超大规模图嵌入系统的实现

王正 1)

摘 要 目前最大的公开 multi-relation graphs 有数亿个节点和数万钟关系。例如，Facebook 拥有超过 30 亿用户，为每个用户学习 400 维嵌入将需要能够有效地存储和访问 5 TB 的嵌入参数，远远超过目前机器所能拥有的 CPU 内存容量。此外，使用维度更大的嵌入 vector 已被证明可以提高下游任务的性能。所以由于这两点原因，训练超大规模 Graph Embedding 的是很有必要的

关键词 图嵌入；图神经网络；深度学习；单机系统

1 绪论

1.1 研究背景

随着信息技术的广泛应用和深度学习技术的逐步发展，当今社会的信息化程度日益增加，人们采集的可用数据呈指数级增长，海量的数据存在于人类社会的各个领域，数据之间的结构复杂程度越来越高。因此，研究如何从未经处理的海量数据中提取潜在有用信息具有非常重要的意义，而数据挖掘技术能够从有用信息和冗余信息混合在一起的海量数据中提取出有价值的信息。

图数据也被称为网络数据，是一种构建对象间复杂关系的数据结构。研究表明，网络数据能够较好的表示数据中的复杂关系，还可建立复杂数据间多对多的结构关系。网络数据无处不在，例如，在社交领域中，用网络数据对社交网络中多个用户之间的 好友关系进行建模；药物领域中用网络数据对药物分子中原子之间的化学键关系进行建模；智慧交通领域中用网络数据对地标之间的道路联系进行建模等。网络可以是现实中许多复杂关系的信息载体，在数据时代，将现实中的数据构建成网络是有优势的。 一方面可以储存相关数据，另一方面也保留了数据的相关信息。经过建模，当需要挖掘各个领域中的数据时，可将其转化为研究如何挖网络数据。因此，利用网络研究复杂关系成为了一个新的研究热点。网络表示学习在网络数据上的典型应用包括节点分类、链路预测、社区挖掘和图分类等。具体来说，可以使用化学分子结构预测药物的抗癌性;利用社交网络对标签未知的节点进行分类;利用图中节点的特征聚合进行图分类。

电子商务和社交网络公司今天面临着分析和挖掘具有数十亿个节点和数百亿到数万亿条边的图形的挑战。近年来，一种流行的学习方法是应用网络嵌入技术来获得每个节点的向量表示。这些学习的表示或嵌入可以很容易地在下游的机器学习和推荐算法中使用。这些表示在各种在线服务中广泛使用频繁更新。例如，阿里巴巴的一个核心项目推荐系统拥有数十亿个项目和用户，当新用户和项目都在线时，需要频繁重新嵌入，并且必须快速重新计算基础嵌入。

1.2 研究现状

图嵌入模型训练任务是计算密集型和内存密集型任务，举例来说，假设使用一个 400 维的向量表示一个结点，那么每个结点需要 1600 字节去存储（int 类型），那么对于一个具有五百万结点的社交网络来说，需要 80GB 的内存去存储，然后还需要使用各种优化算法如 SGD、Adam 等算法去优化损失函数，若主机系统内存或显存不够，这就可能会导致 IO-bound 问题, 即完成计算所需的时间主要由等待 IO 操作完成所花费的时间决定。

因此，若训练任务所需要的内存或显存与主机的内存和显存不匹配，那么就需要大量的数据移动开销，这会大大降低整个系统的资源利用率并且降低训练速度。

目前最大的公开 multi-relation graphs 有数亿个节点和数万钟关系。例如，Facebook 拥有超过 30 亿用户，为每个用户学习 400 维嵌入将需要能够

有效地存储和访问 5 TB 的嵌入参数，远远超过目前机器所能拥有的 CPU 内存容量。此外，使用维度更大的嵌入 vector 已被证明可以提高下游任务的性能。所以由于这两点原因，训练超大规模 Graph Embedding 的是很有必要的

2 相关研究

2.1 DGL-KE

DGL-KE 引入了各种创新优化，通过多进程处理、多 GPU 和分布式并行，加快了对具有数百万节点和数十亿边的 graph 的训练。这些优化旨在增加数据局部性，减少通信开销，使计算与内存访问重叠，并实现高操作效率。在由 8600 万多个节点和三亿条边组成的 graph 上的实验表明，DGL-KE 可以在具有 8 个 GPU 的 EC2 上在 100 分钟内计算 graph embedding，在具有 48 个核心的 4 台机器的 EC2 集群上在 30 分钟内计算嵌入式。

