|  |  |
| --- | --- |
| **学校代码** | **10699** |
| **分 类 号** | **TP311** |
| **密 级** | **公开** |
| **学 号** | **2012201676** |

题目 大规模数据并行图处理算法与计算平台研究

|  |  |
| --- | --- |
| **作者** | **尤立** |

|  |  |
| --- | --- |
| **学科、专业** | **计算机软件与理论** |
| **指 导 教 师** | **陈群** |
| **申请学位日期** | **2015年3月** |

# 

西北工业大学

硕士学位论文

(学位研究生)

题目：大规模数据并行图处理算法

与计算平台研究

作 者： 尤立

学科专业：计算机软件与理论

指导教师： 陈群

二零一五年三月

Northwestern Polytechnical University

A Dissertation Submitted for the

Master Degree

Title: Research of parallel graph algorithms over big data and processing platform

Author YouLi

Specialty Computer Software and Theory

Supervisor ChenQun

February 2015

# 摘要

近年来，大数据已经成为各大公司追逐的热点领域。随着大数据时代的到来，大量、多样、快速的数据给业界带来诸多挑战，同时也带来了巨大的价值和机遇。很多热门研究领域，如社交网络分析、Web文档聚类、实体识别、基因表达分析等，都必须对海量数据进行分析和挖掘。这些应用中通常都包含较复杂的网络结构，实际生产过程中，人们通常使用图来抽象系统中的各个实体之间的关系结构，并通过对抽象的图挖掘从数据中提取有价值的知识。在这些应用中关于图的完全图和近似完全图挖掘已经成为不可或缺的重要组成部分。

完全图和近似完全图是经典的NP Complete问题，算法时空复杂度都非常高。人们一直以来致力于找出此类问题的高效解决方法。过去的研究已经提出了一些在较小数据集上表现良好的算法。但是在大数据环境下，单机算法受制于内存和磁盘以及CPU的限制往往不能够满足实际生产需求。并行化成为一种必然的可行趋势，因此研究集群环境中大规模数据的并行图处理算法尤为重要。本文在充分了解现有的算法基础上，提出了一种基于图分割的完全图和近似完全图枚举算法（Binary），并分别在单机及并行环境中在多种实际和生成数据上，通过实验与经典算法（BK，Bron-Kerbosch）对比，验证了Binary算法的可行性及高效性。考虑到两个算法各自的优劣的基础之上，结合两者的优势提出了一种BK辅助Binary的Hybrid算法，在各种数据尤其是蛋白质交互数据上有更优的表现。

在并行环境中，本文选择利用当今流行的大数据计算框架MapReduce的开源实现Hadoop作为并行算法的运行平台。针对并行环境中完全图和近似完全图枚举过程中出现的负载不均问题，本文提出了一种负载均衡的算法。实验表明此负载均衡算法得到了较好的均衡性和扩展性。最后，本文进一步探讨了Hadoop系统中的负载均衡问题，描述了一种增量分区分配的负载均衡方案的实现以及其改进方案的设计。

**关键词：**完全图，近似完全图，并行算法，负载均衡，Hadoop

# Abstract

Big data has become a hot area for various companies in recent years. Though the three characters volume, variety, velocity raise great challenges, value and opportunities are also derived. Many popular research areas, such as social network analysis, web document clustering, entity recognition and gene expression analysis need to mining and analysis massive data. Typically, these applications deal with complex network structure. Graph is the data structure people used to abstracting relations between entities and also mining value from. Clique and K-Plex mining have become an indispensable part of these applications.

Clique and K-Plex mining are typical NP Complete problem known with high time and space complexity. Great effort has been committed to find efficient solution and several researches have shown good performance on small datasets. But under the circumstance of big data, these solutions are subjected to limit memory, disk and CPU in single machine. Parallelization become a feasible trend, so it’s particularly important to study parallel algorithms in large-scale cluster. In this paper, we fully study the existing algorithms and propose a maximal clique enumeration and maximal k-plex enumeration based on graph partitioning (Binary). Then we conduct both single and parallel experiments on variety of real data and synthetic graphs, our algorithm shows better performance and smaller search tree space then classical algorithm (BK, Bron-Kerbosch). Furthermore, we realize both algorithms have pros and cons. Combining benefits together, we present a BK-aid Binary hybrid algorithm (Hybrid). Hybrid performs better than both BK and Binary on a variety of data especially protein interaction data.

In parallel environment, we choose Hadoop which is an open source implementation of popular big data computing framework MapReduce. We propose a load balancing solution for both problem’s skew in parallel environment. Experiments shown that the load balancing solutions has achieved good balancing and scalability. Last but not least, we discuss general load balancing problem in Hadoop and propose an implementation of incremental partition scheduling strategy and the design of further improvement.

**Keywords:**Clique, K-Plex, Parallel algorithm, Load balance, Hadoop

# 目录

[摘要 I](#_Toc413173301)

[Abstract II](#_Toc413173302)

[目录 1](#_Toc413173303)

[第一章 绪论 1](#_Toc413173304)

[1.1 研究背景 1](#_Toc413173305)

[1.2 选题意义 2](#_Toc413173306)

[1.3 研究现状 3](#_Toc413173307)

[1.3.1 完全图单机算法研究现状 3](#_Toc413173308)

[1.3.2 完全图并行算法研究现状 4](#_Toc413173309)

[1.3.3 近似完全图单机算法研究现状 5](#_Toc413173310)

[1.3.4 近似完全图并行算法研究现状 6](#_Toc413173311)

[1.3.5 分布式计算平台Hadoop系统负载均衡研究现状 6](#_Toc413173312)

[1.4 论文的内容及组织 8](#_Toc413173313)

[1.4.1 主要研究内容及成果 8](#_Toc413173314)

[1.4.2 论文组织结构 8](#_Toc413173315)

[第二章 相关理论及技术 11](#_Toc413173316)

[2.1 图论相关概念 11](#_Toc413173317)

[2.2 完全图和近似完全图 12](#_Toc413173318)

[2.2.2 完全图相关概念 12](#_Toc413173319)

[2.2.3 近似完全图相关概念 13](#_Toc413173320)

[2.3 并行计算模型MapReduce 14](#_Toc413173321)

[2.4 开源分布式计算平台Hadoop 16](#_Toc413173322)

[2.5 负载均衡 17](#_Toc413173323)

[2.6 图数据表示方式 18](#_Toc413173324)

[2.7 本章总结 19](#_Toc413173325)

[第三章 完全图和近似完全图枚举算法 20](#_Toc413173326)

[3.1 完全图枚举算法 20](#_Toc413173327)

[3.1.1 极大完全图BK算法 20](#_Toc413173328)

[3.1.2 极大完全图Binary算法 23](#_Toc413173329)

[3.1.3 极大完全图Hybrid算法 27](#_Toc413173330)

[3.2 近似完全图枚举算法 29](#_Toc413173331)

[3.2.1 基于BK的K-Plex算法Pump 30](#_Toc413173332)

[3.2.2 基于Binary的K-Plex算法 32](#_Toc413173333)

[3.3 算法并行化 34](#_Toc413173334)

[3.3.1 输入图数据 35](#_Toc413173335)

[3.3.2 初始任务分配 36](#_Toc413173336)

[3.3.3 负载倾斜 37](#_Toc413173337)

[3.4 本章总结 38](#_Toc413173338)

[第四章 并行环境下的负载均衡 39](#_Toc413173339)

[4.1 完全图和近似完全图枚举的负载均衡 39](#_Toc413173340)

[4.2 并行平台Hadoop中通用的负载均衡 42](#_Toc413173341)

[4.2.1 负载均衡问题讨论 43](#_Toc413173342)

[4.2.2 Hadoop静态增量分区分配负载均衡 44](#_Toc413173343)

[4.2.3 Hadoop静态负载均衡存在问题及解决思路 46](#_Toc413173344)

[4.3 本章总结 48](#_Toc413173345)

[第五章 实验部分 49](#_Toc413173346)

[5.1 实验条件及数据 49](#_Toc413173347)

[5.1.1 实验条件 49](#_Toc413173348)

[5.1.2 实验数据 49](#_Toc413173349)

[5.2 单机算法实验及分析 50](#_Toc413173350)

[5.2.1 极大完全图枚举 50](#_Toc413173351)

[5.2.2 极大K-Plex枚举 52](#_Toc413173352)

[5.2.3 分裂点选择 54](#_Toc413173353)

[5.3并行算法实验及分析 54](#_Toc413173354)

[5.3.1 并行极大完全图枚举 54](#_Toc413173355)

[5.3.2 并行极大K-Plex枚举 56](#_Toc413173356)

[5.3.3 并行算法加速比 57](#_Toc413173357)

[5.4 Hadoop系统负载均衡实验及分析 57](#_Toc413173358)

[5.5 本章总结 58](#_Toc413173359)

[第六章 总结与展望 59](#_Toc413173360)

[6.1 本文研究总结 59](#_Toc413173361)

[6.2 课题研究展望 59](#_Toc413173362)

[参考文献 61](#_Toc413173363)

[致谢 64](#_Toc413173364)

第一章 绪论

1.1 研究背景

随着互联网尤其是移动互联网的快速发展，社会化交际网络的快速普及，物联网、商业销售、Web语义分析、生物网络信息等相关应用的丰富，云计算相关技术的快速发展，更多的设备被添加到网络中，数据资源呈指数增长。正如人们所说：大数据时代到来了。在大数据时代，对于企业来讲最重要的是获得和使用数据，能否从现有的海量数据中快速、高效、准确地分析提取有用信息已经成为企业能否在新环境下取得成功的关键因素。早在2010CNNIC统计时，仅中国的网页规模就已经达到600亿，且还在以每年78.6%的比率增长；Web搜索领域Google索引了全球超过500亿的网页，每天有20PB数据要处理；同时社交网络的后来居上，Facebook已经超过7亿用户[39]；在环境和气象领域，需要从数以万计的传感器中接受大量数据以进行监控和预测。正是有如此庞大和迫切的需求，快速高效的海量数据处理日益受到学术界和科技界的关注。

在各种各样的海量数据应用中普遍存在网络连接结构，如Web中网页的链接关系，社交网络中人与人之间的好友关系，基因表达数据中各个基因之间的协同表达关系等。而图正是数学上描述各个实体之间的网络关系的一种经典结构，如此总总应的用都可以抽象成关于图的计算。

图论是数学的一个重要分支，早在1736年数学家欧拉关于柯尼斯堡七桥问题的论著中就有关于图论的研究记录，这个问题后来被推广为著名的欧拉回路问题。经过二百多年的发展，图论已经成为一个独立的数学研究分支。图论中的问题主要可以概括为子图问题、染色问题、路径问题、网络流与匹配问题以及覆盖问题。这些问题里各自出现了许多经典的代表，如哈密顿回路、最大团、四色问题、斯坦纳树、最短路径、中国邮递员问题、最小覆盖集等，也出现了许多如戴克斯特拉算法、克鲁斯卡尔算法、拓扑排序等等经典算法。然而经过研究证明，很多图相关的问题都是NP或者NP Complete问题，算法有很高的时间和空间复杂度。在过去的现实需求中数据量较小，传统算法可以具有较好的实用性。但在大数据时代，数据的采集、获取变得非常方便，数据中所蕴含的科学与商业价值推动着对于海量数据的图处理需求迅速提高。传统的单机算法受制于内存和磁盘的限制，尽管不断优化，还是不能够在有意义的时间内完成处理。同时现有的多机并行算法通常利用冗余来保证计算的完整和正确，很难达到较高的并行度，或者在计算过程中带来严重的数据膨胀问题。

1.2 选题意义

随着图在社交网络、生物信息、实体识别等领域广泛应用，业界对高效的图处理算法有着迫切的现实需求。完全图和近似完全图常常用于进行分类和聚类分析，例如在社交网络中，通过检测完全图和近似完全图，从而找到关系十分紧密的群体，群体之间往往具有相同的兴趣爱好或者消费习惯，因此在个性化服务和投放广告方面能做的更加地准确有效，从而增加用户的粘性，提升网站的流量和商业价值都；在生物信息领域，不同基因之间的协同表达是一个重要的研究问题，常常作为新型药物研制的突破口，是生物工程中常用的技术，完全图和近似完全图的检测则是分析基因协同表达中的重要技术，这些技术的应用能够有效缩短新药的研制时间和成本。

在大规模数据的背景下，TB级甚至PB级的大规模数据是比较常见的情况，这些问题难以在单机上进行计算，并行算法成为必然选择。并行算法的过程中经常会出现数据不均衡或者计算不均衡的情况，严重制约平台的吞吐量和计算效率，负载均衡已经成为并行和分布式计算平台研究中最重要的方面之一。

尽管对于完全图与近似完全图的枚举、并行环境的负载均衡等问题现在已经有许多优秀的算法和方案[3][12][14][16][18][20][33]，然而都存在性能瓶颈和难以适用情况，又或者是针对某些具有特定特征的数据定制，没有较好的通用性。文献[1]是Bron和Kerbosch完全图枚举的经典算法（BK），在BK的基础上发展、衍生了一系列算法，文献[3][5]通过实际应用显示BK算法的效率要普遍好于现有的其他算法。然而在本文的研究过程中发现BK算法在许多数据上依然存在搜索空间大，速度慢等等问题。文献[2][4]等并行完全图算法虽然一定程度上缓解了单机计算能力的不足，但存在中间数据过度膨胀，冗余状态量大，需要通过大量的机器数目来存储和计算，资源利用效率低。另外现有的分布算法并行程度低，且当出现负载不均衡的情况，如社交网络数据中少数名人的好友关系非常大，而大部分普通用户关系简单，这种差异会导致系统少数机器有性能瓶颈从而影响整体执行效率，降低系统吞吐量。现有的近似完全图算法一般是借鉴完全图的解决方案稍作调整，但是目前还没有很好的并行算法。已有的近似完全图并行算法[6][7]是在特定的限制环境下的特殊解决方案。同时在近似完全图中同样也存在和完全图枚举过程中的负载不均衡问题。并行完全图和近似完全图枚举问题中所共有的负载不均衡的问题通常有两种解决思路：根据特定问题（本文中即完全图和近似完全图枚举）设计方案，在各个机器节点间互相分担高负载节点的任务量；对计算平台做相应修改，使计算平台本身可以达到自动均衡负载的功能。第一种方案具有简单、易实现的优势，但是需要设计者对输入数据特征和特定算法有较好的了解，而且这样的算法并没有通用性，需要为每个算法重新设计均衡策略；第二种方案，从平台上解决了负载均衡的问题，可以有较好的通用性，只要任务是可以切分的，系统可以自动去均衡各个机器节点的负载，而无需用户介入，缺点是实现比较复杂。

完全图枚举和近似完全图枚举都是经典的NP Complete问题一个问题的有效解决可以对其他NP Complete问题有借鉴意义，同时本文所使用的并行环境下的负载均衡方案既有针对完全图和近似完全图的特定应用方案，也有通用的Hadoop系统中负载均衡问题解决方案，可以用来解决分布式计算平台Hadoop中大部分负载不均衡问题。综上所述，本文的选题具有较大的研究意义和现实意义。

1.3 研究现状

本文主要涉及图论中的两个典型问题极大完全图枚举（Maximal Clique Enumeration，MCE）和极大近似完全图枚举（Maximal Qusi-Clique Enumeration）。这两个问题自提出以来就已被广泛研究，各有单机和并行算法两类算法。分布式计算平台Hadoop上的负载均衡问题近几年也有不少研究成果，下面将从这五方面介绍现有的研究现状做介绍。

1.3.1 完全图单机算法研究现状

完全图中的所有节点都与除自身外的其他节点相邻。在实际应用过程中尤其是计算科学相关领域中最有价值的是从杂乱无章的大量数据图中找出所有的极大完全图。这个问题也因Luce和Perry在社交网络分析中使用Clique表示团体中每一个人都认识其他所有人也成为极大完全图枚举（MCE）。

抽象的图论算法中有多种极大完全图枚举算法，以及一些与之相关的数学性质。这些算法主要分为三类：

第一类是以Bron-Kerbosch为典型代表的基于回溯搜索并应用剪枝算法降低搜索空间。这类算法[1][15]利用完全图必然是邻接点子集的特征，算法每次从候选点集中选取一个节点与现有的节点构成完全图，直到候选节点中不再有点可以与现有结果组成更大的完全图，最后再递归回溯搜索。

第二类类算法[9][11][13]使用反向搜索策略，这类算法的主要特征是可以较容易地得到算法关于结果集中完全图个数的复杂度上界。

第三类算法是基于“组装”的算法[21][22]，将简单的子图自动组装拼接成更大的单元，最终形成极大完全图。该算法首先将图分解，分解过程中如若产生完全图则直接输出，其他的子图按照特定的方式组装若能组装成完全图则输出。

另外还有如Thang Nguyen Bui等提出的基于遗传编程算法的完全图检测[17]。算法将网络连接关系图用树的形式表示，完全图即从根到叶子节点的路径上所涉及到的点集，找出一个点的邻接节点非常方便。根据完全图的性质结合遗传编程的特点可以较快的剪去许多不符合条件的节点，减少候选节点数量提高搜索速度。同时由于图以树的形式表示可以借鉴许多在树上研究得较为成熟的技术。James Cheng等根据大规模图数据中完全图检测的前提下提出一种优化的存储结构H\*--graph[19]，该结构定义图的核心部分及核心部分所邻接的节点。将其他节点数据存在外存磁盘中，可以有效的限制大规模数据的情况下单机完全图检测过程中对内存的消耗。同时H\*--graph只保存了数据图中的核心部分，数据的更新代价大幅降低，避免了传统算法中数据更新时需要重新计算的缺点。

在图论领域有不少关于完全图的性质可以应用到实际的完全图检测算法中。Turon指出，如果一个图有足够多的边，那么它一定包含一个最大的完全图；若一个图中包含条边，则必然包含一个三个节点的完全图。Moon和Moser指出，在一个包含3n个节点的图中，至多包含个最大完全图。拉姆齐理论指出每个图或者其补图包含一个至少有对数个节点的完全图。

1.3.2 完全图并行算法研究现状

随着近十几年互联网、生物信息学、社交网络、移动互联网等众多新兴技术的迅速发展，数据的规模变得越来越大。为了适应变化的需求，原本在单机环境中的完全图检测算法也在近些年被人们扩展到并行环境中。这些并行完全图检测算法[23][24][25][26][27]的基本思想都是将图或者搜索空间切分为独立的、较小的子图或子空间，然后对于这些切分出来的子图或子空间将它们分散到多个机器节点上各自进行计算，以提高整体的计算效率。

Y. Zhang等人首先提出了并行MCE算法pCLique [44]，这个算法是对Kose等工作[45]的一个扩展，也被成为KOSE。实际上KOSE在思想上和BK算法相同，不过与BK深度优先搜索的方式不同，KOSE使用的是广度优先遍历。这种方法可以和经典的关联挖掘算法Apriori一样可以通过大小为K-1的完全图生成大小为K的完全图，生成的Clique可以按非递减的次序出现。然而KOSE的广度优先使得它变得内存敏感，尽管pClique使用了位图向量来操作公共邻接点，但还是需要大量的内存消耗。

S Szabo提出一种基于分解邻接矩阵的并行完全图检测算法[29]，该算法充分利用完全图的性质将邻接矩阵不断切分的数据单元，并将其分散到不同的节点上计算。

另外一种并行MCE算法Peamc是Du等人在[46]中提出的。由于Peamc是一种基于串行的深度优先搜索，Peamc中没有pClique中出现的内存消耗的负担。Peamc使用了一种简单的并行策略，将节点分为多个不想交的独立集合发散到各个计算单元上并行计算，但并没有很好的均衡策略。

