基于邻域的大规模图数据动态分割算法①

张晓媛[1,3], 张 珩[2,3], 翟 健[1]

[1](中国科学院软件研究所 互联网软件技术实验室, 北京 100190)
[2](中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
[3](中国科学院大学, 北京 100190)

摘 要: 随着图数据的规模日益增大,出现大量以动态图数据为基础的分布式处理需求, 划分问题在动态图数据分布式处理领域尤为重要. 对大规模动态图数据上的划分问题进行研究, 根据图结构性质及动态图特点, 提出并实现基于邻域的动态图分割算法. 算法分为静态切分和动态调整两个阶段, 其中基于割边算法整合现有最优化策略提出了大规模图数据的静态切割算法. 在优化后的静态切割算法的基础上, 根据图数据的动态扩张的特性提出动态分割算法. 根据迁移顶点所达到的最小负载值进行顶点迁移, 并在此基础上进行性能及割边控制优化操作. 最后, 改进算法在各类图数据集上进行了验证, 验证的结果显示在平衡度和割边等指标上优化后的算法效果显著, 提高了划分的合理性, 并且在保证割边不增加的情况下提高了图分割的平衡度.

关键词: 图数据; 动态图; 图分割; 负载均衡

Large Scale Dynamic Graph Partitioning Based on Neighborhood Algorithm

ZHANG Xiao-Yuan[1,3], ZHANG Heng[2,3] , ZHAI Jian[1]

[1](Laboratory for Internet Software Technology, Institute of Software, the Chinese Academy of Sciences, Beijing 100190, China)
[2](National Engineering Research Center of Fundamental Software, Institute of Software, the Chinese Academy of Sciences, Beijing 100190, China)
[3](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: To solve the problem of graph data distributed processing, graph partitioning is more and more important. This paper studies the method of partitioning a large-scale dynamic graph. According to the characteristics of graph structure and dynamic graph, a dynamic graph partitioning algorithm based on neighborhood is proposed and implemented. The algorithm consists of two stages, static partition and dynamic adjustment. A static graph partitioning combines the cut-edges and the existing partition tools to get an optimized method. The dynamic adjustment focuses on the transferring of vertexes to its neighborhood. It calculates the value of vertexes transferring load and optimize the operations. Experiments on a series of common graph datasets show significant effect on the indexes of balance and cut-edges, which means the partition is reasonable and balanced.

Key words: graph; dynamic graph; graph partition; load balance

1 引言

图, 作为基本的一种数据结构, 可以抽象表达很多场景, 如社交网络、道路交通、P2P 网络、集群负载网络等. 图论作为数学的一个分支, 涉及多种经典问题及算法, 而这些算法在现实场景中的应用也非常广泛, 如可以使用单源最短路 Dijkstra[20]算法求解计算机网络中路由器传播消息的最短路径; 使用最大流最小割算法在铁路网络中求解最大通量; 使用图的拓扑排序算法来解决操作系统的任务调度问题等.

日益普及的社交网络也可以抽象成为图数据来进行分析处理. 在国外, 全球最大的社交网络 Facebook, 2015 年 8 月宣布该网站达到一个重要的里程碑: 日登

① 基金项目:中国科学院先导专项(XDA06010600);国家自然科学基金(61303163,91318301)
收稿时间:2016-01-18;收到修改稿时间:2016-02-25 [doi:10.15888/j.cnki.csa.005338]

陆用户已经突破 10 亿. 在国内, 2015 微信用户数据报告表示在 2015 年第一季度末, 微信的每月活跃用户数已达到 5.49 亿, 55.2%的微信用户每天打开微信超过 10 次. 微博在微信的打压下略显劣势, 但是 CNNIC 的报告仍显示, 在 2014 年上半年, 43.6%的网民使用过微博. 在社交网络中人与人的关系就构成了一张复杂的关系图, 人们的日常生活不断完善并丰富这张关系图. 这些社交关系图有数十亿的顶点, 并且实时不断更新扩张.

