分布式图神经网络系统面临的挑战以及研究进展

程健[1)]

[1)](华中科技大学计算机科学与技术学院，武汉市 中国 430074)

摘 要 随着图数据量的指数级增长，单机已经不能满足如此大的计算需求了。近年来，分布式图神经网络一直是热门话题。然而，分布式图神经网络面临着通信开销大，模型精度下降和负载均衡的问题。新的 GNN 模型致力于解决以上的问题，从而提升分布式图神经网络的训练效率。分布式图神经网络可以分为单机多 GPU 系统，GPU 集群系统，CPU 集群系统，分别在不同的层面加速 GNN 训练效率，使得大规模的图计算成为可能。

关键词 图神经网络；分布式系统；CPU 集群；GPU 集群

Challenges and Research Progress of Distributed Graph Neural Network System

Jian Cheng[1)]

[1)](Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract With the exponential growth of the amount of graph data, a single machine can no longer meet such a large computing demand. Distributed graph neural networks have been a hot topic in recent years. However, distributed graph neural networks face the problems of high communication overhead, model accuracy degradation and load balancing. The new GNN model is dedicated to solving the above problems, thereby improving the training efficiency of distributed graph neural networks. Distributed graph neural networks can be divided into single-machine multi-GPU systems, GPU cluster systems, and CPU cluster systems, which accelerate GNN training efficiency at different levels, making large-scale graph computing possible.

Key words Graph neural network; Distributed Systems; CPU cluster; GPU cluster

1 背景介绍

图神经网络主要是处理图结构数据，有着广泛的应用，包括社会网络，垃圾邮件检测，社会网络分析，生物信息学，药物发现，交通预测，自然语言处理等等。通过将图结构的信息集成到深度学习模型中，GNN 可以比传统的机器学习和数据挖掘方法取得明显更好的结果。

GNN 模型通常包含多个图卷积层，其中每个顶点聚合其邻居的最新状态，更新顶点的状态，并将神经网络应用于顶点状态的更新。以传统的图卷积网络（GCN）为例，在每层中，顶点使用求和函数聚合邻居状态及其自身状态，然后应用单层 MLP 来转换新状态。如果图的层数为 L，则此类过程重复 L 次。第 L 层生成的顶点状态被下游任务使用，如节点分类、链接预测等。在过去的几年里，许多研究工作在图神经网络模型的设计方面取得了显著进展。比较有影响的模型有 GCN, GraphSAGE, GAT, GIN 和一些特定应用的 GNN 模型。为了在现有的框架上（比如 pytorch, tensorflow）加速 GNN 模型的运行速度，提高收敛的速度，近年来涌现了许多优化的方案，包括 GNN 计算内核优化，高效的编程模型，充分利用

硬件性能等等。然而，这些框架和优化大多集中与单机上的 GNN 训练优化，没有考虑图的可拓展性。

如今，由于大图数据的普遍存在，大规模的图神经网络成为热点话题。一个拥有几十亿个顶点和几万亿条边的图变得常见，比如微博，微信，推特，脸书里的社会网络。然而，许多现有的图神经网络模型只能处理比较小的图数据，当图数据的规模很大的时候，这些模型就无法处理或者性能和效率很低。这是因为以前的 GNN 模型很复杂，在处理大图时需要大量的计算资源。可以通过设计可扩展的 GNN 模型来实现大规模的图形神经网络，也可以在 GNN 训练的时候采用分布式计算的方式。因为在处理大图形时，单个设备的有限内存和计算资源成为大规模 GNN 训练的瓶颈，分布式计算提供了更多的计算资源（比如多 GPU、CPU 集群等）来提高训练效率。

