软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(3):569−586 [doi: 10.13328/j.cnki.jos.005450]

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

分布式图处理系统技术综述*

王童童[1,2], 荣垂田[3], 卢 卫[1,2], 杜小勇[1,2]

[1](数据工程与知识工程教育部重点实验室(中国人民大学),北京 100872)

[2](中国人民大学 信息学院,北京 100872)

[3](天津工业大学 计算机科学与软件学院,天津 300387)

通讯作者: 卢卫, E-mail: lu-wei@ruc.edu.cn, 杜小勇, E-mail: duyong@ruc.edu.cn

摘 要: 图作为一种基本的数据类型,是对现实世界中对象及其关联关系的一种抽象.现实中,许多科学问题都可以被模型化为图的问题,因此,对图数据进行分析非常重要.图数据分析在语义Web分析、社交网络、生物基因分析以及信息检索等领域有着广泛的应用.随着移动互联、物联网等信息技术的发展,图数据的规模处于持续增长的状态.为了能够应对大规模图数据的高效分析和计算,Google 提出了 Pregel 分布式图处理框架.此后,学术界和工业界提出了许多基于 Pregel 框架的优化技术和系统实现.在充分调研和分析的基础上,首先总结出分布式图处理系统的 3 个优化目标;其次,从计算粒度、任务调度、通信方式、负载划分这 4 个维度,综述现有分布式图处理系统中的各类优化技术;最后,对该领域未来的研究内容和发展方向进行了探讨与展望.

关键词: 分布式图处理系统;计算粒度;任务调度;通信方式;负载划分

中图法分类号: TP311

中文引用格式: 王童童,荣垂田,卢卫,杜小勇.分布式图处理系统技术综述.软件学报,2018,29(3):569−586. http://www.jos.org.cn/1000-9825/5450.htm

英文引用格式: Wang TT, Rong CT, Lu W, Du XY. Survey on technologies of distributed graph processing systems. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):569−586 (in Chinese). http://www.jos.org.cn/1000-9825/5450.htm

Survey on Technologies of Distributed Graph Processing Systems

WANG Tong-Tong[1,2], RONG Chui-Tian[3], LU Wei[1,2], DU Xiao-Yong[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering, MOE (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

[3](School of Computer Science and Software Engineering, Tianjin Polytechnic University, Tianjin 300387, China)

Abstract: Well recognized as a primitive data structure, graph is an abstraction of objects and their pairwise connections. There exists a wide spectrum of applications, including semantic web analysis, social network analysis, biological genetic analysis and information retrieval, which can be modeled as graphs. Therefore, it is of great importance to conduct data analysis over these applications. With the development of information technology such as mobile Internet and Internet of things, the scale of graph data is increasing continuously and rapidly. To provide fast analysis over large-scale graph data, Pregel was first proposed as a distributed graph processing framework by Google. Since then, based on Pregel framework, a variety of optimization techniques and systems have been proposed by academic and industrial communities. Through extensive investigation and analysis, this paper first establishes three optimization objectives for the

* 基金项目: 国家自然科学基金(61502504, 61402329, 61732014, 61472321); 中国人民大学科学研究基金(中央高校基本科研业务费专项资金)(15XNLF09)

Foundation item: National Natural Science Foundation of China (61502504, 61402329, 61732014, 61472321); the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China (15XNLF09)

本文由基于图结构的大数据分析与管理技术专刊特约编辑林学民教授、杜小勇教授、李翠平教授推荐.

收稿时间: 2017-08-01; 修改时间: 2017-09-05; 采用时间: 2017-11-07; jos 在线出版时间: 2017-12-05

CNKI 网络优先出版: 2017-12-06 15:23:36, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1523.014.html

state-of-the-arts solutions to build distributed graph processing systems. Subsequently, it reviews mainstream optimizing techniques for the state-of-the-arts solutions from the perspective of computation granularity, task scheduling, communication mode and load balance. Finally, the paper discusses some open research problems and possible future research directions in this field.

Key words: distributed graph processing systems; calculating granularity; task scheduling; communication mode; load balance

随着信息时代的快速发展,人们在进行 Web 分析、社交网络分析、生物基因分析以及文本检索等方面积累了大量的图数据.对这些图数据的分析具有很重要的现实意义,例如:为了提高用户的搜索体验,Google 公司需要定期对 Web 中数以亿计的网页进行影响力排序[1];Facebook 需要对大量的社交网络图进行分析,以掌控社交网络的结构状态和提高广告的推送准确率[2];生物科学家可以通过对蛋白质的子图匹配等分析了解蛋白质之间的相互作用,进而研制更有效的临床医药[3].但由于现实中图数据规模巨大、结构复杂,再加上图分析算法本身较高的计算复杂性,大图数据的分析已超出了单台计算机的存储和运算能力,给图分析带来了巨大的挑战.

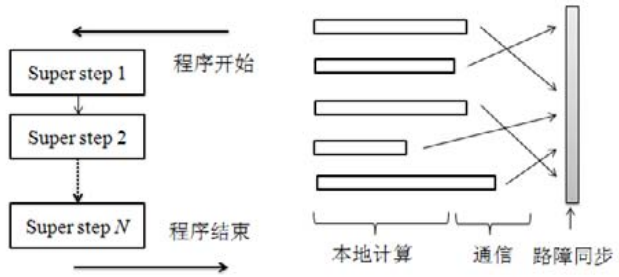
随着计算机软硬件的发展,云计算平台为处理大规模的复杂图数据带来了机遇.MapReduce[4]作为一个简单的分布式计算框架,为海量图数据的处理提供了便利,用户只需要定义 Map 和 Reduce 函数就可以完成相应的运算逻辑.但是由于 MapReduce 框架本身设计上的缺陷,任务的执行需要多次读写分布式文件系统、中间结果无法缓存、任务之间无法共享执行结果等;再加上图的有关算法大多需要多次迭代才能完成,每一次迭代都对应一个 MapReduce 作业,所以 MapReduce 并不能高效地进行图计算.虽然对 MapReduce 有许多改进,如 Haloop[5]、Twister[6]等,提升了其在大图处理方面的性能,但并没有解决 MapReduce 基于分布式文件系统这个根本问题.

与 MapReduce 不善于做迭代运算不同,图灵奖获得者 Valiant 在 1990 年提出的 BSP(bulk synchronous parallel)计算模型[7]尤其适合做数据的迭代运算,其核心思想是:将一个巨大的计算任务分解为一系列的迭代运算,每一个迭代被称为一个超步(SuperStep).BSP 模型具有垂直和水平两种结构.从垂直角度看,一个 BSP 程序由一系列的超步构成,每一个超步完成并同步后进入下一超步,直到计算收敛或者达到最大迭代次数,这种垂直结构与串行程序结构极其类似,如图 1(a)所示.从水平角度来看,在每一个超步内,所有的计算节点并行执行超步内的计算逻辑,该计算逻辑主要可以分为 3 个阶段:本地计算、通信和路障同步,如图 1(b)所示.

(1) 本地计算阶段:各个计算节点相互独立地负责处理存储在本地的数据,各个计算节点的本地计算过程互不干扰;

(2) 通信阶段:在本地计算阶段完成后,各个计算节点之间需要通过网络通信进行信息交换,交换的信息供下次迭代使用.在该阶段,各个计算节点不执行数据的计算处理;

(3) 路障同步阶段:每个计算节点完成本地任务后到达路障同步阶段,在该阶段,已完成计算任务的节点需要等待其他计算节点完成其本地任务,保证在下一个超步开始时每一个计算节点都完成当前超步的全部计算任务.



(a) 垂直结构 (b) 水平结构

Fig.1 BSP model

图 1 BSP 模型

为高效地处理大图迭代计算问题,受BSP计算模型的启发,Google在2010年首次提出以顶点为中心的分布式图计算框架 Pregel[8].在以顶点为中心的计算模型中,用户只需要自定义一个与具体应用紧密相关的图节点计算函数 *compute*()即可,不需要考虑数据的全局处理流程.在 *compute*()函数中定义了图节点需要执行的运算逻辑,例如处理上次超步接收的消息、更新顶点的值、向邻居节点发送消息、改变节点状态等.Pregel 的运行过程为:首先,按照用户定义的图分区函数将图划分为若干分区并分配到不同计算节点上;然后,对每一个图节点迭代执行 *compute*()函数.图节点的每一次迭代运算都对应 BSP 模型中的一个超步,每两个超步之间通过全局同步来进行分离,程序迭代执行直到没有活跃节点且无消息传播或者达到最大迭代次数时停止.Pregel 在图切分、通信管理、同步控制和容错恢复方面都提出了可行的解决方案,并从架构上消除了运算过程中图数据重复加载的问题.虽然 Pregel 框架计算模型简单,具有较好的可扩展性,但该框架存在 3 个方面的限制.

(1) 计算过程收敛速度慢,导致迭代次数多.以顶点为单位的细粒度计算、同步的计算模式,限制了整个任务计算的收敛速度;

(2) 消息量大,导致通信代价高.分布式图处理系统中,为了计算的并行化,需要将整个图进行划分,并把每个分片分配到不同的计算节点进行处理.由于不同分片中的顶点与顶点之间存在关联,顶点间关联的稠密程度影响着消息量大的小.在实际图,特别是社交网络中,图中顶点的度数服从幂率分布,Pregel 系统在处理此类大规模图数据分析时产生大量的消息,从而使得通信的代价成为制约系统性能的瓶颈之一;

(3) 负载不均,存在木桶效应.不合理的数据划分、迭代过程中不同数据分片的收敛情况不一致以及节点之间的计算能力存在差异等原因,造成了节点之间计算的负载不均匀.

