等. 在

这样的跨域环境下, 相对于分布式环境来说有以下几方面的局限: (1) 由于大图数据的数据量级过高, 通常有数十亿个节点和数万亿条边[13,14], 并且跨域环境下的网络传输效率低, 所以难以将数据传输到单一主机上进行汇总处理; (2) 由于地理分布的数据中心之间使用的广域网带宽通常是数据中心内部局域网带宽的千分之一[15], 但是价格却是局域网的数倍, 个性化 PageRank 算法通常会进行多轮迭代, 因此产生高昂的数据传输成本; (3) 由于不同数据中心之间的带宽是异构的, 同一数据中心的上传和下载带宽也是不同的[16], 所以个性化 PageRank 算法通常会因为某个数据中心的低传输效率, 拖慢算法的整体传输效率, 进而遭遇单点性能瓶颈问题. 如表 1 所示, 3 个地区的下载带宽比上传带宽高几倍, 不同数据中心之间的上传和下载带宽都是不同的, 新加坡的上传和下载带宽分别比悉尼的高 17%和 40%.

表 1　3 个地区的 Amazon EC2 实例的上传/下载带宽(GB/s)

带宽	美国	新加坡	悉尼
上行带宽	0.52	0.55	0.48
下载带宽	2.8	3.5	2.5

基于以上挑战, 本研究提出了跨域个性化 PageRank 算法(cross-geo-distributed personalized PageRank, GPPR), 解决跨域环境下分布式个性化 PageRank 算法的相关问题, 主要工作如下.

(1) 首次从跨域环境下带宽异构的角度出发, 提出了算法数据的预处理步骤, 通过基于节点度数和带宽的启发式算法来确定数据和工作节点的对应关系, 可以有效地解决跨域环境下的带宽异构引发的算法单点瓶颈问题, 弥补了现有分布式个性化 PageRank 算法跨域环境下的不足.

(2) 提出了基于概率的 Push 算法, 结合蒙特卡罗模拟来计算节点的个性化 PageRank 值, 减少个性化 PageRank 算法所需要的迭代次数, 从而降低跨域环境下带宽使用成本, 并支持水平扩展, 可以通过增加工作节点来满足大图数据下的计算需求.

(3) 通过搭建真实的跨域环境, 在 5 个真实数据集上, 通过和现有分布式个性化 PageRank 算法进行多组对比实验, 进一步验证和分析了算法的有效性.

本文首先简要介绍个性化 PageRank 算法和跨域数据处理的相关工作(第 1 节); 并给出必要的基本概念与问题定义(第 2 节); 然后介绍基于带宽和节点度数的启发式算法, 用于算法的数据预处理阶段, 解决算法后续迭代计算可能遇到的单点瓶颈问题(第 3 节); 进一步提出基于概率的 Push 算法, 保证结果在一定误差范围的前提下, 减少算法所需要的迭代次数, 降低算法所需要的数据传输量, 并给出结果正确性的相关证明(第 4 节); 为了说明算法的性能, 在 5 个真实数据集上对算法进行实验评估和结果分析(第 5 节); 最后, 对全文整体工作进行总结(第 6 节).

1　相关工作

本节主要针对与单机和分布式环境下个性化 PageRank 算法以及跨域环境下的数据处理密切相关的工作进行简要介绍与总结.

1.1　个性化PageRank算法

1.1.1　单机个性化 PageRank 算法

个性化 PageRank 算法主要有以单一源节点为基准(简称单源节点)、以单一目的节点为基准(简称单目的节点)两大类. 单源节点类别意味着求图中所有节点针对某一个选定源节点的重要性程度, 可以抽象成一对多的关系; 而单目的节点类别则是求选取的目的节点对图中所有其他节点的重要性程度, 是多对一的关系[17,18], 本研究主要研究单源节点条件下的个性化 PageRank 算法相关问题.

个性化 PageRank 算法默认的解决方案是通过矩阵迭代的方式进行计算, 通过个性化 PageRank 的定义推导出的矩阵公式来进行计算, 如公式(1)所示.

$$\vec{r} = \alpha P\vec{r} + (1-\alpha)\vec{s} \tag{1}$$

其中, P 为转移矩阵, P 矩阵定义如公式(2)所示.

$$p_{ij}^{T}=\begin{cases}\dfrac{1}{|\mathrm{O}(n_i)|}, & j\in \mathrm{O}(n_i)\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

α为随机游走的终止概率, 当等式收敛时, $\vec{r}$ 的每个值为对应节点相对于源节点 s 的个性化 PageRank 值. 根据值的大小, 可以得到对于源节点 s 相对更重要的那些点. 但是, 此算法在大图数据场景下的计算效率并不高. 因此, 后续一些工作主要关注如何提高矩阵的计算速度来加快单机环境下的个性化 PageRank 算法的迭代效率[19−21].

1.1.2 分布式个性化 PageRank 算法

分布式环境下的个性化 PageRank 算法有两个主要分支. 一方面, 一些研究通过本地推送的方式来解决个性化 PageRank 算法计算的问题, 如 Forward Push 算法[22]和 Backward Push 算法[23,24]; 另一方面, 有很多工作将 Forward Push 方法和蒙特卡洛方法结合来提高算法的运行效率[25,26]. 其中: Bahmani 提出了 Doubling 算法[27], 通过合并短的随机游走分片组成新的更长的随机游走分片来加快算法速度; Lin 通过引入 pipeline 机制来提高算法的效率[28]; Sarma 主要关注如何在拥塞模型上进行优化[29]; Massively 主要通过引入预采样算法来增加随机游走的复用[30]. 但是这些算法都默认工作节点之间的带宽是相同的, 没有考虑跨域环境下的相关问题.在跨域环境下, 工作节点之间带宽是异构的, 相同的数据传输量会因带宽不同而有相异的传输时间, 进而导致算法出现单点瓶颈问题.

Forward-Push[26]算法如算法 1 所示, Forward-Push 算法通过读取图数据 $G=(V,E)$、源节点 s、一个随机游走的结束概率α和一个残留量阈值 $r_{\max}$. 对于图中的每一个节点 v, 算法为其维护一个残留量 $r_s(v)$和一个储备量 $\hat{\pi}_s(v)$, 其中, 储备量就是图上随机游走在当前节点停止的部分, 残留量就是图上随机游走在当前节点还未结束的部分. 当所有节点的残留量都为 0 时, 节点的储备量就是改节点的准确 *PPR* 值. 在算法的初始化部分, 首先将源节点的残留量设置为 1, 其余节点的残留量都设置为 0, 而所有节点的储备量都设置为 0 (算法 1 的第 1、2 行). 从 s 开始激活随机游走, 对于满足条件 $r_s(v)/|N_{out}(v)|>r_{\max}$ 的所有节点 v, 将 v 节点残留量 $r_s(v)$的α加到 v 节点的储备量, 代表到达 v 的随机游走中有 $r_s(v)\cdot\alpha$停止在 v 节点; 随机游走中的另外 $r_s(v)\cdot(1-\alpha)$部分将继续进行, 向 v 节点的每一个出度邻居节点传递$(1-\alpha)\cdot r_s(v)/N_{out}(v)$部分, 最后清空 v 节点的残留量(算法 1 的第 4– 8 行).

算法 1. Forward-Push.

输入: 图 $G=(V,E)$, 源节点 s, 结束概率α, residue 阈值 $r_{\max}$.

输出: 集合 V 中所有节点 v 的 $\hat{\pi}_s(v)$ 和 $r_s(v)$.