2 分布式 GNN 训练和面临的挑战

2.1 分布式GNN训练过程

分布式 GNN 的训练过程可以分成三个部分，数据划分，GNN 模型优化和梯度聚合。如图 1 所示。

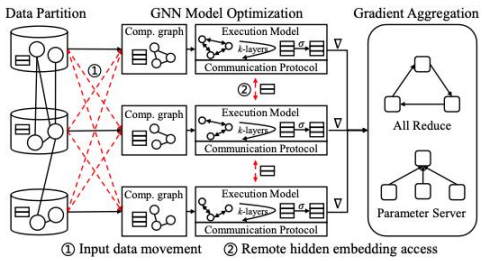

图 1 分布式 GNN 训练过程

数据划分。这是一个实现分布式训练预处理阶段。它将输入数据（即图和特征）分配给一组运行中的程序。由于 GNN 中的训练数据是相互依赖的，数据分区阶段比传统的分布式机器学习阶段变得更加复杂。如图 1 所示，分区数据（即子图）之间的跨程序边缘意味着数据依赖性。如果我们考虑到分区之间的数据依赖性，分布式训练效率就会因通信而降低；如果我们干脆忽略数据依赖性，模型的准确性就会被破坏。因此，数据分区是端到端分布式 GNN 训练效率的关键阶段。

GNN 模型优化。这是分布式 GNN 训练的核心阶段，它执行 GNN 模型的训练逻辑（即正向计算和反向计算）。进一步将这个阶段分为计算图生成、执行模型和通信协议。首先，每个 worker 从分区输入图和特征生成计算图，然后使用计算图执行 GNN 模型来计算损失和梯度。然而，由于数据之间的依赖性，计算图生成和 GNN 模型执行与传统的深度学习模型大不相同。如果不访问远程输入数据，可能无法正确生成小批量训练策略的计算图。执行模型涉及 GNN 模型的 k 层图聚合，聚合表现出不规则的数据访问模式。在分布式全图训练中，每个图层中的图聚合需要通过通信协议访问顶点远程邻居中的隐藏特征，还应考虑图层之间的同步模式。因此，与传统的分布式机器学习相比，由于 GNN 中的数据依赖性，计算图生成和模型执行变得更加复杂。

梯度聚合。这个阶段负责汇总最新的局部梯度，以获得全局梯度并更新模型参数。在 GNN 的环境下，模型大小通常很小，模型更新逻辑与其他机器学习模型相同。经典分布式机器学习中现有的梯度聚合技术可以直接应用于分布式 GNN 训练。

2.2 分布式GNN训练的挑战

由于图数据之间的依赖性，高效地分布式训练 GNN 面临这以下困难。

大量的特征通信开销。对于分布式 GNN 训练，在 GNN 模型优化阶段会发生大规模特征通信。当 GNN 模型以 mini-batch 处理方式训练时，计算图生成（即 mini-batch 构建）需要访问远程图和特征，如图 1 中的操作 1，以创建本地 batch。在进行多 GPU 训练的时候，会有大量的 feature 从 CPU 传输到 GPU 当中，造成巨大的通信开销。研究表明，在端到端训练期间，mini_batch 制造的成本成为 GNN 训练的瓶颈。当 GNN 模型以全图方式训练时，计算图可以与没有通信的分区图相同，但是，每层中的图聚合都需要访问顶点远程邻居中的隐藏特征（图 1 中的操作 2），这会导致巨大的隐藏特征通信开销。总之，GNN 模型的分布式 mini_batch 训练和分布式全图训练都受到大规模特征通信的影响。

模型精度的下降。Mini-batch 的 GNN 训练比全图训练更具可扩展性，因此它被大多数现有的分布式 GNN 训练系统采用。然而，随着模型深度的增加，mini-batch 训练存在邻居爆炸问题。当模型有 L 个图卷积层的时候，mini-batch 需要获取 train vertax 的 L 跳邻居，随着 L 的增大，邻居数量会成指数级增长。解决方案是通过采样或忽略跨 worker 通信来构建近似的 mini-batch。虽然近似的 mini-batch 提高