为了突破 Pregel 框架的限制,现有的工作主要围绕着以下 3 个优化目标展开研究,如图 2 所示.

(1) 加快算法收敛,减少迭代次数.现有的研究主要从以顶点为单位的细粒度计算过渡到以路径、子图为单位的粗粒度计算,从同步的计算模式过渡到异步的计算模式以及混合计算模式这两方面开展工作;

(2) 减少消息数目,减轻网络负载.现有的研究者提出了基于顶点的划分策略和计算模式、基于共享内存的通信方式、消息合并技术以及 Receiver-side scatter 技术等;

(3) 消除木桶效应,实现负载均衡.为了解决这个问题,研究者从图划分策略、动态负载迁移以及调度模式出发提出了多种可行的优化方案.

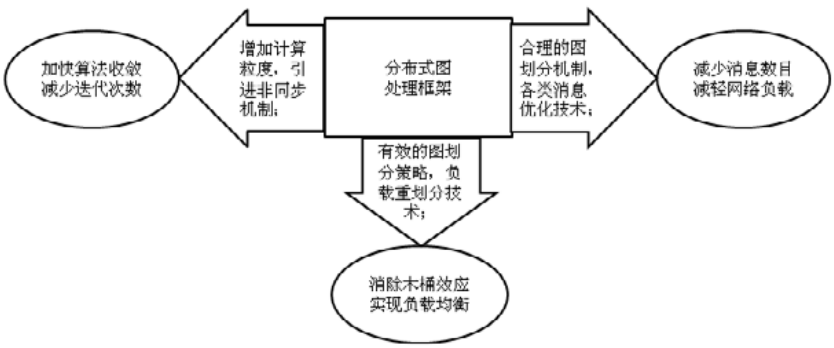


Fig.2 Optimization targets of distributed processing systems

图 2 分布式图计算系统优化目标

为了能够帮助开发者深入了解分布式图处理系统的优化机制,本文以 3 个优化目标为抓手,对现有的各类分布式图处理系统中所采用的优化技术做详细的技术综述.第 1 节阐述常见分布式图处理系统所采用的计算粒度.第 2 节~第 4 节分别对分布式图处理系统的任务调度机制、通信机制和负载划分所采用的技术及优化方案进行综述.第 5 节是对全文的总结及对未来的展望.

1 计算粒度

当前,分布式图处理系统在对图数据进行遍历计算的过程中主要使用了 3 种不同的计算粒度:以顶点为计算粒度、以子图为计算粒度和以路径为计算粒度.以顶点为计算粒度的系统结构简单、易于实现且算法表达能力强,但是不利于算法收敛且网络负载较重;以子图为计算粒度的系统虽然算法表达能力较弱,但可以在减少通信量的同时,有效地减少计算迭代次数;以路径作为计算粒度可以减少对边的随机访问率,提高图遍历算法的性能,但是在分布式环境下缺少有效的实践.表 1 展示了常见图处理系统的计算粒度.

Table 1 Graph processing systems and their calculating granularity

表 1 图处理系统计算粒度对照表

名称	以顶点为计算粒度	以子图为计算粒度	以路径为计算粒度
Giraph[9]	✔	✘	✘
Pregel[8]	✔	✘	✘
Pregel+[10]	✔	✘	✘
GPS[11]	✔	✘	✘
PowerGraph[12]	✔	✘	✘
Hama[13]	✔	✘	✘
GraphHP[14]	✘	✔	✘
HybridGraph[15]	✔	✘	✘
PowerSwitch[16]	✔	✘	✘
X-Stream[17]	✘	✘	✔
PathGraph[18]	✘	✘	✔
GraphSteal[19]	✔	✘	✘

1.1 以顶点为计算粒度

在以顶点为计算粒度的分布式图处理系统中,其迭代处理过程是通过对图节点的遍历计算完成的.首先,用户需要以图中的节点为粒度定义运算函数,这个运算函数的功能包括消息的收发、节点值的更新以及顶点状态的改变等.然后,系统按照该运算函数对图中的每一个顶点进行同步或者异步的更新运算.例如,存在如图 3 所示的包含有两个子图的强连通图(图中实线箭头表示跨子图边,虚线表示子图内部边),且每一个节点都有一个整型数值,图 4 展示了在以顶点为计算粒度的模式下求解图 3 中节点最大值的迭代过程.

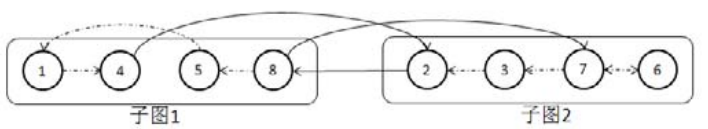


Fig.3 An example of graph data

图 3 图数据示例

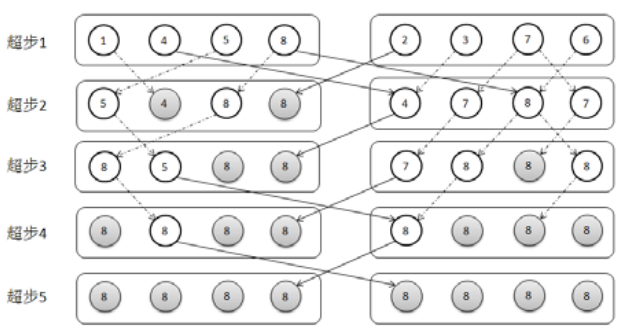


Fig.4 Find the max value in the graph under vertex-centric mode

图 4 以顶点为计算粒度求解图中最大值

在第 1 次迭代(超步 1)中,每一个图节点向其邻居节点广播自己的 *value* 值,如图中箭头所示,其中,虚线箭头表示子图内部通信,实线箭头表示跨子图通信.在之后的迭代中,每个图节点选取其所收到消息的最大值与自己的 *value* 值进行对比:如果消息最大值大于自己的 *value* 值,则对 *value* 值进行更新,并向邻居节点广播更新后的值;否则,将自己的状态设置为非活跃(图中暗色节点).迭代过程直到所有节点都为非活跃状态且没有消息产生为止.在该例中,共进行了 5 次迭代,8 次子图间的信息传递.

以顶点为计算粒度的模式逻辑简单且具有良好的算法表达能力,因此被绝大多数的分布式处理系统所采用,如 Giraph[9]、GPS[11]、Mizan[20]、Pregel+[10]、GraphX[21]等.但该模式却牺牲了图节点访问同一分区中其他节点信息的灵活性,导致使用该模式的系统收敛速度过慢、网络通信量大.

1.2 以子图为计算粒度

为加快算法收敛速度,减轻网络负载,大量以子图为计算粒度的分布式图处理系统被提出,如 GraphHP[14], GoFFis[22]等系统.目前,以子图作为计算粒度的系统的计算模式主要分为两类.

- 第 1 类以顶点为计算粒度的模式为基础,将一个超步划分为两个阶段来实现:超步的第 1 阶段中,所有子图的边界节点进行信息交互;第 2 阶段中,子图内部的节点执行计算逻辑直到内部收敛,然后进入下一个超步;
- 第 2 类是直接在子图上进行运算,并以子图为粒度进行信息交互.例如,图 5 展示了在子图上直接进行运算求解图 3 中节点最大值的迭代过程:在第 2 次迭代中,每一个子图根据遍历或者其他方式求解出子图内部节点最大值,并将其设置为自己的 *value* 值,然后向其邻居子图广播该值(如图中箭头所示),在之后的迭代中,每一个子图选取消息中最大值与自己的 *value* 值进行对比:如果该最大值大于自己的 *value* 值,则将其更新为自己的 *value* 值,并向其邻居子图广播;否则,将自己设置为非活跃状态(图中暗色子图).迭代持续进行,直到所有子图为非活跃状态且没有消息传播.在该例子中共进行了 3 次迭代,产生了 3 条跨子图的消息.

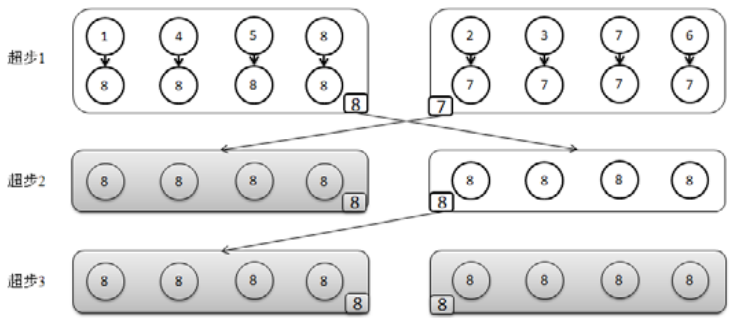


Fig.5 Find the max value in the graph under subgraph-centric mode

图 5 以子图为计算粒度求解图中最大值

通过对比图 4 和图 5 可以发现:以子图为计算粒度的分布式图处理系统不仅可以有效地减少运算过程中的迭代次数,而且可以显著地减轻网络负载.但是此类模式的算法表达能力较弱,并不能够取代以节点为计算粒度的计算模式.GraphHP 和 P++[23]等系统在系统实现时采用了第 1 类以子图为计算粒度的计算模式,GoFFis 使用了第 2 类以子图为计算粒度的计算模式,而 Giraph++[24]和 Blogel[25]等系统同时采用了这两类计算模式.

