**CGraph: A Correlations-aware Approach for**

**Efficient Concurrent Iterative Graph Processing**

（CGraph:一种支持关联的方法，用于高效的并发迭代图形处理）

论文背景综述

姓名：陈静 学号：M201872991 班级：硕1807 论文号：31

Yu Zhang, Xiaofei Liao, Hai Jin, and Lin Gu, Huazhong University of Science and Technology;

Ligang He, University of Warwick; Bingsheng He, National University of Singapore;

Haikun Liu, Huazhong University of Science and Technology <https://www.usenix.org/conference/atc18/presentation/zhang-yu>

**摘要**

随着迭代图分析应用程序的快速增长，图形处理平台必须有效地处理大量并发迭代图形处理（CGP）作业。尽管对单一作业的执行进行了广泛的研究，但由于存在大量的缓存干扰和内存墙，现有的解决方案面临着高比率的数据访问成本，这导致了低吞吐量。在这项工作中，我们观察到不同的CGP作业所发出的数据访问之间存在很强的空间和时间相关性，因为这些并发运行的作业通常需要反复地遍历共享的图形结构，以实现每个顶点的迭代处理。在此基础上，提出了一种基于关联的执行模型，并提出了一种基于核心的调度算法，使这些CGP作业能够有效地共享缓存/内存中的图形结构数据，并充分利用这些相关性。它能够有效地降低数据访问成本的平均比率，从而实现CGP作业的高效执行，从而实现更高的吞吐量。为了证明所提出的方法的有效性，开发了一种称为CGraph的系统，并进行了广泛的实验。实验结果表明，与现有的解决方案相比，CGraph提高了CGP作业的吞吐量提高了2.31倍。

**1 研究现状**

随着图表规模的爆炸，许多系统[18，19,27]专注于实现迭代的高效率图分析。但是，大多数都专注于支持单图处理工作。他们改善了通过充分利用顺序使用来提高效率内存带宽，或通过实现更好的数据因此，局部性和较少的冗余数据访问减少访问数据的量。[17]实现了顺序存储访问采用平行滑动窗。 [23]和Chaos [22]通过使用流分区来改进用于更好地顺序访问核外数据。谢等[28]提出了一种新的面向块的计算方法模型，其中计算在本地迭代高度连接的节点块，改善了每个缓存未命中的计算量。 [31]使用一组基于树的分区模拟大图为了更好的地方。 [32]提出2-级别分层分区方案，以改善局部性并减少I / O的数量。 [11]使用目标排序的子分片结构，用于存储图形更好的地方性和自适应地选择最快的策略充分利用内存并减少数据传输。代替针对更好的地方， [6]建议减少通过重新进入的总数据访问成本加载数据和超出邻域访问。虽然这些系统可以支持高效执行单个迭代图处理作业，多个需要在中创建相同图形的单独副本他们为CGP工作的主要记忆。以下在这个方向上， [29,30]旨在允许多个作业正确共享内存的一个副本图结构数据。最近提出了图数据库[8]支持图表上的并发查询。例如，TAO [9]提供了一个简单的数据模型和API来存储和查询图形数据。[24]使用基于RDMA的提供低延迟并发查询的方法基于图形的大型RDF数据集。但是，这些图表数据库解决方案无法有效地支持执行CGP工作，因为他们致力于图表通常只触及不同小子集的查询只需一次图形，而不是迭代处理多轮的整个图表。

**2 主要思想**

**如何提高CGP工作的吞吐量**：在实践中，CGP作业通常需要反复遍历共享图形，并迭代地处理每个顶点以达到它们自己的目的。它表明，在每个迭代中，这些作业被这些作业访问的图形结构数据之间有大量的交集，即数据访问的空间相关性。共享图形结构的分区可能需要在短时间间隔内通过多个作业来访问，即数据访问的时间相关性。

