# P0 总结

Streaming Graph

1. 提出计算模型；

2. 计算模型往往具有局限性，在该模型范围内的算法实现。

**计算模型**

**离线计算模型**：现有的模型能够在静态图上进行离线计算和处理。即每次用户给定图结构和要运行的算法，系统经过运算给出本次运行结果。

**增量计算模型**：针对动态实时图计算系统，目前已经 有人提出采用增量更新的方式实现实时计算，如KineoGraph[3]，IncGraph[4]等，然而其计算模型都是基于增量消息进行计算，模型表达能力有限，而且更新都是串行执行、实时性有限．

**单机计算模型：**在OSDI 2012上，GraphLab发表了其处理大规模 图数据的单机系统GraphChi[14]。该系统可以使用一台普通的PC机来完成大规模图数据的计算，为许多没有分布式环境的用户提供了可能。

# P1 增量图计算模型 IncGraph

IncGraph：支持实时计算的大规模增量图处理系统

**摘要**

随着社交网络的流行，越来越多的相关应用要求能够实时地在大规模社会网络图上进行分析和计 算。而目前的图处理系统，如**Google的Pregel，是全局、批量处理的图处理系统**，**并不能实现对图的实时计算。** 因此，提出了一种新的**图增量处理模型**，当一个节点发生变化时，只需要以传播的方式更新局部范围内受影响节点。它**本质上将传统的批量全局计算模型，转化成一系列的增量的、局部的图计算**，保证对图变化的实时处理，并通过避免没有更新节点的重复计算来降低开销。基于这种新的图计算模型，设计了一个低开销、实时的图处理系统——IncGraph，它通过**图切分技术**将计算局部化，保证了计算的低开销，同时利用**主动计算触发**和**反向链式更新**技术，保证了计算的实时性和可靠性。利用真实的社交网络数据证明了IncGraph的低开销、实 时性和扩展性。IncGraph的提出会为社交网络应用提供更为灵活的计算框架。

**1. 引言**

社会网络的流行成为构建实时图处理系统的一个重要需求。目前流行的图计算模型Pregel[2]是**全局的、批量的计算模型**，由于社会网络的规模比较大，单次基于图的运算需要很长时间。因此，社会网络的实际应用需要一种实时性很高的图计算系统。目前国际上对这种实时图计算系统的研究刚刚起步，还没有具体的方案被提出。

本文提出的是一种**局部的、增量的计算模型**，保证对图的变化的实时处理，并且避免对没有更新的节点的重复计算来降低开销。主要有以下关键技术：

**（1）图切分技术。**通过将社会网络图切分为多个尽可能独立的、连接紧密的子图，并把每个子图上的运算放在一台机器上，保证了计算的局部性，减少了跨机器间的通迅开销。

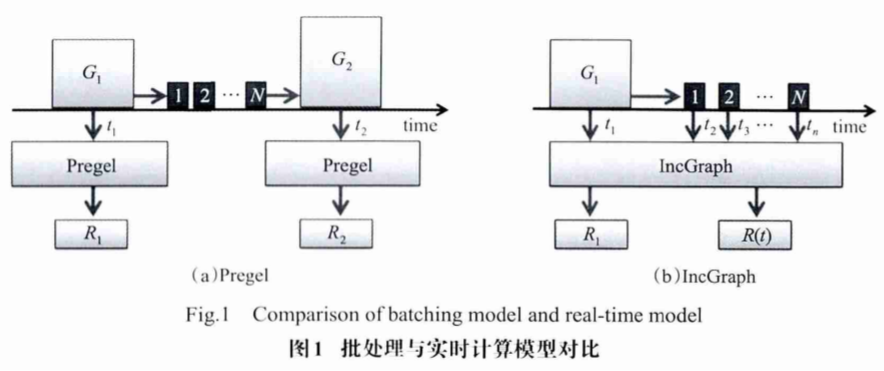
**（2）主动计算触发技术。**当图中节点发生更新 时，主动触发被影响节点的计算，保证了系统只需要 在更新时发生增量的局部计算。

**（3）反向链式更新技术。**通过正向传播计算，反向链式更新节点索引和数据的方式，保证了计算的 低开销和可靠性。

最后，采用大规模在线社交网络（人人网）的真实数据进行验证，证明了系统的计算开销非常小，对结果的更新控制在毫秒级别。

**2. 模型**

**2.1 计算概览**



当前运用广泛的图处理系统如Pregel都是批处理系统。其需要定期将全图重新计算，即使全图局部更 改，也需全局计算，具有较大的开销和延时。，IncGraph不需要每次全局地重新计算全图，只需将每次局部更新实时地输入系统，IncGraph就会将结果更新至最新状态。

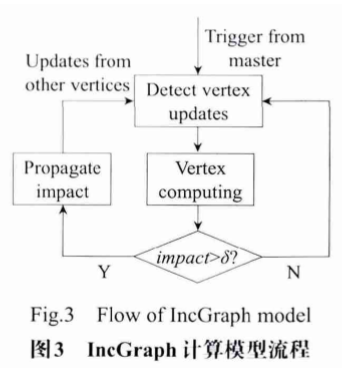
如图1所示， 在Pregel系统中，对图 G1 进行一次批处理得到结果 R1，当经过时间 t 后，变成图 G2，又需要一次重新全局计算。但在IncGraph中，只需在每个时间 t处理局 部的更新，并实时地输出最新的结果R(t)。

