OnFlyP:基于定向边交换的分布式在线大图划分算法

王志刚 谷 峪 鲍玉斌 于 戈
(东北大学信息科学与工程学院 沈阳 110819)

摘 要 随着大数据时代的到来,基于云环境的大图迭代计算已经成为新的研究热点,其中提高图划分算法的执行效率和降低划分后子图之间的通信边规模是改善计算性能的关键.已有工作主要分为离线划分和在线划分两大类,无法在执行效率和通信边规模方面同时满足迭代处理需求.文中针对真实世界的大图,提出了聚簇系数概念,定量分析了顶点分布的局部性,以此为基础设计了一种基于定向边交换模型的分布式在线图划分算法(OnFlyP),可在迭代计算的数据加载阶段快速完成图划分,同时通过出边的交换有效降低通信边规模,以满足迭代计算需求.OnFlyP采用实时控制和最小对称矩阵控制实现负载均衡,前者具有较高的执行效率,而后者对降低通信边规模有较好的优化效果,可根据实际应用的处理需求灵活选择.最后,作者使用多种真实数据验证了OnFlyP算法的有效性.

关键词 在线大图划分;边交换;实时控制;最小对称矩阵
中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2015.01838

OnFlyP: An Online Distributed Partition Algorithm for Large Scale Graphs Based on Edge-Exchange Model

WANG Zhi-Gang GU Yu BAO Yu-Bin YU Ge
(*College of Information Science and Engineering, Northeastern University, Shenyang* 110819)

Abstract With the arrival of the big data era, the iterative computation of large graphs on cloud computing environments has attracted a lot of attention as a new hot topic. However, the overall computation performance greatly depends on graph partition methods in terms of improving the executing efficiency and reducing the number of communication edges among different subgraphs. Although a large of efforts have been made to tackle this issue, such as offline or online partition methods, the requirements of the two aforementioned aspects are hard to be satisfied simultaneously. This paper proposes the concept of "cluster coefficient" and then analyzes the locality of vertex distribution for real-world graphs. Accordingly, an online distributed partition algorithm (OnFlyP) based on a directional edge-exchange model is presented to efficiently support iterative computations. For a specific iterative algorithm, OnFlyP can be executed with high efficiency during the phase of loading data. Meanwhile, it greatly reduces the communication edge scale by exchanging edges. OnFlyP employs the real-time control policy or the minimum symmetric matrix policy, to implement the load balance among subgraphs. They respectively focus on the high-efficiency and the effect of reducing the communication edge scale, and are

收稿日期:2014-07-05;最终修改稿收到日期:2015-04-01.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316201)、国家自然科学基金(61272179,61472071,61173028)、中央高校基本科研业务费专项资金重点课题(N120816001)、教育部博士点基金(20120042110028)和教育部-中国移动科研基金项目(MCM20125021)资助.**王志刚**,男,1987年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为数据库系统与云计算.E-mail:wangzhigang1210@163.com.**谷 峪**,男,1981年生,博士,副教授,中国计算机学会(CCF)高级会员,主要研究方向为图数据管理、空间数据管理等.**鲍玉斌**,男,1968年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为海量数据管理、云计算等.**于 戈**(通信作者),男,1962年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为数据库理论和技术、分布式与并行系统等.E-mail:yuge@mail.neu.edu.cn.

suitable for different real applications. Finally, extensive experiments on various real-world graphs validate the effectiveness of OnFlyP.

Keywords online large-scale graph partition; edge exchange; real-time control; minimum symmetric matrix

1 引　言

图可以表达复杂的结构和丰富的语义，其迭代分析算法在社交网络、Web、时空和科学数据计算等诸多领域都获得了广泛的应用[1-2]. 然而，数据规模的快速增长对高效迭代处理提出了严峻的挑战. 例如，Google 目前索引的网页数目已超过 1 万亿，而 Facebook 2012 年的活跃用户已经超过 10 亿. 为应对大图的迭代分析需求，近年来，基于云计算的大量分布式迭代处理系统被开发出来，如 Pregel，Trinity，GPS，Giraph，Spark 和 BC-BSP 等[3-9]. 这些系统通过表达力丰富的 API 简化了用户的分布式编程工作，并利用云计算环境的海量资源实现了大图的有效处理.

图划分是 Pregel 等系统进行分布式计算的前提. 由于图计算通常按照拓扑结构访问数据，所以每次迭代处理均会引入巨大的通信开销，这成为制约分布式处理性能的关键因素. 因此，一个良好的划分算法应保证划分后的子图在负载均衡的前提下，减少子图之间的交互边（切分边）规模，从而减少网络通信. 另一方面，云计算资源会随着并发处理作业数目的变化和集群中节点的增删而动态变化. Pregel 等系统使用集群总任务数目来定量描述计算资源，使用空闲任务数目表征当前可用计算资源，并提供 *setTaskNum* 接口，允许用户在提交图处理作业时，根据处理需求和计算资源灵活定义本作业的任务数. 因此，同一个图作业在不同时刻被提交时，其分布式任务数目不尽相同，我们称之为分布式处理粒度的弹性变化. 这导致图数据需要按照当前的任务数目重新划分，划分结果的不可重用性使图划分的执行效率成为影响总计算代价的重要因素.

然而，均衡图划分本身是一个经典的 NP-Complete 问题[10]，目前的研究工作可以分为两类，以 Metis[10] 为代表的离线划分算法和以 LDG[11] 为代表的在线划分算法. 前者可以显著优化切分边规模，降低迭代计算过程中的消息通信开销，因此受到学术界和工业界的广泛关注. 然而，离线划分过程需要频繁访问图顶点，引入了昂贵的时间开销. 面对海量图数据，例如 Twitter 图（4700 万顶点，15 亿出边），其执行时间超过 8 个小时[12]，效率低下. 在线划分算法可以在图处理系统的数据加载阶段完成图划分，仅扫描一次图数据. 与离线划分算法相比，在线划分算法通过一定程度上牺牲划分效果，来获得较高的执行效率. 但是，此类算法通常是集中式算法，这样便于维护复杂的启发式规则，保证相对较好的划分效果，但其扩展性显然受到单机处理能力的限制. 虽然已经存在分布式在线划分算法的相关研究工作，但是启发式规则的维护开销仍然显著影响了算法的运行效率.

综上所述，设计一种划分效果优异的快速的图分割算法，已经成为现有大图处理系统亟待解决的问题. 实际上，真实图通常是通过广度优先搜索技术（Breadth-First Search，BFS）爬取得到[13]，以供 Pregel等计算系统使用. 我们称之为 BFS 生成图. 这使得爬取后的真实图的顶点分布具有一定局部性，而已有的相关工作忽视了这一特性. 因此，本文首先提出了反映顶点之间关联关系的聚簇系数概念，定量分析了顶点分布的局部性，然后据此设计了一种基于出边定向交换模型的分布式在线图划分算法 OnFlyP. 利用原始图的局部性特征，OnFlyP 算法能够在保证分布式划分效率的前提下，有效降低实际计算过程中的网络通信开销. 特别地，OnFlyP 支持实时控制（OnFlyP-*R*）和最小对称矩阵控制（OnFlyP-*M*）两种负载均衡策略. 前者可以获得更好的执行效率，而后者可以获得更佳的划分效果. 在实际应用中，用户可以根据具体需求，自主选择不同策略. 例如，针对单源最短路径计算等遍历式算法，迭代过程中通信压力较小，可以选择 OnFlyP-*R* 策略减少划分开销，提高整体处理效率. 而对 PageRank 等通信密集型算法，可以选择 OnFlyP-*M* 策略以提高通信效率. 综述，本文的主要贡献如下：

（1）提出聚簇系数概念，定量分析了真实图（尤其是 BFS 生成图）的顶点分布的局部性.

（2）设计了基于定向边交换模型的分布式在线图划分算法 OnFlyP，并从理论上分析了 BFS 生成

图划分后的通信边规模的上界值.

(3)设计了基于实时控制和最小对称矩阵控制的两种负载均衡策略,适用于不同实际需求.

(4)在大量真实图上,对比了 OnFlyP 算法和集中式在线 LDG 算法,并在 BC-BSP 系统上验证了 OnFlyP 算法的有效性.

本文第 2 节介绍相关工作;第 3 节通过描述 Pregel 系统处理流程,定义图划分概念;第 4 节首先分析 BFS 生成图的局部性特点,然后介绍 OnFlyP 划分算法,最后从理论上对划分效果进行分析;第 5 节给出实时控制和最小对称矩阵控制两种负载均衡策略,并对两种策略的适用性进行详细讨论;第 6 节展示真实图上的实验结果和数据分析;最后,第 7 节总结全文.

2 相关工作

本节首先从离线划分和在线划分两个方面,对现有的静态的图划分算法进行介绍,然后简单总结动态的图划分方法,从而阐述本文工作与已有工作的区别.