**CGP工作的数据访问问题：**两个观察。首先，并发的方法比按顺序执行作业的顺序要好。其次，随着工作机会的增加，每个工作的平均执行时间会显著延长。

**CGP工作之间的相关性**：在每个迭代中由不同的CGP作业处理的图形分区的交集很大（平均超过所有活动分区的75%——空间相关性。一些图形分区可以在短时间内由多个CGP作业（可能超过16个作业）访问——时间相关。

这篇文章提出了一种以数据为中心的负载触发（用LTP表示）模型，通过充分利用其数据访问的相关性来提高CGP作业的吞吐量。它将图形结构数据与每个作业关联的顶点状态分离。在每次迭代中，由多个CGP作业共享的图形结构分区被流到缓存中，并触发相关作业并发处理数据，然后是顶点状态推动收敛。通过这种方式，许多对共享图形分区的访问可以通过一个共同的顺序来处理多个CGP作业。

**3方法**

**挑战**：首先，共享的顶点和边可以由不同的作业在不同的图形路径上单独处理。其次，CGP作业有不同的属性（例如，收敛和提交时间的回合），这减少了在短时间间隔内共享图形结构数据的机会。第三，设计一个高效的分区加载顺序是一项非常重要的任务，它可以实现高缓存利用率。

**主要方法**：这篇文章提出了一种以数据为中心的加载触发（LTP）模型，以充分利用CGP作业之间的空间/时间相关性，以最小化共享图形结构数据的冗余访问和存储成本。在LTP模型中，共享图被划分为一组分区。这些分区按顺序加载到缓存中，对于所有作业都是相同的，其中每个分区都由相关的CGP作业并发处理。通过这种方式，大多数图形结构分区的存取和存储可以由多个CGP作业共享，从而大大减少了数据访问成本。在载入图形分区时，会进一步开发调度算法来指定图形分区的加载顺序（以及相关的特定于工作的数据）。调度程序的目标是充分利用工作数据访问之间的时间相关性，从而最大化缓存的利用率。

**3.1 以数据为中心的LTP执行模型**

假设迭代图算法的数据被表示为，其中是顶点的集合，是顶点的集合，是边的集合，是与边相关联的权值集。在LTP模型中，每个作业的数据都与图形结构数据解耦。和特定于工作的顶点状态。在S中，是由不同的作业共享的，而是图G的第 个分区。每个作业都有它自己的S，并且通过它的处理迭代不断地更新它的S，直到计算结果收敛。每个迭代的处理分为三个阶段：图形加载、并行触发器和推阶段，这些阶段都是按照以下方式形式化的。

**图加载**：在每次迭代中，共享的图形结构分区，例如，，按顺序依次加载到CGP作业中。它执行以下操作负载图分区:,表示一个操作符,加载数据参数列表中指定的“\*”到缓存中,的工作集,表示 的顶点与相关工作状态,和是加载到缓存的数据。是与第作业相关的顶点状态集。通过这种方式,它只需要加载每个共享图像分区的副本,例如,，对于多个CGP作业和分区，也按照一个共同的顺序为这些作业加载，通过充分利用工作的相关性，为保留冗余访问提供机会。

**触发器和并行执行**：对于每一个加载的图分区，相关的CGP作业，即需要处理分区中的顶点而尚未获得收敛结果的作业，则被触发并发执行下列操作符：

。的函数表示被激活的在加载的数据上执行的特定图形处理操作（例如为实现自己的目标，和）。它的输出（由表示）是与的顶点相关的新状态，并且与作业相关联。是与所有CGP作业中的顶点相关的新顶点状态。当对所有相关作业完成处理时，下一个分区就可以被加载了。通过这种方式，它可以使多个作业定期并发地处理共享的图形分区，以实现它们自己的目标，并高效地共享对它们的访问，以降低开销。

如果一个作业，例如，，在一个迭代中处理了所有的活动分区，它的新计算结果，也就是。，在这个迭代中被要求在其不同分区（存储在自己的特定于工作空间）的顶点之间进行状态同步，以实现收敛。然后，作业开始一个新的迭代。请注意，一旦在当前的迭代中处理了所有活动分区，CGP作业就会转移到下一次迭代，因此不同的CGP作业可能在它们的图形处理的不同迭代中。例如，BFS 10可能只需要在每次迭代中处理一些活动分区，而其他算法，例如PageRank 21，可能需要遍历所有分区才能完成一次迭代。