1: $r_s(v)\leftarrow 1$; $r_s(v)\leftarrow 0$ **for** all $v\neq s$;
2: $\hat{\pi}_s(v)\leftarrow 0$ **for** all v;
3: **while** $\exists v\in V$ such that $r_s(v)/|N_{out}(v)|>r_{\max}$ **do**
4: **for each** $u\in N_{out}(v)$ **do**
5: $r_s(u)\leftarrow r_s(u)+(1-\alpha)\cdot r_s(v)/N_{out}(v)$
6: **end for**
7: $\hat{\pi}_s(v)\leftarrow \hat{\pi}_s(v)+\alpha\cdot r_s(v)$;
8: $r_s(v)\leftarrow 0$;
9: **end while**

如果能耗尽所有节点的残留量, 那么将能得到所有节点的准确个性化 PageRank 值, 但这将会导致巨大的计算成本, 所以在 Forward-Push 算法中, 通过一个阈值 $r_{\max}$, 仅当节点的残留值不小于 $r_{\max}$ 和节点出度的乘积时才会继续传递. 通过这样的策略, 可以极大地降低算法的计算成本, 在每轮迭代中, Forward-Push 都会保存如下不变量.

$$\pi_s(t)=\widehat{\pi}_s(t)+\sum_{v\in V} r_s(v)\cdot\pi_v(t) \tag{3}$$

等式中的$\pi_v(t)$通过另一个随机游走阶段来得到, 通过从每个节点 v 取样$\omega_v=\lceil r_s(v)\cdot(2\varepsilon/3+2)\log_2(2/p_f)/\varepsilon^2\delta\rceil$个随机游走, 然后用其中停在 t 的随机游走比例作为$\pi_v(t)$的估计值 $\tilde{\pi}_v(t)$. 最后, 可通过公式(3)得到每个节点满足定义 1 的近似个性化 *PPR* 值.

目前, 分布式环境下求解个性化 PageRank 算法的最新研究为 Delta-Push[30]算法, Delta-Push 算法基于 Spark 框架的 MPC (massively parallel computing, 大规模并行计算)模型, 在分布式环境下, 实现高效的计算个性化 PageRank 算法的迭代计算. 但是其未考虑跨域环境下的带宽异构导致的单点计算瓶颈问题, 并且因为其进行全局的随机游走, 所以算法会消耗更多的计算时间. 本文在该算法的基础上改进全局随机游走为基于概率的随机游走, 降低算法的运行时间, 并在算法中提出图预处理步骤来优化跨域环境下算法的运行效率问题.

1.2 跨域数据预处理

当前, 跨域数据处理研究大都是在已有分布式数据处理框架上进行修改, 以适应跨域带宽延迟高、带宽异构、带宽费用高等问题.

Pu 等人[31]在 Spark[32]的基础上开发了 Iridium 框架, 主要关注如何最小化地理分布环境下的分析查询操作的响应时间. 框架主要包括全局管理器和本地管理器两部分组件: 通过全局管理器协调数据中心之间的查询问题, 并分配任务给本地管理器; 本地管理器负责管理数据中心内部的资源, 并执行全局管理器下发的工作任务. 框架充分考虑广域网的带宽异构问题, 通过贪婪启发式算法来优化数据和任务的分配方式, 缩短数据中心间的数据传输时间, 并降低总体的带宽使用量.

Zhou 等人[33]在 PowerGraph[34]框架的基础上开发了 Geo-Cut 图分区框架, Geo-Cut 主要包括两个优化阶段: 首先, 以流式启发算法划分图, 划分时充分考虑数据中心的上行带宽和下行带宽的异构; 其次, 通过多次切换分区的方法来适应数据中心之间的异构带宽环境, 并通过分辨瓶颈数据中心来平衡负载, 进一步减少数据传输时间.

Yuan 等人[35,36]在 Giraph[37,38]的基础上开发了 GeoGraph 算法, 一个用来在跨域环境下处理图查询的算法, 通过组合数据中心内部数据进行同步计算、协调数据中心之间进行异步计算两个模式来高效、可靠地进行图相关算法的计算[39].

上述的跨域图处理系统都充分考虑了跨域环境, 但是在个性化 PageRank 算法的求解上, 均使用的矩阵迭代的算法, 利用框架提供的分布式能力, 并没有针对分布式个性化 PageRank 算法进行优化, 不能进一步提高性能, 在跨域环境下会产生大量的带宽消耗, 不适用于大图环境下的个性化 PageRank 问题的求解.

2 问题定义与 GPPR 算法框架

本节将对一些基本的问题进行梳理, 对所面向的研究对象个性化 PageRank 算法进行介绍, 对基本的概念做出定义, 并在表 2 中对本文所常用到的一些符号的意义进行简要说明.

表 2 符号说明

符号	描述	符号	描述
$G=(V,E)$	图数据 G, 节点集合 V, 边集合 E	ε	相对错误界限
n, m	节点和边的数量	M	地理分布的数据中心数量
P	集群处理器的数量	U_r/D_r	数据中心 r 的上传/下载带宽
α	随机游走的终止概率	I_v^r	在数据中心 r 上是否存在节点 v
$N_{out}(v)$	节点 v 的出度	R_v	至少有 1 个节点 v 的数据中心集合
$\pi_s(v)$	v 相对于 s 的准确个性化 PageRank 值	$\hat{\pi}_s(v)$	节点 v 对节点 s 的 reserve 值
$\tilde{\pi}_s(v)$	v 相对于 s 的近似个性化 PageRank 值	$r_s(v)$	节点 v 对节点 s 的 residue 值

2.1 GPPR算法框架

GPPR 算法总体共分为 3 步.

(1) 图预处理阶段, 通过启发式算法, 将图数据与工作节点相映射, 根据带宽异构来平衡数据传输负载, 然后进行随机游走的采样.

(2) 通过基于概率的 Push 算法结合蒙特卡洛模拟, 综合第(1)步采样的随机游走, 来计算所有图上节点相对于当前源节点的个性化 PageRank 值.

(3) 针对输入的每一次查询的源节点, 重复算法的第(2)步, 直到返回所有节点及与每个源节点相关的个性化 PageRank 值.

2.2 问题定义

为简化问题, 假设所有计算节点处于无拥塞网络环境, 计算节点间的上行下行带宽的巨大差异是算法的主要网络瓶颈.

定义 1((**ε,δ**)-**近似单源个性化 PageRank**). 给定一个源节点 s, 一个阈值δ, 一个错误边界$\varepsilon\in(0,1]$和一个失败概率 P_f, 一个近似单源个性化 PageRank 查询将会返回每个节点的个性化 PageRank 值的近似值 $\tilde{\pi}_s(v)$, 并且在至少 $1-P_f$的概率下满足公式(4)和公式(5).

$$|\pi_s(v)-\tilde{\pi}_s(v)|\leqslant \varepsilon\cdot\pi_s(v),\ \forall\pi_s(v)\geqslant\delta \tag{4}$$

$$|\pi_s(v)-\tilde{\pi}_s(v)|\leqslant \varepsilon\cdot\delta,\ \forall\pi_s(v)<\delta \tag{5}$$

定义 2(**蒙特卡洛随机游走**). 给定一个含有 n 个节点的有向图, 其中, 节点表示状态, 有向边表示状态之间的转移. 假设从一个节点到通过有向边相连的所有节点的转移概率相等, 每次可以一个节点转移到任意一相连节点(随机游走). 从一个给定的源节点 S 执行ω次随机游走, 并记录随机游走停在节点 t 的概率.

如图 1 所示, 模型中的某一个当前停在节点 0 上的随机游走, 可以进一步走到节点 1、节点 2、节点 5; 停在节点 5 的随机游走可以进一步走到节点 2、节点 7、节点 6.