另外, 大规模图数据处理的研究对于机器学习、数据挖掘等技术的发展和提升非常有帮助. 语音识别、人工智能、深度学习等领域涉及到的算法很大一部分都是迭代图算法, 如基于树的结构有决策树的相关算法, 随机森林、C4.5、CART 算法等; 基于图相关的算法有搜索中计算页面权重的 PageRank 算法; 基于迭代的神经网络算法等. 由于训练这些算法的数据量非常大, 单机版的图算法已然不能满足业界的需求. 因此研究开发分布式图数据系统非常有必要.

2 相关工作

2.1 图计算的发展

在大规模数据处理需求下, 云计算应运而生. 云计算对大规模数据的处理具有一系列的优势, 对海量数据进行存储和维护, 拥有强大的分布式并行处理能力以及良好的可伸展性和灵活性[1]. 在大规模图数据处理的需求下, 单机已经无法满足, 因此将云计算应用到大规模图数据处理中成为一个必然的趋势.

如何分布式化图数据成为一个难点, 然而, 分布式图数据处理需要图数据分割成子图存储到集群中. 传统的分布式存储结构有 HDFS[2]、BigTable[3]等分布式文件系统, 这些存储系统虽然可以存储图数据, 但是对于图数据的各种操作支持不够友好, 不能很好的根据图的性质来存储图数据. 另外, 这些传统的分布式存储系统只是简单地按照行来分割图数据, 并没有考虑到图本身的结构. 因此, 研究者开发了一系列的图数据处理框架, 如 GraphX[5]是基于 Spark 的一套提供图算法解决方案的 API, 可以高效完成图计算的作业; GraphLab[6]通过将算法过程抽象成 Gather、Apply、Scatter 三个过程来对数据进行迭代求解. 另外, 在大规模图数据处理的需求下, 诞生了各种各样的分布式图数据库, 如知名的图数据库 Neo4J[4], 通过对顶点和边建立的索引, 它可以有效快速地处理图数据的查询; Google 发布的 Pregel[7], 是基于 BSP 的分布式图计算框架, 从顶点的角度来思考解决问题; 微软发布使用超图作为基础模型的 Trinity[8], 可以提供批量的查询和后台计算工作并支持同步和异步. 这些分布式图数据系统都有自己一整套的数据分割、存储方案, 除了可以较好地完成一些经典图算法外, 它们作为数据库系统, 也继承了数据库的一些传统特点, 比如事务、一致性等.

2.2 图数据分割算法

在大规模图数据的处理中, 单台机器无法承受所有的图数据, 因此需要使用集群处理大规模的图数据, 而使用集群处理图数据就涉及到子图与子图之间的通信. 如果划分子图不当, 比如子图数据规模相差很大时, 分布式节点的负载会很不均衡; 另外, 如果子图之间的点的关联性较低, 则会导致机器与机器之间通信开销很大; 这些都会影响图数据的处理效率. 因此, 需要使用分割算法来划分子图, 确保集群间的负载均衡、子图内连通性较强并且分割的时间复杂度较低. 由于图的分割是 NP 完全问题[9], 因此工作者主要从优化和启发式的角度来解决这个问题.

2.2.1 启发式划分算法

Kernighan 和 Lin[10]提出了启发式解决图划分问题的 KL 算法, 使用了迭代二分的思想, 先对图进行二路切分, 然后不断地在两个子图之间交换顶点来达到更好的分割结果, 如果需要多个分割子图, 则迭代地调用该算法, 直到分割准确度到达一定的阈值为止. 由于是不断地迭代, 时间复杂度较高. 在 KL 算法的基础上, Fiduccia[11]提出了启发式 FM 算法, 相比较于 KL 算法, FM 算法在交换子图顶点的时候只选取一个最好的顶点来进行交换, 这样使得复杂度大大降低, 从 KL 的平方复杂度降低到线性复杂度.