图3给出了一个例子来说明LTP模型。在这个例子中，图4（a）中的图形被划分为两个分区，需要处理两个作业，即，

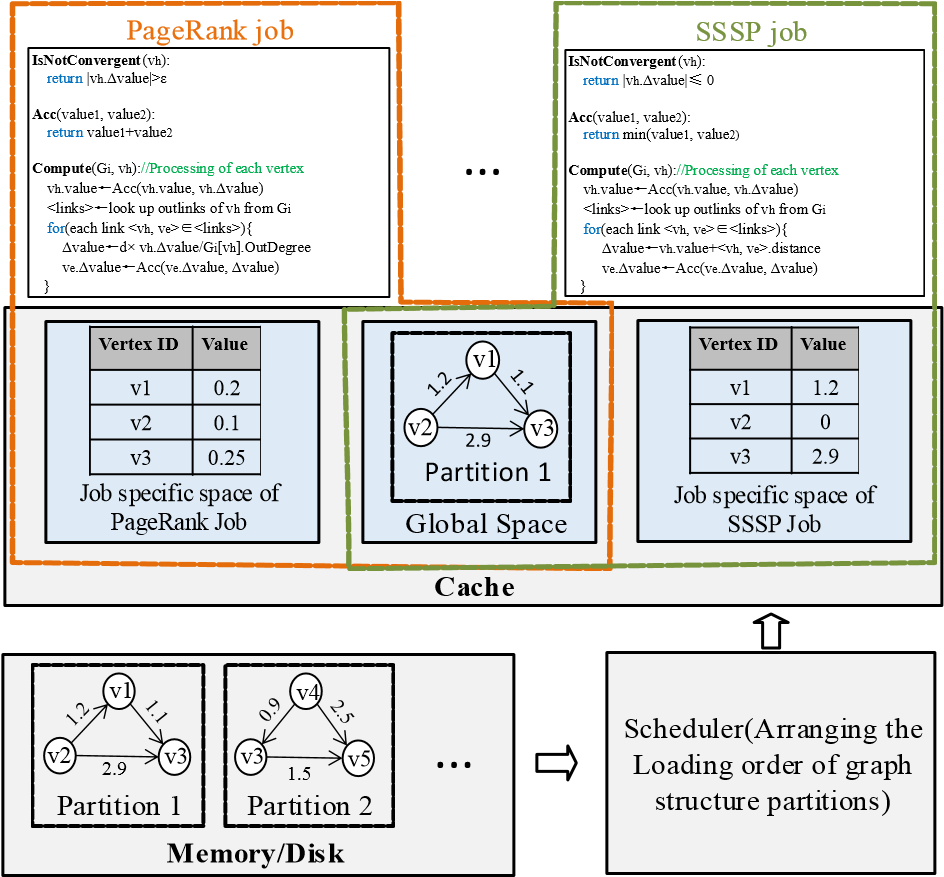


Figure 3: Illustration of our data-centric LTP model

一份PageRank的工作和一个SSSP的工作。图形结构数据存储在全局空间中，并由这两个CGP作业共享，而每个CGP作业则提供特定于工作的空间来存储自己的顶点状态。它可以按照分区1和2的顺序加载两个分区。当分区1和相关的特定于工作的数据被加载到缓存中时，相关的作业（即PageRank作业和SSSP作业）被触发并发地处理它并更新它们自己的顶点状态。当两个作业处理完分区1时，分区2可以被加载进行处理。当两个分区都由作业处理时，每个作业的新迭代就开始了。