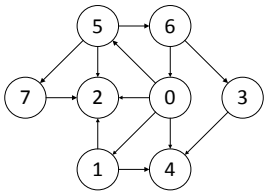


图 1 随机游走示例

定义 3(**α-随机游走**). 给定一个参数$\alpha\in[0,1)$, 默认为 0.2[4]. 假设当前节点为 v(初始化时为 s), 在每一步随机游走中, 有α的可能性停在 v, $1-\alpha$可能性移动到下一节点. 其中, 下一节点的选择有两种可能性: 如果当前节点的 $N_{out}\neq\varnothing$, 则在节点 v 进行蒙特卡洛随机游走, 即等可能选择一个出度邻居; 否则, 跳转至节点 s.

定义 4(**带宽异构**). 在一个通信系统或网络中, 存在多个不同的带宽参数(如传输速率、数据传输能力等), 这些带宽参数在时间和空间上可以变化, 且它们的分布不一定遵循特定的概率分布. 带宽异构可以在多个层次上存在, 包括网络连接、链路、设备或应用程序层面.

本文问题定义为: 给定分布在跨域环境下 k 个工作节点的图 G、一个图内点 s, 设每个图内点相对于 s 的近似单源个性化 PageRank 值为 $\tilde{\pi}_s(v)$, 工作节点所需要的带宽消耗为 B_i, 通过随机游走模型, 在保证 $\tilde{\pi}_s(v)$ 准确率的情况下, 最小化算法整体带宽消耗 $\sum_{j}^{k}B_j$.

3 图预处理

如第 1 节所述, 已有的分布式个性化 PageRank 算法主要有两类, 分别是矩阵迭代和蒙特卡洛模拟. 对于矩阵迭代相关算法来说, 对单机的计算能力和存储能力有极高的要求, 无法适用在分布式环境下. 基于蒙特卡洛模拟的方法是现有分布式个性化 PageRank 算法的主流选择, 主要由 Forward-Push 算法和蒙特卡洛模拟两部分组成, 其中, Forward-Push 算法细节如第 1.1.2 节所示.

但是在跨域环境下, 通过 Forward-Push 算法求解个性化 PageRank 近似值会有这样几方面的局限: 首先是由于带宽的异构, 不合适的工作节点与图数据方式会使得 Forward-Push 算法产生低效的计算节点, 拖慢整个算法的运行速度; 其次, 由于随机游走是动态的, 且每次都从同一源节点开始, 会造成图中所有边的数据传输负载不平衡, 导致很难构建一个满足跨域环境的分布式算法.

本研究的预处理算法由两部分组成: 首先是通过启发式算法解决图数据的分布问题; 其次, 通过全局预取样算法来保证算法每条边上的负载大致相同.

在跨域环境下, 广域网带宽比 CPU、内存等本地计算资源更稀缺, 所以跨域环境下算法的瓶颈会出现在数据中心之间的数据传输上. 事实上, 许多数据中心的拥有者希望将在全球范围内提供服务, 所以会构建自己的基础网络设施, 因此假设数据中心之间的网络连接是拥塞避免网络, 网络的瓶颈只会出现在数据中心的上行/下行链路上.

由于不同工作节点之间带宽是异构的, 所以 M 个地理分布的工作节点之间是互异的. 将 M 个图分区映射到 M 个地理分布的数据中心是一个经典的组合 NP-hard 问题, 并且解空间为 O(M!)[33]. 这个问题之所以是 NP-hard 的原因, 是因为需要探索出 M 个图分区映射到 M 个数据中心的所有可能方式, 而随着分区数量 M 的增加, 可能的映射数量呈阶乘增长(O(M!)), 解空间的指数增长使得在合理的时间内找到大规模问题的最优解成为不可行的计算任务. 当 M 的值很小时, 可以使用广度优先遍历或深度优先遍历来找到一种更低的数据中心间数据传输时间的映射方案. 但是当 M 的值很大时, 广度优先或深度优先遍历的方法需要更长的计算时间, 无法适用, 通常做法采用启发式算法、近似算法等方法在合理的时间内找到近似最优解. 因此, 这里提出一个高效的算法来快速找出最佳的映射方案.

由于算法运行在 MPC 模型上, 所以会在每一轮迭代后进行一次数据同步, 每个工作节点会有两部分数据同步, 分别是上行数据和下行数据, 其中, 上行数据受限于工作节点的上传带宽, 下行数据受限于工作节点的下载带宽. 由于个性化 PageRank 算法的特点, 每个节点的上行数据与此工作节点内部所存储的子图数据的总出度成正比; 同理, 每个节点的下行数据与此工作节点内部的子图数据的总入度成正比. 所以, 将每个节点上行和下行数据的时间用如下公式(6)、公式(7)表示, 则每一次数据同步的耗时如公式(8)所示.

$$T(i)=T_{up}(i)+T_{down}(i)=\max T_{up}^{r}(i)+\max T_{down}^{r}(i) \tag{6}$$

$$T_{up}^{r}(i)=\sum_{u\in R_u}\left(d_{in}(u)-\sum_{r}I_v^r\right)\Big/U_r \tag{7}$$

$$T_{down}^{r}(i)=\sum_{u\in R_v}\left(d_{out}(v)-\sum_{r}I_u^r\right)\Big/D_r \tag{8}$$

二次选择迭代[40]技术已经被证明可以高效地在任务调度问题中找到近似最优调度方案, 并且耗时较低. 最近的研究发现: 在有长尾分布的情况出现时, 如果有更多的选择, 该技术的性能可以得到一定的提升[41]. 如何把部分图数据和工作节点相映射, 也可以模拟成一个类似的任务调度问题, 所以提出如下算法来快速得到映射方案.

算法 2. Partition Mapping Algorithm.

输入: P_{init}: 初始分区映射方案.

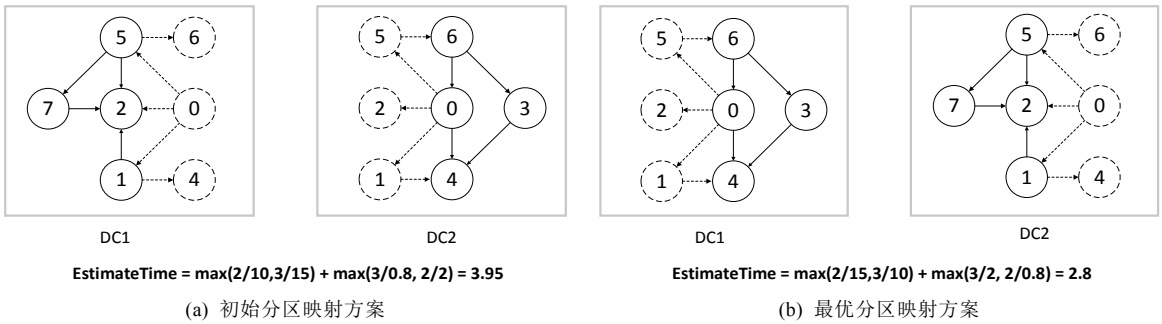
输出: P_{opt}: 最优分区映射方案.

1: P_{opt}=P_{init};
2: **repeat**
3: continue=false;
4: 随机取样 d 对分区映射;
5: Let G_b=0 and p_b=0;
6: **for** i=1 to d **do**
7: 通过交换 $pair_i$ 的映射来产生新的映射计划 p';

8: G=*EstimateTime*(P)–*ExtimateTime*(p');
9: **if** G>0
10: 交换 $pair_i$ 的映射;
11: Continue=true;
12: **end if**
13: **end for**
14: **until** !continue
15: **return** P_{opt}

算法从一个初始化默认分区映射方案开始, 随机选择 d 对工作节点及其内部数据(算法 2 第 4 行), 遍历每一对分区, 通过根据上面提到的公式来估测交换前后的数据传输时间是否有提升, 来决定是否交换两个工作节点内部的数据. 重复以上算法, 直到没有更多的提升. 其中, d 的取值默认取 2, 当工作节点之间的网络性能差异过大时, 可以适当提高.