1.3 以路径为计算粒度

目前,以路径为计算粒度的图处理系统主要是集中式的,采用该计算粒度的目的是为了减少对边数据的随机访问率,提升系统性能.X-Stream[17]是此类系统的代表,为减少对磁盘的随机访问率,该系统将边数据和点数据进行分区,并对边采用顺序访问的方式进行遍历计算.PathGraph[18]同样使用了以路径为计算粒度的集中式图

处理系统,其将图数据按照树的形式划分为若干分区,以此来提高图处理过程中数据的局部性.但是到目前为止,在分布式环境下,以路径为计算粒度还缺少有效实践.

2 任务调度机制

任务调度是分布式图处理系统必须解决的问题.根据图迭代计算过程中每个超步之间是否有明确界限,或者说不同超步之间是否交错运行,可以将分布式图处理系统所采用的调度模式分为 3 种:同步调度模式、异步模调度模式和混合调度模式,其中,每一种调度模式都有自己的优缺点和适用场景.表 2 展示了部分系统及其采用的调度方式.

Table 2 Distributed graph processing systems and their task scheduling modes

表 2 分布式图处理系统调度方式对照表

名称	同步调度模式	异步调度模式	混合调度模式
Giraph	✔	✘	✘
Pregel	✔	✘	✘
Pregel+	✔	✘	✘
GPS	✔	✘	✘
PowerGraph	✔	✔	✘
MOCGraph	✔	✔	✘
Hama	✔	✘	✘
GraphHP	✘	✘	✔
HybridGraph	✔	✔	✘
PowerSwitch	✘	✘	✔
GraphSteal	✘	✔	✘

2.1 同步调度模式

在同步调度模式中,两个相邻迭代之间存在同步控制机制,需要在当前迭代结束后才能进入下次迭代.同步调度模式能够保证在当前迭代过程中,执行运算的图节点所能看到的数据是上次迭代所有节点执行更新后的结果数据.同步调度模式控制机制简单,便于用户理解且具有较强的表达力;在采用同步调度模式的系统上运行图算法,运行过程及结果是确定的,这有助于同步程序的设计、调试和部署;同步调度模式在扩展性方面也具有优越的性能,在该模式下,图运算时间随着机器数目的增加而线性减少[8];鉴于同步调度模式的这些优良特性,其广泛被 Giraph,GPS,Hama[13],GraphX 等系统所采用.例如采用了 Master-Slaver 体系结构的 Giraph 系统,它通过 Zookeeper[22]来实现系统同步.在该系统的迭代过程中,当 Worker 执行完本地计算后,通过 Zookeeper 向 Master 汇报完成状态,Master 会适时检测所有的 Worker 是否都已经完成本地计算.如果所有 Worker 完成本地计算,则进入下一次迭代.

虽然同步调度模式在简单易用和扩展性方面表现良好,但也存在多方面的不足:(1) 计算节点之间的同步会产生额外的开销,例如在对图求解单源节点最短路径[26]这个问题时,同步开销占总运行时间的 80%左右[14];(2) 由于计算节点处理能力不同或者图数据划分不均匀,会导致在处理同一迭代时不同计算节点执行运算的时间差异很大,因同步的存在使得处理最慢的节点成为该次迭代的瓶颈,从而产生木桶效应,再加上图算法较高频次的迭代,加剧了木桶效应的影响;(3) 在同步模式下,两次迭代之间,消息的传播只能发生在直接邻居节点,这种低效的传播方式使得同步算法在某些应用下不能够有效收敛,例如在图染色算法中,如图节点试图获得非邻居节点的颜色值,只能够通过两者的共同邻居节点来获得,但是同步执行过程中可能会出现这两个节点颜色不断翻转的情况,从而无法收敛[27].为解决同步模式中的这些问题,科研工作者提出了异步调度模式和混合调度模式.

2.2 异步调度模式

在异步调度模式中,两次迭代之间没有明确的界限,活跃图节点只要接收到其需要的消息,不需要等待其他所有图节点都获得所需计算资源,就可以动态地接受调度器的调度执行,并向其他图节点广播计算结果.在异步调度模式中,因为摒弃了同步路障阶段,所以能够有效减少木桶效应的影响;经大量的理论分析和实验表明,异

步模式在执行效率和资源利用率方面相较于同步调度模式都有较大的优势,尤其在图处理系统负载分布不均时,这种优势体现的更加明显[12,28];使用异步调度,图运算可以更加快速地收敛,例如执行 PageRank 算法[27]时,大部分图节点在执行一次迭代后就可以收敛,只有 3%的节点为达到收敛需求需要 10 次以上的迭代.另外,异步调度模式提供了灵活的调度方式,用户可以通过调整调度方式,更加高效地处理不同类型的图计算.常见的异步调度系统主要包括 GraphLab,Maiter 以及 PowerGraph 等系统.

PowerGraph 系统是卡耐基梅隆大学提出的基于分布式共享内存的图处理系统[12],该系统首次提出并应用了 GAS 模型,该模型将节点的计算逻辑细分为信息收集阶段(gather)、应用阶段(apply)以及分发阶段(scatter).在信息收集阶段,当前活跃图节点 v 通过一个累加函数收集所有邻居节点及其边的信息;在应用阶段,节点 v 利用收集到的信息进行更新计算;在分发阶段,节点 v 更新计算与 v 相连的边的值.在 PowerGraph 系统中,用户可以设定当 v 在 scatter 阶段收集到需要的全部或部分数据后,无需等待系统同步便可立即接受调度器的调度参与运算来实现异步调度.

尽管异步调度模式可以提高某些算法的性能,但是该模式仍然存在多方面的不足之处.

(1) 异步调度模式的分布式图处理系统设计相较于同步系统来说更加复杂,在设计过程中,不仅需要设计正确高效的调度器,还要考虑数据的一致性问题,必须设计一套额外的机制来保证相同数据对象可以被不同进程互斥访问,例如, GraphLab 通过引进版本控制和分布式锁机制来保证数据的一致性[27];

(2) 异步调度的计算过程和结果具有不确定性,像 statistical simulation 等应用,如果算法设计不合理,迭代计算可能无法达到预期的收敛效果,即使能够收敛,有些图节点在最终收敛前会出现冗余计算的现象[16],不仅浪费 CPU 资源,还会额外增加网络开销;

(3) 相较于 Pregel 等同步系统来说,异步调度的编程难度有所增加,在编程过程中不仅需要编写最基本的运算逻辑,还需要选择合适的调度规则;另外,程序的异步执行对算法的调试也提出了更高的要求.

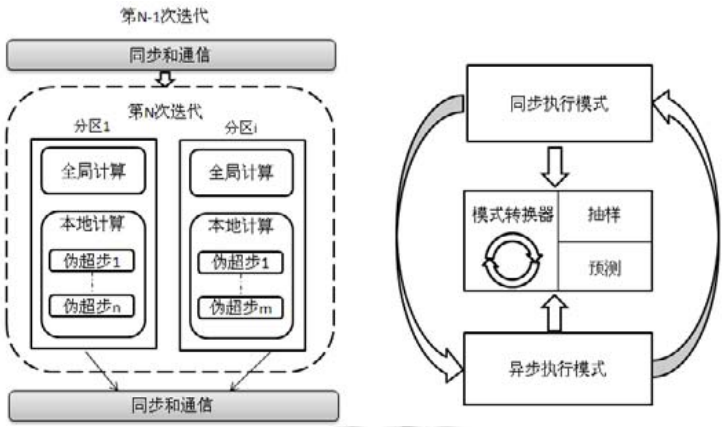
鉴于异步调度模式的这些缺点,MOCGraph[30]、PowerGraph 等系统在提供异步调度方式的同时也支持同步调度.

2.3 混合调度模式

在深入分析同步和异步调度模式系统的优缺点后,许多学者提出了混合调度模式的调度机制,比较有代表性的系统包括 GraphHP 和 PowerSwitch.

GraphHP 系统的混合调度模式是基于 BSP 模型实现的.在该调度模式中,整个计算过程仍由一系列的全局迭代组成,每个全局迭代对应 BSP 模型的一个超步,即包含“计算-通信-同步”这 3 个阶段,但计算阶段被细化为全局计算和本地异步计算两部分.GraphHP 加载图数据时会将每个分区的图节点分为两类:没有邻接点位于其他分区的本地节点和有邻接点位于其他分区的边界节点.GraphHP 的迭代计算过程包含 3 步:第 1 步是对边界节点执行全局计算;第 2 步对每个分区中的全部节点执行由一系列的伪超步构成的本地异步计算,待本地异步计算收敛后,需要把全局计算与本地计算中所产生的要发送给边界节点的消息通过网络传输给相应节点;在最后一步中,系统执行全局同步工作并开始下一次迭代,直到算法终止,如图 6(a)所示.该模型在保留了 BSP 模型简单易用的基础上引进了本地异步计算的思想,可以大幅度地减少迭代次数,进而减少全局同步和通信的代价.另外,P++系统[17]也使用了类似的混合调度模式.

