软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(12):5341−5362 [doi: 10.13328/j.cnki.jos.007063] [CSTR: 32375.14.jos.007063]

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

大规模图的分布式核分解算法*

翁同峰[1,2], 周 旭[1,3], 李肯立[1,3], 胡逸骉[1,3]

[1](湖南大学 信息科学与工程学院, 湖南 长沙 410082)

[2](School of Computing, National University of Singapore, Singapore 117417, Singapore)

[3](国家超级计算长沙中心, 湖南 长沙 410082)

通信作者: 周旭, E-mail: zhxu@hnu.edu.cn; 胡逸骉, E-mail: yikunhu@hnu.edu.cn

摘 要: 随着互联网信息技术的发展, 社交网络、计算机网络及生物信息网络等领域涌现海量大规模图数据. 鉴于传统图数据管理技术在处理大规模图时存在存储及性能方面的局限, 大规模图的分布式处理技术已成为图数据库领域的研究热点, 并得到工业界和学术界的广泛关注. 图的核分解用于计算图中所有顶点的核值, 有助于挖掘重要图结构信息, 在社区搜索、蛋白质结构分析和网络结构可视化等诸多应用中发挥着关键作用. 当前以顶点为中心计算模式的分布式核分解算法中采用一种广播的消息传递机制, 一方面, 存在大量的冗余通信及计算开销; 另一方面, 处理大规模图核分解过程中易产生内存溢出问题. 为此, 分别提出基于全局激活和层次剥离计算框架, 并提出分布式核分解新算法, 通过引入基于顶点核值局部性特点的消息剪枝策略和以计算节点为中心的计算新模式, 保证算法有效性的同时提升其性能. 在国家超级计算长沙中心分布式集群上, 分别针对大规模真实和合成数据集, 算法总耗时性能提升比例为 37%–98%, 验证所提模型和算法的有效性和高效性.

关键词: 大规模图; 分布式算法; 核分解; 图计算

中图法分类号: TP301

中文引用格式: 翁同峰, 周旭, 李肯立, 胡逸骉. 大规模图的分布式核分解算法. 软件学报, 2024, 35(12): 5341–5362. http://www.jos.org.cn/1000-9825/7063.htm

英文引用格式: Weng TF, Zhou X, Li KL, Hu YK. Distributed Approaches to Core Decomposition on Large-scale Graphs. Ruan Jian Xue Bao/Journal of Software, 2024, 35(12): 5341–5362 (in Chinese). http://www.jos.org.cn/1000-9825/7063.htm

Distributed Approaches to Core Decomposition on Large-scale Graphs

WENG Tong-Feng[1,2], ZHOU Xu[1,3], LI Ken-Li[1,3], HU Yi-Kun[1,3]

[1](College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China)

[2](School of Computing, National University of Singapore, Singapore 117417, Singapore)

[3](National Supercomputing Center in Changsha, Changsha 410082, China)

Abstract: With the development of Internet information technology, large-scale graphs have widely emerged in social networks, computer networks, and biological information networks. In view of the storage and performance limitations of traditional graph data management technology when dealing with large-scale graphs, distributed management technology has become a hotspot in industry and academia fields. The core decomposition is adopted to get core numbers of vertices in a graph and plays a key role in many applications, including community search, protein structure analysis, and network structure visualization. The existing distributed core decomposition algorithm applied a broadcast message delivery mechanism based on the vertex-centric mode, which may generate a large amount of redundant communication and computation overhead and lead to memory overflow when processing large-scale graphs. To address these issues, this study proposes novel distributed core decomposition algorithms based on global activation and peeling calculation frameworks,

* 基金项目: 国家自然科学基金 (62172146, 62102143); 湖南省自然科学基金 (2023JJ10016, 2022JJ30009, 2023JJ30083); 湖南省教育厅科学研究重点项目 (22A0592)
收稿时间: 2023-05-03; 修改时间: 2023-08-03; 采用时间: 2023-09-26; jos 在线出版时间: 2024-01-31
CNKI 网络首发时间: 2024-02-02

respectively. In addition, there are several strategies designed to improve algorithm performance. Based on the locality of the vertex core number, the study proposes a new message-pruning strategy and a new worker-centric computing mode, thereby improving the efficiency of our algorithms. To verify those strategies, this study deploys the proposed models and algorithms on the distributed cluster of the National Supercomputing Center in Changsha, and the effectiveness and efficiency of the proposed methods are evaluated through a large number of experiments on real and synthetic data sets. The total time performance of the algorithm is improved by 37% to 98%.

Key words: large-scale graph; distributed algorithm; core decomposition; graph computing

1 引 言

在社交网络、生物信息网络等实际应用中, 图可用于对实体和实体间关系进行建模, 而图核值分解 (简称核分解) 是图数据挖掘与分析的基本操作. 给定图 G, k-核为一种重要的紧密子图结构, 其包含的顶点度数均不小于 k. 顶点 v 的核值 $C(v)$ 表示存在 $C(v)$-核包含顶点 v, 而不存在 $(C(v)+1)$-核包含顶点 v[1]. 核分解用于计算图中所有顶点的核值, 有助于快速获取具备不同紧密性的子图, 已被广泛应用于社区搜索[2,3]、团查找[4]、大规模图网络结构可视化[5]以及用于蛋白质结构分析[6]等实际应用中[7].

随着互联网信息技术的发展, 社交网络等应用中的图数据规模呈指数级增长. 国内外著名的社交平台 Facebook、微信等活跃用户的数量以及互联网网站规模均已超 10 亿, 用户及网站之间的连接数量达到百亿甚至千亿级别, 而传统的单机因内存瓶颈, 已难以满足如此大规模图数据的存储需求. 因此, 亟需通过分布式平台, 实现大规模图的有效存储和高效处理.

图的核分解问题是图数据管理领域的热点研究课题, 相关研究主要包含以下 3 类.

(1) 单机串行核分解算法. 当前最好的串行核分解算法由 Batagelj 等人[8]提出. 该算法迭代删除图中度数最小的顶点, 而顶点被删除时的度数即为其核值. Khaouid 等人[9]评估了多种单机核分解算法性能. Li 等人[10]设计启发式算法查找最小规模社区, 并在精度和速度进行有效权衡. Li 等人[11]针对单机难以满足大规模图数据存储的硬件瓶颈问题, 设计了基于硬盘、空间复杂度为 O(n) 的图最大核值查找算法.

(2) 单机并行核分解算法. Kabir 等人[12]设计了基于共享内存的多核并行核分解算法, 通过原子操作并行化处理核分解过程中部分计算任务, 以提升计算性能. Mehrafsa 等人[13]设计了一种核分解问题的向量化方法, 并通过 GPU 对算法进行加速, 进而显著提升了大规模图核分解效率.

(3) 分布式核分解算法. Montresor 等人[14]首次提出一种分布式核分解算法, 其中所有计算节点维系一个全局顶点核向量, 各顶点根据核向量中包含的邻接点核值对自身核值进行更新. 通过多轮迭代, 收敛后可获取所有顶点最终核值. Mandal 等人[15]在 Spark 上基于以顶点为中心计算模式实现分布式核分解, 相比基于硬盘的 MapReduce 计算方式节省了 I/O 开销. 为减少大规模图数据分解过程中的通信开销, Weng 等人[16]在开源分布式系统上实现了图核分解且可对核值进行增量式维护.

上述方案对核分解问题的研究具有重要价值, 但在处理大规模图数据的核分解时仍然存在巨大挑战. 首先, 第 (1) 类和第 (2) 类方案中基于单机内存的存储方式已无法满足大规模图的存储需求, 若采用硬盘作为主存, 将引入大量的 I/O 开销; 其次, 第 (3) 类方案通过分布式存储解决了大规模图数据的存储问题, 但因计算过程中仍存在大量的冗余计算和通信开销, 常导致算法效率不高, 甚至在处理大规模图时会产生内存溢出问题, 难以获取核分解的准确结果.

如图 1 所示, 核分解主要包含一个迭代过程. 起始阶段, 所有顶点核值初始化为各自的度数. 因顶点 v_{11} 的度数最小, 其将被首先删除, 其核值为 $C(v_{11})=1$. 随后, 顶点 v_{11} 邻接点 v_0 的度数及核值将被更新. 当所有顶点的核值收敛时可终止该迭代过程. 现有分布式核分解算法[14,15]采取广播的消息传递机制, 在计算过程中某顶点核值发生改变时, 将其最新核值发送给所有邻接点. 譬如, 图 1 中顶点 v_0 核值更新为 1 时, 将该核值发送给其邻接点 v_{11} 和 v_2. 在下一轮迭代过程中, 顶点 v_{11} 因接收来自 v_0 的新消息而被激活, 进而重新计算自己的核值. 在大规模图核分解过程中, 这种冗余计算和消息传递会带来额外开销, 从而影响算法的效率.

分布式图计算的性能不仅与算法相关还取决于系统计算模式. 现有 Pregel[17]、Giraph[18]、GraphX[19]和 Quegel+[20]等分布式图计算系统多采用以顶点为中心计算模式. 该模式可高效处理大规模图计算任务, 但是对于以局部计算

为主的任务资源利用率不高[21]. 如图 1, 若顶点 v_{11} 和 v_0 存储在同一计算节点上, 当 v_{11} 被删除时, v_0 可通过访存的方式直接获取该信息并对自身核值进行更新, 但是在以顶点为中心计算模式中, v_0 只能在下一轮计算中被激活并更新其核值, 如此会增加迭代次数, 进而影响算法性能.

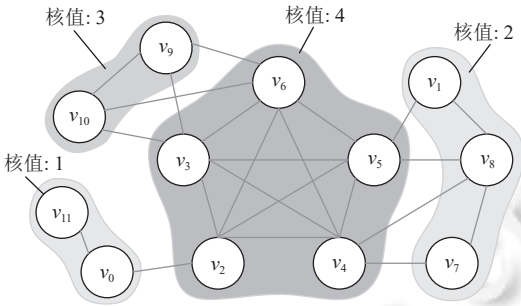


图 1 核分解示例图

针对现有分布式核分解算法和分布式图计算系统存在的不足和面临的挑战, 本文首先针对核分解过程中存在的冗余通信设计有效的消息传递策略. 当带宽资源受限时, 此方案可有效减少通信同步的时间; 其次, 当前串行核分解算法不适用于分布式核分解, 提出一种面向分布式系统的核分解新算法; 最后, 针对以顶点为中心计算模式处理局部计算存在的资源利用率不高的问题, 提出以计算节点为中心的计算新模式, 其中局部计算采用及时处理的策略, 用以减少全局迭代次数. 综上所述, 本文从算法设计、消息传递和分布式系统计算模式这 3 个方面着手, 提出了新的分布式核分解算法, 主要贡献如下.

(1) 面向以顶点为中心的计算模式, 分别设计基于全局激活和层次剥离的核分解算法, 依据顶点核值局部性特征设计有效的消息剪枝策略, 减少冗余的消息传递和计算任务.

(2) 提出以计算节点为中心的计算新模式, 通过局部计算优先处理策略优化具有局部性计算任务时的性能.