为了更好地解释映射算法, 这里给出一个示例. 如图 1 所示的图数据, 与现有分布式个性化 PageRank 算法一样, 首先采用点切分和最长处理时间调度算法来初始划分图数据[42], 最长处理时间调度算法主要用来平衡各机器的负载. 针对图 1 的图结构数据, 初始化分结果如图 2(a)所示, 其中, 实线点为工作节点所存数据节点, 虚线点为需要同步数据的节点以及其相应地数据传输方向. 两个工作节点的上行带宽和下行带宽如表 3 所示, 假设图中所有边上的数据传输量相同且为 1, 可以通过公式(6)计算得到每一轮迭代后所需要的数据同步时间为 3.95 s. 使用算法 2 的分区映射方案交换两个工作节点的分区数据后, 可以得到如图 2(b)所示的优化结果, 根据公式可以计算得到对应的数据同步时间为 2.8 s, 所以交换后的数据与工作节点的映射方案可以更好地适应当前的异构带宽环境, 降低每一轮数据同步所需要的时间, 进而提高算法整体的迭代速度.



(a) 初始分区映射方案 (b) 最优分区映射方案

图 2 分区映射方案选择

表 3 计算节点带宽(MB/s)

计算节点	上行带宽	下行带宽
DC1	2.0	15
DC2	0.8	10

通过引入预取样算法[30], 可以保证图上的每一条边的所需数据传输量的负载均衡. 算法 3 展示了整个预取样阶段的伪代码. 首先, 通过上述公式来进行图数据的划分; 然后, 从每个节点开始 ω_p=⌊m/n⌋个随机游走, 在每轮迭代中, 通过 MPC 模型进行随机游走的模拟. 具体来说, 使用 P=⟨s,v⟩来表示一次从节点 s 开始的现在停在 v 的随机游走, 然后, 每一次随机游走都有α的可能性在当前节点结束, 并记录在集合 w_t 中; 另外, 1–α的可能性继续传播到当前节点的出邻居并更新相应的键值对, 并被记录在集合 W_a 中. 算法会重复以上循环, 直到所有的随机游走都结束.

算法 3. Pre-Sample.

输入: 图 G=(V,E), 结束概率α, P_{opt}.

输出: 预取样的随机游走集合 w_t.

1: $\omega_p \leftarrow \lfloor m/n \rfloor$; $w_t=\varnothing$;
2: **for** 1,…,R **do**
3: **for each** $v \in V$ in parallel **do**
4: **for** 1,…,$\lceil \omega_p/P \rceil$ **do**
5: $w_a^0(v) \leftarrow w_a^0(v) \cup \langle v,v \rangle$;
6: **end for**
7: **end for**
8: $W_a \leftarrow \bigcup_{v \in V} W_a^0(v)$;
9: **while** $W_a \neq \varnothing$ **do**
10: $W_t^* \leftarrow W_a filter\{b \mapsto (Random(0,1)) < \alpha\}$;
11: $W_t \leftarrow W_t \cup W_t^*$;
12: $W_a \leftarrow W_a \setminus W_t^*$;
13: $W_a \leftarrow W_a map\text{-}value\{v \mapsto u \xleftarrow{\$} N_{out}(v)\}$
14: **end while**
15: **end for**
16: **return** w_t

4 基于概率的 Push 算法

对于真实世界的图数据, 常常呈现出一个特别的分布特征, 即幂律分布(或称长尾分布). 在这种分布特征下, 少数节点具有非常高的度数(即相邻邻居数量), 而大多数节点的度数相对比较小. 这意味着这样一些“中心节点”在图数据中具有很高的连通性, 而其他节点有相对较低的连通性. 如算法 1 所述, 在 Forward-Push 算法的每一个 push 操作都会将到达当前节点的随机游走的部分推送给他的出度邻居. 对于这样度数较高的“中心节点”来说, 一方面, 他传递给出度邻居的残留值 $r_s(v)$会因为 d 的值过大而变得很小, 此时对下游节点的影响很小; 另一方面, 由于算法的局限, 即使传递的值很小也需要一次完整的数据传输, 所以中心节点的每次传输需要消耗大量的带宽. 基于以上两方面的观察, 为了减少这样的随机游走数量, 可以把随机游走的部分推送给出度邻居的一个小的随机子集. 根据证明, 这样的随机化 push 可以得到结果是准确值的一个有界无偏估计, 如公式(9)所示.

$$push=\begin{cases} d_{out}(v) \leqslant \left(\sqrt{d_{out}(u)} \cdot (1-\alpha) \cdot r_s(v)/(\alpha \cdot \varepsilon)\right), & \text{确定性推送} \\ d_{out}(v) > \left(\sqrt{d_{out}(u)} \cdot (1-\alpha) \cdot r_s(v)/(\alpha \cdot \varepsilon)\right) \text{ and } d_{out}(v) \leqslant \left(\sqrt{d_{out}(u)} \cdot (1-\alpha) \cdot r_s(v)/(r \cdot \alpha \cdot \varepsilon)\right), & \text{随机推送} \\ d_{out}(v) > \left(\sqrt{d_{out}(u)} \cdot (1-\alpha) \cdot r_s(v)/(r \cdot \alpha \cdot \varepsilon)\right), & \text{不推送} \end{cases} \tag{9}$$

这里给出一个示例, 根据上述条件, 可以将不同出度数的节点分为 3 类, 分别是确定性推送、随机推送、停止推送, 如图 3 所示. 针对确定性推送, 如图 4 节点 x, 其向下一节点推送的残留量为$(1-\alpha)\cdot r_s(v)/d_{out}(v)$; 对于随机推送节点来说, 如图 3 中的节点 u 和 z, 其向下一节点推送的残留量为 $\alpha \cdot \varepsilon \sqrt{d_{out}(u)}$, 但是否推送取决于公式(9)中的随机值 r; 对于停止推送的节点来说, 如图 3 中的节点 y, 其不会向后续节点进行推送, 因为这样的节点通常出度很大, 并且由于残留值是均分给后续出度节点的, 所以每一个后续出度节点都会得到很小的残留值, 较小的残留值对算法的终值影响很小. 传统的方法会向每一个出度节点传送残留值, 在残留值很小的时候也会占用同等的带宽, 导致算法的整体带宽使用量过高.

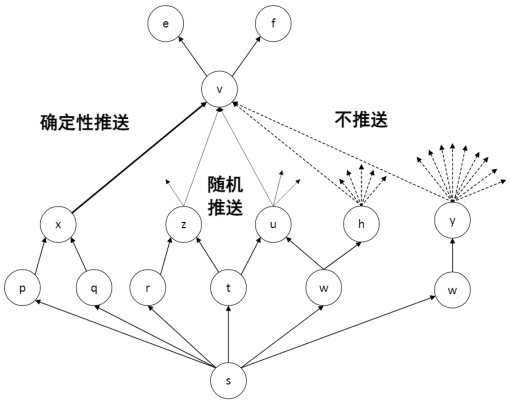


图 3 Random 推送示例

首先给出 GPPR 算法的整体框架, 如算法 4 所示.

算法 4. GPPR.

输入: 图 $G=(V,E)$, 查询集合 Q, 结束概率α, 相对精度ε, 最优分区映射方案 P_{opt}.

输出: 近似单源个性化 PageRank 值 $\tilde{\pi}$.