PowerSwitch 系统是 Xie 等人基于 PowerGraph 开发的采用混合调度模式的计算平台[16].Xie 等人发现:不同调度模式下,图算法的执行算效率受到图应用算法、图划分方式、迭代执行进度、输入图的特性以及集群状态等多方面的影响.因此,他们在 PowerSwitch 中通过一组启发式的算法建立代价收益模型来动态预测同步模型和异步模型两种调度方式的性能,并实现在计算过程中对两种模型的自由切换,如图 6(b)所示.实验结果表明:PowerSwitch 可以准确预测两种模型的性能,并且快速地完成调度模式切换,相比于使用单一的调度模式,在其上执行的大量图算法,如 PageRank、单源节点最短路径等,都在执行效率上得到不同程度的提高[30].



(a) GraphHP 混合调度模式 (b) PowerSwitch 混合调度模式

Fig.6 Hybrid scheduling mode

图 6 混合调度模式

3 通信机制

分布式系统中,计算节点之间相互通信是不可避免的.在分布式图处理系统中,通信指运算过程中图节点之间的信息交互.当图数据规模巨大且执行通信密集的图计算时,计算节点之间的通信量会明显上升.如果没有高效的通信机制,网络通信会成为整个系统的瓶颈.本节将从通信方式和工作模式两个角度对分布式图处理系统所采用的通信机制进行分析.

3.1 通信方式

目前,常见分布式图处理系统所采用的通信方式主要可以分为两类:基于消息传递的方式和基于共享内存的方式.表 3 展示了部分系统及其采用的通信方式.

Table 3 Distributed graph processing systems and their communication modes

表 3 分布式图处理系统通信方式对照表

名称	消息机制	共享内存
Giraph	✔	✘
Pregel	✔	✘
Pregel+	✔	✘
GPS	✔	✘
PowerGraph	✘	✔
Hama	✔	✘
GraphHP	✔	✘
HybridGraph	✔	✔
MOCGraph	✔	✘
PowerSwitch	✘	✔
GraphSteal	✘	✔

3.1.1 基于消息传递的方式

在基于消息传递的方式当中,图节点之间的信息交互通过利用网络通信平台在节点之间发送消息来实现.消息中包含要传送的数据和目标顶点 ID.通常使用的网络通信协议主要包括 4 大类:基于 RPC(remote procedure call)通信[31]、基于 Netty 通信[32]、基于 ActivMQ[33]的通信以及基于 MPI(message Passsing interface)通信[34].在具体分布式图处理系统中,往往会采用一种或多种通信协议,如:Hama 仅使用了 RPC 通信协议;而 Giraph 同时支持 RPC 和 Netty 协议,其通信协议可以通过配置文件进行选择.

基于消息传递机制的系统因不需要其他额外机制就可以保证数据的一致性,且具有优良的可扩展性,因此被大多数的同步图处理系统采用,例如,GPS、Giraph、Hama、Pregel、HybridGraph 等.在此类系统中,图节点在

计算过程中会根据运算逻辑产生相应的消息并发送到目的节点.如果目的节点和源节点位于同一台机器上,则直接将该消息放到相应图节点的消息队列中;否则,将该消息放置到消息发送缓冲池中等待发送.每当有消息添加到消息发送缓冲池后,计算节点都会检查该缓城池的大小是否达到了特定阈值:如果是,则调用相应的发送接口将池中消息批量发送出去.当目的计算节点接收到消息后,会根据消息目的 ID 将其加入到对应图节点的消息队列中.在消息的发送过程中,系统会使用批量发送的方式来优化网络通信.

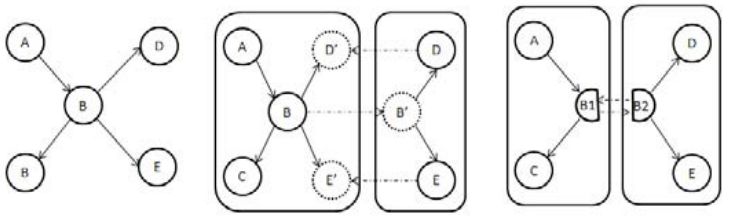
3.1.2 基于共享内存的方式

在基于共享内存进行通信的分布式图处理系统中,每个图节点的数据以共享变量的方式存储在计算节点上,当某活跃图节点在计算过程中需要其他节点数据时,可以直接按照相应的内存地址进行读取.在分布式环境中,因每个计算节点都有自己独立的内存地址且需要保持数据的一致性,所以使得共享内存的通信方式实现起来变得较为困难[35,36]

为有效地管理集群中各个计算节点的内存地址,微软推出的分布式图计算框架 Trinity[37]设计了一套有效的集群内存管理方案.该方案将集群内每个计算节点的内存组织成一个巨大的虚拟内存空间,并按照一定模式给予每个存储单元一个 64 位的存储空间地址,存储在集群内的任意图节点都可以使用该存储空间地址访问虚拟内存空间中的任意单元,从而使得集群共享内存通信在形式上与单机环境类似.

GraphLab 系统为被远程访问的图节点设置了本地 ghost 节点,并在该 ghost 节点中保存与原节点相同的数据信息[27].当其他节点需要访问远程图节点时,可以通过本地的内存操作访问 ghost 节点获取到同样的数据.在 GraphLab 中,ghost 节点和原节点的数据一致性是通过 pipelined distributed locking[38]保证的.图 7(b)是 GraphLab 将图 7(a)所示图加载到两个不同机器上,并使用共享内存通信的示例图.在 GraphLab 加载完数据后,检测到节点 B 的邻居节点 D 位于另一个计算节点上,便会在 B 所在的计算节点上创建与 D 对应的一个 ghost 节点 D',出于同样原因,会创建 E'和 B'两个 ghost 节点,节点 D 和节点 D'通过相应的机制保持数据一致,当 B 需要 D 的数据时,可以直接读取位于本地的 D'对应的内存变量.

PowerGraph 系统同样采用了共享内存的机制来进行通信.与 GraphLap 不同,PowerGraph 系统将图数据按照顶点切分的方式分布在不同计算节点上.当某一图节点需要读取被切分顶点数据时,只需要读取位于本地的 ghost 图节点数据即可.PowerGraph 通过 Chandy-Misra locking[39]机制来保证 ghost 节点和原节点之间的数据一致性.如图 7(c)所示,因为 $B1$ 和 $B2$ 两节点数据是一致的,所以当 E 节点需要读取 B 节点数据时,只需要读取本地 $B1$ 节点的数据即可.在 Spark[40]上开发的专门用于图计算的 GraphX 系统也使用与 PowerGraph 类似的共享内存通信方式.



(a) 原始图 (b) GraphLab 共享内存通信 (c) PowerGraph 共享内存通信

Fig.7 A diagram of shared memory based communication

图 7 共享内存通信示意图

采用共享内存通信方式的系统,虽然需要通过网络通信与额外机制来保证分布在不同机器上主从节点数据的一致性,但相对于采用消息传递机制的系统来说,网路负载大为减小.但是一致性机制的引入,在带来通信优势的同时也严重影响了系统的可扩展性,随着集群中计算节点和图分区的增多,系统需要耗费更多的时间和计算资源来维护执行一致性机制[40];另外,像 GraphLab 这类系统,为实现共享内存通信而引进 ghost 节点,会增加内存的开销[11],当图数据规模巨大且系统内存资源紧张时,部分图节点将被迫转移到磁盘上,从而会引起频繁的

磁盘访问,严重影响系统的性能.

3.1.3 优化机制

鉴于在分布式图处理系统中,通信代价严重影响整个系统的性能,对通信方式进行优化以减少消息的时空和网络开销显得格外重要.在过去的研究中,主要出现了两种优化方式:第 1 种是通过合理的图分割技术降低图分区之间的连通性,减少跨机器的通信请求,虽然该方式不能够减少消息的存储空间,但可以从根本上减少跨机器的消息数量,详见第 4 节;第 2 种是通过 Combine、Receiver-side scatter 等技术减少消息数量,降低网络开销.

Pregel 系统受到 MapReduce 中 Combine 机制[4]的启发,实现了消息的 Combine 机制.Combine 机制会将发送到相同目的顶点的消息合并成一条,在降低网络开销的同时减少消息的存储开销.例如:在计算单源节点最短路径时,因图节点计算过程中仅需要消息的最小值,所以在发送消息时,发送端机器会首先对要发送的消息进行遍历,并将发往同一目的节点的消息求最小值后,仅将该最小值发送出去.当系统采用集中式通信且目的顶点分布集中时,Combine 机制效果明显,但是由于多数系统使用异步发送消息的机制,且消息目的顶点在分布上具有较差的局部性,使得 Combine 机制带来的收益减小,甚至难以抵消消息遍历带来的开销.因此,后来的 Giraph、GPS 等系统摒弃了发送端 Combine 功能,仅保留接收端 Combine 的机制,以减小消息内存消耗.另外,因 Combine 机制的另外几个缺陷是:① 较差的适用性,要求消息必须具有可交换和可合并两个特性,② 消息被合并后不能立即处理仍需驻留内存.Zhou 等人在 MOCGraph 系统中对 Combine 机制进行优化后提出消息在线处理技术[30],该技术不仅放宽了对消息合并的要求,只要求其具有可交换性,而且允许消息到达目的节点后立即参与对目的节点更新计算并释放,节省了内存空间.在与 Giraph 的对比实验中,作者发现:在相同计算环境下,消息在线处理技术可以显著地提高系统的性能[30].