Matthew C.Schmidt等人提出一种基于多核并行的完全图检测算法[28]。该算法使用共享内存来存储全局信息，每个核负责处理部分图数据，各个核将处理出来的结果汇总合并成为最终结果。此算法可以处理较大数据量的图且可以通过核数量的扩展有接近线性的并行性，但是对系统的硬件性能有很高的要求。

之前的算法大多运行在多核机器或者MPI平台上，自从MapReduce计算模型的开源平台Hadoop兴起后，其良好的扩展性、稳定性，适应于异构集群、普通计算节点的特性带来了一波新的浪潮。Shengqi Yang等首先提出了使用MapReduce模型在Hadoop平台上进行完全图检测的并行算法[4]。该算法首先提取数据图中每个节点的两跳邻接点（相邻节点以及相邻节点的相邻节点），并将其发往不同的计算节点进行计算，各个计算节点上使用传统单机算法进行检测。

1.3.3 近似完全图单机算法研究现状

近似完全图相对完全图而言其对图的特征要求相对较低，因此在现实生产环境中有着更加广泛的应用。对于近似完全图的研究一般是完全图的一种变形，一遍的完全图算法都可以稍加修改，减弱限制条件使之成为近似完全图检测的算法[34]。近年来，由于社交网络如Facebook、Twitter等推动，近似完全图在这种场景下对于公司的广告投放、社区发现有着重要的指导依据，文献[30][31]对此有详细阐述。近似完全图在实际应用中一般都有个条件限制（结果集大小超过一定限值或者最多与k个顶点不相邻即K-Plex[34]），由于这些条件有确确实实的意义，如低于4个人以下的近似完全图基本没有实用价值，因此本文也采用这样的限制条件。现在已经有不少研究关注近似完全图的挖掘算法，除去对完全图的BK算法做相应改进的方法外，有下面这些典型的算法和优化方案：

Guimei Liu等提出了使用度数过滤数据图中不可能满足近似完全图条件的节点，同时提出了近似完全图挖掘中的多种优化方案，包括关于图直径、设定最小节点数目阀值、子图中近似完全图能够添加的节点上下界的优化等等[32]。

James Abello等提出了一种先期在外存中使用广度优先搜索，限制输入子图的大小，使得原本由于数据量巨大而不能完全放到内存中图数据可以在有限的内存条件下完成近似完全图检测的工作。算法在对内存中的节点的边进行搜索，同时应用一些剪枝条件和优化策略，使得处理大规模数据成为可能。算法还利用随机适应性贪心搜索的方法通过抽样每次选取可以放入内存的最小子图进行计算[33]。

与完全图检测相比而言，近似完全图由于其限制条件较低，其搜索复杂度、搜索空间以及结果数量都较完全图更大。在大数据环境下，单机算法面临着更加严重的性能问题。

1.3.4 近似完全图并行算法研究现状

近似完全图的并行算法目前还不是很多，主要有两种。

第一种是Bin Wu等在多核并行环境下并行算法优化。整体上该算法同时使用经典算法的主体结构，每个搜索树节点包含三个集合：Res，Cand, Not，分别表示已经形成K-Plex的节点集，可扩展的候选节点集，之前已经扩展过不再需要扩展的节点集。文章中提出两种优化方案：若是一个K-Plex，如果中节点数目大于()，则图的直径小于等于；若极大K-Plex节点数目大于，如果包含节点，则必然在()跳之内[35]。

第二种是Arash Khosraviani等提出的在MapReduce并行框架中做近似完全图检测，这也是首次提出在MapReduce平台中进行相关工作。算法将数据图分解为各自独立的子图，使用MapReduce框架将数据发散到各个机器节点上各自做检测，这样可使得算法可以获得较好的扩展性。算法在分解数据图的过程中使用了数学特性限定的，这样对于所有的搜索工作只需要在两跳数据集里面进行搜索即可。这样的限制使得算法不能够有效检测到较小或者节点数目较少的近似完全图，但是由于这些不能检测到的图基本没有使用价值，因此该优化假设是可以利用的[36]。

现有的并行近似完全图枚举算法首先没有从单机算法上将计算时间减下来，同时还没有能够较全面的考虑并行环境中的负载不均问题。本文从这两方面都做了优化工作。

1.3.5 分布式计算平台Hadoop系统负载均衡研究现状

本文中所涉及到的并行图算法处理工作都是在MapRedue模型的Hadoop平台上完成的，因此本文探讨的平台负载均衡也是针对Hadoop系统而言。

对于在分布式计算平台Hadoop上进行完全图和近似完全图枚举的负载不均衡问题本文首先采用了针对这两种算法特定场景下设计的均衡策略。同时注意到这种方案并不能够通用地解决在Hadoop平台上使用MapReduce模型进行大规模数据条件下图数据并行处理问题中的负载不均问题。因此本文进一步探讨如何在Hadoop平台上实现一种通用的负载均衡机制。

MapReduce模型将计算分为Map和Reduce两个阶段，一般情况下Map主要对数据进行分区操作，Reduce完成对分区的计算。Hadoop系统默认采用的是Hash分区法，同时支持Range和用户自定义分区法，但采用的都是一次分区机制，即对元组仅进行一次划分，并且采用分区与Reduce一一对应的随机指派策略。对于均匀分布的数据集，该分区机制能很好地实现各Reduce接收数据的均衡性，但对于个别值密集的倾斜数据，默认分区方法很难实现对数据的一次性均匀划分。一旦发生数据的划分倾斜，势必会造成Reduce运行的不均衡，从而影响整个作业的运行时间。因此如何保证分区后数据的均衡性，成为近年来研究的热点。

为解决原MapReduce自身一次分区生成机制带来的弊端，研究者们提出了两阶段分区机制，即首先按照原机制生成数据分区，然后对数据量发生倾斜的分区进行一次调整。分区调整策略如文献[37]中的方法：在Map运行到一定时机时，根据采样所得到的分布信息对发生数据量倾斜的分区进行一次拆分，然后在保证数据一致性的基础上将拆分后的分区与较少数据量的分区进行合并，从而产生均衡分区。该方法的关键是何时对分区进行拆分，较早对分区拆分势必会增大采样误差，而较晚调整又会延迟数据从Map端到Reduce端的传输时机，对于不同的数据分布，该方法很难给出一个最优的调整时机。文献[20]提出了一种预处理分区调整策略，即在运行用户的作业前，启动一个采样任务负责调整默认分区策略，以此保证使用均衡的划分策略来运行用户的作业。由于一个作业的启动和停止需要较大的开销，对短作业和数据集经常发生变化的作业，该方法往往不能取得最优的运行效率。另外，这种方法需要多一次对于输入数据集的访问，这也就增大了系统的文件访问和数据传输开销，在繁忙的集群中会带来集群资源的浪费。由此可见，一次分区生成和一次分区调整机制，在处理分布类型复杂的各类数据集时往往存在一定的局限性。

Hadoop上的负载均衡工作还有一类是基于动态调整策略的。华盛顿大学YongChul Kwon等人提出了一种基于负载切分转移的动态均衡方案--SkewTune[38]。任务的划分仍按照Hadoop默认的方式，负载均衡工作时机发生在当检测到系统中有Slot任务完成时。系统使用代价估计模型首先从本地现有的任务中取出一个最繁重的任务，当本地没有需切分的任务时会通过Master节点分析从其他节点选取最繁重的任务，这个选择出来的任务称为Straggler。系统之后通过对Straggler的任务划分，将其中一部分任务转移到出现空闲的Slot中运行，以此达到负载均衡的目的。

1.4 论文的内容及组织

1.4.1 主要研究内容及成果

本文主要关注大规模数据环境下并行图处理算法的研究以及并行计算平台中的负载均衡问题。主要包括完全图枚举、近似完全图枚举和Hadoop系统的负载均衡问题，期望能够为并行环境中高效解决图处理算法和良好的系统均衡提供思路和经验。

本文分析了现有的经典完全图和近似完全图枚举算法的优劣，提出了一种新的完全图枚举和近似完全图枚举切分算法Binary。通过实验证明新算法在大多数据情况下其搜索空间以及搜索效率都优于现有算法。

在分析Binary和经典算法BK各自优势的基础上结合两者的特征将两种算法结合，提出一种BK辅助Binary的Hybrid算法，该算法兼具Binary和BK的优势。经验证Hybrid算法在所有实验数据集上都有最优的表现。

契合当前大规模数据的处理需求，结合之前提出的单机算法，本文提出了基于并行计算平台Hadoop的Binary的并行算法。通过实验证明了并行Binary算法的并行性、高效性。

最后由于在并行计算平台Hadoop上实现BK和Binary的并行图算法都会涉及到负载不均衡问题，本文从图算法本身和计算平台两个方面提出了负载均衡方案。

1.4.2 论文组织结构

本文的组织结构如下：

第一章是绪论部分，从整体上介绍本文的研究背景，主要介绍了完全图枚举、近似完全图枚举和负载均衡三个问题的研究现状，最后简要介绍了本文的研究内容和研究成果。

第二章是相关理论及技术介绍，包含完全图、近似完全图、负载均衡相关概念，MapReduce并行计算框架和其开源实现Hadoop。

第三章对完全图和近似完全图枚举算法进行了深入分析，选取最具代表性的经典算法做为对比实验。分析了现有算法的优缺点。提出了一种新的完全图和近似完全图算法。并结合两种算法各自的优劣提出一种可以充分发挥两者优势的Hybrid算法。

第四章在第三章的基础之上论述了这几种算法的并行实现。分析了算法并行化实现过程中发现的负载均衡问题，提出了针对完全图和近似完全图算法的负载均衡方案。最后进一步深入探讨了并行平台中的负载均衡问题，提出渐进增量分区分配的通用负载均衡方案，分析其优缺点并提出一种任务主动获取的动态均衡方案。

第五章是论文的实验部分。本章选取了一系列真实数据和生成数据通过多种维度的实验验证了本文提出的算法的高效性。同时通过实验验证了负载均衡方案的有效性。

第六章是对本文工作的总结以及未来工作的展望，总结全文的主要内容以及今后可以改进的地方及提升的空间。

第二章 相关理论及技术

本文的主要研究对象时图论中经典的两个问题—完全图和近似完全图以及Hadoop系统的负载均衡。两个图算法中研究者们主要关注的方面是如何从原始数据图中找出所有的且最大的完全图或近似完全图，也就是通常所说的极大完全图枚举（Maximal Clique Enumeration， MCE）和最大近似完全图枚举（Maximal Qusi-Clique Enumeration，也成为K-Plex）。本章将本文涉及到的如图论相关概念、完全图、近似完全图、并行计算模型MapReduce、开源平台Hadoop以及负载均衡等相关技术作简要介绍。

2.1 图论相关概念

作为数学的一个重要研究领域，图论自1736年欧拉在著作中首次研究柯尼斯堡七桥问题开始经过上百年发展现在已经形成了超图理论、拟阵理论、拓扑图论和代数图论等分支，并广泛应用于计算机科学、心理学、商业挖掘、运筹学等等方面。

图论的主要研究对象时图。图是由顶点的有穷非空集合和顶点之间的边的集合组成，通常表示这样一个二元组：，其中，G表示一个图，V是图G中顶点的非空有限集合，E是图G中边的可空有限集合。图中的边是关于节点的一个二元组()，边的两个元素a和b之间可以是有序的也可以是无序的。当边的元组之间无序时将边称之为无向边，相似地边的元组间有序时称边为有向边。由有无向边组成的图为无向图，由有向边组成的图称之为有向图。本文根据实际的应用需求以及普适实用的研究意义将研究对象限定为无向图，如无特别说明在正文中所提到的相关图都是无向图。下图2-1中(a)表示一个有向图，(6)表示一个无向图。

图论中两点之间距离并不是指从一个点到达另一个点的路径中边的权值和最小值，而是指的图的直径是指图中任意两个顶点之间可以到达的路径中所经过的最少节点个数。图的直径值图中任意两点之间的距离的最大值，如图2-1 (b)中图的直径是1。

在有向图中，以节点为始点的边个数称为的出度，以节点为终点的边个数称为的入度，他们的和称为的度。无向图中，与节点相邻的边个数称为的度，记为。给定两个图，，若且，则称图是图的子图。

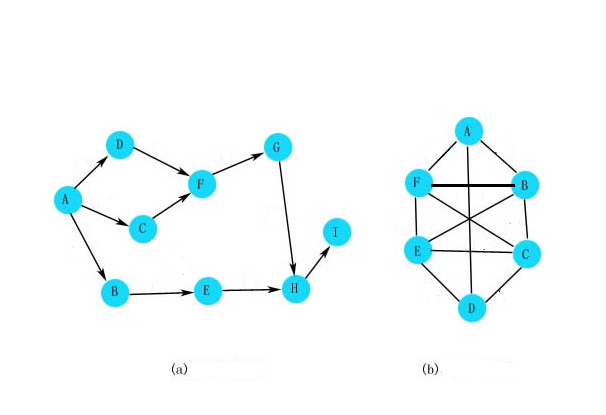


图2-1 有向图和无向图

2.2 完全图和近似完全图

本文的主要研究对象完全图和近似完全图都是图的一种稠密子图，下面将介绍完全图和近似完全图的相关定义和性质。

2.2.2 完全图相关概念

完全图指图中任意两个节点之间都有边相连，在有个顶点的完全图中应有条边，也称这个图是完全的。完全图是它本身的团（Clique）。在上图2-1中B、C、E、F四个点组成的子图就是一个完全图。图2-2直观上展示了-的完全图。

关于完全图有两个相似的容易混淆的概念，极大完全图（Maximal Clique）和最大完全图（Maximum Clique）。一个完全图称为极大完全图指不能够通过从这个图的邻接点中扩展出一个新的更大的完全图，也就是说这个完全图不是其他任意一个完全图的子图。最大完全图指一个图中大小最大的一个完全图，也就是说完全图的顶点数目大小在这个图的所有完全图中是最大的。指定一个顶点，包含顶点的极大完全图可以有很多个，但是包含顶点的最大完全图只要找到最大一个即可。在图2-1中对于点C来说，C、D、E是三个点的一个极大完全图，但是最大完全图是C、B、E、F。

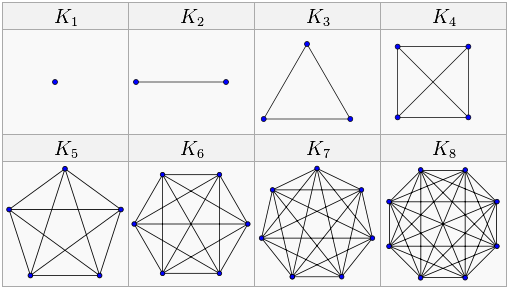


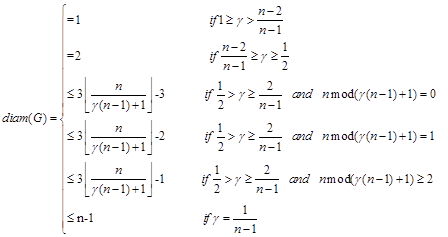
图2-2 完全图

2.2.3 近似完全图相关概念

近似完全图也是一种稠密图，但是可以容许图中可以有部分点之间不相邻。由于完全图对图的连接结构要求非常严格，现实场景中数据很难达到完全图的标准。近似完全图由于具有相似的特性但是要求不那么严格，在现实环境中的应用会更加普遍。一个近似完全图是图的一个子集，包含至少条边，其中。我们称quasi-clique为，其中参数用于表示近似完全图的疏密程度，越小，近似完全图中平均每个节点的邻接点个数越少，也就是说，近似完全图更加稀疏；反之，平均每个节点邻接的点越多，近似完全图更加稠密。特别地，当时，即上一节中提到的完全图。

由于近似完全图要求较松，粗糙的定义给实际挖掘工作带来了较大的麻烦。在实际生产应用中我们通常对和的值进行限制。若节点数目过少，挖掘出来的近似完全图应用价值不高，通常我们会限制。鉴于多数现实应用意义本文中的算法同样使用了这个限制条件。

图论中还有另外一个度量图的密集程度的方式，即图的直径。文献[40]中指出与之间的关系，具体如下面公式2-1所示，其中n为图中节点的个数。由公式2-1可知，为了满足的条件，只需保证即可。因此，在近似完全图的搜索过程中，可以只使用两跳数据集，而无需全部图数据，这从一定程度上减少了搜索过程的时空代价。



公式 2-1

通常对于近似完全图，还有一种较为简洁的表示方法，K-Plex。一个近似完全图K-Plex定义为包含N个节点的图中，每个节点最多与（N-K）个节点不相邻（节点认为与自身不相邻）。当K=1时，所有节点只与自身不相邻，因此就是完全图。实际上，完全图是近似完全图的一种特殊情况。

2.3 并行计算模型MapReduce

随着计算规模的不断扩大涌现了许多优秀的并行计算模型，MapReduce、BSP(Bulk Synchronous Parallel Computing)、LogP等。其中MapReduce模型以其简洁性和通用性获得了长足的发展。MapReduce[41]是Google的J Dean等提出的一种函数式编程模型，可以使得程序员方便地使用简单的函数实现完成复杂的分布式并行计算。其设计主要包括两个模块，分布式存储\*DFS和MapReduce计算框架。

MapReduce框架将计算过程分为Map和Reduce两部分。Map过程称为映射，映射过程就是对逻辑上一致的列表中每个独立元素进行指定的操作，每个元素的操作是独立的，输出使用新的列表保存操作结果。由于操作并没有改变输入列表，Map是可以高度并行的，这对并行计算领域中高性能和高并行度的需求非常有用。Reduce过程称为化简，化简过程是对输入列表中的元素进行适当地合并，归约出最终结果[42]。

在MapReduce模型中输入文件一般是存储在分布式文件系统中的，文件在分布式文件系统中被切分为同样大小的一块块的分片Split，每个Split都有多个备份。

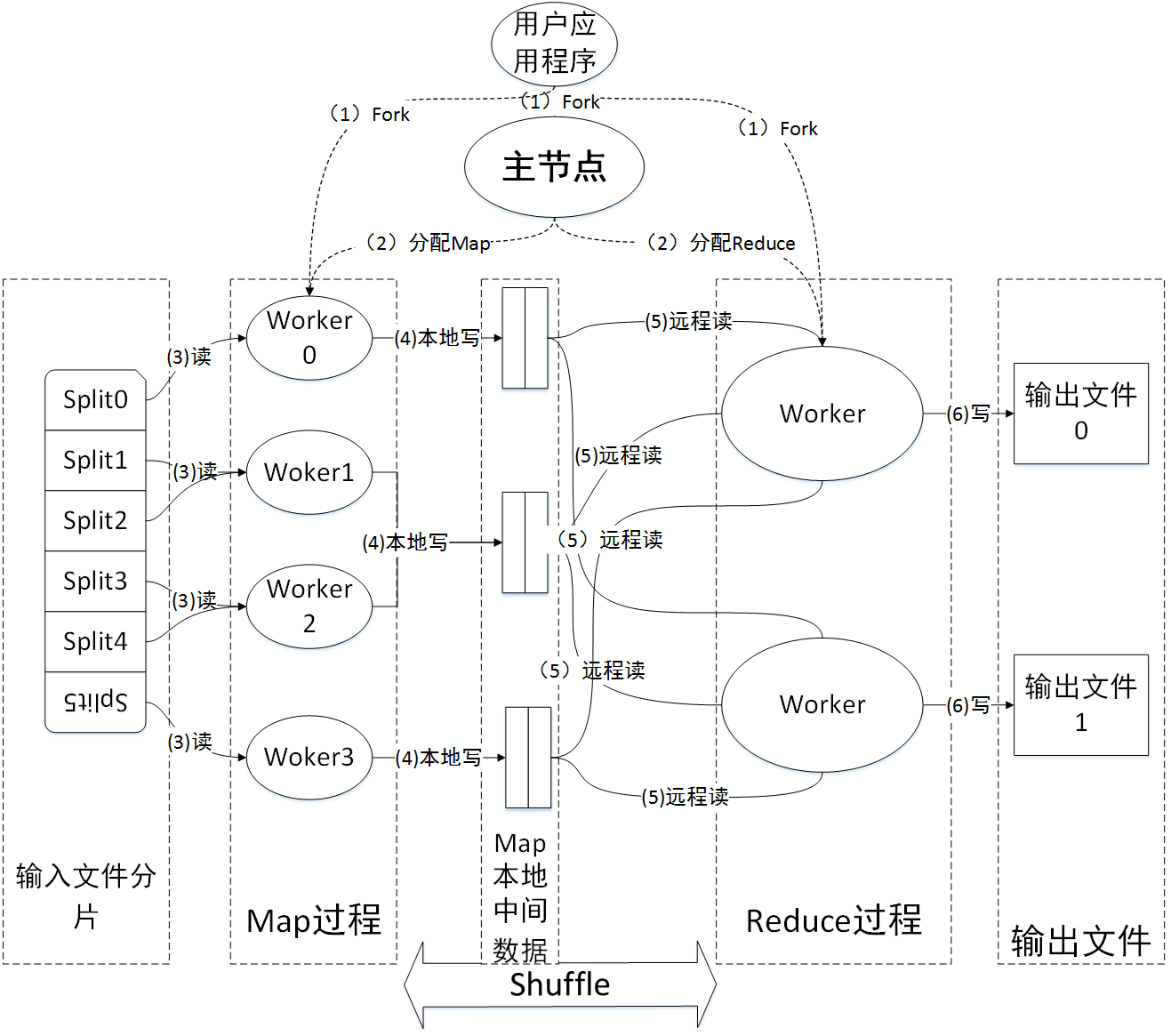


图2-3 MapReduce架构图

MapReduce计算框架设计为Master/Slave架构，图2-3显示了MapReduce的架构设计。MapReduce中涉及到的有这样几个元素：用户Client，Master，Worker。MapReduce计算模型的框架如下：

