GraphFlow:基于状态更新的动态图计算模型

段世凯[1]，许利杰[2]，王伟[3]，赵伟[4]
(1.单位全名 部门(系)全名,省 市(省会城市不用加省名) 邮政编码；2. 单位全名 部门(系)全名,省 市(省会城市不用加省名) 邮政编码；3.……)

摘 要:图数据结构能够很好的表达数据之间的关联性，因此在社交分析、商品推荐、舆论监测和欺诈检测等应用中被广泛使用。而目前分布式处理框架 Google Pregel、Spark GraphX 和 Flink Gelly 等都是处理静态稳定的图数据，这使得传统的图计算模型难以应对动态变化的图数据。而已经提出的增量图计算模型和基于推测机制的并发更新模型，又分别受限于其串行的更新方式和节点更新只与节点接收消息相关的强力约束，使得模型在并发性低，表达能力有限。本文提出了一种基于状态更新的动态图计算模型 GraphFlow，它将动态图处理中图数据的变化抽象成一系列的事件流，将用户关心的图计算结果抽象成图的状态，用户只需要定义图数据如何根据到达的事件增量式的进行状态的转换，就能够完成事件流到状态流的映射，提供实时反馈计算结果能力。通过对独立状态的分布式存储和并发更新策略，以及对关联状态的分区并行更新策略以及细粒度锁的更新策略，能够有效解决关联状态下更新冲突的问题，从而提高了系统的并行性。试验结果表明，GraphFlow 平均请求的相应时间控制在 5ms 内，而且 GraphFlow 的计算偏差在 1%以内。

关键词: 图处理系统；实时计算；分布式系统；状态更新；增量图计算；分区并行更新；细粒度锁
中图分类号: **** **文献标志码:** A **文章编号:**（作者可不填）
doi:10.3969/j.issn.1001-3695 (作者可不填)

GraphFlow: A Dynamic Graph Computing Model Based on State Updating

Duan Shikai[1], Xu lijie[2], Wang wei[3], Zhao Wei[4]

(1.*Dept.*（*School* or *College*）*of* ****, *University, City Province Zip Code, China*；2.*Dept.*（*School* or *College*）*of* ****, *University, City Province Zip Code, China*；3.*Dept.*（*School or College*）*of* ****, *University, City Province Zip Code, China*)

Abstract: Graph data structure can be a good expression of the relevance between the data, so applications like social analysis, commodity recommendation, public opinion monitoring and fraud detection are widely used. The existing processing frameworks and systems, such as Google Pregel, Spark GraphX and Flink Gelly, all run on static stable graph data, which makes it difficult to deal with the dynamic graph data. The incremental computation model proposed in other paper is based on serial updating, which can not make full use of the advantages of parallel updating. In addition, the concurrent updating model based on speculative mechanism can improve the parallelism of the system, but the starting point is that the state update is only related to the receiving message and has nothing to do with the original state, this constraint also makes this model expressive ability is limited. In this paper, a dynamic graph calculation model GraphFlow based on state updating is proposed, which abstracts the change of graph data in dynamic graph processing into a series of event flows, and abstracts the results of graphs concerned by users into the states of graphs. Users only need to define graphs how state can be transformed according to the current state and arrival events. By this way, the system can complete the event flow to the state flow mapping, providing real-time feedback the intermediate state of the dynamic graph computation. Through the distributed storage and concurrent update strategy of the independent state, as well as the update policy of the partition and the update strategy of the fine-grained lock, the problem of updating conflict in association state can be solved effectively, which improves the parallelism of the system .The experimental results show that the GraphFlow can calculate and feedback the results in real time compared with the traditional batch graph computing system. The average request time is within 5ms. Compared with the estimation model of dynamic data, the accuracy of GraphFlow is high, and the calculation error is less than 1%.

Key words: graph processing system; real-time computing; distributed system; state update; incremental graph processing; partition parallelism update; fine-grained lock

0 引言

图是计算机科学中常用的一类数据结构，它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彼此关联而形成的图。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]。

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；在社交网络中，通过传播关系发现意见领袖。图算法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。

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中进行处理，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

收稿日期: 2006-08-20；**修回日期:** **基金项目:** 基金项目 1 全称（基金项目号）；基金项目 2 全称（基金项目号）；……
作者简介: 作者名(出生年－),性别（民族）,****(籍贯,具体到市、县)人,职称，学位(或目前学历),主要研究方向为*****、****(第 1 作者 E-mail 地址)；作者(出生年－),性别,学位(或目前学历),职称,主要研究方向为****、****；作者名(出生年－),性别,学位(或目前学历),职称,主要研究方向为****、****.

本刊不特别标注通信作者，若必须标注，请在通信作者名后加“+”（横短竖长）上标，并在作者简介该作者性别后注明（通信作者）

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增量计算。

现有的较为成熟的图计算系统如 Google Pregel[2],Spark GraphX[3]，是建立在 BSP（Bulk Synchronized Parallel）[4]模型上的针对静态图数据的批量计算，当图数据变化时，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，也浪费了系统资源[5]。而针对图数据不断变化的情况，学术界提出了很多在动态图上直接进行计算的方法，大致可以分为两类：估计计算[6-11]和准确计算[5][13][14]。对于估计计算，大部分的算法是希望通过采样或者设计精简的数据结构的方式来降低时间和空间开销，但其估计的误差在实际的生产环境中往往变得不可控制，文献[12]指出，针对大体量的无法全部载入内存的图数据，近似算法的错误率在 95%-133%之间。对于准确计算，现有的 KineoGraph[13]和 IncGraph[5]提出采用增量计算的模型进行实时计算，然而这种增量式的更新是串行执行的，实时性有限。SpecGraph[14]虽然在上述增量模型的基础上有所改进，提出了基于推测机制的并发更新模型，然而该模型中假设顶点的状态只依赖于顶点当前接收的信息，而与顶点之前的旧状态无关。这种假设使得系统的适用性差，很多算法不仅跟顶点接收消息有关，还跟顶点的旧状态有关，因此模型的表达能力有限。