因为IncGraph主要关注如何处理增量更新计算，所以在进入实时计算之前，需要将现有图的结果以离线批处理的方式计算出最新结果。离线批处理模块可以利用现有成熟模型来完成，如Pregel、Giraph 或MapReduce等。当批处理完成后，IncGraph将进入实时守护状态，监控图的更新，并实时作出计算响应。（注1：即在系统启动阶段，是现有批处理过程，在批处理结束之后，系统进入实时处理状态，接收数据并实时计算）

**2.2 增量图处理模型**

所谓增量式计算，就是随着图数据和结构的不断变化，系统只根据最近的变化来实时计算最新结果，计算是局部方式的计算，只计算由变化所引起的局部的部分结果的改变，并不需要对全图进行重新计算。当然，并不是所有的算法都能进行增量式计算。（注2：该系统定义的是增量式计算模型，需要注意的是并不是所有的算法都符合增量式计算模型。）

模型采用以**节点为中心**的计算模式，每个节点在逻辑上可以并行地执行由用户定义的程序，并将计算相关的结果保存在该节点上。开始，当图的某个节点发生变化时，该节点将会触发由用户自定义的程序，从而 计算出该节点的最新值，该值保存在节点的**state**变量中。每次state变化都会对其他节点带来影响，这个影响值可由用户自定义计算，默认为state的变化，并保存在**impact**变量中。当impact足够大时，节点将该影响传播给其邻居节点，从而邻居节点被触发，也开始重新计算，整个过程直到传播停止为止。



**2.3 API及模型实现**

**// properties**

state: 节点的状态值。

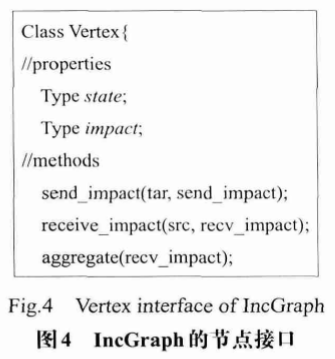
impact: 节点的影响值。每次节点的state变化都会给其他节点带来影响，带来多大的影响可以由用户自己定义。而且当影响足够大时，就会将这个影响传给其他节点。

**// methods**

send\_impact(tar,send\_impact)：发送影响值接口，当某个节点重新计算后，它会发送其对邻接点的影响并触发邻接点的操作。

receive\_impact(src,recv\_impact)：接收影响值接口，该接口是被动触发的，主要功能是接收邻接点发送过来的影响值，并将其聚合到现有的impact上。

aggregate(): 该接口是用来聚合所有接收到的影响值到当前自身影响值impact上。这里的聚合函数需要用户自定义，默认情况是相加。然后判断影响值的大小，如果大于一个阈值，就会触发用户自定义的计算程序进行重新计算，并将计算出的影响值发送给其所有邻居节点，从而触发邻居节点的计算。算法如算法1.



|  |
| --- |
| 算法1： IncGraph Vertex Computation |
| 1. receive\_impact(ipct)  2. impact=aggregate(impact, ipct)  3. If impact < delt:  4. continue  5. Else:  6. //user\_defined\_computation()  7. For nbr in get\_neighbor():  8. send\_impact(nbr, my\_impact) |

**2.4 应用示例**

本文以TrustRank为例来演示如何使用IncGraph实现增量和实时计算。TrustRank是类似于PageRank的网络用户信誉值计算程序，其需要全局迭代计算。TrustRank计算公式如下所示：



T(u) 表示用户u的TrustRank值；

I(u) 表示节点 u的初始化TrustRank值；（在TrustRank中会有一个可信种子集合S，对于集合内元素u，其初始值 I(u) 便是 1/|S|，其他节点的初始值为0）;

应用到IncGraph模型中，每个节点的state便是当前该节点TrustRank值，节点的impact可定义为父节点的TrustRank值除以其出度，即T(v)/out\_deg(v)。这里的聚合函数aggregate（）只需要加法即可，impact的误差delt可以按照TrustRank迭代误差定义，一般取1E-8。

**3. IncGraph系统架构**

**3.1 系统概览**

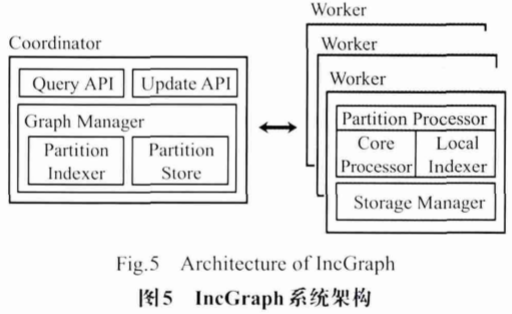
IncGraph是一个实现**大规模实时处理和增量计算**模型的图处理系统。通过**控制节点**和**工作节点**的相互协作，以及运用**图切分技术**、**主动计算触发技术**和**反向链式更新技术**，完成系统实时、增量计算的功能。

