这篇总结的还不错

[大数据图数据库之离线挖掘计算模型](http://blog.csdn.net/malefactor/article/details/39897705)

标签： [大数据](http://www.csdn.net/tag/%e5%a4%a7%e6%95%b0%e6%8d%ae)[系统架构](http://www.csdn.net/tag/%e7%b3%bb%e7%bb%9f%e6%9e%b6%e6%9e%84)[图数据库](http://www.csdn.net/tag/%e5%9b%be%e6%95%b0%e6%8d%ae%e5%ba%93)

2014-10-08 19:51 4813人阅读 [评论](http://blog.csdn.net/malefactor/article/details/39897705#comments)(1) 收藏 [举报](http://blog.csdn.net/malefactor/article/details/39897705#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

大数据（6） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

/\* 版权声明：可以任意转载，转载时请务必标明文章原始出处和作者信息 .\*/  
            author: 张俊林

**节选自《[大数据日知录：架构与算法](http://item.jd.com/11540991.html" \t "_blank)》十四章，书籍**[**目录在此**](http://blog.csdn.net/malefactor/article/details/39312929)

     对于离线挖掘类图计算而言，目前已经涌现出众多各方面表现优秀而各具特点的实际系统，典型的比如Pregel、Giraph、Hama、PowerGraph、GraphLab、GraphChi等。通过对这些系统的分析，我们可以归纳出离线挖掘类图计算中一些常见的计算模型。

    本节将常见的计算模型分为两类，一类是图编程模型，另一类是图计算范型。编程模型更多地面向图计算系统的应用开发者，而计算范型则是图计算系统开发者需要关心的问题。在本节中，关于编程模型，主要介绍以节点为中心的编程模型及其改进版本的GAS编程模型；关于计算范型，则重点介绍同步执行模型和异步执行模型。这几类模型已经被广泛采用在目前的大规模图挖掘系统中。

**14.4.1  以节点为中心的编程模型**

     以节点为中心的编程模型（Vertex-Centered ProgrammingModel）首先由Pregel系统提出，之后的绝大多数离线挖掘类大规模图计算系统都采用这个模型作为编程模型。

     对图*G*=(*V*,*E*)来说，以节点为中心的编程模型将图节点*vertex*Î*V*看作计算的中心，应用开发者可以自定义一个与具体应用密切相关的节点更新函数*Function(vertex)*，这个函数可以获取并改变图节点*vertex*及与其有关联的边的权值，甚至可以通过增加和删除边来更改图结构。对于所有图中的节点都执行节点更新函数*Function(vertex)*来对图的状态（包括节点信息和边信息）进行转换，如此反复迭代进行，直到达到一定的停止标准为止。

     典型的图节点更新函数*Function(vertex)*基本遵循如下逻辑。



      即首先从*vertex*的入边和出边收集信息，对这些信息经过针对节点权值的函数*f*()变换后，将计算得到的值更新*vertex*的权值，之后以节点的新权值和边原先的权值作为输入，通过针对边的函数*g*()进行变换，变换后的值用来依次更新边的权值。通过*vertex*的节点更新函数，来达到更新部分图状态的目的。

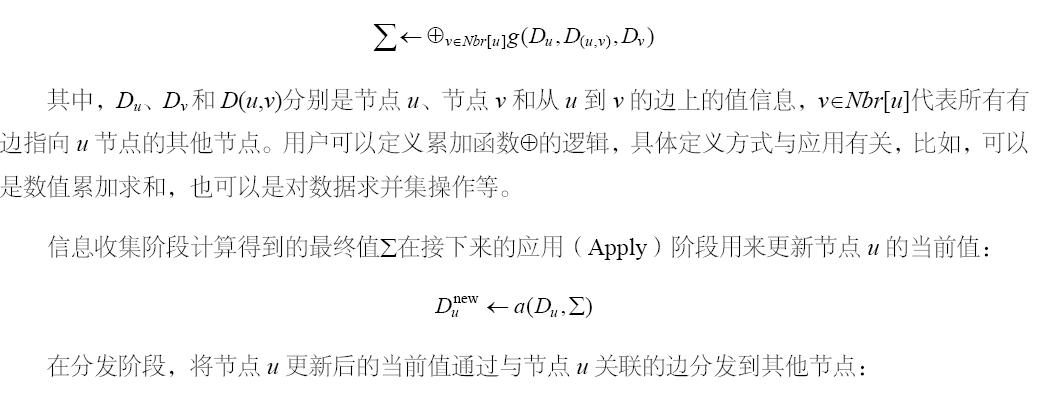
     以节点为中心的编程模型有很强的表达能力。研究表明，很多类型的问题都可以通过这个编程模型来进行表达，比如很多图挖掘、数据挖掘、**[机器学习](http://lib.csdn.net/base/2" \o "机器学习知识库" \t "_blank)**甚至是线性代数的问题都可以以这种编程模型来获得解决。这也是为何以图节点为中心的编程模型大行其道的根本原因。