2.2.2 多层划分算法

Kumar 等人[12,13]提出了多层图划分的思想, 主要由三个阶段构成: 图的粗糙化、简单分割以及反粗糙化, 这三个阶段缺一不可, 共同完成了从大规模图数据到简单小图再重新恢复的过程. 粗糙化阶段的一个常见的做法是把大图通过减少顶点和边的个数来构造成一个小图; 在这个小图的基础上运行 KL/FM 等的分割算法; 在反粗糙化阶段采用一定的技术进行图数据的还原. Metis[14] 是一种经典的多层划分算法, 它在二切分的基础上, 还支持了多层次切分和 K 路切分的功能.

2.2.3 动态分割算法

静态分割算法主要适用于不变化的图数据, 分割后主要是修改点边信息, 不会增加或者删掉某个点或者边, 因此, 图数据划分一次即可. 比如铁道网络的图数据, 变更率较低. 然而, 由于大规模图数据在不断变化中, 随时都可能有增删操作, 而静态分割算法适用于初始图数据分割, 对于后续的关系增删操作不予考虑. 例如社交网络的关系图数据, 用户可以删掉任意一个好友或者添加一个好友, 这样的操作对实时性要求比较高, 如果重新使用静态分割方法就会造成大规模的数据迁移, 造成了大量的通信开销. 因此, 动态分割算法要考虑到增删操作的复杂度, 更好地适应不断变化的环境. Mizan[15]动态图划分算法是基于 BSP 模型的, 它会计算每个节点的负载值, 如果该值大于定义的值的大小, 则会被标记为不平衡节点, 然后来转移不平衡顶点来达到各个节点之间的负载均衡. GPS[16]算法类似地也是使用 BSP 模型, 在划分完静态图之后根据变化来重划分动态图, 根据一定的策略来迁移顶点. 两者都是基于超步的动态迁移顶点来达到动态划分的结果.

从以上的算法可以看出, 动态分割算法的要点就是计算负载和选择转移的顶点. 如果可以实时计算负载并挑选转移顶点, 则可以节省通信量, 提升切割效率. 因此, 需要一个分割算法可以实时计算负载来满足各种切割数据的需求.

3 基于邻域的动态图分割算法

本文根据图结构的性质, 计算转移到邻近子图的代价来考虑是否要转移顶点来达到缩小转移代价的目的, 由此提出了基于邻域的动态图分割算法, 即 NDA 算法(Neighborhood based Dynamic Algorithm). 基于邻域的动态图算法分为两个阶段: 静态切分阶段, 即 NDA-S(Static Partition of NDA); 动态调整阶段, 即 NDA-A(Adjustment of NDA).

给定图 $G=(V,E)$, 其中 $|V|$ 表示顶点的个数, $|E|$ 表示边的个数, 划分为 M 个子图. 在图数据量比较小的情况下, $|E|$ 数量较小, 服务器单节点能够存储, 不需要划分到多个节点时, 可以把所有数据都放在同一个节点上. 当 $|E|$ 达到一定规模时, 单个节点无法承载这么多的边, 则将原图 G 切分为 M 个子图, 每个顶点只属于一个子图. 根据需要, 可以把这 M 个子图分布到 M 机器上. 在切分为 M 个子图之后, 后续的增删操作均使用动态调整, 不再大规模地切分图. 动态调整采用一定的策略来转移顶点, 以达到机器间的负载均衡.

3.1 静态切分(NDA-S)

静态切分策略根据图的性质采用了基于割边的算法. 在无向图中, 如果在子图中任意两点均可由子图中的边到达, 则该子图是连通子图. 如果子图是 G 的连通子图并且将 G 中不在该子图中的顶点加入到子图后, 子图不连通, 则该子图是 G 的极大连通子图, 即连通分量[22]. 假设去掉一条边之后图的连通分量个数增加, 则这条边是图的割边. 标记图中所有割边算法是由 Tarjan[19]提出的, 先对图进行深度优先遍历, 然后按照图的访问顺序对顶点进行编号, 根据顶点编号和顶点所连接顶点的最小编号来判断边是否为割边. 时间复杂度是 $O(|V|+|E|)$, 即只需要遍历一遍图就可以得到所有的割边.