离线划分算法的一个经典处理方案是通过对原图的多级"coarsening"操作不断压缩图的规模,然后对压缩图采用 Kernighan-Lin[14] 或 FM[15] 等复杂算法进行初始划分,之后执行"uncoarsening"操作,得到最终的划分结果. Chaco[16],Metis[10] 和 Scotch[17] 等都是面向集中式的多级划分算法库,且因划分后的子图具有较少的切分边而被业界广为青睐. 此外,ParMetis[18] 和 PT-Scotch[19] 等并行化版本可进一步提高扩展性. 然而此类算法在"coarsening"阶段的最大匹配操作引入了昂贵的执行开销. 因此,MLP[20] 采用分布式的基于标签传播的连通域查找算法替换"coarsening"阶段的最大匹配,以降低执行开销. 此外,部分工作直接采用标签传播进行图划分. 例如,Ugander 等人[21] 提出一种分布式环境下的均衡的标签传播方法来解决图划分问题. 每个顶点采取其邻居顶点具有的最多的标签作为自己的标签,而具有相同标签的顶点属于同一个划分后的子图. 该方法采用线性规划来约束每个子图的大小,实现负载均衡. Rahimian 等人[22] 设计了 JA-BE-JA 算法,将顶点的标签称为 color,每个顶点和自己的邻居以及部分随机顶点交换标签,以减少子图之间的交互边. JA-BE-JA 采用模拟退火算法来避免算法陷入局部最优. 进一步地,为满足实际应用中的多种需求,Slota 等人提出一种基于标签传播的多目标划分方法 PuLP[23]. PuLP 在不同的阶段针对不同的约束条件进行划分调整. 然而,上述 3 种算法在标签传播过程中,每个顶点需要迭代更新自己所属的类别,所以仍需要扫描多次图数据,制约执行效率. 此外,MLP 区分了切分边和通信边的概念. 如果从子图 A 指向子图 B 的切分边具有相同的目的顶点,那么在实际计算中,切分边产生的消息可以通过合并操作转化为一条网络消息,等价于一条通信边. 因此通信边规模代表了实际的网络开销.

在线划分算法假设图数据以顶点流或边流的方式到达,在流处理过程中根据已到达数据的分布信息,通过启发式规则计算决定当前图数据的划分位置. 由于 Pregel 等系统在执行迭代计算前必须加载图数据,因此该方法可以在数据加载过程中实现图的在线划分. 与 Metis 相比,该方法通过一定程度上牺牲切分边规模的优化效果来避免"coarsening"阶段的多次数据扫描,极大地提高了执行效率. 集中式的 LDG[11] 和 FENNEL[12] 算法可以准确地实时维护数据的分布信息,保证了启发式计算结果的精度,切分边规模较少. 进一步地,Nishimura 等人[24] 提出了"restreaming"方法,即相同的图数据被反复加载、处理时(例如,首次加载用于计算单源最短路径,第 2 次加载用于计算 PageRank),可以利用上一次流式划分的结果,来改善本次划分的效果. 实验表明,该机制可以提供与 Metis 近似的划分效果. PowerGraph 提出了基于 Vertex-Cut 的Coordinated 分布式流划分方法[25],通过建立分布式路由表记录数据的分布信息,实现了分布式的启发划分. 鉴于 Vertex-Cut 机制可有效减少迭代计算的通信开销,GraphBuilder,GraphX 和 LightGraph 等系统[26-28],均支持这种划分方法. 然而,为提高路由表信息的准确性,分布式的 Vertex-Cut 机制必须引入昂贵的加锁开销,影响了实际运行效率. 因此,PowerLyra[29] 提供了一种基于 Vertex-Cut 和FENNEL的混合划分方式,用户需设定阈值,如果顶点的出度大于该阈值,执行 Vertex-Cut 以提高通信收益,否则采用 FENNEL 划分以提高划分执行效率并降低划分后子图的维护开销. 此外,我们曾在 BC-BSP[8] 系统中设计了一种简单的 VCCP 方法[30],通过采用 Vertex-Cut 技术[25],可显著减少通信边规模,但对真实图会引入极大的负载偏斜.

Hash划分可以视为一种最简单的分布式流划

分算法，其启发式规则为：通过顶点标签或边标签的 *HashCode* 值决定数据的存放位置. 虽然 Hash 划分通常由于难以保留图的局部性而引入大量的通信开销，但可以在分布式环境下快速实现图的分割，因此被 Pregel 等系统作为默认的划分方式.

上述图划分工作均是围绕静态图展开的，而实际应用中，对于 Pregel 等迭代处理系统，图是动态变化的. 这是因为在迭代过程中，值达到稳定的图顶点通常不再参与后续计算，例如单源最短路径计算，所以实际参与计算的图数据是动态变化的. 这使得初始的图划分可能产生负载偏斜，或者划分效果不再是最优的. 因此，GPS[5]，Mizan[31]和 Xpregel[32]等系统均支持在迭代过程中动态重划分图数据，以提高整体运行效率. 特别地，Shang 和 Jeffrey 等人[33]分析了在不同迭代算法下图的动态变化特征，用于指导动态划分. 动态划分机制是在给定的划分结果上进行动态调整. 因此，与本文的工作是互补的，即本文提出的方法，也可以支持类似的动态调整操作.

3　问题定义

本节首先以 Pregel 系统为例介绍了大图迭代计算流程，然后根据需求定义了图划分的相关概念.

3.1　大图迭代处理流程

Pregel 是 Google 公司为处理海量图数据而开发的基于 BSP 模型的分布式迭代处理系统. 如图 1 所示，作为一个以顶点为中心的基于 Master-Slave 架构的系统，Pregel 采用邻接表组织图数据. Master 是分布式系统的控制中心，而 Slave 为工作节点.

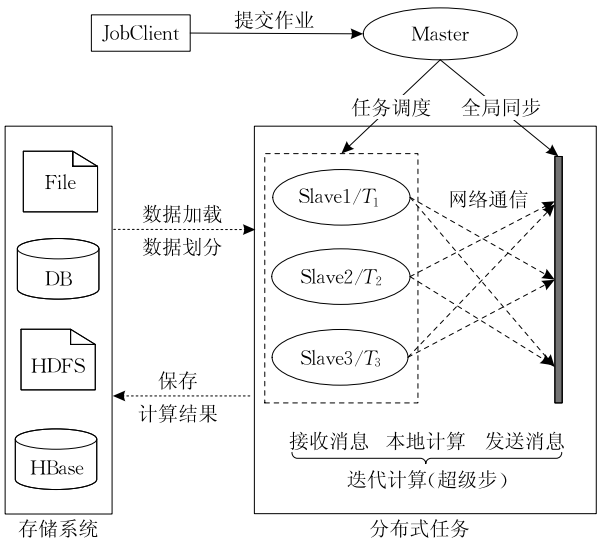


图 1　Pregel 迭代处理流程

用户的图处理作业通过 JobClient 向 Master 提交，而 Master 将一个图处理作业分割为若干任务(T_i)并分配给 Slave 节点执行. 具体计算流程为：(1)数据加载/划分，各任务从分布式存储系统并行加载图数据至本地，然后进行图划分，每个任务处理一个子图；(2)迭代处理，每次迭代计算视为一个超级步，两个超级步之间通过全局同步来协调各任务的处理进度，任务之间通过消息交换中间结果，在每个超级步中，图顶点接收消息、执行本地计算并发送消息，因此计算负载通常由出边数目决定；(3)迭代收敛后，保存计算结果. 根据 Pregel 的处理流程，一个良好的图划分算法应满足如下约束：(1)在线划分，可根据任务数目，在加载的同时，在线完成图的分割；(2)任务是基本的计算单位，应满足出边数目的负载均衡；(3)减少任务之间的网络通信规模，即子图之间的通信边规模.

3.2　大图划分定义

与传统的通用图划分不同，本节将根据 3.1 节的分析，定义分布式迭代计算环境下图划分的概念.

定义 1.　图划分. 给定有向图 $G=(V,E)$，包含 $|V|$ 个顶点和 $|E|$ 条边，图划分是将图 G 划分为 K 个子图 $G_i=(V_i,E_i)$，同时满足式(1)的约束，其中，$|G|$ 为计算负载，这里 $|G|=|E|$，$\bigcup_{i=1}^{K}E_i=E$ 且 $E_i\cap E_j=\varnothing$，$i\neq j$，而 ρ 为负载偏斜因子，理想值为 1.0.

$$\begin{cases}\max\limits_{i\in[1,K]}\{|G_i|\}\leq\rho\cdot\dfrac{|G|}{K}\\ \min\limits\sum\limits_{i,j\in[1,K]}|E_{ij}^{\mathrm{com}}|\end{cases}\tag{1}$$

式(1)中的 E_{ij}^{com} 为子图 G_i 到 G_j 的通信边集合.

定义 2.　通信边集合. 给定一条边 $e=(u,v)$，如果 $u\in V_i\wedge v\in V_j$，且 $i\neq j$，则称 e 为从子图 G_i 到子图 G_j 的一条通信边. 子图 G_i 到 G_j 的所有通信边构成通信边集合 E_{ij}^{com}.

我们将在第 4 节中，结合具体实例进一步介绍定义 1 和定义 2. 此外，为便于分析，本文假设图按照邻接表存储且图顶点按照存储顺序连续编号.

4　分布式在线图划分算法 OnFlyP

本节首先定量分析了 BFS 生成图中顶点分布的局部性，提出了 OnFlyP 划分算法，然后从理论上分析了 OnFlyP 算法的通信边规模的上界值.

4.1 真实图的局部性分析

原始输入图通常是采用广度优先搜索(BFS)从物理世界抽取构建[13],其顶点的分布具有一定局部性,本文的 OnFlyP 算法正是利用这一特性来降低通信边规模.下面首先介绍相关概念.

定义 3. 前向边集.在图的爬取过程中,假设 Q_w 为顶点的 BFS 搜索队列,Q_p 为已搜索过的顶点队列,每次移除 Q_w 的队首元素 v,进行新的 BFS 扩展,然后将 v 放入 Q_p 中.Γ_v 为 v 的出边的目的顶点集合,则本次扩展中新加入 Q_w 的顶点集合为 $V_{\text{new}}=\{u \mid u\in\Gamma_v \wedge u\notin(Q_w\cup Q_p)\}$,而与之关联的出边称为顶点 v 的前向边集 $E_v^{\text{for}}=\{e=(v,u) \mid u\in V_{\text{new}}\}$.

定义 4. 顶点分布的局部性.在 BFS 爬取过程中,采用邻接表顺序存储从 Q_w 中移除的队首元素及其出边,则对于前向边集中的目的顶点,其存储顺序与图构建时的搜索顺序一致,在存储位置上相邻,称之为顶点分布的"局部性".