(3) 基于国家超级计算长沙中心分布式集群, 首先, 在真实数据集上对不同核分解算法进行评测, 从计算负载、通信量、超步数以及总耗时多方面分析验证了所提模型和方法的有效性和高效性; 随后, 通过构建合成数据集并实验, 分析不同算法对图数据规模的可扩展性.

本文第 2 节介绍关于分布式系统和核分解的相关工作. 第 3 节给出核分解相关的基本定义以及关于核分解单机和分布式的基础研究. 第 4 节分别介绍以顶点为中心计算模式下基于层次剥离和全局激活的分布式核分解算法以及对应消息剪枝策略. 第 5 节分别介绍以节点为中心计算模式下基于层次剥离和全局激活的分布式核分解算法. 第 6 节在真实数据集和合成数据集上评估本文所提算法的性能. 第 7 节对本文进行总结并展望未来工作.

2 相关工作

本节将分别介绍图计算系统及其计算模式、图核分解的相关工作.

大规模图数据广泛存在于社交网络、生物信息网络等实际应用中, 因此大规模图数据处理已成为当前研究热点[22]. 分布式图计算系统作为处理大规模图数据的重要工具也日趋成熟[23]. 当前应用最广泛的图计算系统为 Pregel 以及 Pregel-like 系统, 均采用 BSP 模型框架. 如图 2 所示, 超步作为其基本计算单元包括接收消息、计算以及通信 3 个步骤, 单个超步的成本和总成本如公式 (1) 和公式 (2) 所示.

单超步时间成本:

$$CPS = \max(w_i) + \max(h_i g) + L \tag{1}$$

总时间成本:

$$CA = \sum_{i=1}^{t} (\max(w_i) + \max(h_i g) + L) \tag{2}$$

上述公式中, w_i 是进程 i 的局部计算时间, h_i 是进程 i 发送或接收的最大通信包数, g 是带宽的倒数 (时间步/通信包), L 是全局栅栏同步时间. 所以, 在 BSP 计算中, 如果使用了 t 个超步, 则总的运行时间如公式 (2) 所示. 在本文实验中单个计算节点启动 1 个 MPI 进程, 因此此处的进程 i 即为计算节点 i.

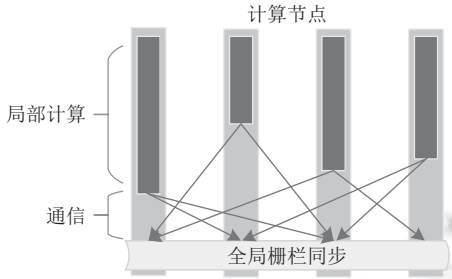


图 2 BSP 模型

赵翔等人[24]提出一种基于分离器-合并器的 BSP 计算模型, 有助于实现负载均衡并减小通信开销. 于戈等人[25]从图数据管理与处理机制两个方面, 分析了大规模图数据处理中计算模型、通信机制、分割策略、索引结构和容错管理等关键问题. 陆李等人[26]提出了度变异系数和分片通达度两个特征参数, 用于优化 CPU+GPU 异构环境下图计算系统研究平台和负载特征感知的在线分割算法的性能. 景年强等人[27]提出了一种并发更新的图计算模型 SpecGraph, 通过解耦合计算模型、异步执行引擎和并发更新机制提高系统实时性.

随着大量新型应用场景的出现, 针对具体应用需求的图计算系统也取得长足发展. Yan 等人[28]针对现有图计算系统不能高效处理轻量级图分析算法的局限, 提出可用于批量处理轻量级任务的图计算系统 Quegel, 并设计了共享超步策略以提高带宽利用率. 为减少全局同步次数, 分布式系统 Blogel[29]基于图划分技术将计算分为块内计算和块间计算, 但是其对原图的预处理划分将带来机器间顶点迁移等额外开销, 由于核值分解针对静态图, 不是增量式算法, 因此额外的图划分开销不宜引入. Yan 等人[30]提出了针对子图挖掘的图计算系统 G-thinker, 通过高并发的顶点缓存技术和轻量任务调度策略, 实现计算与通信时间的相互覆盖, 显著提高了系统性能. Wang 等人[31]就现有图计算系统无法兼顾易操作、高效和可扩展性 3 个方面的局限, 提出灵活便利的图计算系统 Graph3S, 并引入局部顶点优先激活策略, 以减少通信开销, 但是对于轻量级任务可能引入冗余计算开销.

顶点核值是图数据中顶点的重要属性, 可用于挖掘顶点所在社区的紧密性等图结构信息[2−6], 而图核分解的目标是获取所有顶点的核值. 基于单机内存和单机核外图计算系统的核分解问题一直以来都是图数据库领域的研究热点[8,11,12,32]. 文献 [13] 基于 GPU 进一步加速单机核分解计算过程. 因单机存储瓶颈, 传统服务器难以满足大规模图数据的存储需求, 为此基于分布式存储的核分解算法逐渐获得重视[14,15]. 因大量冗余的消息传递和计算开销, 现有分布式核分解算法性能仍存在大量改进空间. Luo 等人[33]研究了概率图上的分布式核值分解问题, 在顶点已知边的存在概率的情况下, 该算法能够以高概率保证得到节点的精确核数. Liu 等人[34]研究了二部图上核值分解算法, 并提出了基于层次剥离的算法由于需要对顶点进行排序不适用于分布式环境的观点.

综上所述, 尽管现有分布式图计算系统可有效处理大部分图计算任务, 但是对于核分解操作的性能仍需进一步优化. 此外, 现有核分解算法存在大量的冗余通信和计算开销, 无法高效分解大规模图数据. 为此, 本文将结合分布式计算和 BSP 模型特点对系统计算模式和核分解算法进行优化, 以提高分布式核分解效率.

3 问题定义

本节首先给出分布式图核分解的理论基础, 然后介绍现有单机最优核分解算法和分布式核分解算法.

3.1 基本定义

本文主要研究无向无权图 $G=(V,E)$ 上核分解问题, 其中 V 为图中顶点集合, E 为所有顶点之间连边的集合,

相关符号定义如表 1 所示. 图中顶点 v 的邻接点集合记为 N_v, $|N_v|$为 v 的度数 $d_G(v)$. 需要声明的是文中顶点指图中的实体, 邻接点指图中顶点的邻居顶点, 计算节点或节点指分布式环境中单个处理器. 对于一个诱导子图 $H(V',E')$, 满足 $V' \subseteq V$ 和 $E' = \{(u,v) \in E | u \in V' \wedge v \in V'\}$. 与诱导子图相对应的诱导度数表示顶点 v 在 H 中的邻接点数, 记作 $d_H(v)$, 其中最小诱导度数记为 $\delta(H) = \min(d_H(v) | v \in H)$.

表 1 常用符号定义

符号	定义
$G=(V,E)$	图 G 由顶点集合 V 和边集合 E 组成
$d_G(v)$	顶点 v 在 G 中的邻接点数, 即 v 的度数
N_v	顶点 v 的邻接点集合
$H(V',E')$	G的子图 H
$d_H(v)$	顶点 v 在子图 H 中的邻接点数
$\delta(H)$	子图 H 中顶点的最小度数
$C(v)$	顶点 v 的核值, 具体值表示为 k

定义 1 (子 k 核). 给定一个连通子图 H, 当且仅当 H 中所有顶点诱导度数大于等于 k 时, H 为一个子 k 核, 即满足 $\delta(H) \geqslant k$.

定义 2 (最大 k 核). 给定一个子图 H, 当且仅当 H 中所有顶点诱导度数大于等于 k, 且不存在其他子图 H'满足 $H \subseteq H' \wedge \delta(H) \geqslant k$, 则 H 为一个最大 k 核, 简称 k 核.

定义 3 (核值). 在图 G 中, 任意顶点 v 满足其属于 k 核但不属于任一 k+1 核, 则其核值为 k, 记作 $C(v)=k$.

如引言中所述通过顶点核值可以用于分析图结构紧密性程度, 具体而言, $C(v)$ 决定顶点 v 所能在的社区分布.

问题 1 (核分解). 给定图 G, 计算图中所有顶点的核值.

由定义 2 和定义 3 可知顶点 v 核值的大小依赖于核值不小于 $C(v)$ 的邻接点数量, 该类邻接点被定义为 v 的支撑点.

定义 4 (支撑点). 顶点 v 的邻接点中, 核值大于等于 $C(v)$ 的邻接点称为 v 的支撑点.

支撑点的存在说明单个顶点核值受其邻接点核值直接影响, 也即核值具有局部性特点.

3.2 基础研究

当前的核分解算法主要分为两类即层次剥离策略的单机核分解算法及基于全局激活策略的分布式核分解算法.

3.2.1 基于层次剥离的单机核分解算法

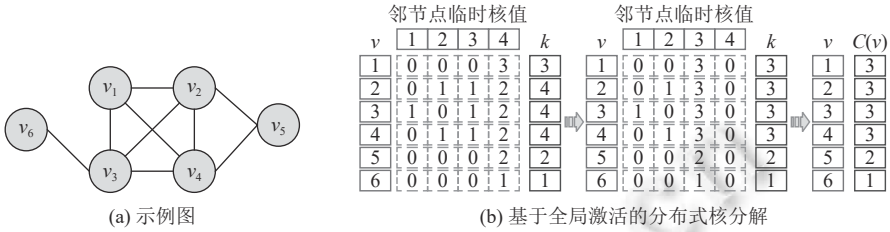
单机图核分解算法[8,9]中逐个删除核值最小的顶点并更新其邻接点核值, 迭代至所有顶点核值收敛. 给定图 $G=(V,E)$, 顶点的核值由各自邻接点核值决定, 同时各顶点的核值也影响着其邻接点的核值. 根据定义 2 和定义 3, 顶点的核值决定其所能在的最大 k 核, 而子图 H 的最小诱导度数 $\delta(H)$ 是由子图中拥有最少邻接点的顶点决定的. 因此从全局来看, 可通过逐个删除拥有最小度数的顶点对全图进行核分解, 而顶点被删除时的度数即为最终核值. 该方法在每一轮迭代中都需要获取全局度数最小的顶点集合, 而分布式环境下对全局顶点进行排序会引入大量的通信同步开销, 因此该核分解方法尚未应用于分布式环境中.

3.2.2 基于全局激活的分布式核分解算法

基于层次剥离的核分解算法不同, 现有基于全局激活的分布式核分解算法在各超步计算完成后, 所有核值发生改变的顶点都将在下一超步中被激活, 并更新自身核值[15,16].

在算法初始化阶段, 各顶点将核值 k 初始化为各自的度数, 即 $v.k = d_G(v)$. 紧接着将核值发送给所有邻接点, 接收到消息的顶点根据邻接点最新核值更新自身核值 k, 如果 k 发生改变 (减小) 则将新的 k 值再次发送给所有邻接点. 尽管该核分解算法不需要进行全局顶点排序, 但是当顶点核值发生变化的时候需要将消息发送给所有邻接点, 该过程会造成大量冗余消息传递, 进而造成更多顶点被激活, 增加了冗余计算量.

