《海量存储技术》论文报告

专 业 名 称 ：计算机技术

学 生 姓 名 ：叶宇轩

任 课 老 师 ：何水兵 副教授

学 号 ：2017282110251

GridGraph：在单个机器上采用两级分级分区进行大规模图形处理

摘要：本文提出了 GridGraph，一个用于在单个机器上处理大规模图数据的系统。GridGraph 使用预处理中的第一种细粒度级分区将图分成一维分区的顶点块 chunk 和二维分区的边块 block，在运行时应用第二粗粒度级分区。通过一种新颖的双滑动窗口方法，GridGraph 可以流化边并应用即时顶点更新，从而减少计算所需的 I / O 量。边的分割还使得能够进行选择性调度，使得可以跳过一些块以减少不必要的 I /O。当活动顶点集合随收敛而收缩时，这是非常有效的。

评估结果表明，GridGraph 无缝地扩展内存容量和磁盘带宽，并且胜过最先进的核外系统，包括 GraphChi 和 X-Stream。此外，我们表明 GridGraph 的性能甚至能与分布式图计算系统相竞争，并且它在云环境中具有显著的成本效应。

1 提出问题

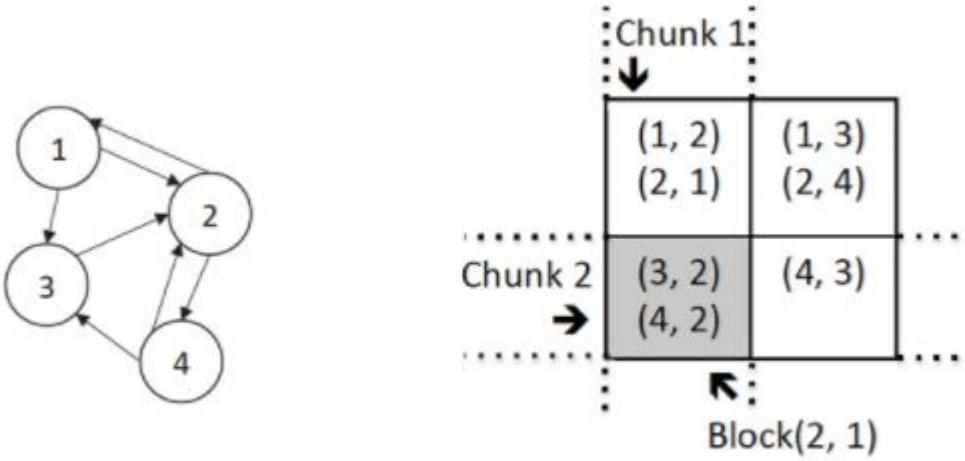
高效的处理大规模图数据在学术界和工业界越来越引起兴趣，现实世界中的很多问题，例如在线社交网络，网页图结构，用户项目矩阵等都能表示成图计算问题。很多分布式系图处理系统在过去的几年中已经被提出，它们能通过集群强大的计算资源来处理大规模的图数据。然而同步开销和容错开销在分布式图计算系统中依然是巨大的挑战，并且用户需要具备在分布式系统集群中调优图算法的能力，这是非常困难的。

本文提出的 GridGraph 将顶点分割成一维的块，根据边的源顶点和目的顶点被分割成二维的边。我们在运行时实现了一种高层次的逻辑分割。为了更大的 I / O 效率，块和边被进一步分组。不同于当前以顶点为中心和以边为中心的处理模型，GridGraph 组合分散和聚集阶段为一个“streaming-apply”阶段，它流式的处理边和将 生成的更新立即应用到源顶点或目的顶点。通过聚合更新，仅需要遍历边一次。这对于迭代全局计算几乎是最优的，并且适合于内存和 out-of-core 的情况。并且 GridGraph 还提供了选择调度，所以可以避免处理没必要的边。本方法提升了很多迭代算法的性能。

2 解决方案

2.1 图的表示

GridGraph 将顶点分割成 P 个均等的块。每个块内的顶点连续排列。全部的 P*P 个边可以看出一个网格，每个边按照如下规则被放入对应的块中:源顶点决所在块的行，目的顶点决定所在 block 的列。下图(b)展示 GridGraph 如何将图(a)分割的例子。图中有 4 个顶点，在例子中我们选择 P=2。{1,2}和{3,4}是两个顶点 chunk。例如边(3,2)被分割到 Block(2,1)，因为顶点 3 属于块 2,顶点 1 属于块 1。同时，GridGraph 创建了一个 metadata 文件，其包含了图的全局信息，有顶点和边的数量，划分的数量和边的类型（是否带权值）。每一个边在磁盘上对应一个文件。