**3.2 核心组件介绍**

**Coordinator(控制节点)：**整个系统的指挥者和**协调者**，主要负责接受外部的更新和查询请求，图的划分和索引的建立，各个划分子图索引与机器的对应，以及索引的存储。

**Worker(工作节点)：**实时增量图处理模型的具体**实施者**，主要负责划分子图的计算，本地索引的建立以及索引和数据的存储等。

此结构设计的基本思想是，通过Coordinator将图更新的任务分解，根据划分子图的规则，将图处理的任务分发到多个Worker，并监控其完成的进度和产生的数据。



**3.2.1 控制节点**

Coordinator由API与Graph Manager两部分组成。其中Query API和Update API是IncGraph对外访问的接口，负责接收来自外部的数据更新请求和数据查询请求，并返回相应的结果。

Graph Manager是Coordinator的核心组件。其中的Partition Indexer主要完成两方面的功能：（Partition Indexer a.划分：将原来的完整的图划分成若干个子图，并且将子图的具体数据发送到对应的Worker上，子图与worker的对应关系作为全局索引存储在Partition Store中；b.转发：将对应节点的请求转发到对应的worker上计算）

（1）在初始化状态中，当一个未处理的完整图提交到系统后，Partition Indexer会根据Worker的数目和图本身的拓扑结构将其划分成均匀的子图，子图所包含的具体数据将分发并存储到相应的Worker上，等待新节点的加入或者更新操作的触发，而子图（包括其中每个节点）与Worker的对应关系，作为全局索引存储在Partition Store中。

（2）在工作状态中，当系统接收节点更新/查询请求后，Partition Indexer找到需要计算的节点数据存放位置，即相应Worker的位置，并将请求转发到相应 Worker上进行服务。

需要指出的是，Partition Indexer建立索引和查询索引的操作都是在内存中进行的，后台会有单独的异步线程将更新的索引定时写到磁盘上，防止内存突然失效时数据的丢失。相应的，在系统开始工作之 前，Partition Indexer也会将索引从Partition Store中读入内存。Graph Manager中的Partition Store主要负责划分子图索引的持久化存储，其中包括定时从内存中读取 索引并写入磁盘的异步程序，以及一套本地文件系统/ 数据库的读写接口。

**3.2.2 工作节点**

Worker 由 Partition Processor 和 Storage Manager 两部分组成。

Partition Processor是Worker的核心组件。其中 的Core Processor主要完成两方面的功能： （执行计算，并且在计算过程中可能转发计算）

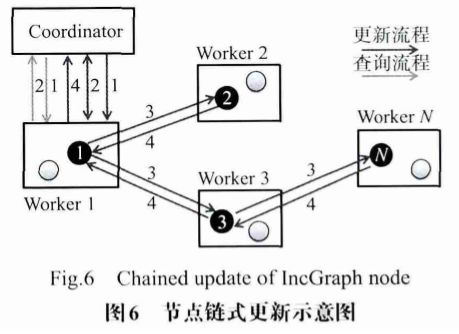
（1）在接收到Coordinator进行计算的命令后， Core Processor开始在内存中运行算法模型，并将更新的索引交由Local Indexer来处理。

（2）在计算过程中，如果某一个图节点的信息没有在 Worker 本地，CoreProcessor 向 Coordinator 的 Graph Manager提交索引查询请求，在索引返回后， CoreProcessor将其上的计算任务转发到相应的Worker上并等待其返回结果，等所有节点都处理完成，向 Coordinator汇报结果（详细流程见2.3节）。

Partition Processor 中的 Local Indexer 主要负责索引的更新，以及定期将内存中的索引交于Storage Manager写入磁盘作持久化存储。 Storage Manager主要负责存储更新后的划分子 图的数据以及相应索引。类似Partition Store，其中也包括定期从内存中异步读取索引并写入磁盘的异步程序，以及本地文件系统/数据库（此系统中用的是 HBase）的一套读写接口。

**3.3 更新和查询协议**

IncGraph基本的工作模式为更新数据和查询数据，相应的工作流程也主要介绍这两方面。



（注3：本文是以节点为中心的，计算和通信的单位都是节点。然而并不是所有的算法都是节点为中心的计算模式-局限性）

**3.4 核心技术**

本节主要介绍IncGraph系统设计中采用的几个 关键技术，分别是图切分技术、主动计算触发技术和 反向链式更新技术。

**3.4.1 图切分**

图切分技术主要目的有两个：一是为了保证各个机器的负载尽量均衡，这里的负载主要包括计算 负载和存储负载。二是为了尽量保证图数据的局部特性，即将相关的紧密的节点存储在同一台机器上，从而可以让机器之间的切边（跨机器边）尽可能少，以减少跨机器的通信代价。采用的是**METIS**

在IncGraph中，图切分还有一个重要的功能是维护动态图的切分管理，比如节点或边的加入或删除。这里使用两个技术动态地维护图划分：一是贪心选择策略，即当有新的节点加入时，贪心地选择与该节点交集最大的机器，将该节点划入那台机器即可，其他的删除操作可以直接处理；第二是定期重新划分，长期的图更新操作会使得图的划分不均衡，造成机器负载不均，从而影响性能。因此，IncGraph会 定期使用METIS重新对图数据进行切分，从而保证 各个机器的负载均衡。（注4：这里会定期的重划分，重划分的代价是非常大的，这里能否结合上面两个因素，减少重划分的次数呢？）