如图 3 所示, 给定一个包含 6 个顶点, 9 条边的图 G. 如图中顶点 v_2 初始化核值为 4. 在下一超步中将接收到邻接点 v_1、v_3、v_4 和 v_5 的核值, 其中仅有邻接点 v_1、v_3 和 v_4 的核值不小于 3, 需将顶点 v_2 的核值 $v_2.k$ 更新为 3, 并将其再次发送给顶点 v_1、v_3、v_4 和 v_5. 图 G 中其余各顶点计算模式与 v_2 相同, 最后所有顶点核值收敛时算法可结束.



(a) 示例图 (b) 基于全局激活的分布式核分解

图 3 基于全局激活的分布式核分解示例

4 以顶点为中心计算模式

分布式图计算系统多采用以顶点为中心计算模式 (vertex-centric), 该模式以超步为基本迭代单位, 每个超步包含接收消息、计算和发送消息 3 个步骤 (注: 在此模式下, 所有顶点都被抽象为独立的计算节点且处理方式一致). 本节基于以顶点为中心计算模式设计基于层次剥离和全局激活的分布式核分解算法, 并根据核值局部性特性提出有效的消息剪枝策略.

4.1 基于 vertex-centric 的层次剥离核分解算法

第 3.2.1 节中介绍的单机核分解算法思想同样适用于分布式环境. 与单机不同之处在于分布式环境下图 G 被分割为多个子图, 并分布式存储于多个不同的计算节点. 本文统一采用文献 [20] 中的图划分方法, 根据顶点编号对原图以 hash 的方式进行划分. 为发挥分布式系统多处理核心的并行计算能力, 在各超步将采用一种新的按层批量剥离的方式删减度数最小顶点.

引理 1 (批量剥离策略). 在核分解过程中, 同时拥有最小核值的顶点可被批量删减.

证明: 给定图 G, 在第 i 轮迭代中余下顶点构成的子图为 $H \subseteq G$, 全局最小核值即为 H 中诱导度数最小 $\delta(H)$ 的顶点集合 S, 则 S 中的顶点拥有且至多拥有 $\delta(H)$ 个核值大于等于 $\delta(H)$ 的邻接点, 依据定义 3, 集合 S 中的顶点核值等于 $\delta(H)$.

在核分解过程中, 单机核分解算法无法直接用于分布式环境, 主要原因如下.

(1) 单节点内部无法获取全局核值最小的顶点;

(2) 无法通过访存的方式获取邻接点的临时核值.

此外, 在 BSP 模型中, 一个超步包括接收消息、计算和通信 3 个步骤, 且顶点在接收到信息后才会被激活并执行计算任务. 针对上述问题本文设计出一种超步分解策略, 将一个超步拆分为两个子超步, 在不同子超步执行不同类型任务.

策略 1 (超步分解). 在分布式环境下实现基于最小核值剥离的核分解算法, 需要将 BSP 模型中单个超步分解为两个子超步, 各自执行任务如下.

子超步 1. 获取全局核值最小顶点, 发送核值信息;

子超步 2. 接收消息与核值更新.

由策略 1 可知, 在第 1 个子超步中所有顶点不会接收到消息, 而核值最小顶点在被删除时确定核值并告知其邻接点以让它们更新各自的临时核值, 因此系统需要主动激活该批节点让核分解流程正常执行; 在第 2 个子超步中, 接收到消息的顶点需要根据接收到的消息更新临时核值. 超步分解及对应任务分配方案如图 4 所示.

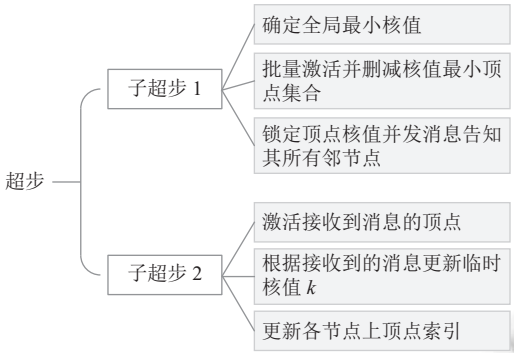


图 4 超步分解示意图

基于引理 1 和策略 1, 本文设计了分布式环境下的以顶点为中心 (vertex-centric) 的基于层次剥离的分布式核分解算法, 伪码如算法 1 所示.

算法 1. vertex-centric 基于层次剥离的分布式核分解算法.

输入: Graph $G=(V, E)$;

输出: Core numbers of all vertices in G.

```
/**********初始化与变量声明**********/
1. Declare ActiveVec and CoreIndex
2. Initialize globalPeelFlag←0, globalMinCore←0
3. 初始化顶点核值为其度数, 即 v.k←d_G(v)
/*************构建核值索引*************/
4. GenCoreIndex(CoreIndex)
/*******查找全局核值最小顶点集合*******/
5. FindPeelVertex(ActiveVec, globalMinCore)
/**************核分解过程**************/
6. WHILE CoreIndex is not empty DO
7.     IF globalPeelFlag=0 THEN // 执行子超步 1 任务
8.       FOR each v in ActiveVec DO
9.           v.corenum←globalMinCore // 确定核值
10.          send message to N_v // 激活邻接点
11.      END FOR
12.    ELSE // 执行子超步 2 任务
13.          msg←the number of messages received
14.          v.k← v.k – msg  // msg 为被删除邻接点数量
15.          v.k←max(v.k, globalMinCore)
16.          update the position of v in CoreIndex
17.      END IF
18.    globalPeelFlag = !globalPeelFlag // 超步模式切换
19.    IF globalPeelFlag = 0 THEN
20.          FindPeelVertex(ActiveVec, globalMinCore)
```

21. ELSE
22. synchronized barrier for communication
23. END IF
24. END WHILE
25. RETURN $C(v)$|for $v \in V$

函数 1. GenCoreIndex(&*CoreIndex*).

输入: Graph $G = (V, E)$;
输出: hash_map *CoreIndex*. // 顶点核值索引

1. FOR each v in V DO
// 以顶点核值为“键”, id 为“值”构建索引
2. IF $v.k \in CoreIndex.key$ THEN
3. put $v.id$ into $CoreIndex[v.k]$
4. ELSE
5. $CoreIndex[v.k] \leftarrow \{v.id\}$
6. END IF
7. END FOR
8. RETURN *CoreIndex*

函数 2. FindPeelVertex(&*ActiveVec*, &*globalMinCore*).

输入: Subgraph $H = (V', E')$;
输出: *ActiveVec*, *globalMinCore*.

1. 定义变量 *globalMinCore* 为全局最小核值
2. 根据键值将 *CoreIndex* 按升序排列并确定 *globalMinCore*
// 统一全局 *globalMinCore*
3. *globalMinCore*←*MPI_Allreduce*(*globalMinCore*)
// 激活核值最小顶点
4. *ActiveVec*←*CoreIndex*[*globalMinCore*]
5. remove *CoreIndex*.[*globalMinCore*] from *CoreIndex*
// 输出下一超步被激活顶点以及全局最小核值
6. RETURN *ActiveVec*, *globalMinCore*

在详细描述算法 1 之前需要声明两个必要的函数, GenCoreIndex() 和 FindPeelVertex(). 其中函数 GenCoreIndex(). 根据顶点的临时核值 k 构建一个存储顶点 id 的 hash_map *CoreIndex*, 由 $(k, < id_0, id_1, \dots >)$ 构成, 即具有相同临时核值 k 的顶点的 id 存储在同一向量中, 其空间复杂度为 O(|V|). 该变量主要用于全局最小核值查找、顶点激活及算法收敛判定. 各计算节点执行函数 FindPeelVertex() 激活拥有全局最小核值的顶点并将这批顶点置于 *ActiveVec* 中并从 *CoreIndex* 中删除.

如算法 1 所示, 基于层次剥离的分布式核分解算法核心思想在于全局 (所有节点) 批量迭代删除核值最小的顶点, 同时更新各自邻接点的核值. 该算法包括初始化和计算两个阶段. 在初始化阶段, 主要定义一些辅助计算的变量及相关预计算. 第 1 行和第 2 行定义了算法逻辑实现中必要的变量, 其中 *ActiveVec* 存储下一超步被激活的顶点; *CoreIndex* 为一个 *hash_map* 类型的变量, 存储了临时核值 k 和顶点 id 的键值对; *globalPeelFlag* 为子超步标志

位, 决定子超步的执行顺序; *globalMinCore* 定义为当前超步全局核值最小值, 首先各个计算节点计算各自存储的子图中所有顶点的最小核值, 将通过 MPI 函数接口同步各计算节点中的最小值. 因顶点的核值上界为顶点的度数, 第 3 行将顶点的核值初始化为顶点度数. 在算法运行过程中, 为发挥分布式系统的并行能力将批量删减核值最小的顶点, 为此需系统主动激活待删除的顶点集合. 第 4 行构建基于临时核值的索引 *CoreIndex*, 在各超步结束阶段用于查找下一轮超步被激活的顶点集合, 即拥有最小核值的顶点集合, 具体实现过程如函数 1 所述. 第 5 行为第 1 超步开始前调用函数 2 获取全局最小核值并激活其对应的顶点集合的预计算部分.

算法第 6–24 行为一个计算超步, 由变量 *globalPeelFlag* 控制两种子超步遵循交替执行模式. 整个算法收敛条件为顶点临时核值索引被删减至空, 此时原图 G 中所有顶点的核值均收敛. 结合图 4 超步分解示意图, 在 *globalPeelFlag* 为 0 时执行子超步 1 中设定的任务, 即锁定被激活顶点的核值并发送核值信息给各自邻接点 (第 8–11 行), 接收到消息的顶点将在下一超步中被激活. 超步计算完成后将 *globalPeelFlag* 取非 (第 18 行), 因此接收到消息被激活的顶点会执行子超步 2 分配的任务 (第 13–16 行), 根据接收到消息的数量更新临时核值, 并更新 *CoreIndex* 的存储结构. 具体来说, 当某个顶点收到 i 个消息表示其核值应该减少 i. 若顶点 v 的临时核值由 k 减少至 $k–i$, 则需要将 v 从 *CoreIndex*[k] 中删除并添加至 *CoreIndex*[$k–i$] 中. 由上所述, 子超步 2 执行过程中不存在消息发送操作, 第 19–21 行调用函数 2 查找剩余顶点中核值最小顶点集合并激活参与下一超步的计算. 第 22 行通过 MPI 协议对子超步 1 产生的消息进行传递并同步. 算法收敛后, 输出顶点的最终核值 (第 25 行).