DGL-KE 将 node embedding parameter 保存在 CPU 内存里，而将 relation embedding parameter 保存在内存里。在每个 mini-batch 的训练中，需要将 node embedding parameter 从内存拷贝到显存中进行计算，计算结束后再将结果从显存拷贝回内存。这类系统的问题在于每个 mini-batch 的训练过程都是同步的，而大量的时间花费在数据的传输上。

DGL-KE 之所以能够有这样的性能，主要是因为采用了许多创新的系统和算法优化, 如图 1 所示，为 DGL-KE。

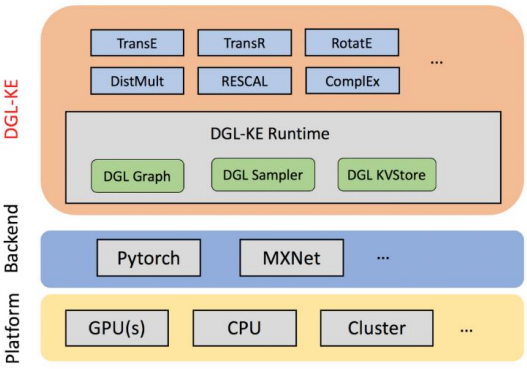

图 1 DGL-KE 架构图

对超大规模的图数据进行训练，分布式训练必不可少。DGL-KE 的思路主要是将原始大图分割成不同的子图，每一台机器负责在一个子图上进行随机梯度下降训练，所有机器之间通过参数服务器（Parameter Server）进行模型的同步。其架构如图 2 所示：

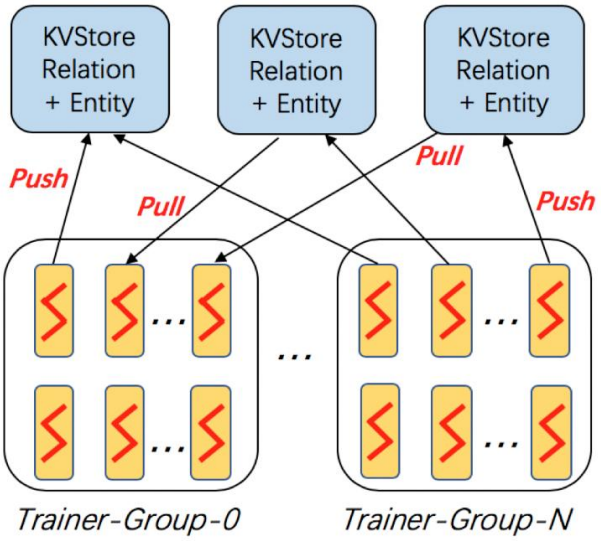

图 2 DGL-KE 分布式架构图

然而，如果只是对一张大图进行随机切割，会造成训练机器与参数服务器之间的数据通信量巨大（本地机器需要从远程机器去请求自己所需的模型数据），从而造成网络瓶颈。为了解决这一问题，DGL-KE 在训练前会预先通过 METIS 图分割算法对原始数据进行切割。METIS 算法可以节省将近 90% 的模型网络传输带宽，从而使分布式训练达到线性加速比。

除了以上优化之外，DGL-KE 还提供了其他若干优化方法。例如，使用 Joint Negative Sampler 加速负采样过程，使用 Relation Partition 来减少训练过程中的数据拷贝，以及使用 Periodic synchronization 保证模型的收敛等。

2.2 PyTorch-BigGraph

Facebook 提出了一种可高效训练包含数十亿节点和数万亿边的图模型的框架 BigGraph 并开源了其 PyTorch 实现。

PBG 将所有的 node 均匀的划分成 P 个 partition，将内存作为外层的一个 buffer，并每次将 k 个 partition 加载到内存中作为一个 mini-batch 进行计算, 计算完成后再将剩余的 partition 与内存中的数据进行 swap，最终直到遍历过所有的边位置。这类系统的问题在于 swap 的开销是巨大的，且 swap 的过程中计算资源是空闲的。通过将图结

构分区为随机划分的 P 个分区，使得可将两个分区放入内存中，PBG 解决了传统图嵌入方法的一些短板。如图 3，一条边的起点在分区 p1，终点在分区 p2，则它会被放入 bucket (p1, p2)。然后，在同一模型中，根据源节点和目标节点将这些图节点划分到 P2 bucket。完成节点和边的分区之后，可以每次在一个 bucket 内执行训练。bucket (p1, p2) 的训练仅需要将分区 p1 和 p2 的嵌入存储到内存中。PBG 结构能保证 bucket 至少有一个之前已训练的嵌入分区。