1: $\delta \leftarrow 1/n$; $p_f \leftarrow 1/n$;

2: $\omega_p \leftarrow p\cdot\lceil \lfloor m/n \rfloor / p \rceil$;

3: $\omega_{\varepsilon,\delta} \leftarrow \lceil (2\varepsilon/3+2)\log(2/p_f)/\varepsilon^2\delta \rceil$;

4: **for each** $s \in Q$ **do**

5: $r_s \leftarrow [\langle s,1\rangle]$; $\hat{\pi}_s \leftarrow \varnothing$;

6: $r_{sum} \leftarrow 1$; $r_{\max} \leftarrow 1$;

7: **while** $r_{\max} > \omega_p/\omega_{\varepsilon,\delta}$ **do**

8: $r_{sum} \leftarrow (1-\alpha) r_{sum}$

9: $(r_{\max}, r_s, \hat{\pi}_s) \leftarrow$ *Random-Push*$(G,\alpha,r_s,\hat{\pi}_s)$;

10: **end while**

11: **for each** $\langle v, r_s(v)\rangle \in r_s$ in parallel **do**

12: $\omega_v \leftarrow \lceil r_s(v)/r_{sum}\cdot\omega_p \rceil$

13: $\tilde{\pi}_s^r(v) \leftarrow [\langle t_i \xleftarrow{\$} w_t(v), r_s(v)/\omega_v \rangle \mid i = 1,...,\omega_v]$

14: **end for**

15: $\tilde{\pi}_s^* \leftarrow \bigcup_{v\in V} \tilde{\pi}_s^r(v)$

16: $\tilde{\pi}_s \leftarrow \hat{\pi}_s \cup \tilde{\pi}_s^*$*reduce-by-key*$\{+\}$

17: **return** $\tilde{\pi}$

18: **end for**

算法读取预处理阶段返回的最优映射方案以及一个查询集合 Q, Q 中包含着所要查询的所有源节点. 首先初始化阈值, 其中, $\omega_{\varepsilon,\delta}$是蒙特卡洛方法满足定义 1 近似个性化 PageRank 所需的随机游走数量(算法 4 第 3 行); 然后, 针对查询集合 Q 中的每一个查询源节点 s, 算法首先向 r_s 中增加键值对$\langle s,1\rangle$, 表示每次从 s 节点开始随机游走(算法 4 第 5 行), 并通过 $r_{\max} > \omega_p/\omega_{\varepsilon,\delta}$条件来保证算法最终结果的准确性, 其中, ω_p 是预取样阶段每个节点取样的随机游走数量. 在迭代的过程中, 引入基于概率的 Push 算法(算法 5), 算法 5 主要通过公式(9)来进行全局基于概率的随机游走(算法 5 解释见下), 并在迭代结束后, 通过公式(10)聚合算法 3 和算法 5 的中间结果, 并得到最终的个性化 PageRank 值:

$$\tilde{\pi}_s(t)=\hat{\pi}_s(t)+\sum_{v\in V} r_s(v)\cdot\pi'_v(t) \tag{10}$$

算法 5. Random-Push.

输入: 图 $G=(V,E)$, 查询集合 Q, 结束概率α, 相对精度保证ε;

输出: 最大残留量 $r_{\max}$, 残留量 r_s, 储备量 $\hat{\pi}_s$.

1: $\pi^*_s \leftarrow ramp\text{-}value\{r_s(v)\mapsto \alpha\cdot r_s(v)\}$;

2: **for each** $\langle v,r_s(v)\rangle\in r_s$ in parallel **do**

3: **if** 满足确定性推送条件

4: $r^*_s(v)\leftarrow[\langle u,(1-\alpha)\cdot r_s(v)/d_{out}(v)\rangle \mid u\in N_{out}(v)]$

5: **else if** 满足随机性推送条件

6: $r^*_s(v)\leftarrow\left[\left\langle u,\alpha\cdot\varepsilon\Big/\sqrt{d_{out}(v)}\right\rangle \mid u\in N_{out}(v)\right]$

7: **else**

8: $r^*_s(v)\leftarrow\varnothing$

9: **end for**

10: $r_s\leftarrow\left(\bigcup_{u\in v} r^*_s(v)\right)$

11: $\hat{\pi}_s\leftarrow\hat{\pi}_s\cup\pi^*_s$

12: $r_{\max}\leftarrow r_s reduce\text{-}value\{\max\}$

13: **return** $r_{\max}$, r_s, $\hat{\pi}_s$

算法 5 实现了公式(9)所定义的基于概率的随机 Push 算法, 算法首先读取当前图数据, 针对每一个节点 v, 算法根据公式(9)所定义的 3 类推送条件, 依据其出度的大小来确定该节点的 push 操作(算法 5 的第 3–8 行). 在这样的条件下, 针对出度大的节点, 可以大幅减少其进行随机游走所需要传送的数据量, 进而降低整个算法的通信量, 提高算法的迭代速度. 最后更新并记录最大的 $r_s(v)$, 返回该值以供上层算法 4 来确定算法中循环的停止条件.

基于概率的 Push 算法, 其结果的无偏性证明如下.

证明: 在第 i 次进入循环时, 边(u,v)的一次 push 操作, 用 $X_{i+1}(u,v)$来表示 $\hat{r}_{i+1}(s,v)$ 在此次 push 的增加量, 根据算法, 当 $(1-\alpha)\hat{r}_i(s,u)/|N_{out}(u)|\geqslant\alpha\varepsilon\Big/\sqrt{N_{out}(u)}$ 时, $X_{i+1}(u,v)=(1-\alpha)\hat{r}_i(s,u)/|N_{out}(u)|$; 否则, $X_{i+1}(u,v)$ 有 $\sqrt{N_{out}(u)}(1-\alpha)\hat{r}(s,u)\Big/\alpha\varepsilon|N_{out}(u)|$ 的概率为 $\alpha\varepsilon\Big/\sqrt{N_{out}(u)}$, 其他情况为 0. 这里使用 $\{\hat{r}_i(s)\}$ 表示 $\sum_{v\in V}\hat{r}_i(s,u)$, $X_{i+1}(u,v)$关于 $\{\hat{r}_i(s)\}$ 的条件期望可以写成 $E[X_{i+1}(u,v)\mid\{\hat{r}_i(s)\}]=(1-\alpha)\hat{r}(s,u)/|N_{out}(u)|$.

因为 $\hat{r}_{i+1}(s,u)=\sum_{v\in N_{out}(u)}X_{i+1}(u,v)$, 所以 $\hat{r}_{i+1}(s,u)$ 关于 $\{\hat{r}_i(s)\}$ 的条件期望是:

$$E[\hat{r}_{i+1}(s,u)\mid\{\hat{r}_i(s)\}]=\sum_{v\in N_{out}(u)}E[X_{i+1}(u,v)\mid\{\hat{r}_i(s)\}].$$

基于前一个等式可以得到 $E[\hat{r}_{i+1}(s,u)\mid\hat{r}_i(s)]=\sum_{u\in N_{in}}((1-\alpha)\hat{r}_i(s,u)/|N_{out}(u)|)$.

又因为 $E[\hat{r}_{i+1}(s,u)]=E[E[\hat{r}_{i+1}(s,u)\mid\{\hat{r}_i(s)\}]]$, 所以可以得到 $E[\hat{r}_{i+1}(s,u)]=\sum_{u\in N_{in}}\{(1-\alpha)E[\hat{r}_i(s,u)]/|N_{out(u)}|\}$.

由归纳法可知 $E[\hat{r}_i(s,u)]=r_i(s,u)$, 由此可推得 $E[\hat{r}_{i+1}(s,u)]=\sum_{u\in N_{in}}((1-\alpha)E[\hat{r}_i(s,u)]/|N_{out}(u)|)=r_{i+1}(s,u)$, 即无偏性成立.

个性化 PageRank 计算的误差证明如下.

