文本复制检测报告单(全文标明引文)

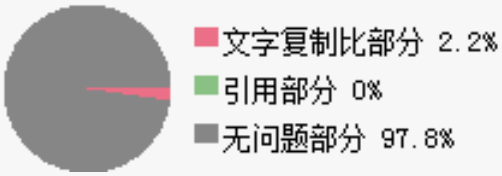
№:ADBD2018R_20170310213717201801090820184015126729 72 检测时间:2018-01-09 08:20:18

检测文献: 1_赵炳_基于分布式图计算的大规模网络分析系统的研究
作者: 赵炳
检测范围: 中国学术期刊网络出版总库
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库
中国重要会议论文全文数据库
中国重要报纸全文数据库
中国专利全文数据库
互联网资源(包含贴吧等论坛资源)
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)
港澳台学术文献库
优先出版文献库
互联网文档资源
图书资源
CNKI大成编客-原创作品库
学术论文联合比对库
个人比对库
时间范围: 1900-01-01至2018-01-09

检测结果

总文字复制比：2.2% 跨语言检测结果：0%
去除引用文献复制比：2.2% 去除本人已发表文献复制比：2.2%
单篇最大文字复制比：0.7%（基于Hadoop平台的广告检测系统研究与实现）

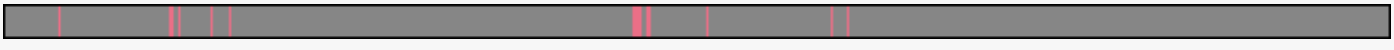
重复字数：	[831]	总段落数：	[7]
总字数：	[37626]	疑似段落数：	[3]
单篇最大重复字数：	[257]	前部重合字数：	[261]
疑似段落最大重合字数：	[570]	后部重合字数：	[570]
疑似段落最小重合字数：	[85]		



指 标：☐疑似剽窃观点 ☑疑似剽窃文字表述 ☐疑似自我剽窃 ☐疑似整体剽窃 ☐过度引用

表格：1 脚注与尾注：0

0%（0）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第1部分（总745字）
3.8%（176）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第2部分（总4596字）
1.1%（85）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第3部分（总8051字）
5.2%（570）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第4部分（总10952字）
0%（0）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第5部分（总7566字）
0%（0）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第6部分（总4907字）
0%（0）	1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第7部分（总809字）



（注释：无问题部分 文字复制比部分 引用部分）

1. 1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第1部分 总字数：745

相似文献列表 文字复制比：0%(0) 疑似剽窃观点：(0)

原文内容 红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

随着互联网的飞速发展，原本相互独立或者关系稀少的数据联系越来越紧密，产生了大量带有复杂关系的数据。这些复杂的关系需要图分析技术才能准确的反映数据的真实状态。但是一方面现有的图计算技术都是对静态图的分析，然而现实世界中的数据模型都是时刻变化的，现有的静态计算模型不足以反映网络的真实状态。另一方面由于图的结构复杂，导致图计算的复

杂度很难降低下来，分布式的成本也居高不下。这些原因一直制约着图分析方向的前景和图计算的发展。

本文中基于这两个关于图方面分析的痛点都出了自己的方案，首先针对于现有的图分析都是基于静态图的痛点，我们设计了适合时序图存储的分布式图存储系统TSGraph，存储模型使用BigTable模型，使用HBase作为后端存储。通过存储结构的优化，RowKey的设计，查询索引的优化，可以在时间维度上快速查询到动态图的某个静态快照。针对现有的图计算算法复杂度居高不下的问题我们，我们基于BSP的思想，使用Spark GraphX的API设计实现了增量式图算法。通过实验表明我们的增量式图算法在一定的准确度丢失的容忍程度下，可以极大地提高图计算的执行效率。同时针对增量式算法会丢失准确度的这个问题，我们设计了全量校正的方案。可以保证我们的计算结果一直在某个准确度丢失的容忍范围内。

最后我们基于以上提到的动态图存储系统和增量式计算系统实现了一个支持动态图计算的综合图分析平台OpenGraph。该图分析平台主要有四个某块，包括数据的管理加载，图关系得查询，增量式算法的实时运行，全面的图属性分析指标。通过该分析平台，用户可以快速的对整个图有一个了解，并且可以根据自己的需求计算复杂的实时性任务。

关键词：TSGraph OpenGraph 动态图增量计算图快照 Spark

2. 1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第2部分

总字数：4596

相似文献列表　文字复制比：3.8%(176)　疑似剽窃观点：(0)

1	面向云服务的超算中心资源综合调度关键技术研究与实现	1.5%（70）
	王春光(导师：吴泉源;吴庆波) -《国防科学技术大学博士论文》- 2015-01-01	是否引证：否
2	论农民工的社会保障权利	1.3%（59）
	张会娟(导师：苗连营) -《郑州大学硕士论文》- 2012-05-01	是否引证：否
3	锂离子电池正极材料LiCoO_2的改性及其薄膜制备研究	1.0%（45）
	戴新义(导师：李晶泽) -《电子科技大学博士论文》- 2016-03-15	是否引证：否
4	计算机组卷与考试系统	1.0%（45）
	吴焱焱(导师：姚望舒) -《苏州大学硕士论文》- 2016-11-01	是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第一章绪论

1.1 研究的背景与意义

随着信息技术的发展，互联网上的信息规模出现了爆炸式增长，如今互联网网页接近47亿，而且用户数目也突破了30亿。与此同时，互联网上的服务模式也越来越丰富，信息之间的联系越来越紧密。超连接图（hyperlink）将不同的网页连接起来，知识图谱（knowledge graph）将不同的实体（entity）联系起来，而社交网络（social network）又将不同的用户连接起来。

这些大规模，高度结构化的数据很大程度的反应了真实世界中的关系，蕴藏着巨大的研究和商用价值。因此，相关领域也出现了大量的图分析算法，他们通过计算数据的结构化特征，提取出重要的信息。其中，具有代表性的技术包括排序（ranking）技术，社区群体（community）分析技术，话题（topic）分析技术等。

在社交网络分析[1]、互联网搜索[2]以及知识图谱等领域里面。图由于其强大的高维度表示能力被广泛的使用。近些年来人们越来越意识到了对于图结构的存储和计算的重要性，一批图数据库和图计算框架相继涌现出来，并且还取得了不错的反响。这其中比较有代表性的有用于存储的Neo4j，TitanDB和用于计算的GraphLab[3]，Spark GraphX[4,5]等。

现有的分布式图计算系统通常是采用边分割或者点分割的方式将一个大图分布式存储到集群里单独计算机的内存上，被分隔开的点或者边都会在其他机器上存在备份。每个节点计算完毕后会把计算结果同步到其他有该节点备份的主机上面。文献[3]提出了一个基于图像处理的开源图计算框架GraphLab，该框架是一个面向机器学习的流处理并行计算框架。GraphLab将数据抽象成Graph结构，并且将计算过程抽象成Gather、Apply、Setter三个阶段，Gatther阶段所有节点从邻接节点接收数据，并且进行计算，Apply阶段将Gather节点计算得到的值发送到Master上，由Master汇总再进行计算并且更新节点值，Setter阶段将会更新产生变化的节点相邻的边信息，并且通知相邻接点该节点已产生变化。文献[4]基于Spark的底层结构实现了一个分布式图计算框架GraphX，它充分的利用了Spark的分布式内存模型RDD的优点，通过Spark的并行处理能力和Pregel[6]的原理，可以实现大部分图计算任务，速度较MapReduce有了很大的提升。文献[7,8]从图分割的角度出发，通过良好的划分算法可以有效的减少分布式系统的通信代价。

现在虽然已经有了增量图计算的方式，但是其算法适用范围受限，并且没有完整的平台系统支持图的存储和分析。因此研究出一套完整的支持大规模动态图存储和分析的系统就非常重要了。

1.2 国内外研究现状

1.2.1 图存储

目前基于图的存储主要有三个主流的方向，一个是基于关系型数据库构建图数据库，这种类型主要是基于当前的RDBMS做一些的修改，仍然使用扩展度非常高的PostgreSQL作为与之对应的查询语言。因为关系型数据库有着几十年的工程积累，通过大量的实践积累了相当丰富的设计经验，所以基于这方面的扩展还是有很多支撑的，近些年也一直有论文讨论相关的话题

[]。 这一类数据库的一个典型代表就是微软的GraphView。

另一派是以Neo4j为代表的称之为原生的图数据库，这种数据库摒弃了关系型数据库的设计模式。主要特点是查询一个点的边或者是边上的端点时不用再次线性查找或者重新走一遍B+树索引，而是直接用指针指向下一度的物理地址，基于关系的查找使用遍历链表的方式实现。它的双向链表结构在内存足够大或者是有SSD盘辅助的情况下查找性能还是不错的，但是在内存不够的时候，性能上会有稍微的折损，但是总的来说还是一个不错的选择。

还有一派呢，则是以JanusGraph为代表的使用了NoSQL存储的分布式图数据库。目前的产品实现方案主要是在NoSQL数据库（比如HBase， Cassandra）上封装了一层逻辑的图结构，存储和查询分离，目前的性能提升空间还有很大。

三中存储方式各有千秋，也各自有难以克服的缺点。图查询的本质难题是数据高度关联带来的大量的随机访问，传统的关系型数据库很难解决这个问题，因为他们是面向磁盘优化，最大化的利用了磁盘顺序读写的优势，对于图数据的查询却未必适用。

1.2.2 图计算

通用的大规模数据存储分析平台，大部分已经集合了并发执行，作业调度，容错管理等一系列复杂的功能，想要使用的话只要部署了相应的环境，并且调用通用的接口就可以实现大部分的算法的执行。目前比较知名的大规模分布式计算平台主要有Google提出的的MapReduce系统，微软的Cosmos、Apache的Hadoop和Spark。这些通用的计算平台当初的设计目的是为了解决大量共有的海量数据分析过程中遇到的难题，因此设计目标比较宽泛，可以支持多种多样任务的计算。如今，这些有名的计算系统已经被广泛的使用在了工业界和学术界，为信息化产业的进步贡献了很大的力量。

然而通用计算平台的设计目标宽泛，主要是为了解决业界遇到的大部分海量计算任务，并没有明确的区分不同的计算任务之间的差别，导致这些通用的计算平台在某些特定的计算任务上表现的并不是很好，特别是在图计算这个领域，这个问题尤为突出。

1.2.3 动态图处理系统

动态图是具有事件属性的图，静态图相当于是动态图在某一具体时刻上的映射。相较于静态图来说，动态图更能显示现实世界中的种种联系，比如说互联网中的网页，每时每刻都有新的网页链接加入进来。人与人之间的社交关系，也是时时刻刻都产生着新的互动。因此想要更深入准确的显示现实世界的关系，可以及时处理动态图的图计算系统就显得尤为必要了，然而现有的研究往往还在静态图计算系统的研究上，主要是动态图计算难度大，涉及方面广，很难做出一个适用广泛的可用系统来。

近些年以来，时序数据库的出现就是一个重要的标志，人们已经越来越意识到单凭关系型数据库已经很难在这个数据爆炸的时代分析各种大数据任务了。比如自动驾驶环境中汽车遇到的各种场景的收集分析，自动化交易算法持续不变的收集着交易场所的数据，智能空调实时的调整房间的温度。从FaceBook推出了beringei以来，其他的工业界巨头也都纷纷的推出了自己的时序数据库产品，包括百度天工的时序数据库，阿里的HiTSDB等。然而在图数据库方面，仍然没有一个时序图数据库的出现。虽然各大研究机构也都纷纷进行了各自的研究，但是从目前的产品来看，效果并不那么显著。其中比较著名的便是微软亚洲研究院的KineoGraph，KineoGraph从存储到计算，提出了一个方便的动态图计算框架，遗憾的是至今并没有听说在工业界使用。其他比较著名的一些关于动态图的计算还有SpecGraph、InscGraph，不论这些产品是否成功，但是在图计算系统的研究上都贡献了宝贵的经验。