针对动态图计算的实时性和准确性要求，本文在上述已有的研究基础之上，提出了**基于状态更新的动态图计算模型**，能够在原有图状态上并发的计算增量信息对状态的影响，而无需在整个图上重新计算，同时通过控制更新影响范围来提高并发性，实现状态的并发更新。本文的工作主要有以下 3 点：

（1）分析现有的图计算的特点，抽象出在流式场景下图算法的典型特征。从影响范围和计算次数两个维度分析了 4 个典型的图算法：节点度分布（DD, Degree Distribution），三角形数目（TC, Triangle Count），单源点最短路径（SSSP, Single Source Shortest Path），和 PageRank（PR）。

（2）根据这些流式图算法的特点，建立基于状态更新的动态图计算模型，该模型能够允许用户自定义状态，并且采用并发更新的方式来快速计算结果。

（3）在上述建立的基于状态更新的动态图计算模型基础上实现了典型的动态图算法：TC 和连通子图（CC, Connected Components），并且从正确性和实时性对算法进行评估，结果表明：基于状态更新的动态图计算模型构建的算法能够得到较为正确的计算结果，计算偏差在 1%以内；而且算法能够在 5ms 内返回增量计算的结果，符合实时性要求。

1 基于状态更新的动态图计算模型

在本节中，首先分析了 DD、TC、SSSP、PR 四个典型的图算法，然后详细阐述了基于状态更新的动态图计算模型的组件以及状态的存储和更新过程，最后列举 TC 和 CC 两个算法说明如何在该模型上设计动态图算法。

1.1 图算法特征分析

（1）流式图数据

所谓流式图数据是指图的数据（包括图的顶点、图的边、图顶点的值和图边的权重）不再是静态的存储在文件或数据库中，而是以流的形式源源不断的添加到系统中。因此系统中的图是随着时间而动态变化的。常见的流图模型有两种[15]，一种是 Cash Register Model：流中的每一项仅仅是数据集中一项，通过流的方式不断的扩充数据集。另一种是 Turnstile Model：在该模型中，有一个初始化为空的集合 D，流中的数据由两项组成，一项是数据集的某一项，另一项是一个标志位，可以对集合 D 进行动态改变。例如，流图中的每一项为(x, U)，如果 U 为＋，就将 x 加入 D，如果 U 为－，就将 x 从 D 删除。

如下图所示，在一个管道中，图的每条边按照一定顺序流入系统中，其中+表示增加一条边，-表示删除一条边，对应这些边的变化，图的结构和状态也在不断变化。在本文中，本文考虑的是边的 Turnstile Model，即图数据流是按照边的添加和删除来进行组织的。

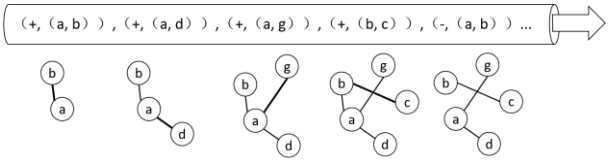
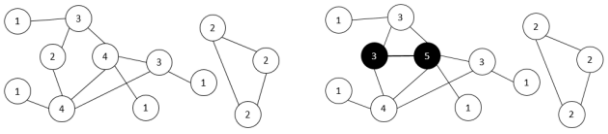


图 1 流式图数据

相比较传统的静态图数据，流式图数据的图算法有很大不同。下面选取了 DD、TC，SSSP 和 PR 这四种典型的图算法来分析在边不断添加的情况下各个算法的特点。

（2）Degree Distribution

DD 算法是统计无向图中各个节点的度。如图 2 展示了在流式场景下如何统计节点的度。图 2 中的每个圆圈表示一个节点，圆圈之间的连线表示一条边，圆圈内部的数字表示当前时刻该节点的度。在某一时刻节点的度分布情况如图 2(a)所示，当增加一条边（图 2(b)中标黑的两个节点之间增加一条边）后，这条边对应的源顶点和目标顶点的度分别增加 1。增加该条边所带来的影响如图 2(b)所示。由此可见，在 DD 算法中，增加的一条边只影响了这条边的源点和目标点。

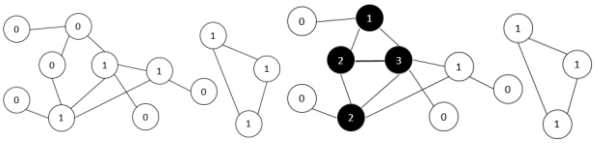


（a） （b）

图 2 节点度分布

（3）Triangle Count

TC 算法是用来统计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。图 3 展示了在流式场景下如何统计三角形的数目。图 3 中节点内的数字表示该节点所拥有的三角形的数目。图 3(a)表示在某一时刻三角形的分布情况，当在三角形数目为 0 和 1 的节点之间增加一条边时，它会使得这两个节点的所有公共邻接点的三角形的数目都增加 1，而这两个节点的三角形数目增加 N,N 为公共邻接点的数目。由此可见，对于 TC 算法，增加的一条边不仅影响了这条边的两个顶点，还影响了这两个顶点的公共邻接点。