Receiver-side scatter 技术是与 Combine 类似的技术,不同的是,该技术是对发往同一目的机器不同目的节点的消息进行合并处理.当一条消息准备发送到某一远程计算节点的多个图节点时,该条消息将会首先发送到对应的远程计算节点,然后再由该远程计算节点分发消息到相应的目的图节点.采用该项技术的系统主要包括 GPS 和 X-Pregel[41]等系统.Pregel+的开发人员 Yan 等人发现,像图染色问题等大量的图算法在计算过程中需要图节点反复请求邻居节点的属性信息[10].为减少该类请求和相应消息的数量,他们使用了 Requset-Response 技术.在该技术中,集群各个计算节点专门开辟了一条用于请求节点属性的 Requset-Response 通道.在该技术中,某计算节点上的多个图节点请求同一远程图节点的属性信息主要包含 3 个步骤:(1) 该计算节点在通道上发布一条请求信息;(2) 被请求节点接到请求后向通道中写入被请求的属性信息;(3) 发布请求信息的计算节点获得响应后,所有请求图节点都可以直接在通道中获取需要的图节点的属性信息.该技术通过对请求及相应消息进行合并,极大地减小了网络负载.

3.2 工作模式

在分布式图处理系统中,工作模式按照信息的流向分为两类:Push 模式和 Pull 模式[15].表 4 展示了部分系统及与其对应的工作模式.

Table 4 Distributed graph processing systems and their operating modes

表 4 分布式图处理系统工作模式对照表

名称	Push 模式	Pull 模式
Giraph	✔	✘
Pregel	✔	✘
Pregel+	✔	✘
GPS	✔	✘
PowerGraph	✘	✔
Hama	✔	✘
GraphHP	✔	✘
HybridGraph	✔	✔
MOCGraph	✔	✘
PowerSwitch	✘	✔
GraphSteal	✔	✘

在 Push 模式中,信息由当前活动图节点流向邻居节点,即:当前活动图节点完成计算并产生相应的数据信息

后,数据信息按照出边传输到相应邻居顶点.在内存资源充足的条件下,该工作模式允许各个计算节点高效并发地对图节点进行处理,但是该模式需要消息产生后立刻发送到目的节点,并要目的节点对消息进行存储,增加了系统对内存的需求.目前,GPS、Giraph、Hama 以及 MOCGraph 等系统支持 Push 模式.图 4(a)展示了 Push 模式下 PageRank 算法的伪代码:节点 *v* 在的某次迭代运算中,首先获取上次迭代收到的存储在本地的消息(第 2 行);然后,利用这些消息计算更新 PageRank 值(第 4 行~第 7 行),并主动向其邻居节点发送消息,即,其 PageRank 值与其出度的商(第 8 行、第 9 行);最后,如果迭代达到了最大迭代次数,则将自己的的状态设置为非活跃(第 10 行、第 11 行).

在 Pull 模式中,信息由邻居节点流向当前活动图节点,即,当前活动图节点在计算过程中需要数据信息时会按照入边向其邻居节点请求数据.在 pull 模式下,因为图节点请求到数据后会立刻参与计算并释放对应的消息,所以可以避免存储大量的消息,减少系统对内存的需求.然而在消息传递方式下使用 Pull 模式时,因图节点向其邻居请求数据时需首先将自己的 ID 发送给对方,然后才能获取到需要的数据,增加了额外的通信请求开销;在共享内存模式下使用 Pull 机制时,虽然图节点可以直接读取相应的请求数据,但是如第 3.1.2 节所述,因 ghost 节点的引入,会使得系统对内存的需求增加.使用 Pull 模式的系统主要包括 Chronos[42]、Seraph[43]以及 PowerGraph 等.图 4(b)展示了 Pull 模式下 PageRank 的伪代码:图节点 *v* 在进行迭代运算时,首先需要向其邻居节点发送数据请求(第 7 行);邻居节点在收到请求后,将消息(邻居节点的 PageRank 值与其出度的商)发送给节点 *v*(第 1 行~第 5 行);随后,*v* 计算并更新自己的 PageRank 值(第 8 行~第 12 行),节点 *v* 在计算完成后不需要向其邻居节点广播消息,只需设置相应 *flag* 通知邻居节点其值已经更新(第 13 行);最后判断是否达到了最大迭代次数,如果是,则将节点 *v* 的状态设置为非活跃(第 14 行、第 15 行).

```
1  vertex.compute(){
2    Message msgs = getRecMsg();
3    doubule  sum = 0.0;
4    for(Message m: msgs.iterator())
5      sum = sum + m.getValue();
6    double newVal =
       0.15/getNumVertices() + 0.85*sum;
7    setValue(newVal);
8    for(Edge e: getOutEdges().iterator())
9    sendMsgTo(e.Target(),
       getValue()/getOutDegree());
10   if(getNumSupersteps() > maxNum)
11    voteTohalt();
12 }
```

(a) Push 模式下 PageRank 伪代码

```
1  Vertex.pullRes(){
2    if(getUpd() is True)
3      for(Edge e: getOutEdge().iterator())
4        sendMsgTo(e.Target(),
           getValue() / getOutDegree());
5  }
6  Vertex.update() {
7    Messages msgs = getRecMsg();
8    double sum = 0.0;
9    for(Message m : msgs.iterator())
10     sum = sum + m.getValue();
11   double newVal =
       0.15/getNumVertices() + 0.85*sum;
12   setValue(newVal);
13   setUpd();
14   if(getNumSupersteps >maxNum)
15   voteToHalt();
16 }
```

(b) Pull 模式下 PageRank 伪代码

Fig.8 Pseudo code of PageRank under push and pull mode

图 8 push 和 pull 模式下 PageRank 伪代码

因采用 Push 或 Pull 模式的系统性能受算法、集群环境等多方面的影响,2013 年,Beamer 提出了一种基于共享内存的广度优先算法,在该算法的具体执行过程中,可以通过对 Push 和 Pull 模式的动态转换获取更高的性能[44];Shun 开发了单机图处理系统 Ligra,该系统可以通过一个特定的阈值来调整 Push 和 Pull 模式,但是 Ligra 并没有设计相应的存储技术用以支持单节上的大图运算[45];在分析深入分析 Push 和 Pull 机制的优缺点之后,Wang 等人在文献[15]中提出了同时支持 Push 和 Pull 机制的分布式图处理系统 HybridGraph.该系统在迭代运算过程中不断地收集运行过程中 IO 和通信量等数据,并通过一个代价函数对使用 Push 和 Pull 的代价进行计算,用以指导系统在 Push 和 Pull 模式之间进行变换.在实验部分,作者对比了 HybridGraph、Giraph 以及 PowerGraph 等系统的性能.实验结果表明:在内存充足和内存紧张两种状态,HybridGraph 的执行效率相较于使用单一模式的 Giraph 和 PowerGraph 都有不程度的提高.

4 图划分

图数据划分是进行分布式图处理的基础,划分结果的好坏严重影响着分布式图处理系统的性能.一个有效的图划分策略能够在使整个系统达到负载均衡的同时,尽可能地减少网络开销.在图划分过程中应遵循两个重要的原则:首先是降低划分后子图之间的连通性,以降低网络开销;其次是保证子图大小均匀,以实现系统的负载均衡.本节将从图划分方式和图划分策略两个方面对图划分进行论述.

4.1 图划分方式

分布式图处理系统在对图数据进行划分时,主要采用了两种划分方式:边切分方式(vertex-cut)[8]和点切分方式(edge-cut)[12].表 5 展示了部分系统及其采用的图划分方式.

Table 5 Distributed graph processing systems and their graph partitioning methods

表 5 分布式图处理系统图划分方式对照表

名称	边切分方式	点切分方式
Giraph	✔	✘
Pregel	✔	✘
Pregel+	✔	✘
GPS	✔	✘
PowerGraph	✘	✔
Hama	✔	✘
GraphHP	✔	✘
HybridGraph	✔	✔
MOCGraph	✔	✘
PowerSwitch	✘	✔
GraphSteal	✔	✘

边切分指在图的划分中对其边进行切分,图数据被划分后,图中每一个节点出现且仅出现在一个子图中,被切断边的两个顶点将出现在两个不同的子图中.该切分模式不仅简单而且节省存储空间,被多数基于 Pregel 的系统所采用;但是在该切分模式下,当对某一切分后的边进行操作时,因需要从不子图上获取其顶点信息,不可避免地增加网络开销.

与边切分对应,点切分是对图的节点进行切分.图数据被切分后,图中的每一条边出现且仅出现在一个子图中,邻居多的点将会出现在多个子图中.因在切分过程中部分节点被存储了多次,增加了系统的存储开销且需要一定的机制来保证被切分节点数据的一致性,增加了系统的设计复杂性.但是点切分方式使每条边仅出现在一个子图中,在需要对边执行大量操作的算法中,相较于边切分方式,该切分方式可以大幅度地减少网络通信,并且其在负载均衡方面也表现出了良好的性能[12].目前,采用点切分方式的系统主要有 PowerGraph、GraphX 以及 PowerSwitch 等系统.

