计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(7): 1805−1815, 2025

一种带缓冲区的分布式流式图划分算法

史惠康[1] 王泽胜[2] 胡克坤[3] 董 刚[3] 赵有健[1]
[1](清华大学计算机科学与技术系 北京 100084)
[2](中国电子技术标准化研究院 北京 100007)
[3](浪潮电子信息产业股份有限公司 济南 250101)
(shk@pku.org.cn)

A Distributed Buffered Streaming Graph Partitioning Algorithm

Shi Huikang[1], Wang Zesheng[2], Hu Kekun[3], Dong Gang[3], and Zhao Youjian[1]
[1] (*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084)
[2] (*China Electronics Standardization Institute*, *Beijing* 100007)
[3] (*IEIT SYSTEMS Co.*, *Ltd.*, *Jinan* 250101)

Abstract Graph partitioning is a key technique for parallel processing of big graphs. Existing graph partitioning algorithms struggle to balance partition quality and efficiency. Offline partitioning algorithms tend to achieve high partition quality but are time-consuming, while online (or streaming) partitioning algorithms are relatively efficient but suffer from suboptimal partition quality. To address the above problem, we propose a distributed streaming partitioning algorithm with a buffering mechanism in this paper. The algorithm utilizes a multi-loader and multi-partitioner architecture, where multiple loaders read graph data in parallel to improve loading efficiency. Each partitioner maintains a buffer to store graph vertices received from the corresponding loader and sorts them in descending order based on vertex degree, providing better decision-making data for partitioning. Four streaming heuristic rules are pre-configured in the partitioners, which perform parallel partitioning on vertices in the buffer with different goals in mind. A reflow mechanism is employed to fine-tune the partitioning results and improve quality. Experimental results in a distributed system environment show that the proposed algorithm improves partition quality (edge-cut ratio) by more than 18.8% compared with the best existing online partitioning algorithm. Additionally, it reduces the proportion of graph data loading time in the total partitioning time from an average of 30.8% in a single-loader single-partitioner architecture to an average of 20.1%.

Key words big graph; streaming partitioning; distributed; buffering; restreaming

摘 要 图划分是大图并行处理关键技术之一.现有图划分算法存在划分质量和效率难以平衡的问题,主要体现在离线划分算法划分质量高但耗时长;在线(也称流式)划分算法相对高效但划分质量不理想.为此,提出一种带缓冲区的分布式流式划分算法.该算法采用多加载器-多划分器架构,多个加载器并行读取图数据,提高图数据加载效率;每个划分器维护一个缓冲区,缓存相应加载器发来的图顶点,并按顶点度数高低排序,为划分器提供更多决策依据.划分器预置有4条流式启发式规则,围绕不同目标,对缓冲区中的顶点实施并行划分,并借助重流机制微调划分结果,改进划分质量.分布式系统环境下的划分质量与性能实验表明:提出算法的划分质量(割边比)比当前最好的在线划分算法改善超过18.8个百分点,

收稿日期:2023-05-23;**修回日期**:2025-01-24
基金项目:国家自然科学基金重点项目(62394322);北京市自然科学基金项目(L222024)
This work was supported by the Key Program of the National Natural Science Foundation of China (62394322), and Beijing Natural Science Foundation (L222024).
通信作者:赵有健(zhaoyoujian@tsinghua.edu.cn)

并将图数据加载时间在划分总时间的占比,从单划分器-单加载器架构流式划分算法的平均 30.8% 缩减至平均 20.1%.

关键词 大图;流式划分;分布式;缓冲;重流

中图法分类号 TP338

DOI: 10.7544/issn1000-1239.202330386 **CSTR:** 32373.14.issn1000-1239.202330386

伴随着数字经济的蓬勃发展,移动互联网、智能终端越发普及,社交网络、智慧交通和生命科学等领域涌现出海量数据[1]. 这些数据规模大,数据元素间关联关系复杂,业界称为图大数据,简称大图. 以全球最大的社交平台 Facebook 为例,其社交网络拥有超过 20 亿用户和上千亿条好友关系. 传统的单机由于性能有限,难以从大图中快速挖掘蕴藏的价值,需要拥有强大计算与存储能力的并行计算系统作为支撑. 为充分利用并行计算系统开展大图并行处理,首先要对大图实施划分.

图划分,是指按照某种策略将大图划分为若干个满足一定约束条件的子图. 子图规模均衡性和子图割边数量,是影响大图分布式并行处理性能和效率的重要因素[2],主要体现在:规模越均衡、割边数越少,映射到各计算节点的负载越均衡、节点间通信代价越低、处理效率越高. 如何获取均衡性好、割边数少的图划分结果,已成为业界关注的重点[3-4]. 国内外学者提出许多图划分算法,主要有离线划分、分布式划分和流式划分. 离线划分算法在实施划分前,将大图全部加载到内存,典型代表有谱划分算法[5]和 Metis 划分算法[6]. 这类图划分算法依据图全局信息实现高质量划分,但时间复杂度高且受内存大小限制,常用于划分小规模图数据. 伴随着图数据规模的持续增大,分布式划分和流式划分逐渐成为研究热点[7-8]. 其中,分布式划分算法充分利用并行计算系统中多个处理节点的算力和存储资源,依据图全局信息实施划分,如 ParMetis 划分算法[9]. 流式划分算法采用边加载边划分的策略,不依赖图全局信息,具有时间复杂度低的特点,典型代表有 LDG[10], FENNEL[11], HeiStream[12]. 该类图划分算法通常采用单加载器-单划分器结构,存在性能瓶颈,且由于缺乏图全局信息,存在冷启动问题,即在划分初始阶段,划分器的部分顶点因缺乏划分决策信息,难以获取最优子图位置编号,导致划分质量不理想. 如图 1 所示,采用 LDG 启发式规则将大图划分为 2 个子图,假设按照顺序加载,且每个子图中最大顶点数为 4,将得到划分结果 1(有 4 条割边),但实际最优结果应为划分结果 2(有 2 条割边). 之所以出现偏差,主要因为该启发式规则并未充分利用高度数顶点等图数据流中的有价值信息.