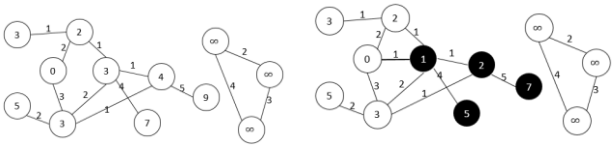
（a） （b）

图 3 三角形数目分布

（4）Single Source Shortest Path

SSSP 算法是在有向图或无向图中，给定一个源点，求解这个源点到图中其它各个点的最短路径问题。最短路径问题是图论算法中的经典问题，也是诸如路径规划、物流规划、GPS 导航、社交网络等现实世界中许多应用的基本问题。[16]图 4 展示了在流式场景下的 SSSP 的求解方法。图中边上的数字表示这两个节点之间的距离，节点内的数字表示当前时刻源点到该节点的最短距离，数字为 0 的节点为源点。图 4(a)反应了某一时刻源点到各个节点的最短路径情况，图 4(b)表示新增一条边（节点内数字为 0 的点和节点内数字为 3 的点新增了一条边，边上的权重为 1）之后的情况。由图 4(b)可知，新增的这条边会将影响沿着某条路径传播下去（即图

中的 0->1->2->7 和图中的 0->1->5)，如图中新增的权重为 1 的这条边，会将原来值为 3 的节点更改为 1，并且该节点的后续节点的值都会被更新。

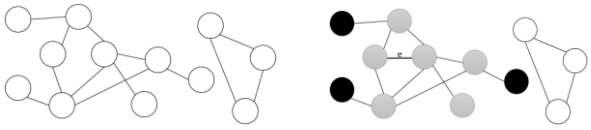


（a） （b）

图 4 单源点最短路径

（5）PageRank

PR 算法[17]是基于网页链接分析来计算各个网页的重要程度。假设网页 A 引用了网页 B，那么 A 就将一定的分数贡献给了 B，该分数（即 PageRank 值，简称 PR 值）就是网页的重要程度的体现。而网页之间是相互引用的，因此经过若干次的迭代之后，网页的得分会趋于稳定，这个分数就反应了该网页的重要程度。图 5(a)展示了某一时刻各个节点的 PR 值，当新增一条边 e 之后，图 5(b)展示了这条边带来的影响：灰色的节点表示新增的这条边直接影响这些节点的值，随后灰色的节点又将这些影响继续往外传播给黑色节点，经过若干次的迭代之后各个节点的 PR 值保持稳定，算法运行结束。由此可见，对于 PageRank 算法，当图中新增一条边时，这条边会影响这条边所在的连通子图内的所有节点。



（a） （b）

图 5 PageRank

（6）总结

当新增一条边时，本文从影响范围、计算次数两个维度来分析这 4 个算法的特点。影响范围是指新增加的这条边可能会影响到哪些节点，而计算次数是指这种影响是否会被计算多次，例如在 DD 算法中，新增加一条边只需要将这条边的源点和目标点对应的度数加 1，且这种运算只需要执行一次，而对于 PR 算法，新增加一条边时，这条边的源点和目标点的输出贡献将会发生变化，因此会首先影响它们的所有邻接点，这些邻接点在下一次的传播中会继续它们的邻接点，经过多次迭代计算之后各个节点的 PR 值会趋于稳定，在迭代计算的过程中，每个节点可能参与多次计算。

表 1 图算法特征表

	影响范围	计算次数
DD	影响新增这条边的源点和目标点	被影响的节点只参与计算一次
TC	影响新增这条边的源点和目标点，以及这两个点的公共邻接点	被影响的节点只参与计算一次
SSSP	以这条边的某个节点为起点，沿着某条路径往其他节点传播影响	被影响的节点可能会参与计算多次
PR	影响这条边的源点和目标点所在的整个连通子图内的所有节点	被影响的节点一般会参与计算多次

1.2 模型定义

传统的图计算模型（例如 BSP 模型）中，图数据是静态的，即在计算的过程中图数据不会发生变化；本文提出的基于状态更新的动态图计算模型，能够很好的解决动态图计算问题，它将动态图在每个时刻抽象成一个状态（State），将流动的图数据抽象成一系列事件流（Event Stream），事件（Event）触发了图由一个状态转换（Transform）成另一个状态。

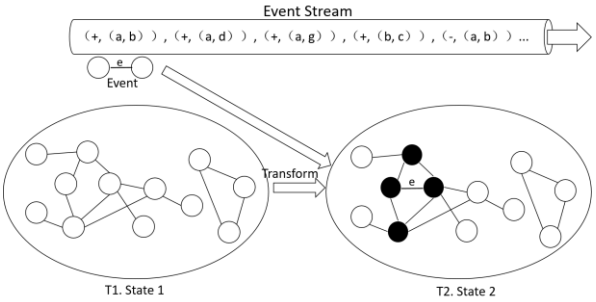


图 6 基于状态更新的动态图计算模型

基于状态更新的动态图计算模型有如下几个定义：

（1）状态（State）：反应了图当前的特征信息，这些特征信息可以以顶点为单位进行体现，也可以由用户自定义的特征信息来体现，状态是由因子（Factor）组成，因子是指组成状态的基本单位，如状态可以以顶点的方式组织，那么这里的因子就是顶点。需要注意的是，状态反应了用户的关注点，虽然是根据流动的图数据而动态计算生成的，但并不等价于图数据本身，即状态不直接存储原始的图数据，而只存储用户关心的图的某些特征信息。这使得系统无需存储庞大的原始图数据，只需要存储设计精巧的状态信息即可反应图的特征信息。例如当统计图的边数，此时 State 可以设计为一个计数器，该计数器反应了当前时刻流入系统中的图的边数，每次新增或者删除边时，增加或减少这个计数器的值，即可实时反应当前图的边数信息。