1. 用户提交作业任务给主节点。
2. 主节点检查作业配置，分配Map和Reduce任务。
3. Map任务从输入文件中读入指定的分片数据，一般每个Map处理一个分片数据。
4. Map任务执行用户指定的map()操作，并将数据写入到本地磁盘中。
5. Reduce通过Shuffle过程从完成任务的Map端主动获取属于自己要处理的Partition。
6. Reduce取到所有Map的对应Partition输出后执行用户指定的reduce()工作并将结果写到分布式文件系统中。

MapReduce中以Slot作为计算资源的分配单位，map任务和reduce任务都是运行在Slot上。Slot可以理解为单位计算资源，与cpu对应。Map和Reduce过程中输入数据逻辑上都是<Key, Value>格式。Map在处理完一条输入数据后会将输出数据先写入到本地内存环形缓冲区中，当缓冲区中的数据达到一定程度（默认为80%）时，Map会由Spill线程将内存中输出数据写到本地磁盘中。在Map运行过程中可能会出现多个输出文件，这些中间输出文件最终将Merge成一个输出文件。Merge过程总是按Partition聚集，按Value值排序。Partition是针对Key的聚集标准，通常情况下，这样可以保证Key值相等的记录会发送到同一个Reduce上处理，这是保证MapReduce计算模型正确性的基本要求。用户可以定制划分方法，但是必须保证达到这个要求。

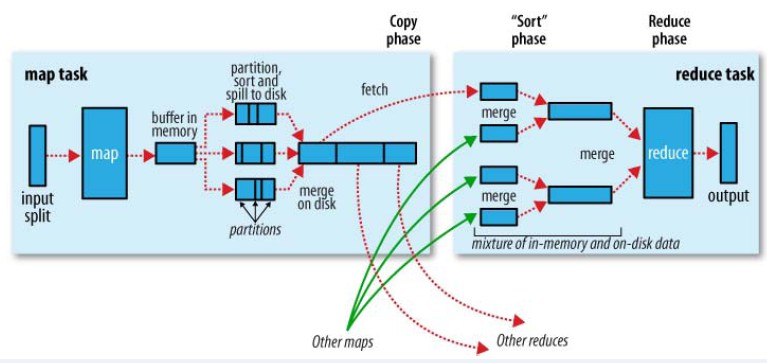


图2-4 Shuffle过程

每一个Reduce都有一个指定的需要处理的Partition数据。Reduce要能正确完成计算任务就需要从所有的Map输出文件中取得它所处理的Partition相关输出。Reduce通过Http协议从Map端拷贝文件到本地，拷贝过程中同时做Merge工作，Merge同样保证按Value有序。可见在MapReduce的Shuffle过程中一直保证了Value的有序性。

2.4 开源分布式计算平台Hadoop

Hadoop[43]是MapReduce的一个开源实现，最初是Apache开源搜索引擎Nutch的一个子项目。在大数据时代，Hadoop以其良好的扩展性、稳定性、容错性以及简洁的编程接口获得了巨大的成功。现在Hadoop已经成为大数据时代的标准。利用Hadoop可以方便有效地对大型数据集进行复杂的操作。

Hadoop系统主要有两个组成部分：分布式文件系统HDFS(Hadoop Distributed File System)和并行计算框架MapReduce。HDFS的设计目标是在通用的异构的廉价硬件上，具有高度容错性、可扩展性，针对一次存多次读，提供高吞吐量并发读的分布式文件系统。适合存储少量的大文件。HDFS放宽了（relax）POSIX的要求以此实现流的形式访问文件系统中的数据

HDFS设计为使用普通的廉价硬件，且大规模的数据必然会使用到大量的磁盘，因此系统中磁盘发生Failure的情况被认为是通常情况，因此HDFS在设计之初就针对经常的磁盘Failure做容错。HDFS有着高容错性的特点是通过对数据块的多备份机制实现的。HDFS中的数据都默认地被切分64M大小的分片，每个分片默认有3个备份，分别存储在一台机器上，与此机器同机架的一台机器上以及其他机架的一台机器上。当发现任意一台机器上发生数据丢失后，系统会自动从其他备份中去读取、复制，保证系统的三个备份。如此每个数据会有多个分片及其多个备份，这就使得在分配Map任务的时候可以保证较大的并发性。

Hadoop现在依然不仅仅是一个计算平台，随着Hadoop生态圈的不断成长，有越来越多的优秀项目加入到其中。如Hive、HBase、Pig等，它们或是对Hadoop的改进或者是对Hadoop的高层次包装抽象，使得其满足一些简单的应用接口定义。Pig提供了更加丰富的数据结构，一般为多值和嵌套的结构，还提供了更加强大的数据变换操作，包括MapReduce中忽略的连接(join)操作，其优点在于可以使用简单的控制台命令轻松处理TB级数据，对于研究人员和工程师而言操作数据更加方便简单；Hive则是构建于Hadoop之上的数据仓库框架，其设计目的是让SQL使用者能更加容易的操作HDFS上的海量数据，由于建立在Hadoop之上，很多组织将它作为一个通用且可伸缩的数据处理平台；HBase是对Google提出的BigTable的一种开源实现，是一个在HDFS上开发的面向列的分布式数据库，它不支持SQL，也不是关系型数据库，它自底向上地进行构建，在廉价的硬件集群上管理超大规模的稀疏表，却能够简单地通过增加节点达到线性扩展，是在HDFS上实时地随机读/写超大规模数据集的有效方案。

由于Hadoop的广泛认同性，且大量基于Hadoop之上的开源实用系统，对Hadoop本身的优化也可以使得这些系统获得进一步提升。因此本文选择Hadoop作为并行计算平台。

2.5 负载均衡

负载均衡发生在大规模的并行或分布式计算平台中，系统对外提供抽象的单机接口，对内使用大量计算节点同时进行计算任务，当内部多个计算节点间的计算量不等时就是负载不均衡，负载均衡的目的是充分利用现有的计算资源使得各个节点的计算量相等或相近，尽快对外提供计算结果。

负载均衡可以有多种划分方法，常见的有如下三种：静态/动态；集中/分布；软件/硬件。从策略上可以分为静态和动态负载均衡，静态负载均衡是在系统分配任务时不考虑即时状态，按照预先设定的均衡目标将任务进行划分，目标是节点间的负载相近；动态负载均衡是在分配任务时考虑现有节点的负载情况，目标是系统的整体执行时间最短；从交互方式上可以分为集中和分布式负载均衡，存在中心节点Master和从节点Slave的负载均衡为集中式负载均衡，各个节点间处于同等地位的是分布式负载均衡；从控制部件的实现方式上可以分为软硬件负载均衡，显然软件控制可以有较好的灵活性而硬件控制具有较高的成本和性能。

实现负载均衡有不少理论上的算法基础，各个算法都有各自的适用场景，评价一个算法的优劣要根据系统负载均衡的种类以及其目的来具体判断。负载均衡算法根据请求的分配结果是否一致可以分为一致性算法和非一致性算法。常见的有轮转法、散列法、最少连接法和最低缺失法。轮转法是一种随机的非一致性算法，算法中将各个节点视为同构的，每次请求依次从中选出一个作为结果，轮转法简单易实现，但不能较好处理系统中各节点不同构的情况常常需要搭配其他算法使用。最少连接法是一种集中式的非一致性算法，Master节点计算各个节点的连接量，每次选择连接最少的节点返回。最低缺失法和最少连接法相似不过每次分配给历史连接总和最少的节点。散列法是一种分布式的一致性算法，利用Hash函数的一致性，可以无需Master节点的情况下将请求一致的分配到节点中。

2.6 图数据表示方式

数据结构中图的表示方法主要有邻接矩阵、邻接表两种方法。下图展示了这两种表示方法，图2-5(a)是需要表示的原始数据图，图2-5(b)是用邻接矩阵方式表示图(a)，图2-5(c)是用邻接矩阵方式表示图(a)。

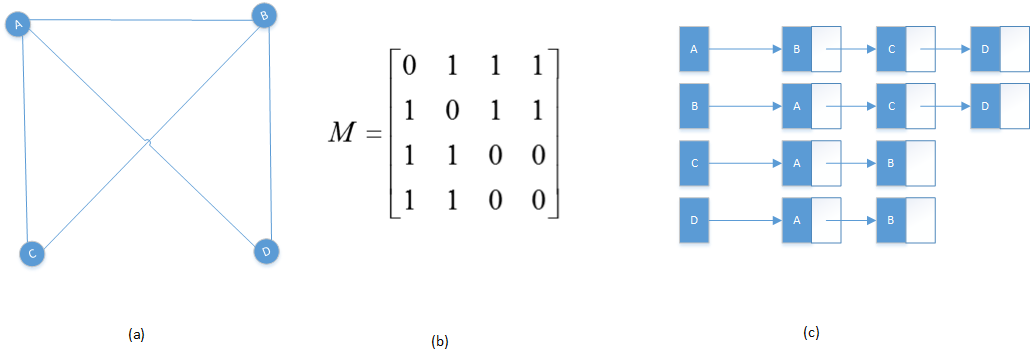


图2-5 图的表示方法

在本文的操作过程中需要有大量的取某个节点的邻接节点，以及邻接节点的交集运算，同时考虑到效率、内存使用以及实现难度，本文选用的哈希表的方式表示图的邻接关系：

GraphData：HashMap<vertex, HashSet<adjList>>;

2.7 本章总结

本章主要介绍了相关的背景技术，主要有三方面的内容，首先阐述了完全图、近似完全图的的相关定义以及数学概念。然后对本文并行算法使用到的编程模型MapReduce、计算平台Hadoop的基本原理做了简要介绍。最后介绍了负载均衡的基本概念，理论以及相关的算法。

第三章 完全图和近似完全图枚举算法

本章重点介绍极大完全图和极大近似完全图的对比算法和基于图分割的Binary单机算法，同时描述了算法在Hadoop平台上的简单并行化方案。

3.1 完全图枚举算法

本文选取现有完全图枚举算法中最具代表性的算法BK作为对比算法，提出了一种基于图分割的Binary算法。两者在基本定义上存在共性之处一并在下文中BK算法中介绍。

3.1.1 极大完全图BK算法

BK通过回溯搜索的方式来检查所有的点以枚举所有的极大完全图。BK搜索过程中的搜索路径组成了一个树状搜索结构。BK算法每次访问一个与搜索路径中已经访问的所有节点都相邻的点来扩展搜索，直到搜索路径不能再扩展，这样一条搜索路径上的所有访问点就形成一个极大完全图。

算法BKCliqueEnumerate显示了BK搜索树的访问过程。搜索树的每一个节点由下面三个点集组成，也就是搜索树的节点状态表示结构。

1. Result集，现有的搜索路径中已经访问过的节点集合
2. Candidate集，不在Result集中且与Result中的每一个点都相邻的点集
3. Not集，与Result中点都相邻且如果与Result集中的点组合将导致产生冗余或者被包含的结果的点集

算法BKCliqueEnumerate在访问一个节点时将其加入到现有结果集中并重新构建新的Candidate集合Not集。新的Candidate集通过筛选现有的候选集要与包括新加入到结果集中的点在内的所有Result集中的点都相邻。同样Not集也是通过筛选现有的Not集要求与包括新加入的点在内的所有Result点都相邻。BKCliqueEnumerate算法通过访问候选集中的点。对于每一个搜索树节点，首先访问的点是当前候选点中连接了最多候选点的点（也就是在候选点组成的子图中度数最大的点）。在搜索完第一个点之后只用那些与第一个点不相邻的候选点来扩展搜索路径。这样保证了在每一节点的搜索路径中只搜索最少的那些有可能生成新的且不冗余的点。一旦候选点Candidate中的任意一个点被访问后都会将其加入到Not集中，以表示包含这个点和当前结果集中的的极大完全图已经在新生成的子图状态中考虑了，搜索树中其他兄弟子图节点状态不应当在搜索相关节点。当BKCliqueEnumerate已经搜索完所有的子节点状态后将回溯到之前的一个未搜索子图状态。

表3-1 BK算法主过程

|  |
| --- |
| 算法1：BK算法 |
| 输入：数据图G，图G的节点集V以及边集E  输出：所有的且不重复、不冗余的极大完全图 |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. 调用方法BKCliqueEnumerate (Result, Candidate, Not) |

表3-2 递归极大完全图BK枚举方法

|  |
| --- |
| 方法BKCliqueEnumerate (Result, Candidate, Not) |
| 1. If Candidate =  then 2. If Not =  then 3. 输出结果Result 4. Else 5. fixp 🡨 Candidate 中与其他Candidate中相邻点最多的点 6. cur\_v 🡨fixp; 7. While cur\_v  NULL do 8. new\_not 🡨Not集中所有与cur\_v相邻的点 9. new\_cand 🡨 Candidate集中与cur\_v相邻的点 10. new\_res 🡨 Result + cur\_v 11. CliqeEnumerate(new\_res, new\_cand, new\_not) 12. Not 🡨 Not + cur\_v 13. Candidate 🡨 Candidate – cur\_v 14. If Candidate中存在点v与fixp不相邻 then 15. Cur\_v 🡨 v 16. Else 17. Cur\_v 🡨 NULL 18. return |



图3-1 输入数据图

考虑图3-1中的图G(V,E)作为输入图，初始状态中所有点都作为候选点包含在Candidate集合中。在所有候选节点中度数最大的点为标号为3的点，与3不相邻的点有{5,6,7}，因此根据BK算法将根节点子图扩展为包含3,5,6,7的四个子图。其搜索状态结构如图3-2所示。搜索状态中每个状态节点由三个结合组成，最上面的表示结果集Result，中间表示候选集Candidate，下面表示Not集。



图3-2 BK搜索状态树

搜索状态树中连接线上点表示用来扩展的节点。搜索树中叶子状态都是需要找的极大完全图，除了一些被排除的状态。最右侧用节点7扩展的状态中由于Not集中一个点6与当前所有候选点{4,8}都相邻，因此这个状态节点可以被排除，因为这个状态所能扩展出来的所有极大完全图都已经在之前出现过或者被之前出现过的极大完全图包含了。事实上，可以观察到点7扩展的子图状态所能找到的最终结果{7,4,8}被之前6扩展的结果{6,4,7,8}包含了。一个完整的抽象的BK搜索树扩展结构如图3-3所示。



图3-3 完整BK扩展树结构

尽管递归的BK算法描述非常简洁，但在实际应用环境中递归由于其大量的栈空间和内存、cpu消耗并不实用。BKCliqueEnumerate的搜索过程可以通过栈来模拟实现，如图3-4所示。下文Binary算法也具有相似的栈式实现，在实际实验过程中都是使用的基于栈的算法实现，后文不再赘述。

图3-4 基于栈的BK算法过程



3.1.2 极大完全图Binary算法

BK算法在执行过程中需要计算每个子图中各个候选节点的度数，但只是用来寻找度数最大的某个点，关于其他点的度数计算结果并没有充分使用。同时，观察到许多算法没有使用到完全图一个明显的特征没有使用：。另外不同于BK算法取最大度数点然后取所有与最大度数点不相邻的点来扩展搜索路径的粗粒度方式，本文提出一种高效地衡量当前状态动态地选择扩展点的搜索算法：Binary算法。

在BK的数据结构设计的基础上当前子图状态表示为，Binary算法的基本过程是在当前候选节点集中选择一个扩展点（分裂点）并将当前搜索图G划分为包含当前G的结果集且包含点的所有Clique，生成新的子图，子图状态通过如下方式生成，，；包含当前G的结果集且不包含点的所有Clique，生成新的子图子图状态通过如下方式生成，，。

下面证明这种划分方式的正确性。对于当前子图G的状态的子分支只有两种情况，包含点和不包含点，不存在第三种情况。对于包含点的状态，结果Clique中如果包含点，那么可以将点加到结果集中即：；状态的候选点由于需要满足包含节点的条件，因此需要保证候选点也必须都是的邻接点即；状态的Not集需要满足Not集的定义，Not集中的点可以与当前结果集节点组成完全图，因此Not集中点需要能够与相邻即：。对于不包含节点的状态，只将点从候选点中删除，同时由于Not的定义且包含点的子图必然都会在中处理，因此将点加入到Not集中，这样保证了所有不包含节点的完全图都是的子图。有以上可得证Binary划分方式的正确性。

表3-3 Binary算法主过程

|  |
| --- |
| 算法1：Binary算法 |
| 输入：数据图G，图G的节点集V以及边集E  输出：所有的且不重复、不冗余的极大完全图 |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. 调用方法BinaryCliqueEnumerate(Result, Candidate, Not) |

表3-4 递归极大完全图Binary枚举方法

|  |
| --- |
| 方法BinaryCliqueEnumerate(Result, Candidate, Not) |
| 1. If Not中存在一个点与Candidate中所有点都相邻 then 2. 此子图往下都是冗余结果，返回 3. If SubGraph(Candidate)是Clique then 4. 输出 ResultCandidate 5. Else 6. 在Candidate中选择一个分裂点v 7. 🡨 8. 🡨 9. 🡨 10. BinaryCliqueEnumerate() 11. 🡨 12. 🡨 13. 🡨 14. BinaryCliqueEnumerate() |

Binary的划分方式需要每次从当前子图状态中找到一个分裂点，如何选择分裂点对算法的效率有着至关重要的影响。考虑到图数据的各种特征且充分利用现有信息，本文使用候选点的度数作为选择切分点的衡量指标。显然，用度数作为选择指标有三种选择方式：选择度数最小的点、选择度数最大的点、随机选择一个点。根据Binary的划分方式，包含点的子图候选点的规模，子图候选点的规模。在给定子图G的情况下是不变的，由此可见选择最小度数点作为切分点则会生成一个较大的子图和一个较小的子图可以减少搜索空间。下图3-5是最大和最小度数切分的一个对比实例，后续的实验也验证了这一论断。