1.3 本文的研究内容

本文首先提出了使用现有的图数据库加上存储结构的优化和索引技术的加持，实现一个高效的、可用的可以存储动态图的存储系统。动态存储的基础是Neo4j，Neo4j作为一个原生的图数据库，在图查询方面有着得天独厚的优势，相对于传统的关系型数据库在深层次关系的查询方面更是有着几倍甚至十几倍的速度优势。针对动态图的时间属性，可以使用索引技术优化。这样就可以方便快速的得到整个时序图中的某个时间区段快照。

其次设计了既有主体存储与计算副本并存的存储系统。对于主体存储来说，最重要的是实现快速、精准的查询。对于快速查询的需求，主要是从图的底层存储结构，硬盘的加载以及索引技术加持，实现低延迟的查询。对于准确性的要求就需要数据库操作满足原子性、持久性、一致性和容错性等特点，也就是要支持事务，可以实现事务的提交与回滚。对于计算副本来说，最重要的就是要适合多次的迭代计算，可以保持数据与主体存储的一致。在这里主要采用Spark的RDD存储，每次迭代计算RDD都会检测和上次计算的差别，从而决定是否要进行下一轮计算。主题存储和计算副本之间采用Mq系统通信，既可以保证数据的一致性，也可以缓解网络通信的压力。

然后设计了可以用于动态图计算的增量式算法。增量式算法的重要特点是在动态图中每次变化部分的计算都可以借助上次计算的结果，实验表明，在数据的变化量在每次10%-20%的时候，增量式算法相较于非增量式算法在时间效率上大大提升，但是准确率却并没有下降多少。增量式算法的有效性主要体现在增量式算法相较于全量式算法的完整性和收敛性上面。也就是将新增部分对于原有部分上影响使用一个阈值来决定是否进行进一步传播，一旦新增的部分对原有部分的影响小于阈值立即停止传播，从而可以快速的达到收敛。

最后则是基于Neo4j，Spark，Docker，Spring Boot等技术实现了一个完整可用的图计算平台，这个计算平台主要包括存储模块，计算模块，数据导入模块，结果展示模块，状态监控模块等。

1.4 论文结构安排

本文主要分为六个部分，每个部分按照下列结构进行组织：

第一章，绪论。本章主要有研究背景，国内外研究现状，本文研究内容等部分。研究背景主要介绍了对于本文中研究内容

的选题来源以及研究的必要性。国内外研究现状主要描述了国内外的学术界和工业街在这个问题上的最新研究，主要包括图存储系统，图计算系统和动态图计算系统，同时表达了本课题的研究潜力。研究内容着重阐述了本文中的主要研究点以及最后达到的效果，最后的论文组织结构则是对全文的论文组织做了综合概述。

第二章，关键技术。关键结束方面主要介绍了本文中用到的一些前沿研究技术，以及这些技术现在的发展状况。主要包括图存储的的实现方式，分布式图存储和单机图存储，以及分布式图存储涉及到的图分割技术。图计算方面主要包括图计算的基础理论，分布式图计算和单机式图计算。

第三章，动态图存储的研究与实现。本文的主要研究点之一，动态图相对于静态图多了一个时间维度，因此实现起来的难度大大增加。图的动态存储最重要的是可以获取某个时刻整个网络的快照，在这里主要是利用了特殊的数据结构和划分时间区间的存储来实现。通过RabbitMQ系统实现主体存储和计算副本之间的相互通信，保证了主体存储和计算副本之间的一致性。

第四章，分布式图计算系统的研究与实现。本文的主要研究点之一，本文中的分布式图计算系统一个重要的特点就是可以支持增量式计算，也就是支持动态图的分析。本文中的实现方式是对现有的全量式算法进行改造，实现了增量式PageRank，增量式ShortestPath，增量式TrustRank和增量式TunkRank算法。并且在这一章的最后做了大量的实验，验证了增量式算法的有效性。

第五章，第五章是三四章的工程实现。第五章利用三四章的研究成果实现了一个完整的可用的图计算平台。这个图计算平台包括，数据导入模块，图存储模块，图计算模块和结果展示模块，并且包装了可以直接调用的接口。

第六章，总结与展望。这一章主要总结了前几章的工作成果，做了一个全面的总结。并对前几章中未解决的为题提出了思考，作为下一步待解决的问题。

指　标
疑似剽窃文字表述

1. GraphLab将数据抽象成Graph结构，并且将计算过程抽象成Gather、Apply、Setter三个阶段，
2. 1.4 论文结构安排
   本文主要分为六个部分，每个部分按照下列结构进行组织：
   第一章，

3. 1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第3部分

总字数：8051

相似文献列表　文字复制比：1.1%(85)　疑似剽窃观点：(0)

1	黄永建1 -《学术论文联合比对库》- 2012-05-03	1.1%（86） 是否引证：否
2	黄永建 -《学术论文联合比对库》- 2012-05-07	1.1%（86） 是否引证：否
3	黄永建2 -《学术论文联合比对库》- 2012-05-08	1.1%（86） 是否引证：否
4	黄永建 -《学术论文联合比对库》- 2012-11-02	0.5%（43） 是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第二章相关技术介绍

2.1 图存储系统的介绍与研究

本章的重点是介绍图存储的发展历程及最新成果。先是是对图存储的数据结构、算法以及工业界的实现做了全面具体的梳理。这些问题为本文中动态图存储系统的设计提供了基本理论和优化方法。后一部分是针对于近些年大数据技术的兴起，图计算在这一方面的发展，详细的阐述了图计算的应用场景。最后是总结了现有图计算和存储方面的成果和缺陷，表名本文研究的必要性。

2.1.1 图存储的实现方式

2.1.1.1 基于邻接矩阵的存储结构

图的邻接矩阵（Adjacency Matrix）的存储方式是一种比较浅显易懂的存储方式。主要是利用一个一维数组和一个二维数组，一维数组用来存储图中的顶点信息，二维数组（也就是邻接矩阵）用来存储边之间相互联系的信息。

图主要分为点和边两种主要的结构，点（Node）上主要存储节点的属性，边（Edge）则是建立两个点之间的联系。直接将两种结构合并在一起表示还是有些困难的，因此，人们在已有的数据结构上发掘出了简单的表示图的方式。一个基本的思路是将点和边分开矩阵。一维数组存储点及点的属性，点之间的关系很难由一维数组表示，那就考虑用二维数组来表示。这就是邻接矩阵表示法的诞生。

假设图G是一个有n个顶点的图，则邻接矩阵的表示就是一个n*n的方针，具体的定义如下：

两个节点有边连接，则arc[i][j] = 1，否则arc[i][j] = 0。邻接矩阵具有以下特点：

1. 每个图的邻接矩阵表示都是唯一的；

2. 无向图的邻接矩阵是一个对称矩阵。对称矩阵可以进行压缩，压缩的时候可以只存储上三角或者下三角，可以节省存储空间；

3. 对于一个无向图来说，邻接矩阵的第i行（或者是第i列）非零元素的个数正好是第i个顶点的度；

4. 对于一个有向图来说，邻接矩阵的第i行（或者是第i列）非零元素的个数正好是第i个顶点的出度（入度）；

图1-1 无向图的邻接矩阵存储

对于一个无向图，如图1-1所示，邻接矩阵的主对角线都为0，因为没有自身到自身的边。v1的度即为第二行所有非零元素的个数。并且该邻接矩阵关于主对角线对称。

图1-2 有向图的邻接矩阵存储

对于一个有向图来说，如图1-2所示，v1的出度为第二行所有非零元素的个数之和，v1的入度为第二列所有非零元素的个数之和。

邻接矩阵是一个简单易懂的存储结构，但是我们也可以很容易的看出，在有n个节点的图中，每个节点都要维护它与另外n-1的节点的关系，这在节点n的数量非常巨大，并且是稀疏图的时候会造成很大的空间浪费。针对这个缺点，又产生了适合存储稀疏矩阵的存储方法，就是下一节要介绍的邻接表。

2.1.1.2 基于邻接表的存储结构

当邻接矩阵存储边比较少的图时，对于存储空间来讲，会造成极大地浪费，因此邻接矩阵不适合存储稀疏图。因为图上每个顶点并不一定和所有的其他顶点都有关系，每个顶点只需维护和他周围节点的关系即可表达整个图。因此可以考虑将每个顶点周围的顶点存储在一个线性表中。整个图就组成了一个数组和链表组成的存储结构，这就是邻接表（Ad-jacency List）的实现方式。

图1-3 无向图的邻接表存储

在邻接表中顶点用一维数组或者是链表来表示，不过一般使用一位数组的情况较多，因为使用数组存储可以快速查找。另外每个节点会存储其指向的第一个邻接节点的指针，要想查找该顶点的邻接顶点或者是查找该顶点的边，需要从该节点开始遍历链表。

图1-4 有向图的邻接表存储

如图1-3所示，无向图的邻接存储主要有几个结构，首先是data域，用来存储节点及其节点数据。Firstedge用来存储指向第一个邻接节点的指针，adjvex是用来存储第一个邻接节点的地址，next用来存储下一个邻接节点的指针。

如图1-4所示，有向图的存储和无向图基本一致，只是无向图中每个链表只能存储单向的关系，如某个节点的出度。这样当想要计算每个节点的入度时就无从下手，因此可以建立有向图的逆邻接表，这样要想查找一个点的入度或者上游节点就可以直接查找逆邻接表。

2.1.1.3 基于十字链表的存储结构

对于有向图来说，邻接表的存储结构仅仅能表现出一个方向上的链路。比如建立了邻接表，要想获取一个节点的上游节点就要遍历全表，这造成了极大的不方便。

因此有人提出将邻接表和逆邻接表结合的实现方案，这就是十字链表（Orthogonal List）。十字链表使用类似于双线链表的结构，从一个节点开始，即可以找到它的上游节点也可以找到它的下游节点。

十字链表的顶点和边分别用两种存储机构，顶点的存储结构如图1-5所示，存储边的结构如图1-6所示。

图1-6 十字链表边的存储结构

图1-7 十字链表边的存储结构

十字链表将点和边都以一种特殊的结构存储，并且维护他们之间的指向关系。点的存储结构是data存储节点数据，firstin存储指向第一条入边的指针，firstout存储指向第一条出边的指针。tailvex存储这条边的起始节点下标，headvex存储这条边的结尾节点下标。Headlink存指向下一条入边的指针，taillink存指向下一条出边的指针。

如图1-7所示，要查找v0节点的上游节点即入边起始节点，从firstin的指针入手，首先根据1所示先查找第一条入边，再从这个入边的tailvex域可以得到这条入边的起始节点为v1，同样也可以得到第二个入边起始节点为v2。查找v0的下游节点即出边结尾节点，从firstout指针入手，首先可以找到第一条出边，再从这个出边的headvex可以得到该出边的结尾节点为v3。

十字链表结合了邻接表和逆邻接表的优势，可以方便的查询一个节点的上下游节点。而且又保证了查询速度与邻接表在同一个级别。在本文中我们的主体存储结构也正是采用的这种方式。

2.1.2 单机图存储

现在图研究领域在图存储方面主要分为两个派别，单机式图存储和分布式图存储。两个派别争论的焦点便是究竟图的存储适不适合分布式存储。首先介绍单机式图存储。

2.1.2.1 单机图存储的依据

在分布式技术成熟以前，长久以来存储技术一向是单机的。特别是图这种结构，由于其复杂的结构特征，它的存储一直是让人们比较头疼的问题。上一节介绍了图的存储结构，但是不论从哪一种结构上来看，分布式的实现代价都不小。而且要想分布式存储，图的最优分割点是一个NP难的问题，一旦分割的算法实现的不好，所造成的通信干扰非常之大，严重的影响到了图查询和图计算的效率。