**3.4.2主动计算触发**

在IncGraph中 实现了主动计算触发技术，保证只有消息到达时，才 触发计算，当计算完成后，将消息发送给其他节点， 并进入休眠守护状态，直到等待下次消息的到来。

**3.4.3 反向链式更新**

反向链式更新是为了实现系统增量处理功能。图中的每条边，正向传播的是计算请求（例如，寻找需要计算的节点，并将计算转发），反向传播的是更新结果。Worker只有在后续节点计算成功返回时，本地索引才会更新。

**4. 实验评测**

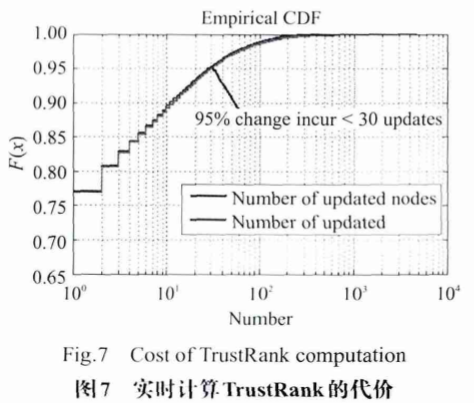
**数据集：**人人网的北京大学子图，共约20万个节点，500万条申请记录（边）。

**算法**： TrustRank.

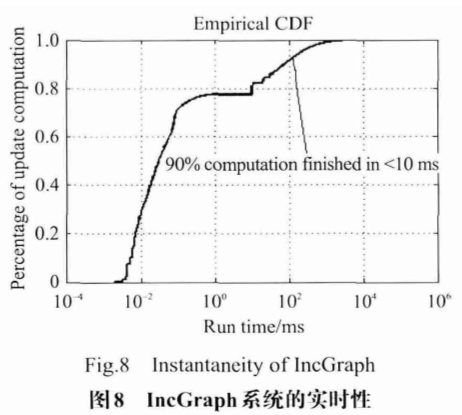
**设备**：实验在云计算平台上完成，共使用50台虚拟节 点和1台中心节点，所有机器均在同一集群内，机器 间以高速设备互相连接。

评价指标：

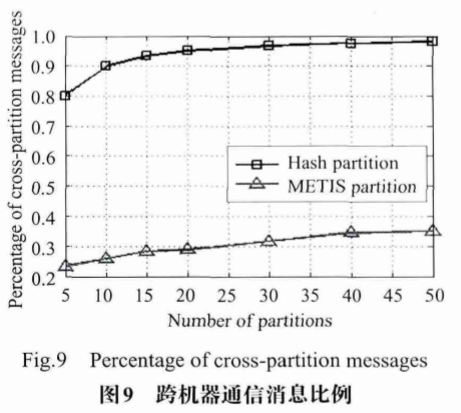
a. 实时更新代价;



b. 实时性



c. 扩展性



**5. 相关工作**

**BSP模型：**Pregel及其改进。=> 离线批处理。

**实时计算系统：**Storm等。 => 只适用于流式数据，对于基于图数据的一些复杂图算法，该 类型的实时系统就无法处理。

**单机图处理系统**：扩展性……

**总结：**

已经提出的增量图计算模型，其算法适用范围受限，而且都是基于串行执行增量更新，当图结构变化比较迅速时，往往结果的实时性不够高．

# P2 并发更新的图计算模型

SpecGraph：基于并发更新的分布式实时图计算模型

提出了一种新的基于并发更新的图计算模型SpacGraph，它通过**解耦合的计算模型**、**异步执行引擎**和**基于推测执行的并发更新机制**，达到更广的算法适用性和更高的实时性要求。

然而，随着广告、社交网络等互联网服务的高速发展，人们对于图计算系统有了更高的要求：

１）基于运算的**图数据大多并不是静态的**，而是动态变化的，例如网站新建了链接，用户取消了关注，这时图结构已经发生了改变；

２）朋友推荐、广告投放、热点检测等服务都**需要应用最新的数据来计算**．

基于以上两方面，人们对图计算系统的需求变为能够在图结构不断变化的情况下，持续给出反映最新数据的算法结果.

为了能够保证实时性， SpacGraph采用了以下3个关键技术：

１）**解耦合的计算模型**．在**面向顶点的编程模型**中， SpacGraph规定顶点的状态只依赖于邻居顶点发来的消息，而需要发送的消息只与顶点状态有关．这一限定使得实现并发更新的实现变得简单，同时又没有降低模型的表达能力．

２）**异步计算引擎**．在增量更新中， SpacGraph 采用异步计算引擎，使增量计算只需局部资源，并且能够在不同顶点执行不同层面的任务，从而为并发更新提供了可能性．

３）**基于推测执行的并发更新**．利用异步计算引擎， SpacGraph采用推测执行，并发进行增量更新， 一致性视图和回退机制保证了结果的正确性，通过最大限度地提高并发收益来提高系统计算结果的实时性．在实际应用中，并发更新对于用户是透明的，用户只需实现一般增量计算的接口函数即可，并发更新完全是由系统根据图结构的变化自动完成的．