定义 5. 回溯边集.在 BFS 爬取过程中,令 E_v 为顶点 v 的出边集合,则与前向边集对应,回溯边集 $E_v^{\text{back}}=\{e \mid e\notin E_v^{\text{for}} \wedge e\in E_v\}$.

如图 2 所示,以 1 号顶点为起始顶点进行 BFS 搜索,邻接表中顶点的存储顺序就是原图中顶点的 BFS 拓扑排序结果.例如 1 号顶点的前向边集 $E_1^{\text{for}}=\{(1,2),(1,3)\}$,则顶点 2 和 3 的存储位置必然是相邻的.

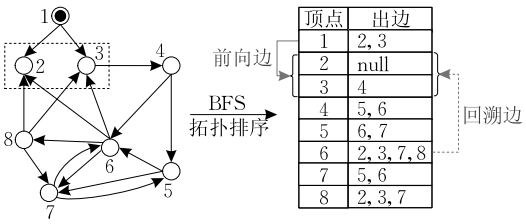
此外,在真实世界中,顶点的出边行为通常具有相似性,即指向相同的顶点,以社交网络为例,这种相似性表现为 A 与 B 的好友在很大程度上是重叠的,比如他们共同关注某些明星.这种特性使得回溯边集中出边的部分目的顶点,与前向边集中的目的顶点相似,也具有顶点分布的局部性.定义 6 给出了出边分布相似性的度量方法,可用于度量图中顶点分布的局部性.

定义 6. 出边分布的相似性.$\forall v,u\in V$,使用 Jaccard 系数度量其出边的相似性(重叠程度[34]):

$$s(\Gamma_v,\Gamma_u)=J(\Gamma_v,\Gamma_u)=\frac{|\Gamma_v\cap\Gamma_u|}{|\Gamma_v\cup\Gamma_u|},$$

$s(\Gamma_v,\Gamma_u)$ 值与相似性程度成正比.

如图 2 中的顶点 6,有 $E_6=\{(6,2),(6,3),(6,7),(6,8)\}$ 和 $E_1=\{(1,2),(1,3)\}$,故 $s(\Gamma_1,\Gamma_6)=0.5$.其中(6,7)和(6,8)属于 E_6^{for},而(6,2)和(6,3)的目的顶点与 E_1^{for} 中的目的顶点相同,所以集合{7,8}和{2,3}中集合内的顶点的存储位置临近.因此,$s(\Gamma_1,\Gamma_6)$,可以反映出对应目的顶点分布的局部性.



顶点	出边
1	2,3
2	null
3	4
4	5,6
5	6,7
6	2,3,7,8
7	5,6
8	2,3,7

图 2 BFS 生成图的邻接表存储顺序

由上述分析可知,$\sum_{v,u\in V}s(\Gamma_v,\Gamma_u)$ 与邻近存储的目的顶点的规模成正比.然而,如果直接使用 $\sum_{v,u\in V}s(\Gamma_v,\Gamma_u)$ 作为局部性的衡量指标,其过高的计算复杂度 $O((\max d)^\eta)$ 对于大图是不可行的,其中 $\max d$ 为图 G 的最大出度,而 $\eta=|V|(|V|-1)/2$,$|V|$ 为顶点数目.考虑到顶点顺序编号的情况下,编号值反映了其存储位置的"邻近程度",因此我们采用聚簇系数来定量描述顶点分布的局部性.

定义 7. 聚簇系数.给定真实图 $G=(V,E)$,令 G 和 G_{rand} 的局部系数的比值为真实图的聚簇系数:$C(G)=F(G_{\text{rand}})/F(G)$,其中,$G_{\text{rand}}$ 是与真实图 G 对应的随机图,具有 $|V|$ 个顶点和 $|E|$ 条出边但顶点是随机分布的.局部系数 $F(G)$ 由 G 中所有顶点的局部距离 $f(v)$ 通过求和计算得到,$f(v)$ 的计算方式为

$$f(v)=\begin{cases}\sum_{u\in sort(\Gamma_v)}\{|Code(u_{i+1})-Code(u_i)|\}, & |\Gamma_v|>1\\ 0, & |\Gamma_v|\le 1\end{cases},$$

其中,$sort(\Gamma_v)$ 表示将 Γ_v 中顶点按编号升序排序,$Code(u_{i+1})$ 为升序序列中第 $i+1$ 个顶点 u_{i+1} 的编号值.

$F(G)$ 的值与局部性程度成反比,而 C 值则反映了真实图与随机图的局部距离的比值.给定顶点规模和出边规模后,$F(G_{\text{rand}})$ 的值是固定的,因此,C 值与 $F(G)$ 的值成反比,即与真实图的局部性程度成正比.第 6 节中的实验结果显示,真实图均具有较高的 C 值,尤其对于 BFS 生成图.聚簇系数只是用于验证真实图具有较高局部性的衡量指标,并不是 OnFlyP 算法的直接输入参数.但是这种局部性,是 OnFlyP 算法能够有效减少通信边规模的基础(见 4.2.3 小节).

4.2 OnFlyP 划分算法

OnFlyP 算法依赖于 4.1 小节所分析的局部性,因此本节首先介绍我们在 BC-BSP 系统上已经实现的 Range 划分,可保留原始图局部性,然后描述 OnFlyP 划分的处理流程,最后分析 BFS 生成图经过

OnFlyP 划分后，各子图间通信边规模的上界值.

4.2.1　BC-BSP 的 Range 划分

Range 划分是保留原始图局部性(邻接表存储顺序)的一种直观方法，已被 Giraph 和 BC-BSP 所采用[6,8]. 在 BC-BSP 中，Range 划分直接将原始图的邻接表文件切分为若干连续子块. 若以 HDFS 为存储介质，在数据加载前，通过读取元数据信息，可将输入图按照字节大小均等划分为若干分片($Split$)，每个任务负责加载一个分片所对应的邻接表数据. K 个任务并行加载的图数据直接驻留任务本地，然后建立反映顶点分布的路由信息表 $R=\{R_1,R_2,\cdots,R_K,R_{K+1}\}$，为后续迭代计算中的消息寻址提供服务. 其中 R_{K+1} 为 G 的最大顶点编号，R_i 是 T_i 所加载图数据的最小顶点编号，且有 $R_x<R_{x+1}$. 给定消息的目的顶点 v，根据式(2)可计算得到其所在任务 T_i.

$$i=x,\ R_x\leq getId(v)<R_{x+1},\ x\in[1,K] \quad (2)$$

以图 2 为例，假设通过 Range 方法将原图划分为 3 个子图，则划分后的顶点集合为 $V_1=\{1,2,3,4\}$，$V_2=\{5,6\}$，$V_3=\{7,8\}$. 而路由信息表 $R=\{1,5,7,8\}$. 由于按照字节大小均等分割，所以每个子图的负载 $|G_i|=|E_i|$ 是相对均衡的. 在本例中，有 $|G_1|=5$，$|G_2|=6$，$|G_3|=5$，负载偏斜因子 $\rho=1.125$.

Range 划分保留了第 4.1 节中的局部分布特点. 如顶点 1、2 和 3 均位于子图 G_1，由任务 T_1 处理，所以边(1,2)和(1,3)不引入网络通信. 注意到顶点 5 和 6 均是 4 号顶点的出边顶点且在存储位置上相邻，但由于三者所在子图不同，(4,5)和(4,6)成为通信边. 类似地，回溯边(6,2)和(6,3)也是通信边. 在 4.2.2 小节中，我们将介绍基于定向边交换模型的 OnFlyP 算法来降低上述通信边的规模.

4.2.2　OnFlyP 划分

首先引入定向边交换模型的概念.

定义 8.　定向边交换模型. Range 划分模式下，任务 T_i 加载的子图为 $G_i=\{V_i,E_i\}$，任取顶点 $v\in V_i$，则 $E_{ij}^{com}[v]=E_{ij}^{com}\cap E_v$ 中的边可以被定向地迁移到任务 T_j，即为定向边交换模型.

OnFlyP 划分是建立在定义 8 的基础之上. 在 OnFlyP 划分中，迁移的出边 $E_{ij}^{com}[v]$ 在任务 T_j 中仍以邻接表格式组织，其中 T_j 中的顶点 v 是冗余备份，记为 v^b，而 T_i 中的顶点 v 是主备份. 为同步 v^b 和 v，迁移后为 G_i 添加一条出边，$E_i=E_i\cup\{(v,T_j)\}$. 而对 G_j，有 $E_j=E_j\cup E_{ij}^{com}[v]$，$V_j=V_j\cup\{v^b\}$. 因此，定向迁移 $E_{ij}^{com}[v]$ 后，按照定义 2，E_{ij}^{com} 的规模变为 $|E_{ij}^{com}|-|E_{ij}^{com}[v]|+1$. 显然，只有 $|E_{ij}^{com}[v]|\geq 2$ 时，才会降低通信边集的规模. 具体地，对于顶点 v，其出边迁移方向的集合为 $\{T_x||E_{ix}^{com}[v]|\geq 2\}$，令 Γ_v 表示 v 的出度顶点集合，则 $|E_{ix}^{com}[v]|=|\Gamma_v\cap V_x|$，后者可以通过路由表 R 使用式(2)快速计算得到.