证明: 考虑从节点 v 产生ω_v个随机游走,定义一个伯努利变量 $X^i_v(t)$ (当第 i 次随机游走停止在 t 节点上则为 1, 否则为 0), 可得其期望为$\pi(v,t)$. 如果一个随机游走在 t 点结束, 则该节点的 PPR 估计值 $\tilde{\pi}_s(v)$ 会有一个

增量为 $r_s(v)/\omega_v$. 令增加值为μ_v, 则可以得到等式 $E\left[\sum_{i=1}^{\omega_v}(\mu_v \cdot X_v^i(t))\right]=r_s(v)\cdot\pi_v(t)$, 其中, $\sum_{i=1}^{\omega_v}(\mu_v \cdot X_v^i(t))$ 是 $\tilde{\pi}_s(v)$ 从节点 v 获取的增量. 将这个增量表示为ψ_v, 可得等式 $E\left[\sum_{v\in V}(\psi_v)\right]=\sum_{v\in V} r_s(v)\cdot\pi_v(t)$. 此外, 引入 FORA[26]所提出的定理, 即: 在每个$v\in V$节点产生$\omega_v=r_s(v)\cdot\omega_{\varepsilon,\delta}$个随机游走的环境下, $\tilde{\pi}_s(v)$ 和 $\sum_{v\in V}(\psi_v)$ 组合所得到的个性化 PageRank 估计值 $\tilde{\pi}_s(v)$ 满足定义 1. 由于本文算法已保证 $r_{\max}\leqslant\omega_p/\omega_{\varepsilon,\delta}$成立, 可进一步得到 $r_{\max}\cdot\omega_{\varepsilon,\delta}\leqslant\omega_p$, 即, 任意一个节点 v 都已经预先存储了足够数量的随机游走用于保证 FORA 定理的成立. 综上, 本算法满足定义 1.

5 实 验

本节将通过多组实验验证评估本算法在跨域环境下的有效性. 首先对实验数据集、对比算法和参数设置进行描述, 在此基础上进行多组对比实验, 并对实验结果进行详细的分析.

5.1 实验设置

- 实验环境

算法使用 Scala 语言, 基于 Spark 框架实现. 使用 20 台分布在各大洲(共选取 10 个地理分布的数据中心, 分别为中国张家口、中国香港、中国成都、英国伦敦、德国法兰克福、美国弗吉尼亚、美国硅谷、日本东京、澳大利亚悉尼、阿联酋迪拜, 默认选择其中相距较远的 6 的数据中心进行实验)的阿里云服务器进行实验, 每台云服务器拥有 16 核 64 GB 内存. 为模拟跨域环境, 限制每台服务器的上行和下行带宽, 最大相差 8 倍. 数据集使用 GrQc、DBLP、Stanford、Pokec、LJ、Orkut、Twitter、Friendster 共 8 个图数据集, 其中, 顶点数和边数见表 4, 并以 k=10^3, M=10^6, B=10^9 的方式呈现, 详细数据可以在 KONECT[43]项目找到并下载.

表 4 实验数据集属性

名称	顶点数	边数	平均度数	图类型
GrQc	5.2k	29.0k	2.77	无向
DBLP	613.6k	2.0M	6.6	无向
Stanford	281.9k	2.3M	16.4	有向
Pokec	1.6M	30.6M	37.5	有向
LJ	4.8M	69.0M	28.2	有向
Orkut	3.1M	117.2M	76.3	无向
Twitter	41.7M	1.5B	70.5	有向
Friendster	65.6M	1.8B	75.7	无向

- 对比算法

Spark 框架提供的 Power 算法、Delta-Push 算法和 DistPPR 算法. 其中, Power 算法使用迭代的方式来实现, 利用框架所提供的分布式能力, 并限制算法在保证绝对误差不超过 10^{-20} 的情况下停止迭代; Delta-Push 算法使用全局 Push 算法, 并且实现了 Map-Reduce 模型, 适用于分布式环境; DistPPR 算法使用 pipeline 机制, 通过多路并行机制来实现算法的迭代加速, 可以直接得出图上所有节点之间的个性化 PageRank 值, 但是算法运行时间长.

- 参数设置

实验中的相关参数设置遵循 Delta-Push[30]算法, 设置δ=1/n, p_f=1/n 来保证算法求出的个性化 PageRank 值维持在一定的误差范围内, 并满足定义 1 所述. 参照默认实现[4], 设置α=0.2, ε=0.5.

5.2 实验结果

5.2.1 运行时间

第 1 个实验检查了在相同实验条件下, 4 个算法在不同数据集中的运行时间; 并且为了便于展示, 纵坐标以对数形式呈现. 如图 4 所示, 在分布式环境下, 本研究提出的 GPPR 算法相对于分布式下的 Power 算法可以有 3–5 倍的提升, 与 Delta-Push 算法相比有少量提升. 这主要是因为相对于 Delta-Push 算法来说, GPPR 算法

虽然加快了迭代的效率, 但是增加了数据的预处理步骤, 增加了算法的耗时. DistPPR 算法非常耗时, 并且在后 4 个数据集上的运行时间超过了 24 h, 在这里不再记录.

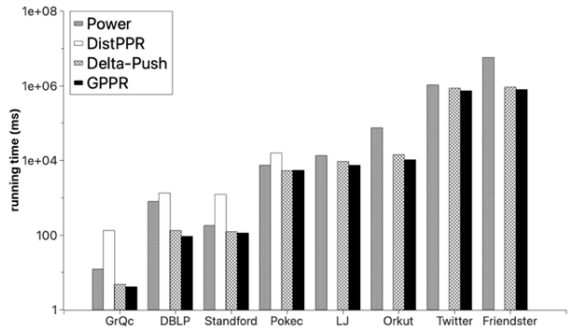


图 4 平均运行时间

5.2.2 总通信成本

第 2 个实验检查了在相同实验条件下, 4 个算法在不同数据集中的总通信成本, 如图 5 所示. 由于 GrQc 的通信成本过低, 暂不考虑. 可以得出: Power 算法会产生最大的总通信成本, 这是因为其采用的矩阵迭代方法会增加每一对工作节点之间的通信量; Delta-Push 虽然使用了预采样算法, 但由于跨域环境带宽的异构问题, 其通信成本也比较高; DistPPR 的通信成本与其他算法相比相对不高, 但是其运行时间随图数据量级增长得过快, 不适用于大图环境. 由于其在后 4 个数据集上运行时间超过 24 h, 所以这里不再记录其通信成本.本研究提出的 GPPR 算法可以显著地降低跨域环境下计算个性化 PageRank 算法的总通信成本.

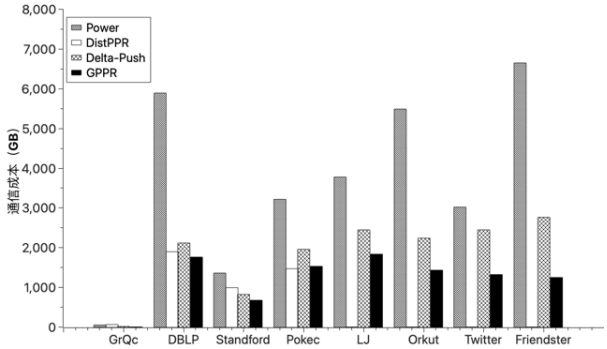


图 5 总通信成本

5.2.3 算法运行平均轮次

第 3 个实验说明了在相同实验条件下, 4 个算法在不同数据集中的平均迭代轮次, 结果如图 6 所示.