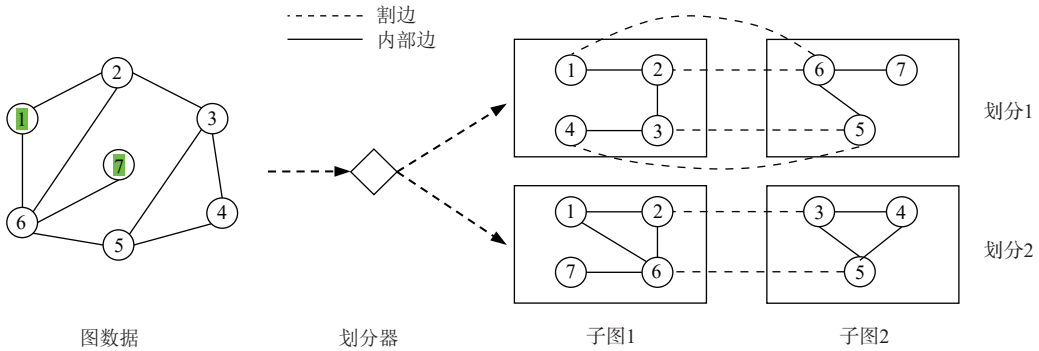


Fig. 1 Cold start problem of graph partitioning

图 1 图划分冷启动问题

针对上述问题,本文提出一种带缓冲区的分布式流式划分算法. 该算法采用多加载器-多划分器架构,各加载器以顶点为单位,按照特定顺序并行加载图数据;各划分器维护一个缓存,对相应加载器输出的图顶点流数据先缓存,然后按顶点度数重新排序,并依据不同划分目标实施划分. 在充分利用缓冲区辅助信息提高划分质量的同时,借助分布式流式机制提高划分算法性能. 为进一步提高划分质量,我们借助重流机制对划分结果进行微调优化. 本文的创新点及贡献有 4 个方面:

1)设计一种多加载器-多划分器的分布式图划分架构. 通过多个加载器和多个划分器分别并行读取

和划分图数据，提高图数据的加载效率和划分效率，缓解传统流式划分算法存在的性能瓶颈.

2）提出一种分布式图数据缓冲机制. 每个划分器维护一个缓冲区，采用快速排序算法对缓冲区内图数据按照顶点度数由高到低排序，优先将高度数顶点作为“种子”实施划分，有效缓解冷启动问题.

3）设计 4 条流式启发式规则. 分别以最优均衡性、简单高效原则、最大化模块度兼顾均衡性、最小化割边比兼顾均衡性为目标，将大图划分为一系列小规模子图，可以满足不同应用场景下的划分需求.

4）在真实的分布式系统中开展一系列实验，全面检验所提算法划分真实大图与合成大图的划分质量和划分性能.

1 相关工作

图划分算法主要包括 3 类：离线划分、流式划分和分布式划分. 谱划分[5]是典型的离线划分算法，它首先计算相似矩阵的拉普拉斯矩阵，然后求出其第 2 大特征值及其对应的特征向量，对图进行递归二分. 该算法对很多类型的图数据都能生成高质量划分结果. Metis[6]采用多级 k 路划分策略，将图划分为 3 个阶段：1）第 1 阶段通常借助收缩匹配或聚簇等策略来构建输入图的连续粗略近似，组成层次化结构；2）一旦图只剩下少数几个顶点，第 2 阶段借助常用的离线图划分算法直接将其划分为 k 个子图；3）第 3 阶段将该划分结果连续投影到层次结构中的更精细级别，并使用局部搜索算法对划分结果进行细化. 与谱划分类似，Metis[6]计算开销较大，不便用来划分大规模图数据. 例如，Tsourakakis 等人[13]采用 Metis 对拥有超过 14 亿条边的 Twitter 图进行划分，其划分总时长超过 8.5 h.

在流式划分方面，Stanton 等人[10]将流模型引入到图划分，并提出包含 LDG 在内的 10 条流式启发式规则，在数据由磁盘加载到内存的一次性过程中，仅利用图局部信息即可实施划分. Tsourakakis 等人[13]在文献 [10] 的基础上，定义一个融合 LDG 和 Folklore[14]两种启发式规则的流式划分框架 FENNEL，将顶点分配到具有更多邻居的子图. Nishimura 等人[11]提出重流思想，并设计了 LDG 和 FENNEL 的重流版本，它们保持图划分目标函数不变，并以上一轮流式划分的结果作为本轮的初始划分，对划分结果不断微调优化，主要适用于相同或相近的大图持续出现的场景. Awadelkarim 等人[15]研究顶点顺序对流式划分质量的影响，根据特定优先级对流式加载的顶点进行静态或动态排序，并提出 LDG 的优先级版本 reLDG. Patwary 等人[16]提出基于贪心策略的流式划分算法，设计含有数百个顶点的滑动窗口，为候选顶点提供更多辅助信息. 类似地，Faraj 等人[17]提出带缓冲区的流式划分算法，首先构建代表批处理顶点和已划分顶点的结构模型图，再利用传统多级图划分算法完成划分. Zeng 等人[18]提出非均衡流式划分算法，可按照任意比例关系流式地将大图划分为一系列子图. Sun 等人[19]将博弈理论与流式划分相结合，采用多人博弈纳什均衡，实现各个时间窗口内流数据的最优划分. 这些流式划分算法均采用单加载器-单划分器架构，容易出现性能瓶颈. 此外，Özsu 等人[20]、Abbas 等人[21]关注流式划分算法的性能评测问题.

在分布式划分方面，许多学者基于多级图划分范式设计了划分算法，如基于共享内存的 KaMinPar[22]，Mt-KaHIP[23]和基于分布式内存的 ParMetis[9]，ParHIP[24]等. 其中，KaMinPar[22]进一步将多级算法延伸到初始划分阶段，结合递归二分和直接 k 路划分计算初始划分方案. Mt-KaHIP[23]将多级图划分算法与 KaHIP 图划分工具相结合，旨在提供适用于大规模图数据的高质量图划分解决方案. ParMetis[9]是 Metis[6]的并行扩展版本，专门用于处理大规模图数据的划分和排序问题，旨在优化分布式内存体系结构上的性能. ParHIP[24]是一种用于超图划分的并行算法，同时支持共享内存和分布式内存计算环境. 总的来说，基于共享内存的分布式划分算法[22-23]可实现较高的划分质量和合理的可扩展性，但基于分布式内存的分布式划分算法[9]划分质量依然不高，且可扩展性低、难以保证划分均衡性. 此外，Battaglino 等人[25]则积极尝试并行化流式划分算法，在 FENNEL 算法的基础上提出了分布式流式划分算法 GraSP，这是当前性能最好的分布式划分算法.

上述划分算法[10,13,15-17,19,24-25]均属于割边划分，还有部分算法专注于割点划分[26-29]，这类算法用连接边流替代顶点流作为输入，目标通常是最小化割点数. 尽管从应用角度来看，它们与本文研究工作密切相关，但该类问题并不是本文工作的重点. 更多有关图划分工作请参考文献 [30].

2 图划分概念

2.1 划分问题形式化描述

图划分按照某种方式将大图 G 切割成多个小的

子图$G_i(i\in\{1,2,\cdots,k\})$, 即$G=G_1\cup G_2\cup\cdots\cup G_k$. 设大图$G=(V,E)$, 其中$V=\{v_1,v_2,\cdots,v_n\}$, $E=\{e_{ij}|v_i,v_j\in V\}$分别代表其顶点集和连接边集, n为顶点数. $\forall v_j\in V$, 将与其关联的连接边条数称为度数, 记为$deg(v_j)$; 与v_j有连接边相连的顶点集合称为邻居, 记为$N(v_j)=\{v_k|v_k\in V\wedge e_{jk}\in E\}$, 则有$deg(v_j)=|N(v_j)=\{v_k\in V\wedge e_{jk}\in E\}|$. $\forall i\in\{1,2,\cdots,k\}$, $G_i=(V_i,E_i)$是G的一个子图, 并满足$V_i\subseteq V$和$E_i\subseteq E$. 这些子图集合$\pi_k=\{G_1,G_2,\cdots,G_k\}$为$G$的一个$k$路划分, 当且仅当$\forall_{i,j}\in\{1,2,\cdots,k\}$, $i\neq j$, 均有$V_i\cap V_j=\varnothing$, $V=\bigcup_{i=1}^{k}V_i$, $E=\bigcup_{i=1}^{k}E_i$. 横跨2个子图G_i和G_j的连接边称为割边, 上述2个子图的割边集记为$E(G_i,G_j)$.