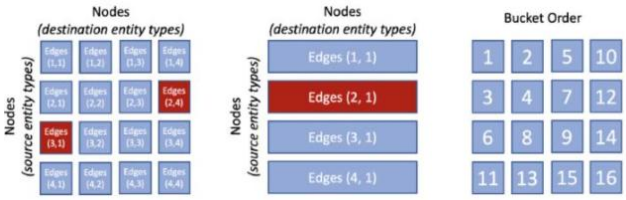

图 3　PBG 示例

PBG 的另一大创新是训练机制的并行化和分布式。PBG 使用 PyTorch 自带的并行化机制实现了一种分布式训练模型，这用到了前面描述的模块分区结构。在这个模型中，各个机器会协调在不相交的 bucket 上进行训练。这会用到一个锁服务器（lock server），其负责将 bucket 分派给工作器（worker），从而尽可能地减少不同机器之间的通信。每台机器都可以使用不同的 bucket 并行地训练模型。

如图 4 中，机器 2 中的 Trainer 模块向机器 1 上的锁服务器请求了一个 bucket，这会锁定该 bucket 的分区。然后该 trainer 会保存它不再使用的所有分区并从共享分区服务器载入它需要的新分区，此时它可以将自己的旧分区释放回锁服务器。然后边会从一个共享文件系统载入，并在没有线程内同步的情况下在多个线程上进行训练。在一个单独的线程中，仅有少量共享参数会与一个共享参数服务器持续同步。模型检查点偶尔会从 trainer 写入到共享文件系统中。这个模型允许使用至多 P/2 台机器时，让一组 P 个 bucket 并行化。

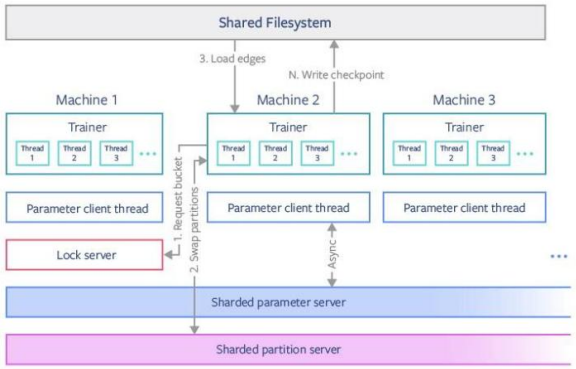

图 4　PBG 分布式架构

2.3　GraphVite

GraphVite是一个通用的图嵌入系统/框架，主打多CPU多GPU快速/超大规模训练。这是一款单机框架，不支持分布式集群，主要是为没有分布式集群的用户打造。最大特点是速度快：与目前最快的框架相比，GraphVite在性能几乎没有任何牺牲的情况下，速度提高了约50倍（图1中以LINE算法作为对比）。其次，GraphVite在单机四卡上，最大可以训练一张6千万点，18亿边的图，训到收敛也只需要不到1天的时间。图嵌入算法通常包括两个阶段：网络增强和嵌入训练。网络增强阶段先对原始网络用随机游走产生增强网络，然后从增强网络中采样正边，这一阶段涉及到大量的随机访问，而CPU在随机访问上GPU性能好出太多，所以网络增强阶段应该放在CPU上进行。嵌入训练主要是通过上一阶段生成的样本训练节点嵌入。这一阶段主要的运算是矩阵计算，所以应该放在GPU上。对于大多数现有算法，网络增强和嵌入训练两阶段是顺序进行的，而每个阶段是用多个CPU线程并行。如下图所示：

Algorithm 1 General framework of node embedding

$E' \leftarrow E$
for $v \in V$ **do** ▷ parallelizable
  for $u \in$ WALK(v) **do**
    $E' \leftarrow E' \cup \{(v, u, \text{WEIGHT}(v, u))\}$
  end for
end for

for each iteration **do** ▷ parallelizable
  $v, u \leftarrow$ EDGESAMPLING(E')
  TRAIN(**vertex**[v], **context**[u], $label = 1$)
  for $u' \in$ NEGATIVESAMPLING(V) **do**
    TRAIN(**vertex**[v], **context**[u'], $label = 0$)
  end for
end for

图 5 传统算法

一个高性能 CPU/GPU 图嵌入框架主要需要解决三个问题：（1）怎么有效地进行网络增强（2）怎么把嵌入训练放到 GPU 加速训练；（3）多 CPU 和多 GPU 间怎么以最小代价同步参数。

针对上述这三个问题，GraphVite 提出三个解决方案：（1）并行在线增强（2）并行负采样（3）协同策略。作者提出的 GraphVite 的框架图如下所示:

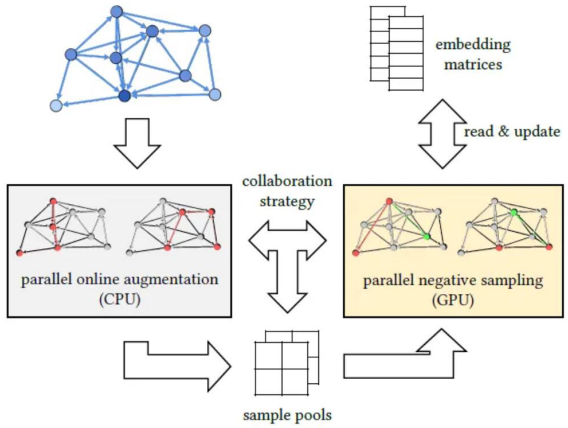

图 6 graphVite 框架图

2.4 Marius

图嵌入模型训练任务是资源密集型任务，具体来说主要是计算密集型和内存密集型任务，举例来说，假设使用一个 400 维的向量表示一个结点，那么每个结点需要 1600 字节去存储（int 类型），那么对于一个具有五百万结点的社交网络来说，需要 80GB 的内存去存储，然后还需要使用各种优化算法如 SGD、Adam 等算法去优化损失函数，若主机系统内存或显存不够，这就可能会导致 IO-bond 问题，即完成计算所需的时间主要由等待 IO 操作完成所花费的时间决定。

因此，若训练任务所需要的内存或显存与主机的内存和显存不匹配，那么就需要大量的数据移动开销，这会大大降低整个系统的资源利用率并且降低训练速度。

比如之前达到 state-of-art 的系统 DGL-KE 和 PBG 在单机训练 时 GPU 的利用率如下所示

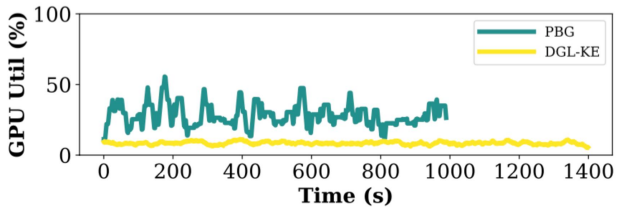

图 7　PGB 与 DGL-KE 的 GPU 利用率

可以发现 PBG 系统的 GPU 平均利用率大概在 30%, 而 DGL-KE 系统的 GPU 平均利用率在 10%左右。

GPU 的低利用率主要是由于系统在处理数据移动的时候产生的时间开销。对于超出显存负载能力的训练任务，DGL-KE 的做法是把参数存放在内存中，然后再用 GPU 使用小批量进行同步计算，因此 CPU 内存大小限制了该系统最终的效率。然而，PBG 则是图分区，这让模型不必完全载入到内存中，但是分区交换会大大限制 GPU 的利用率。

作者引入了一种新的流水线训练架构，可以交错数据访问、传输和计算以实现高 GPU 利用率

先前实现 Scaling Beyond GPU-Memory，普遍有两种方法，即:

1.使用内存去存储参数

2.将模型参数分块，然后分块存储

对于第一种算法 DGL- KE 和 GraphVite，把 node embedding parameters 存入 CPU 内存，把 relation embedding parameters 存入 GPU 显存

算法实现如下：

Algorithm 1: Synchronous Embedding Training

for i *in range*$(num\_batches)$ **do**
1. $\mathbf{B_i} = \text{getBatchEdges}(i)$;
2. $\mathbf{\Theta_n} = \text{getCpuParameters}(\mathbf{B_i})$;
3. $\text{transferBatchToDevice}(B_i, \mathbf{\Theta_n})$;
4. $\mathbf{\Theta_r} = \text{getGpuParameters}(\mathbf{B_i})$;
5. $\mathbf{B}_\theta = \text{formBatch}(\mathbf{B_i}, \mathbf{\Theta_n}, \mathbf{\Theta_r})$;
6. $\mathbf{G_n}, \mathbf{G_r} = \text{computeGradients}(\mathbf{B}_\theta)$;
7. $\text{updateGpuParameters}(\mathbf{B_i}, \mathbf{G_r})$;
8. $\text{transferGradientsToHost}(\mathbf{G_n})$;
9. $\text{updateCpuParameters}(\mathbf{B_i}, \mathbf{G_n})$;