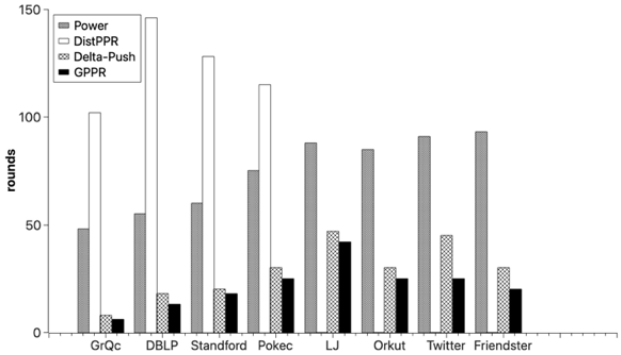


图 6 平均轮次

可以看出: 4 个算法中, DistPPR 算法需要最多的迭代次数, 所以在数据量大的后 4 个数据集上运行时间会很长; Power 算法因为使用了矩阵迭代的思想, 为了达到近似条件, 需要进行更多轮迭代; Delta-Push 算法和 GPPR 算法迭代次数相差不大, 由于 GPPR 在 Delta-Push 算法的基础上加了随机 Push 算法, 所以可以一定程度上降低算法的迭代次数, 降低算法的运行时间.

5.2.4 参数变化对算法的影响

第 4 个实验说明了不同参数α的情况下, 本研究提出的 GPPR 算法在 Pokec 和 LJ 上的运行时间变化. 如图 7 的两条曲线所示, 可以看出: 参数α越大, 算法所需要的耗时越少. 因为当α变大时, 增加了每一次随机游走停止的概率, 减少了算法每次迭代所需要的时间, 从而导致计算的总体耗时减少.

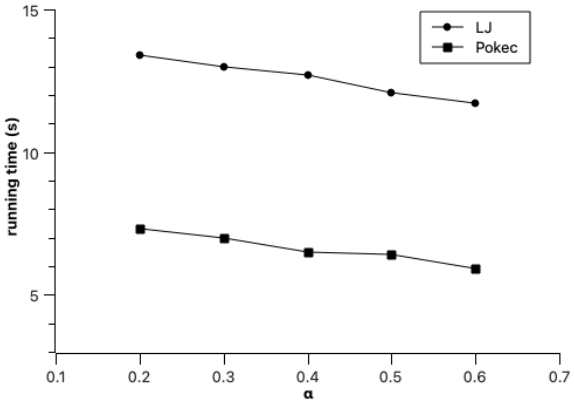
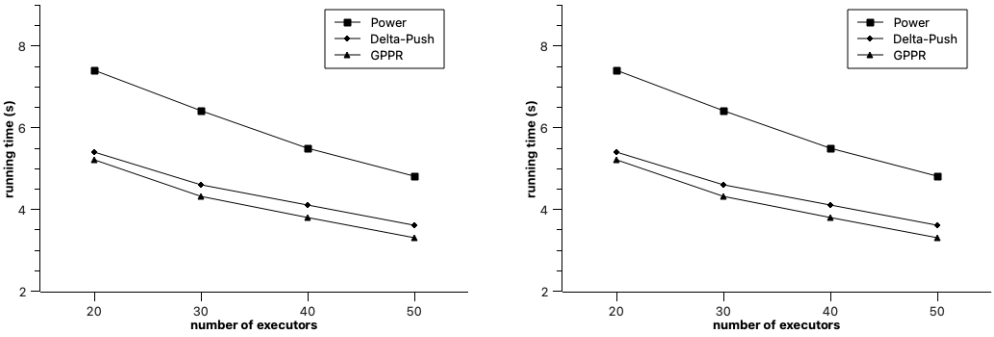


图 7 α变化

5.2.5 算法的可扩展性

第 5 个实验说明了在集群数量不变、计算节点数量不同的情况下, 本研究提出的 GPPR 算法以及 Power 算法、Delta-Push 算法在 Pokec 和 LJ 两个数据集上的运行时间, 如图 8(a)和(b)所示, 分别选取集群工作节点总数量为 20, 30, 40 和 50. 从图中数据可以得知: 随着计算节点数量的增加, 4 个算法的耗时不断减少; 但是当计算节点数量增加到一定程度之后, 对算法运行时间的降低量不再显著. 这是因为更多的计算节点数量会导致在算法的每一轮迭代过程中都产生更多的通信数据量, 进一步由于跨域环境带宽的特殊性, 更低的广域网带宽会进一步拖慢算法的迭代速度. 根据 3 个算法的趋势对比可以看出: 本研究提出的 GPPR 算法不会影响算法的可扩展性, 在增加计算节点的情况下, 可以降低算法的运行耗时, 提升算法的迭代效率.



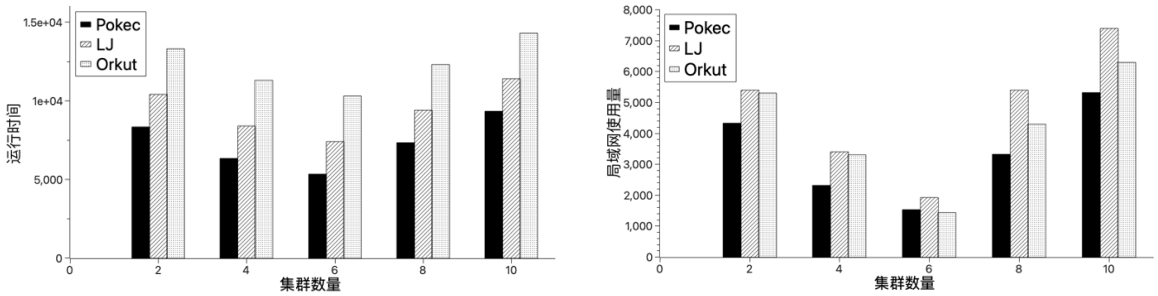
(a) Pokec (b) LJ

图 8 工作节点数量变化

5.2.6 集群数量敏感性

第 5 个实验说明了在集群数量不变的情况下, 增减单个集群内计算节点观察到的算法扩展能力相关实验; 实验 6 通过设定每个集群内计算节点数量不变的情况下, 增加集群数量所带来的算法性能变化情况(如图 9 所

示). 通过设定算法的集群数量为 2, 4, 6, 8, 10 (每组数据总计算节点数量为 20), M 相应为 2, 4, 6, 8, 10. 当 M 比较小时, 图分区映射更建议使用简单的 BFS/DFS 来找到最佳执行计划(如第 3 节所述); 当 M 的数量超过地理分布数据中心的数量时, 即一个地理位置有两个以上相互独立的数据中心时, 算法耗时会增加. 因为相对于一个地理位置一个数据中心来说, 更多的数据中心需要额外映射计算开销, 进而增加算法整体耗时. 根据数据可以得出: 集群数量为 6 时, 算法效果最好, 因为 6 正好是地理分布的数据中心个数(即中国、欧洲、美国、日本、澳大利亚、阿联酋). 因此, 当 M 为地理分布的数据中心个数时, 算法可以得到最优的运行效率.



(a) 算法耗时 (b) 局域网使用

图 9 集群数量变化

6 总 结

为了解决跨域环境下分布式个性化 PageRank 算法所存在的通信量高、单点瓶颈问题, 本文提出了 GPPR 算法. 通过基于带宽和节点度数的启发式优化算法, 综合考虑通信数据量和通行时间, 在保证算法迭代运行时间的前提下, 有效降低了算法的通信数据量和通信代价. 同时, 引入了基于概率的 Push 算法利用图的结构信息, 减少了算法的迭代次数, 加快了算法所需的执行时间. 实验结果表明: 该方法在保证结果准确性的前提下, 显著减少了通信数据量和通信时间.

References:

[1] Sahu S, Mhedhbi A, Salihoglu S, *et al*. The ubiquity of large graphs and surprising challenges of graph processing. Proc. of the VLDB Endowment, 2017, 11(4): 420−431.

[2] Angles R, Gutierrez C. Survey of graph database models. ACM Computing Surveys (CSUR), 2008, 40(1): 1−39.