图3-5 最大度数切分和最小度数切分



使用Binary算法处理同样的输入图，其搜索树的结构如下图3-6所示。图（a）是算法描述的Binary搜索树结构左子节点表示G-右子节点表示G+。由于递归算法的栈消耗以及潜在的内存使用问题，本文采用栈记录子图状态的迭代方式实现Binary的递归算法。实际上在算法迭代过程中会将栈顶子图一直选最小度数点切分直到栈顶子图成为一个完全图或者栈顶子图已经不可能形成新的有意义的完全图。迭代算法切分过程中G-是一个可变化的子图状态，将左子树节点G-都映射到同一个子图状态，如图（b）所示。Binary算法的划分方式可以在很大程度上减少搜索子图的个数，这一点在后续的试验中得到了有效验证。

图3-6 Binary搜索树状态结构



Binary算法的实现中使用与BK相似的数据集合定义Result，Candidate，Not。不同的是，Binary算法需要始终跟踪Candidate节点中度数最小的点，而且在算法执行过程中由于需将与分裂点相邻的点度数减一，Candidate中节点的度数不断变化的。对此，最直接的方法是每次需要选度数最小的点时遍历当前所有候选点的度数，从中选出度数最小的点。假设子图候选点个数为n，Binary算法需要切分子图O(n)次，第i次切分需要筛选n-i个节点个数，因此在筛选最小度数点上需要的时间复杂度，在先期的实验中此部分成为算法效率的一个主要瓶颈。本文设计了一个最小度数结构用来更新和维护各个候选节点的度数，提供了O(1)的最小度数点选取操作，以及O(1)每个节点度数更新的操作。

如下图3-7（a）所示圆形表示度数，方形表示节点标号。最小度数结构将度数相同的节点放在一个哈希桶内并用它们的度数作为桶的标志，圆形度数标志之间形成一个有序的链表结构，同时每个节点标号有指向本身所在桶的引用。

图3-7 最小度数集合结构



在图3-7（a）表示的结构中进行图分割时，从度数链表选取头结点所指向的痛中选取第一个节点4（标号为4的节点度数为1，假设候选点中与4号节点相邻的是标号为5的点）作为分裂点，在不包含节点4的子图中需要更新度数集合，将与4相邻的节点5的度数减一。度数减一操作如图3-7（b）所示，将节点5从原来的桶中移除并将其加到其之前度数小一的桶中（不存在的话需新建一个桶，并将度数节点插入到原链表中）同时更新节点5所在的桶引用。其中所有操作都是O(1)复杂度，建立这个最小度数集合的额外复杂度是O(klogk)（k表示所有候选点的不同度数个数）。

3.1.3 极大完全图Hybrid算法

BK算法和Binary算法一个重要的区别在于关于图分割的定义，BK算法将图分割为包含最大度数节点的子图其余与该店不相邻的点依次罗列，相当于一次选点即可将当前搜索图全部切分完；Binary算法每次选用最小度数点将子图划分为多个小的子图，多次迭代分步将搜索图切分掉。在不同的情况下两者存在着各自的优势，当图中存在一个节点的与大部分节点都相邻时BK算法可以将图有效的切分为较少的几个子图；当图中节点度数偏小时Binary算法可以有效地将图切分为冗余较少的且易计算的多个小子图。鉴于实际应用中数据图的多样性以及算法的适用性，本文提出了一种Binary和BK综合的算法Hybrid。

表3-5 Hybrid算法枚举过程

|  |
| --- |
| 方法HybridCliqueEnumerate(Result, Candidate, Not) |
| 1. If Not中存在一个点与Candidate中所有点都相邻 then 2. 此子图往下都是冗余结果，返回 3. If SubGraph(Candidate)是Clique then 4. 输出 ResultCandidate 5. Else 6. 从最小度数结构中选取度数最小的点 7. If Deg()> then 8. 从最小度数结构中取度数最大节点 9. cur\_v 🡨; 10. While cur\_v  NULL do 11. new\_not 🡨Not集中所有与cur\_v相邻的点 12. new\_cand 🡨 Candidate集中与cur\_v相邻的点 13. new\_res 🡨 14. CliqeEnumerate(new\_res, new\_cand, new\_not) 15. Not 🡨 16. Candidate 🡨 17. If Candidate中存在点v与fixp不相邻 then 18. Cur\_v 🡨 v 19. Else 20. Cur\_v 🡨 NULL 21. Return 22. BinaryCliqueEnumerate() 23. Else 24. 🡨 25. 🡨 26. 🡨 27. 🡨 28. 🡨 29. 🡨 30. BinaryCliqueEnumerate() |

由前两节的内容可以看出BK算法和Binary算法在搜索树节点状态的整体结构上相似，因此两者可以较方便地实现融合。Hybrid算法以Binary算法为基础，在选择分裂点时考虑最小度数点邻接的候选节点比例，当被切割的图中最小度数点都邻接了大部分节点时，使用BK的切分方式从候选点中选择度数最大的点作为分裂点来切分搜索图，否则还是使用最下度数点分割搜索图。由上一节中最小度数集合的设计可以看出，选最大度数点也只需从链表中取尾节点的桶中取任意一个点即可。Hybrid算法的实现要求，BK和Binary都可以较好地支持，不需做数据结构上的大改动。Hybrid算法中使用到了一个变量，表示最小度数点与候选点中邻接个数比例，实验部分中验证该值一般取0.6能够获得较好的效果。

3.2 近似完全图枚举算法

K-Plex算法是BK算法的一种变形，其本质上也是图的分割。为简化下文算法的描述有如下定义：

定义1：顶点与顶点集合相邻，要求满足用表示；顶点与不相邻，要求满足用表示。

K-Plex算法Pemp与完全图算法相似，也有关于搜索节点相似的定义Result，Candidate，Not。在K-Plex中用来扩展现有搜索路径的Candidate候选节点v需要满足以下两个条件

条件1：v与Result中至少个点相邻

条件2：与中至少个点相邻

显然任意一个满足条件1和条件2的点都可以与现有的结果集组成一个新的更大的K-Plex。根据Candidate和Not集合的定义，Candidate和Not集中的点都需要满足条件1和条件2。

定义2：搜索路径上一个子图节点的结果集Result中的点与中k-1点不相邻，则称是临界点；由临界点组成的集合即临界点集合Critical。

为了满足条件2，当搜索路径子图中临界点集合Critical不为空时，要求可用来扩展的Candidate中的任意一点必须与临界点集合中的所有点都相邻。否则，那么当不满足此条件的点用来扩展现有路径时中会至少存在一个临界点与K个点不相邻，也就是违背了K-Plex的必要条件。

3.2.1 基于BK的K-Plex算法Pump

在Pemp算法中为Result、Candidate和Not集中点都关联了一个计数：counter1。counter1用来记录其关联的顶点与Result中的多少个点不相邻，其中Result的counter1表示Result中的点与中多少个点不相邻。为了满足条件1，只需要保证点的。

表3-6 KPLEX单机算法主过程

|  |
| --- |
| KPLEX单机算法Pump |
| 输入：数据图G，图G的节点集V以及边集E  输出：所有的且不重复、不冗余的KPlex |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. While Candidate 5. 依次取 6. Result 🡨 Result + cur\_v 7. 调用方法FindAllMaximalKplex(Result, Candidate, Not) 8. Not 🡨 Not + cur\_v 9. Candidate 🡨 Candidate – cur\_v 10. Result 🡨 Result – cur\_v |

表3-7 FindAllMaximalKplex方法

|  |
| --- |
| 方法FindAllMaximalKplex(Result, Candidate, Not) |
| 1. Connected\_Candidate 🡨 Candidate中与Result中任意一个点相邻的点 2. Connected\_Not 🡨 Not中与Result中任意一个点相邻的点 3. While Connected\_Candidate 4. If Connected\_Candidate 5. If Connected\_Not 6. 输出Result 7. Else 8. Result是冗余结果 9. cur\_v🡨SelectExpandNode(Result, Critical\_Res, Connected\_Candidate, Connected\_Not) 10. If cur\_v = NULL 11. Return 12. Candidate 🡨 Candidate – cur\_v 13. Cur\_cand 🡨 Candidate; Cur\_not 🡨 Not; Cur\_res 🡨 Result 14. Cur\_res 🡨 Cur\_res + cur\_v 15. 更新Cur\_cand和Cur\_not中各个点的计数，同时移除计数counter1大于k-1的点 16. Critical\_Res 🡨 Result中与其他节点不相邻个数等于k-1的点 17. 移除Cur\_cand和Cur\_not中与Critical\_Res不相邻的点 18. FindAllMaximalKplex(Result, Candidate, Not) 19. Not 🡨 Not + cur\_v 20. Connected\_Not 🡨 Connected\_Not + cur\_v |

算法去冗余的方法是通过最后Connected\_Candidate为空且Connected\_Not不为空表示Connected\_Not中的任意一个点都可以和现有的结果集组成一个K-plex，但是包含点的所有K-plex已经在搜索路径的其他分支上已经生成了，故而无需输出。如果可以在搜索路径的过程中提前判断Connected\_Not中存在一个点不可能被过滤掉，那么就可以保证这个节点下的整棵子树都是冗余的，可以提前剪枝。如果Connected\_Not中存在一个点与Result和Candidate中的所有点都相邻，那么在此之后无论如何扩展这个搜索路径，都无法通过条件1和条件2来将这个点过滤掉。也就是说在此之后所有的结果中Connected\_Not都不为空，此分支可以剪去。

剪枝条件：

在算法中为Not关联一个新的计数器counter2，counter2表示Not中的点与Result和Candidate中有多少个点不相邻。当Not中存在点的counter2 = 0时也就是剪枝条件满足了。

为了加快剪枝速度，算法尽可能的使得counter2降到0。为此提出一个新的集合Prunable\_Not，Prunable\_Not是Not的一个子集，包含Not中与Result集中所有点都相邻的点，也就是counter1=0的那些点。注意到，每当选择一个新的点v加到Result中来扩展搜索时，Not中与v不相邻的点的counter2计数就会每次降1。如果一直从Candidate中选与Prunable\_Not集中某个点v不相邻的那些点来扩展，那么点v的counter2很快降到0且将不可能存在Not集中的其他点会比v更快满足剪枝条件。显然，为了能够最快地达到剪枝条件应该选择Connected\_Candidate中与Prunable\_Not中counter2最小的点v相邻的那些点。但是在算法刚开始时还不存在临界点集合时Connected\_Candidate中不存在与Prunable\_Not不相邻的点，因此还是需要按顺序取扩展点。详细过程在方法SelectExpandNode中描述。

表3-8 Pump寻找扩展点方法

|  |
| --- |
| 方法SelectExpandNode (Result, Critical\_Res,Connected\_Candidate, Connected\_Not) |
| 1. If Result && Critical\_Res 2. Return Connected\_Candidate中的第一个点 3. Else 4. Prunable\_Not 🡨 Connected\_Not中与Result中所有点都相邻的点 5. If Prunable\_Not 6. If 7. Return NULL 8. Else 9. Return, 10. Else 11. Return Connected\_Candidate中的第一个点 |

3.2.2 基于Binary的K-Plex算法

K-Plex的Binary算法使用与完全图相似的切分方式。K-Plex的Binary算法中对一个子图状态的节点关联两个度数cDeg和rDeg。cDeg表示关联的节点与Candidate中有多少个点不相邻，rDeg表示关联的节点与Result中有多少个点不相邻。参考文献指出有意义的K-Plex的所有节点必然包含在两跳以内，因此本文在使用候选节点时挑选两跳数据节点。

表3-9 K-Plex算法主过程

|  |
| --- |
| KPLEX单机算法Binary |
| 输入：数据图G，图G的节点集V以及边集E  输出：所有的且不重复、不冗余的KPlex |
| 1. Result 🡨 2. Candidate 🡨 V 3. Not 🡨 4. While Candidate 5. 依次取 6. Result 🡨 Result + cur\_v 7. TwoHop 🡨 8. 调用方法BinaryMaximalKplex(Result, ,) 9. Not 🡨 Not + cur\_v 10. Candidate 🡨 Candidate – cur\_v 11. Result 🡨 Result – cur\_v |

Binary的K-Plex算法中依然使用到了3.1.2节中使用到的最小度数集合结构。不同的是，在K-Plex算法中最小度数集合结构中的度数指的是rDeg与cDeg的和。每次取分割点，从最小度数集合结构中去度数最大的点(rDeg+cDeg度数最大的点，亦即与结果集和候选集相邻最少的点，符合Binary算法中关于最小度数的点分割的预期)。判断一个Candidate是否能够形成K-Plex可以通过判断其中点的cDeg+rDeg<k-1。对于以上两个需求最小度数集合都可以高效地适用，寻找分割点只需要从最小度数集合结构中选取最后一个度数关联的点集中任意一个点即可；判断是否是K-Plex可以通过最大的cDeg+rDeg（即最小度数结构中最大值）是否满足条件即可。Binary的K-Plex算法中使用的剪枝条件也是Not集中存在一个点与Candidate和Result中所有点都相邻。

表3-10 Bianry的K-Plex枚举算法

|  |
| --- |
| 方法BinaryMaximalKplex(Result,Candidate,Not) |
| 1. 计算最小度数结构、临界点Critical、 2. Prunable\_Not 🡨 Not中rDeg=0的点 3. While Candidate 还有切分的意义 4. If Prunable\_Not 中存在cDeg=0的点 5. 此分支可以减掉；Return 6. If 7. 输出为K-Plex；Return； 8. 从最小度数结构中选出rDeg+cDeg最大的点cur\_v   12. 更新Result中各点的rDeg和cDeg 13. 🡨 14. BinaryCliqueEnumerate () 15. 🡨 16. 更新Cur\_cand和Cur\_not中各个点的计数，同时移除计数cDeg大于k-1的点 17. Critical\_Res 🡨 Result中与其他节点不相邻个数等于k-1的点 18. 移除Cur\_cand和Cur\_not中与Critical\_Res不相邻的点 19. Not 🡨 Not + cur\_v 20. If 21. Prunable\_Not 🡨 Prunable \_Not + cur\_v |

在K-Plex问题中注意到K-Plex问题比完全图问题更加复杂，搜索空间是K-Plex最主要的性能瓶颈。事实上Pemp算法的剪枝策略并没有很好的提前剪枝，其搜索空间依然十分庞大，本文Binary的K-Plex算法继承了Binary算法中减少搜索空间以提高搜索效率的优势，在搜索空间上远小于Pemp算法，并在实验中得到了验证。

3.3 算法并行化

本文旨在研究图算法以及其高效并行化。在并行化方面，本文采用分布式计算平台Hadoop使用MapReduce编程模型实现。并行算法的可用性和高效性主要通过算法的可分割性以及可均衡性来实现。算法可分割便可以并行化，算法可均衡负载则并行算法可以解决长尾问题，使得算法能够有很好的扩展性从而能够通过机器数目的简单叠加获得计算时间的降低，提高吞吐量。本文提出的三个算法都可以较好地满足可分割性和可均衡性。

对于可分割性：综合以上完全图和近似完全图的几个算法可以发现它们本质上都是对于图的一个树状搜索，在搜索树路径上任意一个子树的搜索过程都与其不相交的另一个子树不存在任何依赖关系，因而可以较方便地将不同的子树分配到多个机器节点上并行地进行运算。另外，由于不相交子树间没有信息依赖，因而可以适用于Hadoop这类Share-Nothing的计算平台。

对于可均衡性：在算法真正遍历完搜索之前，并不能够从子图数据中准确计算单个子图的搜索代价，同时由于实际数据存在大量的计算倾斜，以上算法在实际的分布环境中都存在负载不均衡的问题。得益于分割后的子图的独立性，在Hadoop平台上，算法可以暂停当前任务，将剩余子图及新分割出来的子图通过Hadoop的Shuffle过程随机分发到各个计算单元上进行新一轮的计算，通过多轮作业迭代算法可以较好地均衡各节点间的计算负载。算法的负载均衡相关内容将在下一章中做详细介绍。

此外，要提升并行算法的加速比，还需要尽可能降低算法并行化所带来的额外消耗。Hadoop平台中各个Slave节点之间是异构、等价的，Slave节点之间不直接交互作业信息。同时Hadoop系统中所有中间结果或者数据都通过磁盘进行持久化，并行算法需要考虑尽可能地降低磁盘读写数据，减少数据传输量。

极大完全图枚举和极大K-Plex枚举在Hadoop并行化算法上具有相似的结构，包括读入输入图数据、初始任务分配、负载均衡和终止条件，负载均衡策略将在下一章中详细介绍。

3.3.1 输入图数据

在并行Hadoop程序中，输入图数据是在Map中完成的，Map函数将输入数据按需求发送到各个Reduce端，在Reduce端进行完全图和近似完全图搜索工作。

对于极大完全图检测：传统检测算法中一般使用两跳数据集作为输入数据集，一个点的两跳数据可以保证找到包含这个点的所有极大完全图的正确性和完整性。引入两跳数据集的初衷是用来减少不相关点的干扰，提前筛去无意义点，但实际数据中两跳数据集的筛选能力较差，依然导致大量无效的中间输出。注意到所有有实际意义的极大完全图（组成极大完全图的点的个数不小于4的完全图）都是由多个三角形构成的。本文采用文献[3]中的方法挖掘出基本图中所有的三角形，极大完全图挖掘的输入数据是一行行的三角形数据，每行三个点<A,B,C>。对于每个三角形<A,B,C>，map函数按Key值将其分别映射到三个点上输出，<A,<B,C>>、<B,<A,C>>、<C,<A,B>>。这样在每个Reduce端都可以接收到包含一个点的所有三角形(比如点A，将接收到<A,<<B,C>,<B,D>,<C,D>>等)。三角形数据同时包含了检测一个点的完全图所需的边连接信息，Reduce端将除当前点外的一条边，构建成邻接表结构（即将边<B,C>,<B,D>,<C,D>组成邻接结构）。

表3-11 并行算法Map过程

|  |
| --- |
| 输入数据Map |
| 输入：三角形数据<A,B,C>，每行一个 |
| 1. For Each <A,B,C> 2. Context.Write(<A,<B,C>>) 3. Context.Write(<B,<A,C>>) 4. Context.Write(<C,<A,B>>) |

对于极大近似完全图检测：需要指出的是在近似完全图的定义中并没有对邻接性作要求，比如两个独立的三角形整个的来看也可以形成一个K-Plex(k>3)，事实上这样的近似完全图并没有应用价值。K-Plex应当时作为现实环境中完全图要求过于严格的一种弱化条件，因此本文所找出的K-Plex都是一个连通图，这样的结果具有实际意义。根据在1.3.4中介绍的文献[35][36]对于近似完全图的理论研究，一个点的两跳数据集可以满足检测出包含这个点的所有有意义的K-Plex。在其Map中只需要根据输入数据的所有节点将其分发到各个Reduce上，在Reduce端各自从本地文件中读入并构建两跳数据集。

3.3.2 初始任务分配

在单机算法中尽管搜索树中处于同一层的节点间相互独立，满足可分布的特性，但初始搜索状态中所有点都是候选点，直接将这样的结构发散到各个计算节点上必然会带来大量的重复计算。另外，单机算法的搜索树中根节点到第一层子节点的扩展是集中计算的，不适合MapReduce编程模型Share-Nothing的特征，也是并行算法切分任务的基本部分。BK算法搜索树节点由根节点扩展到第一层子节点时，通过最大度数点和与其不相邻的点来分割，这有可能导致分割出来的第一层子图个数过少从而导致单个计算节点的负载过高。Binary算法搜索树由根节点扩展到第一层子节点时，可以较好的均匀各个子图负载，但一次分割耗时过多且需集中分割。

考虑到并行系统的分布特性及MapReduce的模型要求，本文初始任务分配选择每一个顶点作为初始任务的分割点。也就是说对于每一个顶点都在第一层生成该节点的搜索子图。Hadoop中每一个计算单元Slot都有一个唯一的ID（MapTask\_ID或者ReduceTask\_ID），Map任务可以通过节点ID和Slot\_ID判断这个节点的计算任务需要发送到个Slot上进行计算，这样任务就可以无需集中切分以实现第一搜索树节点的发散。