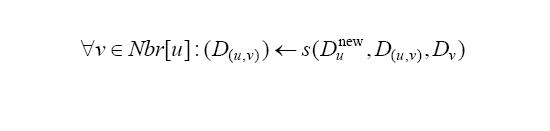
**14.4.2  GAS编程模型**

      GAS模型可以看作是对以节点为中心的图计算编程模型的一种细粒度改造，通过将计算过程进一步细分来增加计算并发性。GAS模型明确地将以节点为中心的图计算模型的节点更新函数*Function(Vertex)*划分为三个连续的处理阶段：信息收集阶段（Gather）、应用阶段（Apply）和分发阶段（Scatter）。通过这种明确的计算阶段划分，可以使原先的一个完整计算流程细分，这样在计算过程中可以将各个子处理阶段并发执行来进一步增加系统的并发处理性能。

     这里假设当前要进行计算的节点是*u*，并以此为基础来说明GAS模型。

      在信息收集阶段，将*u*节点的所有邻接节点和相连的边上的信息通过一个通用累加函数收集起来：





      通过以上三个阶段的操作，可以定义以图节点为中心的高度抽象的GAS计算模型。在GAS模型中，节点的入边和出边在信息收集和分发阶段如何使用取决于具体的应用，比如，在PageRank计算中，信息收集阶段只考虑入边信息，分发阶段只考虑出边信息，但是在类似于Facebook的社交关系图中，如果边表达的语义是朋友关系，那么在信息收集和分发阶段则是所有边的信息都会纳入计算范围。

**14.4.3  同步执行模型**

      同步执行模型是相对于异步执行模型而言的。我们知道，图计算往往需要经过多轮迭代过程，在以节点为中心的图编程模型下，在每轮迭代过程中对图节点会调用用户自定义函数*Function(vertex)*，这个函数会更改*vertex*节点及其对应边的状态，如果节点的这种状态变化在本轮迭代过程中就可以被其他节点看到并使用，也就是说变化立即可见，那么这种模式被称为异步执行模型；如果所有的状态变化只有等到下一轮迭代才可见并允许使用，那么这种模式被称为同步执行模型。采用同步执行模型的系统在迭代过程中或者连续两轮迭代过程之间往往存在一个同步点，同步点的目的在于保证每个节点都已经接受到本轮迭代更新后的状态信息，以保证可以进入下一轮的迭代过程。

      在实际的系统中，两种典型的同步执行模型包括BSP模型和MapReduce模型。关于BSP模型的介绍及其与MapReduce模型的关系，可以参考本书“机器学习：范型与架构”一章，这里不再赘述。下面介绍图计算中的MapReduce计算模型，总体而言，由于很多图挖掘算法带有迭代运行的特点，MapReduce计算模型并不是十分适合解决此类问题的较佳答案，但是由于Hadoop的广泛流行，实际工作中还有一些图计算是采用MapReduce机制来进行的。