了训练效率，但理论上不能保证模型收敛。因此，对于分布式 mini-batchGNN 训练时，我们需要在模型准确性和训练效率之间进行权衡。

负载均衡。工作负载均衡是分布式计算中的一个内在问题。然而，GNN 模型的各种工作负载特征增加了在 worker 之间划分训练负载平衡的难度。因为很难以简单和统一的方式对 GNN 工作负载进行建模。没有正式的成本模型，经典的图分区算法就无法用于平衡 worker 之间的 GNN 工作负载。此外，分布式 mini-batch 的 GNN 训练要求每个 worker 处理相同批次大小，而不仅仅是平衡子图中的顶点数量。总之，在分布式环境中训练 GNN 时，很容易遇到负载不平衡，从而导致 worker 之间相互等待并降低训练效率。

3 分布式 GNN 训练系统

3.1 单机多GPU

GPU 是训练神经网络模型的最强大的计算资源。分布式 GNN 训练的一个常见解决方案是在一台机器上使用多 GPU。

3.1.1 NeuGraph

NeuGraph 提出了一种新的框架，根据图神经网络是将标准神经网络与迭代图传播结合起来的这一特点，NeuGraph 在数据流中引入以顶点为中心的消息传递模型，将图模型和数据流模型结合，来支持并行图神经网络计算。NeuGraph 还将图计算的优化方法如数据分区、调度，引入到了数据流框架中，来支持高效的图神经网络训练。NeuGraph 提出了一种新的处理模型 SAGA-NN，它将数据流和顶点编程模式相结合来表示图神经网络的计算。 SAGA-NN 将前向计算分为 4 个阶:Scatter、ApplyEdge、Gather 和 ApplyVertex。如图 2 所示。

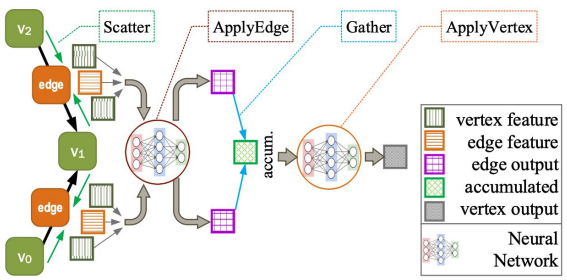

图 2 SAGA-NN 模型

ApplyEdge 和 ApplyVertex 提供了两个用户定义函数，供用户在边和顶点上声明神经网络计算。 ApplyEdge 函数定义每个边上的计算，以 edge 和 W 作为输入，其中 edge 是指边数据，W 包含图神经网络模型的可学习参数。 ApplyVertex 函数定义了顶点的计算，它以顶点张量、顶点聚合累积量和可学习参数 W 作为输入，并在应用神经网络模型后返回新的顶点表示。Scatter 和 Gather 执行数据传播和收集，由系统隐式触发和执行。 SAGA-NN 中的顶点程序采用以顶点为中心的编程模型来表达图神经网络的计算，对图神经网络中的通用阶段进行建模，并在图计算和数据流调度中实现优化。NeuGraph 在数据流抽象的基础上引入了特定的图分区方法，可以解决 GPU 内存的物理限制问题。如图 3 所示。