如图 3 所示，OnFlyP 划分后，T_1 的顶点 4 将出边(4,5)和(4,6)迁移到任务 T_2，对应地，任务 T_2 中的顶点 6 也将出边(6,2)和(6,3)迁移到 T_1，出边(6,7)和(6,8)迁移到 T_3. 而 T_3 中的顶点 7 和顶点 8 也做了类似处理. 以 G_1 为例，划分后，$V_1=\{1,2,3,4\}\cup\{6,8\}$，其中{6,8}为冗余备份顶点. 任务 T_1 与 T_2 之间的通信边集为 $E_{12}^{com}=\{(4,T_2)\}$ 和 $E_{21}^{com}=\{(6,T_1)\}$，其中 T_2 表示 4 号顶点在 T_2 上存在冗余备份，而 T_1 表示 6 号顶点在 T_1 上存在冗余备份. 显然，从网络通信角度，执行 OnFlyP 划分后，$|E_{12}^{com}\cup E_{21}^{com}|$ 的值由 4 降为 2，类似地，$|E_{13}^{com}\cup E_{31}^{com}|$ 的值由 2 降为 1，$|E_{23}^{com}\cup E_{32}^{com}|$ 的值由 5 降为 3.

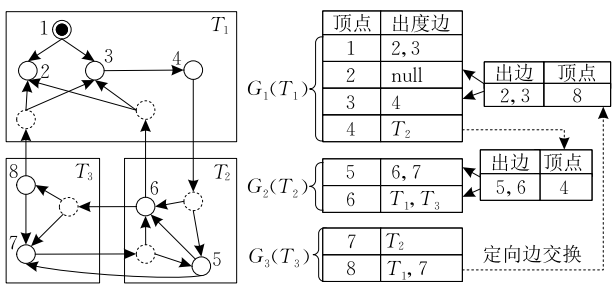


图 3　OnFlyP 划分结果

OnFlyP 算法采用 Master-Slave 架构，如算法 1 所示，在各任务加载数据前，Master 首先从分布式文件系统上读取图 G 的元数据信息，建立 K 个 Split 分片供各任务加载图数据，然后 Master 读取每个分片对应的首条邻接表数据，得到每个任务负责处理的最小图顶点编号，从而构造路由表 R，之后将 R 和 Split 信息发送给所有任务.

算法 1.　分布式在线图划分算法 OnFlyP.

输入：待划分图 G，任务数目 K，图顶点数 $|V|$

输出：图划分结果 $\{G_i\}$

OnFlyP($G,K,|V|$)

1. *Master*:
2. $R=build(G,|V|)$; //构建路由表 R
3. send R and $Splits$ to all Tasks;
4. *Task i* (T_i):
5. receive R and $Split_i$ from *Master*;
6. WHILE $Split_i\neq\varnothing$ DO
7. $adj(v)=loadGraph(Split_i)$; //加载邻接表

8. $\{(T_x, E_{ix}^{com}[v])\} = get(getEdges(adj(v)), R)$;
9. FOR $\forall (T_x, E_{ix}^{com}[v]) \in \{(T_x, E_{ix}^{com}[v])\}$ DO
10. IF $assert(T_x, E_{ix}^{com}[v])$ DO
11. $send\ (v, E_{ix}^{com}[v])$ to T_x;
12. $updateEdges(adj(v))$;
13. END IF
14. END FOR
15. END WHILE
16. $V_i = V_i \cup V^b, E_i = E_i \cup E^{syn} \cup E^{in}$;
17. RETURN $G_i = (V_i, E_i)$;

在 Slave 端,任务 T_i 的处理流程为:首先接收路由表 R 和对应的分片信息 $Split_i$,然后逐条加载邻接表数据 $adj(v)$,将 $adj(v)$ 中的出边按照定向边交换模型计算交换方向,其中 $(T_x, E_{ix}^{com}[v])$ 表示顶点 v 可以迁移到任务 T_x 的出边集合为 $E_{ix}^{com}[v]$. 对集合 $\{(T_x, E_{ix}^{com}[v])\}$ 中的每个元素,将调用 $assert()$ 进行迁移判定,如果判定结果为 true,则将 $(v, E_{ix}^{com}[v])$ 发送到任务 T_x,同时将 $adj(v)$ 的出边中的 $E_{ix}^{com}[v]$ 替换为 T_x,否则不予处理. $assert()$ 的判定标准为当 $i \neq x$ 时,取 $|E_{ix}^{com}[v]| \geq 2$ 的判定结果作为 $assert()$ 的返回值,否则 $assert()$ 为 false,以保证通信收益. 不同的 $assert()$ 判定方式将影响负载均衡和划分效果,我们将在第 5 节详细介绍. 各任务上冗余备份顶点的集合记为 V^b,用于同步冗余备份顶点的出边记为 E^{syn},接收的迁移边记为 E^{in}. 显然,算法 1 可在数据加载阶段在线完成图划分.

4.2.3 OnFlyP 算法的通信边集与计算开销分析

本节针对 BFS 生成图,从理论上分析 OnFlyP 算法划分后,子图之间的通信边集规模的上界值.

引理 1. 图 $G=(V,E)$,假设通过 OnFlyP 算法分成 K 个子图 G_i,由 G_i 与 G_j 之间的前向边集 E_{ij}^{for} 所产生的通信边集记为 $for(E_{ij}^{com})$,若 K 远小于 $|V|$,则所有前向边集构成的通信边集规模的上限为

$$\max_{i,j}\Big\{\sum_{i\in[1,K]}\sum_{j\in[1,K]}|for(E_{ij}^{com})|\Big\} = |V| + (K-1).$$

证明. 因 K 远小于 $|V|$,可认为对 $\forall v \in V$,有 $|E_v^{for}| < |E_v| = |\Gamma_v| < \min\{|V_i|\}, i \in [1,K]$,$\Gamma_v$ 为目的顶点集合. 另一方面,由第 4.1 节分析知,E_v^{for} 所对应的目的顶点是严格连续分布的. 因此,E_v^{for} 被定向迁移后,目的顶点最多跨越两个子图,而这种情况最多发生 $(K-1)$ 次. 因此冗余顶点的上限值为 $|V|+(K-1)$,即为通信边集规模. 证毕.

引理 2. 图 $G=(V,E)$,假设通过 OnFlyP 算法分成 K 个子图 G_i,由 G_i 与 G_j 之间的回溯边集 E_{ij}^{back} 所产生的通信边集记为 $back(E_{ij}^{com})$,若 K 远小于 $|V|$,且回溯边所对应的目的顶点严格的连续分布,则 G 中由所有回溯边集构成的通信边集规模的上限为

$$\max_{i,j}\Big\{\sum_{i\in[1,K]}\sum_{j\in[1,K]}|back(E_{ij}^{com})|\Big\} = 2|V|.$$

证明. 与引理 1 的证明过程类似,但由于回溯边集的目的顶点并非全局有序,每个顶点的回溯边均可能指向两个子图. 所以每个顶点的回溯边被迁移后,最多产生两个冗余备份顶点. 证毕.

定理 1. 对于 BFS 生成图,采用 OnFlyP 算法划分为 K 个子图后,若 K 远小于 $|V|$,且前向边集和回溯边集所对应的图顶点均严格连续分布,则所有子图之间的通信边集规模的上界值为 $3|V|+(K-1)$.

证明. 对于顶点 v,其边集 $E_v = E_v^{for} \cup E_v^{back}$,且有 $E_v^{for} \cap E_v^{back} = \varnothing$,根据引理 1 和引理 2,显然冗余备份顶点规模的上限为 $|V|+(K-1)+2|V|$. 证毕.

需要注意的是,OnFlyP 算法虽可降低通信规模,但也增加了额外开销:备份顶点的同步消息开销和更新时的 CPU 计算开销,其对整体性能的影响可用式(3)估算. 其中,x 为备份顶点数目,c_{net} 为一条消息的网络通信开销,c_{cpu} 为一个备份顶点更新时的计算开销,而 $assert()$ 的判定标准为 $|E_{ij}^{com}[v]| \geq 2$,因此降低的通信边规模 $X \geq 2x$. 此外,单位数据的网络传输开销通常大于其 CPU 计算开销,即 $c_{net} > c_{cpu}$,故有 $\Phi(x) \geq 0$,即不考虑负载偏斜所引起的水桶效应的前提下,OnFlyP 会提高整体计算性能,且提升比例与备份顶点数目成正比.

$$\Phi(x) = X \cdot c_{net} - x \cdot (c_{net} + c_{cpu}) \quad (3)$$

5 负载均衡控制机制

真实图通常存在幂率偏斜特点,即少数顶点关联了大部分边,在第 4 节的 OnFlyP 算法中,我们使用 $|E_{ij}^{com}[v]| \geq 2$ 作为 $assert()$ 的判定标准,可能会导致大量的出边被迁移到少数任务上,造成负载偏斜. 本节我们将介绍两种负载均衡控制机制,作为 OnFlyP 算法中 $assert()$ 的判定标准.

5.1 实时控制机制

在实时控制机制中,我们为每个任务 T_i 设置一个维度为 K 的实时阈值向量 $\boldsymbol{\theta}_i$. 假设 t 时刻 G_i 欲向 G_x 执行 send 操作,则 $assert()$ 判定方法为:如果满足 $|E_{ix}^{com}[v]| \geq \boldsymbol{\theta}_{ix}^t$,结果为 true,否则为 false. 其中,$\boldsymbol{\theta}_{ix}^t$ 是 t 时刻指向任务 T_x 的出边迁移阈值,而 $\boldsymbol{\theta}_{ii}^t \equiv +\infty, x \in [1,K]$. $\boldsymbol{\theta}_{ix}^t$ 的初始值为 2,并根据 send 操作

的结果实时更新. 更新规则为：如果子图 G_x 在 t 时刻的负载大于平均负载，即 $real(G_x)|_t > |E|/K$，则 send 操作的返回值为 false，$\boldsymbol{\theta}_{ix}^t$ 被设置为无穷大，否则仍为 2. 据此，当 send 所指向的目的任务的负载过重时，将禁止其它任务向其继续转移出边. $real(G_x)|_t$ 由式(4)计算得到，其中 $j,l \in [1,K]$ 且均不等于 x，$load(E_x)|_t$ 是截至 t 时刻已加载的 G_x 中的出边数，V_t^{rec} 是 G_x 所接收的冗余顶点集合，而 V_t^{sen} 则是 G_x 所发送的冗余顶点集合.

$$real(G_x)|_t = load(E_x)|_t + \sum_{v \in V_t^{\text{rec}}} |E_{jx}^{\text{com}}[v]| - \sum_{u \in V_t^{\text{sen}}} |E_{xl}^{\text{com}}[u]| \tag{4}$$

由于 $\boldsymbol{\theta}_{ix}^t = +\infty$ 之后 send 操作被禁止，所以无法继续更新，故之后 G_i 不可以向 G_x 转移出边，即使在 $t+\Delta t$ 时刻 $real(G_x)|_{t+\Delta t} \leq |E|/K$. 我们称这种均衡控制机制下的划分为 OnFlyP-R 划分.