**1. 模型及API**

**1.1 顶点环境闭包**

SpacGraph采用**基于图顶点的计算模型**，以顶点为中心进行编程。，每个顶点并行的执行由用户自定义的程序将计算相关的信息保存在节点上，并通过消息的方式进行通信．系统为顶点在计算的过程中维护一个**环境闭包**，包括与顶点相关的图结构、与顶点关联的计算信 息以及其他顶点发给该顶点的最新消息．算法的运行过程就是各个顶点闭包更新、通信的过程。

定义１．环境闭包 closure(v)．顶点ｖ的环境闭包 closure(v)，是一个三元组(G(v)， status，message\_map)．

G 是与顶点 v 相关的图结构信息，包括 节点 v 本身及与 v 相连的所有边e；

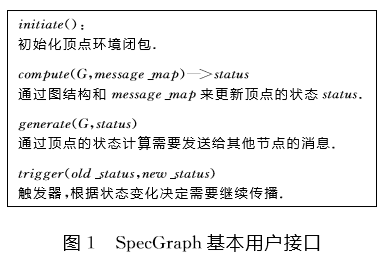
状态 status 是与该顶点相关的具体计算数据，随着计算过程不断进行更新；

message\_map 是一个从顶点到消息的映射 ｛ vertex\_id: last\_message｝，记录其他顶点最后一次 发给该顶点的消息．

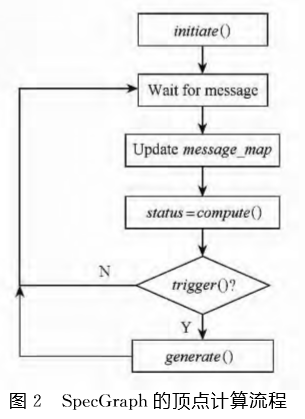
环境闭包简记为CLS(V) ＝（ Ｇ，Ｓ，Ｍ ）．

**1.2 基本计算接口**

用户通过实现这些接口来定义自己的算法，接口都是以顶点为单位．



　SpacGraph的顶点计算流程如图２所示，顶点v在开始计算后首先进行初始化，然后进入消息监听状态，当收到消息时，便按照用户实现的接口函数进行计算，并发送消息，进行迭代．



ＳｐｅｃＧｒａｐｈ提供的模型与传统的图计算系统有一定区别，传统图计算系统将 ｃｏｍｐｕｔｅ 和 ｇｅｎｅｒａｔｅ 接口合到一起，只提供一个 ｃｏｍｐｕｔｅ （ Ｇ

， ｏｌｄ ＿ ｓｔａｔｕｓ，ｍｅｓｓａｇｅ ＿ ｍａｐ ）接口，**用户利用旧的状态和最新接收到的消息计算出顶点的新状态，并发送相应的消息**，而不关心相应的依赖关系．**ＳｐｅｃＧｒａｐｈ提供的这种模型要求计算能够进一步解耦合，即顶点的状态ｓｔａｔｕｓ 只依赖于最新接收到的消息，而与旧状态无关（注1：解耦合模型的局限性）；顶点需要发送的消息则只依赖于顶点当前的状态，而与接收到的消息无关．**用符号表示就是：

Ｓ ＝ ｃｏｍｐｕｔｅ （ Ｇ， Ｍ）；

Ｍ ＝ ｇｅｎｅｒａｔｅ （ Ｇ，Ｓ）．

**1.3 增量更新**

为了能够实现在动态图结构上的实时计算， ＳｐｅｃＧｒａｐｈ采用增量更新的模型，利用图结构的变化信息，在之前的计算结果基础上进行增量计算并进行传播，直到算法收敛．

为了实现增量更新，用户需要实现**增量计算接口**，按照顶点（边）的增减分别实现相应的增量算法，发送一些调整消息通知相应的顶点重新进行计算，然后系统进入一般计算状态，即各顶点在接收到消息后，根据自身的消息映射 Ｍ 更新顶点的状态 Ｓ，并通过 ｔｒｉｇｇｅｒ决定是否生成并发送更新消息，直到计算收敛．

**1.4 应用实例**

ＳｐｅｃＧｒａｐｈ的**解耦合模型（**即status新值与历史值无关**）**为系统的增量更新及并发更新提供了很大的便利性和可操作性，然而**这种模型也使得系统的适用性有了一定局限性，即算法必须能够表达成这种解耦合的模型**．根据我们的实验，常见的图算法以及统计信息都能够在该模型下实现，所以这种解耦合的模型并没有显著降低模型的表达能力．以下是３种图中常用算法在 ＳｐｅｃＧｒａｐｈ接口下的实现示例，算法分别是网页排名ＰａｇｅＲａｎｋ、单源最短路长度ＳＳＳＰ（ｓｉｎｇｌｅ ｓｏｕｒｃｅ ｓｈｏｒｔｅｓｔ ｐａｔｈ）［ ５］以及求弱连通分量划分 ＷＣＣ （ ｗｅａｋｌｙ　ｃｏｎｎｅｃｔｅｄ　ｃｏｍｐｏｎｅｎｔ） ［ ６］

**1.5 并发更新**

并发更新是 ＳｐｅｃＧｒａｐｈ和其他增量图计算系统的**最大区别**，也是ＳｐｅｃＧｒａｐｈ能够提高实时性的关键技术，然而这一特性在应用层是完全透明的，ＳｐｅｃＧｒａｐｈ能够实现对用户透明的并发更新，得益于之前设计的**解耦合计算模型**．