图划分问题就是: 给定大图G, 寻找它的一个k路划分$\pi_k^*=\{G_1^*,G_2^*,\cdots,G_k^*\}$, 使得对于$k$路划分空间$\Omega$中的任一划分$\pi_k$, 在保证各子图顶点数均衡的前提下, 割边数最少. 该问题可形式化描述为式(1)和式(2).

$$f(\pi_k)=\arg\min\sum_{G_i,G_j\in\pi_k}|E(G_i,G_j)|, \tag{1}$$

$$\text{s.t.}\left|1-\frac{V_i}{V}\right|\leqslant\varepsilon,\ \forall i\in\{1,2,\cdots,k\}. \tag{2}$$

本文用到的主要符号及其含义如表1所示.

Table 1 Symble and Meaning

表1 符号及含义

符号	含义
G, G_i, G_i^τ	图、子图、时刻τ的子图
V和E	顶点集和边集
v和e	某个顶点和某条边
$N(v)$	顶点v的邻居集
π_k和π_k^τ	k路划分和时刻τ的k路划分
$E(G_i,G_j)$	割边集
m_i和b_i	计算节点及其缓冲区

2.2 划分评价标准

不同划分算法的划分质量参差不齐. 这里采用业界常用的划分质量评价指标来构建评价体系, 主要包括:

1)均衡性$\rho=\max_{i\in\{1,2,\cdots,k\}}(|V_i|/(|V|\cdot k))$, 用来度量子图顶点数以及理想情况下每个子图应拥有的顶点数的偏离程度. 偏离理想值1.0越大, 意味着并行处理阶段各节点间负载越不均衡, 处理时间越长.

2)割边比$\lambda=\frac{1}{|E|}\left|\bigcup_{i=1,j=1}^{k}E(G_i,G_j)\right|$, 用来度量划分$\pi_k$的割边数占总边数的比例, λ值越大, 并行处理阶段各处理节点间的通信开销越大, 处理时间越长.

3)划分总时间$\tau=\max_{i\in\{1,2,\cdots,k\}}(\tau_{i,\text{ld}}+\tau_{i,\text{part}})$. 其中, $\tau_{i,\text{ld}}$和$\tau_{i,\text{part}}$分别表示第$i\in\{1,2,\cdots,k\}$个划分进程的数据加载时间和划分时间. 对于采用单加载器-单划分器的流式划分算法以及离线图划分算法而言, 其划分总时间$\tau=\tau_{\text{ld}}+\tau_{\text{part}}$, τ_{ld}和τ_{part}分别表示数据加载时间和划分时间.

4)效率$\varepsilon=\frac{\tau_{\text{seq}}}{proc\times\tau_{\text{par}}}$. 其中, τ_{seq}和τ_{par}分别表示划分进程数$proc=1$和$proc>1$时的划分总时间. ε用来表征图数据规模不变的情况下, 划分总时间随划分进程数增加的变化情况. ε值越接近于1.0, 表示效率越高, 即并行计算资源有效利用率越高.

3 带缓冲区的分布式流式划分算法

3.1 算法整体流程

流式划分算法中普遍存在冷启动问题. 为缓解该问题, 本节提出带缓冲区的分布式流式划分算法. 具体工作流程如图2所示.

本文算法采用多加载器-多划分器架构, 运行于分布式系统的不同处理节点上. 不妨设分布式系统由k个同构计算节点组成, 节点间通过高速通信链路连接, 记所有节点集合为$M=\{m_1,m_2,\cdots,m_k\}$. 其中, 每个计算节点上运行1对加载器-划分器, 加载器负责从分布式文件系统中读取大图$G=(V,E)$, 并传递给随后的划分器, 因而共有k对加载器-划分器. 每个划分器维护一个大小为w的缓冲区, 用以存放图数据流. 每个划分器都保存一份划分决策信息π_k^τ, 即时刻τ大图G的k路划分.

k个加载器按照随机原则, 以顶点为单位从文件系统中成批并行读取w个图顶点, 并发送给相应划分器; 划分器首先缓存收到的图数据流, 待缓冲区满后, 利用快速排序算法对缓冲区内的图顶点按照度数由高到底排序; 随后根据所提4条启发式规则BH(buffered hashing), BB(buffered balance), BWM(buffered weighted modularity), Hybrid(3.2节中介绍)中的某一条, 计算缓冲区内各个顶点应放置到右侧分布式系统的哪个处理节点上并实施放置; 重复上述过程$ceil(n/w)$批次, 直至数据加载完毕时, G被划分为k个子图. 其中, $ceil(*)$表示向上取整函数, n为G的顶点数. 最后, 将已有划分方案作为已知信息, 采用重流方式, 对划分方案微调以提高划分质量. 具体算法如算法1所示.

算法1. 分布式流式图划分.

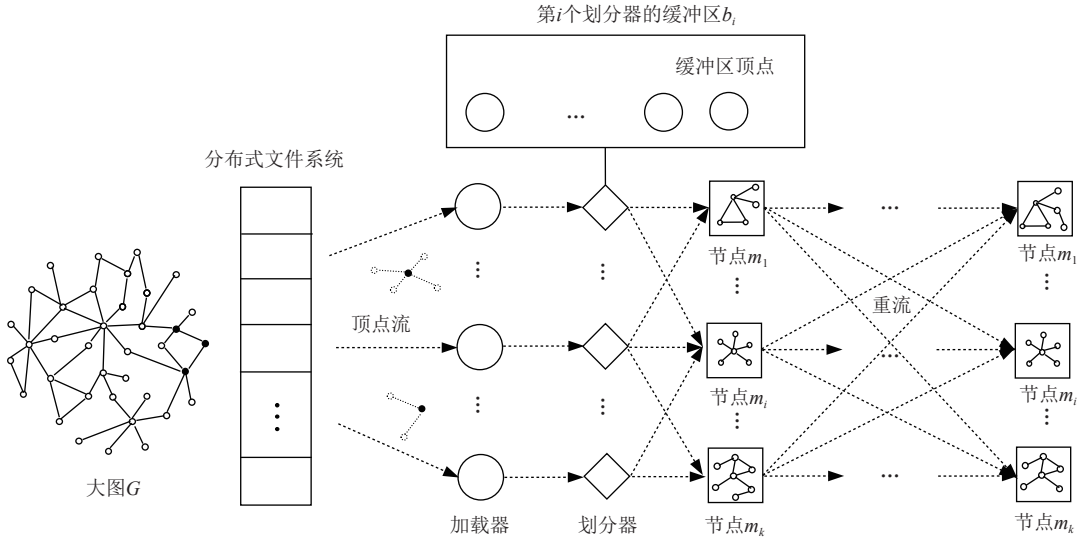


Fig. 2 Algorithm flow of distributed buffered streaming graph partitioning

图 2 带缓冲区的分布式流式图划分算法流程

输入：G, k, b_i, w, *num_streams*;

输出：π_k^τ.