在社交网络中, 由于图的多样性以及用户关系的复杂性,存在大量的割边. 在这些图上, 如果去掉割边, 就形成了很多个连通分量. 对于这些连通分量, 我们对比较大的连通分量内部再进行划分, 对于小的连通分量, 合并划分, 来达到静态切分的结果. 如图 1(a), 割边将图划分为两个连通分量, 对于划分的上面的连通分量, 里面的顶点个数较多, 可以继续划分. 直到图 1(c), 使得每个连通分量的顶点个数处于均衡状态.

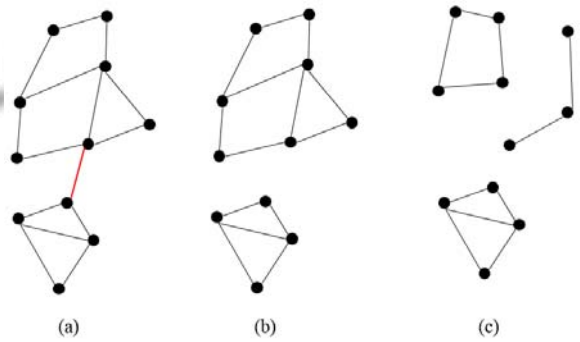


图 1 静态划分

另外, 如何衡量每个节点上顶点个数, 文献[17]定义了不平衡度的概念: 对于无向图 $G=(V,E)$, 为了衡量顶点分布是否均衡, 定义 S_{opt} 为理想状态下每个子图应该有的顶点数, 即

$$S_{opt}=ceil\left(\frac{|V|}{M}\right) \tag{1}$$

其中$|V|$是顶点个数, M为分割的子图数. 对于切分好的子图, 子图中最大的顶点数为S_{max}, 即:

$$S_{max}=S_{opt}\times\left(\frac{100+x}{100}\right) \tag{2}$$

其中x即不平衡度. 不平衡度越小说明子图中顶点的个数越平均.

对删掉割边后的连通分量, 将顶点数量小于S_{max}的连通块按照从大到小排序, 然后使用广度优先搜索进行切分, 从顶点数目小的连通块出发, 搜索该连通块连接的其他连通块进行合并. 切分块数根据这些连通块的总顶点数决定, 切分块数为 P. 对于顶点数大于S_{max}连通块, 我们直接采用 Metis 进行切分, 切分块数为余下的块数, 即$(M-P)$. 切分完之后合并两者的切分结果, 即完成了静态切分.

3.2 动态调整(NDA-A)

由于图在不断地扩张, 删边的操作比加边的操作少得太多, 删边的操作只需要标记即可. 因此, 动态图分割算法只考虑加边、加点的操作.

在解决图分割问题时, 为了衡量划分的效果, 本文定义了子图S负载值的概念, 即:

$$Load(S)=\frac{(|V^S|+|E^S|)\times W^S_{cut}}{|E|} \tag{3}$$

其中$|V^S|$为子图S顶点数, $|E^S|$为子图S边数, W^S_{cut}为子图S与其他子图之间的割边数, $|E|$为原图G的总边数.

针对动态图的特性, 对于新加边$e(u, v)$中u, v是否存在, 动态调整阶段包含三个部分: a. u, v都存在; b. v不存在; c. u, v都不存在. 根据这三种情况, 采取不同的应对方案, 具体的流程图见图 2. 每种策略应对不同的调整算法.

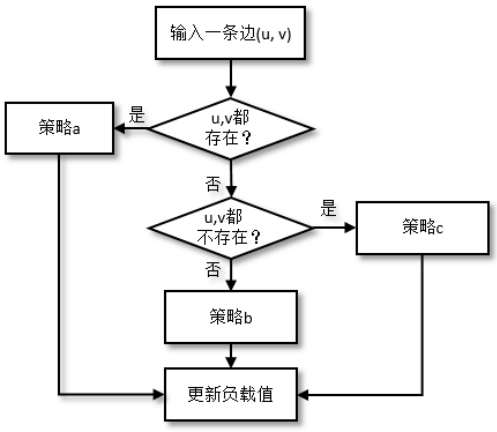


图 2　NDA-A 流程图

3.2.1 u, v都存在(策略 a)