因此支持单机式图存储的人们认为把一个复杂网络分割后存储并不是一个最好的选择，随着硬件技术的革新，完全可以通过增加硬件配置的方式完成一个大规模网络的存储，并且可以保持图的原生特性。

2.1.2.2 典型实现方式

Neo4j是这派技术的典型代表产品。Neo4j利用类似于十字链表的方式，将点和边分别采用定长字段存储，每个点存储第一条入边和第一条出边的索引，每条边存储它的首尾节点索引和上一条边与下一条边的索引。每当在Neo4j中进行一次关系查询时，都会使用深度优先或者广度优先的方式从查询顶点开始遍历链表，直到找到相应的顶点或者边为止。这样的查询相较于传统的关系型数据库来说查询速度能高到一个数量级。

2.1.3 分布式图存储

2.1.3.1 分布式图存储的必要

随着近些年来分布式技术的逐渐稳定，图分割任务在现有的计算能力下看似也不是一个困难的问题了。虽然单机式图存储可以通过堆积硬件来实现，但是近些年数据的扩展规模越来越快，硬件的革新速度根不是数据的爆发速度。因此图的分布式存储还是一个特别紧迫的寻求，特别是在一些海量计算任务上面，分布式存储可以方便的与分布式计算技术结合，将会使处理效率大大提升。

分布式存储的难点是图分割技术，图分割的方式选择将会影响到之后的查询和计算效率。图分割的方式会直接决定之后在其上运行系统的效率。

衡量一个图分割算法好坏主要在三个方面：

1. 通信代价：图计算的主要依据是图遍历，而图遍历会涉及到不同节点之间的通信，加入一个遍历操作的节点分布在多个机器上，不同机器之间的网络通信将会极大地影响计算效率。因为在本机内存上查找的时间可能在纳秒级别，而不同机器间的网络通信会在毫秒级别。

2. 负载均衡：主要是要平均的使用分布式系统上每台机器的计算能力，假如图的分布规模不均匀，那会造成部分机器过载和另一部分机器空载的情况，影响分布式系统计算能力的发挥。

3. 存储冗余：对于一个分布式系统来说，为了保证容错性往往会进行复制以保证高可用。对于分割不好的图也会造成大量的存储冗余，比如分割点在比较重要的节点处。

2.1.3.2 Hash分割

Hash分割是最简单的一种分割，这种分割不会去考虑通信代价、负载均衡、存储冗余等问题。比较适合图比较小的时候的分割，因为对图分割也是需要时间消耗的。研究表明在图的节点规模在10亿以下时，采用Hash分割的方式完全可以满足大部分的需求，当图的节点规模在10亿以上时，采用复杂的算法分割效果更优。

2.1.3.3 启发式算法分割

图的分割问题是经典的NP完全问题，因此大多数的图分割算法都是基于启发式算法来实现的。主要分两种实现方式：一种是启发式的交换点对，另一种是多层次的划分。

启发式方法的代表是KL和FM方法。对于图分割的问题，早在上世纪的八十年代便有了深入的研究，主要是科宁汉（Kernighan）和林（Lin）提出的一种非常有效的启发式算法叫做KL算法。KL算法的主要思想是在每轮的划分中都把一个图随机的进行2等分，然后交换两部分的节点，分别计算交换了点之后得到的收益，再把收益率最高的节点进行再次对换。该算法的最坏时间复杂度为O(n^2)，对于复杂的图分割问题来讲这个时间复杂度是可以忍受的。但是这种方法也有其缺点，他最多只能处理10的4次方以内的节点数量，再高规模的节点数量效果就有些不好了。

为了处理大规模的图切分任务，库玛（Kumar）等人之后提出了图层图划分算法那METIS。METIS算法的核心是先将大图进行一定方法的“粗糙化”，再对粗糙化之后的每个小图执行KL算法，这样的改进成功的克服了KL算法无法支持大规模数据划分的缺点。百万级的图分割问题基本上可以在秒级时间代价内完成。

2.2 图计算系统的介绍与研究

2.2.1 图计算基础理论

随着互联网迅速的发展，大规模图分析的需求也越来越紧迫。因此近些年涌现出来了一批图计算系统，这其中具有标志性意义的有Pregel，GraphLab，VENUS和GraphChi等。他们实现了类似于Hadoop和Spark这样的框架，通过简单的更新函数就可以完成大规模图计算。现有的图计算的基本思想是使用“Think like a vertex”去表达图计算的过程，使得庞大的计算任务可以转换成顶点任务就可以完成。

顶点任务程序只用计算更新图中顶点和边的更新状态，然后传播到周围的节点。图计算中有两个重要的计算机制，一个是BSP整体同步计算模型，另一个是GraphLab的异步并行计算模型。

具体的来说就是，图计算系统会预先定义好一个顶点更新函数update()，用户要想使用该系统做自己的计算任务，只需要重载该函数即可。update函数可以使用到节点的信息，也可以使用到邻接边的信息。每次迭代过程就是每个节点都执行自己的update()函数。

2.2.1.1 BSP算法

BSP（Bulk Synchronous Parallel，整体同步计算模型）是一种新兴的分布式计算模型，是由哈佛大学的Viliant和牛津大学的Bill Coll提出的并行计算模型，最开始被称为“大同步模型”，又因为他们提出这个模型的初衷是改变现有的计算机体系结构，同来作为计算机体系结构和计算机语言之间的沟通桥梁，因此它还有个别名“桥模型”。

BSP由一系列的全局超步计算过程组成（每个全局超步就是一次迭代过程），每个超步主要进行三个任务：

1. 局部计算：每个processer都只计算自身的计算任务，他们只读取存储在本地内存中的值，不同的计算任务之间是相互独立的并且异步执行。

2. 交换信息：每个processor计算完自身的任务之后，会将消息传递到与他相关联的processor。

3. 整体同步：所有的processor都处理完毕并且消息交换完成以后，会进行下一个超步，也就是下一轮迭代。

图2-1 BSP计算模型

2.2.1.2 Pregel算法

Pregel最早是由Google提出的大规模图计算模型，现在已被大量的新兴图计算框架所采纳。Pregel是基于BSP实现的并行图处理系统，可以运行在多台廉价计算机组成的集群上面。通常一个图计算任务会被分解到多台计算机上同时执行，任务执行的过程中，临时数据一般会存储在本地磁盘，而持久化数据会存储在分布式文件系统。

图2-2 Pregel体系结构

Pregel的体系结构如图2-3所示：

集群中多台机器一起执行计算任务，其中一台机器作为Master，其他的机器作为Worker。

Master负责着管理其他的Worker，与Worker进行通信。每当一个计算任务来临Master将图分为多个分区，并且将分区分配给每个Worker，监控Worker的运行。

Worker会向Master注册自己的信息。每个Worker管辖自己分区的内存状态，计算状态被保存在自身内存中。每个超步中每个Worker都会遍历自己内存中的节点，并且计算每个节点的update函数。

Master使用定期发送消息的方式进行容错。Master定期向Worker发送消息，一旦消息在一定的时间内没有的到响应，Master会认为这个计算节点出现了错误，然后会启动恢复模式。

Pregel顶点间的信息传递采用纯消息传递的模式，在使用异步和批量消息传输的方式下，可以极大地提升整个系统的性能。

图2-2 消息传递模式 2-3 状态机图

图2-2是Pregel的消息传递模式图，该函数描述了一个顶点在一个超步S需要进行的操作。该函数读取前一个超步（S-1）发送过来的消息，在这个节点上做聚合，聚合以后修改本身的信息，然后沿着出边发送自身的信息。这些在超步S发送的信息会在（S+1）的超步被别的节点接收。

图2-3是Pregel执行的状态机图，每个收到消息的节点都会被设置为活跃，活跃的节点可以参与本次计算并且发送消息，非活跃的节点如果收到消息就会变为活跃。迭代算法执行到图中所有的节点都变为非活跃，就可以结束。

2.2.2 单机图计算系统

单机图计算系统的代表是华为的VENUS和伯克利大学的GraphChi。VENUS提出了以顶点为中心的流水化图计算模型，这样的做法显著地降低了基于磁盘的图IO，极大地改善了磁盘IO与计算速度不匹配的情况。VENUS使用了独有的基于外存算法的存储扩展方式，叫做分级存储的基于流水线的图计算模型，可以突破单机图计算的可扩展能力，达到了不错的计算效果。

VENUS系统使用了数据分片的方法以及独有的外存模型。在该系统中，每个数据分片被分为v-shard和g-shard两个部分。V-shard用来存储图中的点部分，数量不大，并且需要频繁的修改。G-shard用来存储图中的边数据部分，数据量巨大，但是几乎不发生改变。在图计算的执行过程中，首先磁盘按照顺序读取g-shard分片的数据，每当遇到g-shard所关联到的点的时候，就将v-shard中的点也加载到内存。处理g-shard中边对应的更新函数，此时g-shard关联的点已经在内存中了，可以直接读取计算。一旦计算完成，立马丢弃g-shard中加载过来的边数据，这样可以保证系统在加载数据的同时进行计算，极大地减少了运行的时间。

另一个基于单机的图计算系统是GraphChi，在GraphChi中一个图也会被分割成多个数据片，保证每个数据片都可以放在内存里面。每个数据片内存存储的都是一个子图，包括一系列的顶点和边。图计算过程由多个迭代过程组成，每个迭代过程都会对图中的节点进行更新计算。在GraphChi中，每次迭代系统都会一次的处理一个数据分片，并且在每个数据分片上执行更新函数。这部分的处理主要分为单个具体的步骤。首先是载入，将一个数据分片载入到内存中。然后是更新，在每个数据分片中的每个节点上执行更新操作。最后是写回，将计算好的信息重新写会到磁盘上。这种处理方法的缺点主要有二，一个是在数据加载的过程中将会产生很大的IO，而这部分IO的过程中无法同时进行计算。第二个是每个分片计算完后会传播本身的信息给相邻分片的节点，这样将会产生大量的随机IO。

2.2.3 分布式图计算系统

2.2.3.1 GraphLab

GraphLab是一个基于内存的分布式图计算系统，在GraphLab中一个图被分割成多个子图，每个子图被存储在一台单独计算机的内存上，图分割采用的是顶点切分的方式。图分割之后，被切分的顶点会在每个与他有关联的自图中都存在一个备份。这样一个顶点就可能会存储在多个主机上，每次一个子图上发生计算，更新过信息的节点就会将计算结果同步到其他有该节点备份的主机上。图上的数据包括图的顶点和边，以及顶点和边上的值。这样做的好处就是不用网络传输图结构信息了，但是同步节点信息仍然会需要大量的网络通信。

2.2.3.2 Spark GraphX

Spark GraphX是基于Spark平台的一个分布式图计算系统，随着Spark的开源影响力的进一步增加，Spark GraphX也逐渐的被更多的人熟知。

GraphX将图数据以RDD的模式分布式的存储在集群节点上，使用顶点RDD（VertexRDD）和边RDD（EdgeRDD）分别存储顶点的集合和边的集合。顶点RDD一般是按照Hash的方式分区，各个顶点以Hash的方式分布在不同的节点上。边RDD按照特定的分区策略分区（一般是使用边Id的Hash值分区），分布在不同的节点上。此外，顶点RDD还存有顶点指向边分区的路由信息，也叫做路由表。路由表一般和顶点RDD一起存储，他记录的是顶点RDD和边分区的映射关系。当进行计算的时候，顶点RDD会根据路由表信息将顶点信息数据分发到边RDD分区。具体如图2-4所示。