在这里，本文将状态抽象成一个接口，该接口的基本方法表见表 2，用户可以扩展该接口来实现更加复杂的状态信息。

表 2 State 接口方法表

方法签名	方法作用
State GET-STATE(Factor)	获取指定因子的状态
SET-STATE(Factor, State)	设置指定因子的状态
SET-STATE(State)	向图中添加一个状态
Map GET-STATE()	获取整个图的状态

（2）事件（Event）：触发图由 T1 时刻的 State1 转换为 T2 时刻的 State2 的事件，例如在 T2 时刻新增加了一条边，将使得图由 State1 经过某种运算得到 State2。事件是由事件值（Event Value）和事件类型(Event Type)组成。如“增加一条边 e(v1,v2)”这个事件中，e(v1,v2)是事件的值，“增加”是事件的类型。一般来说，事件的值分为两种：（顶点编号，顶点的值）和（边起点，边终点，边值）；而事件的类型分为三种：新增（ADD），删除（DELETE），更新（UPDATE）。这样总共可以组成 6 种事件：新增边，删除边，更新边；新增顶点，删除顶点，更新顶点。这 6 种事件基本涵盖了所有的图变化的情形。事件的接口方法表见表 3.

表 3 Event 接口方法表

方法签名	方法作用
Value GET-VALUE(Event)	获取指定事件的值
Type GET-TYPE(Event)	获取指定事件的类型

（3）转换（Transform）：由事件触发的图的更新过程，即图是如何根据相应的事件来由 State1 转换成 State2。如图 6 所示，在 T1 时刻图的状态为 State1，在 T2 时刻，图接收了一条边 e，这个事件会触发图状态转换函数（Transform），将图的状态转换为 State2。转换函数是动态图计算模型中的计算逻辑，详细定义了图如何根据到达的事件，从一个状态转变成另外一个状态，可以称之为状态更新的图计算模型的驱动程序，驱动图从一系列的事件流转换成一系列对应的状态流。

1.3 状态的存储和更新

（1）状态类型

基于状态更新的动态图计算模型中，一个核心问题是状态如何存储和更新。状态是从用户的视角来进行设定的。即用户关心什么数据，就可以将该数据设置为图的一个状态，这些状态可以以顶点为单位进行保存：图的状态由各个顶点的状态组成，也可以以边或者其他的方式来组织。相比较传统的顶点编程模型或边编程模型来说，用一个高度可自定义的状态能够直接反应用户关心的结果，使得模型的表达能力更强。

根据上述 3.2 节中图算法分析，在图计算中大致分为两类状态：**独立状态**和**关联状态**。所谓独立状态，是指状态内的各个因子之间是独立的，一个因子的状态的变化不会引起其他因子的状态的变化，如 DD 算法就是属于独立状态范围，每增加一条边，这个事件只会影响增加这条边的两个节点，不会影响到其他的节点；所谓关联状态，是指状态内的各个因子之间相互关联，一个因子的状态的变化会影响到其它因子状态的变化，诸如 TC、SSSP、PR 算法中增加一条边，不仅会影响增加这条边的两个顶点的状态，还会影响到这两个顶点的公共邻接点，甚至整个连通子图内的所有顶点。可以用下图表示这两种情况。

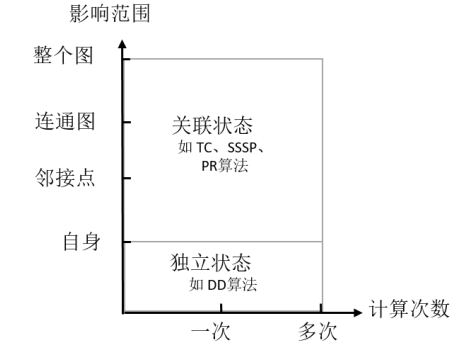


图 7 状态分类图

（2）独立状态的存储和更新

在独立状态中，因为状态内的各个因子之间不会相互影响，因此独立状态可以分布式的存储和并发的更新。即可以按照状态的组织形式，分布式的存储在多个节点上，而且每个节点上的状态都可以同时进行更新，并向用户实时反馈更新结果。这样充分利用了分布式的特点，提高存储和计算效率。状态的并发更新过程如图 8 所示：

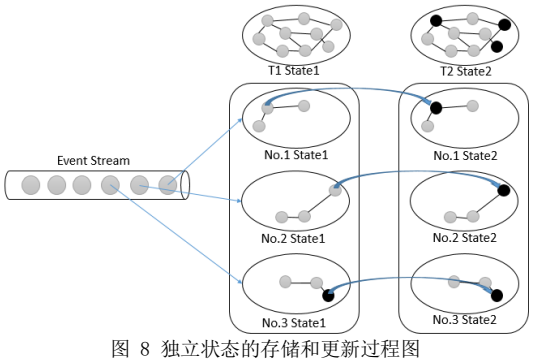


图 8 独立状态的存储和更新过程图

该图演示了独立状态的并行存储和更新的过程。系统接收到事件流（Event Stream）之后，将事件流按照某种分片规则（即特定的图的划分算法），分发到不同的计算节点上（如图所示的 No.1, No.2, No.3 这 3 个计算节点），然后分别在各个计算节点上独立进行状态更新（如图所示对应计算节点的状态从 State1 转换到了 State2），这样使得图由 T1 时刻 State 1，更新成了 T2 时刻的 State 2，注意到这种更新过程充分利用了分布式的优势，多个节点同时独立进行更新，提高了计算效率。