① *num_batches*←*ceil*(n/w);

② $b_i \leftarrow \varnothing$;

③ $\tau \leftarrow 0$;

④ *initialize*(*num_streams*);

⑤ *initialize*(π_k^τ, $\varnothing$);

⑥ for each s in {1, 2, ⋯, *num_streams*}

⑦ for each batch l in {1, 2, ⋯, *num_batches*}

⑧ for each partitioning process p_i in {1, 2, ⋯, k} do in parallel /*各划分器p_i并行执行*/

⑨ $b_i \leftarrow$q*ucikSort_vertexDegree*(b_i); /*用快排算法对缓冲区中的顶点按照度数非递增排序*/

⑩ for each vertex v in buffer b_i

⑪ $j \leftarrow$*streamingHeuristic*(v, π_k^τ); /*根据启发式规则计算顶点应放置的子图编号*/

⑫ $G_j^{\tau+1} \leftarrow v$; /*放入对应子图*/

⑬ end for

⑭ end for

⑮ $\pi_k^{\tau+w} \leftarrow$*MPI_partitionUpdate*($\pi_k^\tau$); /*各划分器同步各个子图的最新状态*/

⑯ $\tau \leftarrow \tau + w$;

⑰ end for

⑱ end for

不妨设$\pi_k^\tau = \{G_1^\tau, G_2^\tau, \cdots, G_k^\tau\}$为时刻$\tau$大图$G$的$k$路划分. 其中$G_i^\tau (i \in \{1,2,\cdots,k\})$表示时刻$\tau$子图$G_i$的状态. 每个划分器$p_i (i \in \{1,2,\cdots,k\})$所维护的缓冲区$b_i (i \in \{1,2,\cdots,k\})$中存放已按度数由高到低有序排列的顶点，记为$\{v_{i1}^\tau, v_{i2}^\tau, \cdots, v_{iw}^\tau\}$. 时刻$\tau+1$, 划分器$p_i (i \in \{1,2,\cdots,k\})$基于当前的划分决策信息$\pi_k^\tau$, 按照某种启发式规则, 决定缓冲区$b_i$中某个顶点$v$应“流”向哪个子图. 待缓冲区内所有顶点分配完毕, 各划分器同步更新π_k^τ. 若出现多个划分器同时更新某一子图G_k^τ的状态, 则按照划分器编号由小到大的顺序按序更新. 重复上述过程直至所有顶点加载完毕, 即可得到G的k路划分π_k. 将π_k作为已知信息, 对图数据$G=(V,E)$实施重流, 利用支持重流的启发式规则进一步优化π_k.

3.2 分布式流式启发式规则

本节设计了 4 条分布式流式启发式规则, 每个规则分别对应一个算法. 各个划分器并行执行, 用以确定各自缓冲区中当前顶点v应放置的子图编号*ind*.

1）BH. 按照简单高效原则实施划分, 尽可能提高划分效率. 设计哈希函数$H: B_i \to \{1,2,\cdots,k\}$, 将缓冲区中最新的顶点$v_g$分配给子图, 子图编号为*ind*:

$$ind = H(v_g) + 1, \tag{3}$$

本文取$H(v_g) = v_g \bmod k$.

2）BB. 按照最优均衡性目标实施划分. 在时刻τ, 将缓冲区中最新的顶点v_g“流”向当前已分配顶点数最少的子图G_i^τ. 若存在多个具有最少顶点数的子图, 则随机指派v_g到其中任一子图, 子图编号为:

$$ind = \underset{i\in\{1,2,\cdots,k\}}{\arg\min}\left(\left|V_i^\tau\right|\right). \tag{4}$$

BH 和 BB 启发式规则不便利用上一次流式划分的结果来指导当前流中顶点的划分, 因而重流效果

不明显.

3)BWM. 在兼顾最优均衡性原则的同时, 按照最大化子图模块度目标实施划分. 在时刻t, 将缓冲区中最新的顶点v_g“流”向某个子图, 使得: ①v_g的邻居顶点集与子图G_i^τ顶点集交集阶数最大; ②添加惩罚函数项以惩罚顶点数多的子图, 从而避免失衡. 若2个或多个子图的模块度相等, 则随机指派v_g到其中任一子图:

$$ind=\arg\max\left(\left|N\left(v_g\right)\cap V_i^\tau\right|\times\phi\left(V_i^\tau\right)\right), \tag{5}$$

其中$\phi\left(V_i^\tau\right)=1-\alpha\left|V_i^\tau\right|^\gamma$. 通过合理地选择$\alpha$和$\gamma$, 来控制$\phi\left(V_i^\tau\right)$在整个函数中所占的比重. 在重流时, 将上一次流式划分的结果(记为$\pi_k=\{G_1,G_2,\cdots,G_k\}$)作为已知信息, 按照式(6)计算缓冲区中最新的顶点v_g应“流”向的子图编号.

$$ind=\underset{i\in\{1,2,\cdots,k\}}{\arg\max}\left(\left|N\left(v_g\right)\cap V_i\right|\times\phi\left(V_i^\tau\right)\right), \tag{6}$$

其中V_i表示上一次流式划分结果中第$i(i\in\{1,2,\cdots,k\})$个子图G_i的顶点集, V_i^τ表示子图G_i在本次流式划分中时刻τ的状态.

4)Hybrid. 在兼顾最优均衡性原则的同时, 按照最小化割边比目标实施划分. 给定一种区分顶点度数高低的算法, 如设定一个合理的阈值η, 在时刻τ, 对度数超过该阈值的顶点采用BH决定其“流”向哪个子图, 否则采用BWM. 该规则可形式化描述为:

$$\text{while}\left(stream\left(v_g\right)\neq\varnothing\right)\begin{cases}\text{BH, if } deg\left(v_g\right)\leqslant\eta,\\ \text{BWM, if } deg\left(v_g\right)>\eta.\end{cases} \tag{7}$$

4 实验结果与分析

4.1 实验环境

1)实验数据. 我们从SNAP[31]中搜集了wiki-Talk, LiveJournal, Friendster这3个真实数据集, 详见表2.

Table 2 Information of Real-World Big Graph

表2 真实大图信息

数据集	顶点数	边数	顶点平均度数	数据集类型
wiki-Talk	2 394 385	5 021 410	4.2	维基百科网络
LiveJournal	4 847 571	68 993 773	28.5	社交网络
Friendster	65 608 366	1 806 067 135	55.1	社交网络

此外, 我们还利用图合成工具PaRMAT[32]合成了一系列不同规模且顶点度数分布均服从幂率分布的图数据集, 详见表3.