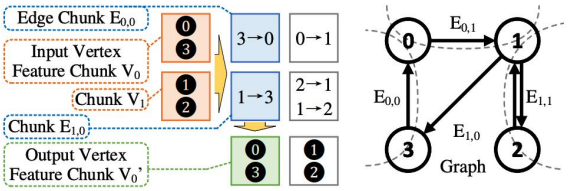

图 3 图的 2D 分区，P=2

通过 2D 图分区方法，NeuGraph 将顶点数据分割成 P 个大小相等的不相交顶点块，并将邻接矩阵分为 P×P 个边块。通过将图数据分割成块，在逐个处理边块信息时，只需要边块所对应的源顶点块和目标顶点块即可。在训练过程中，与顶点块或边块相关的一些中间特征数据将用于反向传播。为了节省 GPU 内存，它们在前向计算期间交换到主机内存，在反向传播期间交换回。NeuGraph 为降低主机和 GPU 内存之间的数据传输做了一系列优化: 在处理边块 E 时，NeuGraph 设计了一个过滤器，来过滤每个顶点块内的必要顶点，并将其传输到 GPU 中; 通过一种局部感知的图划分算法，NeuGraph 将连接同一顶点的边尽可能地压缩到一个块内，通过这种方法，NeuGraph 可以获得更好的顶点数据访问局部; NeuGraph 设计流水线调度进一步重叠数据传输和计算，以隐藏传输延迟。

3.1.2 PaGraph

当图数据庞大的情况下，在 GNN 训练的过程中，顶点的特征从 CPU 加载进 GPU 所需要的时间比计算的时间还多，而 GPU 当中内存的利用率很低，不足 20%。基于这种情况，PaGraph 提出了计算感知的缓存策略，应用于 GNN 训练当中。

Graph Store Server 用于缓存图的结构信息以及顶点的特征，在程序运行开始的时候 CPU 中。全图在预处理阶段会被分割成 GPU 数量的子图，每

个子图包含几乎均等的 train vertax 数量，并且获取这些 train vertax 的 L 跳邻居顶点，并构成完整的子图信息。L 为 GNN 的卷积层数。图 4 中灰色顶点即 train vertax,白色顶点是其 L 跳的邻居。GPU 训练的时候生成子图的 mini-batch，从 GPU 中获取 mini-batch 的特征向量进行训练。每个 epoch 结束的时候，使用 ring-allreduce 同步 GPU 之间的梯度。

PaGraph 的整体结构如图 4 所示。

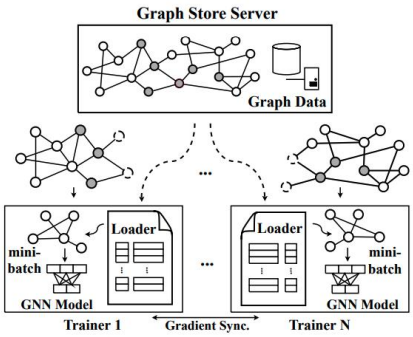

图 4 PaGraph 整体结构

由于 GPU 的内存利用率低，PaGraph 使用静态缓存的方式，将顶点的特征预先加载进 GPU，避免反复地加载顶点特征。缓存情况下的数据加载方式如图 5 所示。

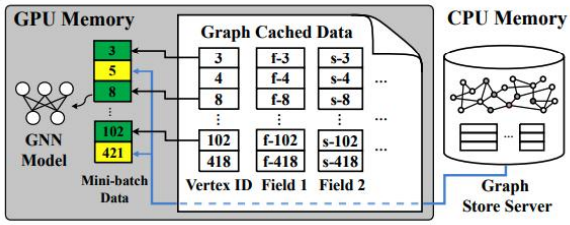

图 5 PaGraph 数据加载方式

PaGraph 采用静态缓存的方式来决定 GPU 内缓存的内容。缓存的时机在第一轮迭代训练结束之后。在获得第一个训练子图结构时，Cacher 还没有缓存图数据，将完全从全局图存储服务中获取所需要的训练数据，并开始记录 GPU 的内存资源使用情况。在第一轮迭代训练完成后，PaGraph 在获得训练过程中 GPU 所需要的内存空间，并将剩余不被利用的 GPU 内存空间来作为缓存区域。PaGraph 根据当前工作子图的顶点出度，对顶点进行排序，并按照顶点出度从小到大的顺序依次将顶点数据缓存入 GPU 内存中，直至完全缓存或达到分配的缓存空间上限。在随后的每一步训练过程中，PaGraph 通过同时访问 GPU 缓存部分和 CPU 全局图存储服务来完成数据加载过程。