以图 5 为例, 给定一个包含 10 个顶点、11 条边的图 $G=(V,E)$, 并分布式存储在两个计算节点上. 算法 1 中初始化所有顶点的核值为各自的度数, 其中 $v_0.k=1$, $v_8.k=4$. 然后根据顶点的临时核值构建核值索引 *CoreIndex*. 如图 6 所示, 该索引结构由多组键值对组成, 其中“键”的值为现有顶点核值的集合, “值”由拥有相同核值的顶点的编号组成, 如初始化阶段, 顶点 v_6 的核值为 2, 则 v_6 的编号 6 存储在键为 2 的列向量索引中. 在第 1 个超步开始前, 顶点 v_0 和 v_5 具有全局最小核值, 因此这两个顶点将首先被激活. 由于此时 *globalPeelFlag* 为 0, 因此第 1 个超步将执行子超步 1 的任务, 即确定被激活顶点的核值 $C(v_0)=1$ 和 $C(v_5)=1$, 然后发送核值信息给各自邻接点. 第 1 个超步结束后 *globalPeelFlag* 被置为 1, 表示下一超步执行子超步 2 设定的任务, 即顶点 v_1 和 v_6 的临时核中由 2 减至 1, 同时更新 *CoreIndex* 结构. 第 2 个超步结束后 *globalPeelFlag* 被置为 0, 依据算法 1 第 22 行所述, 需要调用函数 2 查找剩余顶点中核值最小顶点集合并激活. 如此完成一个完整的删减迭代过程, 其余详细迭代过程如图 6 所示.

如图 5 所示, 顶点被删减后会给所有邻接点发送消息, 在第 2 轮核值确定时, 尽管 v_0 和 v_5 已经在第 1 轮计算中被删减了, 但是依然会在第 2 轮计算中分别收到来自 v_1 和 v_6 的消息, 这将导致额外的通信和计算开销. 对于此类现象, 系统为各顶点构建一个存储邻接点状态的向量, 并依据邻接点对应标志位对消息传递进行剪枝.

策略 2 (消息剪枝策略 ①). 给定的图 G 中任意顶点的核值依赖于其邻接点的核值. 为此, 当某顶点获取其精确的核值后将该核值信息发送给核值待定的邻接点.

增加策略 2 后, 对图 5 进行核分解, 各超步消息量和被激活顶点数如表 2 所示. 其中“*C*”表示当前超步被激活顶点数, 即计算量; “*M*”表示当前超步发送消息的数量, 即通信量. 由表中结果可以看出, 部分超步计算量和通信量都得到优化, 如超步 3 的计算量由 4 降为 2, 超步 5 的通信量由 10 降为 4.

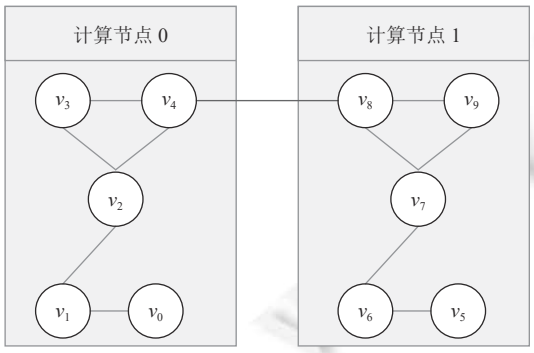


图 5 图数据分布式存储示例

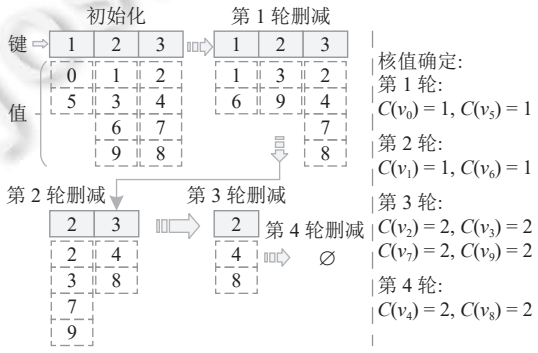


图 6 基于层次剥离的分布式核分解示例

表 2 基于层次剥离的分布式核分解算法性能对比

算法	超步 1		超步 2		超步 3		超步 4		超步 5		超步 6		超步 7		超步 8	
	C	*M*	*C*	*M*	*C*	*M*	*C*	*M*	*C*	*M*	*C*	*M*	*C*	*M*	*C*	*M*
算法 1	2	2	2	0	2	4	4	0	4	10	10	0	2	6	6	0
算法 1+策略 2	2	2	2	0	2	2	2	0	4	4	4	0	2	0	/	/
算法 3	4	4	6	0	6	14	6	0	/	/	/	/	/	/	/	/

4.2 基于 vertex-centric 的全局激活核分解算法

根据公式 (1), 单超步成本主要受最大单节点计算负载和通信量两个因素影响. 根据图 3 中示例的核分解过程解析可以看出只要顶点核值发生改变就会发消息给所有邻居, 这也将导致这些接收到消息的顶点在下一超步中被激活, 因此公式 (1) 中的 w_i 和 h_i 都存在大量冗余. 据此, 提出针对基于全局激活的分布式核分解的消息传递剪枝策略.

策略 3 (消息剪枝策略②). 所有顶点在临时核值更新后只发消息给核值比自身大的邻接点.

证明: 给定图 $G=(V,E)$, 假设存在顶点 v 及其邻接点 u, 且 $v.k<u.k$. 由于所有顶点的临时核值被初始化为各自的度数, 因此在核分解过程中各顶点的临时核值只有可能逐步减小. 在一轮超步计算完成后, v 和 u 的临时核值分别更新为 $v.k'$和 $u.k'$. 注意此时 v 和 u 的邻接点临时核值向量中存储对方的临时核值依然为 $u.k$ 和 $v.k$, 下面就 v 和 u 临时核值间的 4 种相对情况分别论证策略 3 结果的准确性.

(1) $v.k'<u.k'$ 且 $u.k'>v.k$. 顶点 u 的核值在更新过程中可能因为其他邻接点临时核值的变化使得 $u.k$ 减少至 $v.k$ 以下, 即 $u.k''<v.k$, 尽管 $v.k'<u.k'$, 顶点 v 不能作为顶点 u 的支撑点, 不会在当前超步对其核值更新产生影响. 但是如果出现 $u.k''<v.k$ 而 $u.k''>v.k'$ 时, 顶点 v 本不该成为 u 的支撑点, 而因为 u 的邻接点临时核值向量中关于 v 的记录依然是 k 使得 v 成为 u 的支撑点则会导致结果错误. 又因为临时核值全程处于非增状态, $u.k'>v.k$ 则 u 可永远作为 v 的支撑点, 不会影响其后续核值更新, 因此无须向 v 发送 $u.k'$.

(2) $v.k'<u.k'$ 且 $u.k'=v.k$. v 需要向 u 发送 $v.k'$如 (1) 中所证. 因 $v.k'\leqslant v.k$, 根据定义 4, u 可永远作为 v 的支撑点, 因此无须向 v 发送 $u.k'$.

(3) $v.k'<u.k'$ 且 $u.k'<v.k$. 因 $v.k<u.k$, 根据定义 4, u 为 v 的支撑点. 尽管 $v.k'<u.k'$, 此时实际上 u 依然为 v 的支撑点, 但是 u 的邻接点临时核值向量存储的关于 v 的临时核值依然是 $v.k$, 由于 $u.k'<v.k$, 所以 u 需要向 v 发送 $u.k$. 此次消息传递因 BSP 模型消息延迟特性而稍显冗余, 但是对最终结果无影响.

(4) $v.k'>u.k'$. 此种情况下 u 依然不能作为 v 的支撑点, 而 v 成为 u 的支撑点, 需要 v 和 u 互传消息以扭转支撑关系. 由于 v 和 u 存储对方临时核值为 $u.k$ 和 $v.k$, 而 $v.k'<u.k$, $u.k'<v.k$, 按策略 3 规定, v 和 u 会互相给对方发送临时核值, 与所期一致.

综上, 策略 3 可保证结果正确性.

基于全局激活核分解思想, 结合策略 3 消息剪枝策略, 在以顶点为中心的计算模式下, 本文设计的基于全局激活的分布式核分解算法伪代码如算法 2 所示.

算法 2. vertex-centric 基于全局激活的分布式核分解算法.

输入: Graph $G=(V,E)$;
输出: Core numbers of all vertices in G.

/**********初始化与变量声明**********/
1. 初始化顶点核值为其度数并激活, 即 $v.k\leftarrow d_G(v)$
2. 定义 *NeiCoreVec* 存储邻接点核值
/**********核分解过程**********/

3. FOR v in *ActiveVec* DO
   // 更新邻接点核值向量 (*NeiCoreVec*)
4. FOR *msg* in messages DO
5. *NeiCoreVec*[*msg.u*]←*msg.k*
6. END FOR
   // 根据 *NeiCoreVec* 重新计算当前顶点核值
7. UpdateCoreNum(&*NeiCoreVec*)
   // 将更新后的核值发送给核值大于自身核值的邻接点
8. Send *msg*(v, *v.k*) to N_v 中核值比自身大的顶点
9. END FOR
10. clear *ActiveVec*
/* 全局消息同步, 接收到消息的顶点将被激活 */
11. synchronized barrier for communication
12. IF *ActiveVec* is not empty DO
13. goto Line 3
14. END IF
15. $C(v)$←*v.k* for all vertices
16. RETURN $C(v)$|for $v \in V$

函数 3. UpdateCoreNum(&*NeiCoreVec*).

输入: *NeiCoreVec*;
输出: updated *v.k*.

1. Declare *NeiCore* // 邻接点核值累积向量
   // 根据邻接点核值构建 *NeiCoreVec*
2. FOR each *value* in *NeiCoreVec* DO
3. IF *value* > $d_G(v)$ THEN
4. *value*←$d_G(v)$
5. END IF
6. *NeiCore* [*value*]←*NeiCore*[*value*]+1
7. END FOR
   // 根据支撑点累积数量更新核值
8. Find the maximal index *kindex* that satisfies
9. *v.k*←*kindex*

$$\sum_{i=\deg ree}^{kindex} NeiCore[i] \geqslant kindex$$

10. RETURN *v.k*

基于全局激活的分布式核分解算法核心逻辑在于根据邻接点核值对当前顶点的临时核值进行更新, 其计算过程如函数 3 所示. 根据定义 1 和定义 4, 可通过公式 (3) 对顶点核值进行更新.

$$C(v) = \max(key),\ \text{s.t.}\ |\{u \in N(v) | C(u) \geqslant key\}| \geqslant key \tag{3}$$

结合函数 3, 对算法 2 给出以下解析. 在初始化阶段 (第 1, 2 行), 各顶点初始化核值为其度数并激活, 同时构建

向量 *NeiCoreVec* 存储邻接点的临时核值. 在计算阶段, 各激活顶点根据接收到的消息更新各自 *NeiCoreVec* 中对应邻接点的临时核值, 再调用函数 3 对当前顶点的核值进行更新 (第 4–7 行). 如果顶点 v 的核值发生改变, 则需要依据策略 3 给核值比 $v.k$ 大的邻接点发送 $v.k$ (第 8 行). 在一个超步计算任务全部完成后, 即所有被激活的顶点都参与了计算, 需要通过 MPI 完成消息传递, 同时将接收到消息的顶点作为下一超步中被激活的顶点添加至 *ActiveVec* 中 (第 11 行). 通过多轮超步迭代, 当无顶点核值发生改变程序结束, 此时各顶点所维系的临时核值即为各自的核值 (第 15, 16 行).