Table 3 Information of Composite Big Graph

表3 合成大图信息

尺度参数	顶点数	边数	顶点平均度数
26	67×10^6	1.07×10^9	32
27	134×10^6	2.14×10^9	32
28	268×10^6	4.29×10^9	32
29	537×10^6	8.58×10^9	32
30	$1\,070\times10^6$	17.1×10^9	32

2)对比基准. 选择当前离线划分质量最好的KaMinPar[22]和在线划分质量最好的HeiStream[12], GraSP[25]作为对比基准. 其中, KaMinPar[22]是一种基于共享内存的深层多级图划分算法, 除粗化阶段外, 在初始划分阶段继续采用多级算法并利用递归二分和直接k路划分计算初始划分方案. HeiStream[12]是一种单加载器-单划分器的缓冲流式划分算法, 它按批接收顶点并缓存在缓冲区中, 联合批次中的节点和已存在的划分结构构建待划分的图, 再利用多级图划分算法实施划分. GraSP[25]是一个采用MPI实现的分布式流式划分算法, 其划分规则是FENNEL. 通过使用调整的划分参数对分布式图进行重新流式处理, 以实现对大图的快速k路划分.

分布式系统运行在一个由2台DELL Precision 7920塔式工作站通过网线直接连接而成的工作站机群. 其中, 每台工作站都配有2颗英特尔至强金牌6254@3.1GHz CPU、512GB主存以及1TB SSD、8TB HDD. 操作系统版本是Centos 7.9.2009, 编译器版本是GCC 9.3, MPI版本是MPICH-3.4.3.

4.2 结果分析

本节设计3组实验, 分别讨论所提算法的划分质量与性能. 第1组实验考查划分真实图和合成图的划分质量, 评价指标包括均衡性ρ、割边比λ; 第2组实验考查划分合成图的性能, 评价指标包括划分总时间τ和效率ε; 第3组实验考查缓冲区大小对划分质量的影响, 评价指标包括均衡性ρ、割边比λ和划分总时间τ. 在每组实验中, 取BWM的参数α和γ分别为$\lceil n/k\rceil$和1, 其中n表示图顶点数, k表示划分路数; 取Hybrid的参数η为所划分数据集的平均顶点度数; 取重流次数为5; 每种算法均取5次实验的平均值作为最终结果.

1)划分质量实验. 本实验主要目的是考查所提算法中4条流式启发式规则BH, BB, BWM, Hybrid对2个真实图和6个合成图的划分质量. 不妨取划分进程数$proc=8$、划分路数$k=8$、缓冲区大小$\omega=1\,024$, 均衡性和割边比实验结果分别见图3、图4和表4.

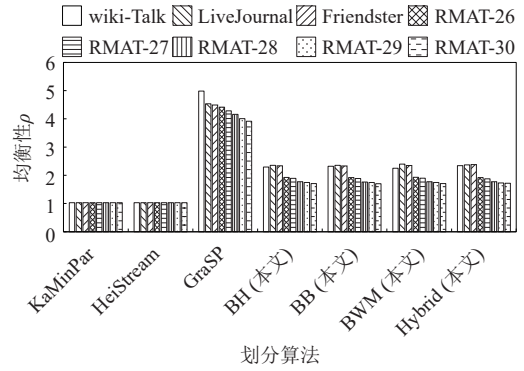


Fig. 3 Partition equalization comparison of different partitioning algorithms

图 3 不同划分算法划分均衡性对比

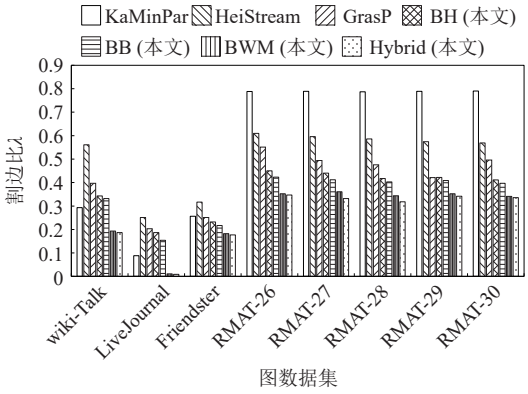


Fig. 4 Partitioning cut edge ratio comparison of different partitioning algorithms

图 4 不同划分算法划分割边比对比

不难看出，无论真实图还是合成图，所提算法 4 条启发式规则 BH, BB, BWM, Hybrid 的划分质量均明显优于当前最好的在线划分算法 GraSP 和 HeiStream，但在均衡性指标上逊色于当前最好的离线划分算法 KaMinPar.

具体来讲，以 Hybrid 启发式规则为例，割边比指标λ较 KaMinPar 缩减至少 36.2 个百分点（wiki-Talk 数据集），较 HeiStream 和 GraSP 分别缩减至少 40.4 个百分点和 18.8 个百分点（RMAT-29 数据集）；以 BH 为例，其均衡性指标ρ较 KaMinPar 和 HeiStream 均增加 128.4 个百分点（LiveJournal 数据集），而 GraSP 的均衡性相较于 KaMinPar 和 HeiStream 均增加 358.3 个百分点. 但也应看到 KaMinPar 和 HeiStream 为保证均衡性，付出了高昂的时间代价. 此外，BH 和 BB 在均衡性指标ρ优于 BWM 和 Hybrid，但在割边比指标λ方面却劣于 BWM 和 Hybrid. 这是因为前两者侧重于均衡性，而后两者追求最小化割边比.

由此可见，与不使用缓冲区的 GraSP 算法相比，本文提出缓冲区机制能够明显提高划分质量，有效缓解冷启动问题. 而使用缓冲区的 HeiStream 算法表现之所以较本文算法存在一定差距，是因为它将每个已划分子图“池化”一个超级顶点，与缓冲区中各顶点构造一张图，再利用传统多级图划分算法对其实施划分. 该“池化”操作造成划分决策信息损失较大，降低了划分质量.

2）划分性能实验. 本实验主要目的是测试所提 4 条流式启发式规则 BH, BB, BWM, Hybrid 的划分性能. 不妨固定图数据为 RMAT-22、缓冲区大小$\omega = 1\,024$、划分路数$k = 128$，调整划分进程数$proc \in \{1,2,4,8,16,32,64,128\}$，记录不同划分算法的总划分时间. 实验结果见图 5、图 6、图 7、表 5 和表 6 所示.

从图 5、图 6 和表 5 可以看出，当$1 < proc < 128$时，所提算法 4 条启发式规则 BH, BB, BWM, Hybrid 在划分总时间指标τ上显著优于当前最好的离线划分算法 KaMinPar 和当前最好的在线划分算法 GraSP.