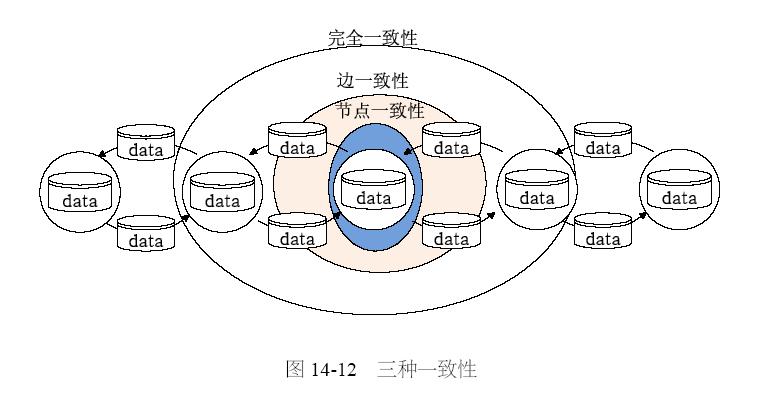
**14.4.4  异步执行模型**

      异步执行模型相对于同步执行模型而言，因为不需要进行数据同步，而且更新的数据能够在本轮迭代即可被使用，所以算法收敛速度快，系统吞吐量和执行效率都要明显高于同步模型。但是异步模型也有相应的缺点：其很难推断程序的正确性。因为其数据更新立即生效，所以节点的不同执行顺序很可能会导致不同的运行结果，尤其是对图节点并发更新计算的时候，还可能产生争用状况（Race Condition）和数据不一致的问题，所以其在系统实现的时候必须考虑如何避免这些问题，系统实现机制较同步模型复杂。

     下面以GraphLab为例讲解异步执行模型的数据一致性问题，GraphLab比较适合应用于机器学习领域的非自然图计算情形，比如马尔科夫随机场（MRF）、随机梯度下降算法（SGD）等机器学习算法。

在讲解异步模型的数据一致性问题前，先来了解一下GraphLab论文提出的图节点的作用域（Scope）概念。对于图G中的某个节点*v*来说，其作用域*Sv*包括：节点*v*本身、与节点*v*关联的所有边，以及节点*v*的所有邻接图节点。之所以定义图节点的作用域，是因为在以节点为中心的编程模型中，作用域体现了节点更新函数*f*(*v*)能够涉及的图对象范围及与其绑定的数据。

     在并发的异步执行模型下，可以定义三类不同强度的数据一致性条件（见图14-12），根据其一致性限制条件的强度，由强到弱分别为：完全一致性（Full Consistency）、边一致性（Edge Consistency）和节点一致性（Vertex Consistency）。



      完全一致性的含义是：在节点*v*的节点更新函数*f(v)*执行期间，保证不会有其他更新函数去读写或者更改节点*v*的作用域*Sv*内图对象的数据。因此，满足完全一致性条件的情形下，并行计算只允许出现在无公共邻接点的图节点之间，因为如果两个图节点有公共邻接图节点，那么两者的作用域必有交集，若两者并发执行，可能会发生争用状况，而这违反了完全一致性的定义。

     比完全一致性稍弱些的是边一致性条件，其含义为：在节点*v*的节点更新函数*f*(*v*)执行期间，保证不会有其他更新函数去读写或者更改节点*v*，以及与其邻接的所有边的数据。即与完全一致性条件相比，放松了条件，允许读写与节点*v*邻接的其他图节点的数据。在满足边一致性条件下，并行计算允许出现在无公共边的图节点之间，因为只要两个节点*u*和*v*不存在共享边，则一定会满足边一致性条件。

     更弱一些的是节点一致性，其含义为：在节点*v*的节点更新函数*f*(*v*)执行期间，保证不会有其他更新函数去读写或者更改节点*v*的数据。很明显，最弱的节点一致性能够允许最大程度的并发，之所以说其限制条件较弱，是因为除非应用逻辑可以保证节点更新函数*f*(*v*)只读写节点本身的数据，否则很易发生争用状况，使得程序运行结果不一致。

     选择不同的一致性模型对于并行程序执行的结果正确性有很大影响，所谓并行执行的结果正确性，可以用其和顺序执行相比是否一致来进行判断。因此，可以定义“序列一致性”如下：

     如果对所有可能的并发执行顺序总是存在与序列执行完全一致的执行结果，在此种情形下，我们可以将这个并发程序称为是满足序列一致性的。

      是否满足序列一致性可以帮助我们验证将一个顺序执行的程序改造为并行执行程序后的正确性。在并行的异步图计算环境下，以下三种情形是可以满足序列一致性的。

情形一：满足完全一致性条件。

情形二：满足边一致性条件，并且节点更新函数*f*(*v*)不会修改邻接节点的数据。

情形三：满足节点一致性条件，并且节点更新函数*f*(*v*)只会读写节点本身的数据。

     上面三种情形可供应用者在设计算法时参考，以在并发性和结果正确性之间做好权衡：一致性条件越弱，则并发能力越强，但是争用状况发生概率越高，即结果可能越难保障正确性。如果应用能够明确节点更新函数的数据涉及范围，就可以根据上述几种情形来进行选择，更好地做到在保证结果正确性的前提下提高并发性能。