图2-4 SparkGraphX的存储结构

在图计算的过程中，有些边的计算需要需要点数据的支持。比如在计算PageRank的时候会需要计算出边的权值，这是顶点RDD分区要做的就是把顶点数据传送到出边所在的RDD分区。GraphX会根据路由表，生成顶点RDD与边RDD对应的重复顶点视图，因为在进行以顶点划分的分区的时候，同样的顶点会处在不同的边分区中。生成的的这个重复顶点视图与边分区的RDD个数相同，当要进行计算的时候，顶点会与边一起形成三元组。因为顶点的数据相对于边的数据非常少，每次生成重复顶点视图的代价并不大。并且随着迭代次数的增加，每次需要更新的节点数量也会越来越少，这样顶点RDD的更新代价也会越来越小，从而SparkGraphX可以快速的执行。

2.3 本章小结

本章主要介绍了本课题中相关的关键技术。由于本课题中要研究实现一个图计算系统，涉及到的技术主要分为图的分布式存储和分布式计算。第一节介绍了现有的图存储方式以及他们之间的优劣关系，为下一章文中提出的存储结构提供依据。第二章主要介绍了现有的图计算技术，以及他们各自适用的场景，分析了他们的适用情况以及存在的不足，为本文的图计算系统做出了铺垫。

4. 1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第4部分

总字数：10952

相似文献列表　文字复制比：5.2%(570)　疑似剽窃观点：(0)

1	基于Hadoop平台的广告检测系统研究与实现 杨宁(导师：李伟) -《复旦大学硕士论文》- 2012-09-27	2.3%（257） 是否引证：否
2	大数据：Hbase原理、基本概念、基本架构_CDerek -《网络（http://blog.sina.com）》- 2015	1.8%（202） 是否引证：否
3	Hbase原理、基本概念、基本架构 - 坦GA的博客 - 博客频道 - CSDN.NET -《网络（http://blog.csdn.net）》- 2017	1.8%（202） 是否引证：否
4	大数据(五) - HBase - IT十年 - 博客频道 - CSDN.NET -《网络（http://blog.csdn.net）》- 2017	1.4%（157） 是否引证：否
5	眼科专科影像云服务平台研发 张旭东(导师：宋文爱;王青) -《中北大学硕士论文》- 2017-04-11	0.8%（93） 是否引证：否
6	一种基于Sqoop的数据交换系统 于金良;朱志祥;梁小江; -《物联网技术》- 2016-03-20	0.8%（86） 是否引证：否
7	基于图计算模型的矩阵分解并行化研究 戴世超(导师：包晓安) -《浙江理工大学硕士论文》- 2016-04-14	0.8%（83） 是否引证：否
8	一种基于NoSQL数据库的医疗监护大数据存储设计 黎茂林;张鹤;吕德奎; -《电脑与信息技术》- 2016-12-15	0.5%（53） 是否引证：否
9	基于有向图的虚开增值税发票行为检测方法研究 刘丽萍(导师：罗晓霞;董富强) -《西安科技大学硕士论文》- 2017-06-01	0.4%（42） 是否引证：否
10	大数据分析技术在风电设备异常预测中的应用 张慧亭;王坚;凌卫青; -《电脑知识与技术》- 2017-03-23 1	0.3%（36） 是否引证：否
11	我国经济政策审计评价研究 王延军(导师：张通) -《财政部财政科学研究所博士论文》- 2015-05-22	0.3%（36） 是否引证：否
12	基于云平台的全域匿名算法的研究与实现 胡庆庆(导师：刘君强) -《浙江工商大学硕士论文》- 2015-01-01	0.3%（36） 是否引证：否
13	组学大数据的检索系统设计与实现 徐康(导师：臧天仪) -《哈尔滨工业大学硕士论文》- 2015-06-01	0.3%（34） 是否引证：否

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第三章动态图存储系统的研究与实现

3.1 适用于OLTP的主体存储系统

在图处理领域一个重要的目标是实现图关系的快速查询，现实中的主要场景有在社交网络中查询某个人的朋友，查询某两个人的共同朋友，推荐可能认识的朋友，查询任意两个人怎么样才能快速认识。在科研论文方面查询引用的论文，有共同作者的论文，有共同关键词的论文。互联网网页中一个页面链出的页面，链入的页面等。这些需求的共同点是查询的准确性和快速

性，这就要求我们的存储系统可以快速的响应查询，同时还要有一定的安全性。

3.1.1 图的存储结构设计

第二章提到图的存储方式主要有邻接矩阵、邻接表和十字链表等。邻接矩阵存储稀疏图会造成巨大的浪费，单一邻接表无法做到双向遍历，不能满足一些复杂查询，十字链表可以双向遍历但是在范围查询方面表现的并不是很好。在这里我们采用另一种灵活的邻接表存储方式。同时为了快速查询，采用BigTable模型作为存储后端。

3.1.1.1 内存中的存储结构设计

如图3-1所示，一个图可以抽象为节点（vertex）和边（edge）的集合。每个点都有自己的标签（label）和属性（property），每个边也具有自己的标签（label）和属性（property）。

图3-1 BSP计算模型

一个大图由一系列的节点和边组成，具体组成如下所示。

1. 点的集合

每个节点都有一个唯一的标识。

每个节点都有一个出边的集合。

每个节点都有一个入边的集合。

每个节点一个由键值对组成的属性的集合。

2. 边的集合

每条边都有一个唯一的标识。

每条边都有一个尾部节点id。

每条边都有一个头部节点id。

每条边都有一个由键值对组成的属性的集合。

根据以上描述我们设计出几个重要的类，如类图3-2所示，主要包括GraphFactory类，用来连接存储后端。Graph类，图上一系列操作的集合。Edge类，边属性和行为的集合。Vertex类，节点属性和行为的集合。Property，节点和边的属性。GraphIndex类，索引行为的集合。

图3-2 BSP计算模型

GraphFactory用来建立与后端存储的连接。主要是根据键值对的配置来确定需要实例化的图。GraphFactory有build，open，close等主要方法。build方法用于使用配置操作实例化一个图。open方法用于使用传入配置实例的方法实例化一个图。close方法用于销毁这个实例化的图。

Graph类是图上一系列操作的集合。主要包括traversal，addVertex，addEdge，removeVertex，removeEdge等方法。teaversal是获取edge和vertex的实例，用于执行图上的遍历。addVertex用于增加节点，addEdge用于增加边，removeVertex用于删除节点，removeEdge用于删除边。

Edge类是图上边以及边行为的集合。主要包括edgeLabel，inVertex，outVertex，otherVertex，property，addPropertey，removeProperty等方法。edgeLabel用来获取这条边上的标签类型，inVertex用来获取这条边起始的节点，outVertex用来获取这条边尾部的节点，otherVertex用来获取边上另一侧的节点，property用来获取边上的属性。

Vertex类是图上的节点以及节点行为的集合。主要包括vertexLabel，property，addProperty，removeProperty，addEdge，removeEdge，edges，inEdges，outEdges等方法。vertexLabel用来获取节点的标签类型，addEdge用来给两个节点之间增加一条边，removeEdge用来删除两个节点之间的边，edges用来获取一个节点周围所有的边，inedges用来获取一个节点的入边集合，outEdges返回节点的出边集合。

Property类是节点和边属性及行为的集合。主要有key，propertykey，element等方法。

3.1.1.2 后端存储结构设计

为了满足快速检索以及分布式存储的特性，经过多次实验对比，我们采用BigTable作为底层存储模型。在BagTable的数据模型下，每个表都是行的集合，每一行都由一个唯一主键标识。每行由任意有限大小的单元（cell）组成，每个单元由一个列（column）和值（value）组成。在BagTable存储模型下，每行都支持大量的单元格，并且这些单元格不用像在关系数据库中那样需要预先定义。

图3-2 BSP计算模型

如图3-1所示，我们将每个节点的邻接表作为一行存储在Hbase中。顶点id是该顶点邻接表的行关键字，每条边（包括出边和入边）和属性都作为一个独立的单元存储在行中，这样可以高效的删除和插入。特别是在使用vertex id作为关键字之后，我们可以将该表按照id进行排序，从而在图切分的时候，将经常互相访问的节点存储在同一个分区中，可以极大地加快访问速度。

每一行的数据可以按照某个type，或者是按照我们自定义的sort key排序，可以加速我们查询某个节点的邻接关系时的速度。

图3-2 BSP计算模型

边（Edge）和属性（Property）的存储结构如图3-2所示。每个顶点的邻接边和属性都作为一个单独的单元存储在以这个顶

点id为键的行中。每个单元都被划分为两部分，column和value。在Edge的存储单元中，column是由label id&direction，sort key，adjacent vertex id，edge id等四部分组成。label id&direction表示边类型以及方向，sort key表示排序字段，adjacent vertex id表示邻接节点的id，edge id代表边本身的id。

由于每个边和属性都作为一个单独单元存储在对应顶点的行中。序列化的时候保证Edge和Property按照一定的顺序排序，并且使用可变ID编码方案和压缩对象序列化来保证每个边或属性占用的存储单元尽可能小。如图中浅色框部分表示支持可变长编码方案的存储区域，这部分经过一定的编码处理可以有效的减小他们消耗的字节数。深色部分框部分表示使用关联属性表中压缩数据元序列化的属性值（即对象）。

3.1.2 时序图数据的存储

为了满足实时数据的分析，近些年来涌现了很多时序数据库。比较有名的有InfluxDB，RRDtool，Graphite，openTSDB等。InfluxDB和RRDtool都是老牌的单机时序数据库，Graphite是基于whisper开发的时序数据库，较RRDtool做了些优化。openTSDB是一个分布式，可伸缩的时序数据库，支持较高的吐出量，存储峰值可达每秒百万级别，并且支持时间精度到毫秒级别的数据存储。但是针对图数据的存储，至今仍然没有一个统一的方案。主要是图数据的存储和传统时序数据库的存储目标有较大差别。传统的时序数据库只关注一段时间内的数据，分析主要也是针对阶段时间数据做分析，比如传感器获取的实时数据，摄像头采集的实时数据等。但是在图的模型中，需要全量的数据才能反映整个网络的状态，大图中某个区域的节点和边的变化影响的是整个图的属性。因此我们可以采用图快照的方式反映大规模网络演化过程中的某个时刻的静态图。

首先按照传统的图表示方式，不含时序信息的静态图记为有序对：G=(V,E)。我们记这样的图为静态图，其中V表示顶点（Vertices或Nodes）的集合，E表示边（Edges或Links）的集合。这里的E中的一个元素e是V中的两个元素u，v组成的二元组e=(u, v)。在有向图中我们称这里的u为源点，v为目标顶点。

对于有时序信息的动态图，我们可以相应的记为：

Gt=(Vt,E(t))

这里的t代表的是时序信息，V(t)和E(t)则分别代表了具有时序信息的顶点和边。动态图的时序信息表示的是该结构或者该属性值的生命周期，可以使用非重合时间区间的集合来表示。一个完整周期的动态图我们用G={Gt1, Gt2…Gtn}表示，任意时刻的静态图即为Gtn。

对于时序图数据的存储来说最重要的是我们要获取动态图的某一时刻快照，为此我们想的是给图中的节点和边数据都加上时间(timestamp)属性，用来代表数据的版本。在这里我们采用Hbase作为存储后端。HBase是一个构建在HDFS上的分布式存储系统，是基于Google Bigtable模型开发的典型的key value存储系统。HBase设计的初衷就是实现HDFS上的海量数据的存储，因此设计的理念也不同于一般的数据库，他是一个很适合存储非结构化海量数据存储的数据库，主要特点是它是基于列的存储，而不是基于行的存储，这样的存储结构可以方便读取海量数据的内容。HBase的基本存储结构就是类似于哈希表的概念，但又不仅仅是简单的一个key对应一个value。每个key对应的可能是多个属性的数据结构，但是又没有传统数据库中那么多的关联关系，也可以叫做是松散数据。