具体来讲，不妨以表现最好的 BH 为例，其较 KaMinPar 缩减至少 80.4 个百分点（对应$proc = 2$）；较

Table 4 Partition Quality Comparison of Different Partitioning Algorithms

表 4 不同划分算法的划分质量对比

数据集	KaMinPar		HeiStream		GraSP		BH（本文）		BB（本文）		BWM（本文）		Hybrid（本文）	
	ρ	λ	ρ	λ	ρ	λ	ρ	λ	ρ	λ	ρ	λ	ρ	λ
wiki-Talk	1.030	0.293	1.030	0.561	4.986	0.397	2.293	0.343	2.320	0.332	2.247	0.193	2.342	0.187
LiveJournal	1.030	0.088	1.030	0.251	4.534	0.203	2.353	0.187	2.353	0.154	2.398	0.011	2.369	0.009
Friendster	1.030	0.256	1.030	0.317	4.489	0.251	2.336	0.232	2.331	0.217	2.347	0.182	2.381	0.177
RMAT-26	1.030	0.788	1.030	0.610	4.412	0.551	1.923	0.450	1.919	0.423	1.930	0.352	1.919	0.347
RMAT-27	1.030	0.789	1.030	0.596	4.284	0.494	1.890	0.440	1.883	0.412	1.898	0.361	1.876	0.332
RMAT-28	1.030	0.787	1.030	0.586	4.158	0.476	1.769	0.417	1.763	0.403	1.770	0.344	1.769	0.318
RMAT-29	1.030	0.789	1.030	0.574	4.007	0.421	1.745	0.422	1.742	0.409	1.742	0.352	1.727	0.342
RMAT-30	1.030	0.790	1.030	0.569	3.913	0.496	1.706	0.411	1.701	0.397	1.709	0.341	1.715	0.336

注：KaMinPar 和 HeiStream 均取非均衡因子$\gamma = 0.03$.

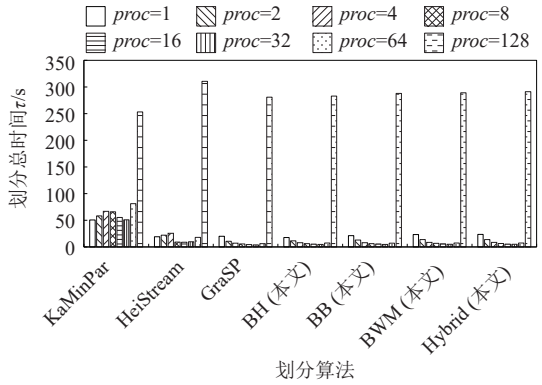


Fig. 5 Total time comparison for partitioning of different partitioning algorithms

图 5 不同划分算法划分总时间对比

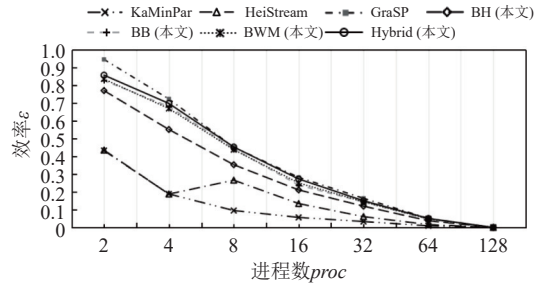


Fig. 6 Efficiency comparison of different partitioning algorithms

图 6 不同划分算法效率对比

HeiStream 和 GraSP 分别缩减至少 30.7 个百分点和 12.9 个百分点(对应 $proc=8$). 当 $proc=1$时, GraSP 和所提 4 条流式启发式规则均蜕化成单加载器-单划分器架构划分算法, 它们的划分总时间与 HeiStream 的差异不明显. 当 $proc=128$时, 此时进程数已超过 CPU 物理核心数(80), 引发严重的计算资源争用, 导致所有划分算法的划分总时间都显著增加. 在效率指标ε上, 无论 $proc$如何变化, 所提 4 条启发式规则

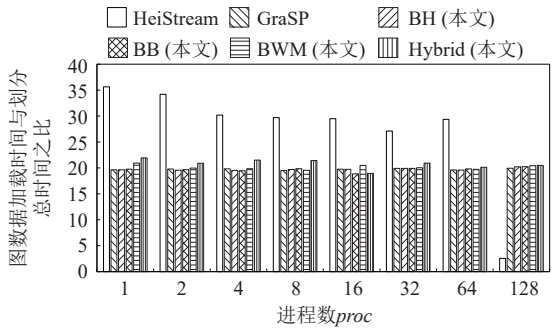


Fig. 7 Proportion of data loading time in total partitioning time of different partitioning algorithms

图 7 不同划分算法的数据加载时间占总划分总时间比例

与 GraSP 表现接近, 且均显著优于 KaMinPar 和 HeiStream($proc=2$除外). 当 $proc=2$ 时效率最高(并行计算资源得到充分利用), 而当 $proc$ 进一步增加时, 效率迅速降低, 主要是因为并行划分带来的收益不足以弥补并行开销。

从图 7 和表 6 可以看出, 当划分进程数 $proc<128$时, 所提划分算法的平均划分时间比采用单加载器-单划分器架构的 HeiStream 算法的划分时间缩减 29.8%.

具体来看, HeiStream 流式划分算法的数据加载时间占划分总时间的比例平均为 30.8%, 最高达到 35.6%. 而本文所提算法采用多加载器-多划分器架构, 其数据加载时间占划分总时间的比例平均为 20.1%, 最高仅为 21.9%. 这表明多加载器-多划分器架构能够有效缓解传统流式划分算法存在的性能瓶颈.

3)缓冲区大小影响力实验. 本实验主要目的是测试缓冲区大小对所提算法划分质量和性能的影响. 不妨以 LiveJournal 图数据为例, 固定划分进程数 $proc=8$, 调整划分窗口$\omega\in\{128, 256, 512, 1024, 2048\}$, 记录不同划分算法的划分质量和划分总时间, 实验

Table 5 Partition Performance of Different Partitioning Algorithms

表 5 不同划分算法划分性能对比

k	KaMinPar		HeiStream		GraSP		BH（本文）		BB（本文）		BWM（本文）		Hybrid（本文）	
	τ	ε	τ	ε	τ	ε	τ	ε	τ	ε	τ	ε	τ	ε
1	50.431		19.175		19.941		17.621		21.104		23.054		23.453	
2	58.156	0.338	21.975	0.934	10.525	0.663	11.420	0.641	12.737	0.644	13.776	0.669	13.658	0.669
4	66.688	0.167	25.418	0.250	6.884	0.669	7.974	0.658	7.741	0.653	8.597	0.681	8.386	0.681
8	65.703	0.090	8.981	0.115	5.510	0.369	6.223	0.362	5.958	0.352	6.569	0.362	6.460	0.362
16	55.226	0.047	8.872	0.057	4.403	0.209	5.196	0.202	5.518	0.204	5.732	0.211	5.356	0.211
32	50.826	0.028	9.735	0.029	3.779	0.106	4.571	0.105	4.571	0.103	4.968	0.104	4.855	0.104
64	80.996	0.012	17.813	0.014	6.051	0.041	7.152	0.039	7.066	0.039	7.353	0.042	7.159	0.042
128	253.41	0.001	310.58	0.005	281.08	0.005	283.05	0.005	288.04	0.005	289.23	0.005	291.11	0.005