对于大规模图，GPU 内存不可能完全按缓存所有的图数据，因此，数据加载阶段依然制约着 GNN 训练速度。GNN 的训练过程中，不同轮次迭代之间的数据加载时间和模型计算时间几乎相同，因此，可以牺牲一部分缓存空间，在做当前 mini-batch 的模型计算时预加载后续 mini-batch 的数据，以此将数据加载过程隐藏在模型计算阶段中。

对于大图，面临训练性能和单机资源限制，可以进行图分割，采用多机多 GPU 的数据并行分布式训练。在不破坏算法性质的情况下，图采样过中需要访问被采样顶点的所有邻居顶点。子图划分的算法如下所示。

Input: graph $\mathcal{G}$, train vertex set TV, number of hops L, partition number K
Output: graph partition $\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_k$

for $i \leftarrow 1$ *to* K **do**
  $\mathcal{G}_i \leftarrow \varnothing$ // Initialization
for *each train vertex* $v_t \in TV$ **do**
  $IN(v_t) \leftarrow$ IN-NEIGHBOR$(v_t, \mathcal{G}, L)$
  $score_{v_t} \leftarrow$ SCORE$(v_t, IN(v_t))$
  $ind \leftarrow \arg\max_{i \in [1,K]} \{score_{v_t}^{(i)}\}$
  $\mathcal{G}_{ind} \leftarrow \mathcal{G}_{ind} \cup \{v_t\} \cup \{IN(v_t)\}$
return $\{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_K\}$

PaGraph 根据图神经网络模型的层数，能确定采样获得的训练子图所需要的层数，即一个顶点最多可能访问的跳数信息。为了避免跨子图访问所带来的额外开销，PaGraph 根据跳数信息在传统图分割的基础上引入最少冗余顶点，来完全避免跨子图访问。

与 DGL 相比，PaGraph 通过缓存和图分区策略加快了 GNN 训练速度，最终实现了高达 96.8%的数据传输和高达 16 倍的性能改进。

3.1.3 GNNLab

对于一个 GNN 模型来说，其输入的图数据包含图结构数据（graph topological data）和图特征数据（graph feature data）两部分。随着图数据规模的不断增大，如何高效地训练 GNN 成为一个难题，目前在大规模的数据集上普遍采用基于图采样的 mini-bacth 训练方法。该方法包含 Sample、Extract 和 Train 三个阶段，也被称为 SET 模型。此方法将所有的训练节点划分为若干个小的 mini-bacth，对于每个 mini-batch，我们首先在 Sample 阶段利用图采样算法为每个顶点选取若干个邻居构成一个 sample，然后在 Extract 阶段在为 sample 中的每个顶点抽取对应的特征后进入 Train 阶段，即进行模型训练。整个训练流程不断重复上述过程，直到模型的准确率达到期望值。图 6 展示了在一个两

层 GNN 模型上训练顶点 7 时 SET 模型的执行步骤。

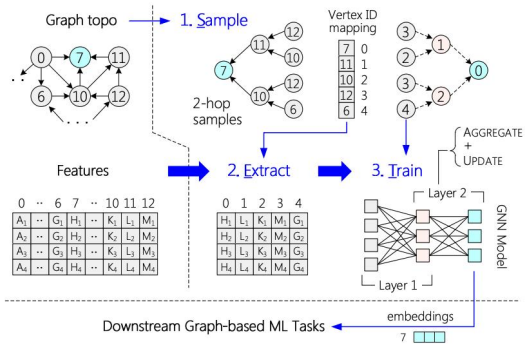

图 6 SET 模型执行步骤

与传统的深度学习一样，GPU 已经广泛应用于 GNN 训练的加速。GPU 最早被应用于 Train 阶段的加速，DGL、PyG 等 GNN 训练系统针对 GPU 的硬件特性实现了高度优化的 NN runtime。在 Train 阶段引入 GPU 后，Sample 和 Extract 阶段在端到端的训练时间中占据了绝大部分，这也导致了 GPU 的资源利用率很低，因而最近工业界和学术界也开始尝试利用 GPU 加速 Sample 和 Extract 阶段。