4.2 图划分策略

目前,分布式图处理系统对图数据进行划分时主要采用了 3 种划分策略:离线划分策略、流式划分策略以及动态重划分策略.

4.2.1 离线划分策略

离线划分策略是使用离线图划分算法,在图数据被分布式图处理系统加载前将其划分为若干子图.启发式规则是离线图划分算法常用的策略,Kernighanhe 和 Lin 提出的 Kernighan-Lin(KL)[46]算法是一种典型的基于启发式规则的图划分方法.为实现图的二分划分,KL 引进了增益系数 Q,定义为两个子图内部边数目与两个子图之间边数目的差.KL 算法的基本思路是:首先,将图随机划分为两等份;然后,从中选择任意两个节点进行模拟交换并计算 Q 值的增加量;最后,选取使 Q 增加最多的模拟交换进行实际交换.KL 算法在有较好的初始化分时能够求得一个局部最优解,否则需要不断地迭代来对初始化分进行重划分.当图规模较大时,该算法会消耗大量时间来进行划分.为此,Fiduccia 提出了 Fiduccia-Mattheyses(FM)[47]算法对 KL 进行改进.FM 算法与 KL 算法最大的不同是:每次迭代划分,FM 算法只允许一个节点进行移动,并根据移动过程中割边减少的数目来确定划分代

价.实验结果表明,FM 算法在执行效率上优于 KL 算法.

为了对更大规模的图数据进行划分,Karypis 提出了多层图划分框架 METIS[48].在该框架中,将图划分细分为 3 个阶段:粗糙化阶段、初始划分阶段、反粗糙化阶段.粗糙化阶段,将大图归约成可以接受的小图;初始划分阶段,使用 KL、FM 等算法对规约后的小图进行划分;反粗糙化阶段,将小图划分还原成大图划分.该框架在大图划分方面有较好的时空复杂度.Chaco[49]、Scotch[50]等算法与 EMTIS 算法类似,均采用了集中式的多级划分策略.另外,METIS 的并行化版本 ParMetis[51]和 Scotch 的并行化版本 PT-Scotch[52]等进一步提高了算法的执行效率.

因图划分与社交网络的社区挖掘具有极高的相似度,Ugander 等人基于社区挖掘领域的标签传播算法来解决图划分问题[53].因标签传播算法不能有效地限制每一个子图的大小,Ugander 采用线性规划来约束每一个子图的大小实现均匀划分.借助标签传播的思想,Rahimain 等人设计了 JA-BE-JA 算法[54],其不仅可以与自己的邻居交换标签,还会随机地选取其他顶点进行标签交换,以减少子图之间的连通边.为防止陷入局部最优,JA-BE-JA 还使用了模拟退火算法的部分思想.同样,基于标签传播算法,Slota 等人提出了多目标划分方法 PuLP[55],其可以通过在不同阶段使用不同的约束条件来对划分进行调整,以满足实际应用中的多种需求.

另外,离线划分方法中比较有影响的方法还包括谱分解方法[56]、几何方法[57,58]以及数学规划[59]等方法,以及在其上的各种改进算法.

4.2.2 流式划分策略

流式划分策略是在数据加载过程中对图数据边加载边划分,其假定数据以节点流或者边流的方式到达,在划分过程中,按照已到达数据的分布信息,通过一组启发式的规则来决定当前到达数据的划分位置.与 METIS 等方法相比,流式划分方法只需要对数据扫描一遍即可实现数据划分,极大地提高了划分效率.但是因为在划分过程中只能够依据部分数据来决定当前数据的划分位置,牺牲了部分划分精度.现在常用的流式划分算法是 HASH 划分、FENNEL[60]和 LDG[61]算法.

Hash划分是最简单的图流式划分算法.系统从磁盘读取相应的图数据时,会根据图数据相应的Hash函数值来确定其所属的分区.Hash 划分方式简单易于实现,且不需要系统维护一张巨大的路由表来保存节点的分区信息,因此被 Pregel、Giraph 等系统作为默认的分区方式.但是它存在两方面的不足之处:(1) 在划分过程中没有考虑图的拓扑特性,完全打破了图的内在结构,导致在运算过程中通信代价过大;(2) 无法保证计算节点之间的负载均衡.

FENNEL 是 Tsourakakis 等人提出的一个采用贪心启发式规则的图流式划分框架,该框架对多种流切分方式建模,并通过大量的合成和真实数据集进行测试,调整模型中各个分量的参数,以此获得较好的切分效果.FENNEL 相对于离线划分算法 MEITS,可以在更短的时间内达到类似的划分效果.在 Tsourakakis 对 FENNEL 的测试中,其可以在 40 分钟内对包含有 14 亿条边的 Twitter[62]数据集完成均匀划分,划分效果甚至要好于 MEITS.因其优异的性能,PowerLyra[63]和 GraphLab 等分布式图处理系统在图划分时均采用了 FENNEL 技术.LDG 同样是采用了贪心启发式规则的流式划分算法,其在划分过程中首先计算待划分节点 v 的所有邻居节点与各个子图中已有节点的交集,然后将节点 v 划分到交集最大的子图,即,节点 v 属于哪一个分区是由其邻居顶点的分布来决定的.同时,LDG 允许用户自定义惩罚函数,来满足不同的划分需求.例如:为保证划分后各个子图大小均匀,可以根据子图的容量来调整启发式规则的计算结果,以改变图的划分方式.此外,Nishimura 等人提出了 restreaming 的流式分方法[64],当数据被反复加载处理时,可以利用前几次的流划分结果来提高本次的划分质量.实验表明,该机制可以有效提高图划分效果.

4.2.3 动态重划分策略

在分布式图处理系统的运行过程中,造成负载不均的因素有很多,包括初始图数据划分不均、集群环境的改变、活跃节点数目的变化等.为此,许多系统都设计了相应的动态重划分策略来保证负载均衡.图的动态重划分策略一般包含 3 个步骤:(1) 收集系统运行过程中的状态数据,例如活跃节点的数目、消息收发的数量等;(2) 根据收集到的状态信息建立相应的代价收益模型,制定动态重划分策略;(3) 根据制定的策略进行数据迁

移.根据重划分的粒度,动态重划分策略可以分为两类:以数据块为中心的动态重策略和以顶点为中心的动态重划分策略.

以数据块为中心的动态重划分策略,基本的调整单位是数据块.Giraph 系统[9]是采用该策略的典型系统.Giraph 在加载数据时,会将图数据流式划分为多个子图,子图数目往往是计算节点数目的整数倍,每个计算节点加载多个子图并建立子图与对应计算节点之间的映射表.在每次迭代结束前,Worker 节点都会向 Master 汇报本次迭代过程中每一个子图的状态信息,包括运行时间、收发消息的数目以及活跃节点数目等信息;在下次迭代开始前,Master 将根据收集到的信息制定相应的动态重划分调整策略,并以子图和计算节点映射表的形式告知 Worker;每个 Worker 节点根据调整策略发送或接收相应的子图.这种以数据块(子图)为中心的动态重划分策略划分粒度大易于实现,但是因调整粒度过大,不能对负载进行精细地调整.

以顶点为中心的动态重划分策略,基本的调整单位是图顶点及其对应的邻接边.Mizan 系统是采用了该策略的典型系统.Mizan 系统在加载完数据后需要构建图节点与对应计算节点的路由表,以方便消息的发送.Mizan 系统的动态重划分过程与 Giraph 类似,需要系统在迭代过程中不断收集执行状态信息.在当前迭代结束后,每一个 Worker 都要根据收集到的信息创建迁移计划,然后,Worker 之间根据一定的规则相互迁移节点,并更新对应的路由表.GraphSteal 同样采用了以顶点中心的动态重划分策略[19],不同于 Mizan、GraphSteal 在收集最近一次迭代运算状态信息时,不仅需要收集图数据的运行状态信息,还需要收集集群中每一个计算节点计算资源的状态信息,主要包括节点的内存使用/剩余量.GraphSteal 根据所收集的这些状态信息,将集群中计算节点分为两类:straggler 节点和 fast 节点,并将部分计算任务从 straggler 节点迁移到 fast 节点,以此实现在异质集群下的负载均衡.以顶点为中心的动态重划分技术可以实现细粒度的动态调整,但是图节点与计算节点的路由表的维护,为系统带来了额外的负担.

无论是以顶点为单位还是以分区为单位的动态重划分技术,都能够在特定场景下对系统的负载均衡起到积极的作用;但是在数据的迁移过程中,不可避免地带来了网络及时空开销.许多实验的结果都表明,动态重划分技术对系统性能的提高很小甚至没有贡献.Bao 和 Suzumura 在实验过程中发现:使用动态迁移技术虽然可以减少系统网络开销,但是在总执行时间上并没有实质性的提高[40];Han 等人在对 GPS 和 Mizan 等系统进行测试时得到了类似的结果[65].不过,这些实验使用的算法都是 PageRank,其在执行过程中所有节点的状态都是不变的;而动态重划分主要针对节点状态不断变化的程序设计的,因此,效果不明显也在情理之中.