实时控制机制是在划分时实时监控出边的定向交换，因此可以在数据加载的同时完成图数据划分，不会增加额外的划分开销. 然而，该机制缺乏对全图的统计信息，考虑到各任务实际划分进度的不同，使得 $\boldsymbol{\theta}_{ix}^t$ 被设置为无穷大的时间点具有随机性，难以保证最终划分效果. 例如，当任务 T_x 的划分进度较慢时，则 t 时刻子图 G_x 的迁出边较少，而其它任务迁移到 G_x 的出边较多，增大了 t 时刻 $\boldsymbol{\theta}_{ix}^t$ 被设置为无穷大的概率. 且在后续处理过程中，即使 G_x 继续迁移出边而导致 $real(G_x)|_{t+\Delta t} < |E|/K$，由于 G_i 端的 $\theta_{ix}^{t+\Delta t} = \boldsymbol{\theta}_{ix}^t$，故 G_i 仍不会向 G_x 迁移出边. 最终导致整体的出边迁移规模减少，限制了对通信边规模的优化效果. 以图 2 中的图 G 为例，图 4 展示了实时控制机制下的一个可能的处理流程.

子图状态信息表

	t_0时刻		t_2时刻		t_4时刻	
	$real(G_i)\|_{t_0}$	$\theta_i^{t_0}$	$real(G_i)\|_{t_2}$	$\theta_i^{t_2}$	$real(G_i)\|_{t_4}$	$\theta_i^{t_4}$
G_1	0	$(+\infty,2,2)$	5	$(+\infty,2,2)$	7	$(+\infty,2,2)$
G_2	0	$(2,+\infty,2)$	6	$(2,+\infty,2)$	6	$(2,+\infty,2)$
G_3	0	$(2,2,+\infty)$	0	$(2,+\infty,+\infty)$	3	$(+\infty,+\infty,+\infty)$

定向交换顺序表

t_1时刻			t_3时刻		
交换方向	交换边	$assert()$	交换方向	交换边	$assert()$
$G_1 \longrightarrow G_2$	{(4,5),(4,6)}	true			
$G_2 \longrightarrow G_1$	{(6,2),(6,3)}	true	$G_2 \longrightarrow G_3$	{(6,7),(6,8)}	true
$G_3 \longrightarrow G_2$	{(7,5),(7,6)}	true	$G_3 \longrightarrow G_1$	{(8,2),(8,3)}	true

图 4　实时控制机制处理流程示意图

在 t_0 时刻，各任务完成初始化，准备加载图数据，因此实时负载均为 0，阈值向量除 $\boldsymbol{\theta}_{ii}^t$ 之外均为初始值 2. 假设 t_1 时刻各子图之间产生 send 请求，则 $assert()$ 判定结果均为 true，允许执行. 在 t_2 时刻，以 G_1 为例，已经加载 5 条边{(1,2),(1,3),(3,4),(4,5),(4,6)}，其中{(4,5),(4,6)}迁移到 G_2，同时接收 G_2 发送的出边{(6,2),(6,3)}，故实时负载仍为 5. 需要注意的是，G_1 的 send 操作返回值为 true，因为此时 G_2 的负载为 4，小于负载均值(16/3). 但是 G_3 的 send 操作返回值为 false，因为接收 G_3 发送的出边后，G_2 的负载为 6，大于负载均值. 所以，在 t_2 时刻，$\boldsymbol{\theta}_{12}^{t_2}$ 不变，但 $\boldsymbol{\theta}_{32}^{t_2} = +\infty$. 类似地，$t_3$ 时刻，G_3 向 G_1 发送完出边{(8,2),(8,3)}后，G_1 的负载为 7，超过均值，故 $\boldsymbol{\theta}_{31}^{t_4} = +\infty$. 显然，只有执行 send 操作之后才可以更新 $\boldsymbol{\theta}_{ix}^t$，这种滞后性导致无法准确控制负载均衡. 在最坏情况下，其余 $(K-1)$ 个子图均需执行 send 之后，才可以完成阈值向量的更新. 假设每次 send 的出边规模达到最大出度 $\max d$，则负载偏斜因子为

$$\rho = 1 + \frac{(K-1)\max d}{|E|/K}.$$

5.2　最小对称矩阵控制

本节介绍一种最小对称矩阵控制机制，通过两次遍历图数据，避免了实时控制机制下划分效果的不确定性. 两次遍历分别完成如下工作：(1) 加载数据并建立交换矩阵 $\overline{\boldsymbol{M}}$；(2) 在 $\overline{\boldsymbol{M}}$ 的指导下，完成划分. 该机制下的划分称为 OnFlyP-M，算法 2 描述了划分过程. 其中，第 20 行的“exchange edges”过程，除 $assert()$ 判定机制，均与算法 1 的第 8～14 行相同.

算法 2.　最小对称矩阵控制的 OnFlyP 划分.

输入：待划分图 G，任务数目 K，图顶点数 $|V|$

输出：图划分结果 $\{G_i\}$

OnFlyP-$M(G,K,|V|)$

1. receive R and $Split_i$ from $Master$;
2. initialize M by $m_{ij}=0$; //初始化边交换矩阵
3. WHILE $Split_i \neq \varnothing$ DO
4. $adj(v) = loadGraph(Split_i)$;//加载邻接表
5. $\{(T_x, E_{ix}^{\text{com}}[v])\} = get(getEdges(adj(v)), R)$;
6. IF $|E_{ix}^{\text{com}}[v]| \geq 2$ and $i \neq j$ DO
7. $m_{ix} += |E_{ix}^{\text{com}}[v]|$;
8. END IF
9. $save(adj(v))$; //保存邻接表数据至本地
10. END WHILE
11. initialize $\overline{\boldsymbol{M}}$ by $\overline{m}_{ij}=0$; //初始化最小对称矩阵
12. FOR $\forall m_{ij} \in M$ DO

13. IF $i \neq j$ DO
14. $\bar{m}_{ij} = \bar{m}_{ji} = \min\{m_{ij}, m_{ji}\}$;
15. END IF
16. END FOR
17. initialize *graph*; //初始化图数据的本地读取句柄
18. WHILE *graph.hasNext*() DO
19. $adj(v) = graph.getNext()$;//读取本地邻接表数据
20. exchange edges;
21. END WHILE
22. $V_i = V_i \cup V^b$, $E_i = E_i \cup E^{syn} \cup E^{in}$;
23. RETURN $G_i = (V_i, E_i)$;

M 是一个 K 阶方阵，其中 m_{ij} 的值表示子图 G_i 与 G_j 之间的满足 $|E_{ij}^{com}[v]| \geq 2$ 约束的通信边规模，即 E_{ij}^{com}. 矩阵建立完毕后，采用最小交换原则得到对称的最小控制矩阵 $\bar{\boldsymbol{M}}$. 在划分阶段，如果 G_i 向 G_j 迁移的总出边规模小于 $\bar{m}_{ij}$，则 *assert*() 为 true，否则为 false，禁止 G_i 向 G_j 继续迁移出边.

OnFlyP-M 划分是建立在 Range 划分基础上，而 Range 划分本身可以保证各任务出边规模的近似均衡，故对 OnFlyP-M，保证每个任务中迁入和迁出边的规模相等(对称交换)，即可实现负载均衡.

图 5 分析了 OnFlyP-M 算法的适用性. 其中 M 是图 G 所对应的 K 阶方阵. 这里，由于 $|E_{23}^{com}[v_5]| = |\{(5,7)\}| < 2$，不计入统计信息，故 $m_{23} = 2$. 此外，根据最小化原则，在最终的控制矩阵 $\bar{\boldsymbol{M}}$ 中，有 $\bar{m}_{13} = \bar{m}_{31} = \min\{m_{13}, m_{31}\} = 0$. 下面我们简单讨论 OnFlyP-$M$ 算法的适用性. 根据出边在子图之间的分布特征，我们可以归纳出 3 种极端分布：

(1) 均匀分布，指向各个子图的出边数目相同. 这种分布下，对于控制矩阵 $\bar{\boldsymbol{M}}$，显然有 $\bar{m}_{ij} = \bar{m}_{ji} = \min\{m_{ij}, m_{ji}\} = m_{ij} = m_{ji}$. 另一方面，对于无负载均衡控制的 OnFlyP 算法，等价于“最大控制矩阵”，$\bar{m}_{ij} = \bar{m}_{ji} = \max\{m_{ij}, m_{ji}\} = m_{ij} = m_{ji}$. 因此，在均匀分布下，OnFlyP-$M$ 可以在保证负载均衡的前提下，最大化参与迁移的出边规模，从而减少通信边规模.

(2) 对角分布，各任务加载的出边均指向子图内部的顶点，子图之间没有交互边. 此时已经达到最优划分，控制矩阵为 0 矩阵，不必执行出边迁移.

(3) 环形分布，当子图数目大于 2 时，显然有 $\bar{m}_{ij} = \bar{m}_{ji} = \min\{m_{ij}, m_{ji}\} = 0$，控制矩阵为 0 矩阵，参与交换的出边规模为 0，无法优化通信边规模.