按照节点ID和计算单元Slot\_ID发散任务过程中需要考虑搜索树状态节点的三个数据集的完整和正确性。包含节点v的所有Clique中所有点都应该在v的邻接表中，三角形数据可以完整地复原一个顶点的相邻节点的邻接表状态。节点v的第一层子树状态中只有一个结果集Result的元素即v本身。Not集的含义是之前已经搜索过的节点，但是在分布的算法中无法获知哪些节点已经搜过了。因此分布算法中各个节点的搜索次序需要保持逻辑上的全序，即逻辑上认为先搜索点的子树再搜索的子树…最后搜索的子树。这样按照Candidate和Not的定义，对于一个点的邻接点集合，比该点ID值大的点认为没有搜索过加入到Candidate集中；比该点ID值小的点认为已经搜索过了，加到Not集中。至此，各个计算单元可以独立地并行搜索各个子图。

表3-12 并行算法Reduce过程

|  |
| --- |
| 初始任务分配Reduce |
| 输入数据：三角形数据<A1,<<B1,C1>,<D1,E1>…<F1,G1>>>, <A2,<<B2,C2>,<D2,E2>…<F2,G2>>>,…<An,<<Bn,Cn>,<Dn,En>…<Fn,Gn>>> |
| 1. For Each <Ai,<<Bi,Ci>,<Di,Ei>…<Fi,Gi>>> 2. 构建三角形除当前顶点Ai之外第三条边<Bi,Ci>,<Di,Ei>…<Fi,Gi>构成的邻接表Graphi(Vi,Ei) 3. Result 🡨 4. For 5. If > Ai 6. Candidate🡨 7. Else 8. Not🡨 9. EnumeAllMaximalClique(Candidate, Result, Not) |

类似于3.1.2节中关于Binary切分点选择的问题，对于第一层子树各个点的全序可以多种设计方式，本文选取最简便直接的点ID大小。对应于BK和Binary的分裂点设计思想可以有各自的按点度数的全序。按BK的设计思路按照先搜索度数大的点在搜索度数小的点，这样可以使得度数大的点更可能出现在度数小的点的Not集中。同时，度数大的点期望邻接的候选点也会更多，这样就可以使得小度数点的子图更容易达到剪枝条件，从而加快搜索速度。反之，按照Binary的设计思想应先搜索度数小的点从而减少各个Size-1子图的搜索树大小。这两种方式都要求预先处理输入数据，计算出各个点的度数大小，并且将结果在全局发布。这限制了算法的分布性和适用性，同时带来了额外数据处理代价。综合考量，本文最终选择简便的ID作为全序设计标准。

初始任务分配完成以后，各个Reduce的Slot各自运行单机算法执行搜索过程，直到所有Slot都完成各自的搜索任务后整个作业完成。K-Plex的分布算法与Clique的分布算法思路基本一致，不再赘述。

3.3.3 负载倾斜

在上一节中分布算法只对初始任务进行分散划分，每一个Reduce Slot处理一棵Size-1的搜索子树。但是由于在处理这棵子树之前无法通过底层子节点的三个状态集合以及图的邻接表计算出这棵子树的搜索代价，因此无法通过预先制定分散策略使得各个Slot的负载相对均衡。在实际数据的处理过程中观察到，大多数现实数据存在着严重的负载倾斜问题。社交网络数据是一个明显的实例，一些名人的节点虽然数目少，但是由于名人效应他们的关注者非常多。这些点都连接着数十万乃至上百万的点，其搜索代价也远大于其他普通节点。

图3-8 各机器节点处理时间对比

上图是Pokec在线社交网络数据各个Size-1子树的搜索代价分布，可以看出大部分Reduce Slot可以在很少的时间内搜索完成，但是少量的却需要很长时间才能计算结束。显然，将负载直接分散到各个计算单元而不做负载均衡处理未能完全发挥集群的计算能力，出现了长尾现象。同时由于单棵搜索树的瓶颈效应，即使向集群中添加更多的机器，不能够带来处理效率的相对提升，阻碍了算法的可扩展性。因此进一步的负载均衡势在必行，下一章将会详细讨论算法的负载均衡问题。

3.4 本章总结

本章主要详细介绍了极大完全图和极大近似完全图枚举的对比算法BK及其变种和本文提出的两个问题的Binary算法。在综合两者各自的优点之上提出了极大完全图的Hybrid算法。算法都保证枚举出数据图中所有的有意义的且不重复的极大完全图或极大近似完全图，应用剪枝方法避免了冗余搜索路径的消耗和重复结果的出现。介绍了在Hadoop平台下算法的并行化方案，在保证搜索结果完整且不冗余的基础上，简洁且无中心的分散计算任务。最后提出了一次分散并行中负载倾斜的问题。

1. 并行环境下的负载均衡

在3.3节中讨论了算法并行化的初步实现，由于子树的一次划分以及各个子树的搜索代价不可预估带来了负载倾斜问题。为了提升算法的扩展性以及实用性本文提出了基于Hadoop平台的完全图和近似完全图负载均衡方案。考虑到针对特定算法的负载均衡方案的局限性，本文进一步提出了一种系统层面的负载均衡方案，并描述了在分布式计算平台Hadoop上的具体实现。最后本章探讨了系统负载均衡存在的问题及可行的解决方案。

4.1 完全图和近似完全图枚举的负载均衡

第三章中根据Size-1子树之间的独立性，通过对Size-1子树的发散获得了一定的并行性。在第三章发散Size-1子树方案的基础上，注意到整个搜索树中，每个节点的子节点之间都可以相互独立计算。因而可以在子图切分过程中将部分子图保存暂时不予搜索，在下一步中再发散到其他计算单元上进行计算。在Hadoop中，各个Slave之间不能够直接交互，因而需要将临时保存的子图在下一个作业中Shuffle到其他节点上。通过Hadoop的Shuffle过程发散子图需要涉及到存储子图到磁盘和启动新作业发散数据两个过程。这两个过程需要磁盘I/O以及网络I/O，它们相对于CPU的运算来说都是非常耗时的。这就出现了两个矛盾的因素：

a．尽快停止当前作业的Slot搜索过程，发散子图，避免任何Slot空闲

b．充分发挥CPU的计算能力，尽力搜索一个子图，避免启动新任务的开销

因素a.的最理想状态是在任意一个Reduce Slot处理任务完成时立即停止当前作业，发散任务以避免系统中出现空闲的计算单元。在MapReduce的同步计算模型限制下，各个Reduce Slot负载处理已分配的任务，截止条件不依赖于其他Slot，这种方式只是一个理想情况。因素b.希望尽可能减少启动新任务迭代的开销，其极端情况就是第三章中提出的只有一次分散任务的并行方法。综合两个因素考量，其关键问题在于何时停止当前子图搜索以及何时开始新的一轮任务发散。

为解决以上两个问题本文引入两个参数N和T。参数N指切分出一个子图G后如果G中的候选点个数大于N则不再深度遍历这个子图而直接将其存到本地磁盘，以待下一次Shuffle到其他Slot计算；反之，直接将子图G的所有路径搜索完成找到所有可能的结果。参数N的设计基于一个假设：候选节点越多的子图其计算量可能越大。参数N的设置可以避免花费大量时间来计算同一棵子下的节点，而导致其他节点得不到处理，以至于发散任务后计算量还是无法分散到各个节点上。参数T指单个Reduce Slot运行时间累计到达T时不再进行子图搜索枚举结果的工作，转而做子图切分工作，以便可以在下一个作业中发散未计算子图到其他节点。参数T的设置可以避免单个节点负载重的情况下一直计算而无法进行同步发散数据的问题。

有负载均衡的MapReduce实现分为两类作业，第一类作业是从原始输入数据进行搜索工作，第二类是从第一类作业以及第二类作业本身未处理完的切分出来的子图作为输入数据进行完全图和近似完全图挖掘工作。第一类作业的输入Map处理与第三章中的方式相同，Reduce的算法逻辑如下。

表4-1 并行负载均衡Reduce过程

|  |
| --- |
| 第一类作业Reduce |
| 输入：三角形数据<A1,<<B1,C1>,<D1,E1>…<F1,G1>>>, <A2,<<B2,C2>,<D2,E2>…<F2,G2>>>,…<An,<<Bn,Cn>,<Dn,En>…<Fn,Gn>>> |
| Reduce.SetUp  从配置文件读入N，T；获得本地临时输出文件  Reduce.Reduce   1. Time🡨0 2. For Each <Ai,<<Bi,Ci>,<Di,Ei>…<Fi,Gi>>> 3. 构建三角形除当前顶点Ai之外第三条边<Bi,Ci>,<Di,Ei>…<Fi,Gi>构成的邻接表Graphi(Vi,Ei) 4. Result 🡨 5. For 6. If > Ai 7. Candidate🡨 8. Else 9. Not🡨 10. If Time<T 11. Time += ComputeOneSubgraph(Candidate,Result,Not,Time) 12. Else 13. SpillGraphToDisk(Candidate,Result,Not) |

第一类作业的Reduce首先在初始化时读入配置文件中的T,N等相关参数设置，获取改Reduce一个本地临时文件用作超时或者超大子图的临时写入区。然后读入Map发送过来的三角形数据，按第三章中的方法构建Size-1子图状态，最后调用ComputeOneSubgraph来处理这个子图。当该Reduce的计算时间超过设定时间T时，构建出来的子图不再进行搜索计算，而是直接写到本地输出文件中。

表4-2 单个子图处理流程

|  |
| --- |
| 方法ComputeOneSubGraph |
| 输入：Candidate,Result,Not,Time  输出：找到的结果，输出子图文件，耗时 |
| 1. tmpTime🡨Time 2. 将输入子图<Candidate,Result,Not>入栈Stack 3. While Stack 4. Start🡨SystemTime 5. 取栈顶子图curCandidate,curResult,curNot 6. 按单机算法的方式切分子图 7. If tmpTime<T and <N 8. 调用单机算法将计算完 9. Else 10. 将输出到新的临时文件 11. End🡨SystemTime 12. tmpTime+=End-Start |

在ComputeOneSubGraph中使用单机算法的切分方式切分输入的子图。对于切分出来的子图考虑是否超时和是否超大两个因素，只有当前Reduce的计算时间未超时且切分出的子图未超大小时才调用单机算法直接将这个子图搜索完成，其他情况下都先将写入到本地临时输出文件中，待下一步处理。

还有另外一种情况，当该Reduce的计算负载比较小，可能未达到时间阀值T，Reduce的搜索工作就已经完成。如果停止工作等待其他Reduce完成再进行下一步的处理就浪费了本地的处理能力。未充分利用计算资源，本文负载均衡过程中利用Reduce的清理工作函数CleanUp利用剩余的工作时间，将Reduce中输出的超大文件读入到内存中处理。这样在一个Reduce的执行过程中除非本地的所有任务都已经计算完成，Reduce在规定的时间阀值T内都会一直处于计算状态，从而提高了系统的计算资源利用率。

表4-3 并行负载均衡Reduce清理过程

|  |
| --- |
| Reduce.CleanUp |
| 输入：本地临时输出文件  输出：搜索结果，新的本地临时输出文件 |
| 1. 获取新的输出文件 2. 从本地输出文件中读入一个子图G(Candidate,Result,Not) 3. If G=Null 4. 本地所有任务都已计算完成，退出 5. Else If Time>T 6. 将原输出文件中未计算的部分直接拷贝到新输出文件后，退出 7. Else   Time += ComputeOneSubgraph(Candidate,Result,Not,Time) |

和两个因素a,b之间的矛盾一样，时间参数T和子图大小参数N之间也存在着权衡和制约。当T过小时，一次作业能够处理的任务减少，增加了任务数，带来大量读写文件及启动作业的代价；当T过大时每个Slot都可以充分计算本地任务，但是有些任务少的机器的计算资源被闲置了。当N过小时，很少有子图能够在本地计算掉，需要等到下一步的Shuffle发散，带来了网络传输量的增大；当N过大时，由于深度搜索会出现一直在计算前几个子图而其他子图得不到切分，计算量不能够有效发散的问题。

可以预期的是T和N的取值存在一个中间的较优情况，这一点也在后续的实验中得到验证。本文的分布实验过程中都是选择的T、N的较优情况作为参数配置。与一次任务发散的分布算法相比，负载均衡的分布算法表现出更优的性能和扩展性。

本章的负载均衡算法可以应用于第三章中所描述的BK完全图枚举、Binary完全图枚举、Hybrid完全图枚举、Pump K-Plex枚举以及Binary K-Plex枚举。这里只给出了完全图的描述，其余K-Plex等分布算法的负载均衡完全类似，不再赘述。

4.2 并行平台Hadoop中通用的负载均衡

在上一节中本文根据极大完全图枚举和极大近似完全图枚举这两个特定算法设计了负载均衡策略。笔者发现负载均衡问题是一个在所有并行和分布式计算平台中存在的普遍的且重要的问题。诚然，可以要求算法实现者根据特定应用的特征，通过先验知识以及算法流程设计去均衡负载。一个更加用户友好的方式是使得系统平台本身就是可自动均衡负载的。这样用户可以通过简洁的一致的编程接口，无需考虑如何去均衡负载同时又能够在并行系统中获得较好的均衡性和扩展性。本文进一步深入研究并实现了并行计算平台Hadoop系统层面的负载均衡方案。

4.2.1 负载均衡问题讨论

负载均衡问题是一个广泛而普遍存在的问题。在所有的分布式系统中几乎都会提及到“长尾问题(Long Tail Problem)”，其实也就是大家常说的“短板理论”，系统的整体表现取决于表现最差的一部分。常见的分布式系统如分布式缓存，分布式存储，分布式计算，分布式数据库等等，都存在这个问题。分布式缓存中可能会遇到短时间内集中访问同一个缓存的情况；分布式存储可能单机磁盘使用过度；分布式计算可能会有单点的计算负担过重；分布式数据库可能会有单机访问量过大。有些问题早有了较成熟的解决方案，像分布式缓存系统中常见的一致性哈希算法等。

笔者总结对负载均衡问题的定义：在多点协作的系统中由于不合理的任务分配导致某个或者少量的某些节点处理负担过重，最终拖延整个系统对外的响应效率。

负载均衡的主要解决方案有两种。一种是被动解决，当系统中发现倾斜后将负载迁移到空闲节点。另一种是主动预防，包括用户先验知识的介入预防以及从系统层面上分配任务的策略上预防倾斜发生。本文所提出的负载均衡方案属于系统任务分配策略主动预防方式。

对于MapReduce模型的Hadoop系统负载均衡问题的解决在大多数情况下是存在一个极限的，这取决于具体作业的可划分性。通常地，Map函数的工作是从HDFS中读取上输入文件，读入的数据是一个个键值对<MapInputKey/MapInputValue>，根据作业需求处理后输出一个个<MapOutputKey/MapOutputValue>对。后台的输出线程会把输出的文件按照MapOutputKey把对应的MapOutputValue合并起来<MapOutputKey🡪MapOutputValue0,MapOutputValue1,…>，同时还会将输出按照MapOutputKey排序（每一个Map都会有同样的样的输出，不同的Map会有同样的Key值输出）。逻辑上，我们可以将不同Map输出的同一个Key的数据合起来看做一个小Partition（Finer-Partition）。Reduce将Map输出的各个Finer-Partition拷贝到本地(拷贝过程中一直是保证键值对的有序性)，然后对于每个键值对序列<ReduceInputKey🡪 ReduceInputValue0，ReduceInputValue1…>做处理。对于MapReduce模型本身，如果要保证计算的正确性，至少要保证的条件是：

单个键的Finer-Partition必须要保证完整的拷贝到同一个Slot上。而不是看起来的，同一个Hash值对应到的Partition的多个键的Finer-Partition数据要保证到同一个Slot上。

多数计算作业中即使多个键的Finer-Partition被Shuffle到同一个Slot上，处理时还是每次以一个键的Finer-Partition作为独立的计算输入单元。这一点也是本文提出的各种负载均衡算法的基本依据。如果用户程序不满足上的条件，那么对于这种应用只能做Reduce任务分配级别的均衡，再低层的均衡会影响程序的正确性。而这种问题，Hadoop本身的推测执行机制基本能够满足需求，因此本文不做讨论。单个Finer-Partition粒度的均衡也是MapReduce均衡算法能达到的上界(单个Key的Finer-Partition是Reduce输入数据的最小不可分单元)。

4.2.2 Hadoop静态增量分区分配负载均衡

MapReduce计算模型包含Map和Reduce两个阶段，Map阶段将输入数据处理，通过一次分区映射到Reduce中。Hadoop系统默认采用Hash的分区方式将Map输出的键值进行划分，系统要求划分数目和Reduce数目相等且一一对应。对与分布均衡的输入数据该方式可以在各个Reduce之间取得较好的计算量的平均分散。然而实际生产应用中数据通常会有一定的聚集性，比如做日志分析时，大多数日志来自少量的几个活跃的用户。默认的分区方式会导致一些密集的键值划分到同一个分区中，从而导致对应的Reduce的负载量过大。文献[47]通过大量实验发现采用默认的Hash分区方法，在92%的任务中出现了Reduce运行的不均衡，而这些倾斜的Reduce运行时间一般都高于正常任务的22%和38%。

本文提出的增量式分区分配策略（IPS，Incremental Partition Scheduling），打破了分区和Reduce一一对应的限制，使用多个更细粒度的分区，在Map执行过程中不断对各个细粒度的分区采样、汇总，分批次将细粒度的分区增量地分配到各个Reduce上，最终组成逻辑上的Reduce输入数据分区。系统概要设计逻辑如下图所示。一般地，将数据分散到更多的分区上能够获得更好的均衡性。更为重要的是，更多的细粒度分区为后续的分区组合提供了回旋余地，即使小分区之间依然存在着倾斜问题，还可以通过小分区的组合达到最终的大分区的数据均衡。

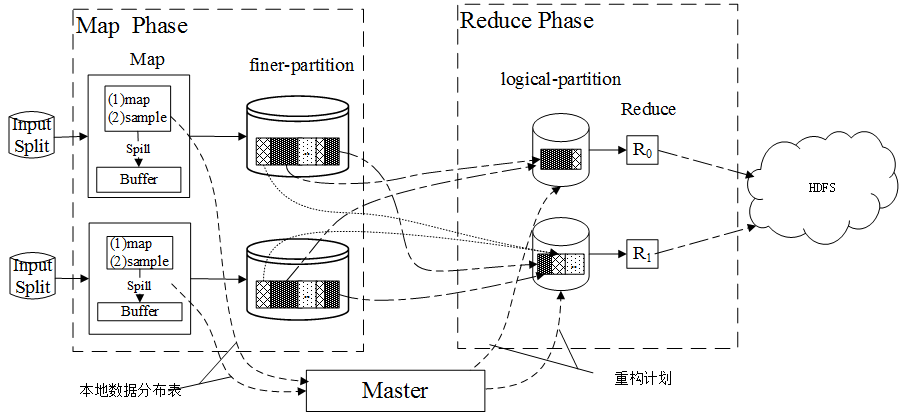


图4-1 IPS逻辑架构图

Hadoop系统原有默认分区方式是通过获得，Reduce\_Num是用户在作业中指定的Reduce个数。IPS尽可能保证原有Hadoop系统的默认假设，为获得更多的细粒度分区，本文引入一个放大系数。在IPS中采用的方式获得个细粒分区。后续工作的目标是将这个细粒度分区分配到个Reduce中，使得各个Reduce上分配到的数据量均衡，本文使用简单的方差作为数据均衡性的衡量标准。下图是IPS对原有Hadoop系统修改后的处理流程逻辑结构设计