(a) An example graph (b) Grid representation

GridGraph 预处理（图分割）的步骤如下：

主线程从原始无序的边列表中顺序的读取每条边，并把它们分割成边批处理，然后依次将每个批处理推入到工作队列中（为了实现大量的顺序磁盘带宽，我们选择每个边批量的大小为 24MB）。

每一个工作线程从工作队列中获取任务，计算批处理中的每条边属于哪个边，并添加到对应的边文件中。为了提升 I/O 吞吐，每个工作线程为每一个边维护一个本地的缓冲，当缓存满时会一次性冲洗到对应的边文件中。

图分割处理结束后，GridGraph 就可以准备计算了。然而现实世界图无规律的结构，有的边可能会非常小，而不能在 HDDs 上实现大量的顺序带宽。图 2 展示了 Twitter 图 32 X 32 分割边 block 大小的分布情况，这符合 power-law 特性，大量的小文件和极少的大文件。因此不能实现完全的顺序带宽，有时是由于潜在的频繁磁盘寻道。为了避免性能损失，GridGraph 在基于 HDD 系统上需要一个额外的合并阶段来活动更好的性能。

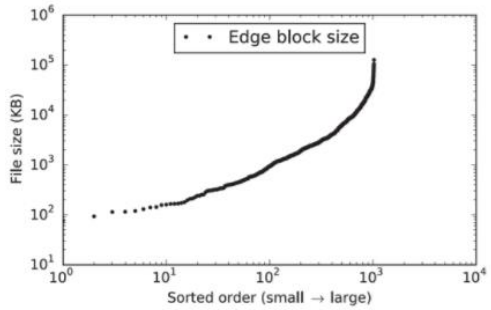


Figure 2: Edge block size distribution of Twitter graph using a 32×32 partitioning.

由于 GridGraph 不需要对每个 block 中的边进行排序。这点重大的减少了预处理阶段的 I/O 和计算开销。我仅需从磁盘上读和写边一次，而不需要像 GraphChi 那样多次遍历边，这种轻量级的预处理可以快速的完成。因此 GridGraph 只花费很短的时间就能完成完整的图分割预处理，并且同一个图生成的网格能很好的用于所有的算法。通过分割，GridGraph 能进行选择调度来减少没必要的边访问。

划分 P 值得选取十分重要。一个细粒度的划分（一个很大的 P 值）会使预处理时间变长，会有更好的顶点数据本地性，更好的选择调度可以被实现。因此划分时 P 优选一个大值。当前我们选择 P 的方法是顶点数据可以全部载入到最快的 cache 中。选取 P 为最小的整数：

$$\frac{V}{P} \times U \leq C,$$

C 是最快级别 cache 的容量，U 是每个顶点的大小（所占字节数）。这样的划分不仅带来很好的性能（特别是 in-memory 的情形）而且具有合理的预处理成本。

2.2 流应用处理模型分析

2.2.1 编程算法

GridGraph 使用 streaming-apply 的处理模型，只需要遍历（读）边集一次和遍历（被优化的写 I/O）顶点集一次。GridGraph 提供两个方法来流式遍历顶点（算法 1）和边（算法 2），具体的算法代码在这里不在粘贴。本算法主要是接受一条边或者一个顶点作为输入，并且返回一个 R 类型的值。返回值被累积并作为最终结果返回给用户。这个值经常被用于获取活跃顶点的数量，但不限于这个用法，用户也可以使用这个方法获得在 PageRank 算法中不同迭代的差值之和来决定是否停止计算。

GridGraph 在磁盘上存储顶点数据。每一个顶点数据文件对应一个顶点数据向量。我们使用内存映射机制来引用文件中的顶点数据。它提供方便和透明的向量访问，并简化了编程模型。开发人员可以将其视为正常数组，就像它们在内存中一样。

我们使用 PageRank 作为例子展示（见算法 3）如何用 GridGraph 实现该算法。PageRank 是一个链接分析算法，它计算图中每个顶点的数字权重，以测量它在顶点之间的相对重要性。

每个顶点的 PR 值被初始化为 1.在每次迭代中，每个顶点发送它们对邻居的贡献，这是当前 PR 值除以其出度。每个顶点总计从邻居收集的贡献并将其设置为新的 PR 值。当平均差达到某个阈值时，就收敛了。