**3.2相关工作的执行**

本节讨论**如何有效地实现LTP模型**，用于执行多个CGP作业。

**3.2.1 多个CGP作业的图形存储**

我们首先展示了如何在我们的方法中有效地存储CGP作业的图形。

**图分区的数据结构**：一个顶点可能有多个副本（例如，图4（b）中的v3），其中一个副本被指定为主顶点，而其他副本被视为镜像顶点。通过这种方式，它不仅获得了分区的平衡负载，而且在处理每个分区时也不会产生通信成本。只有当同一顶点的副本不同时才会发生通信。

v1

v2

v3

v4

v5

2.9

1.5

Partition 1 Partition 2

1. Example graph

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PageRank Job | | | | | | | … | SSSP Job | | | | | | |
| **Vertex ID** | | Value | Vertex ID | | | Value |  | Vertex ID | | Value | Vertex ID | Value | |
| v1 | | 0.2 | v3 | | | 0.05 | v1 | | 1.2 | v3 |  | |
| v2 | | 0.1 | v4 | | | 0.1 | v2 | | 0 | v4 |  | |
| v3 | | 0.25 | v5 | | | 0.3 | v3 | | 2.9 | v5 |  | |
| Private Table Partitions | | | | | | | Private Table Partitions | | | | | |
| **Vertex ID** | | Edge List | | | Flag | Master Location | | | | | Information Associated with Its Edges | | | |  | |
| v1 | | v3 | | | Master | Partition 1 | | | | | 1.1 | | | |
| v2 | | v1, v3 | | | Master | Partition 1 | | | | | 1.2, 2.9 | | | |
| v3 | | Ø | | | Master | Partition 1 | | | | | Ø | | | |
|  | |  | | |  | | | | | | | | | |
| **Vertex ID** | | Edge List | | | Flag | Master Location | | | | | Information Associated with Its Edges | | | |
| v3 | | v5 | | | Mirror | Partition 1 | | | | | 1.5 | | | |
| v4 | | v3, v5 | | | Master | Partition 2 | | | | | 0.9, 2.5 | | | |
| v5 | | Ø | | | Master | Partition 2 | | | | | Ø | | | |
|  | |  | | | Graph Structure Partitions | | | | | | | | | |