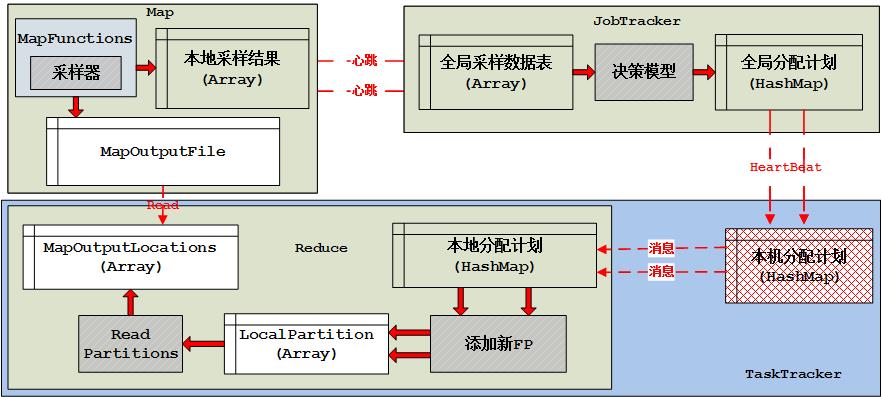


图4-2 IPS实现设计

IPS在Hadoop系统的Map过程中添加了采样模块，Map函数在执行时会将输出键值对写到内存中，采样模块自动在Map输出数据时进行本地采样，记录每个Finer-Partition的元组个数即本地采样数据表。各个Map节点会定期通过Hadoop原有的心跳机制将本地的采样结果发送到JobTracker中汇总。通过不断聚集汇总各个Map节点发送过来的采样信息，JobTracker中将会有一个全局的采样信息二维表，该二维表记录了所有Map Slot上产生的各个Finer-Partition的元组大小。

Map的采样过程是一个不断更新的过程，随着Map任务的执行能够采样的结果就越接近真实结果，JobTracker能够做出的预测也就更加精确。但考虑到Hadoop的Shuffle数据过程，只要有任意一个Map任务完成了就可以开始Map到Reduce的数据传输过程。因此提前分配一些小Finer-Partition到Reduce上可以使得Reduce提前开始数据的Shuffle过程，从而提高系统的执行效率。基于以上考虑IPS中JobTracker的决策模块根据全局采样数据表分批将Finer-Partition分配到各个Reduce上，以期望最终各个Reduce上的数据量方差最小。IPS在初始化作业时先给每个Reduce随机分配一个Finer-Partition，以后决策模块考虑各个Reduce已分配的负载量和数据本地化的基础上不断地分配新的Finer-Partition以修复之前分配后的偏差。

JobTracker在做出决策后将分配计划<Finer-Partition🡪Reduce>先存放在本地，当TaskTracker向它发送心跳信息时，JobTracker会将分配给在该TaskTracker上运行的Reduce的新分配Finer-Partition在应答信息中一起发送到TaskTracker上。TaskTracker会将这些分配信息通过消息机制分别传递给对应的Reduce。

IPS的Reduce与原生Hadoop的Reduce不同，Hadoop中每个Reduce只处理一个分区，该分区号在Reduce初始化时就已经指定了，而IPS中Reduce需要随时处理JobTracker给其指定的新添加Finer-Partition。IPS的Reduce在接收到新添加的Finer-Partition时，首先将该分区添加到自身的已分配分区表中，然后对作业中的每个Map生成该分区的输出文件路径MapOutputLocation，最后ReduceCopier线程根据JobTracker返回的Map完成事件以及本地维护的MapOutputLocations集合从Map端将该Partition对应的数据下载到本地，并等到所有Finer-Partition的数据都取到后最后执行Reduce函数的工作。

4.2.3 Hadoop静态负载均衡存在问题及解决思路

静态负载均衡可以较好地解决数据倾斜的问题，然而实际的生产环境复杂多变，Hadoop系统本身设计目标之一是期望使用廉价异构的集群获得较大的吞吐量。静态负载均衡通过细粒度分区组合使得分区数据均衡，但是如果集群中异构的机器处理能力存在较大的差异，处理时间上依然会出现倾斜问题。另外即使机器同构且数据均衡，一些特定的应用如PageRank等处理代价取决于输入数据的特征，这些情形下静态的负责均衡无能为力。本文提出一种主动请求的任务动态分配模式设计，期望可以使得Hadoop系统可以应用动态的负载均衡策略获得更好的效率和适用性，同时可以在节点故障时获得更及时的恢复能力。

动态均衡借鉴静态均衡的假设，也引入放大系数，通过细粒度的分区期望获得更好的均衡性。在获得细粒度分区后，给启动的所有Reduce分配一个默认的Finer-Partition并按照原生的Hadoop系统机制运行。Reduce从已完成的Map端获取已分配Finer-Partition的数据，该Finer-Partition数据Shuffle完成后即可立即执行reduce函数工作。为充分利用现代计算机的多核架构以及DMA等存取技术，在reduce函数开始处理已经Shuffle完成的Finer-Partition数据时，可以请求并开始下一个Finer-Partition的数据传输。当第一个默认的Finer-Partition处理完成后且第二个Finer-Partition的数据Shuffle到本地后开始第二个Finer-Partition数据的处理以及第三个Finer-Partition的申请。这样可以使得Reduce一边执行的过程中一边从Map端准备下一个Finer-Partition的数据。JobTracker中记录各个Finer-Partition的分配及执行情况，以便在发生节点故障时指派其他机器重新执行该节点的任务。与Hadoop原有的故障恢复机制相比，动态均衡可以在发生故障时只需将分配给故障节点的Finer-Partition重新指派给多个申请新任务的节点，充分利用系统的处理能力提高故障恢复效率。一个完整的Finer-Partition处理过程如下图所示。



图4-3 分区主动获取概念图

上一节Hadoop的静态负载平衡中为了能够尽早且均衡地分配Finer-Partition到Reduce，IPS在原有Hadoop中添加了采样、信息维护和决策模块，这些部分带来了额外的系统开销。Map中添加采样模块以及JobTracker中添加分配决策模块会带来一定的CPU消耗。更重要的问题是JobTracker中采样信息表的维护所带来的内存消耗。假设系统中有n个Reduce，m个Map，放大系数为，一条记录大小使用Java中的long型存储，JobTracker中需要字节的内存消耗。默认配置下，输入数据达到TB级别时，JobTracker中单条作业就需要GB级的内存消耗。相比而言，动态负载均衡策略只带入了极少的系统额外消耗，同时可以解决更普遍的机器异构以及计算倾斜问题，是一种更优的选择。

4.3 本章总结

本章提出了一种针对分布式极大完全图和近似极大完全图的负载均衡方案，通过MapReduce作业的多次迭代，在各个计算单元上从搜索树的任意位置切分均衡负载。根据Hadoop作业特性在极大完全图和近似极大完全图两个算法中引入T和N两个参数，以提高整体作业的扩展性和均衡性。将极大完全图和近似极大完全图负载均衡问题推而广之，介绍了一种静态负载均衡的增量分区分配策略，从系统层面解决了Hadoop数据量负载倾斜的问题。最后探讨了Hadoop系统层上解决计算倾斜的设计思路，为后续工作做基础铺垫。

第五章 实验部分

本文选取多组实际数据以及SSCA和RMAT两种特征可控数据对文中所提出的算法和做了充分的实验。单机算法验证了本文所提出的图分割算法高效性，并行实验验证了负载均衡策略的有效性。实验数据佐证了算法设计中关于切分点选择的讨论。最后实验表明系统层面的负载均衡较好地解决了系统中的数据倾斜问题。

5.1 实验条件及数据

5.1.1 实验条件

软件环境：Ubuntu10.04，JDK1.6，Hadoop-1.1.2。开发环境：Eclipse3.6，JDK1.6。硬件环境：10台1.87GHz的4核Xeon E5502 CPU，16G内存，160G硬盘的曙光系列服务器。

5.1.2 实验数据

表5-1 实验数据集

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 数据集描述 | 顶点个数 | 边数目 |
| D1 | EU研究机构的邮件网络 | 265,009 | 364,481 |
| D2 | 谷歌Web图 | 875,713 | 4,322,051 |
| D3 | Berkeley&Stanford Web | 685,230 | 7,600,595 |
| D4 | Wikipedia通讯网络 | 1,928,669 | 3,494,674 |
| D5 | Pokec在线社交网络 | 1,632,803 | 30,622,564 |
| D6 | Twitter社交圈 | 11,316,811 | 85,331,846 |
| D7 | 编号4952蛋白质交互网络数据 | 5,099 | 681,252 |
| D8 | 编号568206蛋白质交互网络数据 | 5,816 | 313,628 |
| D9 | 编号329726蛋白质交互网络数据 | 8,176 | 457,991 |
| R-MAT | 符合幂律分布和小世界特性的合成图 | 点和边数目可设置 | |
| SSCA#2 | 基于距离度量的分层团边分布 | 点数和最大Clique大小可设 | |

表5-1列举了本文实验中所使用到的数据集，各个数据的解释以及特性。D1-D8是实际世界中的真实网络数据集，其中数据集D9规模非常大，超过了单机以及本文中使用到的集群的计算能力，因此在实验中对其采样抽取部分子图进行实验。在生产应用过程中小于4个节点的Clique和K-Plex结果基本没有实际意义，因此本文所做的实验中默认枚举出所有的节点数不小于4的Clique和K-Plex。

5.2 单机算法实验及分析

单机算法实验使用较小的数据集D1-D5，在一台服务器上从搜索树规模和搜索时间两个方面对比BK及其变种与Binary算法在极大完全图枚举和极大近似完全图枚举两个问题上的表现。利用数据属性可控的模拟数据R-MAT和SSCA#2控制数据的点边比和最大完全图大小对比两类算法的适应性。最后对Binary算法中分裂点度数选择策略，最大、最小和随机三种方式的结果做了验证。

5.2.1 极大完全图枚举

极大完全图枚举以及极大近似完全图枚举单机算法的输入数据都是每行一条<A,B>形式的无向边信息。默认地，无特别说明Binary算法中切分点选择策略都是使用的最小度数切分。关于Binary算法对分裂点选择的敏感性实验再5.2.3节中详细描述。

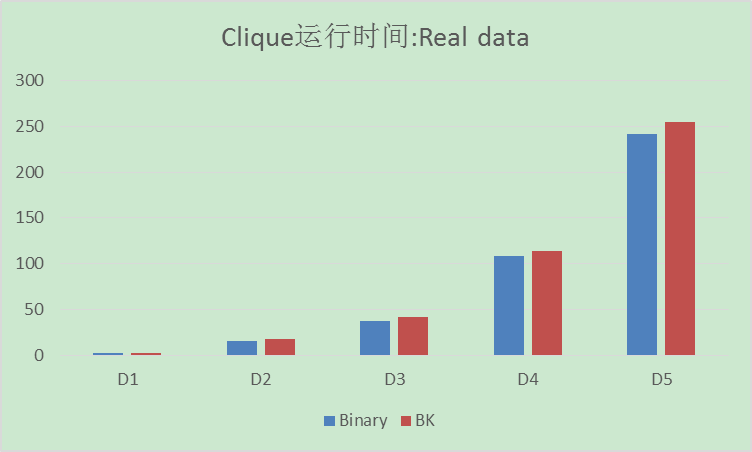
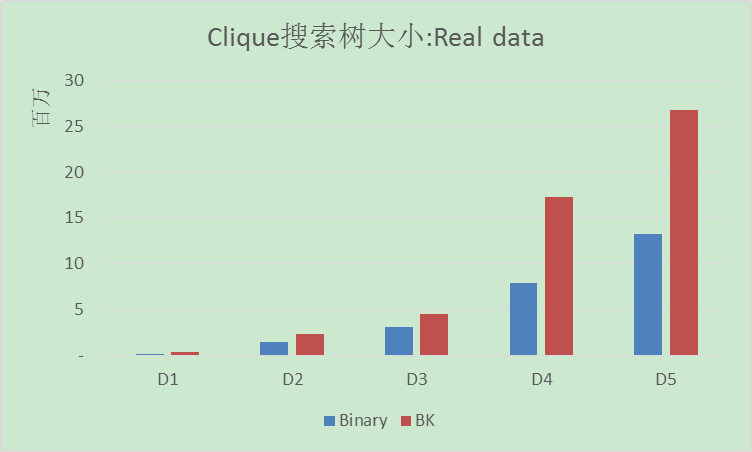


图5-1极大完全图枚举

在图5-1中可以看到Binary的搜索树大小远小于BK算法，一般是BK的一半左右。由于Binary算法过程中需要保证每次选取最小度数点作为切分点，每次切分过程的代价会高于BK一次选择全部切分的策略，但是Binary在搜索树大小上占有绝对优势。总的来说可以从图5-1中看出，极大完全图枚举中Binary算法比BK算法在运行时间上还是有一定的优势。

图5-1中的实验结果使用D1-D5五个较小的实际数据验证了Binary和BK搜索树大小和运行时间，另外本文还使用RMAT和SSCA#2两种属性可控的数据集来测试Binary算法和BK算法在数据特性变化时的适应性。图5-2中显示了RMAT数据在点边比变化时Binary和BK极大完全图枚举算法的搜索树大小和运行时间表现。RMAT数据中下表数据RmatX-Y，X、Y单位为万表示Rmat生成数据时设定生成的数据图中点的个数为X万个边的条数为Y万条，运行时间纵坐标单位为秒。图5-2的实验中测试了100:500到100:5000六个区间点边比RMAT数据Binary和BK算法的性能表现。

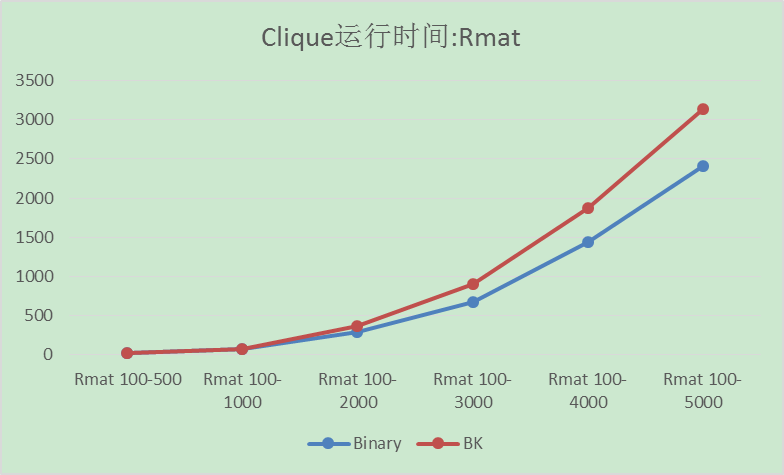
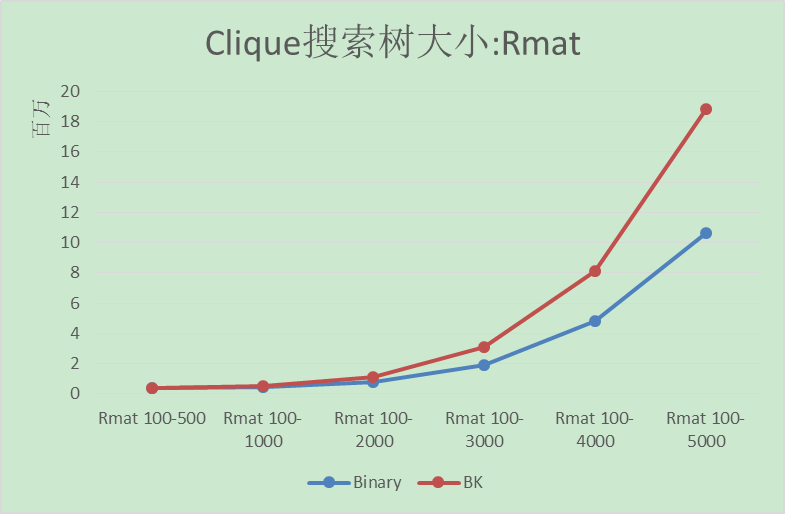


图5-2 RMAT数据特性变化极大完全图枚举算法性能对比实验

图5-3中选用SSCA#2数据设定SSCA#2生成测试数据时点的个数为，控制数据中最大完全图大小由20一直增大到100五个区间点时Binary和BK算法的搜索树大小和运行时间表现。

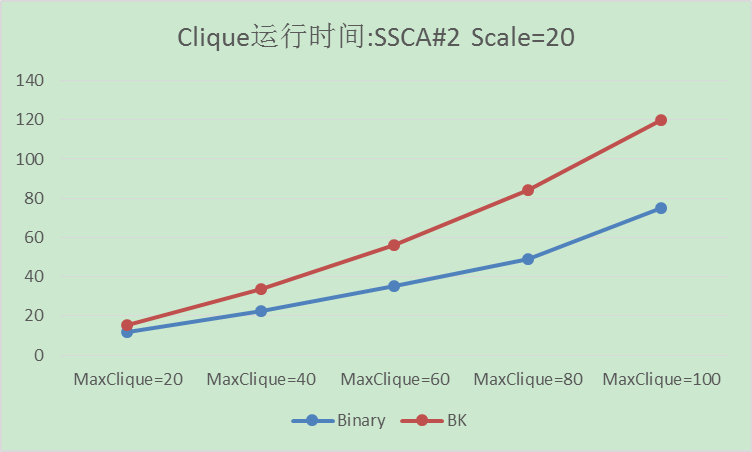
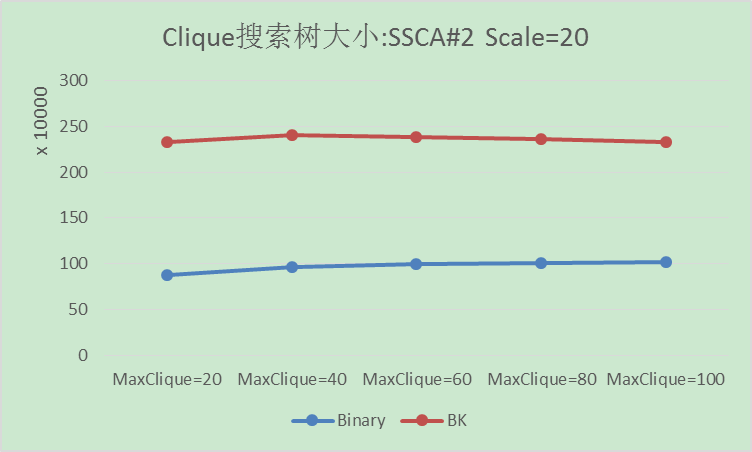


图5-3 SSCA#2数据特性变化极大完全图枚举算法性能对比实验

从图5-2和图5-3的实验可以看出，Binary算法的搜索树大小都要小于BK算法，尤其是在SSCA#2数据集上Binary算法搜索树大小一直稳定在BK算法搜索树大小的二分之一左右。从运行时间上看，随着RMAT点边比增大、SSCA#2中最大完全图大小增加，也就是在数据变稠密的情况下，Binary极大完全图枚举算法比BK极大完全图枚举算法优势更加明显且稳步提升。

至此，整体上看无论是实际数据还是生成数据，Binary算法在运行时间以搜索树空间上都要优于BK算法，但是有一类数据蛋白质交互网络数据D6-D8时出现了反例。如下图5-4所示D6-D8中，Binary算法的运行时间都要远大于BK算法。通过对数据的分析，出现这样的结果是因为蛋白质交互网络中存在大量的点与大部分点都相邻，给Binary的切分带来了极大的代价。通过上文介绍的Hybrid算法的优化，从图5-4中可以看出Hybrid算法在全部数据集上都有不错的表现，Hybrid算法处于Binary和BK之间或者比两者都优。只在D3比两者都高，但D3总运行时间不长，因此微弱的劣势也就显得不是那么重要。