Table 6 Proportion of Data Loading Time in Total Partitioning Time of Different Partitioning Algorithms

表 6 不同划分算法的数据加载时间占总划分总时间比例

k	HeiStream		GraSP		BH（本文）		BB（本文）		BWM（本文）		Hybrid（本文）	
	τ_{ld}	τ_{ld}/τ	τ_{ld}	τ_{ld}/τ	τ_{ld}	τ_{ld}/τ	τ_{ld}	τ_{ld}/τ	τ_{ld}	τ_{ld}/τ	τ_{ld}	τ_{ld}/τ
1	6.833	0.356	3.921	0.196	3.465	0.196	4.171	0.197	4.831	0.209	4.628	0.219
2	7.517	0.342	2.084	0.198	2.233	0.195	2.506	0.196	2.753	0.199	2.664	0.209
4	7.671	0.301	1.365	0.198	1.557	0.195	1.508	0.194	1.708	0.198	1.665	0.215
8	2.668	0.297	1.074	0.194	1.227	0.197	1.184	0.198	1.284	0.195	1.277	0.214
16	2.618	0.295	0.870	0.197	1.025	0.197	1.041	0.188	1.175	0.204	1.046	0.189
32	2.640	0.271	0.753	0.199	0.910	0.199	0.910	0.199	0.996	0.200	0.957	0.209
64	5.232	0.293	1.188	0.196	1.401	0.195	1.401	0.198	1.451	0.197	1.423	0.201
128	7.972	0.025	56.106	0.199	57.252	0.202	58.379	0.202	59.113	0.204	59.005	0.204

结果见表 7.

由表 7 可以看出，所提算法 4 条启发式规则 BH, BB, BWM, Hybrid 在不同缓冲区下划分质量变化并不明显，但划分总时间τ随着缓冲区ω的增加先降低后提高，当$\omega = 1\ 024$时，划分总时间最短. 这是因为当滑动窗口过小时，需要频繁从磁盘中读取图数据流，I/O 开销较大. 当ω进一步增加时，划分总时间逐渐增加，可能的原因是缓冲区顶点排序操作时间增大.

总体来说，如表 8 所示，在 4 条启发式规则中，BH 和 BB 更侧重于划分的均衡性，适用于实时性要求高、负载均衡敏感、通信开销不敏感的图计算场景；而 BWM 和 Hybrid 兼顾了划分的均衡性和割边比，适用于实时性要求低、负载均衡和通信开销敏感的图计算场景.

Table 7 Impact of Different Buffer Zone Sizes on Partitioning Quality and Performance

表 7 不同缓冲区大小对划分质量和性能的影响

ω	BH（本文）			BB（本文）			BWM（本文）			Hybrid（本文）		
	ρ	λ	τ	ρ	λ	τ	ρ	λ	τ	ρ	λ	τ
128	2.369	0.182	3.039	2.375	0.151	2.818	2.421	0.010	2.901	2.358	0.009	2.896
256	2.347	0.187	2.974	2.353	0.156	2.810	2.398	0.011	2.891	2.347	0.009	2.845
512	2.347	0.187	2.813	2.347	0.157	2.796	2.392	0.011	2.844	2.336	0.010	2.841
1 024	2.347	0.187	2.793	2.352	0.154	2.792	2.398	0.011	2.805	2.370	0.009	2.802
2 048	2.347	0.187	2.840	2.347	0.157	2.780	2.387	0.012	2.803	2.336	0.011	2.798

Table 8 Advantages, Disadvantages and Application Scenarios of Different Heuristics

表 8 不同启发式规则的优缺点以及适用场景

特性	BH（本文）	BB（本文）	BWM（本文）	Hybrid（本文）
优点	简单高效	划分均衡性好	划分模块度高、均衡性好	拓扑结构感知、划分割边比低、均衡性好
缺点	划分割边比高	划分割边比高	计算复杂	计算复杂
适用场景	实时性要求高、负载均衡敏感、通信开销不敏感的图计算场景	实时性要求高、负载均衡敏感、通信开销不敏感的图计算场景	实时性要求低、负载均衡和通信开销敏感的图计算场景	实时性要求低、负载均衡和通信开销敏感的图计算场景

5 结　论

本文针对现有图划分算法在划分大图时存在的划分质量和效率难以平衡的问题，提出一种带缓冲区的分布式流式划分算法，通过结合带缓冲区的流式划分算法与分布式划分算法两者的优势，在利用缓冲区辅助信息提高划分质量的同时，借助多加载器-多划分器架构的分布式流式机制提高划分算法性能. 此外，为满足不同划分场景需求，我们设计了 4 条启发式规则，分别按照最优均衡性、简单高效原则、最大化模块度兼顾均衡性、最小化割边比兼顾均衡

性实施划分. 多节点分布式系统上的划分质量与性能实验表明: 本文所提算法的划分质量(割边比)比当前最好的在线划分算法改善至少 18.8 个百分点; 多加载器-多划分器架构的设计将图数据加载时间占划分总时间的比例, 从单加载器-单划分器架构流式划分算法的平均 30.8% 降低至平均 20.1%, 缓解了单加载器-单划分器划分算法存在的性能瓶颈; 分布式缓冲机制通过优先划分高度数顶点, 有效缓解了冷启动问题.

作者贡献声明: 史惠康提出了算法思路和实验方案; 王泽胜、胡克坤、董刚完成实验并撰写论文; 赵有健提出指导意见并修改论文.

参 考 文 献

[1] Sakr S, Boifati A, Voigt H, et al. The future is big graphs: A community view on graph processing systems[J]. Communications of the ACM, 2021, 64(9): 62−71

[2] Du Yujie, Wang Zhigang, Wang Ning, et al. Optimization methods for distributed iterative computing performance over multi-dimensional large graph[J]. Journal of Computer Research and Development, 2023, 60(3): 654−675 (in Chinese)
(杜玉洁, 王志刚, 王宁, 等. 分布式多维大图迭代计算性能优化方法[J]. 计算机研究与发展, 2023, 60(3): 654−675)

[3] Fan Wenfei. Big graphs: Challenges and opportunities [C]// Proc of the 48th Int Conf on Very Large Data Bases. New York: ACM, 2022: 3782−3797

[4] Zhai Xiaomeng, Zhang Hong, Huang Xu, et al. Graph partitioning strategies: One size does not fit all[J]. The Journal of Supercomputing, 2022, 78(17): 19272−19295

[5] Dhillon I. Co-clustering documents and words using bipartite spectral graph partitioning [C]//Proc of the 7th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2001: 269−274

[6] Karypis G, Kumar V. Multilevel graph partitioning schemes [C]// Proc of the 1995 Int Conf on Parallel Processing. Piscataway, NJ: IEEE, 1995: 113−122

[7] Jafari N, Selvitopi O, Aykanat C. Fast shared-memory streaming multilevel graph partitioning[J]. Journal of Parallel and Distributed Computing, 2021, 147(1): 140−151