（3）关联状态的存储和更新

在关联状态中，一个因子的状态的变化会影响到其它因子状态的变化，因此，多个事件触发的更新可能会影响同一个因子，引起更新冲突问题。解决更新冲突的方法有很多，最为简单的方式是将多个事件的更新串行化，即对两个事件 A 和 B，事件 A 先触发更新，A 更新完毕后事件 B 再触发更新，在事件 A 触发更新期间，其他任何事件都不得触发更新，以免引起更新冲突问题。诸如 IncGraph[5]就是采用这样的更新模型，这使得即使不会发生更新冲突的两个事件也不能同时更新，无法充分利用多机并行的优势。因此本文提出了两种解决更新冲突的方法：基于分区的并行更新策略和基于细粒度锁的并行更行策略。

基于分区的并行更新策略是将原来的图划分成若干个子图，使得子图内部的节点之间联系比较紧密，子图之间的节点之间几乎没有边相连或者联系较少。这样可以假设子图内节点更新的影响范围只限于子图内部，不会传播到其它子图中节点。因此子图与子图之间的更新可以同时进行，而子图内部的更新则需要串行进行，这样在一定程度上能够提高更新的并行度。如图 9 所示。

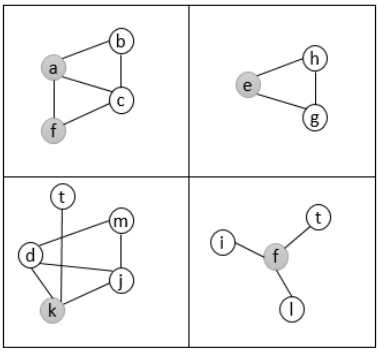


图 9 基于分区的并行更新策略

图 9 展示了基于分区的并行更新策略，按照连通性将原来的图分成如图 9 所示的四个连通子图，在每个连通子图的更新是串行的，如在左上角子图中 a 节点和 f 节点的更新需串行进行，而连通子图之间的节点的更新是并行的，如 a 节点、e 节点、k 节点和 f 节点的更新可以同时进行。如果分区策略划分的好，可以充分利用分布式的优势，实现多个分区并行更新。

基于分区的并行更新策略需要谨慎的选择子图划分算法，该分区算法要能够很好的将原来的大图切分成若干个子图，保证子图之间节点的联系是松散的，子图内部的节点之间的联系是紧密耦合的。关于图划分的问题，现有的研究工作也很多，Ioanna Filippidou 和 Yannis Kotidis[18]提出了一种基于精简生成树结构的图分割算法，它不仅能对任意的演变图进行图分割，还允许不同的应用按需来调整分区; Stanton 和 Kliot[19]提出了一种只依赖于图结构的启发式算法，相对于基于散列的分割方法和 METIS，分割效果有很大提升。另外，Charalampos E. Tsourakakis[20]等人提出了一个新颖的 one-pass 流图分割算法，该算法统一了两个看似正交的启发式算法：将新到达的顶点放置在具有最大数量邻居结点的分区中或者具有最小数量的非邻居结点的分区中。相对于 METIS，平衡分割时间更短，效果更好。本文的工作重心不是比较这些图划分算法的优劣，而是希望借鉴这些现有的图划分算法来提高系统的并行更新能力。

基于分区的并行更新策略不可避免的会出现多个节点的更新会集中在一个子图上的情况，这种情况会严重影响系统整体的并行度。基于分区的并行更新策略本质上是一个范围锁，锁住一定范围内的所有节点数据。这样粗粒度的锁会大大影响系统的并发性，因此本文又提出了基于细粒度锁的并行更新策略，即每次只需要锁住组成状态的单个因子本身，而不需要锁住范围内的所有节点。

传统的 BSP 模型是将整个图的迭代计算过程分解为若干个超步，超步内部的节点之间并行计算，超步之间进行同

步。这使得在每个超步内，计算最慢的节点拖慢整个超步的计算速度，因此会出现短板效应，而本文的基于细粒度锁的并行更新策略有效弥补了这个不足，因子（这里的因子等价于 BSP 模型中的节点）与因子之间的更新都是并行的，只有属于一个因子的多个更新请求才会被串行执行，这样真正实现了多个因子的并行更新策略，而且没有显示的同步过程，消除了短板效应。但这需要因子的更新满足无序性，即对于任何一个节点的若干个更新，这些更新的顺序不会影响整个节点的最终状态。大多数算法（如本文中的 DD、TC、SSSP 等算法）都满足这个条件，因此模型的表达能力不会受到太大影响。

1.4 模型应用举例

基于状态更新的动态图计算模型将图的状态从用户的角度出发，只保存用户关心的数据，相比较传统的基于顶点的编程模型来说表达能力更强。在此本文选取了 TC 算法和 CC 算法来说明如何在该模型上进行算法设计，前者采用的是传统的面向顶点编程方式，后者采用的是用户自定义状态的编程方式。

（1）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。

由前文所知，基于状态更新的动态图计算模型，有 State, Event, Transform 三个重要的概念。针对 Triangle Count 算法，这三个组件的定义如下：

State:图的 State 由每个顶点对应的邻接点的信息组成，即 $\text{State}=\{s_1,s_2,...,s_n\}$，其中 $s_k=(v_k,N_k,t_k)$，表示节点 v_k 的邻接点的集合为 N_k，节点 v_k 构成的三角形的数目为 t_k；

Event:图的 Event 为图到达一条边相关的事件，那么 Event 构成的序列就形成事件流，即 $EventStream=z_1,z_2,...,z_m$，其中 z_k =(e_k ,TYPE)，TYPE ∈{ADD,UPDATE,DELETE},这三种状态对应的事件分别为增加一条边，更新一条边和删除一条边；