图5-4 Hybrid算法与Binary和BK算法对比

5.2.2 极大K-Plex枚举

极大近似完全图枚举的计算复杂度高于完全图枚举且随着K-Plex中K值的增大其运算复杂度也越来越大，本文选择K=2的2-Plex作为测试问题。枚举出数据集D1-D5中的所有近似极大完全图超过了单机计算能力或可容忍时间，因此本文选取10% 的D1，0.1%的D2-D5中的点来其在全局图中的2-Plex。基于同样的原因，2-Plex的RMAT实验数据也减少了规模，点边比由1万:10万到1万:30万。

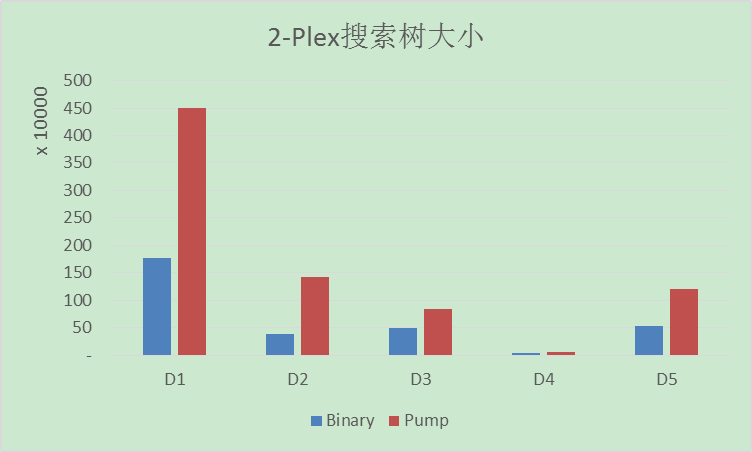
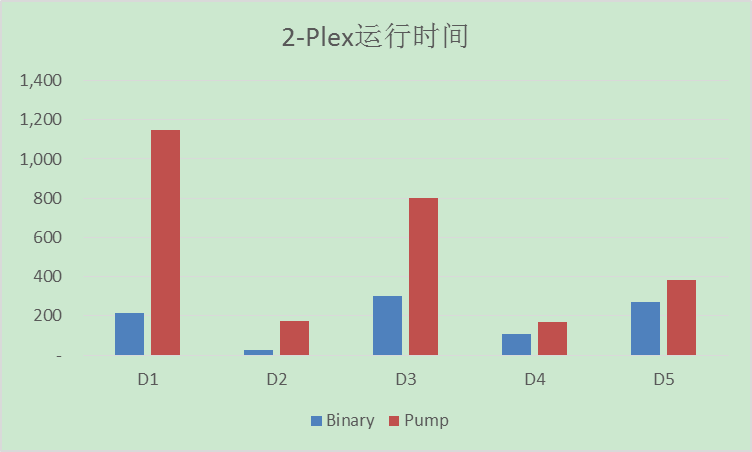


图5-5 2-Plex在实际数据中的性能实验

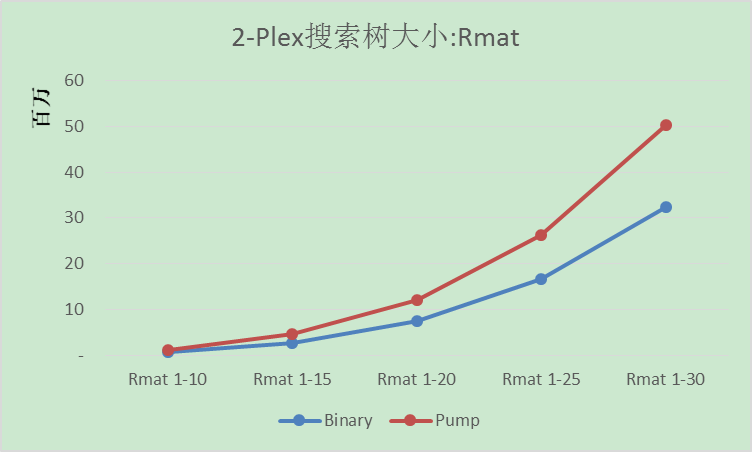
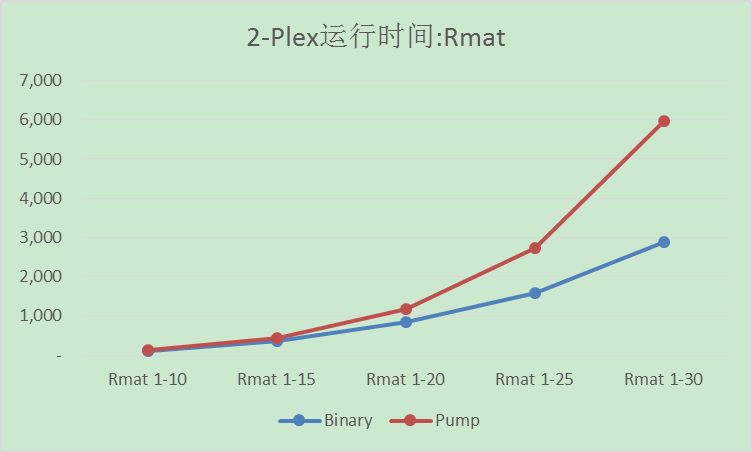


图5-6 2-Plex在RMAT数据中的性能实验

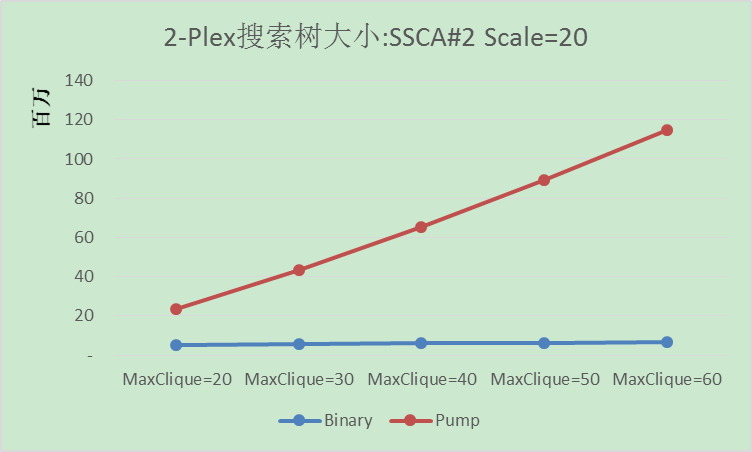
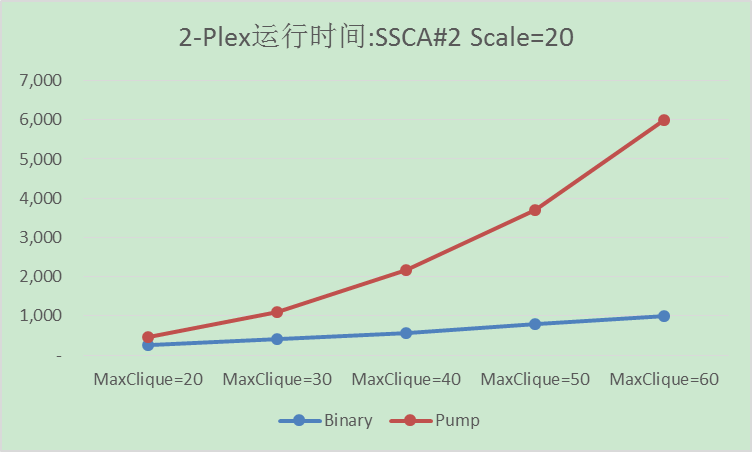


图5-7 2-Plex在SSCA#2数据中的性能实验

与极大完全图枚举的实验结果相似，整体上2-Plex的Binary算法比BK有较大优势，且Binary算法随着数据图的密度增大其优势越来显著。

5.2.3 分裂点选择

Binary算法中关于分裂点选择的最大度数、最小度数和随机度数选取三种策略。经过实验验证了正文中相关分析，一般地，最大度数切分的效果最差，其次随机，最好的是使用最小度数切分。因此文中的其他Binary实验中都选取最小度数切分策略。K-Plex中关于分裂点选择的问题有相似的结果，不再赘述。

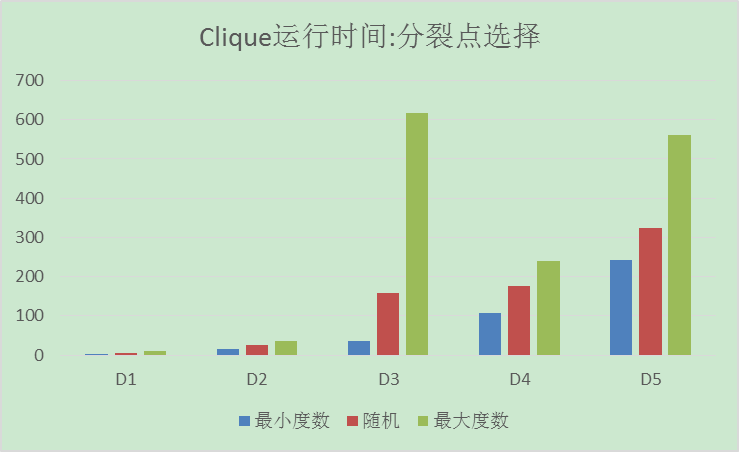
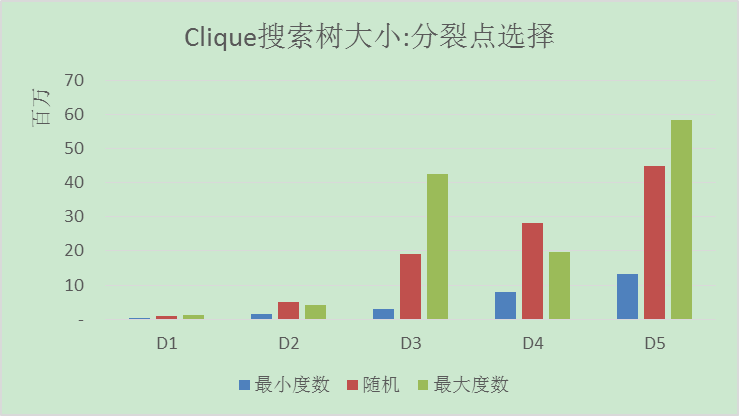


图5-8 Binary算法分裂选择策略

5.3并行算法实验及分析

并行算法中本文将Binary算法与BK的及其变种算法做对比，由于所有的算法都使用同样方法获取同样的输入数据图，这样排除了由于数据图获取或输入使用的数据图带来的影响。并行极大完全图中使用三角形数据作为输入数据，并行极大近似完全图中使用两跳数据作为输入数据。并行算法中由于用户一般关注运行时间且作业启动及准备时间对MapReduce计算模型有较大影响，本文主要考量总运行时间和作业迭代次数两个指标衡量算法的性能。

5.3.1 并行极大完全图枚举

在并行算法部分本文引入了两个参数T、N，两个参数的设置直接影响到算法的整体效率。本文经过多组实验选取各种不同的T、N组合，找到算法表现最好的一组参数作为算法的输入设置。图5-9显示了一个典型的实验结果，本节中极大完全图枚举算法的设置中选取最优设置T=300S，N=50。较小的一些数据D1-D5可以使用单机在很快的时间内处理，因此使用并行实验并不合适。另外SSCA#2的数据在一次发散后就已经取得了较好的均衡不存在负载均衡问题，这里也不再展示其并行结果。

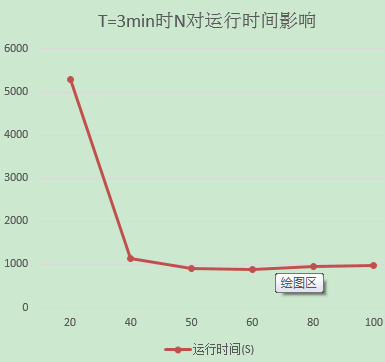
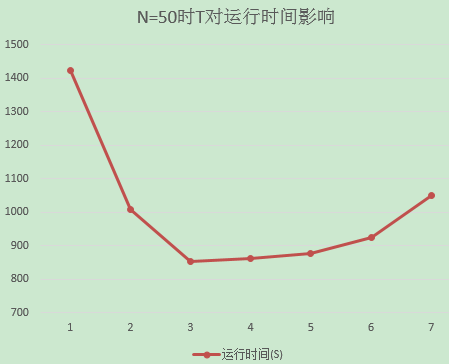


图5-9 参数T、N设置对于算法性能影响

本文选取最大的实际数据集D6 Twitter中的部分采样点做为测试数据，在5个样本的实验中都显示了Binary并行化算法比BK并行化算法具有更少的处理时间以及更少的迭代次数。从图5-10中可以看出不采用负载均衡方案的情况下其运行时间都明显高于采用了负载均衡方案的两种方法，尤其是在样本2和样本3上其运行时间已经超过了4小时，从而进一步证明了负载均衡方案的重要性。

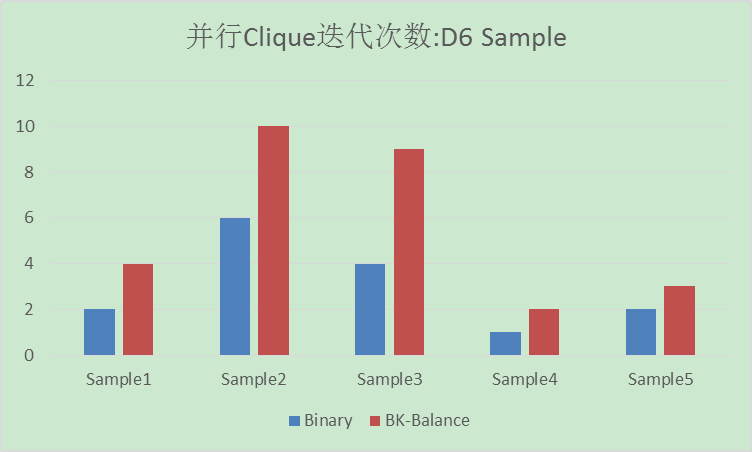
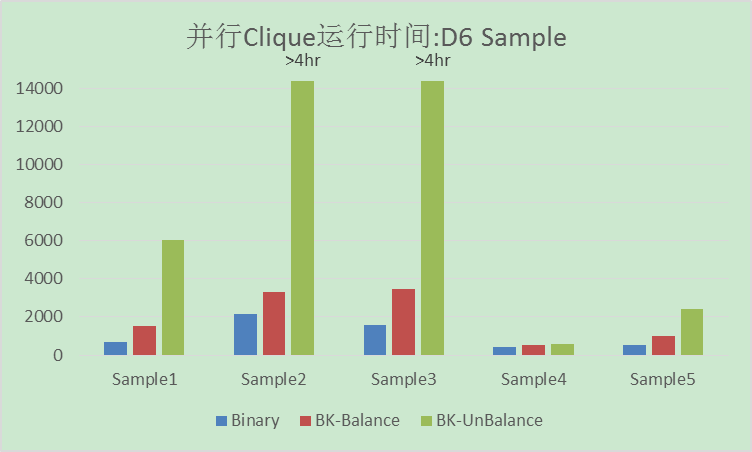


图5-10 D6的多个样本数据并行Clique实验

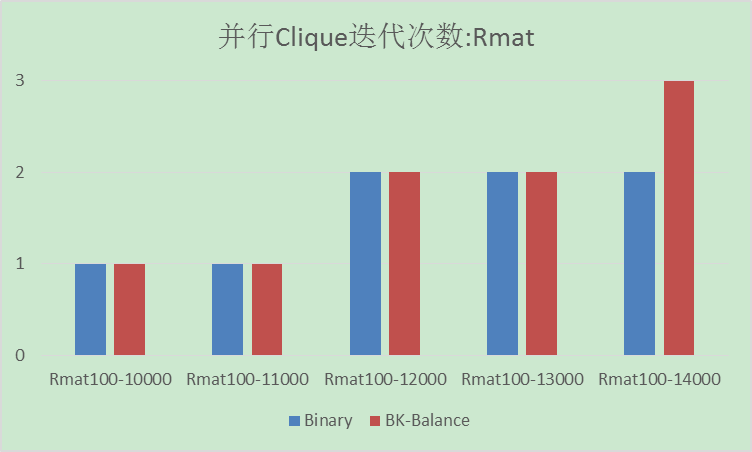
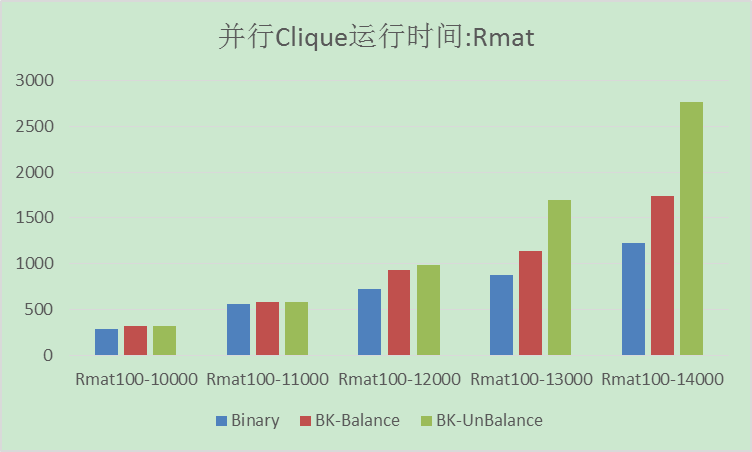


图5-11 并行Clique在RMAT变化时性能表现

在并行的算法中本文再次增加了RMAT数据图的规模，图5-11中点边比的单位是万，可以看到随着数据规模和点边比的增加，并行Binary算法的优势逐渐体现。

5.3.2 并行极大K-Plex枚举

由于K-Plex计算的复杂度非常高，并行算部分依然采用K=2的2-Plex作为实验内容，另外实际数据D3-D5无法再本文的计算集群中全部处理，因此对数据集D3-D5，本实验中随机取1%的点作为采样点，计算这些点的2-Plex。与并行极大完全图枚举相似地，K-Plex也需要将参数T和N设置为其最优情况，在2-Plex实验中T设置为500s，N=50。可以在实验结果图5-12中看到，并行2-Plex实验中Binary表现出比对比算法极大的优势。

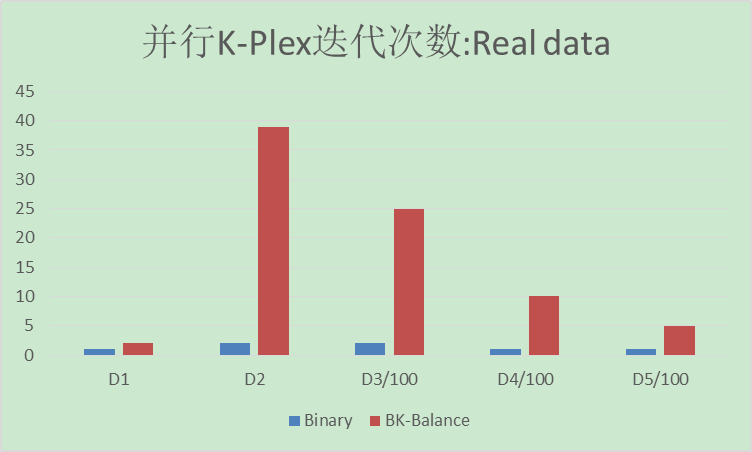
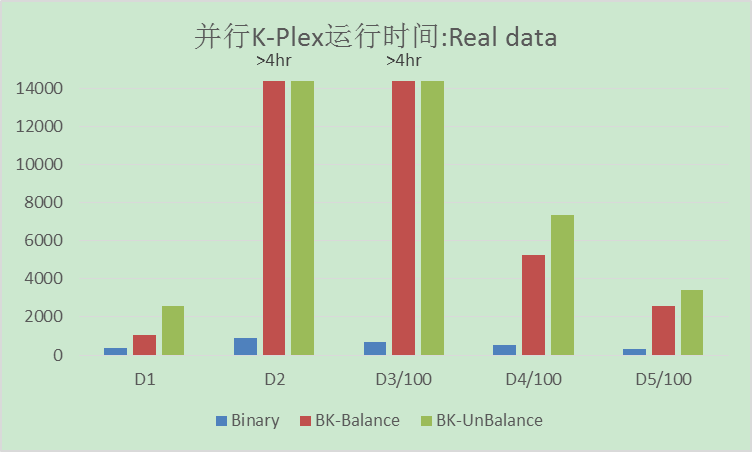