1. Details for the related tables

图4：展示如何为多个作业存储数据的示例，其中图被划分为两个分区。

**分区同步状态**:为了有效地存储CGP作业的图形分区，建立了多个键值表。 见图4。

**图分区的合适大小**:为了有效地使用CPU的并行性并确保良好的缓存位置，当每个核心都有数据要处理时，缓存预计将被完全加载。因此，每个图形结构分区的适当大小，即，是由CPU核心的数量决定的。*N*和缓存的大小，如*C*，的值是最大的

Timestamp3

Timestamp 2

Timestamp 1

Job 1

Job 2

Job 3

Partition 4

Partition 2

Partition 4

Partition 1

Partition 2

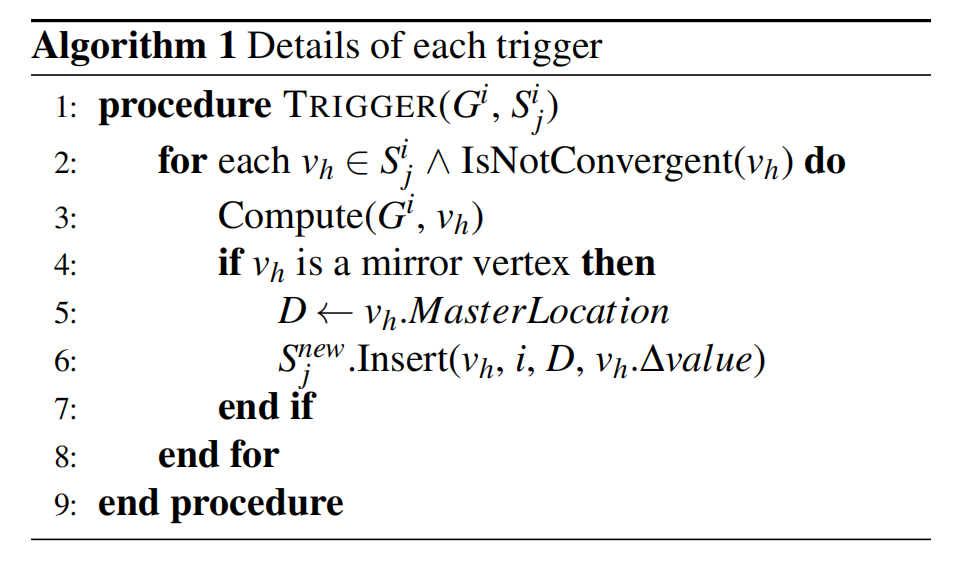
Partition 3

Partition 4

Time

图5：一个例子说明了如何为在不同时间戳中提交的CGP作业存储不断变化的图形结构，其中分区2和4在时间戳2中被更改，分区4在时间戳3中更改。

**3.2.2加载图分区**



不活动顶点的加载和处理可以跳过相关作业，以获得较低的开销。例如，，即需要处理的CGP作业。当没有作业来处理时，它不会加载。在中，顶点的状态对于所有的作业都是不活动的。

**3.2.3图形分区的并行处理**

在将图形分区加载到缓存中之后，它会触发相关的CGP作业（例如，j）来并发地处理与该分区相关联的私有顶点状态（例如）。CGP作业的数量可能超过了CPU核心的数量。假设一个分区是由的作业数共享的。当的值大于*N*时，这些CGP作业被分配为不同的批处理，其中共享的图形结构分区是固定在缓存中的，并且只替换特定于作业的分区。

在这个分区的私有表中找到有最多未处理的顶点的作业。如图6所示，它在**逻辑上将未处理的顶点划分为私有分区将其分解为碎片并将其分配到自由核心以协助其处理**。

**Cache**

Private Partition 1 of Job 1

Private Partition

of Job

2

1

Private Partition

3

of Job

1

Graph Structure Partition 1

**Core 1**

Job 1

Core 2

Job 1

Core 3

Job 2

Core 4

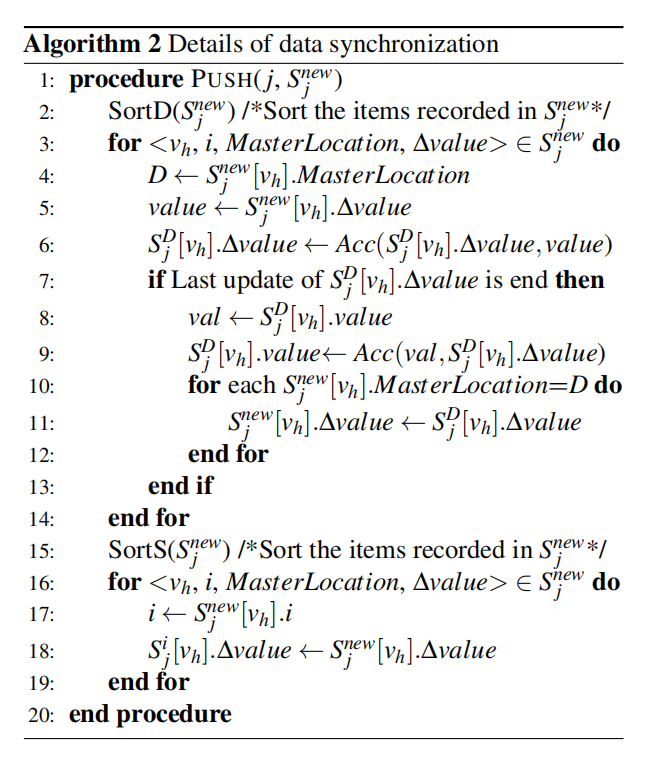
Job 3

图6：一个例子来说明如何获得平衡负载，核心1和核心2一起处理作业1中私有表的分区

**3.2.4数据同步**

当有多个CGP作业来同步顶点状态时，为了减少资源争用，它是由一个接一个地完成的，因为在作业之间没有数据共享。为了在副本中高效的顶点状态同步，如算法2所示，它们在每个作业的这个阶段分批完成，以避免在运行时频繁地加载私人表的分区。在队列中缓冲的条目（带有镜像顶点的新状态，例如）首先根据IDs进行排序（例如，。与相关主顶点（见第2行）的分区的主位置，在推它们之前（见第2行）。