简单的说，在HBase中存储的数据可以看做是一张巨大的表，这个表的属性是可以根据我们的需求动态增加的，在HBase中没有表与表之间的关联查询。在存储数据的时候我们只要制定了要存储数据的column和family即可，不用关心数据的类型，在HBase中所有的数据都默认为字符的存储。HBase存在着如下的特点：

HBase可以存储的数据规模非常大，一个表可以由数十亿行和上百万列；

每行的数据都有一个可排序的主键和任意多的列，并且列的数量是可以根据需求动态增加的，同一张表中的不同数据可以有完全不同的列。

面向列的存储和控制，面向列的独立检索；

空列并不会占用存储单元，这样可以让我们的表设计的非常稀疏

每个单元中可以存储多版本的数据，在默认情况下版本号是自动分配的，是插入的时间戳，并且可以由用户定义。

数据类型单一，HBase中存储的数据都是字符串的形式，并不区分他们的具体类型。

因为Hbase的每个单元内的存储支持多版本存储，每个版本的版本号用时间戳表示，经过一定的修改完全可以快速的查询到某个时间段的静态图。Hbase中一个数据多版本的表示方法如下：

COLUMNCELL v:c1timestamp=1499088390024,value=value7 v:c1timestamp=1499088387559,value=value6 v:c1timestamp=1499088385347,value=value5 v:c1timestamp=1499088383228,value=value4 v:c1timestamp=1499088380943,value=value3

在Hbase中我们可以设计存储多个版本的数据，版本个数和版本号表示方式都可以自己定义，如上图所示，c1的值有5个版本，每个版本都对应了唯一的timestamp。查询的时候可以具体到某个版本的数据，对应到我们的图存储中，我们将每个时间区间的静态图存储为同一个版本，获取某个时间区间的静态图用版本号查询就可以做到。

首先，我们想一下图3-1的数据用关系型数据库该怎么存储，图中有person和tweets两种节点，并且有post和follow两种关系。Person和tweets节点可以笨别转化成一个person表和一个tweets表，post关系可以将tweets表的外键指向person的主键id。Follow关系可以用另一个follow表表示。

tweets

id topic person

1 study 1
2 sports 1
3 food 1
4 book 1
5 movie 1
person
id name age
1 peter 35
2 vadas 27
3 john 27
4 josh 35
follow
id person1 person2
1 1 2
2 3 1
3 4 1

将这个数据转化到BigTable的存储模型上，可以将以上三张表转化为一张表，我们将该数据转化成图3-2和图3-3描述的存储模型。整个表以vertex为row key，并且排序，然后将经常访问的vertex存储到同一个regin中。Famly有edge和property两种。存储单元有edge和property两种。把上述数据转化为BigTable存储模型的Schema为下表所示。

row key family Attributes
vertexid edge {columns:value} 3versions
property {columns:value} 3versions

我们以图3-1中person节点中的josh作为例子，写出该数据的存储模型。

rowkey Edge:edge1 Property:property1
direction sortkey adjacentvertexid Edgeid Key id
3 out@time1out@time2 001@time1001@time2 003@time1003@time2 001@time1001@time2 {name:josh,age:35}@time1{name:josh,age:35}@time2

在查询的时候我们需要指定数据的Version Number。Version可以由用户自己定义，有两种定义version的方式，第一种是保存数据的n个版本，二是保存近一段时间内的版本。对于我们的图模型来说，每增加或者修改节点或者边，相应的修改节点和边的版本号为最新版本。每删除一个节点或者边就不更新版本号，当我们查询某个版本节点或者边不存在的时候有两种情况，一种情况是该节点或者边在当前版本被删除了，另一种情况是该节点或者边从来就不存在过，我们要注意这两种情况的分别。

假如在time3我们删除了josh到perter的follow关系。此时josh的存储结构更新为下表所示，property随着版本更新到了time3，但是edge并没有相应的更新，表示在time3时刻，jsoh指向perter的边被删除。

rowkey Edge:edge1 Property:property1
direction sortkey adjacentvertexid Edgeid Key id
3 out@time1out@time2 001@time1001@time2 003@time1003@time2 001@time1001@time2 {name:josh,age:35}@time1{name:josh,age:35}@time2{name:josh,age:35}@time3

查询任以时刻的节点信息使用如下语句：

>> get 'twitterGraph','3', {Column => ['edge:edge1:direction'], version = time2}>>
COLUMNCELLedge:edge1:directionversion=time2, value=out

3.1.3 计算快照的生成

我们在进行图计算的时候，如果直接在后端存储上进行计算会造成多种问题。比如如果一个计算的计算过程持续时间过长，并且可能会多次访问图的存储结构，如果正在参与计算的节点或者边此时发生了更新，就会出现不一致的状态。为了解决这个问题，最简单的方式就是在正在进行计算的数据上加锁，直到计算任务完成才释放锁。但是这样又造成了新的问题，系统的吞吐量被严重限制，所以计算与更新共用同一个存储结构并不是一个好的策略。

因此我们可以考虑这样的操作，将某个时间点的快照拿出来用于分析计算，剩下的部分仍然可以继续进行图更新操作。这样图更新和图计算的所操作的就是不同的版本了，因此也不会造成冲突。而且图更新和图计算负责的主要任务不同，图更新主要执行可以改变图结构的操作，例如增加边，增加节点。而图计算则可以更新图的应用信息。这样的高效隔离使系统有了更高的鲁棒性。

基于以上两点，下一步就可以分析图快照的生成了。生成动态图某个节点的快照我们采用时段提交和累加区间提交两种方式。两种方式分别对应不同性质的任务。两种提交方式都要依赖Transcation（事务）日志和Trigger（触发器）技术。

时段提交需要设定好时间区间，设一个记录全局的上一次生成快照的时间Tlasttime，设一个阶段提交时间区间Tperiod，每

次提交事务的时候都获取当前时间，并且做检查Tdiff = Tpresent – Tlasttime。如果Tdiff>Tperiod则生成快照Si。

Begin Transaction{Tperiod = Date.timestampTdiff = Tpresent – TlasttimeIf(Tdiff >= Tperiod){GenerateSubgraph(vertex.timestamp > Tdiff && vertex.timestamp <=Tpresent)Tlasttime= Tpresent}}End Transcation

累加区间提交则是以数据规模来决定计算频度的，适用于实时性稍低的一些任务，节省计算资源。累加区间提交将会设一个记录全局事务id的TransCount，每次更改都会使TransCount加1，并且设置当前事务操作的TransCount的值作为节点和边的属性，同时还会有一个TransCountlasttime记录上一次生成数据快照的事务id，每当执行设定值N个事务之后，生成一次数据快照。

Begin Transaction{TransCount = TransCount + 1TransCountdiff = TransCount – TransCountlasttimeIf(TransCountdiff >= n){GenerateSubgraph(vertex.transId > TransCountlasttime && vertex.transId <= TransCount)TransCountlasttime=TransCount }}

通过这两种快照生成方式，可以保证图的更新和图的计算在两套系统中执行，大大减小了阻塞发生的概率。生成快照的本质就是获取动态图的某个时间段内的静态图。根据3.1.2节，我们可以发现某个时间区间内的静态图获取可以转化为获取HBase上某个版本数据的集合。因此计算快照的获取可以转化为对Hbase上存储的数据进行按版本查询。然而根据我们的了解可以知道，在HBase上的查询只有按照某一固定的rowkey查询和按照rowkey的范围查询和全表扫描三种方式。根据我们上一节的设计方式来看，要获取某个版本的数据集合，只能进行全表扫描，这在数据版本非常多的时候效率十分低下。

然而幸运的是HBase也为我们提供了一种二级索引的机制，因此我们可以在rowkey与版本的对应关系上建立二级索引。二级索引建立的本质是建立各列值与rowkey之间的关系，在这里我们使用version-rowkey来建立二级索引。建立好二级索引的数据存储形式如下表所示。

Rowkey ColumnFamily:Index ColumnFamily:Edge ColumnFamily:Property

version1-rowkey1 /

version1-rowkey2 /

version1-rowkey3 /

version2-rowkey1 /

rowkey1edge1 property1

rowkey2edge1 property1

rowkey3edge1 property1

我们会生成一个新的列簇叫做Index，Index并不用来存储数据，仅仅是为了将索引数据和主数据区分开来（因为在Hbase中同一列的数据会放在一起压缩存储）。我们设计了version-rowkey之间的对应关系，是为了快速地找到version到rowkey的映射，将rowkey经过排序以后，同一个version开头的rowkey会聚集到一起，这将对我们根据version选择图的快照提供很大的帮助。

3.2 适用于OLAP的计算副本存储

生成的存储快照会可以转储到Spark的分布式内存RDD中，作为我们的计算副本。RDD是Spark独有的一个容错的，并行的数据结构，并且还支持用户显式的将数据存储到磁盘和内存中，能够灵活的控制数据分区的数量。是Spark进行计算的主要数据结构。

3.2.1 顶点分区和边分区的存储结构

关于图分割，主要有两个常用的分割方法。一个是点分割，另一个是边分割。Spark采用的是点分割的方式，并且采用不同的RDD（VertexRDD和EdgeRDD）存储顶点和边的集合。顶点RDD默认情况下会使用顶点的ID进行Hash分区，将顶点数据以多分区的形式分布在集群上。边RDD会按照指定的分区策略进行分区（默认情况下采用的是边的id进行Hash分区，在数据量非常大的情况下可以采用Metis算法进行分区，可以有效地降低通信成本），将边数据以不同的分区形式分布在集群上面。此外，为了顶点RDD与边RDD的快速通信，顶点RDD还存储了顶点到边RDD的路由表。路由表是顶点RDD中的一个特殊数据结构，它记录了顶点RDD中所有的顶点和边RDD的对应该关系。在计算过程中边RDD需要顶点的数据的时候，顶点RDD会根据路由表将顶点数据发送到边的RDD分区。顶点RDD和边RDD的分布如下图所示。

在大部分的图计算过程中，边的计算都需要两端顶点的数据，即形成三元组的视图，例如进行PageRank算法的时候需要生成出边的权值，这就需要顶点将自身的权值发送给它的出边所在的RDD分区。Spark会根据路由表在顶点RDD中生成与边RDD对应的重复顶点视图，根据我们之前的介绍，进行点切分的顶点会分布在不同的分区中。重复顶点视图的作用是作为中间RDD，他会将顶点数据传送到相关的边RDD分区中。重复顶点视图的分区数量和边RDD相同，如图所示，重复顶点视图A中存在着边RDD分区A中的所有顶点。在进行计算的过程中，Spark会将重复顶点视图和边RDD进行merge操作，即将重复顶点视图和边RDD一一对应起来，从而将边数据和顶点数据组合起来，形成三元组。在形成三元组的过程中，只有根据顶点RDD形成的重复顶点视图需要在不同的边分区之间移动，merge操作不需要移动顶点数据和边数据。而且在大部分图中顶点数据都是远远的小于边数据的，随着迭代次数的增加，需要进行更新的边数据也越来越少，这样可以大大的减少数据的移动量，从而加快整个计算过程。

Spark在顶点和边RDD的存储中采用数组的方式存储顶点数据和边数据，这样做可以减少访问性能的下降。Spark还在存储

图数据的时候建立了众多的索引结构，这些索引结构可以辅助快速的访问顶点数据或者是边数据。

3.3 存储主体与计算副本的同步

存储主体和计算副本分别执行了图结构的更新任务和图属性的计算任务。他们之间通过消息队列互相传递数据。从整体上说，这个过程Hbase生成数据快照传递给Spark，Spark计算完以后将计算的结果返回到Hbase中。