在很多情况下，图数据需要很大的存储空间的原因并不是图结构数据本身，而是图中的顶点和边带有丰富的特征数据。尽管 GPU 的内存容量相较于 host 的内存小了很多，但其已经足够存储很多大规模图数据的图结构。基于这个观察，如下图所示，DGL、NextDoor 等系统将图结构直接载入 GPU 内存，利用 GPU 对 Sample 阶段进行加速。

现有的基于 GPU 的 GNN 系统（例如 DGL）通常采用时间共享（time sharing）的设计思想，即一个 GPU 需要同时负责执行 Sample、Extract 和 Train 三个阶段的任务。为了在 GPU 上执行 Sample 阶段，需要将图结构数据载入 GPU 内存中。由于 GPU 的内存容量十分有限，图数据所占用的存储空间会使得留给 cache 可用的内存空间大大减小，而 cache 空间的减小会大大降低 cache 的命中率。如下图所示，在 16GB V100 GPU 上，将图结构数据载入 GPU 内存后，留给 cache 可用的空间会从 11.4GB 减小至 3.7GB，从而导致 Extract 阶段执行的时间增长 2.1 倍。

3.2 GPU集群

GPU 集群是具有多 GPU 的单台机器的扩展，当图形和输入功能太大而无法放入一台机器时，它提供了多机多 GPU。对于 GPU 集群，应该考虑 CPU 和 GPU 之间的通信以及机器之间的通信。以下系统已根据其原始实验在 GPU 集群上进行了测试。

3.2.1 P3

P3 系统设计了 Pipelined Push-Pull 的模式。将图的结构和 feature 分别存储，并且将 feature 按列切分。假设有 N 个 worker，将顶点的 feature 分成 N 等分，每个 worker 分到 feature/N 维的 feature。可以大大减少 feature 加载进 GPU 的时间。划分方式如图 7 所示。

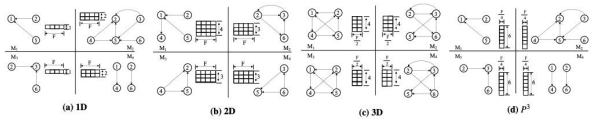

图 7 P3 的 partition 方式

采用正常的 pull base 的机制。从起始 seed node 开始，每次向外拉取之间相连的邻居的 node，重复 K 次，构建整体 K-hop neighborhood 信息，不过，这里只需要拿到对应的图结构信息，并不需要拉取对应的 feature，所以整体的网络消耗会非常低。而为了避免正常的拉取全部 feature 所带来的巨大的网络消耗，这里采用了 hybrid parallelism approach（model parallelism + data parallelism), P3 将其命名为 push-pull parallelism。

P3 的执行步骤如图 8 所示。

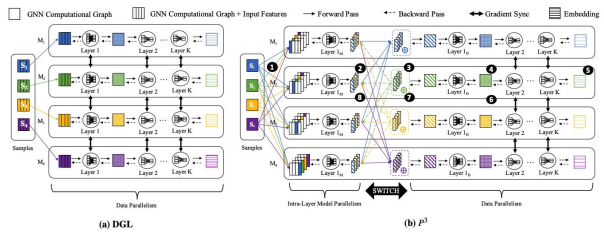

图 8 P3 执行步骤

将每台机器生产 layer 1 的 computation graph 推到所有 worker 上去。因为将 feature 均匀分布在不同机器上，这里每台机器拿到 layer 1 的 computation graph 以后就可以开始进行前向传播，拿到一个 partial activations 结果。 这里就可以类比于 model parallel, 因为每台机器只有 F/N 维的 feature，所以相当于只用到了第一层网络的(F/N * Hidden size) 个参数。这里因为没有什么网络消耗，所以 GPU 的使用率(非空闲时间/总时间)也大大提高。然后将所有在其他机器的 partial activations 结果 aggregate 起来，做一个 reduce 操作，就得到了对应的第一层输出。当然可能也存在一些不可聚合的函数，例如 ReLU 函数。所以这里后面还需要接一个 Layer 1D，进一步的转化结果。后面就可以进行正常的 data parallelism。最后经过后面 K-1 层网络，得到最终