**对镜像顶点进行批处理状态的更新**（参见第10-12行）。注意，在传统的解决方案中，不可能对数据同步的两个细节进行算法



**3.3 基于核心子图的调度**

为了最大限度地提高加载到缓存中的每个分区的利用率，提出了一种基于核心子图分区的调度算法。关键的想法是首先将核心顶点（比给定的阈值更高）放在一起，然后让加载的分区通过尽可能多的作业共享

**安排图分区的加载顺序**：它首先识别一个核心子图，由核心顶点和它们之间路径上的边组成。然后，它根据这样的子图均匀地划分图形，在这个子图中，这个子图的边缘被放在几个相同大小的分区中，剩下的边被划分为其他相同大小的分区。

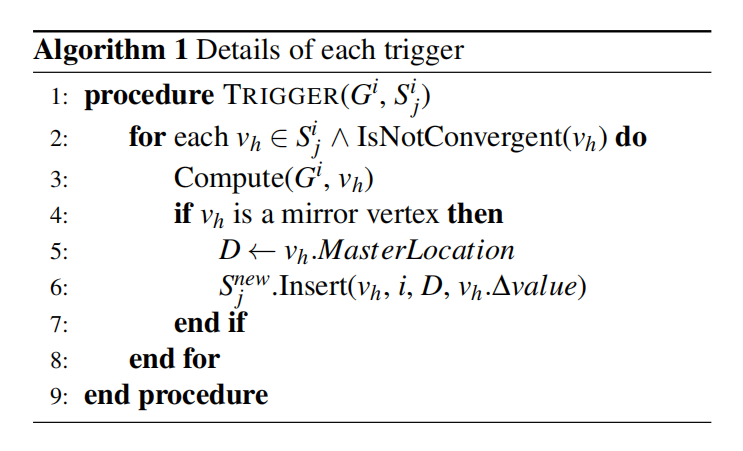
规则表达式：

(1)

其中是处理的作业数，用于捕获CGP的时间相关性。是P的顶点的平均度，而是在前一个迭代中所有作业的P点的平均状态变化。和和的的初始值在预处理时间得到，而和在执行时增量地更新。在这里，是运行时系统在预处理时设置的比例因子，以确保具有高的分区首先处理的最大值，其中和是任何分区的和的最大值。通过这种方式，加载到缓存中的分区可以提供尽可能多的工作，而其他分区在一段时间间隔后有更多的机会需要更多的工作，通过减少平均数据访问成本进一步提高吞吐量。

**3.4实现和接口**

在算法3中描述了CGraph的实现细节。它会根据调度算法（参见第4行），反复地将未处理的分区加载到缓存中，例如，全局表的，将其加载到缓存中（参见第4行）。



**4实验**

**实验环境：**

**结果**：的体积是hyperlink14的47.1%。和、、相比分别使CGP作业的吞吐量提高了3.29倍、4.32倍和2.31倍。

**5 结论**

这篇文章观察发现了许多冗余数据访问因其强烈的时间和空间特性在工作中的相互关系。提出一种新的以数据为中心的LTP模型，并且提出了一种高效的调度算法。观察到这些工作中的数据访问相关性并允许多个CGP作业有效地摊销数据访问成本为更高的吞吐量。实验结果表明这样的方法与现有的技术相比显著地改进了CGP工作的吞吐量。

**展望**：这篇文章主要研究静态图形处理。在未来，可研究如何进一步优化演化图分析的方法并将其扩展到分布式平台和异构平台，从而得到更高的性能CGP作业的吞吐量。

**支持项目：**

论文由国家重点研究开发项目(批准号:2018YFB1003500)、国家自然科学基金(批准号:61702202、61628204)、国家博士后科研流动站(批准号:2017M610477、2017T100555)资助。