当加入新的边$e(u, v)$并且u, v已经存在时, 已有的M个节点可以看做M个聚类中心$c_1,c_2,\ldots,c_M$. 对于每个聚类中心可以根据公式来计算他们的负载值, 即$l_1,l_2,\ldots,l_M$. 对于点u, v, 如果u和v在同一个子图中, 那么直接把新的边加到该子图中, 如图 3(a); 如果u和v不在同一个子图, 假设u在子图G_u, v在子图G_v中, 分别计算e在G_u, G_v之间; u移动到G_v; v移动到G_u, 如图 3(b)、(c)、(d).

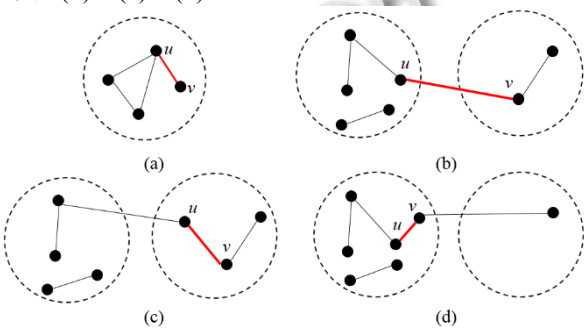


图 3　顶点迁移方案

分别计算这三种情况的负载值, 取三种方案中 M个节点负载值差值最小的方案进行点的移动. 算法的具体流程图见图 4.

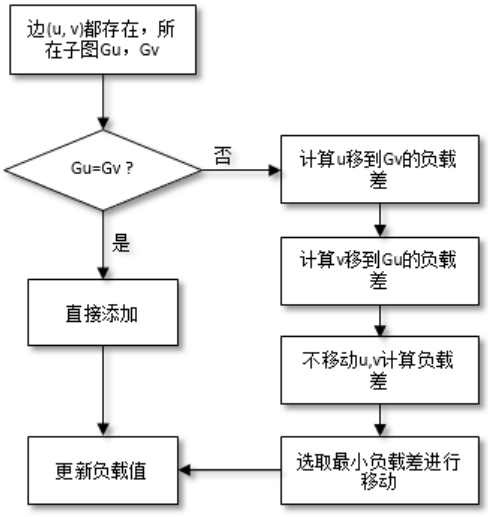


图 4　u, v都存在时顶点转移流程图

3.2.2 v不存在(策略 b)

对于点的调整, 我们采用了类似KNN算法的思想来动态调整图切分. 其中, KNN(K-NearestNeighbor)是一种分类算法, 其核心思想是, 如果一个样本在特征空间中 K 个最相邻的样本中大多数属于同一类别, 那

么就将该样本划分到这一类中. 类似地, 图切分也是这样的道理, 如果一个节点连接的边大多数都属于一个子图, 那么按照割边最小化原理, 显然它也属于这个子图可以达到的割边数最少.

在单独新增加一个顶点时, 如果没有边关联到这个点, 则这个点是孤立的, 不予考虑和其他边的关系, 也不会加到大图上, 只标记到数据库中即可. 当新加入边 $e(u, v)$ 且 v 不在大图上时, 假设 u 在子图 G_u 中, u 所连接点所在的不同子图个数为 p, 即子图为 $G_1,G_2,\ldots,G_p$, 计算它们各自的负载值为 $l_1,l_2,\ldots,l_p$. 由于当子图 G' 增加一个点和割边必然导致 G' 的负载变大, 为了使得最小负载值和最大负载值差距不变大, 因此找到子图 G_{min}, 它具有最小的负载值, 把 v 点加到 G_{min} 中, 连接 G_u 和 G_{min}, 并同时更新它们的负载值, 如图 5.