存储主题和计算副本之间的同步设计图如上图所示。主要分为两个部分，第一个部分是图的导入与导出，第二个主要部分是计算请求的交互。整体交互流程如下所示：

平台发起一个计算任务，任务描述信息发送到消息队列中

HBase将相应的数据发送到HDFS上

Spark从消息队列中读取计算任务

Spark根据计算任务从HDFS中读取相应的数据，并且执行计算

Spark计算完成，返回相应的数据到HDFS中

Spark告知本次计算任务完成，并且继续读取下一个计算任务

Hbase从HDFS上读取计算完的结果并且更新自身的存储

通过以上的交互过程，有效的同步了发送请求与计算速度之间的差异，采用消息队列的方式解决了发送请求频率与计算速度不匹配的问题，极大地提升了系统的容错性。

3.4 存储体系完备性讨论

本文采用存储主体和计算副本并存的方式，有效地解决了图更新和图计算并发执行的问题，提升了整个系统的吞吐量。存储主体支持事务的特点使得其可以方便的进行图查询和图更新操作，计算副本采用Spark RDD存储，极大地加快了图计算的执行速度。在主体存储和计算副本之间设置了缓冲地带，有效的解决了发起计算请求的速度和平台计算速度的不匹配，大图导入导出的速度和平台计算速度的不匹配等，增加了整个平台的可靠性。

该存储系统支持OLTP任务的执行，可以支持图中节点的增加，删除，修改以及日常查询任务。比如在社交网络中返回一个人的朋友，朋友的朋友，所发的微博，他的关注人都有哪些，哪些人都关注了他，参与了怎样的话题等。并且可以按照时间区段查询，真实地反映一个人的社交生活。同样该系统还支持OLAP任务的执行，在社交网络中我们要实时的获取网络中节点的重要性，社团的演化，两个人之间的最短认识路径，这种需要计算才能返回值的操作可以加载到计算副本中，执行完计算任务之后再返回到主体存储中。

3.5 本章小结

本章从图存储的角度设计了适合动态图计算的复杂图存储系统。首先是主题存储系统的设计，因为我们要存储的是动态时序图数据，现有的数据库都无法满足我们的需求。我们跟根据分析图查询的便利性，和时序数据存储的可行性采用了BigTable数据模型作为底层存储结构，同时为了图操作的便利性，我们设计实现了一个中间存储层，用来衔接存储后端和上层应用。然后为了查询的方便我们还在存储结构上进行了进一步的索引优化，从而方便了一个快照图的快速生成。为了提升整个系统的吞吐量，我们采用了存储主题和计算副本并存的存储方案，计算副本采用Spark 的RDD存储，并且使用Spark进行计算，不但解决了存储主题和计算副本相互冲突的问题，同时大大的提升了整个系统的计算能力。整个系统无论在功能的完备性方面还是架构的稳定性方面都有着不错的表现。

指　标
疑似剽窃文字表述

1. Hbase作为存储后端。HBase是一个构建在HDFS上的分布式存储系统，是基于Google Bigtable模型开发的典型的key value存储系统。HBas
2. 不同于一般的数据库，他是一个很适合存储非结构化海量数据存储的数据库，主要特点是它是基于列的存储，而不是基于行的存储，
3. 不仅仅是简单的一个key对应一个value。每个key对应的可能是多个属性的数据结构，但是又没有传统数据库中那么多的关联关系，也可以叫做是松散数据。
   简单的说，在HBase中存储的数据可以看做是一张巨大的表，这个表的属性是可以根据我们的需求动态增加的，在HBase中没有表与表之间的关联查询。
4. 都有一个可排序的主键和任意多的列，并且列的数量是可以根据需求动态增加的，同一张表中的不同数据可以有完全不同的列。
5. 并不会占用存储单元，这样可以让我们的表设计的非常稀疏
   每个单元中可以存储多版本的数据，在默认情况下版本号是自动分配的，是插入的时间戳，
6. 关系。在计算过程中边RDD需要顶点的数据的时候，顶点RDD会根据路由表将顶点数据发送
7. 需要移动顶点数据和边数据。而且在大部分图中顶点数据都是远远的小于边数据的，随着迭代次数的增加，需要进行更新的边数据也越来越少，这样可以大大的减少数据的移动量，从而加快

第四章分布式的图计算系统的研究与实现

4.1 图算法的并行化结构

由第二章的介绍可以得知，BSP是一个在图计算中广泛使用的分布式计算模型。在这里我们也采用BSP架构，构建图的分布式计算系统。如下图所示，我们的计算机群中主要有两种角色，一种角色是Master，另一种角色是Worker。Master作为集群的中控系统，主要负责管理集群中的Worker，Master会时刻保持与Worker之间的通信。每当一个计算任务下达的时候Master首先会根据指定算法进行图分区，然后将分区后的子图分别分配给参加计算的Worker。每个Worker都会向Master注册自身的信息，Master中维护着Worker详细信息的状态表，Mater和Worker之间保持着固定频率的心跳，每当Worker的状态发生改变以后就会通知Master，Master会相应的修改Worker详细信息表中该Worker的信息。

在每个超步的计算过程中，Worker都会首先遍历自身的内存，计算内存中每个节点的值。每个Worker计算完成后会暂时休眠，等待其他Worker也完成自身任务的计算，当所有的Worker都计算完毕以后，他们之间会进行互相通信，计算出关联节点的信息，最后统一由Master更新所有节点的值。

采用该计算模型的意义是图分割后的数据可以存储在不同的Worker上，在良好的图分割模型下可以有效地减少Worker之间的通信量。图中的顶点之间的计算采用纯消息传递的方式，每个Worker之间异步计算，Worker之间采用批量消息传输的方式交互信息，整个系统的执行效率大大提升。

我们以PageRank算法来举例说明并行图算法的具体执行过程。PageRank最初只是用来计算网页重要性的，后来逐渐演化成了衡量网络中节点重要性的一个重要指标。

如上图所示，可以认为是一个现实网络中大图的简化。图中有6个节点和11个关系。通过执行PageRank算法，会得到一个各个顶点的概率分布，用来表达链入任意一个节点的可能性。PageRank的计算过程开始时图中的各个节点概率分布是相同的，需要经过多轮迭计算他们的真实值概率分布，通过多轮迭代可以不断地调整每个顶点近似的PageRank值，从而使计算结果更接近于各个节点的理论真实值。

下面我们看一下PageRank算法在BSP并行框架上的具体运行过程。系统接收到一个计算任务之后首先会将任务分配给Master，再由Master统一调度。Master会先根据指定的图分割算法将大图分配到各个Worker上，同时向Worker下达计算任务。

针对图来说，系统接收到该图之后首先会将该图划分为3个部分。节点1，2被划分到Worker1，节点3，4被划分到Worker2，节点5，6被划分到Worker3。每个Worker中维护着本地节点和其他Worker中有关联节点的关系。

Compute，计算本地节点数据的过程，包括合本地其他节点传递过来的消息和分配要传递给出边节点的消息。Worker1节点会计算本地内存中的1，2节点值，Worker2会计算本地内存中的3，4节点的内容，Worker3相应的计算本地内存中5，6节点的内容。

Send，向当前节点的出边节点发送消息。如图所示，Worker1中的节点1会向节点2，3，4发送消息，节点2向节点1发送消息。Worker2中的节点3分别向节点2，4发送消息，节点4分别向节点3,5发送消息。Worker3中的节点6向节点3，5发送消息，节,5向节点2发送消息。

Barrier，同步边界。在系统中不同的Worker异步的执行自身的任务，当一个Worker执行完自身任务时会先陷入休眠状态，等待其他的Worker执行自身的计算任务。当所有的Worker都执行完自身的计算任务之后，不同的Worker之间会互相通信，同步计算结果到关联的的Worker中。

Parse，节点上消息的聚合。每个Worker上的节点进行消息的聚合。Worker1中节点2接收来自节点1,3,5的消息。节点1接收来自节点2的消息。Worker2中节点3接收来自节点1,4,6的消息，节点4接收来自节点1,3的消息。Worker3中节点5接收来自节点节点4,6的消息。

如上图所示，左上角的部分是该系统中不同Worker上的内存模型。右上角的部分是计算过程中一个单独的顶点需要执行的行为。下面是该计算过程中每个超步的具体执行过程。该模型虽然可以完美的实习图的分布式计算，但是潜在的问题也是非常大的，每个Worker之间的信息同步太多，严重的影响到了系统的整体执行效率。

因此我们的优化重点可以放到Worker之间的信息同步花费上。上图采用的是边分割的图分割方式，采用边分割的好处是可以减少存储的花销。但是带来的影响就是通信花销会变大，在这个存储资源代价远低于计算和网络代价的时代，这个优势可以忽略不计。因此我们可以考虑另一种图的分割方式，点分割（Vertex Cut），点分割的方式是一个节点可以跨多个机器存储，这些个存储里有一个主存储（MainStore）和若干个副本存储(Replica)。每个superstep计算完之后我们只用同步跨机器存储的MainStore和Replica即可，可以有效的降低通信代价。具体操作如下图所示：

改进的BSP图计算模型执行过程如下，系统接到计算任务之后会把数据分成三份，并且被切分的节点要生成一个备份。如上图所示，Worker1存储节点1，2并且存储节点3，5的Replica。Worker2存储节点3，4的MainStore，并且存储节点1，6的Replica。Worker3存储节点5，6的MainStore并且存储节点4的Replica。

Compute，计算本地节点数据的过程，包括本地其他节点传递过来的消息和分配要传递给出边节点的消息。Worker1中节

点1接收来自节点2的消息，并且发送消息给节点2。Worker2中节点3发送消息给节点4，并且接收来自节点1,4,6的消息。节点4接收来自1,3的消息，并且发消息给节点3。Worker3中节点5接收来自节点4的消息，节点6发消息给节点5

Send，被分割的节点由MainStore向Replica发送消息。

Barrier，不同的Worker同步消息，主要是被分割的节点MainStore和Replica进行消息的同步。

通过上面的对比可以看出，使用点分隔的方式可以大大地减少不同Worker之间的通信代价，使更多的计算都在本地完成，极大地提高了系统的执行效率。

4.2 并行化增量式算法的研究与实现

动态图计算是为了适应快速变化的网络结构的一种实时计算方式，动态图计算的原理是，每进行一次新的计算都利用到原有的计算结果，从而减少总体计算规模，可以更加快速的生成最新的计算结果。动态图计算将动态图存储为阶段性的静态图组合，G={Gt1, Gt2…Gtn}表示该图在t1到tn的动态集合。t1,t2…tn如上一节介绍，是一系列的非重合时间区间。

动态图的计算过程是：

步骤1：计算两个时刻静态图的区别。标记Gt1–Gt2为旧的节点，记为Gold，标记Gt2-Gt1为新发生改变的节点，记为Gnew。

步骤2：如图1-(b)所示，设置Gnew为活跃顶点，设置Gold为非活跃顶点。

步骤3：如图1-(a)所示，从每个活跃顶点开始计算，并且将计算结果传递到相邻顶点。

步骤4：对每个顶点检查旧值与新值之间的diff，当diff>detla则设置该节点为活跃节点，detla为阈值，否则设置该节点为非活跃节点。

步骤5：重复步骤3到5，直到所有节点都变为非活跃定点。

动态图算法相对于全量静态图算法最大的特点是，对于动态图的计算，每次只要计算上一个时间和当前时间中整个图改变的部分，从而在计算一开始的时候可以只激活变化的那部分节点，而全量静态图算法则要激活所有的节点。因此动态图算法相对于全量静态图算法在图规模非常大，并且每次变化的节点数量很少的情况下具有很大的优势。动态图算法只对两个时间产生变化的那部分数据敏感，计算时间不会随着整个图的规模增大而线性增加，这对于计算资源有限的情况是非常有益的。

在以上背景下，我们实现了一个基于分布式图处理框架的动态图算法，嵌入的图算法主要包括PageRank，TrustRank和ShortestPath等。分布式动态图算法的两个设计难点是，消息传递方式和同步模型的设计，在本文中我们采用触发式消息传递模式和大整体同步模型，每个计算节点内进行异步计算，不同的计算节点之间再进行整体同步。