图5-12 并行2-Plex在真实数据中性能表现

5.3.3 并行算法加速比

并行算法的加速比是并行算法可用性的一个重要衡量标准。本文实验中将集群机器数目由2增加到10，观察在各种数据上的集群的加速比情况。

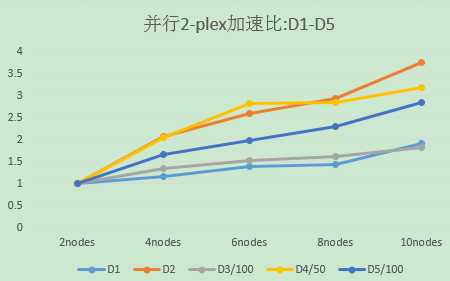
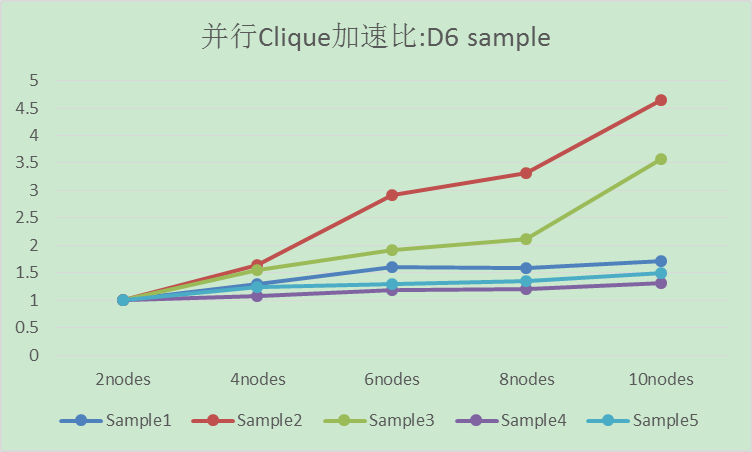


图5-13 并行加速比实验

本文对Binary算法并行完全图枚举以及2-Plex并行枚举在实际数据集上做了相关测试，图5-13中可以看出算法在大多数时候取得了较好的加速比。但在样本1、4、5的Clique枚举中加速比不明显，这是由于最少的两个节点时作业的迭代次数就很小，增加机器节点后由于初次任务划分的不均衡，不能带来明显的加速。

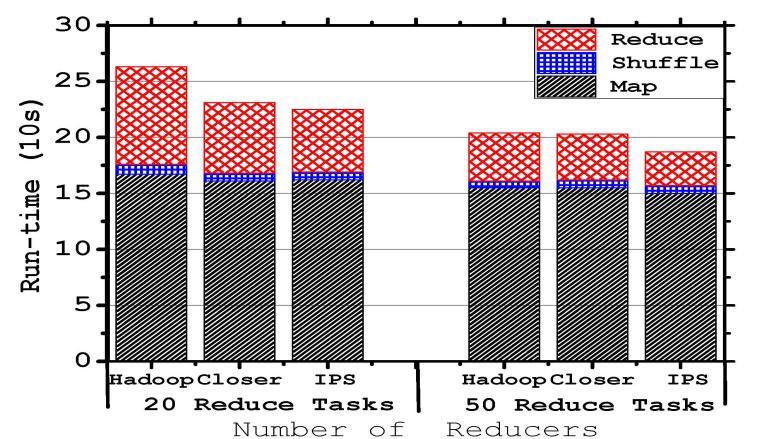
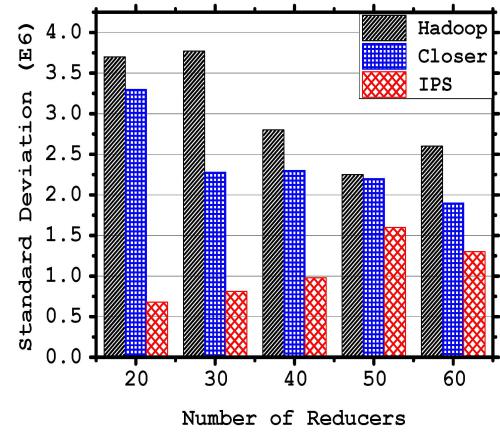
5.4 Hadoop系统负载均衡实验及分析

在Hadoop系统负载均衡实验部分实验数据采用真实数据集和合成数据集，两类数据集的详细描述如表5-2所示。合成数据包含11个数据子集，每个子数据集满足指数为的标准Zipf分布，取值为1到10的整数，的值越大，数据的分布越倾斜。各子集为单列整数数值数据，数值的取值范围为[1,1000]，并包含10亿条记录。为满足实验用例特点，对真实数据集做去杂处理，由于杂质记录很少，此操作并未改变原数据集分布特点。

表5-2 Hadoop系统负载均衡实验数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据集 | 运行实例 | 大小(GB) | 元组数(亿) | 描述 |
| Zipf-γ | WordCount | 0.4 | 10 | 标准Zipf分布 |
| BTS | WordCount | 60 | 190 | US航空数据 |
| UK-2002 | PageRank | 4.7 | 1.8 | 2002年网站爬的Uk数据 |

本文选取WordCount、PageRank两种Hadoop上的典型作业作为验证算法，将本文中提出的IPS模型和原生Hadoop系统以及对比系统Closer[10][37]做对比实验。



(a) (b)

图5-14 系统负载均衡实验

图5-14 (a)是三系统的WordCount对比实验结果，横坐标是集群ReduceTask的数目。对于10个处理节点的集群，当ReduceTask数目较少时，Hadoop和Closer的均衡效果都比较差，而IPS却基本能保证各节点具有相同的数据接收量。在ReduceTask数目为50时，IPS的均衡性最差，说明此时所采用的增量参数没有使得系统产生好的效果，受系统参数的影响，但均衡性都要明显优于其他两者。

集群中数据分布的均衡性最终反映到运行时间上，由于WordCount在Reduce端只进行简单的相加操作，运行时间效果的差异性不能明显地反映出来，因此本文在Reduce端进行相对复杂运算的PageRank算法。图5-14(b)是在真实数据集UK-2002上运行RageRank一次迭代的结果。在Reduce阶段，可以发现IPS始终是三个系统中最优的，在ReduceTask为20时，IPS与Closer运行时间基本相同，但基本是Hadoop运行时间的一半。在ReduceTask为50时，Closer与Hadoop基本有相同的运行时间，而IPS却有一倍的提升。

5.5 本章总结

本章主要介绍了实验条件与实验数据，通过充分且全面的对比实验验证了本文提出的Binary算法和Hybrid算法的有效性和高效性。单机算法表明Binary算法比BK算法具有更小的搜索树空间且整体上运算时间更优，为并行算法打下良好基础。进而由并行实验验证了Binary算法及其负载均衡方案具有良好的并行性以及可扩展性，可以通过机器的简单叠加获得整体处理效率的提升。最后通过实验结果证实了系统层面的负载均衡可以对Hadoop系统起到较好的自动均衡负载的作用，提升作业的响应速度。

第六章 总结与展望

6.1 本文研究总结

本文提出了一种基于图分割的极大完全图枚举和近似极大完全图枚举算法Binary。在大多数据集上与经典对比算法BK相比，新算法能够有效地减少极大完全图枚举和近似极大完全图枚举过程中搜索树大小，同时节约搜索时间，提升搜索效率。在分析两种算法各自优劣的情况下，以Binary为主，BK为辅，提出一种结合两者优势的混合算法Hybrid。Hybrid获得了两者在性能和搜索空间上的权衡，避免了极端差的情况出现，甚至在一些连接度较高的蛋白质数据上获得比两者都好的表现。

针对本文提出的Binary算法以及完全图枚举和近似完全图枚举的特定应用场景，本文提出了该算法在Hadoop平台上的并行化方案以及其负载均衡方案。进行负载均衡后的Binary并行化算法比同样条件下的BK算法具有更少的迭代次数和运行时间，且具有较好的扩展性。

最后本文提出一种Hadoop系统层面的增量分区分配均衡策略IPS，与原生Hadoop系统以及解决相似问题的Closer系统相比，IPS具有比另外两者都更好的均衡性和适用性。在分析IPS存在的问题的前提下，本文提出一种任务主动获取的动态均衡设计思路，期望以此解决Hadoop平台下的计算倾斜问题。

6.2 课题研究展望

本文提出的Binary算法需要在切分过程中持续维护最下度数集合，每次选取最小度数点作为切分点，导致平均单次切分过程中切分代价大于BK算法，如何减少Binary算法的单次切分代价是一个可以继续探讨的问题。

在近似完全图枚举过程中，其剪枝条件与完全图一样，要求存在Not集中的一个点与候选集以及结果集中的点都相邻，才可保证此分支都是冗余的。但近似完全图的条件弱于完全图，剪枝条件相对也是可以进一步放松的，找到一种较弱的剪枝条件将能够极大地提升近似完全图枚举的效率。本文使用Hadoop作为并行计算平台，业界还有许多优秀的广泛应用的并行计算平台，Binary算法在这些平台上的表现值得进一步验证与优化。

另外，本文所讨论的问题是完全图和近似完全图的枚举。在实际生产环境中，可能对完全图和近似完全图的查询工作可能会有更多的需求。同时注意到在实际生产环境中，输入图数据的网络结构、点边等信息是动态变化的。显然对于这样变化的图数据，每次重新挖掘枚举将会带来极大的冗余工作。如何在并行平台上应用Binary思想高效地实现这两种问题的并行查询处理，同时兼容增量更新也是一个重要挑战。

对于Hadoop等MapReduce计算平台中更广为存在计算倾斜问题，本文只是提出了一种设计思路，按照IPS的设计模式实现其Finer-Partition的动态传输与执行，验证了其可行性。但具体在什么样的时机去获取新Finer-Partition、Finer-Partition是否直接随机分配还是有序分配、恢复及推测执行策略如何实现等详细设计问题是下一步的工作方向。

# 参考文献

1. C.Bron and J.Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. Communications of the ACM, 16(9):575–577, 1973.
2. Li Lu and Yunhong Gu. D MaximalCliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution. IEEE International Conference on Data Mining Workshops. 2010.
3. F.Cazals and C.Karande. A note on the problem of reporting maximal cliques. Theoretical Computer Science, 407(1):564–568, 2008.
4. Shengqi Yang, Bai Wang and Haizhou Zhao. Efficient dense structure mining using MapReduce. IEEE International Conference on Data Mining Workshops. 2009.
5. D.Eppstein and D.Strash. Listing all maximal cliques in large sparse real-world graphs. In 10th International Symposium on Experimental Algorithms, 2011.
6. Bin Wu and Xin Pei. A parallel algorithm for enumerating all the maximal k-plexes. PAKDD 2007 workshops, LNAI 4819, 476-4831.
7. Arash Khosraviani and Mohsen Sharifi. A distributed algorithm for γ-Quasi-Clique extractions in massive graphs. INCT, 422-431, 2011.1
8. E.Tomita, A.Tanaka, and H.Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science, 363(1):28–42, 2006.
9. S.Tsukiyama, M.Ide, and I.Shirakawa. A new algorithm for generating all the maximal independent sets. SIAM Journal on Computing, 6(3):505–517, 1977.
10. B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on, pages 522–533. IEEE, 2012.
11. N.Chiba and T.Nishizeki. Arboricity and subgraph listing algorithms. SIAM Journal on Computing, 14(1):210–223, 1985.
12. E.A.Akkoyunlu. The enumeration of maximal cliques of large graphs. SIAM Journal on Computing, 2(1), 1973.
13. K.Makino and T.Uno. New algorithms for enumerating all maximal cliques. In SWAT, Lecture Notes in Computer Science 3111, pages 260–272, 2004.
14. V.Stix. Finding all maximal cliques in dynamic graphs. Computational Optimization and Applications, (27):173–186, 2004.
15. Robert E. Osteen and Julius T. Tou. A clique detection algorithm based n neighborhoods in graphs. International Journal of Computer and Information Sciences, Vol.2, No.4, 1973.
16. D. Eppstein, M. L ofﬂer, and et al. Listing all maximal cliques in sparse graphs in near-optimal time. In ISAAC(1), pages 403–414,2010.
17. Thang Nguyen Bui and Paul H. Eppley. A hybrid genetic algorithm for the maximum clique problem. In Larry Eshelman, editor, Proceedings of the Sixth International Conference on Genetic Algorithms, 478-484, 1995.
18. S.Tsukiyama, M.Ide, and I.Shirakawa. A new algorithm for generating all the maximal independent sets. SIAM Journal on Computing,6(3):505–517, 1977.
19. James Cheng, Yiping Ke, Ada Wai-Chee Fu and Jeffrey Xu Yu. Finding maximal cliques in massive nerworks by h\*-graph. SIGMOD, June 6-11, 2010.
20. S. C. Racha. Load balancing map-reduce communications for efficient executions of applications in a cloud. 2012.
21. R.Rama, Suresh Badarla and Kamala Krithivasan. Clique detection algorithm using clique-self-assembly. 6th International Conference on Bio-Inspired Computing: Theories and Applications, 2011.
22. Jose L. Walteros and Panos M. Pardalos. A decomposotion approach for solving critical clique detection problems. SEA 2012, LNCS 7276, 393-404, 2012.
23. B. Wu, S. Yang, and et al. A distributed algorithm to enumerate all maximal cliques in mapreduce. In International Conference on Frontier of Computer Science and Technology, pages 45–51, 2009
24. L. Lu, Y. Gu, and et al. dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution. In IEEE International Conference on Data Mining Workshops, pages 1320–1327, 2010.
25. S. Yang, B. Wang, and et al. Efﬁcient dense structure mining using mapreduce. In IEEE International Conference on Data Mining Workshops, pages 332–337, 2009.
26. M.C.Schmidt, N.F.Samatova, K.Thomas, and B.H.Park. A scalable, parallel algorithm for maximal clique enumeration. Journal of Parallel and Distributed Computing, (69):417–428, 2009.
27. J. Cheng, L. Zhu, and et al. Fast algorithms for maximal clique enumeration with limited memory. In KDD, 2012
28. Mattew C.Schmidt. A scalable, parallel algorithm for maximal clique enumeration. Journal of Parallel and Distributed Computing, Vol 69, pp 417-428, 2009.
29. S Szabo. Parallel algorithms for finding cliques in a graph. 5th International Workshop on Multi-Rate Processes and Hysteresis. Journal of Physics: Conference Series, 2011.
30. B.Balasundaram, S.Butenko, I.V.Hicks, and S.Sachdeva. Clique relaxations in social network analysis: the maximum k-plex problem. Operations Research, 59(1):133-142, 2011.
31. M.Haraguchi and Y.Okubo. A method for pinpoint clustering of web pages with pseudo-clique search. Federation over the web, Vol.3847 of Lecture Notes in Computer Science, pp 59-78, 2006.
32. Guimei Liu and Limsoon Wong. Effective prunning techniques for mining quasi-cliques. Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science, volume 5212, pp 33-49, 2008.
33. James Abello, Mauricio G.C. Resende, and Sandra Sudarsky. Massive Quasi-clique detection. LATIN’02 LNCS 2286, 586-612, 2002.
34. B.McClosky and I.V.Hicks. Combinatorial algorithms for the maximum k-plex problem. Journal of Combinatorial Optimization, (23):29–49, 2012.
35. Bin Wu and Xin Pei. A parallel algorithm for enumerating all the maximal k-plexes. PAKDD 2007 workshops, LNAI 4819, 476-483.
36. Arash Khosraviani and Mohsen Sharifi. A distributed algorithm for γ-Quasi-Clique extractions in massive graphs. INCT, 422-431, 2011.
37. G. Benjamin, A. Nikolaus, R. Angelika, and A. Kemper. Handing data skew in mapreduce. In Proceedings of the 1st International Conference on Cloud Computing and Services Science, volume 146, pages 574–583, 2011.
38. Kwon Y C, Balazinska M, Howe B, et al. Skewtune: mitigating skew in mapreduce applications[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 25-36.
39. 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[J]. 计算机学报, 2011, 34(10): 1753-1767.
40. Jian Pei, Daxin Jiang and Aidong Zhang. On mining cross graph quasi-cliques. KDD’05 August 21-24, 2005.
41. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
42. MapReduce in <http://zh.wikipedia.org/wiki/MapReduce>
43. Hadoop in <http://hadoop.apache.org/>
44. Y. Zhang, F. Abu-Khzam, N. Baldwin, E. Chesler, M. Langston, N.F. Samatova, Genome-scale computational approaches to memory-intensive applications in systems biology, in: Proc. of the 2005 ACM/IEEE Conference on Supercom-puting, 2005, p. 12.
45. F. Kose, W. Weckwerth, T. Linke, O. Fiehn, Visualizing plant metabolomics correlation networks using cliquemetabolite matrices, Bioinformatics 17 (12)(2001) 1198-1208.
46. N. Du, W. Bin, X. Liutong, W. Bai, P. Xin, A parallel algorithm for enumerating all maximal cliques in complex network, in: Proc. of ICDM Workshops, 2006,pp. 320324.
47. Kai Ren, Garth Gibson, YongChul Kwon, Magdalena Balazinska, Bill Howe: Poster: Hadoop's Adolescence; A Comparative Workloads Analysis from Three Research Clusters. SC Companion 2012: 1453

# 致谢

在本文完成后，我即将结束为期三年的研究生生涯，我衷心地感谢在我硕士期间，所有对我的学习、工作及生活给予帮助的人。

首先，感谢我的硕士生导师陈群教授，感谢他在我研究生期间对我的关心和帮助。生活中，陈老师像家长一样，常常与我进行沟通与交流，并给予我鼓励与安慰，使我倍感温暖；工作中，陈老师一丝不苟，从严治学，他严谨的工作风格给我留下了深刻的印象，是我以后科研及工作的榜样。我毕业论文的顺利完成，与陈老师的悉心指导是分不开的。再一次向陈老师致以真诚的敬意。

我还要感谢我的另一位导师，潘巍老师。在进行项目研究的过程中，潘老师丰富的经验以及扎实的理论功底，使我受益匪浅，收获颇丰。无论是研究点的设立、算法的设计，还是科研文章的撰写、做科研的态度，潘老师都会一一耐心地教导我，使我逐步走进科研的殿堂，同时，潘老师活跃的思想，谦逊的性格以及追求真理的态度是我以后科研道路上的榜样。

同时，我要感谢教研室的各位同学们，是他们在我遇到技术难关时，及时给予帮助，在我困惑于一个科研问题时，牺牲时间与我讨论，再一次感谢他们。

最后，我要特别感谢我的父母，我的家人，他们是我学习、科研的坚强后盾，一直在为我默默地付出，无私地奉献。无论我遇到什么困难，什么挫折，他们总会站在我身边，给我最强有力的支持，他们的支持是我顺利完成学业的保证。

**西北工业大学业**

**学位论文知识产权声明书**

本人完全了解学校有关保护知识产权的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属于西北工业大学。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西北工业大学。

保密论文待解密后适用本声明。

学位论文作者签名： 指导教师签名：

年 月 日 年 月 日

——————————————————————————————————

**西北工业大学**

**学位论文原创性声明**

秉承学校严谨的学风和优良的科学道德，本人郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容和致谢的地方外，本论文不包含任何其他个人或集体已经公开发表或撰写过的研究成果，不包含本人或其他已申请学位或其他用途使用过的成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式表明。

本人学位论文与资料若有不实，愿意承担一切相关的法律责任。

学位论文作者签名：

年 月 日