Algorithm 3 PageRank

function CONTRIBUTE(e)
    Accum($\&NewPR[e.dest]$, $\frac{PR[e.source]}{Deg[e.source]}$)
end function
function COMPUTE(v)
    $NewPR[v] = 1 - d + d \times NewPR[v]$
    return $|NewPR[v] - PR[v]|$
end function
$d = 0.85$
$PR = \{1, \ldots, 1\}$
$Converged = 0$
while $\neg Converged$ **do**
    $NewPR = \{0, \ldots, 0\}$
    StreamEdges(Contribute)
    $Diff$ = StreamVertices(Compute)
    Swap(PR, $NewPR$)
    $Converged = \frac{Diff}{V} \leq Threshold$
end while

2.2.2 双滑动窗口

GridGraph 的特点是逐块流边。当流传输到特定的块时，例如，第 i 行和第 j 列中的块时，与该块相关联的顶点数据落入第 i 个数据块和第 j 个数据块。通过选择 P 使得每个块足够小以适合快速存储（即用于内核情况的存储器或用于存储器内情况的最后一级高速缓存），当访问与块相关联的顶点数据时，我们可以确保良好的局部性。

基于更新模式，块的访问顺序可以是行优先或列优先。 假设顶点状态从源顶点传播到目的顶点（这是许多应用中的典型模式），即读取源顶点数据并写入目的顶点数据。由于每个边缘块的列对应于目的顶点块，因此在这种情况下优选列取向访问顺序。当 GridGraph 从上到下地在同一列中流动块时，目标顶点块被高速缓存在存储器中，使得昂贵的磁盘写入操作被聚集和最小化。这种性质在实际使用中非常重要，特别是对于 SSD 来说，因为写入性能

可能由于写入放大现象而在写入大量数据之后降低。另一方面，由于 SSD 具有写周期的上限，因此重要的是尽可能多地减少磁盘写入以实现理想的耐久性。

由于 GridGraph 将现场更新应用到活动顶点数据块时，可能存在数据争用现象，即一个顶点的数据由多个工作线程同时修改。因此在处理函数内，用户需要使用原子操作来对顶点应用线程安全更新，以确保算法的正确性。利用对快速级存储的并行随机访问的带宽仍然比慢存储（例如主存储器与磁盘以及高速缓存与主存储器）的顺序带宽更高数量级的事实，应用更新的时间与边流重叠。 GridGraph 只需要在边上进行一次读取，这与其它的处理方法相比性能更加优化。

利用双滑动窗口的方法主要由以下两个优势：第一，通过只读访问边，GridGraph 所需的内存非常紧凑。实际上，它只需要一个小的缓冲器来保存正在流传输的边数据，以便页面缓存可以使用其他空闲内存来保存边数据；第二，它不仅支持经典 BSP 模型，而且允许异步[3]更新。 由于顶点更新是即时的，更新的效果可以通过随后的顶点访问并看到，这使得大量迭代算法更快地收敛。

2.2.3 两级分层分割

我们首先在完整的流应用迭代中给出 GridGraph 的 I / O 分析，其中所有的边和顶点都被访问。，假定边块以列为序的顺序被访问。 边缘被访问一次，源顶点数据被读取 P 次，而目的顶点数据被读取和写入一次。因此每轮迭代的 I/O 量是

$$E + P \times V + 2 \times V$$

所以，应该优选较小的 P 以最小化 I / O 量，这似乎与在上文中中讨论的网格划分原理相反，较大的 P 可以确保更好的局部性和选择性调度效果。为了解决这个困境，我们在边网格上应用第二级划分，以减少顶点的 I /O 访问。 较高级分区由 Q×Q 边网格组成。 给定指定量的存储器 M 和每个顶点数据 U（包括源和目的顶点）的大小，Q 被选择为满足条件的最小整数

$$\frac{V}{Q} \times U \leq M.$$

正如在上文中提到的，选择 P 以将顶点数据拟合到最后一级缓存中，其容量远小于存储器。因此 P 比 Q 大得多，即 Q×Q 分割比 P×P 分割更粗粒化，使得我们可以通过调整块的访问顺序来对边块进行虚拟分组。

通过分析我们可以得知，这种 2 级分层分区不仅提供灵活性而且提供效率，因为更高级分区是虚拟的并且 GridGraph 能够利用较低级分区的结果，因此不添加更多的实际开销。同时，原始细粒边缘网格的良好属性（例如更多选择性调度机会）仍然可以被利用。

2.2.4 系统具体执行

GridGraph 顺序地流传输每个边块。 在流之前，GridGraph 首先检查每个顶点块的活跃状态。边块在双滑动窗口序列中一个接一个地流动，并且如果块的相应源顶点块是活动的，则将其添加到任务列表。