Transform:图在动态变化过程中，State 的更新过程见算法 1。

算法 1 Dynamic Triangle Count

```
1   for each z ∈ EventStream
2     do N1 ← ∅
3        N2 ← ∅
4        (v1 , v2 ) ← GET-VALUE( z )
5        S_v1 ← GET-STATE( v1 )
6        S_v2 ←GET-STATE( v2 )
7        if S_v1 ≠ ∅
8           then N1 ← GET-NEIGHBOR( S_v1 ) ∪ { v2 }
9        else N1 ← { v2 }
10       if S_v2 ≠ ∅
11          then N2 ← GET-NEIGHBOR( S_v2 ) ∪ { v1 }
12       else N2 ← { v1 }
13       cross ← N1 ∩ N2
14       type ← GET-TYPE( z )
16       for each v ∈ cross
17          do S_v ← GET-STATE( v )
18             t ← GET-TRIANGLE( S_v )
19             N ← GET-NEIGHBOR( S_v )
20             if type = ADD then S_v ←( v ,N,t+1)
21             elseif type = DELETE then S_v ←( v ,N,t-1)
18             SET-STATE( v , S_v )
19       t1 ← GET-TRIANGLE( S_v1 )
20       t2 ← GET-TRIANGLE( S_v1 )
21       if type = ADD
22          then S_v1 ← ( v1 , N1 , t1 +|cross|)
22             S_v2 ← ( v2 , N2 , t2 +|cross|)
23       elseif type = DELETE
24             S_v1 ← ( v1 , N1 , t1 -|cross|)
25             S_v2 ← ( v2 , N2 , t2 -|cross|)
26       SET-STATE( v1 , S_v1 )
27       SET-STATE( v2 , S_v2 )
```

在算法 2 中，为了能够统计图中三角形数目，用户需要扩展 State 的方法：GET-NEIGHBOR(S_v)用来获取因子 S_v 中的邻接点集合，GET-TRIANGLE(S_v)用来获取因子 S_v 中的三角形数目。有了这些方法和 State 接口原来提供的方法，就可以实现动态的 TC 算法。

（2）Connected Components

如果一个图中，每对顶点都有路径相连，则称其为**连通图**。如果图的子图中任意两个顶点都是可达的，则这个子图称之为图的**连通分支**。连通分支反应了一个大图中子图的聚集情况，可以根据连通分支将原来的大图分解成若干个连通分支，算法独立并行的在连通分支上进行。连通分支在好友推荐、循环引用判断等诸多问题上被广泛使用。下面将介绍在不断增加边的情况下，如何针对无向图做连通分量的计算。由前文所知，基于状态更新的动态图计算模型有三个概念：State, Event, Transform，下面将详细介绍如何定义这三个基本组件：

State:当前图的所有的连通分支，$\text{State}=\{s_1,s_2,...,s_n\}$，其中 s_k 表示第 k 个连通分支，$s_k=(v_{k_{\min}},\{v_{k_1},v_{k_2},...,v_{k_r}\})$，其中集合 $\{v_{k_1},v_{k_2},...,v_{k_r}\}$ 表示由这些顶点构成了一个连通分支，$v_{k_{\min}}$ 是这些顶点中标号最小的点。

Event:图的 Event 为图中新增了一条边，那么这些 Event 构成的序列就形成事件流，即 $EventStream=z_1,z_2,...,z_m$，其中 $z_k=(e_k,add)$，表示新增边 e_k ；

Transform:图在动态变化过程中，State 的更新过程见算法 2：

算法 2 Dynamic Connected Components

```
1  for each Z ∈ EventStream
2    do ( v1 , v2 ) ← GET-VALUE( Z )
3       S_v1 ← GET-STATE( v1 )
4       S_v2 ← GET-STATE( v2 )
5       if S_v1 ≠ ∅ and S_v2 ≠ ∅
6          then UNION-STATE( S_v1 , S_v2 )
7       elseif S_v1 ≠ ∅ and S_v2 = ∅
8          then UNION-STATE( S_v1 , v2 )
9       elseif S_v1 = ∅ and S_v2 ≠ ∅
10         then UNION-STATE( S_v2 , v1 )
11      else
12         s ← (min( v1 , v2 ),{ v1 , v2 })
13      ADD-STATE(s)
```

在算法 2 中，S_{v_1} 表示节点 v_1 所在的连通分支，S_{v_2} 表示节点 v_2 所在的连通分支，新增的这条边 $e=(v_1,v_2)$ 使得这两个节点所在的连通分支合并，构成一个大的连通分支，算法中用户自定义的 UNION-STATE()函数完成两个连通分支的合并。

2 GraphFlow 系统

基于状态更新的动态图计算模型充分考虑了现有的图算法转移到动态图上之后具有的特征信息，针对不同类型的状态信息采用不同的存储和更新机制，充分利用分布式计算的优势，实现了动态图上的实时计算能力。GraphFlow 系统就是一个基于状态更新的动态图计算模型的图处理系统。

系统采用主从架构，由客户端（Client）向主节点（Master）提交任务请求，主节点将任务分配给各个工作节点（Worker），工作节点执行任务完毕后向主节点反馈信息。现有的这样主从式的大数据处理架构有很多，诸如 Spark/Flink 这样高可靠、高性能的分布式内存计算框架，一经推出，在学术界和工业界都备受关注。但遗憾的是无论是 Spark 还是 Flink，Graph 组件提供的算法是运行在静态稳定的图数据上，无法处理流式的动态图数据。因此本文借助 Flink 的流处理引擎，在该引擎之上实现了一套动态图处理的系统，整个系统的架构如图 10 所示。