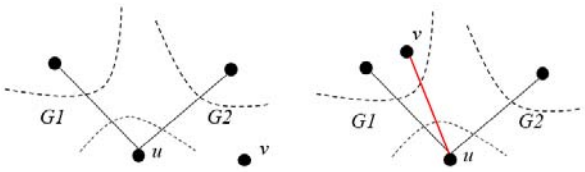


图 5 新增顶点方案

3.2.3 *u, v* 都不存在(策略 c)

如果 *u, v* 都不存在, 说明 *u, v* 都是新点. 对于新点, 对以后的加边是不可预测的, 因此, 为了平衡最小负载节点和最大负载节点, 直接找出最小负载值的节点, 将边 $e(u, v)$ 放入该节点中, 并更新该节点的负载值即可.

4 分割实验和结果分析

4.1 实验环境及数据

实验运行环境为 Ubuntu 14.04, i7 处理器, 3.4GHz, 内存为 16GB, 算法采用 C++实现. 用单机模拟子图分割, 每个子图模拟一台机器, 即下文所说的切割数即机器数.

实验采用不同规模的七个数据集[17,18]. 这些数据集都是图分割常见的数据集, 见表 1.

表 1 数据集特征

数据集	顶点数	边数
facebook	4,039	88,234
ca-HepPh	12,008	237,010
memplus	17,758	54,196
fe_ocean	143,437	409,593
catster	149,700	5,449,275
dogster	426,820	8,546,581
familylinks	623,766	15,699,276

实验主要使用了这七个数据集, 以及在数据集上的动态加边数据. 其中动态加边的数据是在原数据集上随机生成的边, 这些边跟原有的边都不重合.

4.2 静态切割评估

NDA-S 算法是根据割边删掉后的连通分量来进行下一步的切割. 表 2 显示了这几个数据集中去掉割边后的连通分量的个数.

表 2 去掉割边后数据集的连通分量个数

数据集	连通分量
facebook	75
ca-HepPh	1,178
memplus	109
fe_ocean	39
catster	6,299
dogster	27,466
familylinks	27,331

NDA-S 算法是基于割边的一个切割算法, 在不同的切分数目下, 算法的表现可能不同, 表 3 显示了数据集在切割数为 8、64、256 的情况下的切割效果, 并同时与 Metis 工具进行对比切割后的割边和不平衡度. 其中不平衡度越小说明切割结果越平衡.

表 3 NDA-S 与 Metis 对比

数据集	M = 8				M = 64				M = 256			
	NDA-S		Metis		NDA-S		Metis		NDA-S		Metis	
	割边数量	(100+x)%	割边数量	(100+x)%	割边数量	(100+x)%	割边数量	(100+x)%	割边数量	(100+x)%	割边数量	(100+x)%
facebook	3772	102.97%	4313	102.97%	48622	101.56%	49271	101.56%	73876	100.00%	74126	100.00%
ca-HepPh	32732	103.00%	33482	103.00%	84198	102.66%	88072	102.66%	137256	102.64%	139584	102.13%
memplus	12809	102.97%	12892	102.97%	18202	102.52%	18600	102.52%	22785	101.43%	23110	101.43%
fe_ocean	4843	101.03%	5307	101.76%	22975	102.63%	23907	102.90%	44572	102.85%	44986	102.85%
catster	2693481	103.00%	2797900	103.00%	3922808	102.95%	3943373	102.95%	4518011	102.91%	4525229	102.91%
dogster	3843811	103.00%	3889826	103.00%	5288897	102.98%	5338481	102.98%	6135847	102.94%	6191819	102.94%
familylinks	4803167	103.00%	4989776	103.00%	8559641	103.00%	8574981	103.00%	10032816	102.95%	10071272	102.95%

由表 3 数据可以看出，前四个数据集中，对比 NDA-S 和 Metis，在切割数为 8 时，割边数量减少了 13%、2.2%、0.6%、8.7%、3.7%、1.2%、3.7%；切割数为 64 时，割边数量减少了 1.3%、4.4%、2.1%、3.9%、0.5%、0.9%、0.2%；切割数为 256 时，割边数量减少了 0.3%、1.7%、1.4%、0.9%、0.2%、0.9%、0.4%. 而平衡度相比几乎都一样，均在[0%, 3%]之间波动，基本可以忽略. 因为 Metis 是采用了先粗糙化然后细化的切割方法，对于分布密集的图切割的效果会比较好，

因此，将没有割边的连通块使用 Metis 切割效果比较好. 然而，NDA-S 采用了基于割边的搜索处理，对于其他的连通块使用割边连接，这样即保证了割边的优势，又保证了密集连通块也具有较好的切分.