的 embeding，计算出 loss，完成前向传播。然后到 layer 1D 之前，完成正常的 data parallelism，更新对应的网络参数。之后 push 对应的 error gradient 到所有机器，又切换回 model parallelism。然后进行本地对应的模型参数更新。

之前的一些设计都是为了减少网络消耗的时间，为了更好的 overlap 通信和计算过程，P3 采用了类似于 PipeDream 的 pipeline 机制。如图 9 所示。

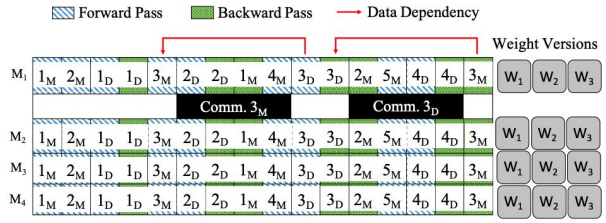

图 9 P3pipeline 策略

以让两个 forward 阶段和两个 backward 阶段同时进行。

但是，当 hidden_size 比较大的时候，通信开销会比较大，在这个情况下性能不佳。

3.2.2 BNS-GCN

BNS-GCN 的核心思想是在每个 epoch 训练前，随机选择边界点的一个子集，存储和通信只发生在这个子集上，而不是原来整个边界点集。

BNS, Boundary Node Sampling，首先对图进行分区，使得边界点尽可能少，然后对边界点进行随机采样，进一步减少通信和内存开销。

分布式 GCN 训练的过程如图 10 所示。

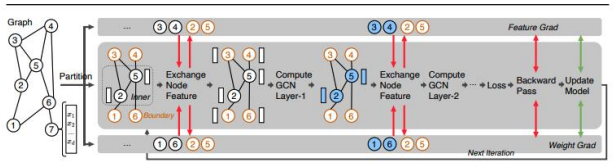

图 10 BNS 训练过程

首先需要对图进行切分，将每个子图分配到不同的 worker 上，在做聚合邻居前，需要先通过通信获取远程依赖节点的 feature，然后进行聚合操作和顶点更新操作得到下一层的顶点表示，这样就完成了一层的 GCN 计算，之后每一层的计算都是类似的过程，最后计算 loss 进行反向计算，反向计算逻辑和前向计算基本类似，最后通过 AllReduce 进行参数的更新。

BNS-GCN 的具体算法步骤如下：

输入是每个分区上的子图信息以及对应 feature 和 label。首先根据边界点得到 Inner Nodes，对于每个 epoch，根据采样概率 p 对边界点进行采样得到训练的子图，然后通过 broadcast 的方式将采样到的节点信息通知到其他节点；节点收到来自不同分区的采样信息后，对 Inner Nodes 取交集进而确定向不同分区发送的 feature；在具体训练过程中，首先通过通信获取远程节点的 feature，然后在本地进行前向计算；反向计算逻辑类似，将所需的梯度发送到对应的分区进行反向计算；最后通过 AllReduce 进行参数的更新。

由于其简单的策略，BNS-GCN 可以合并到任何基于边缘切割分区的分布式 GNN 训练系统中，而无需引入过度计算。该策略使用 DGL，实现了全图精度，而采样方法加快了训练时间。这两种基本 GNN 加速技术，即分布式 GNN 和基于采样的 GNN 的组合。

3.3 CPU集群

与 GPU 集群相比，CPU 集群易于扩展且经济。现有的大数据处理基础设施通常使用 CPU 集群构建。有许多工业级分布式 GNN 培训系统在 CPU 集群上运行。