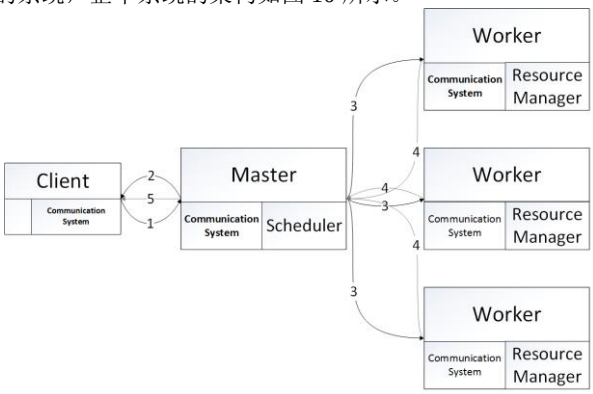


图 10 GraphFlow 系统框架图

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 Worker 节点提交任务，并等待 Worker 节点的执行结果；4. Worker 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。

任务提交到各个 Worker 节点之后，GraphFlow 系统会将原来的边流分片，并且映射成对应的事件流，到达的事件会触发用户预先定义的 Transform 函数，驱动图随着事件的变化不断的更新图的状态，并且将最新的状态及时的反馈给用户。图 11 简要展示了整个系统的执行流程。需要注意的是，Transform 函数在各个 Worker 节点并行执行的，在执行过程中由系统来解决更新冲突的问题。

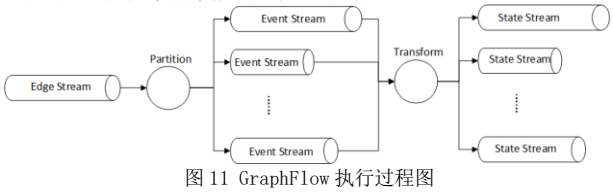


图 11 GraphFlow 执行过程图

3 系统评测

3.1 实验准备

为了能够在不同规模的数据集上测试系统的正确性和实时性，本文按照一定规则生成了 10 组不同规模的测试数据，这 10 组数据的规模保持线性增长的趋势，希望观测在不同数据规模上算法的性能表现，10 组数据的详细配置见表 4。

表 4 测试数据规模表

编号	顶点数目	边数目
D1	1,000,000	8,184,128
D2	1,000,000	16,659,357
D3	1,000,000	25,863,060
D4	1,000,000	33,785,996
D5	1,000,000	42,213,013
D6	1,000,000	47,937,736
D7	1,000,000	55,969,690
D8	1,000,000	64,636,602
D9	1,000,000	72,270,336
D10	1,000,000	79,476,515

本实验主要通过测试图计算中常用的 Triangle Count 算法进行验证。Triangle Count 算法是用来统计有向/无向图中的不同三角形的数目。实验在由 10 台计算机构成的集群上运行，这 10 台计算机中 1 台作为 Master 节点，分发计算任务和收集计算结果，9 台作为 Worker 节点执行计算。单独每台计算机的配置见表 5.

表 5 计算节点配置表

参数	配置
处理器	8 核 Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
内存	16G RAM
硬盘	2 * 1TB SATA
操作系统	Ubuntu 11.04

3.2 正确性

相比较流处理模型，批处理模型一个显著的优点是计算的正确性高。而 GraphFlow 系统没有选择传统的估算模型，而是采用了准确计算模型，所以计算偏差不会太大。如图 12 所示，在小数据集（如 D1,D2,D3）上的偏差会小很多，在大数据集上（如 D8,D9,D10）的偏差也不会太高，而且在这 10 组数据集上的计算偏差都在 1%以内。因此 GraphFlow 系统的计算结果较为准确。

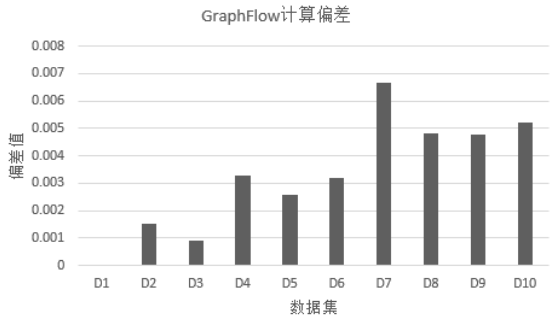


图 12 GraphFlow 系统计算偏差

3.3 实时性

实时性是流式计算系统的重要衡量指标。在这里分别测试基于 BSP 模型的 Flink Gelly 系统和基于状态更新的动态图计算模型 GraphFlow 系统在不同数据集下的表现。图 13 展示了这两个系统在不同数据规模下的吞吐量，由该图可知，当在小规模数据集（如 D2、D3、D4）上运行时，Flink Gelly 和 GraphFlow 的吞吐量相差不大，都达到了 60 万条/秒的处理速度，当在较大数据规模的数据集（如 D8、D9、D10）上运行时，因为顶点之间的通信代价增加，Flink Gelly 的处理速度持续下降，而基于增量更新的 GraphFlow 采用了细粒度锁的方式来解决冲突，所以保持在 40 万条/秒左右的处理速度。Flink Gelly 是基于批处理的系统，处理速度显然要比 GraphFlow 这样基于流处理的系统吞吐量要高，但根据图 13 发现，这种处理的速度差距在一个数量级以内，而且当数据规模增大时，GraphFlow 的吞吐量的变化相比 Flink Gelly 稳定。

此外，本文还比较了 GraphFlow 在并行处理和串行处理下的响应时间差距，如图 14 所示，即使在小数据集（如在 D1、D2、D3）上，并行处理和串行处理的响应时间也相差 10 倍左右，而且随着数据规模的增大（如 D8、D9、D10）时，这种差距更加明显。这是因为当数据规模增大时，两个不同的节点更新冲突的概率会变得很小，那么这样的两个节点可以独立的进行计算，因此并行处理相比串行处理优势会更加明显。