根据第 6 节的测试结果，上述 3 种极端分布通常是不存在的. 真实图中的出边分布介于 3 者之间，OnFlyP-M 在实际应用中可以达到较好的优化效果.

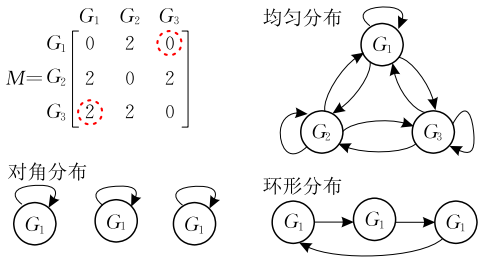


图 5 OnFlyP-M 算法适用性分析

6 实验结果与分析

6.1 实验方案与实验环境

由于 OnFlyP 算法是利用原始图的局部性，实现划分效果和划分效率的综合收益，因此本文首先测试不同真实图的局部性(即聚簇系数 C，见 6.3 小节)，然后从通信边比例 $\lambda = |E^{com}|/|E|$ 和负载偏斜因子 ρ(第 3.2 小节)两个方面测试不同划分算法在不同数据集上的划分效果(见 6.4 和 6.5 小节)，之后以划分时的网络通信开销作为效率衡量标准测试了各算法的特征表现(见 6.6 小节)，最后通过运行 PageRank 算法验证 OnFlyP 算法的实际运行效果(见 6.7 小节).

参与对比测试的方法包括：Hash(分布式在线划分)、LDG(集中式在线划分)和 OnFlyP(分布式在线划分). 其中，OnFlyP 算法按照均衡控制策略的不同，分为 OnFlyP-B 即无均衡控制策略(见 4.2.2 小节)，用于 6.3 小节中测试真实图的局部性；OnFlyP-R，实时控制策略(见 5.1 小节)；OnFlyP-M，最小对称矩阵控制策略(见 5.2 小节).

本文在类 Pregel 的 BC-BSP-1.0 系统上实现了 Hash 划分和 OnFlyP 划分. 输入数据采用邻接表组织，存放在分布式文件系统 HDFS(版本：Hadoop-0.20.2). HDFS 的文件块大小为默认的 64 MB. 对于 Hash 划分，实现方式为将原始数据按照文件大小从逻辑上等分为 K 份(K 为分布式任务数目，任务编号为 1～K)，然后各任务并行加载数据. 针对每条邻接表记录，根据其源顶点的 *HashCode* 值，按照 *HashCode*() mod $K+1$ 计算其所属的任务编号，并发送到对应的任务. 而 LDG 为集中式算法，因此输入数据和划分后的子图均位于本地文件系统. 对于 OnFlyP 算法，先采用 Range 划分，将 HDFS 上的输入数据按照文件大小等分为 K 份，然后运行算法 1 完成划分，具体的执行逻辑随负载均衡控制策略而变化(见 5.1 节和 5.2 小节的描述).

6.2 集群配置与数据集

实验集群由 26 台节点构成，使用一台 Gigabit 以太网交换机连接，每个计算节点配置酷睿 i3-2100 双核处理器，8 GB 内存，7200 RPM 硬盘，安装 Red Hat Enterprise Linux 6.1 操作系统和 JDK 1.6.0 编程环境. BC-BSP 系统中每个任务，即每个 JVM，内存大小设定为 3 GB. 全部实验在 4 个真实图数据集上完成，包括 Web 数据：Ber-Stan① 与 Wiki②，和社交网络数据：S-LJ③ 与 Twitter[35]，具体描述如表 1 所示. 其中，$|V|$ 表示图顶点的数目，而 $|E|$ 表示出边数目. Ber-Stan 是 BFS 生成图，其余 3 个数据集的性质未知.

表 1　真实数据集描述

数据集	$\|V\|$ (10^4)	$\|E\|$ (10^6)	图性质	文件大小
Ber-Stan	68	8	BFS 图	99 MB
S-LJ	484	69	未知	553 MB
Wiki	571	130	未知	1.02 GB
Twitter	4170	1470	未知	12.9 GB

6.3 真实图的局部性分析

我们在 4 个真实数据集上测试 OnFlyP-*B* 划分方法，以证明真实图具有良好的局部性以及对 OnFly-*B* 算法的影响. 表 2 展示了 $K=10$ 时的测试结果. 其中 C 为第 4.1 节中的聚簇系数，$\hat{\lambda}$ 是采用 4.2.3 小节中定理 1 得到的 BFS 图的 λ 上界值.

表 2　OnFlyP-*B* 与 *C* 的关系（$K=10$）

数据/指标	C	$\hat{\lambda}$	λ	ρ
Ber-Stan	10.50	0.27	0.03	1.21
S-LJ	2.50	0.21	0.19	1.25
Wiki	2.37	0.13	0.14	1.12
Twitter	8.62	0.13	0.07	2.89

显然，C 越大，局部性越佳，实际 λ 值越低. 然而，由于 OnFlyP-*B* 未采取负载均衡控制策略，所以负载偏斜因子 ρ 的值均比较大. 此外，非 BFS 生成图的 C 均小于 BFS 生成图 Ber-Stan，故 λ 值高于 Ber-Stan. 特别地，对于 C 值较小的 Wiki，有 λ 大于上界值 $\hat{\lambda}$. 这是因为 Wiki 图的局部性相对较差，不能满足回溯边集目的顶点的严格连续分布，破坏了 4.2.3 小节中引理 2 的前提条件，导致冗余备份顶点的规模大于理论上限值 $2|V|$，因此实际 λ 值比理论估计值 $\hat{\lambda}$ 偏大.

6.4 划分效果分析

本小节介绍 Hash，LDG，OnFlyP-*R* 和 OnFlyP-*M* 4 种算法在不同数据集上的划分效果. 本小节实验均设置 $K=20$. 图 6 展示了 λ 值.

显然，Hash 划分会导致 90% 以上的边成为通信边，这会在后续迭代计算中引入大量的通信开销. 而集中式 LDG 和分布式 OnFlyP-*R* 则可以将 λ 值降低为 40% 至 80%，OnFlyP-*M* 进一步将该值降低为 13% 至 45%. 其中，除 Hash 外，其余 3 种划分算法对于原始图的局部性敏感. 特别地，对于 BFS 生成图 Ber-Stan，LDG 算法的通信边规模是 OnFlyP-*M* 算法的 2.6 倍；Hash 算法的通信边规模是 OnFlyP-*M* 算法的 7.25 倍.

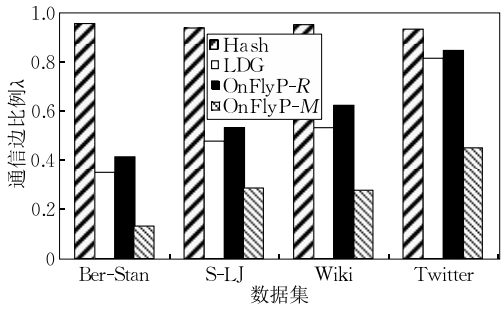


图 6　不同划分算法的 λ 值

图 7 则对比了各种划分算法的负载偏斜因子. 其中，Hash 划分由于无法考虑出边数目的均衡，因此在幂率图 Twitter 上，ρ 值较大，为 1.53. 此外，OnFlyP-*R* 由于分布式环境下的负载检测具有滞后性，导致负载偏斜因子略高于 1.0. 而 OnFlyP-*M* 的偏斜由 Range 划分决定，由于后者只能保证各子图负载近似均衡（见 4.2.1 小节），故 $\rho \approx 1.0$.

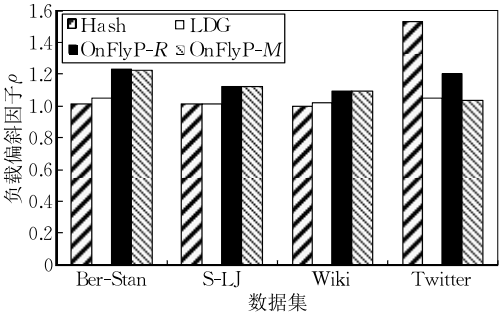


图 7　不同划分算法的 ρ 值

6.5 *K* 值对划分效果的影响

本组实验分别在 S-LJ 和 Twitter 数据集上测试了不同 K 值对各种划分算法的 λ 值的影响. 如图 8 和图 9 所示，随 K 值变化，OnFlyP-*M* 的 λ 始终最低.

① SNAP：Network datasets：Berkeley-Stanford web graph, http://snap.stanford.edu/data/web-BerkStan.html 2014, 6, 12

② Using the Wikipedia link, http://haselgrove.id.au/wikipedia.htm 2014, 6, 11

③ SNAP：Network datasets：LiveJournal social network, http://snap.stanford.edu/data/soc-LiveJournal1.html 2014, 6, 12

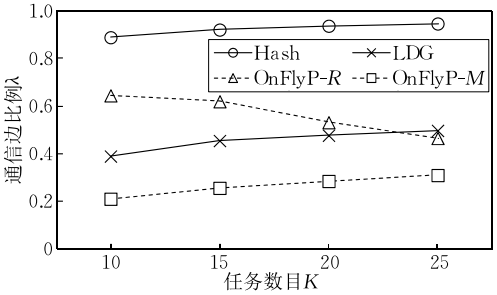


图 8 不同 K 值对 λ 的影响(S-LJ)

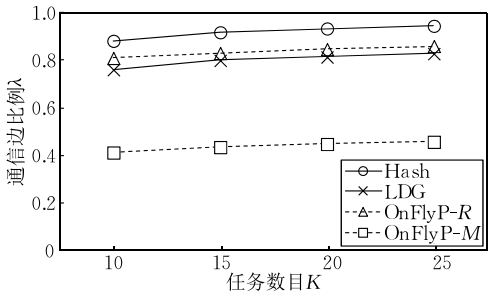


图 9 不同 K 值对 λ 的影响(Twitter)

特别地,对于 S-LJ 上的 OnFlyP-R 算法,随着 K 值增大,λ 值逐渐降低.这是由于分布式粒度的增大,延迟了 $\boldsymbol{\theta}_{ix}^{t}$ 被设置为无穷大的时间点,使得参与交换的出边规模增大,因此降低了 λ 值.理论上,当分布式任务的水桶效应较低时,OnFlyP-R 算法的 λ 值有可能小于 OnFlyP-M 算法.然而,真实图通常具有幂率偏斜特点,尤其是 Twitter,少数顶点关联了大部分出边,因此,在实际实验过程中,OnFlyP-R 算法的 λ 值均大于 OnFlyP-M 算法.