3.3.1 ByteGNN

ByteGNN 提供了一套自定义的采样编程接口，不同 GNN 模型将统一用该接口来实现不同的采样逻辑。ByteGNN 基于该接口自动地将采样逻辑解析成为一个计算图 DAG。采样变为计算图最大的好处是可以把采样过程抽象成多个小的任务，没有前后依赖的任务直接可以并行且形成流水线。如图 11 所示。

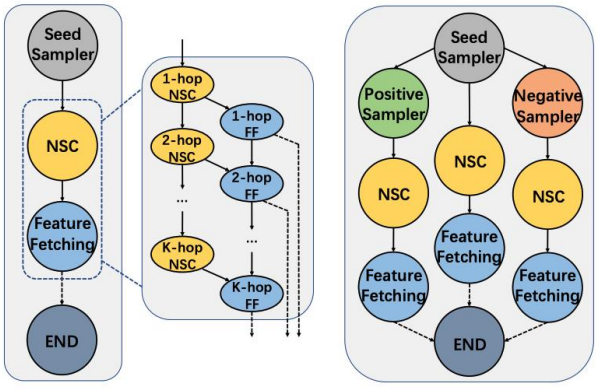

图 11 采样工作流的 DAG

通过中间调度层，将采样计算图和训练计算图软连接到了一起，通过一个调度器动态地平衡采样与训练这两个阶段的资源消耗和消费速率，使得系统的 CPU 利用率和内存开销达到最优。

ByteGNN 提出新的切图方式。如图 12 所示。

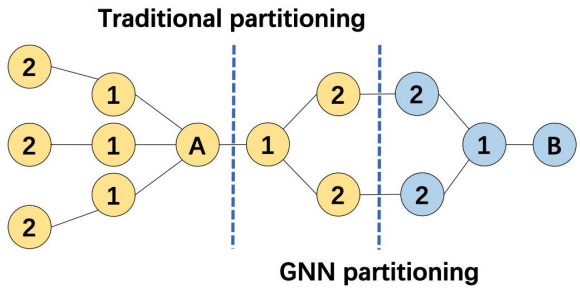

图 12 传统分割与新分割对比

传统切图方法（左边虚线）标准：k 跳邻居尽量在一起且切的边尽可能少，但是传统切图没有考虑图上顶点是否打标签,比如图 7 中 A 和 B 有标签，GNN 算法一般都是从有标签的顶点出发找 N 跳邻居。基于 GNN 采样的 workload 游走特点，作者先随机选取图中 M 个种子顶点进行 BFS 扩散，直到扩散相互碰撞而停止消息传递。这样全图会被分成多个块（右边虚线），使得在 GNN 游走过程中网络开销最优。

4 总结

分布式图神经网络主要解决单机下难以处理的大图问题。目前的系统遇到的困难有通信开销大，模型精度下降，负载不均衡等问题。现有的研究也在探寻解决方案，比如 PaGraph 中预先把数据加载进 GPU 以降低通信开销，NerGraph 降低精度的损失。

随着未来数据量进一步地增加，分布式图神经网络会有更加广阔的应用空间，解决方案也会不断创新。

参 考 文 献

[1] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In 2019 USENIX Annual Technical Conference. 443 – 458.

[2] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. PaGraph: Scaling GNN Training on Large Graphs via Computation-Aware Caching. In Proceedings of the 11th ACM Symposium on Cloud Computing. 401 – 415.

[3] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. 2022. GNNLab: A Factored System for Sample-Based GNN Training over GPUs. In Proceedings of the Seventeenth European Conference on Computer Systems. 417–434.

[4] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed deep graph learning at scale. In 15th USENIX Symposium on Operating Systems Design and Implementation. 551 – 568.

[5] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling. In Proceedings of Machine Learning and Systems. 673–693.

[6] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: efficient graph neural network training at large scale. Proceedings of the VLDB Endowment 15, 6 (2022), 1228–1242.