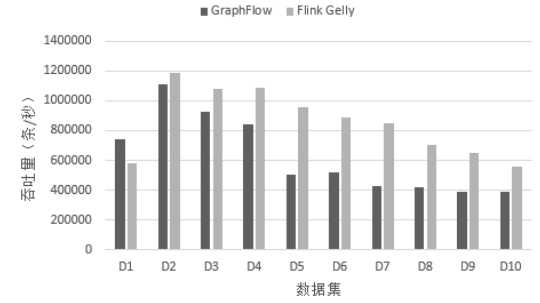


图 13 GraphFlow 和 Flink Gelly 吞吐量

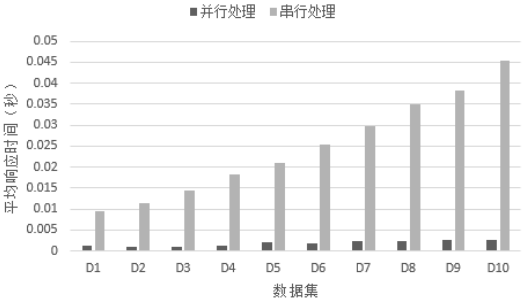


图 14 系统的实时性

4 相关工作

相比较本文提出的基于状态更新的动态图计算模型，传统的批处理计算模型（例如 BSP 模型）是在静态稳定的图数据上迭代进行计算的，它无法感知诸如新增边或顶点这样的外来事件。而且在每一步的计算中，所有活跃的顶点都会参与计算，通信和计算代价较高，而基于状态更新的流图计算模型只针对新增的事件进行状态更新，代价较小。相比较 KineoGraph[13]和 IncGraph[5]的增量计算的模型，本文进一步的引进状态、事件和更新三个概念，根据图状态的不同采用不同的存储和更新方式，提高了并行度。相比较 SpecGraph[14]提出的基于推测机制的并发更新模型，本文的状态更新模型允许用户自定义状态信息，更符合现有的图计算的场景，因此模型的表达能力更强。

5 结束语

本文提出一种基于状态更新的动态图计算模型，通过对不同状态采用不同的存储和更新策略，提高了系统的并行度的同时，保证了系统的计算结果的正确性。基于此模型设计了 GraphFlow 系统，利用基于细粒度锁的更新策略来有效解决更新冲突问题，保证了系统的高吞吐量和低延迟。未来希望能够在系统上实现更多的算法和应用，进一步评测系统的性能。

参考文献:

[1]袁培森，舒欣，沙朝锋，徐焕良. 基于内存计算的大规模图数据管理研究[J]. 华东师范大学学报(自然科学版)，2014，05:55-71.

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 135-146.

[3] Xin R S, Gonzalez J E, Franklin M J, et al. Graphx: A resilient distributed graph system on spark[C]//First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013: 2.

[4] Valiant L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8): 103-111.

[5] 申林，薛继龙，曲直，杨智，代亚非. IncGraph:支持实时计算的大规模增量图处理系统[J]. 计算机科学与探索，2013，12:1083-1092.

[6] Bar-Yossef Z, Kumar R, Sivakumar D. Reductions in streaming algorithms, with an application to counting triangles in graphs[C]//Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2002: 623-632.

[7] Tsourakakis C E, Kang U, Miller G L, et al. Doulion: counting triangles in massive graphs with a coin[C]//Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009: 837-846.

[8] Buriol L S, Frahling G, Leonardi S, et al. Counting triangles in data streams[C]//Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2006: 253-262.

[9] S. Baswana. Streaming algorithm for graph spanners – single pass and constant processing time per edge. Inf. Process. Lett. 106(3):110–114, 2008.

[10] M. Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. ACM Transactions on Algorithms, 7(2):20, 2011.

[11] A. A. Bencz´ ur and D. R. Karger. Approximating s-t minimum cuts in ˜O(n2) time. In ACM Symposium on Theory of Computing, pages 47 – 55, 1996.

[12] Chu S, Cheng J. Triangle listing in massive networks and its applications[C]//Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011: 672-680.

[13] Cheng R, Hong J, Kyrola A, et al. Kineograph: taking the pulse of a fast-changing and connected world[C]//Proceedings of the 7th ACM european conference on Computer Systems. ACM, 2012: 85-98.

[14] 景年强，薛继龙，曲直，杨智，代亚非. SpecGraph:基于并发更新的分布式实时图计算模型[J]. 计算机研究与发展，2014，(S1):155-160.

[15] S. Muthukrishnan. Data Streams: Algorithms and Applications. Foundations and Trends in Theoretical Computer Science, 1(2), 2005.

[16] 张钟. 大规模图上的最短路径问题研究[D]. 中国科学技术大学，2014.

[17] Page L, Brin S, Motwani R, et al. The PageRank citation ranking: bringing order to the web[J].

[18] Ioanna Filippidou and Yannis Kotidis. Online and On-demand Partitioning of Streaming Graphs. In 2015 IEEE International Conference on Big Data (Big Data), 2015.

[19] Isabelle Stanton and Gabriel Kliot. Streaming Graph Partitioning for Large Distributed Graphs. In KDD '12, pages 1222-1230, 2012.

[20] Charalampos E. Tsourakakis, Christos Gkantsidis, Bozidar Radunovic and Milan Vojnovic. FENNEL Streaming Graph Partitioning for Massive Scale Graphs. In WSDM '14, pages 333-342. 2014.

[21] Luis M. Vaquero, Felix Cuadrado, Dionysios Logothetis and Claudio Martella. Adaptive Partitioning of Large-Scale Dynamic Graphs. In SOCC '13, article No. 35, 2013.