图 10 和图 11 则展示了不同 K 值对负载偏斜因子 ρ 的影响.LDG,OnFly-R 和 OnFly-M 这 3 种划分算法对于数据集和 K 值的变化不敏感.而对 Hash 划分,由于 Twitter 数据集中顶点的出边规模具有幂率偏斜特性,故难以保证出边数目的均衡,ρ 值较大.而 K 的不同取值,可能导致高出度顶点的分布发生随机变化,故 ρ 值随 K 的变化呈现随机波动.如图 11 所示,当 $K=10,15,20,25$ 时,ρ 值为 1.51,1.26,1.53,1.24.

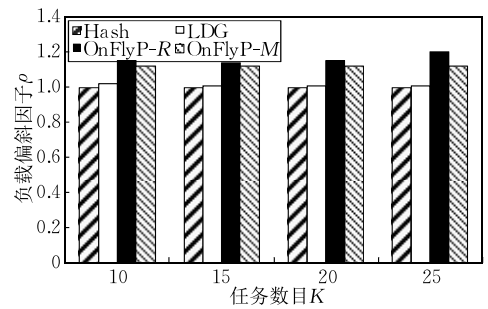


图 10 不同 K 值对 ρ 的影响(S-LJ)

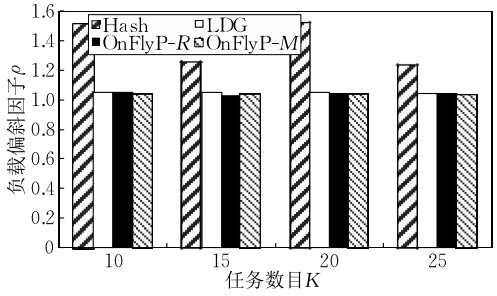


图 11 不同 K 值对 ρ 的影响(Twitter)

6.6 网络通信开销分析

对于 Hash,OnFlyP-R 和 OnFlyP-M 划分,各任务加载的数据不一定驻留本地,例如任务 T_i 加载的数据,经过 Hash 运算后,需要通过网络发送给任务 T_j 维护.而对 OnFlyP-R 和 OnFlyP-M 划分,其加载的出边也可能需要通过网络发送给任务 T_j.我们称该过程为 shuffle 阶段.显然,shuffle 阶段会引入网络通信开销,且主要与参与 shuffle 的出边数目 $|ShuffleEdge|$ 成正比.为便于对比不同数据集在划分时的网络开销,我们定义 $ratio=|ShuffleEdge|/|E|$,作为衡量指标.由于 LDG 属于集中式划分算法,不计入对比分析.图 12 显示了不同划分算法在各种真实数据集上的测试结果($K=10$).Hash 划分由于不考虑局部性,大量出边被迁移,因此 $ratio$ 值较高.而 OnFlyP-R 和 OnFlyP-M 仅需要迁移部分出边数据,且 OnFlyP-R 由于实时负载监控导致 θ_{ix}^{t} 提前被设置为无穷大,故迁移边规模小于 OnFlyP-M.

图 13 和图 14 分析了任务数目(子图数目)K 对 $ratio$ 的影响.对于 OnFlyP-R 和 OnFlyP-M 算法,

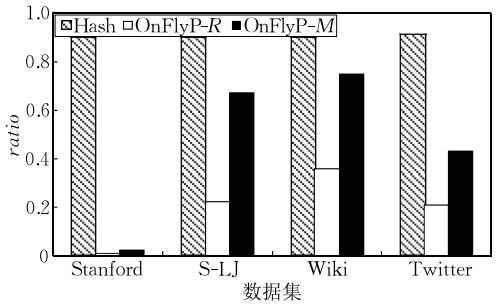


图 12 不同划分算法的 $ratio$ 值

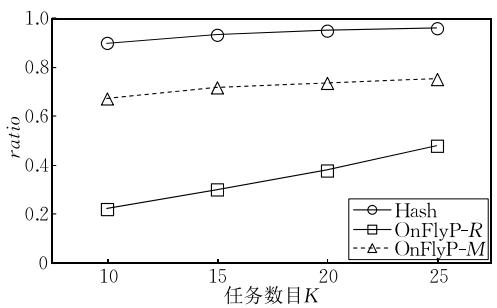


图 13 K 值对 $ratio$ 值的影响(S-LJ)

图 8 与图 9 和图 13 与图 14 的对比进一步表明 *ratio* 值与 λ 值成反比，即参与迁移的出边越多，对通信的优化效果越明显.

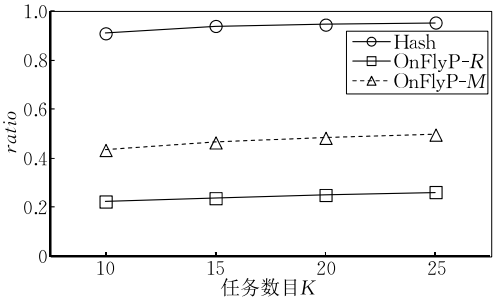


图 14　K 值对 *ratio* 值的影响(Twitter)

6.7　不同划分算法对实际迭代性能的影响

我们在 BC-BSP 系统上运行 PageRank 算法，来验证各种划分方法对于提高实际迭代处理性能的效果. 本实验采用 Twitter 数据，$K=25$，PageRank 运行中平均每次迭代的计算时间如表 3 所示. 其中划分时间包括数据加载开销、shuffle 阶段的网络开销(6.6 小节)和划分后的数据本地组织与存储开销三部分. 特别地，由于 LDG 是集中式算法，本文首先在一台物理机上对数据完成 LDG 划分，然后将划分后的子图分别上传到 HDFS，最后由 BC-BSP 系统的分布式任务按照 Range 划分，分别加载到本地并完成迭代计算. 这里，每个子图是一个单独的文件，由一个任务独立加载. 因此，LDG 的划分时间，仅包含单机处理时间.

表 3　不同划分算法在 BC-BSP 系统上的运行效果

划分算法	划分时间/s	迭代时间/s	提升比例
Hash	195	70.4	0.00
LDG	494	57.7	0.18
OnFlyP-*R*	132	62.4	0.11
OnFlyP-*M*	168	51.2	0.27

在 BC-BSP 系统中，各任务加载的数据均是通过 *Split* 方式按照字节大小均衡切分得到，因此 Hash，OnFlyP-*R* 和 OnFlyP-*M* 的加载时间相同. 根据 6.6 小节的分析，Hash 划分由于需要大量交换图数据，网络开销增大，划分时间较高. OnFlyP-*M* 由于需要扫面两次数据且 shuffle 的数据规模较多，所以划分时间高于 OnFlyP-*R*. 与上述 3 种方法对比，受单机处理能力限制，集中式 LDG 的划分时间最高. 另一方面，从每步迭代时间来看，LDG 划分下的迭代性能比 Hash 快 18%，与文献[11]所报告的结果相近；而 OnFlyP-*M* 划分下的迭代性能比 Hash 划分快 27%. 从实验结果可知，OnFlyP-*R* 和 OnFlyP-*M*，分别在提高执行效率和降低通信边规模方面有较好的效果. 对高频迭代算法，可选择 OnFlyP-*M*，虽然执行效率略低，但是累加的迭代收益，会使整体性能优于 OnFlyP-*R*；反之，应该选择 OnFlyP-*R*.

7　结论和进一步工作

分布式大图处理系统的研究与广泛应用，对图划分算法的效果和执行效率提出了严峻挑战. 本文利用真实世界中图的局部性，在两种指标上获得了综合收益. 具体地，本文首先提出聚簇系数概念，定量分析了顶点分布的局部特性，并在大量真实图上验证了结论的可靠性. 其次，以真实图的局部性为基础，设计了高效的分布式在线图划分算法 OnFlyP，支持实时控制(OnFlyP-*R*)和最小对称矩阵控制(OnFlyP-*M*)两种负载均衡策略，可由用户根据实际应用需求自行选择. 大量真实图的实验结果和在 BC-BSP 系统上的实际迭代性能表明，OnFlyP 在执行时间和划分效果方面，均优于现有的在线划分算法 Hash 和 LDG. 其中，OnFlyP-*R* 的执行时间比 OnFlyP-*M* 的执行时间略低，但后者具有较低的通信边规模，对于高频迭代应用中网络通信开销的优化效果显著.

根据 OnFlyP-*R* 和 OnFlyP-*M* 的实验对比结果，显然参与 shuffle 的出边规模与通信收益成正比. 然而，目前的最小对阵矩阵控制机制，在面对环形分布或近环形分布(5.2 小节)时，会严重降低 shuffle 的出边规模. 实际上，最小对称矩阵的计算过程是针对仅有 2 个顶点构成的环路，可以扩展到环形分布中的多顶点环路. 因此，下一步工作将研究环形分布下的控制矩阵计算方法，以增大出边交换的总规模，进一步降低通信边比例，提高 OnFlyP-*M* 算法的适用性.

