**面向连续流式图计算的系统设计与实现**

**（面向流批混合的图计算系统的设计与实现）**

# 问题背景

## 研究背景与意义

[什么是图数据]

图是计算机科学中常用的一类数据结构，它能够很好的表达了数据之间的关联性。现实世界中有很多数据都可以抽象成图数据，例如Web网页之间的链接、社交人物之间的互动以及买卖双方的交易都可以抽象成图数据结构。而随着互联网的快速发展，图数据的总量也在急剧增加。如截至2014年第一季度Facebook包含了12.3亿个活跃用户，每个用户平均好友130个；web链接图顶点数达到Ｔ级，边的个数达到Ｐ级[1]。

[图计算模型的运用]

因为图数据能够很好的表达数据之间的关联性和聚集情况，因此针对图数据表达的关联关系可以挖掘出很多有用信息。比如，通过为购物者之间的关系建模，就能很快找到口味相似的用户，并为之推荐商品；或者在社交网络中，通过传播关系发现意见领袖。图数据及相关的处理框架已经广泛运用在社交分析、商品推荐、舆论监测、欺诈检测等各个领域。

[图计算模型面临的挑战]

处理这些海量动态的图数据也对现有的图计算模型提出了挑战。一方面，这种超大规模的图数据很难一次性的全部导入内存进行处理，即使能够借助外存一批一批的处理图数据，也使得计算延迟显著增加；另一方面，这些数据又是动态变化，实时更新的，现有的图计算模型要能够在这种动态的数据集上进行计算。

[传统的解决方案]

现在有很多成熟的图计算系统，例如Google Pregel, Spark GraphX, 这些图计算模型都采用了分布式的集群和BSP消息同步机制来处理图数据。然而这些系统都是在静态的图数据结构上进行的离线处理。即每次针对整体的图进行计算，这使得当面临图动态变化时，需要在变化后的整个图上重新计算一遍。这使得用户等待周期长，无法满足实时计算的要求，也是浪费了系统资源。因为针对某些场景，可能只需考虑图变化部分所带来的的影响，而无需整个图都重新计算一遍。

[本文解决方案]

本文工作主要有以下几点：

（1）本文分析了现有的图计算模型:**全局静态的批量处理模型**和**连续流式的增量处理模型**，分析总结了现有的批处理模型的特点和应用场景，同时给出了面向连续流式图数据的处理模型的定义、解决方案和应用场景，最后对比分析两者之间的区别和联系；

（2）针对现有的图计算模型的优缺点，本文设计了一套针对连续流式图数据的处理模型，很好的解决了海量动态图数据的处理和计算问题，这使得用户不需要将全部的图数据导入内存之后再进行计算，能够在有限的内存内完成对无限的图数据流的处理，而且这种计算是实时的，即能够在计算过程中实时返回计算结果，而无需等到全部计算结束时才返回计算结果；

（3）本文在自己建立的连续流式图处理的框架上面，实现了几个典型的图算法，并且结合具体场景讲述了这些算法的用途，最后给出了这些算法的评测方案和评测结果。

1. 为什么批处理的模型不适用？

2. 问题的定义：（流图？计算？） 单机，分布式，精确结果，非精确结果，通用模型，

3. 引出我们要解决的问题，模型改进、算法改进。

Streaming不能直接拿来用，把过程讲清楚。

把为什么要讲清楚。

## 图计算模型现状

### 批处理图模型

[同步计算模型]

图数据能够丰富表达事物之间的关联情况，它被广泛运用于社交分析、商品推荐、欺诈检测等各个领域。图算法一般需要多次迭代，计算过程也依赖于顶点之间的通信，而传统的MapReduce模型更倾向于处理彼此独立的任务，因此MapReduce及其开源实现的Hadoop为代表的传统面向数据并行（Data-Parallel）的计算模型难以提供高效的支持。

为了解决海量图计算问题，Google公司提出了基于BSP(Bulk Synchronous Parallel)思想的大规模分布式图计算平台Pregel。专门解决网页链接分析、社交数据挖掘等图计算问题。Pregel使用了以顶点为中心的计算模型,将整个计算过程分解成由若干个顺序运行的超步（superstep），在每个超步中，活跃的节点(active vertex)将将接收上个超步中其他节点发送过来的消息，并据此执行用户自定义的计算函数，改变自己的状态，同时将更新的状态再发送给其他节点，这些消息会在下个超步中被其他节点接收并处理，然后该点进入不活跃状态（inactive vertex）。不活跃的点在下个超步中接收到其他节点的消息会变得活跃，反之如果没有接收其他节点的消息，将继续保持不活跃的状态，也不会向其他节点发送消息。超步内，各个节点可以并行处理，而超步之间会对消息进行同步，通过这样以超步为单位的方式迭代运行，直至所有节点都变得不活跃或没有新的消息产生。用户只需要自己定义超步内节点的计算逻辑，即可实现计算功能。计算模型如下图1所示：



图 1‑1 Pregel计算模型

[异步计算模型]