以图 3(a) 为例对算法 2 的计算逻辑与过程进行举例说明. 因第 1 个超步需要进行初始化操作, 所有顶点被激活并将临时核值都赋值为各自的度数, 然后将临时核值发送给所有邻接点; 第 2 超步中, 所有顶点都接收到来自第 1 个超步的消息, 因此所有顶点被激活并调用函数 3 更新各自的临时核值, 示例中顶点 v_2、v_3 和 v_4 的核值由 4 变为 3. 以 v_3 为例, 接收到第 1 个超步各邻接点发送来的消息后, 其邻接点核值向量为 $NeiCoreVec=\{1,3,4,4\}$, 由函数 3 计算得其核值为 3, 相较于原先的 4 减少了 1, 因此需要发送消息给其邻接点. 依据策略 3, 在 *NeiCoreVec* 中 v_2 和 v_4 的临时的核值大于 3, 则 v_3 需要发送 $v_3.k$ 给 v_2 和 v_4. 同理, v_2 和 v_4 处理过程与 v_3 相仿. 所有顶点临时核值不再改变后即可确定各自核值. 各超步总计算负载和通信量如表 3 所示, 其中“*C*”表示当前超步被激活顶点数, 即计算量; “*M*”表示当前超步发送消息的数量, 即通信量. 可以看出, 第 2 个超步中算法 2 的通信量较基准算法减少 50%, 第 3 超步中算法 2 的计算量减少 50%, 且总体来看, 算法 2 在所有超步中计算负载或通信量较基准有提升或持平.

表 3　基准算法与算法 2 性能对比

算法	超步 1		超步 2		超步 3	
	C	*M*	*C*	*M*	*C*	*M*
基准算法	6	18	6	12	6	0
算法2	6	18	6	6	3	0

由全局激活的分解基本逻辑可看出该方案单超步通信负载比层次剥离的方式大, 尤其是第一个超步所有顶点被激活并发送临时核值给所有邻接点, 空间复杂度为 O(2m), 其中 m 为边数. 在实验中导致如 Friendster 这种超大规模数据集在计算过程中面临内存溢出问题, 对此采用了一种环式消息传递模式[16].

策略 4 (环式消息传递). 在消息传递过程中, 各节点以自身为起点, 依次将所需传递消息按计算节点编号传递给其他计算节点.

如图 7 所示, 各计算节点编号为 MPI 进程号, 在单个超步的通信阶段, 首先对消息按目标节点进行分类, 如节点 n_1 将消息分为$\rightarrow n_1$(即自身)、$\rightarrow n_2$、...、$\rightarrow n_m$ 消息块, 表示需要发送给各节点的消息; 在执行过程中按顺时针顺序依次处理消息的传递任务, 如节点 n_1 首先传递消息$\rightarrow n_1$ 给节点 n_1, 再传递消息$\rightarrow n_2$ 给节点 n_2, 以此类推. 此外, 在节点 n_1 传递消息给节点 n_1 时其他节点也执行消息传递给自身的任务, 如此可实现子轮次消息传递过程中单个计算节点只接收一个节点传递来的消息, 从而解决内存溢出问题.

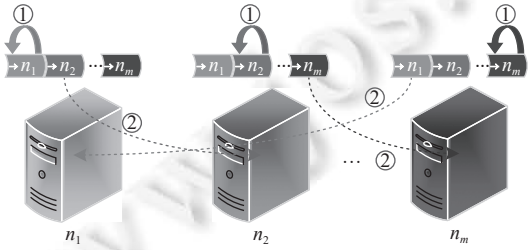


图 7　环式消息传递示意图

5　以节点为中心计算模式

第 4 节中, 在以顶点为中心的计算模式下实现了基于层次剥离的分布式核分解算法, 本质上是以核值最小的

顶点作为源点, 超步为单位, 将顶点被剥离的影响力传递给各自的邻接点. 在核分解过程中, 顶点再根据接收到的消息进行计算时只需要根据消息量更新当前临时核值即可, 因此对消息的处理次序无依赖性.

以图 5 为例, 顶点 v_0 被删减并确定核值后, 顶点 v_1 可通过访存的方式直接获取本该下轮才能接受到的消息, 进而提前执行相同的计算任务 (子超步 1 的任务), 通过多轮累积可减少总超步数. 基于 BSP 模型总开销公式 (2), 超步数对整体效率具有关键性影响, 节点间通信构建次数与之成正比. 基于上述理论与示例分析, 本节提出一种以节点为中心计算模式 (worker-centric), 实现同节点内部顶点尽早执行相同的计算任务.

定义 5 (以节点为中心计算模式). 超步计算任务完成后, 对于即将发送的消息, 如果其目的顶点与源顶点在同一计算节点上, 则立即激活该目的顶点并依据消息内容执行计算任务.

5.1 基于 worker-centric 的层次剥离核分解算法

根据超步分解思想, 只有执行子超步 1 的任务时才会有消息的传递. 此外, 为保证子超步 1 与子超步 2 交替执行, 需提出有效的节点内优先计算和消息传递机制.

策略 5 (优先激活策略). 对于子超步 1 模式下产生的消息传递, 如果目的顶点与源顶点存储在同一计算节点上且目的顶点的临时核值 $k = globalMinCore + 1$, 则该批目的顶点被立即激活并执行子超步 1 的计算任务.

证明: 假设当前为子超步 1 模式, 顶点临时核值索引为 *CoreIndex*, 剩余顶点中全局最小核值为 *globalMinCore*, 根据算法 1 第 20 行, 此时 *CoreIndex* 中临时核值为 *globalMinCore* 的顶点已被激活并删减. 按算法 1 执行逻辑, 当计算任务完成后, 确定核值的顶点需要发消息通知其邻接点, 其中临时核值为 *globalMinCore*+1 顶点的临时核值将在下一超步更新为 *globalMinCore* 并成为下一轮核值最小、需要被激活删减的顶点, 且最终确定核值为 *globalMinCore*. 根据策略 5, 在子超步 1 的模式下直接激活本该在下一超步被激活的临时核值为 *globalMinCore*+1 的顶点, 并将其核值确定为 *globalMinCore*, 再发送消息给其邻接点. 由于所有顶点计算更新核值的过程相互独立, 计算过程和通信过程与以顶点为中心计算模式相同.

依据策略 5 以及消息剪枝策略 2, 以节点为中心的基于层次剥离的分布式核分解算法如算法 3 所示.

算法 3. 以节点为计算中心的基于层次剥离的分布式核分解算法.

输入: Graph $G = (V, E)$;

输出: Core numbers of all vertices in G.

```
/**********初始化与变量声明**********/
1. Declare ActiveVec, CoreIndex
2. Initialize globalPeelFlag←0, globalMinCore←0
3. 初始化顶点: 核值为其度数, 即 v.k←d_G(v); 各邻接点顶点核值确定标志位 NeiCoreFlag 为 0
/*************构建核值索引*************/
4. GenCoreIndex(CoreIndex)
/*******查找全局核值最小顶点集合*******/
5. FindPeelVertex(ActiveVec, globalMinCore)
/**************核分解过程**************/
6. WHILE CoreIndex is not empty DO
7.    IF  globalPeelFlag = 0  THEN // 执行子超步 1 任务
8.       FOR each v in ActiveVec DO
9.          v.corenum←globalMinCore // 确定核值
10.         激活 {u|u ∈ N_v 且 u ∈ CoreIndex[globalMinCore + 1]}
11.         Send C(v) to other neighbors s.t. NeiCoreFlag[u] = 0
12.      END FOR
```

13. ELSE // 执行子超步 2 任务
14. update core number according to reveived messages
15. update the position of *v* in *CoreIndex*
16. END IF
17. *globalPeelFlag* =!*globalPeelFlag* // 超步模式切换
18. IF *globalPeelFlag* = 0 THEN
19. FindPeelVertex(*ActiveVec*, *globalMinCore*)
20. ELSE
21. synchronized barrier for communication
22. END IF
23. END WHILE
24. RETURN $C(v)$|for $v \in V$

由算法 3 描述可知, 其与算法 1 主要差别在于融合了策略 2 消息剪枝的基础上对同节点内部的顶点依据策略 5 提前激活并计算. 相比算法 1, 可有效减少总超步数. 以图 5 为例, 如表 3 所示, 总超步数较算法 1 优化了 50%, 但是单个超步的计算量和通信量增加了, 因此该优化策略更适用于带宽小、通信能力较差的网络环境.

5.2 基于 worker-centric 的全局激活核分解算法

以节点中心的计算模式同样可用于加速基于全局激活的分布式核分解算法, 根据定义 5, 各顶点临时核值发生改变并发送消息给邻接点时, 如果存在目的邻接点与当前顶点存储在同一计算节点上, 则可直接通过访存方式获取消息并直接计算. 又因为基于全局激活的核分解方式不具备基于层次剥离的核分解方式中明确的分层激活机制, 因此此处不对本地顶点提前激活进行其他限制.

为避免因部分顶点在同一超步中被存储在同一计算节点的邻接点多次激活, 从而导致消息传递混杂造成计算错误, 将对本地激活顶点进行分批处理, 同时在处理消息的时候增加一层防错机制. 具体步骤如算法 4 所示.

算法 4. worker-centric 基于全局激活的分布式核分解算法.

输入: Graph $G = (V, E)$;
输出: Core numbers of all vertices in G.

/**********初始化与变量声明**********/
1. Declare *ActiveVec*, *LocalActiveVec*, and *LocalMsg*
2. 初始化顶点: 核值为其度数, 即 $v.k \leftarrow d_G(v)$; 各邻接点顶点核值确定标志位 *NeiCoreFlag* 为 0
/**************核分解过程**************/
3. FOR *v* in *ActiveVec* DO
4. FOR *msg* in messages DO
// 防错机制: 邻接点核值只可能减少
5. IF $msg.k \geqslant NeiCoreVec[msg.u]$ DO
6. $NeiCoreVec[msg.u] \leftarrow msg.k$
7. END IF
8. END FOR
// 根据 *NeiCoreVec* 重新计算当前顶点核值
9. UpdateCoreNum(&*NeiCoreVec*)
10. IF *v.k* changed DO
11. FOR *u* in N_v DO

```
  // 目的顶点在同计算节点内的消息将目的顶点核消息分别存储于 LocalActiveVec 和 LocalMsg 中
12.         IF local(u) and NeiCoreVec[u] > v.k
13.           LocalActiveVec.push_back(u)
14.           LocalMsg.push_back(msg(v, v.k))
  // 将核值发送给核值大于自身核值的邻接点
15.         ELSE IF NeiCoreVec[u] > v.k
16.           send msg(v, v.k) to u
17.         END IF
18.       END FOR
19.     END IF
20. END FOR
21. clear ActiveVec
  // 同计算节点内接收到消息的顶点在当前超步被激活并执行计算任务
22. IF LocalActiveVec is not empty DO
23.     ActiveVec←LocalActiveVec
24.     messages←LocalMsg
25.     goto Line 3
26. END IF
  // 全局消息同步, 接收到消息的顶点将被激活
27. synchronized barrier for communication
28. IF ActiveVec is not empty DO
29.     goto Line 3
30. END IF
31. C(v)←v.k for all vertices
32. RETURN C(v)|for  v ∈ V
```