4.3 动态切割评估

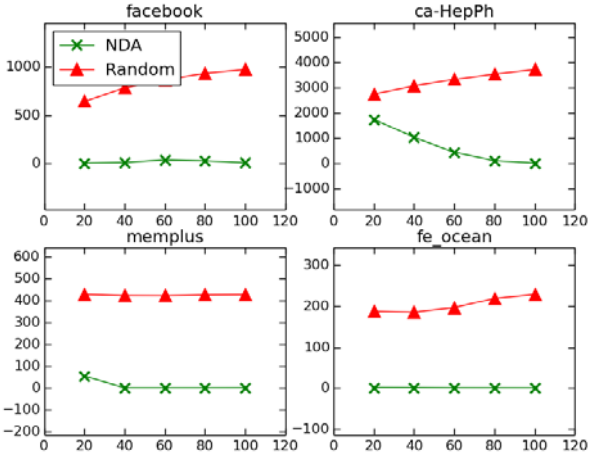
动态切割实验主要使用了两个评估指标：割边数量以及负载差. 子图间的割边数量是衡量一个分割算法好坏的重要因素之一，因为割边越多，子图间的通信量越大，因此，减少割边也是图划分算法的一个重要目的. 由于NDA算法的动态调整是根据负载值来变化的，因此在加边的过程中，各个子图的负载值发生变化. 衡量负载变化引入了负载差的概念，即：

$$\Delta_G = l_{max} - l_{min} \tag{4}$$

其中 l_{max} 是子图中最大的负载值，l_{min} 是子图中最小的负载值. 显然，负载差越小，说明最大负载值和最小负载值之间的差越小，即整体的负载值相差不大，即它们的负载都比较均衡.

4.4 实验结果与分析

随着边的不断增加，平均负载值越来越大. 对比随机加入边与使用 NDA 策略，加的边数量分别为 2 万、4 万、6 万、8 万和 10 万，如图 6，其中横坐标是加边的数量，单位为千，纵坐标是负载差.



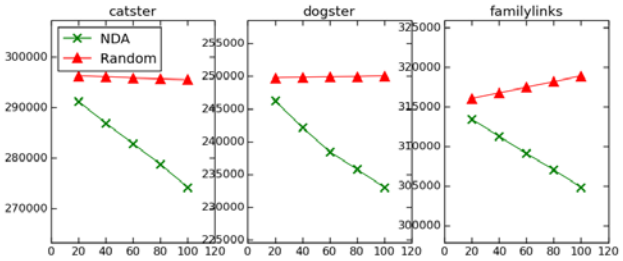


图 6　负载差对比

在切割数为 8 的情况下，可以看出，随着边的增加，随机算法直接将边随机加入到点所在的子图，因此，负载差缓慢上升，说明最大负载值和最小负载值的差值越来越大. 然而，在 NDA 的策略下，负载差不随边的增加而增加，反而呈下降趋势. 因为 NDA 策略不断地调整加入边所在机器的负载，使得最大负载和最小负载的差值稳定在一定的范围内，并且缩小它们的差距，达到负载均衡的效果.

另外，对于采用大图加 10 万条边后的新图，对比采用 NDA 策略的加边、直接使用 Metis 切割新图以及随机加边. 表 4 显示了在切割数为 8 的情况下，它们的割边数量和负载差的对比. 可以看出，采取 NDA 策略后的割边数量与采取 Metis 后割边数量略显优势，割边分别比 Metis 减少了 7%、1%、0.1%、9%、1%、0.1%、0.8%，说明使用动态调整策略割边的效果也不比 Metis 差，甚至略好于 Metis 的切割结果，负载差比 Metis 的小很多. 而随机的割边比使用 NDA 策略的割边数量多很多，负载差也大很多，说明使用 NDA 的转移策略，在整体效果上比较好.