[8] Ji Shengwei, Bu Chenyang, Li Lei, et al. Local graph edge partitioning[J]. ACM Transactions on Intelligent Systems and Technology, 2021, 12(5): 1−25

[9] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs[J]. Journal of Parallel and Distributed Computing, 1998, 48(1): 96−129

[10] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs [C]// Proc of the 18th ACM Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2012: 1222−1230

[11] Nishimura J, Ugander J. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing [C]//Proc of the 19th ACM Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2013: 1106−1114

[12] Faraj M, Schulz C. Buffered streaming graph partitioning[J]. ACM Journal of Experimental Algorithmics, 2022, 27(1): 1−26

[13] Tsourakakis C, Gkantsidis C, Radunovic B, et al. FENNEL: Streaming graph partitioning for massive scale graphs [C]// Proc of the 7th ACM Int Conf on Web Search and Data Mining. New York: ACM, 2014: 333−342

[14] Prabhakaran V, Wu Ming, Weng Xuetian, et al. Managing large graphs on multi-cores with graph awareness [C]// Proc of the 2012 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2012: 41−52

[15] Awadelkarim A, Ugander J. Prioritized restreaming algorithms for balanced graph partitioning [C]// Proc of the 26th ACM Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2020: 1877−1887

[16] Patwary M, Garg S, Kang B. Window-based streaming graph partitioning algorithm [C/OL]// Proc of the 2019 Australasian Computer Science Week Multiconference. New York: ACM, 2019[2023-04-21]. https://www.engineeringvillage.com/app/doc/?docid=cpx_M7eaf008a16906a5f79aM765510178163167&pageSize=25&index=1&searchId=16b73b1aa0124198b09f65fff2d40076&resultsCount=2&usageZone=resultslist&usageOrigin=searchresults&searchType=Quick

[17] Faraj M, Schulz C. Buffered streaming graph partitioning[J]. ACM Journal of Experimental Algorithmics, 2022, 27(9): 1−26

[18] Hu Kekun, Zeng Guosun, Jiang Huawen, et al. Partitioning big graph with respect to arbitrary proportions in a streaming manner[J]. Future Generation Computer Systems, 2018, 80(1): 1−11

[19] Sun Zhipeng, Zeng Guosun, Ding Chunling, et al. A streaming graph partitioning method to achieve high cohesion and equilibrium via multiplayer repeated game[J]. IEEE Transactions on Computational Social Systems, 2022, 11(1): 803−814

[20] Pacaci A, Özsu M. Experimental analysis of streaming algorithms for graph partitioning [C]// Proc of the 2019 Int Conf on Management of Data. New York: ACM, 2019: 1375−1392

[21] Abbas Z, Kalavri V, Carbone P, et al. Streaming graph partitioning: An experimental study[J]. Proceedings of the VLDB Endowment, 2018, 11(11): 1590−1603

[22] Gottesbüren L, Heuer T, Sanders P, et al. Deep multilevel graph partitioning [C]// Proc of the 2021 European Symp on Algorithms. Wadern, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021[2023-04-21]. https://www.engineeringvillage.com/app/doc/?docid=cpx_M6ed4f62317f8aacca4fM60ed1017816328&pageSize=25&index=3&searchId=34b88e32991449758b0b66925964e1eb&resultsCount=4&usageZone=resultslist&usageOrigin=searchresults&searchType=Quick

[23] Akhremtsev Y, Sanders P, Schulz C. High-quality shared-memory graph partitioning[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(11): 2710−2722

[24] Meyerhenke H, Sanders P, Schulz C. Parallel graph partitioning for complex networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(9): 2625−2638

[25] Battaglino C, Pienta R, Vuduc R. GraSP: Distributed streaming graph partitioning [C]// Proc of the 2015 High Performance Graph Mining. Barcelona, Spain: Barcelona Supercomputing Center, 2015[2023-04-21]. https://xueshu.baidu.com/usercenter/paper/show?paperid=9bb30523f8b9a6aa66ddf40fa944ca00&site=xueshu_se

[26] Hanai M, Suzumura T, Tan Wenjun, et al. Distributed edge partitioning for trillion-edge graphs[J]. Proceedings of the VLDB Endowment, 2019, 12(13): 2379−2392

[27] Mayer C, Mayer R, Tariq M, et al. Adwise: Adaptive window-based streaming edge partitioning for high-speed graph processing [C]// Proc of the 38th IEEE Int Conf on Distributed Computing Systems. Piscataway, NJ: IEEE, 2018: 685−695

[28] Petroni F, Querzoni L, Daudjee K, et al. Hdrf: Stream-based partitioning for power-law graphs [C]// Proc of the 24th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2015: 243−252

[29] Sajjad H, Payberah A, Rahimian F, et al. Boosting vertex-cut partitioning for streaming graphs [C]// Proc of the 5th IEEE Int Conf on Big Data. Piscataway, NJ: IEEE, 2016[2023-04-21]. https://www.engineeringvillage.com/app/doc/?docid=cpx_4922c18f15887a4b174M79af10178163171&pageSize=25&index=1&searchId=9eb5c2ddc91c49e88880e18ea9b99248&resultsCount=1&usageZone=resultslist&usageOrigin=searchresults&searchType=Quick

[30] Çatalyürek Ü, Devine K, Faraj M, et al. More recent advances in (hyper) graph partitioning[J]. ACM Computing Surveys, 2023, 55(12): 1−38

[31] Stanford University. SNAP dataset [DB/OL]. [2023-04-21]. http://snap.stanford.edu/data/index.html

[32] Khorasani F, Gupta R, Bhuyan L. Scalable SIMD-efficient graph processing on GPUs [C]// Proc of the 24th Int Conf on Parallel Architecture and Compilation. Piscataway, NJ: IEEE, 2015: 39−50



Shi Huikang, born in 1975. PhD. His main research interest includes electronic and information.

史惠康, 1975 年生. 博士. 主要研究方向为电子与信息.



Wang Zesheng, born in 1987. PhD, senior engineer. His main research interests include cloud computing, and information technology service.

王泽胜, 1987 年生. 博士, 高级工程师. 主要研究方向为云计算、信息技术服务.



Hu Kekun, born in 1987. PhD, senior engineer. Member of CCF. His main research interests include graph processing and graph deep learning.

胡克坤, 1987 年生. 博士, 高级工程师. CCF 会员. 主要研究方向为图计算、图深度学习.



Dong Gang, born in 1979. PhD, senior engineer. Member of CCF. His main research interests includes FPGA design, architecture and applications.

董　刚, 1979 年生. 博士, 高级工程师. CCF 会员. 主要研究方向为 FPGA 芯片架构设计及其应用.



Zhao Youjian, born in 1969. PhD, professor, PhD supervisor. His main research interests include high-speed Internet architecture, Switching and routing, and high-speed network equipment.

赵有健, 1969 年生. 博士, 教授, 博士生导师. 主要研究方向为高速互联网体系结构、交换与路由、高速网络设备.