结合全局激活思想, 如算法 4 所示, 在以节点为中心的计算模式下, 存储于同节点中的顶点会被提前激活并参与计算. 算法 4 在初始化阶段构建了两个额外的向量 *LocalActiveVec* 和 *LocalMsg*, 分别用来存储局部激活顶点及其接收到的消息. 第 5 行增加防错机制, 依据所有顶点的临时核值在整个分解过程中只会呈现非增趋势, 通过限制只接收比当前临时核值小的消息即可规避因多次激活造成的紊乱. 与算法 2 相同, 顶点的临时核值发生改变时需要告知其邻接点, 不同的是对于存储在同一计算节点上的邻接点及其将接收到的消息存储在 *LocalActiveVec* 和 *LocalMsg* 中 (第 12–14 行). 其中 *local*(*u*) 用以判断邻接点 *u* 是否与当前被激活顶点存储在同一计算节点上. 当一轮计算结束后, 若 *LocalActiveVec* 不为空则说明有本地节点可被直接激活, 分别提取激活顶点和对应消息后直接进入计算流程 (第 22–26 行). 实现本地消息传播突破单跳邻接点, 如此处 *u* 作为 *v* 的本地邻接点被直接激活, 如果 *u* 的临时核值发生改变则可继续扩散给自己的邻接点, 这也有可能使得顶点 *v* 的临时核值再次改变并再次发送消息给邻接点, 此时第 9 行设计的防错机制将起到关键性作用. 当所有计算节点不存在局部激活顶点时即可进入全局栅栏同步, 通过 MPI 进行消息传递并进入下一超步.

6 实　验

本节设计了多组实验验证以顶点为中心和以节点为中心两种计算模式下基于层次剥离和全局激活两种分布式图核分解算法在真实数据集上的整体性能, 并通过对通信量和计算量的统计分析评估了优化策略的有效性. 此外, 为验证不同算法对图规模属性的适应性, 本文还在多个不同规模合成数据集上验证并分析了算法性能趋势.

6.1 实验配置

● 实验环境. 本文分布式核分解算法被部署于国家超级计算长沙中心 (TH-I) 上, TH-I 配备 160 Gb/s 高速互联系统, 单个计算节点包含 2 块 Intel(R) Xeon(R) CPU, 拥有 48 GB 主存. TH-I 底层 MPI 为自主实现, 基于 Intel 编译器进行编译, 本文中算法的实验代码契合此环境. 本文采用 10 个计算节点构建分布式环境. 实验代码采用 C++编写, 通过 MPI (消息传递接口) 实现不同计算节点间消息通信, 最后由 mpicxx 编译为可执行文件, 因此通用性较好, 适用于不同分布式环境.

● 实验设置. 本文以现有基于全局激活的分布式核分解算法[15,16]和基于层次剥离的分布式核分解算法作为基准, 引入消融实验对比分析其他两种计算模式及各种优化策略的结果. 具体来说, 首先对比不同分布式核分解算法的整体性能, 包括总执行时间 (T) 和超步数 (S). 考虑到数据集规模差异导致执行时间和超步数相差悬殊, 此处采用相对基准算法性能优化比例 θ, 其计算方式为:

$$\theta(t)=\frac{T_b-T_a}{T_b} \tag{4}$$

其中, $\theta(t)$ 表示其他算法与基准算法在执行时间消耗上的优化比例, T_b 为基准算法消耗的时间, T_a 为其他对比算法时间的消耗. 然后对于不同计算模式和优化策略, 依据 BSP 系统特性, 落脚于单超步最大计算量和总通信量两个方面, 分析各模式和策略的优劣性. 最后, 构建与真实数据集同分布的合成数据集, 评估不同算法对图规模的可扩展性.

● 数据集. 本文通过真实数据集和合成数据集验证算法性能. 表 4 给出了 7 个真实数据集的基本情况. 图 8(a)为所用真实数据集核值分布情况, 纵坐标为顶点累积量, 横坐标为核值. 由顶点累积量增长率可以看出真实数据集核值分布服从幂律分布, 实验中此作为构建合成数据集的准则. SNAP[35]工具包可用于生成多种模式数据集, 文献中常用的为 Erdos-Renyi 模式 (ER)、Barabasi-Albert 模式 (BA) 和 R-MAT 模式 (R-MAT). 以 2^{24} 顶点规模为例, 3 种模式生成的图中顶点核值分布如图 8(b) 所示, 可以发现只有图 RMAT-24 的核值分布服从幂律分布, 因此需在 R-MAT 模式下构建顶点规模为 2^{16}–2^{24} 的合成数据集 (RMAT-16 至 RMAT-24) 来评估算法对数据集规模的适应性. Hash 划分是一种基于 edge-cut 的通用图划分方式, 并广泛应用于分布式图计算系统中[28], 具有划分效率高, 保障各计算节点上存储的顶点数量均衡和辅助消息传递过程中确定目的顶点所在计算节点编号等特点. 为此, 本文采用的所有数据集通过 hash 的方式进行分布式存储, 即根据顶点编号的 hash 值确定该顶点所在机器编号.

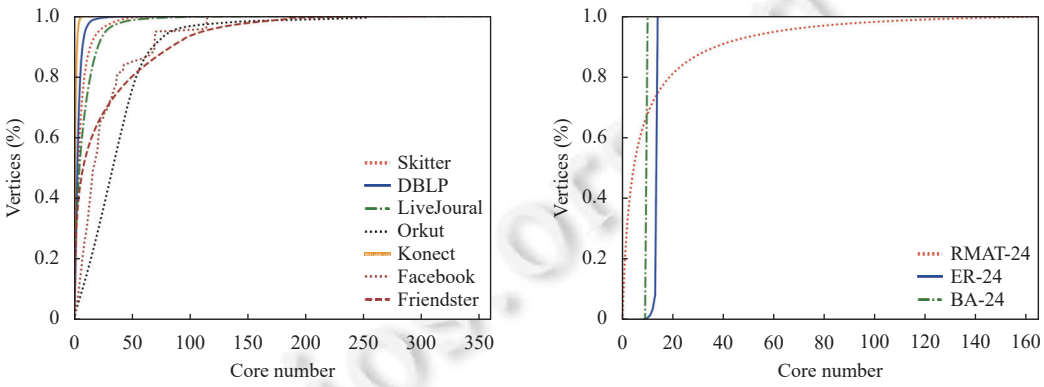
6.2 实验结果分析

本节首先从宏观角度对比不同算法相对基准算法的性能优化比, 对于实验结果所呈现的多种不同情况, 结合 BSP 系统特性再具体分析.

● 总超步数. 在基于 BSP 模式下的分布式系统中, 超步作为基本执行单元可用以统计算法迭代轮次, 最直观的反馈就在于计算节点间构建通信的次数. 因此超步数可用于衡量分布式系统用于消息同步的固有开销, 即公式 (1) 和公式 (2) 中的 L. 对于不同算法在真实数据集上的超步数消耗如表 5 所示. 通过消融实验的方式验证不同策略的性能, 其中 G 表示基于全局激活的方式进行核分解, P 示基于层次剥离的方式进行核分解, V 表示采用以顶点为中心的计算模式, N 表示采用以节点中心的计算模式, “*+”表示配合使用消息剪枝策略 (策略 2 或 3). 如“GV”表示在以顶点为中心计算模式下采用基于全局激活的方式进行核分解, 并且过程中不对消息作剪枝操作, 即基准算法; “PN+”表示在以节点为中心计算模式下采用层次剥离的方式进行核分解, 并且过程中配合策略 3 进行消息剪枝, 即算法 4. 由表中数据可发现基于层次剥离的分解方式所消耗的超步数比基于全局激活的分解方式多很多, 原因在于前者将一轮计算分为子超步 1 和子超步 2 两步完成, 此外子超步 1 中手动激活的顶点为临时核值相同的且为全局最小的顶点, 显然基于全局激活的方式单轮超步激活的顶点数要更多. 对于以顶点为中心和以节点为中心两种不同模式, 可发现后者在所有数据集上都获得提升, 即超步数消耗更少. 更具体地, 在基于全局激活的核分解算法中, 以顶点为中心计算模式较以节点为中心计算模式 (GN+ vs. GV) 超步消耗减少 4.0%–12.2%, 而在基于层次剥离的和算法中, 优化比例 (PN+ vs. PV) 仅为 4.3%–8.1%, 由此可以看出, 以节点为中心的计算模式给基于全局激活的核分解算法带来更多的增益.

表 4 真实数据集

数据集	$\|V\|$	$\|E\|$	$\|E\|/\|V\|$	RES
DBLP	317080	1049866	3.31	673M
Facebook	4039	88234	21.85	23M
LiveJournal	3997962	34681189	8.67	10G
Skitter	1696415	11095298	6.54	4G
Orkut	3072441	117185083	38.14	15G
Konect	59216211	92522017	1.56	115G
Friendster	65608366	1806067135	27.53	264G



(a) 真实数据集顶点核值分布 (b) 合成数据集顶点核值分布

图 8 数据集顶点核值分布

表 5 不同算法执行超步数

数据集	GV[16]	GV+	GN	GN+	PV	PV+	PN	PN+
Facebook	23	23	22	22	705	705	675	675
DBLP	25	25	24	24	497	497	461	461
Skitter	65	65	58	58	2391	2391	2205	2205
Orkut	193	193	171	171	11333	11333	10419	10419
LiveJournal	90	90	79	79	6009	6009	5589	5589
Konect	73	73	67	67	929	929	857	857
Friendster	—	—	—	—	20069	20069	18647	18647

● 总时间. 对于算法评估而言, 总时间是最直接也是最有效的衡量标准. 本文以基于全局激活的分布式图核分解算法 (不附加消息剪枝策略) 耗时为基准, 以算法 1、算法 3 及二者与策略 2 结合的方案, 算法 2、算法 4 及二者与策略 3 结合的方案为对比, $\theta(t)$ 作为衡量指标, 在真实数据集上结果如表 6 所示. 表中数据对应为各算法相对基准算法的优化比值, 其中基于层次剥离的算法总耗时需要加上初始顶点核值索引构建时间. 表 6 中加粗数值表示当前数据集相对基准算法性能提升最多的算法. 结合据表 4 中各数据集顶点数量以及边数量和顶点数量的比值, 对于稀疏数据集 ($|E|/|V|$小) 基于全局激活的核分解算法性能更优 (Facebook 由于顶点数量过少不具备代表性). 这是因为稀疏数据集其核值分布更为分散, 这将给算法 1 和算法 3 中顶点临时核值索引为 *CoreIndex* 的构建和维护带来更大开销, 降低基于层次剥离核分解算法性能. 值得注意的是 *CoreIndex* 的构建复杂度为 O($|V|$), 可见 Konect 数据集上两类算法表现差异很大的原因在于该数据集规模太大导致 *CoreIndex* 的构建和维护成本突增. 由于 Friendster 数据集规模太大以至于执行基于全局激活算法时消息传递量过大, 尤其是第 1 个超步所有顶点被激活且需要向邻接点发消息阶段 (算法 2 首次执行第 4–6 行), 在执行基准算法时采用策略 4 规避内存溢出问题, 因此在此处不作为常规分析.