动态图的计算是根据触发机制执行的，大图中发生变化的节点或者边会直接影响到其周围的节点或者边。用计算PageRank算法来举例，图2是一个PageRank算法动态计算的消息传播过程。PageRank最初是用来计算网页重要性的，后来逐渐发展成为衡量网络中节点重要性的一个重要指标。

图2 PageRank动态计算消息传递

Fig. 2 PageRank dynamic calculation of messaging model

PageRank的计算过程中每个节点都会出边指向的邻居节点传递自己的重要性贡献，直到整个网络中的每个节点重要性趋于稳定为止。在G1时刻整个图的节点包括{A,B,C,D,E,F}，G2时刻新增了节点{G,H}，新增了边{c,d,e}，而且减少了节点{F}，减少了边{a,b}。对应的PageRank的动态算法执行过程为，首先{G,H,F,a,b,c,d,e}作为发生改变的节点和边，会直接影响到与他们直接相连的节点{A,B,D}。节点F以及边{a,b}的消失主要影响到节点{A,B}，对于节点A来说，A需要减去来自节点F的重要性贡献，然后A用新的重要性值和旧的重要性值比较，如果阈值大于detla则进一步将影响力传递到{B,D}。对于节点B的影响则是B将要重新划分影响力的传递，将原先传递到F的影响力也传递到C。节点{G,F}的增加和边{c,d,e}的增加则主要影响到节点{B,D}，对于节点B来说，主要是接收从节点G经由边c传递过来的影响力，变化值高于detala则要进一步传播。对于节点D来说，需要将自身的影响力传递给H。

对于PageRank这种迭代式算法，利用动态计算方式可以在计算初始只激活与更改部分相关的节点，利用差量的计算方式，以及一个变化阈值detla，可以在一个节点的差量变化小于detla时停止进一步传播，从而加速整个计算过程。

图3 Shortest Path动态计算消息传递

Fig. 3 Shortest Path dynamic calculation of messaging

另一个图计算中常用的算法即最短路径，在地图中的路径规划，社交关系网络中的好友推荐，最短路径是一个常用的图算法。在这里我们采用一种在大规模网络中比较实用的一种基于地标的最短路算法[12]。该算法的特点是采用一些节点S作为地标集合。图中的每个节点都会维护与地标集S的最短距离。这样求图中的任意两个节点v1和v2的最短距离就会演化成求v1到地标集S的最短距离加上地标集S到v2的最短距离。该算法已经证明在地标选取合理的情况下是有效的。

如图3演示了基于地标的最短路算法动态计算过程。{S1,S2}是我们选取的地标集。在G1时刻，图中的顶点主要包括{A,B,C,D,E,F,G,H,I}。在G2时刻图中新增了点{J,K}，以及减少了点H。对于点H的消失来说，首先我们会检查点H的邻居节点，查看H的邻居节点到地标集S的最短距离是否有更新。可以得到A到S1的最短距离从2更新为3，A到S2的最短距离从3更新到4。然后继续检查A的邻居节点是否需要更新，如果不需要更新则停止传播。对于新增的节点J来说，检查J的邻居节点G到地标集的最短距离是否需要更新。G到S2的最短距离由4更新到2，G到S1的最短距离不需要更新。再检查G的邻居节点，以此类推。

动态图的计算相对于静态图计算的的最大特点就是触发计算的方式，动态图采用更新，触发的方式驱动整个计算的进行

，因此我们设计出三个重要的函数。首先是SendMessage函数，SendMessage是整个计算的发起点。在每次动态计算中两个不同时刻的图快照会先进行快速对比，对比得到两个图之间的差异之后就会将改变的节点设置为活跃，然后由活跃节点向邻居节点发出信息。另一个重要的是ReceiveMessage函数的设计，ReceiveMessage函数主要负责从邻居节点接收信息，将接收到的信息和节点本身的信息合并，并且更新节点本身信息。AggreateMessage函数主要负责将不同计算节点上得到的结果聚合。

整个算法的运行过程是，首先在每个Worker上检查图中的活跃节点，由活跃节点开始SendMessage。然后活跃节点影响到的节点会ReceiveMessage，并且和自身的原有信息合并。最后不同的Worker之间会相互通信，做全图聚合处理。

算法：PageRank动态图算法

输入：目标图 G，其时间区间划分为{Gt1, Gt2… Gtn}

输出：一系列时刻计算结果的集合

SendMessage

1: input(Gt, Gt+1)

2: Gold= Gt ∩ Gt+1

3: Gnew= Gt+1- (Gt ∩ Gt+1)

4: Gactive= active(Gnew)

5: for 任意的 vj∈ Gactive do

6: if absvjprev- vjcur>detla

7: for 任意的 v ∈ vj的邻居

8: if v ∈ Gold

9: SendMessage(vjprev- vjcurdegree)

10: else

11: SendMessage(vjcurdegree)

12: end if

13: end for

14: end if

ReceiveMessage

1: if v ∈ Gold

2: if message>detla

3: updatev; active(v)

4: else

5: update(v)

6: end if

7: else

8: if message- vcur>detla

9: update(v); active(v)

10: else

11: update(v)

12: end if

13: end if

Dynamic Algorithm

1: while Gactive.count>threshold

2: SendMessage

3: ReceiveMessage

4: end while

5: output(Gt+1new)

4.3 增量式算法效果评估

阿斯顿我们在Spark GraphX图处理框架上实现了上述的分布式动态图算法。整个系统部署在一个四个节点的集群上运行，每个计算节点的配置为64 GB DDR3内存和3.10 GHz Intel Xeon E3-1220 v2 CPU，操作系统为Ubuntu16.04，我们使用了从斯坦福大学公开的SNAP[14]上获取的4个开源数据集。数据集的详细信息如表1所示，主要包括数据集名称、图类型、节点数目和边数目。

表1： 实验所用数据集

Tab. 1 Datasets

数据集类型节点数目边数目

wiki-Vote 有向图 7715 1036892
soc-Slashdot0811 有向图 77360 905468
email-EuAll 有向图 265,009 420,045
web-Google 有向图 875,713 5,105,039

我们在表1数据集上对PageRank的动态图算法上进行了实验。数据规模从7000多到百万级别，分别以12%的增量规模进行多次验证。如图5所示，图中横轴代表的是超步次数，纵轴代表的是每个超步被激活的节点数量。

从图中可以看出，使用动态算法计算的方式，每次初始阶段只用激活少量的节点，这种方式可以大大的加快收敛速度。从具体效果来看，除了wiki-Vote数据集上动态图算法的性能提升不明显，其他几个数据集上都使整体收敛速度提升了3倍左右。证明了使用动态算法的确可以极大地加速算法的收敛，证明了该算法在提升算法效率方面的有效性。

图5 在不同数据集下动态式算法和普通算法的性能差异

Fig. 5 Performance differences between dynamic and common algorithms for different data sets

另一方面，动态算法使用了触发、传播的计算方式。而触发的方式是根据值的改变是否大于阈值而决定的，这就会造成一定的准确度丢失。下面一个实验，我们在web-Google数据集上分别用不同规模的增长量验证了该算法在准确度方面的表现。

如图6所示，我们采用500K的节点作为已有原始图，使用其他300K的节点作为增量图。分别采用了2%、4%、8%、16%、32%的增量方式进行了多次实验。实验结果如图中所示，随着增量规模的增加，准确度的变化呈现了一个先增后减的态势。在节点规模增加到800K的时候，8%增量的准确度最高，其次依次是4%增量、16%增量和2%增量，32%增量的表现最差。为我们在工程实践中的具体使用提供了依据。

实验体结果表明，动态图计算相较于全量静态图计算在计算效率上有很大的优势，但是同时也会带来一些准确度的损失。同时也表明，随着时间的推移，我们需要使用全量式算法对动态计算的结果做校正，从而保证结果不至于产生太大偏差。

4.4 小结

节基于Spark GraphX实现了一种基于动态图计算的算法，并且在多个大规模数据集上进行了验证，数据规模从几千到上百万。根据实验结果来看，我们的算法对于大图计算的时间效率带来了较大的提升。虽然同时也有准确度方面的损失，但是只要定期使用全量式算法进行更正，还是可以满足很多实时系统的需求的。同时针对准确度损失这方面，从实验的结果来看，过大或者过小的动态更新都会造成准确度的下降。在这方面，未来希望可以在本文的基础上作进一步研究，选择适合的全量更新点，让动态图算法可以获得更好的效果。

第五章分布式图计算分析平台的设计与实现

5.1 总体设计

本章首先介绍了该分布式图计算平台的详细架构。然后详细描述了各个模块的详细设计过程，最后展示了系统的用户操作界面。

5.1.1 系统架构

根据前几节的介绍，我们实现了一个完整的分布式图计算平台，在这里我们叫做OpenGraph。OpenGraph封装了前几节我们提到的各种功能，包括时序图的存储能力，动态图的实时计算能力，图关系的快速查询能力，还有基于历史的分析能力，下面我们简要介绍一下OpenGraph的主要架构。

OpenGraph主要有4个组成模块，分别是底层的时序图存储模块，时序图存储模块是基于HBase实现的一个时序图存储模型，主要由HBase做后端存储，TSGraph上层封装，主要负责执行常见的查询任务，特点是响应速度快，准确率高。第二个模块是计算副本模块，计算副本模块是为了实现存储与计算隔离而专门设计的模块，该模块主要由Hadoop HDFS和Spark RDD构成，主体存储模块的数据会定期以快照的形式发送到HDFS上，Spark计算时会读取HDFS上的数据并且按通信代价最小的目标做分割，以Spark RDD的形式存储在Spark集群的内存上，供Spark调度计算。第三个模块是增量式计算模块，在该模块中我们实现了增量式算法，因为在图的实时分析过程中随着图的规模增大全量式算法会非常浪费时间，增量式算法可以在计算初始阶段只用激活少量的节点从而加快整个计算过程，在可容忍的准确性损失范围内，增量式算法可以较大的提升性能。最后一个模块就是我们的上层封装应用，我们基于以上三个模块搭建了一个易用的图计算平台，该计算平台主要支持图的增量式导入，图的增量式计算，基于图的查询还有在图上基于历史事件的分析。

5.2 详细设计与实现

5.2.1 主体存储模块

主体存储模块主要分两个部分，后端HBase和上层TSGraph。后端HBase用来负责图数据在磁盘上的存储。为了快速的检索图数据，我们设计了适合存储图数据的BigTable存储模型，通过该存储模型和Hbase中RowKey的优化，使得单个节点检索及一层关系检索降低到了毫秒级别，完全可以满足日常中的大部分检索任务。对于深层次的图检索，我们在Edge的存储上设

置索引，并且采用广度优先搜索和深度优先搜索结合的方式，在百万级别的图上可以快速地检索到10跳以内的图关系。

基于HBase的后端存储我们实现了一个TSGraph（Time Series Graph）的上层封装，TSGraph的作用是让我们用图的思想来进行图上的查询。该查询方式借鉴了Gremlin语言，该语言是一种DSL语言。TSGraph还实现了时序图的检索，通过在HBase上将时间属性和Vertex联合作为RowKey的设计，时间序列相同或者近似的数据会存放在物理位置靠近的位置，查询的时候通过顺序加载可以方便的加载出一个时刻的静态图，使得静态图的查询速度大大加快。TSGraph是和平台业务交互的一个主要模块，也算是一个中间模块。有了TSGraph的存在，上层平台业务不需要知道底层到底有什么东西，只用向TSGraph提交计算任务即可。

同时TSgraph还负责和计算副本之间的交互，每当一个计算任务产生的时候，TSGraph会向计算副本提交一个所需存储的快照。当计算副本得到计算结果的时候会重新返回到TSGraph中，TSGraph此时还要负责将返回的数据和主题存储中的数据做Merge（因为计算过程中主题存储的数据也可能发生变化）。Merge成功以后会修改主题存储的数据。