GridGraph 的计算过程如下：首先，主线程将任务推送到队列，包含发出每个读取请求的文件，偏移和长度。（长度设置为 24MB，与在预处理中以实现完整的磁盘带宽时相同。）其次，工作线程从队列中提取任务，直到空，从指定位置读取数据并处理每个边。

每个边先由用户定义的过滤器函数 F 检查，并且如果源顶点是活跃的，则在该边上调用 Fe 以将更新应用到源或目标顶点上。需要注意，用户如果将更新同时应用于源和目标顶点，这可能使得存储器映射向量遭受意外写回到慢级存储。

3 实验测试与结论

3.1 评价

我们通过与 d2.xlarge4 和 i2.xlarge 实例上的 GraphChi 和 X-Stream 的最新版本进行比较来评估 GridGraph 的处理性能。

对于每个系统，我们在 4 个数据集上运行 BFS，WCC，SpMV 和 Pagerank：LiveJournal，Twitter，UK 和 Yahoo。所有图都是具有幂律度分布的真实世界图。 LiveJournal 和 Twitter 是社交图，显示每个在线社交网络中的用户之间的以下关系。 英国和雅虎是由网页之间的超链接关系组成的网络图，其具有比社交图更大的直径。表 2 显示了每个图的幅度，以及我们对 P 的选择。 对于 BFS 和 WCC，我们运行它们直到收敛，即没有更多的顶点可以找到或更新;对于 SpMV，我们运行一次迭代来计算乘法结果;对于 PageRank，我们在每个图上运行 20 次迭代。

Dataset	V	E	Data size	*P*
LiveJournal	4.85M	69.0M	527MB	4
Twitter	61.6M	1.47B	11GB	32
UK	106M	3.74B	28GB	64
Yahoo	1.41B	6.64B	50GB	512

Table 2: Graph datasets used in evaluation.

	i2.xlarge (SSD)				d2.xlarge (HDD)			
	BFS	**WCC**	**SpMV**	**PageR.**	**BFS**	**WCC**	**SpMV**	**PageR.**
LiveJournal								
GraphChi	22.81	17.60	10.12	53.97	21.22	14.93	10.69	45.97
X-Stream	6.54	14.65	6.63	18.22	6.29	13.47	6.10	18.45
GridGraph	2.97	4.39	2.21	12.86	3.36	4.67	2.30	14.21
Twitter								
GraphChi	437.7	469.8	273.1	1263	443.3	406.1	220.7	1064
X-Stream	435.9	1199	143.9	1779	408.8	1089	128.3	1634
GridGraph	204.8	286.5	50.13	538.1	196.3	276.3	42.33	482.1
UK								
GraphChi	2768	1779	412.3	2083	3203	1709	401.2	2191
X-Stream	8081	12057	383.7	4374	7301	11066	319.4	4015
GridGraph	1843	1709	116.8	1347	1730	1609	97.38	1359
Yahoo								
GraphChi	-	114162	2676	13076	-	106735	3110	18361
X-Stream	-	-	1076	9957	-	-	1007	10575
GridGraph	16815	3602	263.1	4719	30178	4077	277.6	5118

Table 1: Execution time (in seconds) with 8GB memory. "-" indicates that the corresponding system failed to finish execution in 48 hours.

GraphChi 以异步模式运行所有算法，并且当顶点的数量足够小时使用内存中优化，使得顶点数据可以分配并保持在存储器中，因此边缘在计算期间不被修改，这有助于在 Twitter 和 UK 图表上的性能。

表 1 显示了所选算法在不同图形和系统上的性能，内存限制为 8GB，以说明适用性。在此配置下，只有 LiveJournal 图形可以适合内存，而其他图形需要访问磁盘。我们可以看到 GridGraph 在基于硬盘的 d2.xlarge 和基于 SSD 的 i2.xlarge 上的性能优于 GraphChi 和 X-Stream。

实例，并且性能不变化很大，除了对于在计算期间经历大量搜索的 Yahoo 上的 BFS，因此使得 SSD 比 HDD 更有利。实际上，有时在 d2.xlarge 实例上可以实现更好的结果，因为 d2.xlarge 上的 3 个 HDD 的峰值顺序带宽略大于 i2.xlarge 上的 1 个 SSD 的峰值顺序带宽。

在个系统的磁盘带宽使用情况方面，X-Stream 和 GridGraph 可用于利用高顺序磁盘带宽，而 GraphChi 不那么理想，因为在许多分片上进行更多的片段化读取和写入。GridGraph 尽量减少写入量，因此更多的 I/O 花在读取，而 X-Stream 必须写更多的数据。