**2. 基于推测执行的并发更新机制**

**2.1异步执行引擎**

**同步执行引擎（BSP模型）：**在传统图计算系统中，大多采用**整体同步并行计算模型**（ｂｕｌ ｋ　 ｓｙｎｃｈｒｏｎｏｕｓ　 ｐａｒａｌｌｅｌ，ＢＳＰ）［７］的同步计算引擎，即在计算过程中有**超步**（Ｓｕｐｅｒｓｔｅｐ）这一概念，各个顶点在更新状态并发送消息完成后，要等待其他所有顶点也完成这一Ｓｕｐｅｒｓｔｅｐ才能进行下一轮计算。

**异步执行引擎**：而异步执行引擎是只要当顶点收到了消息就进行状态更新并发送消息，而不管其他顶点状态，异步执行引擎的流程如图2所示。

**2.2 并发更新的动机和实现原理**

**（**在图很大的情况下，两个顶点冲突的概率很小，而且顶点是无状态的，所以回退代价也很小,因此并发更新带来的收益很大**）**

每次更新结果必须基于一致性视图，即当我们在执行增量更新任务 Ｔ０的时候，如果 Ｔ１时刻图结构发生了变化，那么不能将这个变化应用到Ｔ０中，否则得到错误的计算结果；即使通过算法设计使计算不会出错，等到系统计算完新的变化而无法及时更新计算结果，使得之前非实时的结果保持更长时间，所以这时需要重新提交一个新的增量任务Ｔ１．

基于以上分析，Ｔ１的计算依赖于Ｔ０的计算结果，那是否一定要在Ｔ０更新完成后才能计算Ｔ１呢，答案是否定的．事实上，Ｔ１对Ｔ０的依赖只限于Ｔ０更新时涉及到的顶点，而对于其他顶点Ｔ１依赖的是更早之前的计算结果，所以Ｔ０没有涉及到的顶点Ｔ１是可以不用等待直接计算的．

但是在没有执行完 Ｔ０和 Ｔ１之前，我们是无法知道它们各自涉及到的顶点，而且这种涉及范围无法预测，因为它跟具体算法相关．采用**推测执行**的方法来实现并发：在Ｔ０执行时，尝试并发的执行Ｔ１， 对于Ｔ１涉及的顶点ｖ，如果已经被Ｔ０涉及到，Ｔ１需要在其上等待Ｔ０的完成；如果Ｔ０尚未涉及ｖ， 可以执行Ｔ１的更新；如果在之后Ｔ０更新涉及到ｖ，那么需要在ｖ上对Ｔ１进行**回退**，待执行完Ｔ０后重新执行Ｔ１．

由于增量更新涉及的顶点范围相较整个图结构很小，所以２次更新之间顶点冲突的概率很小，只要保证冲突处理的代价较小，就可以保证并发更新带来实时性的收益．

**ＳｐｅｃＧｒａｐｈ的解耦合模型能够以较小的代价实现这种回退机制**，模型中顶点的状态只依赖于其最新的 ｍｅｓｓａｇｅ ＿ ｍａｐ ，所以只要在更新的过程中保证面向各个视图的顶点 ｍｅｓｓａｇｅ ＿ ｍａｐ正确，就能保证运算结果的正确性，这便是ＳｐｅｃＧｒａｐｈ并发机制实现的基本原理．

**2.3基于推测执行的并发机制设计**

通过版本方式来控制更新。具体步骤没有读懂。

**3. 系统架构**

讲的比较简单，没有太大的参考价值。

# P3 增量计算模型 KineoGraph

# P4 图内存计算总结

基于内存计算的大规模图数据管理研究

**摘要**

文章综述了大规模图数据处理相关技术进展，研究了典型的**基于内存计算**的大规模图数据管理系统，最后总结了基于内存计算的图数据管理的关键点．

**0 引言**

大规模图数据在分布式集群上处理涉及的关键技术：

（1）**图数据划分**：图的划分是图数据管理的关键步骤，不仅与数据存储与数据均衡、负载均衡有关，还与计算节点间通信与数据移动量有关；

（2）**计算模型**：关系到计算表达能力和粒度。

本文在**大规模图数据管理需求**和**内存计算**大发展背景下，研究了大规模图数据并行计 算的**编程模式**、**计算策略**、**图划分策略**及**计算同步**等问题，接着介绍了内存计算相关的概念、 设计理念和产品．主要介绍了基于内存计算机制的图数据管理进展和典型的系统，总结了**基于内存计算的大规模图处理**的关键。