5 总结与展望

本文首先总结出分布式图处理系统的 3 个优化目标,即减少迭代次数加快算法收敛、减少消息数目减轻网络负载、实现负载均衡消除木桶效应,然后从计算粒度、任务调度、通信方式、负载划分这 4 个维度,对现有分布式图处理系统中的各类优化技术进行了详细对比阐述.但受篇幅所限,不能将所有技术纳入本文进行综述,例如,本文不包括我们在快速容错方面的工作[66].表 6 展示了部分系统与其采用技术的对照关系.

Table 6 Distributed graph processing systems and their techniques

表 6 分布式图处理系及其采用技术对照表

名称	计算粒度	任务调度机制	通信方式	通信工作方式	图划分方式
Giraph	顶点	同步调度	基于消息	Push	边切分
Pregel	顶点	同步调度	基于消息	Push	边切分
Pregel+	顶点	同步调度	基于消息	Push	边切分
GPS	顶点	同步调度	基于消息	Push	边切分
PowerGraph	顶点	同步/异步调度	基于内存	Pull	点切分
Hama	顶点	同步调度	基于消息	Push	边切分
GraphHP	子图	混合调度	基于消息	Push	边切分
HybridGraph	顶点	同步/异步调度	基于消息/内存	Push/Pull	点/边切分
MOCGraph	顶点	同步/异步调度	基于消息	Push	边切分
PowerSwitch	顶点	同步/异步调度	基于内存	Pull	点切分
GraphSteal	顶点	异步调度	基于内存	Pull	边切分

虽然现有系统围绕着这 3 个优化目标采用一种或多种优化技术来对自身进行优化,但是它们往往都是在优化某个目标的同时牺牲了其他性能,因此未来分布式图处理系统的开发,仍将围绕着这 3 个优化目标而展开.

1) 加快算法收敛,减少迭代次数

图算法通常是由大量的迭代构成,减少运算的迭代次数,加快算法的收敛速度,可以极大地提高系统的执行效率.目前,在系统层面主要采用两种方式来减少运算的迭代次数:第 1 种,加大计算粒度,计算粒度的增加虽然可以加快算法收敛,但是牺牲了部分算法的表达能力;第 2 种,采用异步调度模式来加快算法的收敛速度,但是相较于同步模式的系统,采用异步模式的系统在开发和使用难度上都有所增加.因此,让系统提供灵活、可选的计算粒度和一套统一的同步/异步编程接口,将会极大地提高系统性能和编程灵活性.

2) 减少消息数目,减轻网络负载

在分布式处理环境下,制约系统性能的一个重要瓶颈就是网络负载,特别是当算法在执行 PageRank 这类通信密集的算法时,不合理的通信方式将会使得系统的执行效率大打折扣.目前,各个系统都采用了相应的策略来对网络通信进行优化,如 Combine、Receiver-side scatter 和 Request-Response 等技术.但是每种技术仅在特定的环境下可以有效减少网络通信量,并没有从根本上缓解网络负载这一瓶颈问题.例如,Request-Response 技术仅针对属性请求消息做出了优化,Combine 技术仅适合满足交换律和结合律的消息.如何对这些技术进行整合,让系统根据实际运算选择合适的优化方式,将会对系统性能的提高起到积极的作用.

3) 实现负载均衡,消除木桶效应

木桶效应是分布式图处理系统面临的一个严重问题,在同步调度模式下,该问题更为严重.如文中所述,高质量的图划分策略和动态重划分策略不仅可以大幅度地减轻网络负载,而且能够对系统的负载均衡起到积极的作用.但是在负载均衡上仍然存在两方面的问题:一方面,目前各图处理系统所采用的图划分策略难以在连通性、均衡性和时间复杂性方面同时取得较好的性能,且当集群中各个计算节点的内存、CPU 等计算资源不均时,高效的均衡图划分方法仍会导致系统的负载不均;另一方面,如文献[41,65]所述,虽然在特定场景下动态重划分技术能够保证系统负载均衡,但是网络开销的引入,使得系统的总体执行效率并没有得到实质性提高.因此在实现负载均衡方面,如何结合图数据的拓扑特性以及集群中计算资源分布设计高效的图分割策略与动态重划分策略,仍将是一个研究热点.

另外,在未来系统开发上也将考虑以下几个方面的工作.

(1) 对新硬件的支持

随着计算机技术的发展,GPU、多核心以及固态硬盘等硬件设备极大地改变了目前的计算环境.旧的计算模式、通信方式以及存储方法等已不能够充分发挥新硬件的性能,因此,如何在该环境下搭建新的高效统一的分布式图处理系统,将会成为一个重要的研究方向.

(2) 避免数据的重复加载

目前,大部分分布式图处理系统在对图数据进行处理时,都会为每一个批处理作业启动新的 job,该 job 涉及数据的加载、计算以及结果的输出.当同时提交多个批处理任务时,系统将会对数据进行重复加载,严重影响系统在单位时间内处理任务的数量.因此在未来研究中,设计一套合理的任务调度策略,在减少数据加载次数的同时又能够使各个作业之间共享中间结果,将会极大提高系统的吞吐量.

(3) 兼顾事务型查询

目前,图处理系统主要分为两类:批处理系统和事务型系统.两类系统的分离为图数据的管理带来了极大的不便.如何将两类系统进行整合,开发一套既支持事务型查询又支持批处理的系统,会对图数据的使用带来极大便利.

References:

[1] Google. How search works. http://www.google.com/insidesearch/howsearchworks/thestory/

[2] Edwards J. ‘Facebook Inc.’ Actually Has 2.2 Billion Users Now—Roughly One Third of the Entire Population of Earth. http://www.businessinsider.com/facebook-inc-has-22-billion- users-2014-7

[3] Pržulj N. Protein-Protein interactions: Making sense of networks via graph-theoretic modeling. Bioessays News & Reviews in Molecular Cellular & Developmental Biology, 2011,33(2):115−123. [doi: 10.1002/bies.201000044]

[4] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1): 107−113. [doi: 10.1145/1327452.1327492]

[5] Bu Y, Howe B, Balazinska M, Ernst MD. Haloop: Efficient iterative data processing on large clusters. Proc. of the VLDB Endowment, 2010,3(1-2):285−296. [doi: 10.14778/1920841.1920881]

[6] Ekanayake J, Li H, Zhang B, Gunarathne T, Bae SH, Qiu J, Fox G. Twister: A runtime for iterative MapReduce. In: Proc. of the ACM Int'l Symp. on High PERFORMANCE Distributed Computing. 2010. 810−818. [doi: 10.1145/1851476.1851593]

[7] Valiant LG. A bridging model for parallel computation. Communications of the ACM, 1990,33(8):103−111. [doi: 10.1145/79173.79181]

[8] Malewicz G, Austern MH, Bik AJ, 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. ACM Press, 2010. 135−146. [doi: 10.1145/1807167.1807184]

[9] Apache Giraph. http://giraph.apache.org/

[10] 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 International Conference on World Wide Web. 2015. 1307−1317. [doi: 10.1145/2736277.2741096]

[11] Salihoglu S, Widom J. Gps: A graph processing system. In: Proc. of the 25th Int'l Conf. on Scientific and Statistical Database Management. ACM Press, 2013. 22. [doi: 10.1145/2484838.2484843]

[12] Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C. PowerGraph: Distributed graph-parallel computation on natural graphs. In: Proc. of the Usenix Conf. on Operating Systems Design and Implementation. USENIX Association, 2012. 17−30.

[13] Apache Hama. http://hama.apache.org/

[14] Jing SU, Bo S, Chen Q, Wei P, Li ZH. GraphHP: A hybrid platform for iterative graph processing. Journal of East China Normal University, 2016 (5):112−120.

[15] Wang Z, Gu Y, Bao Y, Yu G, Yu JX. Hybrid pulling/pushing for I/O-efficient distributed and iterative graph computing. In: Proc. of the 2016 Int'l Conf. on Management of Data. ACM Press, 2016. 479−494. [doi: 10.1145/2882903.2882938]

[16] Xie C, Chen R, Guan H, Zang B, Chen H. SYNC or ASYNC: Time to fuse for distributed graph-parallel computation. ACM SIGPLAN Notices, 2015,50(8):194−204. [doi: 10.1145/2858788.2688508]

[17] Roy A, Mihailovic I, Zwaenepoel W. X-Stream: Edge-Centric graph processing using streaming partitions. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. ACM Press, 2013. 472−488. [doi: 10.1145/2517349.2522740]

[18] Yuan P, Zhang W, Xie C, Jin H, Liu L, Lee K. Fast iterative graph computation: A path centric approach. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2014). IEEE, 2014. 401−412. [doi: 10.1109/SC.2014.38]

[19] Kumar D, Raj A, Dharanipragada J. GraphSteal: Dynamic re-partitioning for efficient graph processing in heterogeneous clusters. In: Proc. of the 2017 IEEE 10th Int'l Conf. on Cloud Computing (CLOUD). IEEE, 2017. 439−446. [doi: 10.1109/CLOUD.2017.63]

[20] Khayyat Z, Awara K, Alonazi A, Jamjoom H, Dan W, Kalnis P. Mizan: A system for dynamic load balancing in large-scale graph processing. In: Proc. of the 8th ACM European Conf. on Computer Systems. ACM Press, 2013. 169−182. [doi: 10.1145/2465351.2465369]

[21] Xin RS, Gonzalez JE, Franklin MJ, Stoica I. GraphX: A resilient distributed graph system on Spark. In: Proc. of the Int'l Workshop on Graph Data Management Experiences and Systems. ACM Press, 2013. 1−6. [doi: 10.1145/2484425.2484427]

[22] Simmhan Y, Kumbhare A, Wickramaarachchi C, Nagarkar S, Ravi S, Raghavendra C. GoFFish: A sub-graph centric framework for large-scale graph analytics. In: Proc. of the European Conf. on Parallel Processing. Springer-Verlag, 2014. 451−462. [doi: 10.1007/978-3-319-09873-9_38]

[23] Zhou X, Chang P, Chen G. An efficient graph processing system. In: Chen L, *et al*., eds. Proc. of the Web Technologies and Applications, 2014. LNCS 8709, 2014. 401−412. [doi: 10.1007/978-3-319-11116-2_35]

[24] Balmin A, Balmin A, Corsten SA, Tatikonda S, Mcpherson J. From think like a vertex to think like a graph. Proc. of the VLDB Endowment, 2013,7(3):193−204. [doi: 10.14778/2732232.2732238]

[25] 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]

[26] Lynch NA. Distributed Algorithms. Morgan Kaufmann Publishers, 1996.