对于所有顶点在计算中都是活动的算法，如 SpMV 和 PageRank（因此每个边缘都是流式的），GridGraph 具有显着减少完成计算所需的 I/ O 量。由于双滑动窗口的顶点访问模式很好，顶点更新聚集在一个块中，这在实际使用中可能是一个非常有用的功能。对于所有顶点数据都可以缓存在内存中的图形，无论需要多少次迭代，都需要在第一次迭代中读取一个磁盘，在最后一次迭代后写入一个磁盘。

对于只有整个顶点集的一部分参与一些迭代（例如 BFS 和 WCC）的计算的迭代算法，GridGraph 可以受益于网格划分的另一个良好属性，我们可以使用选择性调度来跳过大量无用的读取。综上所知，GridGraph 可以在有限资源的大规模现实世界图表上表现良好。 I/O 量的减少是性能增益的关键。

同时论文还从其它几个方面评价了系统：在预处理成本方面，GridGraph 在预处理时间的性能优于 GraphChi， GridGraph 使用基于轻量级范围的分区，因此只需要在输入边列表上进行一次顺序读取，并且只需要对 P×P 边缘块文件进行仅追加的顺序写入，这可以由操作系统很好地处理；在分区的粒度方面，GridGraph 中使用的 2 级分层分区对于实现内存和 out-of-core 场景的良好性能是非常重要的；在可扩展性方面，通过观察增加更多硬件资源时的改进来评估 GridGraph 的可扩展性，得出了更强大的硬件资源可以利用 GridGraph 扩展的结论；在与分布式系统比较方面，当更强大的硬件可用时，GridGraph 甚至与分布式系统竞争。

3.2 结论

在本文中，系统 GridGraph 通过分割顶点和边缘分别为 1D 块和 2D 块，通过一个基于轻量级范围的重排有效地生产。将第二逻辑级分区应用于该网格划分，并且适应于存储器内和非内核场景。

GridGraph 使用一个新的流应用模型，依次流边缘，并立即应用更新到顶点。通过以局部友好的方式为边缘流化边缘块，GridGraph 能够访问存储器中的顶点数据，而不涉及 I / O

访问。此外，GridGraph 可以跳过不必要的边块。因此 GridGraph 实现了比先进的 out-of-core 图形系统（如 GraphChi 和 X-Stream）明显更好的性能，并且可以在 HDD 和 SSD 上工作。在某些情况下，GridGraph 甚至比需要更多资源的主流分布式图形处理系统更快。

GridGraph 的性能主要受 I / O 带宽的限制。

4 阅读体会

通过这篇高水平论文，我第一次接触到大图计算领域。众所周知，图可以用来表征不同实体间复杂的依赖关系。因而，在许多实际的应用当中，如社交网络分析、网页搜索、商品推荐等都可以使用图来进行问题的建模和分析。以社交网络为例，众多社交平台的用户量以亿来计算，用户之间的数量关系更加复杂，因此大图计算不仅是计算密集型，同时也是存储密集型问题，如何在可以接受的时间内对大图进行计算，是需要解决的难题。

目前处理图计算机的系统主要由两种：第一种是使用分布式并行计算的系统，典型的有 PowerGragh、GraphX 等，这些分布式系统大部分采用 “think like a vertex”的思想，即以点为中心(vertex-centric)的计算模型,模型中的所有的点从其入边的邻点获取数据，执行用户自定义的函数对自己的状态进行更新,然后将自己的更新状态通过消息发给其出边的邻点。虽然这些系统具有较好的扩展性，但是处理图计算的效率太低，对系统的性能造成较大的影像。第二种是在单台机器上进行大图计算的单机系统，其特点是编程和计算模型简单，硬件开销很低，但是计算能力有限，无法满足某些计算需求。从计算模型来看，现在大图计算的计算模型主要分为两种：以点为中心的计算模型和以边为中心的计算模型。在分布式处理系统 Pregel、GraphLab 等以及单机系统 GraphChi 主要使用了以点为中心的计算模型，这种计算模型更易于编程和理解，以边为中心的计算模型主要用于单机的系统，如 X-Stream。除了这两种主要的计算模型之外，还有一些系统从数据的局部性出发，提出一些新的计算模型来提升系统的性能，但从本质上来说，这些计算模型是基于以点为中心的计算模型，只是针对数据的布局，做出了相应的修改。

尽管已经有了很多针对大图计算的研究，目前已经有科研人员提出了新的研究点。在分布式系统方面，可以设计更效率的图划分方法，保证机器负载合理；然后，容错也是值得改善的重要性能，我们在减少容错的同时尽可能提高错误恢复的速度。在单机系统方面，我们可以发挥计算机多核的特点，使得 I/O 和计算并行，并且提高计算时的并行度。