表 4　NDA 和 Metis 对比

数据集	NDA 割边	NDA 负载差	Metis 割边	Metis 负载差	随机割边	随机负载差
facebook	91030	11.7279	98097	1852.5	384280	1917.13
ca-HepPh	122819	0.751781	124117	4332.54	413326	6443.82
memplus	91639	0.669256	91729	3441.14	400948	777.033
fe_ocean	121280	0.804242	133439	196.734	384912	402.575
catster	5806961	252.76	5864168	425.138	13805842	255.045
dogster	7706823	423.464	7711683	3094.75	17593646	426.97
familylinks	12616007	4698.56	12718959	8681.39	27617266	4712.13

5　结语

本文根据图数据的动态扩张的特性，提出了一种基于邻域的图数据分割算法. 在改进的静态切割算法的基础上，动态加入新边，对新边考虑转移顶点，在

转移顶点时考虑邻近子图的负载, 从而达到负载均衡的效果. 通过文中的实验分析, NDA 算法的切割方案对于图的负载均衡效果显著, 并且切割出来的图在子图割边控制上也能达到比较好的效果. 今后, 本文将设计图数据的存储系统, 同本文提出的算法一起构成一个完整的图数据系统.

参考文献

1 于戈,谷峪,鲍玉斌,等.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 Borthakur D. HDFS architecture guide. HADOOP APACHE PROJECT. http://hadoop. apache. org/common/docs/current/hdfs design. pdf.

3 Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM Trans. on Computer Systems (TOCS), 2008, 26(2): 4.

4 Webber J. A programmatic introduction to neo4j. Proc. of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity. ACM.2012. 217–218.

5 Xin RS, Gonzalez JE, Franklin MJ, et al. Graphx: A resilient distributed graph system on spark. First International Workshop on Graph Data Management Experiences and Systems. ACM. 2013. 2.

6 Low Y, Bickson D, Gonzalez J, et al. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012, 5(8): 716–727.

7 Malewicz G, Austern MH, Bik AJC, et al. Pregel: A system for large-scale graph processing. Proc. of the 2010 ACM SIGMOD International Conference on Management of Data. ACM. 2010. 135–146.

8 Shao B, Wang H, Li Y. Trinity: A distributed graph engine on a memory cloud. Proc. of the 2013 ACM SIGMOD International Conference on Management of Data. ACM. 2013. 505–516.

9 Garey MR, Johnson DS, Stockmeyer L. Some simplified NP-complete graph problems. Theoretical Computer Science, 1976, 1(3): 237–267.

10 Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell System Technical Lournal, 1970, 49(2): 291–307.

11 Fiduccia CM, Mattheyses RM. A linear-time heuristic for improving network partitions. 19th Conference on Design Automation. IEEE. 1982. 175–181.

12 Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs. 1995.

13 Karypis G, Kumar V. Multilevelk-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 1998, 48(1): 96–129.

14 Karypis G, Kumar V. METIS-Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. ResearchGate. 1995.

15 Khayyat Z, Awara K, Alonazi A, et al. Mizan: A system for dynamic load balancing in large-scale graph processing. Proc. of the 8th ACM European Conference on Computer Systems. ACM. 2013. 169–182.

16 Salihoglu S, Widom J. Gps: A graph processing system. Proc. of the 25th International Conference on Scientific and Statistical Database Management. ACM. 2013. 22.

17 Walshaw C. The graph partitioning achive, http://staffweb.cms.gre.ac.uk/~c.walshaw/partition.

18 Kunegis J. The Koblenz Network Collection. http://konect.uni-koblenz.de/.

19 Hopcroft J, Tarjan R. Algorithm 447: Efficient algorithms for graph manipulation. Communications of the ACM, 1973, 16(6): 372–378.

20 Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik, 1959, 1(1): 269–271.

21 宁正元,王秀丽.算法与数据结构.北京:清华大学出版社,2006.