**1. 图数据管理**

**1.1 图数据表示**

**G = (V,E,P)** 顶点，边，属性。

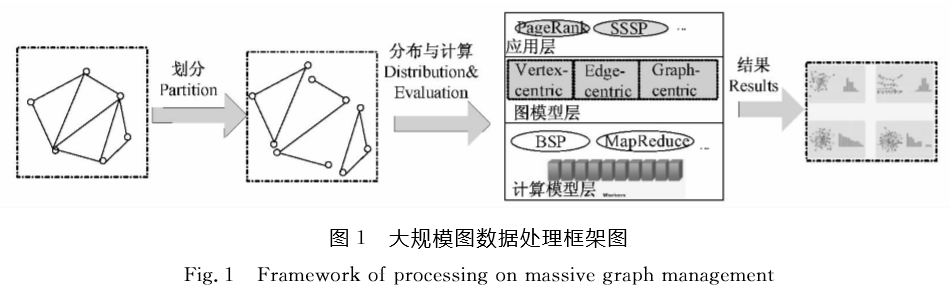
**1.2 计算框架**

图数据的分布式并行计算一般分为三个步骤：

① 图划分，

② 图数据分布与计算，

③ 最终结果产生．



对于海量的图数据，首先是对图根据一定的规则进行**划分**，把图数据划分为若干个不相 交的部分，进而把数据分布到集群上的计算节点．典型的**图分割算法**包括：基于边的切割 （ ｐ －ｗａｙ　ｅｄｇｅ－ ｃｕｔ）、基于顶点的切割（ ｐ －ｗａｙ　ｖｅｒｔｅｘ

－ ｃｕｔ）、基于随机的哈希方法和启发式的划分等．图的划分质量对工作负载均衡、计算节点通信、存储和计算效率都有着极大影响．

第二步，将划分后的数据分布到计算节点上进行**分布式存储**和**并行计算处理**．该步骤是**图计算的核心**，从逻辑上可以划分为三层：最上层的应用、中间的模型、底层物理层．

应用层的应用从实时性可以划分为**在线查询**和**离线分析**．在线查询实时查询，一般无 法预测查询与图结构的关系，而离线分析数据访问模式是可以预测．从计算方法分三大类 图计算：① 图的遍历，例如最短路径、连通分量计算等；② 随机行走，例如 ＰａｇｅＲａｎｋ［１３］， ＨＩＴＳ［１４］等；③ 图聚集计算，例如图概要［１５］、图粗化［１６］等．

中间层是**图计算模型层**，该层次是图的计算策略，包括三种策略：① 以顶点为中心的策 略（Ｖｅｒｔｅｘ－ ｃｅｎｔｒｉｃ）；② 以边为中心的策略（Ｅｄｇｅ－ ｃｅｎｔｒｉｃ）；③ 以图为中心的策略（Ｇｒａｐｈ－ ｃｅｎｔｒｉｃ）．

底层为**物理层**，包括**计算同步策略**，**数据存储方式**、**编程框架**和**容错机制**等．同步策略 包括ＢＳＰ（Ｂｕｌｋ　 Ｓｙｎｃｈｒｏｎｏｕｓ　Ｐａｒａｌｌｅｌ）同步［ ４］、异步［１７］和异步混合模式［ ３，１７］等．存储方法包括分布式文件、ｋｅｙ －ｖａｌｕｅ方式存储、数组、ＢｉｇＴａｂｌｅ等． 最后是计算最终分析结果．

**2. 图的计算**

**2.1 图的编程模式**

图的分布式并行计算分两点：

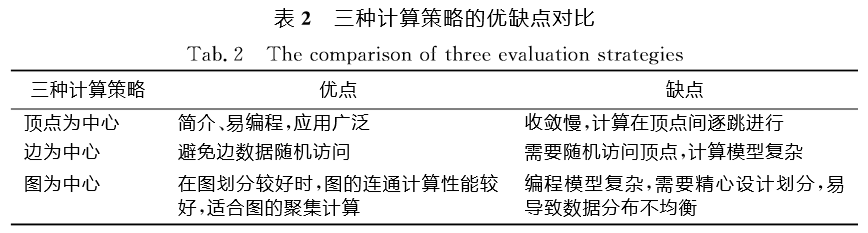
（1）在顶点或边上迭代计算，并更新状态，直到计算状态收敛到一个固定点；

（2）计算的每次迭代可以在图的顶点或边的层面并行独立进行。

可以根据计算中间结果扩展和聚集方法的不同可以把编程 模式分为**SG**和**SAG**两种策略。

**2.2 图的计算策略**

图数据的计算包括两部分：图的顶点计算和图的边计算．根据**对图的处理视角的不同**， 可以把计算模型划分为三种：**以顶点为中心**、**以边为中心**的和**以图为中心**．三种计算模型的 表达能力和计算能力不同，各有优缺点。

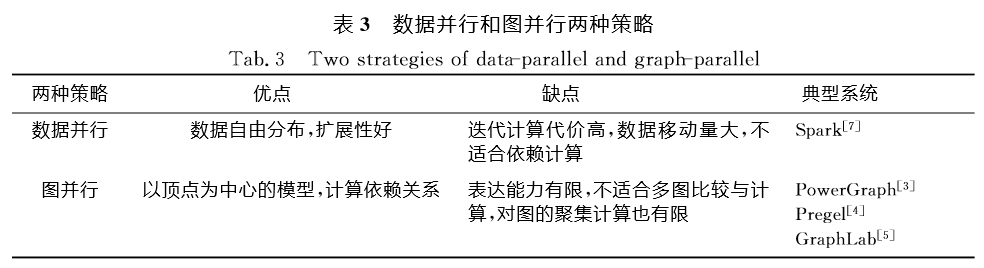


**2.3 图的并行策略**

集群平台的图计算的典型**并行策略**有两种：一种不考虑数据依赖关系，称为**数据并行**（ Ｄａｔａ －ｐａｒａｌｌｅｌ）；另一种是考虑图元素之间的依赖关系，根据依赖关系迭代计算和通信，称为**图并行**（Ｇｒａｐｈ －ｐａｒａｌｌｅｌ）．