[27] Low Y, Gonzalez JE, Kyrola A, Bickson D, Guestrin CE, Hellerstein J. GraphLab: A new framework for parallel machine learning. In: Proc. of the 26th Conf. on Uncertainty in Artificial Intelligence. 2010. 340−349.

[28] Zhang Y, Gao Q, Gao L, Wang C. Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation. IEEE Trans. on Parallel & Distributed Systems, 2014,25(8):2091−2100. [doi: 10.1109/TPDS.2013.235]

[29] Page L. The PageRank citation ranking: Bringing order to the Web. Stanford Digital Libraries Working Paper, 1998,9(1):1−14.

[30] Zhou C, Gao J, Sun B, Yu JX. MOCgraph: Scalable distributed graph processing using message online computing. Proc. of the VLDB Endowment, 2014,8(4):377−388. [doi: 10.14778/2735496.2735501]

[31] Srinivasan R. RPC: Remote procedure call protocol specification: Version 2. Sun Microsystems, 2009,11(3):5531.

[32] Netty project. http://netty.io/

[33] Apache ActiveMQ. http://activemq.apache.org/

[34] Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing, 1996,22(6):789−828. [doi: 10.1016/0167-8191(96)00024-5]

[35] Protic J, Tomasevic M, Milutinovic V. Distributed shared memory: Concepts and systems. IEEE Parallel & Distributed Technology Systems & Applications, 2002,4(2):63−71. [doi: 10.1109/88.494605]

[36] Nitzberg B, Lo V. Distributed Shared Memory: A Survey of Issues and Algorithms. IEEE Computer Society Press, 1991. [doi: 10.1109/2.84877]

[37] Shao B, Wang H, Li Y. Trinity: A distributed graph engine on a memory cloud. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2013. 505−516. [doi: 10.1145/2463676.2467799]

[38] Dijkstra EW. Hierarchical ordering of sequential processes. Acta Informatica, 1971,1(2):115−138. [doi: 10.1007/BF00289519]

[39] Chandy KM, Misra J. The drinking philosophers problem. ACM Trans. on Programming Languages & Systems, 1984,6(4): 632−646. [doi: 10.1145/1780.1804]

[40] Apache spark. http://spark.apache.org/

[41] Bao NT, Suzumura T. Towards highly scalable pregel-based graph processing platform with x10. In: Proc. of the Int'l Conf. on World Wide Web Companion. 2014. 501−508. [doi: 10.1145/2487788.2487984]

[42] Han W, Miao Y, Li K, Chen E. Chronos: A graph engine for temporal graph analysis. In: Proc. of the 9th European Conf. on Computer Systems. ACM Press, 2014. 1−4. [doi: 10.1145/2592798.2592799]

[43] Xue J, Yang Z, Qu Z, Hou S, Dai Y. Seraph: An efficient, low-cost system for concurrent graph processing. In: Proc. of the 23rd Int'l Symp. on High-Performance Parallel and Distributed Computing. ACM Press, 2014. 227−238. [doi: 10.1145/2600212. 2600222]

[44] Beamer S, Asanovic K, Patterson D. Direction-Optimizing breadth-first search. In: Proc. of the Int'l Conf. for High PERFORMANCE Computing, Networking, Storage and Analysis. IEEE Computer Society, 2012. 1−10. [doi: 10.1109/SC.2012.50]

[45] Shun J, Blelloch GE. Ligra: A lightweight graph processing framework for shared memory. In: Proc. of the ACM Sigplan Symp. on Principles and Practice of Parallel Programming. ACM Press, 2013. 135−146. [doi: 10.1145/2517327.2442530]

[46] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal, 1970,49(2): 291−307. [doi: 10.1002/j.1538-7305.1970.tb01770.x]

[47] Fiduccia CM. A linear-time heuristic for improving network partitions. Papers on 25 Years of Electronic Design Automation. ACM Press, 1988. 241−247.

[48] Karypis G, Kumar V. METIS—Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. 1995.

[49] Hendrickson B, Leland R. The Chaco User's Guide Version 2.0. Technical Report SAND95-2344, Sandia Nat'l Laboratories, 1995.

[50] Pellegrini F, Roman J. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In: Proc. of the High-Performance Computing and Networking. Springer-Verlag, 1996. 493−498. [doi: 10.1007/3-540-61142-8_588]

[51] Karypis G, Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. Society for Industrial and Applied Mathematics, 1998. [doi: 10.1137/S1064827595287997]

[52] Chevalier C, Pellegrini F. PT-Scotch: A Tool for Efficient Parallel Graph Ordering. Parallel computing, 2008,34(6): 318−331. [doi: 10.1016/j.parco.2007.12.001]

[53] Ugander J, Backstrom L. Balanced Label Propagation for Partitioning Massive Graphs. ACM Press, 2013. 507−516. [doi: 10.1145/2433396.2433461]

[54] Rahimian F, Payberah AH, Girdzijauskas S, Jelasity M, Haridi S. JA-BE-JA: A distributed algorithm for balanced graph partitioning. In: Proc. of the Int'l Conf. on Self-Adaptive and Self-Organizing Systems. IEEE, 2013. 51−60. [doi: 10.1109/SASO.2013.13]

[55] Slota GM, Madduri K, Rajamanickam S. PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks. In: Proc. of the IEEE Int'l Conf. on Big Data. IEEE, 2015. 481−490. [doi: 10.1109/BigData.2014.7004265]

[56] Pothen A, Simon HD, Liou KP. Partitioning sparse matrices with eigenvectors of graphs. Siam Journal on Matrix Analysis & Applications, 1990,11(3):430−452. [doi: 10.1137/0611030]

[57] Berger MJ, Bokhari SH. A partitioning strategy for nonuniform problems on multiprocessors. IEEE Trans. on Computers, 1987, 36(5):570−580. [doi: 10.1109/TC.1987.1676942]

[58] Ou CW, Ranka S. Parallel incremental graph partitioning using linear programming. In: Proc. of the Supercomputing'94. IEEE, 1994. 458−467. [doi: 10.1109/SUPERC.1994.344309]

[59] Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs. In: Proc. of the IEEE/ACM SC'95 Conf. IEEE, 1995. 28. [doi: 10.1109/SUPERC.1995.242799]

[60] Tsourakakis C, Gkantsidis C, Radunovic B, Vojnovic M. FENNEL: Streaming graph partitioning for massive scale graphs. In: Proc. of the ACM Int'l Conf. on Web Search and Data Mining. ACM Press, 2014. 333−342. [doi: 10.1145/2556195.2556213]

[61] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs. In: Proc. of the KDD. 2012. 1222−1230. [doi: 10.1145/2339530.2339722]

[62] Kwak H, Lee C, Park H, Moon SB. What is Twitter, a social network or news media? In: Proc of the Int'l Conf. on World Wide Web. 2010. 591−600.

[63] Chen R, Shi J, Chen Y, Chen H. PowerLyra: Differentiated graph computation and partitioning on skewed graphs. In: Proc. of the 10th European Conf. on Computer Systems. ACM Press, 2015. 1−15. [doi: 10.1145/2741948.2741970]

[64] Nishimura J, Ugander J. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. In: Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. ACM Press, 2013. 1106−1114. [doi: 10.1145/2487575.2487696]

[65] Han M, Daudjee K, Ammar K, Wang X, Jin T. An experimental comparison of pregel-like graph processing systems. Proc. of the VLDB Endowment, 2014,7(12):1047−1058. [doi: 10.14778/2732977.2732980]

[66] Shen Y, Chen G, Jagadish HV, Lu W, Ooi BC, Tudor BM. Fast failure recovery in distributed graph processing systems. Proc. of the VLDB Endowment, 2014,8(4):437−448. [doi: 10.14778/2735496.2735506]

王童童(1989－),男,山东潍坊人,硕士,CCF 学生会员,主要研究领域为图数据存储与管理.

卢卫(1981－),男,博士,副教授,CCF 专业会员,主要研究领域为云计算与大数据管理,空间与文本数据库管理,索引技术.

荣垂田(1981－),男,博士,副教授,CCF 专业会员,主要研究领域为大数据云计算数据库信息检索.

杜小勇(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为数据库系统,智能信息检索.