5.2.2 计算副本模块

计算副本模块主要由HDFS和Spark RDD两个模块。HDFS主要用来存储一些中间计算结果，Spark RDD则是主要用来工计算任务的调用。计算副本模块主要连接两个模块，一个是主体存储模块，另一个是增量计算模块。

计算副本模块主要是为了与主体存储模块的数据隔离开来，一方面是这样的设计可以增大整个计算的吞吐量，同时也减少了实时计算任务的延时。计算副本模块和主体存储模块与增量计算模块的交互是这样的，每次TSGraph收到计算任务，会先根据任务的需要加载需要计算的数据。加载完需要的数据之后会存储到HDFS上，如果该任务是一个新的计算任务则HDFS直接存储。如果该任务是一个增量式计算任务则要将本次加载的数据与HDFS上本身的数据先进行Merge操作，Merge操作的主要作用是为了寻找两次数据之间的差异，从而确定本次增量式计算一开始要激活的节点。进行该步骤操作之后，整个图中的数据会被标记为两部分，一部分是未发生变化的节点，记做Gold，另一部分是发生改变的节点，记做Gnew，增量式计算首先会从Gnew开始触发计算。

当Spark要进行计算时会先将HDFS上处理好的数据加载到Spark RDD上面。Spark RDD会根据上一步已经做好的标记确定计算的执行过程，在整个迭代式计算的过程中，如果内存一一直可用，则整个计算过程中的中间计算结果都会一直存放在Spark RDD上。如果计算过程中内存不够，此时还可以将部分计算结果存放在HDFS上，下次计算的时候再从HDFS上读取。

采用存储与计算隔离的方式，一方面是增加了整个系统的吞吐量，另一方面也降低了一个存储的负载，减少了读写锁的操作，大大地简化了系统的复杂性。

5.2.3 增量计算模块

增量计算模块是整个动态图计算的核心，采用增量式计算的思想，使得大规模图的实时计算变为了可能。更好的反映了现实世界网络中的真实状态。增量式计算的主要思想是差异化计算，在增量式计算的初始阶段，每次开始只用激活少量相较于上次发生改变的节点，从而可以达到快速收敛的效果。

和增量式计算模块主要交互的两个模块是主体存储模块的TSGraph和计算副本模块的Spark RDD。每次TSGraph拿到要计算的任务首先会分解任务，一方面加载数据到计算副本模块，另一方面发送计算任务到增量计算模块。同时为了任务不至于丢失，保证任务的顺序执行，在TSGraph和增量式计算模块中间使用了RabbitMQ，RabbitMQ是一种消息队列。RabbitMQ的作用一方面是TSGraph向Spark提交计算任务，另一方面是Spark完成任务的时候会返回任务完成消息的给TSGraph，此时TSGraph会从计算副本抓取计算结果并和本地数据Merge后更新HBase中的本地数据。因为TSGraph提交任务的速度和Spark执行任务的速度不一定会匹配，Spark返回任务完成的速度和TSGraph从计算副本同步数据的速度也可能不匹配，如果不使用消息队列的话会造成消息的覆盖，从而造成任务丢失或者是数据丢失。而采用消息队列的模式，可以保证任务一个个的执行，从而保证系统的容错性。

增量式算法模块还是用Spark GraphX的API，实现了基于BSP的增量式算法，主要包括增量式的PageRank，增量式的TrustRank，增量是的TunkRank和增量式的ShortestPatth。当接收到任务的时候，增量式计算模块会选择适合的计算方式保证计算的正常执行。增量式计算模块还有一个重要的功能是全量同步功能，增量式计算虽然可以带来计算效率的提升，但是同时也会造成结果准确度的算式，特别是在多次增量式计算之后结果的准确度会持续下降。此时需要全量计算来重新修正计算结果。修正计算结果的方法可以周期性的修复也可以按照增量数据的规模修复，具体可以由用户自己选择。

5.2.4 平台业务模块

平台业务模块主要是后端与前端的交互。为了实现一个易用的分布式图分析平台，我们使用MVC架构实现了一个业务平台。

平台业务模块主要包括DataLoad Servixe，Runtime Service，Query Service和Analysis Service几个模块。Dataload Service主要负责数据的加载，OpenGraph的数据模型和一般计算平台的数据模型不一样，特别是对于增量式计算的方式，需要增量式的加载数据，因此我们在这里写了一个单独的DataLoad Service模块，专门用来加载数据。Runtime模块同样也是为了增量式计算服务，增量式计算是一个自动触发持续执行的过程，在这个执行过程中很少需要人工的干预，包括数据的增量式加载计算，数据的更新，结果的修正等都包含在Runtime模块中。Query模块主要执行图上的一些查询操作，这个操作不涉及增量式计算的过程，主要是一个OLTP任务。Analysis模块则是图上的集成分析模块，因为并不是所有的算法都适合增量式计算，有的计算还是得全量式计算才能得到准确的结果，Analysis模块就是为了执行除了增量式计算之外的图计算任务。

5.3 OpenGraph演示

如图所示就是OpenGraph动态图计算平台的首页，首页上首先有实时分析和历史分量两个导航栏。实时分析主要是执行增量式计算的任务，历史分析主要是为了计算图上的一些属性随着图演变的变化。

首先看实时分析页面，最上层是一个查询任务框。这个框的作用主要是实现一些基本查询的任务，查询过程中我们使用了OpenCypher这种基于图的查询语言。如图中所示，我们运行match (n)-[r]-(t) return n,r,t会返回整个图的结构，图中彩色节点部分就是我操作的大图。图中不同节点的大小代表不同节点的PageRank，图中点的大小越大的节点，代表该节点的PageRank也比较大，表示该节点在整个网络中比较重要。图中不同颜色的节点代表不同的社区，由图中可以看出来同一个社区的节点关系比较复杂一点，不同社区的检点关系比较稀疏一点。 该图会随着下层存储中节点数据的变化而变化，反映着整个网络的真实状态。

在该页面上我们不仅展示了图的动态计算，同时也展现了一些其他图上信息的基本操作。这些操作包括一系列的基于图的属性统计和计算，首先是该图的平均度，平均度就是该图中所有节点的度的平均值。平均加权度指在计算平均度的基础上加了边权值的信息。网络直径表示任意两个节点之间的距离最大值。图密度是衡量图上的节点和边的比值的一种方式。平均聚类系数表示一个图形中节点聚集程度的系数。平局路径长度代表任意两个几点之间的平均最短距离。特征向量中心度表示图中节点在网络中的重要性程度。

这些算法都会根据用户的需要点击运行按钮而进行实时计算。例如平均度的计算，用户只需要点击平均度的运行按钮，系统就会生成图中所有节点的平均度。并且会弹出一个新的窗口，新的窗口上有图中所有节点的度分布。可供用户详细查看。

然后还有一个历史分析页面，因为我们的系统可以存一个动态图的历史信息。所以我们可以利用历史信息做一个图属性的历史分析。执行图属性的历史分析首先要选择历史分析的时间范围。

我们这里的历史分析主要以天为单位，主要是因为我们的数据更新并不是很快，以天为单位已经可以很好的反映数据的变化状态了。

如图中所示，我们选择了2017年7月17日至7月28日的历史分析。主要针对三角计数属性进行分析。我们导入的数据是Twitter的评论数据，可以看出随着时间的增加三角计数的个数持续增加，证明了在这段时间内，该图的联系是越来越密切的。同时我们还支持紧密度中心性，结束中心性，连通分量，最大连通分量等属性的历史分析。大大的扩展了我们系统的适用范围。

5.4 小结

本章我们利用前几章的研究搭建了一个易用的动态图分析平台。第一节我们介绍了该平台的总体架构，围绕着平台的架构图做了功能模块的简单介绍。第二节我们从四个方面详细的介绍了该平台的设计理念，这四个模块包括时序图存储模块，计算副本模块，增量计算模块，平台业务模块。分别阐述了这四个模块之间的相互调用关系，分析了这样实现整个平台的优势。最后展示了该平台的前端交互页面，阐述了用户可以在该平台上做的事情。

7. 1_赵炳_基于分布式图计算的大规模网络分析系统的研究_第7部分

总字数：809

相似文献列表　文字复制比：0%(0)　疑似剽窃观点：(0)

原文内容　红色文字表示存在文字复制现象的内容; 绿色文字表示其中标明了引用的内容

第六章总结与展望

随着互联网的飞速发展，原本相互独立或者关系稀少的数据联系越来越紧密，产生了大量带有复杂关系的数据。这些复杂的关系需要图分析技术才能准确的反映数据的真实状态。但是一方面现有的图计算技术都是对静态图的分析，然而现实世界中的数据模型都是时刻变化的，现有的静态计算模型不足以反映网络的真实状态。另一方面由于图的结构复杂，导致图计算的复杂度很难降低下来，分布式的成本也居高不下。这些原因一直制约着图分析方向的前景和图计算的发展。

本文中基于这两个关于图方面分析的痛点都提出了自己的方案，针对于现有的图分析都是基于静态图的痛点，我们设计了适合时序图存储的分布式图存储系统。通过存储结构的优化，查询索引的优化，可以在时间维度上快速查询到动态图的某个静态快照。针对现有的图计算算法复杂度居高不下的问题我们，设计实现了增量式图算法。通过实验表明我们的增量式图算法在一定的准确度丢失的容忍程度下，可以极大地提高图计算的执行效率。同时针对增量式算法会丢失准确度的这个问题，我们设计了全量校正的方案。可以保证我们的计算结果一直在某个准确度丢失的容忍范围内。

最后我们基于以上提到的动态图存储系统和增量式计算系统实现了一个支持动态图计算的综合图分析平台。该图分析平台主要有四个某块，包括数据的管理加载，图关系得查询，增量式算法的实时运行，全面的图属性分析指标。通过该分析平台，用户可以快速的对整个图有一个了解，并且可以根据自己的需求计算复杂的实时性任务。

虽然本文实现了一个完整的图分析平台，但是在一些地方仍然存在着优化空间。一个是在设计TSGraph的时候，没有考虑到更加复杂的情况，对读取和写入数据的安全性方面目前只加了读写锁，没有达到事务级别的支持，这对于一些复杂的操作可能会导致不一致的情况。另一方面关于增量式算法的研究，目前改造的增量式算法还比较少，下一步可以从更多算法的角度考虑，实现这些算法的增量化实现

表格检测详细结果

原文表格1：未获取到表格标题	共有1个相似表格	
follow		
id	person1	person2
1	1	2
2	3	1
3	4	1

相似表格1：表2湿气对2种体系附着力的影响

相似度：69.23%

来源：一种用于激光固化快速成形的低翘曲光敏树脂的研究-段玉岗,王学让,王素琴,李涤尘,卢秉恒-《西安交通大学学报》-2001-11-20

放置时间/周	0	1	2	3	4
阳离子体系/级	1	2	3	4	5
混杂体系/级	1	1	1	1	1

说明：1.总文字复制比：被检测论文总重合字数在总字数中所占的比例

2.去除引用文献复制比：去除系统识别为引用的文献后，计算出来的重合字数在总字数中所占的比例

3.去除本人已发表文献复制比：去除作者本人已发表文献后，计算出来的重合字数在总字数中所占的比例

4.单篇最大文字复制比：被检测文献与所有相似文献比对后，重合字数占总字数的比例最大的那一篇文献的文字复制比

5.指标是由系统根据《学术论文不端行为的界定标准》自动生成的

6.红色文字表示文字复制部分;绿色文字表示引用部分

7.本报告单仅对您所选择比对资源范围内检测结果负责

amlc@cnki.net

http://check.cnki.net/

http://e.weibo.com/u/3194559873