参　考　文　献

[1] Yu Ge, Gu Yu, Bao Yu-Bin, Wang Zhi-Gang. Large scale graph data processing on cloud computing environments. Chinese Journal of Computers, 2011, 34(10): 1753-1767(in Chinese)
(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)

[2] Li Xian-Tong, Li Jian-Zhong, Gao Hong. An efficient frequent subgraph mining algorithm. Journal of Software, 2007, 18(10): 2469-2480(in Chinese)

(李先通，李建中，高宏. 一种高效频繁子图挖掘算法. 软件学报，2007，18(10)：2469-2480)

[3] Malewicz G，Austern M H，Bik A J C，et al. Pregel：A system for large-scale graph processing//Proceedings of the 2010 International Conference on Management of Data. Indianapolis，USA，2010：135-145

[4] Shao B，Wang H，Li Y. Trinity：A distributed graph engine on a memory cloud//Proceedings of the ACM SIGMOD International Conference on Management of Data. New York，USA，2013：505-516

[5] Salihoglu S，Widom J. GPS：A graph processing system//Proceedings of the 25th International Conference on Scientific and Statistical Database Management. Baltimore，USA，2013. Article No. 22

[6] Avery Ching. Giraph：Large-scale graph processing infrastructure on Hadoop//Proceedings of the Hadoop Summit. Santa Clara，USA，2011

[7] Zaharia M，Chowdhury M，Franklin M J，et al. Spark：Cluster computing with working sets//Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing. Boston，USA，2010：10-10

[8] Bao Y，Wang Z，Gu Y，et al. BC-BSP：A BSP-based parallel iterative processing system for big data on cloud architecture //Hong B，Meng X，Chen L，et al，eds. Database Systems for Advanced Applications. Germany：Springer Berlin Heidelberg，2013：31-45

[9] Pan Wei，Li Zhan-Huai，Wu Sai，Chen Qun. Evaluating large graph processing in MapReduce based on message passing. Chinese Journal of Computers，2011，34(10)：1768-1784(in Chinese)
(潘巍，李战怀，伍赛，陈群. 基于消息传递机制的 MapReduce 图算法研究. 计算机学报，2011，34(10)：1768-1784)

[10] Karypis G，Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing，1998，20(1)：359-392

[11] Stanton I，Kliot G. Streaming graph partitioning for large distributed graphs//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Beijing，China，2012：1222-1230

[12] Tsourakakis C E，Gkantsidis C，Radunovic B，Vojnovic M. Fennel：Streaming graph partitioning for massive scale graphs//Proceedings of the 7th ACM International Conference on Web Search and Data Mining. New York，USA，2014：333-342

[13] Najork M，Wiener J L. Breadth-first search crawling yields high-quality pages//Proceedings of the 10th International World Wide Web Conference. Hong Kong，China，2001：114-118

[14] Kernighan B W，Lin S. An efficient heuristic procedure for partitioning graphs. Bell Systems Journal，1970，49(2)：291-308

[15] Fiduccia C M，Mattheyses R M. A linear-time heuristic for improving network partitions//Proceedings of the 19th Design Automation Conference. Las Vegas，USA，1982：174-181

[16] Hendrickson B，Leland R. The Chaco User's Guide：Version 2.0. New Mexico：Sandia National Laboratories，Technical Report：1994-2692，1994

[17] Pellegrini F，Roman J. Scotch：A software package for static mapping by dual recursive bipartitioning of process and architecture graphs//Liddell H，Colbrook A，Hertzberger B，Sloot P eds. High-Performance Computing and Networking. Germany：Springer Berlin Heidelberg，1996：493-498

[18] Karypis G，Kumar V. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. Journal of Parallel and Distributed Computing，1998，48(1)：71-95

[19] Chevalier C，Pellegrini F. PT-scotch：A tool for efficient parallel graph ordering. Parallel Computing，2008，34(6-8)：318-331

[20] Wang L，Xiao Y，Shao B，Wang H. How to partition a billion-node graph//Proceedings of the 30rd International Conference on Data Engineering. Chicago，USA，2014：61-72

[21] Ugander J，Backstrom L. Balanced label propagation for partitioning massive graphs//Proceedings of the 6th ACM International Conference on Web Search and Data Mining. Rome，Italy，2013：507-516

[22] Rahimian F，Payberah A H，Girdzijauskas S，Jelasity M，Haridi S. JA-BE-JA：A distributed algorithm for balanced graph partitioning//Proceedings of the 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems. Philadelphia，USA，2013：51-60

[23] Slota G M，Madduri K，Rajamanickam S. PuLP：Scalable multi-objective multi-constraint partitioning for small-world networks//Proceedings of the 2014 IEEE International Conference on Big Data. Washington，USA，2014：481-490

[24] Nishimura J，Ugander J. Restreaming graph partitioning：Simple versatile algorithms for advanced balancing//Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Chicago，USA，2013：1106-1114

[25] Gonzalez J E，Low Y，Gu H，et al. Powergraph：Distributed graph-parallel computation on natural graphs//Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation. Hollywood，USA，2012：17-30

[26] Jain N，Liao G，Willke T L. Graphbuilder：Scalable graph ETL framework//Proceedings of the 1st International Workshop on Graph Data Management Experiences and Systems. New York，USA，2013：4

[27] Gonzalez J E，Xin R S，Dave A，et al. Graphx：Graph processing in a distributed dataflow framework//Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation. Broomfield，USA，2014：599-613

[28] Zhao Y，Yoshigoe K，Xie M，et al. LightGraph：Lighten communication in distributed graph-parallel processing//Proceedings of the 2014 IEEE International Congress on Big Data. Anchorage，USA，2014：717-724

[29] Chen R，Shi J，Chen Y，Chen H. Powerlyra：Differentiated graph computation and partitioning on skewed graphs. Shanghai Jiao Tong University，China：Technical Report：IPADSTR-2013-001，2013

[30] Wang Z, Bao Y, Gu Y, et al. A BSP-Based parallel iterative processing system with multiple partition strategies for big graphs//Proceedings of the 2013 IEEE International Congress on Big Data. Santa Clara, USA, 2013: 173-180

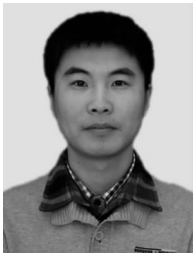
[31] Khayyat Z, Awara K, Alonazi A, et al. Mizan: A system for dynamic load balancing in large-scale graph processing//Proceedings of the 8th ACM European Conference on Computer Systems. Prague, Czech Republic, 2013: 169-182

[32] Bao N T, Suzumura T. Towards highly scalable pregel-based graph processing platform with X10//Proceedings of the 22nd International Conference on World Wide Web Companion. Rio de Janeiro, Brazil, 2013: 501-508

[33] Shang Z, Yu J X. Catch the wind: Graph workload balancing on cloud//Proceedings of the 29th IEEE International Conference on Data Engineering. Brisbane, Australia, 2013: 553-564

[34] Wu Xin-Dong, Li Yi, Li Lei. Influence analysis of online social networks. Chinese Journal of Computers, 2014, 37(4): 735-752(in Chinese)
(吴信东，李毅，李磊. 在线社交网络影响力分析. 计算机学报，2014，37(4)：735-752)

[35] Kwak H, Lee C, Park H, Moon S. What is Twitter, a social network or a news media?//Proceedings of the 19th International Conference on World Wide Web Conference. Raleigh, USA, 2010: 591-600



WANG Zhi-Gang, born in 1987, Ph. D. candidate. His major research interests include database system and cloud computing.

GU Yu, born in 1981, Ph. D., associate professor. His major research interests include graph data management and spatial data management, etc.

BAO Yu-Bin, born in 1968, Ph. D., professor. His major research interests include massive data management, cloud computing, etc.

YU Ge, born in 1962, Ph. D., professor. His major research interests include database theory and technology, distributed and parallel systems, etc.

Background

Graphs are of growing importance in modeling complicated structures and widely applied in multiple scenarios, ranging from the Internet to social networks, biological networks and geographic information systems. For distributed graph computations, the key challenge derives from the need to distribute it over a cluster, while satisfying three objectives: load balance, minimizing the number of communication edges and prominent executing efficiency. That is a well-known NP-Complete problem and there are two different research paradigms, namely offline or online partition methods, attempting to tackle it approximately in items of reducing communication edges and improving executing efficiency respectively. Neither of them can hold the requirements of the two aforementioned aspects simultaneously.

In practice, for real-world graphs, the vertex distribution has an inherent locality, which can be leveraged to design a prominent graph partition method for iterative computations. In this paper, we incur the concept of "cluster coefficient" to analyze this feature, and then propose an online distributed partition algorithm (OnFlyP) based on a directional edge-exchange model, which holds the three objectives of graph partition: being executed during loading graph with high efficiency; greatly reducing the communication edge scale by exchanging edges; resorting to two policies, real-time control and minimum symmetric matrix, to implement the load balance among subgraphs. Note that the two policies respectively focus on the high-efficiency and the effect of reducing the communication edge scale, and can be chosen flexibly according to real applications. Extensive experiments on various real-world graphs validate the effect of OnFlyP. In the future, we plan to study more sophisticated policies to address the issue of load balance, and enhance the effect of reducing the communication edge size.

This research was supported by the National Basic Research Program (973 Program) of China under Grant No. 2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61272179, 61472071, 61173028, the Fundamental Research Funds for the Central Universities under Grant No. N120816001, the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No. 20120042110028, and the Ministry of Education of China and China Mobile Foundation under Grant No. MCM20125021.

Large scale graph processing is one of core technologies in the era of big data. And a prominent graph partition method is a key precondition for high-performance iterative graph computations. The authors of this paper have been engaged in graph data management and cloud computing. They have published some papers in this area.