[3] Adoni HWY, Nahhal T, Krichen M, *et al*. A survey of current challenges in partitioning and processing of graph-structured data in parallel and distributed systems. Distributed and Parallel Databases, 2020, 38(2): 495−530.

[4] Page L, Brin S, Motwani R, *et al*. The PageRank citation ranking: Bringing order to the Web. Technical Report, Stanford University, 1998.

[5] Chung F. A brief survey of PageRank algorithms. IEEE Trans. on Network Science & Engineering, 2014, 1(1): 38−42.

[6] Yuan Y, Wang G, Chen L, *et al*. Efficient subgraph similarity search on large probabilistic graph databases. arXiv:1205.6692, 2012.

[7] Gleich DF. PageRank beyond the Web. SIAM Review, 2015, 57(3): 321−363.

[8] Park S, Lee W, Choe B, *et al*. A survey on personalized PageRank computation algorithms. IEEE Access, 2019, 7: 163049−163062.

[9] Dolev S, Florissi P, Gudes E, *et al*. A survey on geographically distributed big-data processing using MapReduce. IEEE Trans. on Big Data, 2017, 5(1): 60−80.

[10] Aws global infrastructure. 2019. https://aws.amazon.com/about-aws/global-infrastructure/

[11] Windows azure regions. 2019. https://azure.microsoft.com/en-us/regions/

[12] Google datacenter locations. 2019. https://www.google.com/about/datacenters/inside/locations/index.html

[13] Yuan Y, Wang G, Chen L, *et al*. Efficient keyword search on uncertain graph data. IEEE Trans. on Knowledge and Data Engineering, 2013, 25(12): 2767−2779.

[14] Yuan Y, Wang G, Wang H, *et al*. Efficient subgraph search over large uncertain graphs. Proc. of the VLDB Endowment, 2011, 4(11): 876−886.

[15] Nawab F, Agrawal D, El Abbadi A. The challenges of global-scale data management. In: Proc. of the 2016 Int'l Conf. on Management of Data. 2016. 2223−2227.

[16] Dolev S, Florissi P, Gudes E, *et al*. A survey on geographically distributed big-data processing using MapReduce. IEEE Trans. on Big Data, 2017, 5(1): 60−80.

[17] Fujiwara Y, Nakatsuji M, Yamamuro T, *et al*. Efficient personalized PageRank with accuracy assurance. In: Proc. of the 18th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2012. 15−23.

[18] Bi X, Nie HJ, Zhang GL, Hu L, Ma YL, Zhao XG, Yuan Y, Wang GR. Boosting question answering over knowledge graph with reward integration and policy evaluation under weak supervision. Information Processing & Management, 2023, 60(2): Article No. 103242.

[19] Maehara T, Akiba T, Iwata Y, *et al*. Computing personalized PageRank quickly by exploiting graph structures. Proc. of the VLDB Endowment, 2014, 7(12): 1023−1034.

[20] Shin K, Jung J, Lee S, *et al*. Bear: Block elimination approach for random walk with restart on large graphs. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. 2015. 1571−1585.

[21] Zhu F, Fang Y, Chang KCC, *et al*. Incremental and accuracy-aware personalized PageRank through scheduled approximation. Proceedings of the VLDB Endowment, 2013, 6(6): 481−492.

[22] Andersen R, Chung F, Lang K. Local graph partitioning using PageRank vectors. In: Proc. of the 2006 47th Annual IEEE Symp. on Foundations of Computer Science (FOCS 2006). IEEE, 2006. 475−486.

[23] Andersen R, Borgs C, Chayes J, *et al*. Local computation of PageRank contributions. In: Proc. of the WAW, Vol. 4863. 2007. 150−165.

[24] Jeh G, Widom J. Scaling personalized Web search. In: Proc. of the 12th Int'l Conf. on World Wide Web. 2003. 271−279.

[25] Wang S, Tang Y, Xiao X, *et al*. HubPPR: Effective indexing for approximate personalized PageRank. Proc. of the VLDB Endowment, 2016, 10(3): 205−216.

[26] Wang S, Yang R, Xiao X, *et al*. FORA: Simple and effective approximate single-source personalized PageRank. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2017. 505−514.

[27] Bahmani B, Chakrabarti K, Xin D. Fast personalized PageRank on MapReduce. In: Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of Data. 2011. 973−984.

[28] Lin W. Distributed algorithms for fully personalized PageRank on large graphs. In: Proc. of the World Wide Web Conf. 2019. 1084−1094.

[29] Das Sarma A, Molla AR, Pandurangan G, *et al*. Fast distributed PageRank computation. In: Proc. of the 14th Int'l Conf. on Distributed Computing and Networking (ICDCN 2013). Mumbai: Springer, 2013. 11−26.

[30] Hou G, Chen X, Wang S, *et al*. Massively parallel algorithms for personalized PageRank. Proc. of the VLDB Endowment, 2021, 14(9): 1668−1680.

[31] Pu Q, Ananthanarayanan G, Bodik P, *et al*. Low latency geo-distributed data analytics. ACM SIGCOMM Computer Communication Review, 2015, 45(4): 421−434.

[32] Zaharia M, Chowdhury M, Franklin MJ, *et al*. Spark: Cluster computing with working sets. In: Proc. of the HotCloud. 2010.

[33] Zhou AC, Shen B, Xiao Y, *et al*. Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters. IEEE Trans. on Parallel and Distributed Systems, 2019, 31(7): 1707−1723.

[34] Gonzalez JE, Low Y, Gu H, *et al*. PowerGraph: Distributed graph-parallel computation on natural graphs. In: Proc. of the Presented as part of the 10th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2012). 2012. 17−30.

[35] Yuan Y, Ma D, Wen Z, *et al*. Efficient graph query processing over geo-distributed datacenters. In: Proc. of the 43rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. 2020. 619−628.

[36] Yuan Y, Wang G, Chen L, *et al*. Efficient keyword search on uncertain graph data. IEEE Trans. on Knowledge and Data Engineering, 2013, 25(12): 2767−2779.

[37] Martella C, Shaposhnik R, Logothetis D, *et al*. Practical Graph Analytics with Apache Giraph. Berkeley: Apress, 2015.

[38] Bi X, Nie HJ, Zhang XY, Zhao XG, Yuan Y, Wang GR. Unrestricted multi-hop reasoning network for interpretable question answering over knowledge graph. Knowledge-based Systems, 2022, 243: Article No. 108515.

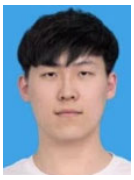
[39] Yuan Y, Chen L, Wang G. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In: Proc. of the 15th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2010). Tsukuba: Springer, 2010. 155−170.

[40] Ren X, Ananthanarayanan G, Wierman A, *et al*. Hopper: Decentralized speculation-aware cluster scheduling at scale. In: Proc. of the 2015 ACM Conf. on Special Interest Group on Data Communication. 2015. 379−392.

[41] Mitzenmacher M. The power of two choices in randomized load balancing. IEEE Trans. on Parallel and Distributed Systems, 2001, 12(10): 1094−1104.

[42] Williamson DP, Shmoys DB. The Design of Approximation Algorithms. Cambridge University Press, 2011.

[43] Kunegis J. KONECT: The Koblenz network collection. In: Proc. of the 22nd Int'l Conf. on World Wide Web. 2013. 1343−1350.



陈子俊(1999－), 男, 硕士生, CCF 学生会员, 主要研究领域为地理分布式计算.

王一舒(1993－), 女, 博士, 讲师, CCF 专业会员, 主要研究领域为图数据管理与分析, 时空数据管理与分析.

马德龙(1990－), 男, 博士生, 主要研究领域为分布式大图数据处理与分析, 跨域计算.

袁野(1981－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为大数据管理与分析, 基于大数据的人工智能, 分布式大数据计算.