异步计算模型GraphLab

[单机计算模型]

### 流处理图模型

KineoGraph使用区间提交协议

Naiad Streaming 增量计算支持

Trinity 动态图查询

## 本文的研究贡献

# 系统设计

## 系统架构

[总领原文的段落]

[物理层面]

从**物理层面**上来看，面向连续流式图计算的系统架构是采用经典的Master-Slave形式的架构。用户通过Client向Master节点提交任务，Master节点是主节点，负责任务的调度和分配，通常情况下至少有一个主节点，可以通过建立多个主节点来提高系统的可用性；Slave节点是真正的工作节点，负责执行具体的任务，并且向Master节点反馈任务的执行情况。他们的通信过程如下图所示：



图 2‑1 系统架构图

系统的架构如图2.所示，系统的各个组件之间的交互过程为：1. 用户通过Client向Master节点提交任务；2.Master节点立刻向Client反馈任务提交状态，客户端再此异步等待任务执行结果，同时Master扫描Client提交的程序，制定执行计划；3. Master节点向各个Slave节点提交任务，并等待Slave节点的执行结果；4. Slave节点执行Master节点分配的任务，并且向Master节点返回执行结果；5. Master节点向Client返回整个任务的执行结果。

[逻辑层面]

从**逻辑层面**上来看，任务（更准确的说是算子）是在Slave节点上并行执行的。这种并行，既体现在同一台的Slave节点上的不同线程之间的并行执行，又体现在不同Slave节点上的不同进程之间的并行执行。如下图所示。



图 2‑2 任务的并行执行图

如上图所示是一个简单的边流，通过Undirected（）算子操作之后，使得原来的一条边流变成两条边流（即原来边流中边（a, b）通过undirected()映射之后会变成（a, b）（b, a）两条边），这两条边可以独立运行在两个Slave节点上，或者在同一个Slave的两个线程上，再通过keyBy(0)操作（0表示按照边的起始点进行划分）后，将原来的边流分割成多个不同的边流，后续的处理可以在每条流上独立的进行。这样在逻辑层面，任务可以分解成不同的子任务，并在不同的线程或进程中执行。这样的分解充分利用了多核的优势，加快了任务的执行速度。

## 框架设计

传统的面向批处理的图计算框架，是在全部静态图数据上进行计算的。而我们提出的针对连续流式的图计算框架，是希望能够处理动态图数据，这使得原有的解决方案无法满足现有的数据类型。为此我们需要构建新的图计算框架。本文设计的面向连续流式图数据的计算模型如下图所示：



图 2‑3 面向连续流式图数据的计算模型

框架的核心组件主要有：

* **Application**：面向用户的上层运用，这些运用涵盖了典型的使用场景，例如链接分析、欺诈检测、社区发现等，是针对某个具体问题的具体应用；
* **Library**: 框架提供给用户使用的丰富的库函数和图算法。诸如PageRank, Triangle Count, Connected Components等算法包都会在该层中体现；
* **Graph Streaming**: 该层屏蔽了底层的实现细节，向用户提供了一个统一的流式图数据的处理模型，该层需要充分考虑图计算和流处理的特点，针对图计算的核心问题：存储、切分和计算以及流处理的核心问题：计算，分片和窗口，要能够很好的将两者融合起来，为上层用户提供一个统一的视角；
* **Specific Engine**: 最底层的具体的引擎，例如可以借用现有的Spark或Flink这样的分布式并行计算框架作为整个系统的底层执行引擎。

本文的工作重心是希望能够很好的融合流处理和图计算这两个问题，希望能够提供一个面向连续流式图数据的计算模型，并且在该模型之上构建丰富的库函数，方便用户能够根据具体的应用场景来选择合适的图算法，最后会以欺诈检测这样一个具体的应用来演示和验证本文系统的正确性。

## 融合模型

传统的图计算模型中，图数据是静态的，即在计算的过程中不会发生变化，此时需要图计算和批处理进行融合，如在1.2节中分析的同步计算模型，异步计算模型和单机模型都是图计算和批处理的融合，而本文的融合模型是**图计算和流处理的融合**。

融合模型需要充分考虑图计算和流处理的特点，针对不同的应用场景选择合适的方法进行融合。

## 算法设计

[总结]

### Triangle Count

Triangle Count算法是用来统计有向/无向图中的不同三角形的数目。该算法在复杂网络分析、链接标签和推荐等多个领域中都是非常基础重要的度量，也是一些诸如复杂网络、聚集系数等图运算中的基本方法。

在社交网络分析中，社交网络中的三角形数目越多，表明社区内人物之间的联系越紧密。如常见的微博粉丝网络如下左图所示，而微信朋友圈网络如下右图所示，图中橘色标出的三角形即为Triangle Count算法中需要统计的三角形。

 

图 2‑4 微信朋友圈网络 图 2‑5 微博粉丝网络

传统的基于批处理的Triangle Count算法

### Connected Components

### PageRank

# 系统实现

## 框架实现

## 算法实现

# 系统验证