数据并行是把图数据作为相互独立的部分并行处理，通过把图数据划分为独立的部分， 分布到集群各节点，在不同的分布节点上独立计算．典型的为Ｓｐａｒｋ［ ７］，该类系统的优点是在分布节点上对数据在计算节点间的移动不做限制，扩展性好．但是由于图分析迭代计算的性质，对数据并行系统提出了挑战，例如图结构、数据Ｊｏｉｎ导致数据移动代价较高．典型的操作为 ｍａｐ，ｒｅｄｕｃｅ，ｆｉｌｔｅｒ， ｊ ｏｉｎ等．

图并行策略是把根据图分割算法把图划分为具有依赖关系的部分，在具有依赖关系的 数据上进行迭代并行计算，依赖部分的计算是通过在邻居节点迭代以及节点间的通信完成 的．在该并行策略下，典型的计算策略是以顶点为中心的计算．该方法大幅提升了图并行处理的性能．典型的系统为Ｐｏｗｅｒ　Ｇｒａｐｈ［ ３－ ５］．但是该类型系统的计算表达能力不强，例如图构造、图结构更新、多图合并计算等．两种并行策略的对比如表３所示． 以上两种并行策略各有优缺点，通过两者的结合，**在图数据加载阶段采用数据并行，而在图数据分析阶段采用图并行**，利用二者的优点．典型的系统为ＧｒａｐｈＸ［ ６］．



**2.4 图划分**

图划分是大规模图数据计算的重要操作，典型的图分割算法包括随机划分、基于边的平 衡划分、基于顶点的平衡划分和启发式划分［ ６］、流划分、谱划分（ Ｓｐｅｃｔｒａｌ 　 Ｐａｒｔｉｔｉｏｎｉｎｇ） ［ ３５］ 等．文献［１９］研究了经典的图划分方法．

图划分的质量对在**工作负载均衡**、**计算节点通信**、**存储**和**计算效率**等都有极大影响．优 化划分的原则是：减少边跨越划分的个数，减少计算节点之间的通信，加快计算收敛速度． 图分割的难点在于真实世界的图一般符合幂率分布，这对分布式的工作平衡带来挑战，使得 图难于均匀分割［ ３］；第二，基于 Ｈａｓｈ的图节点划分技术导致数据局部性非常差；第三，不平 衡数据的分布导致计算节点间大量的通信开销；第四，度数高的节点对计算和存储的扩展性 带来挑战．此外，过于复杂的分割带来的计算开销也是必须要考虑的一个重要因素，基于哈 希的分割简单易于实现，代价较小，应用也比较广泛［

（评1：图的划分也是一个重要的工作，可以深入研究，例如在动态图场景下如何划分？）

**2.5 图处理的同步策略**

由于图结构依赖性导致其计算往往需要多次迭代，复杂性的结构使得达到稳定点计算 步骤不同，需要在计算步之间进行控制．常见的同步方式有：**同步计算**、**异步计算**和**混合方 式**．其中**ＢＳＰ**（Ｂｕｌｋ　 Ｓｙｎｃｈｒｏｎｏｕｓ　Ｐａｒａｌｌｅｌ） ［ ２０］是最常用的同步模型．

**2.6 图处理的容错机制**

集群上的图数据计算需要稳定可靠的处理环境，系统容错至关重要，尤其是内存计算环 境下内存数据的易失性．典型的容错机制包括：分布式检测点机制［ ３－ ５］；分布式文件备份［２７］ 等．

**3. 内存计算**

内存计算兴起的过程，原因，产品等。

**4. 基于内存计算的图数据处理方案**

**图计算是一种I/O密集型运算,**大部分的应用计算需要多次迭代，计算的状态信息需要在计算节点间消息传递和频繁更新，尤其是大规模的图数据，需要在集群的节点间频繁的消息传递和中间结果存储。**图计算是一种有依赖关系的运算。**文献[４，３７］指出，现有的 ＭａｐＲｅｄｕｃｅ模式不适合大规模图数据处理，原因一是 ＭａｐＲｅ－ ｄｕｃｅ的优势在于**并行处理无依赖关系的计算**，对在有依赖关系的图数据很难大规模并行； 二是 ＭａｐＲｅｄｕｃｅ共享数据的唯一方式是把数据写到分布式文件中，增加了数据复制带来的 Ｉ ／Ｏ。

为了提升大规模图数据计算的效率，研究者提出了基于内存的图处理系统［ ７，１７，２７］．图的内存计算系统大致可以分为三种：第一种是**基于内存分布式集群系统**，例如Ｔｒｉｎｉｔ ｙ［ ２７］系 统；第二种是**基于内存共享的分布式系统**，例如文献［ ５－ ７］所介绍的；第三种是在**多核单机上多线程共享大内存系统**，例如ＧＲＡＣＥ［１７，３５－ ３７］．下面介绍几个典型的基于内存计算的图处理系统．