表 6 不同算法总耗时性能提升比例

数据集	GV[16]	GV+	GN	GN+	PV	PV+	PN	PN+
Facebook	—	0.21	0.11	**0.37**	−1.13	−1.09	−1.05	−0.99
DBLP	—	0.41	0.07	**0.46**	−0.17	−0.05	−0.25	−0.11
Skitter	—	0.65	0.03	**0.66**	0.12	0.21	0.00	0.12
Orkut	—	0.6	0.08	0.61	0.85	**0.89**	0.84	0.88
LiveJournal	—	**0.57**	0.15	0.60	0.22	0.29	0.15	0.24
Konect	—	0.8	0.95	**0.98**	−17.01	−17.00	−18.52	−19.41
Friendster	—	—	—	—	0.67	0.69	0.66	**0.70**

为更细致分析不同算法性能差异性, 依据 BSP 模型成本计算公式 (公式 (1) 和公式 (2)) 对不同算法计算过程中单超步通信和计算负载进行分析.

● 单超步成本分析. 由公式 (1) 可知, 在 BSP 模型中, 单超步成本主要由最大计算负载和通信负载与带宽比值决定. 在本组实验中, 通过比对不同计算模型和策略对分布式系统中单超步最大计算负载和单超步总通信量的影响实现深层次分析不同算法对数据集规模的适应性. 为了更全面体现不同算法性能优化比的关系, 选取 DBLP 和 Orkut 作为深度分析数据集示例. 策略 2 和策略 3 提出的消息剪枝策略旨在降低当前超步通信量, 进而减少下一超步中被激活顶点数, 使得计算量进一步减少. 图 9 为 DBLP 和 Orkut 两个数据集在以顶点和节点为中心计算模式下单超步总通信量优化比.

$$ratio = \frac{M - M_+}{M} \tag{5}$$

其中, M 表示无消息剪枝策略下单超步通信量总和, 此处通信量为消息条数, M_+表示辅以消息剪枝策略下单超步通信量总和. 各子图表示各数据集在对应模式下各个超步的通信量优化比, 如 DBLP-GV 表示在以顶点为中心计算模式下通过全局激活的方式对数据集 DBLP 进行核分解. 很显然, 图 9(a), 图 9(b), 图 9(e), 图 9(f) 中的通信量优化比大多大于 0.6, 相较于子图 9(c), 图 9(d), 图 9(g), 图 9(h) 中的结果更加稳定也效果更佳. 该现象说明消息剪枝更适用于基于全局激活的核分解方式, 且与系统计算模式无明显关系. 因此表 5 和表 6 证明了消息剪枝策略的有效性.

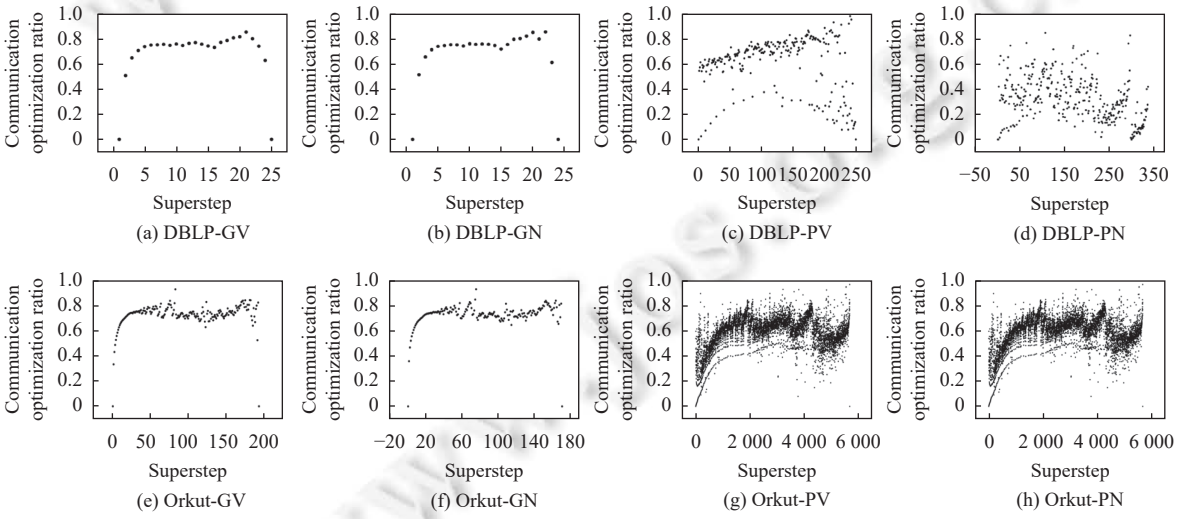


图 9 消息剪枝策略通信量优化比

由表 5 可知, 以基于层次剥离的方式对 Orkut 数据集进行核分解时超步消耗远大于基于全局激活的方式, 但是表 6 中显示后者的相对基准总耗时优化比远大于前者, 如 $T_{\text{PN-1}} = 49.4$ s, $T_{\text{GN-1}} = 162.4$ s. 根据公式 (1), 单超步成

本受计算量影响, 表 7 和表 8 为 DBLP 和 Orkut 第 1–10 超步各超步最 大单机计算负载, 即公式 (1) 中的 $\max(w_i)$. 在基于层次剥离的核分解算法过程中, 每轮迭代删减核值最小的顶点. 同一核值顶点数量是固定的, 在顶点剥离过程中, 同一核值顶点逐渐被删减, 总数逐渐减少, 因此呈现被激活顶点数逐渐减少的情况; 当最小核值顶点被删减完后, 次小核值顶点进入被剥离过程, 因此总激活数又呈现增多再减少的现象. 如此往复, 激活顶点数呈现减少增多交替的情况. 此外, 在服从幂律分布图数据中, 约 20% 的顶点拥有约 80% 的边, 尽管前面部分超步被同时删减的顶点数量多, 但是由于其度数小, 总通信量也不会导致内存溢出的问题, 因此基于层次剥离的核分解算法不会出现由于通信量过大导致内存溢出的场景.

表 7 不同算法在 DBLP 上单超步最大计算负载

超步	GV[16]	GV+	GN	GN+	PV	PV+	PN	PN+
S_1	31814	31814	31814	31814	4370	4370	4383	4383
S_2	31814	31814	43947	35725	2763	2763	2773	2773
S_3	31784	9033	37632	10666	232	232	213	213
S_4	24547	4836	26990	5516	463	220	448	211
S_5	17207	2978	17895	3229	28	28	23	23
S_6	11716	1952	11491	2032	55	24	48	24
S_7	7808	1327	7237	1352	3	3	2	2
S_8	4987	922	4578	862	6	3	4	2
S_9	3398	625	2970	579	6111	6111	6154	6154
S_{10}	2271	446	1842	392	7635	7323	7680	7350

表 8 不同算法在 Orkut 上单超步最大计算负载

超步	GV[16]	GV+	GN	GN+	PV	PV+	PN	PN+
S_1	307249	307249	307249	307249	6873	6873	6873	6873
S_2	307249	307249	963700	611207	4122	4122	4122	4122
S_3	307249	230518	880363	471221	6	6	6	6
S_4	306827	206751	813034	402691	9	5	8	5
S_5	306227	186168	756912	347060	4411	4411	4413	4413
S_6	305262	168495	707862	302883	6746	6742	6747	6743
S_7	303975	153312	665392	266056	60	60	59	59
S_8	302161	140074	629514	235450	166	114	161	110
S_9	299886	128632	595008	210208	4	4	4	4
S_{10}	296865	118601	562113	188457	16	11	15	10

从整体性能来看, 不难发现使用基于全局激活的方式进行核分解时单超步最大单机计算负载很高, 尤其是前几个超步. 因此对于 Orkut 数据集尽管基于层次剥离的方式在核分解时损耗的超步数更多, 但是单超步耗时相比基于全局激活的方式小, 从而使得总时间消耗更短. 此外, 由于单超步激活顶点多的情况下也会造成更高的通信负载, 会给消息同步带来更高延迟. 而随着图规模进一步增大, 构建顶点核值索引耗时增加, 使得总耗时对比再次翻转, 如 Konect 上, 基于层次剥离的分解方式的性能骤降, 主要原因在于其顶点核值索引耗时长.

● 可扩展性. 上述说明了不同算法在不同数据集上的不同性能表现, 但是所用的几个真实数据集规模悬殊较大, 无法定量确定根据数据集基本情况选择对应核分解算法. 鉴于此, 实验中构建 9 个合成数据集, 顶点规模为 2^{16}–2^{24}, 用以验证不同算法的可扩展性. 表 9 中加粗数值表示核分解总耗时最小. 在合成数据集总耗时结果可以看出在数据规模小于 2^{23} 时基于层次剥离的方式耗时更短, 主要原因在于图规模增长到 2^{24} 时顶点核值索引构建耗时相较于其他规模数据集大幅度增加 (表 9 *CoreIndex* 列), 使得基于层次剥离算法的总耗时超过基于全局激活的算法耗时. 从整体趋势来看, 合成数据集的结果未出现真实数据集上两类算法性能翻转的现象, 这与实际数据集核值分布有着极大关系, 影响计算过程中顶点核值索引更新频率, 是一项无法量化的消耗.

表 9 不同算法合成数据集核分解总耗时 (s)

数据集	GV[16]	GV+	GN	GN+	PV	PV+	PN	PN+	*CoreIndex*
RAMT-16	0.86	0.36	0.83	0.37	0.27	0.22	0.26	**0.21**	0.004
RAMT-17	2.14	0.84	2.10	0.86	0.50	0.40	0.50	**0.40**	0.012
RAMT-18	4.78	1.75	4.75	1.85	0.99	**0.74**	1.02	0.76	0.043
RAMT-19	9.69	3.33	8.44	3.18	1.85	**1.37**	1.93	1.46	0.172
RAMT-20	26.26	8.51	24.27	8.77	4.73	**3.50**	5.26	3.98	0.620
RAMT-21	71.01	20.68	63.87	21.12	**10.02**	7.07	14.59	11.49	2.611
RAMT-22	166.13	46.16	138.63	43.56	35.28	**28.87**	41.18	34.03	10.953
RAMT-23	482.17	124.16	389.45	112.67	112.29	**97.54**	132.66	119.30	41.501
RAMT-24	1 029.39	252.63	823.96	**221.11**	349.17	319.72	401.54	374.52	172.663

6.3 讨 论

分布式图计算系统中, 超步数和计算量为两大重要性能评估指标, 决定了分布式图分析算法总时间消耗. 如表 5 和表 6 所示, 尽管基于层次剥离算法超步消耗远大于基于全局激活的分解方式, 但算法总耗时性能提升比例更大. 因为前者各超步计算量远小于后者, 若在网络延迟高或带宽小的环境下, 过多的超步将造成更多时间消耗. 此外, 由表 6 可以总结出对于层次剥离的分布式核分解算法, 以节点为中心的计算模式优化效果并不佳, 甚至在大部分情况下是负优化, 因此在处理超大规模图数据时可以考虑使用以顶点为中心的层次剥离算法. 因此, 在网络延迟比较高, 通信时间为瓶颈的环境下, 尽量少的超步数可以有效提升算法效率, 宜采用基于全局激活的核分解算法; 同样环境下, 对于规模较小或较大的图数据, 以节点为中心的计算模式可以取得更好的性能.

7 结 论

本文提出分别基于层次剥离和全局激活的分布式核分解算法, 并根据顶点核值由支撑点分布决定的局部性特点设计了消息剪枝策略, 减少单超步通信量和计算量; 在系统优化方面, 提出一种以节点为中心的计算模式用于改善以顶点为中心计算模式处理局部计算时资源浪费的问题, 减少了执行超步数. 国家超级计算长沙中心集群上验证算法在真实和合成数据集上的性能. 通过实验结果分析, 消息剪枝策略通用性较好, 对基于全局激活的核分解算法提升效果更显著; 以节点为中心的计算模式有效减少了分布式核分解的超步数. 此外, 对于不同算法在不同数据集上的不同表现, 本文结合 BSP 系统超步成本公式就计算量和通信量两个方面进行深度分析. 由于实际应用中的图数据分布具有不确定性, 根据图数据本身属性自适应选择最有效的核分解算法是下一步的研究工作.

References:

[1] Seidman SB. Network structure and minimum degree. Social Networks, 1983, 5(3): 269–287. [doi: 10.1016/0378-8733(83)90028-X]

[2] Giatsidis C, Thilikos DM, Vazirgiannis M. D-cores: Measuring collaboration of directed graphs based on degeneracy. Knowledge and Information Systems, 2013, 35(2): 311–343. [doi: 10.1007/s10115-012-0539-0]

[3] Peng CB, Kolda TG, Pinar A. Accelerating community detection by using K-core subgraphs. arXiv:1403.2226, 2014.

[4] Balasundaram B, Butenko S, Hicks IV. Clique relaxations in social network analysis: The maximum K-plex problem. Operations Research, 2011, 59(1): 133–142. [doi: 10.1287/opre.1100.0851]

[5] Alvarez-Hamelin JI, Dall'Asta L, Barrat A, Vespignani A. Large scale networks fingerprinting and visualization using the k-core decomposition. In: Proc. of the 18th Int'l Conf. on Neural Information Processing Systems. Vancouver: MIT Press, 2005. 41–50.

[6] Li XL, Wu M, Kwoh CK, Ng SK. Computational approaches for detecting protein complexes from protein interaction networks: A survey. BMC Genomics, 2010, 11(S1): S3. [doi: 10.1186/1471-2164-11-S1-S3]

[7] Malliaros FD, Giatsidis C, Papadopoulos AN, Vazirgiannis M. The core decomposition of networks: Theory, algorithms and applications. The VLDB Journal, 2020, 29(1): 61–92. [doi: 10.1007/s00778-019-00587-4]

[8] Batagelj V, Zaversnik M. An O(m) algorithm for cores decomposition of networks. arXiv:cs/0310049, 2003.

[9] Khaouid W, Barsky M, Srinivasan V, Thomo A. K-core decomposition of large networks on a single PC. Proc. of the VLDB Endowment, 2015, 9(1): 13–23. [doi: 10.14778/2850469.2850471]

[10] Li CG, Zhang F, Zhang Y, Qin L, Zhang WJ, Lin XM. Efficient progressive minimum k-core search. Proc. of the VLDB Endowment,

2019, 13(3): 362–375. [doi: 10.14778/3368289.3368300]

[11] Li RH, Song QS, Xiao XK, Qin L, Wang GR, Yu JX, Mao R. I/O-efficient Algorithms for degeneracy computation on massive networks. IEEE Trans. on Knowledge and Data Engineering, 2022, 34(7): 3335–3348. [doi: 10.1109/TKDE.2020.3021484]

[12] Kabir H, Madduri K. Parallel k-core decomposition on multicore platforms. In: Proc. of the 2017 IEEE Int'l Parallel and Distributed Processing Symp. Workshops (IPDPSW). Lake Buena Vista: IEEE, 2017. 1482–1491. [doi: 10.1109/IPDPSW.2017.151]

[13] Mehrafsa A, Chester S, Thomo A. Vectorising k-core decomposition for GPU acceleration. In: Proc. of the 32nd Int'l Conf. on Scientific and Statistical Database Management. Vienna: ACM, 2020. 24. [doi: 10.1145/3400903.3400931]

[14] Montresor A, De Pellegrini F, Miorandi D. Distributed k-core decomposition. IEEE Trans. on Parallel and Distributed Systems, 2013, 24(2): 288–300. [doi: 10.1109/TPDS.2012.124]

[15] Mandal A, Al Hasan M. A distributed k-core decomposition algorithm on spark. In: Proc. of the 2017 IEEE Int'l Conf. on Big Data (Big Data). Boston: IEEE, 2017. 976–981. [doi: 10.1109/BigData.2017.8258018]

[16] Weng TF, Zhou X, Li KL, Peng P, Li KQ. Efficient distributed approaches to core maintenance on large dynamic graphs. IEEE Trans. on Parallel and Distributed Systems, 2022, 33(1): 129–143. [doi: 10.1109/TPDS.2021.3090759]

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

[18] Sakr S, Orakzai FM, Abdelaziz I, Khayyat Z. Large-scale Graph Processing Using Apache Giraph. Cham: Springer, 2016. [doi: 10.1007/978-3-319-47431-1]

[19] Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I. GraphX: Graph processing in a distributed dataflow framework. In: Proc. of the 11th USENIX Conf. on Operating Systems Design and Implementation. Broomfield: USENIX Association, 2014. 599–613.

[20] Yan D, Cheng J, Lu Y, Ng W. Effective techniques for message reduction and load balancing in distributed graph computation. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence: Int'l World Wide Web Conf. Steering Committee, 2015. 1307–1317. [doi: 10.1145/2736277.2741096]

[21] Bouhenni S, Yahiaoui S, Nouali-Taboudjemat N, Kheddouci H. A survey on distributed graph pattern matching in massive graphs. ACM Computing Surveys, 2021, 54(2): 1–35. [doi: 10.1145/3439724]

[22] Yang JY, Yao W, Zhang WJ. Keyword search on large graphs: A survey. Data Science and Engineering, 2021, 6(2): 142–162. [doi: 10.1007/s41019-021-00154-4]

[23] Davoudian A, Chen L, Tu HW, Liu MC. A workload-adaptive streaming partitioner for distributed graph stores. Data Science and Engineering, 2021, 6(2): 163–179. [doi: 10.1007/s41019-021-00156-2]

[24] Zhao X, Li B, Nan HC, Xiao WD. A revised BSP-based massive graph computation model. Chinese Journal of Computers, 2017, 40(1): 223–235 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2017.00223]

[25] Yu G, Gu Y, Bao YB, Wang ZG. Large scale graph data processing on cloud computing environments. Chinese Journal of Computers, 2011, 34(10): 1753–1767 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01753]

[26] Lu L, Hua B. A Platform-and-workload aware online graph partitioning algorithm. Chinese Journal of Computers, 2020, 43(7): 1230–1245 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2020.01230]

[27] Jing NQ, Xue JL, Qu Z, Yang Z, Dai YF. SpecGraph: A distributed graph processing system for dynamic result based on concurrent speculative execution. Journal of Computer Research and Development, 2014, 51(S1): 155–160 (in Chinese with English abstract).

[28] Yan D, Cheng J, Özsu MT, Yang F, Lu Y, Lui JCS, Zhang QZ, Ng W. A general-purpose query-centric framework for querying big graphs. Proc. of the VLDB Endowment, 2016, 9(7): 564–575. [doi: 10.14778/2904483.2904488]

[29] Yan D, Cheng J, Lu Y, Ng W. Blogel: A block-centric framework for distributed computation on real-world graphs. Proc. of the VLDB Endowment, 2014, 7(14): 1981–1992. [doi: 10.14778/2733085.2733103]

[30] Yan D, Guo GM, Chowdhury MMR, Özsu MT, Ku WS, Lui JCS. G-thinker: A distributed framework for mining subgraphs in a big graph. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1369–1380. [doi: 10.1109/ICDE48307.2020.00122]

[31] Wang XB, Qin L, Chang LJ, Zhang Y, Wen D, Lin XM. Graph3S: A simple, speedy and scalable distributed graph processing system. arXiv:2003.00680, 2020.

[32] Zhou X, Weng TF, Yang ZB, Li BR, Zhang J, Li KL. Distributed tip decomposition on large bipartite graphs. Int'l Journal of Software and Informatics, 2022, 12(1): 89–105. [doi: 10.21655/ijsi.1673-7288.00277]

[33] Luo Q, Yu DX, Li F, Cheng XZ, Cai ZP, Yu JG. Distributed core decomposition in probabilistic graphs. Asia-Pacific Journal of

Operational Research, 2021, 38(5): 2140008. [doi: 10.1142/S021759592140008X]

[34] Liu Q, Liao XK, Huang X, Xu JL, Gao YJ. Distributed (α, β)-core decomposition over bipartite graphs. In: Proc. of the 39th IEEE Int'l Conf. on Data Engineering (ICDE). Anaheim: IEEE, 2023. 909–921. [doi: 10.1109/ICDE55515.2023.00075]

[35] Leskovec J, Sosič R. Snap: A general-purpose network analysis and graph-mining library. ACM Trans. on Intelligent Systems and Technology, 2017, 8(1): 1. [doi: 10.1145/2898361]

附中文参考文献:

[24] 赵翔, 李博, 商海川, 肖卫东. 一种改进的基于 BSP 的大图计算模型. 计算机学报, 2017, 40(1): 223–235. [doi: 10.11897/SP.J.1016.2017.00223]

[25] 于戈, 谷峪, 鲍玉斌, 王志刚. 云计算环境下的大规模图数据处理技术. 计算机学报, 2011, 34(10): 1753–1767. [doi: 10.3724/SP.J.1016.2011.01753]

[26] 陆李, 华蓓. 平台和负载特征感知的在线图分割算法. 计算机学报, 2020, 43(7): 1230–1245. [doi: 10.11897/SP.J.1016.2020.01230]

[27] 景年强, 薛继龙, 曲直, 杨智, 代亚非. SpecGraph: 基于并发更新的分布式实时图计算模型. 计算机研究与发展, 2014, 51(S1): 155–160.

翁同峰(1992－), 男, 博士, CCF 学生会员, 主要研究领域为分布式图计算.

李肯立(1971－), 男, 博士, 教授, CCF 会士, 主要研究领域为并行分布式处理, 超级计算与云计算, 面向大数据和人工智能的高效能计算.

周旭(1983－), 女, 博士, 副教授, CCF 高级会员, 主要研究领域为并行计算, 图数据管理, 图计算.

胡逸駪(1988－), 男, 博士, 助理教授, 主要研究领域为并行分布式处理.