上述方法受限于内存的大小, PBG 方法把 node embeding parameter 分成 p 个不相交的部分，然后把他们存储起来，分区的主要缺点是分区交换成本高昂，并导致 GPU 在交换发生时处于空闲状态。事实上，在交换期间，利用率趋于零，如图 1 所示。我们发现 PBG 的平均 GPU 利用率为 28%。为了最好地利用资源，使用分区来扩展计算机内存大小的系统需要减少交换分区产生的开销。

通过将图结构分区为随机划分的 P 个分区，使得可将两个分区放入内存中，PBG 解决了传统图嵌入方法的一些短板。举个例子，如果一条边的起点在分区 p1，终点在分区 p2，则它会被放入 bucket (p1, p2)。然后，在同一模型中，根据源节点和目标节点将这些图节点划分到 P2 bucket。完成节点和边的分区之后，可以每次在一个 bucket 内

执行训练。bucket (p1, p2) 的训练仅需要将分区 p1 和 p2 的嵌入存储到内存中。PBG 结构能保证 bucket 至少有一个之前已训练的嵌入分区。

针对 DGL-KE 系统存在的 Synchronous Training 的问题，Marius 使用了 pipeline 的方法，将各个计算步骤通过流水线的方式并行。整个 pipeline 包括了五个步骤：

1.加载数据

2.将数据从内存拷贝到显存

3.计算梯度

4.将数据从显存拷贝到内存

5.更新参数

将这五个步骤进行流水线并行，可以有效地提高系统的吞吐。然而由于训练过程从同步变成了异步，会引入 staleness 的问题，也就是上一次 mini-batch 训练的结果还没有更新，下一个 mini-batch 的数据就已经传输到 GPU 中了，导致 mini-batch 的参数可能并不是最新的。Marius 通过减少 pipeline 中 mini-batch 的数量来降低出现 staleness 的概率。

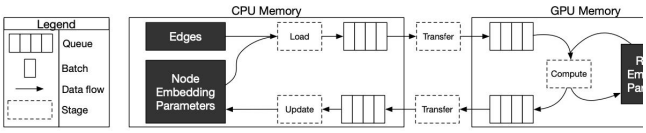

图 6　Marius 流水线示意图

针对 PBG 存在的 swap 开销大的问题，本项工作提出了 Buffer-aware Edge Traversal Algorithm 来遍历图，可以有效地降低 swap 的次数。BETA 算法的核心思路是尽量复用现有 buffer 中的 node

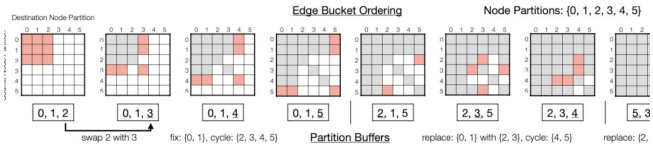

图 8 Partition 流程

2.5　LightNE

为了获得准确、高度可扩展且经济高效的图嵌入训练解决方案，LightNE 横空出世；LightNE 非常轻量，但是具有很强大的性能保险，可以对训练数十亿节点和数千亿条边的 graph 的 embedding。LightNE 的设计目标如下：

（1）可扩展：在 1.5 小时内嵌入具有 1B 边缘的图形。

（2）轻量级：占用硬件成本低于 100 美元（按云租金计算），以处理 1B 至 100B 边。、

（3）准确：在相同的时间预算和类似的资源下，在下游任务中实现最高的准确度。

LightNE 的算法涉及包含两个步骤：

1. 使用 NetSMF 新的 edge downsampling 算法，提高了采样的复杂性。

2. 使用 ProNE 的 spectral propagation 增强了 NetSMF 的 embedding。

3 总结和展望

近几年，图数据处理引起了众多学者的广泛关注，节点分类和图分类是复杂网络研 究中的重要研究分支。目前大部分任务都是处理图结构数据，为了完成图相关任务，首先需要将图中节点和边的信息学习为低维向量，从而完成下游机器学习任务。

虽然图嵌入算法在处理高维稀疏数据、计算效率和嵌入效果上已有大幅提升，但面对不断发展和变化的数据及应用要求，图嵌入算法仍需进一步发展和创新。

Marius 是一个用于在单个机器上计算大规模图嵌入模型的新框架。为了优化数据的移动并最大限度地提高 GPU 利用率，提出了一种利用分区缓存和 BETA 排序的流水线架构。但是如何将 Marius 背后的思想应用于分布式系统仍需要探索。

DGL-KE 改善局部性，减少通信，同时利用并行计算能力。 DGL-KE 规模几乎是线性的机器资源，同时仍然实现了非常高的模型精度。

PyTorch BigGraph 可以扩展到具有数十亿个节点的图。为了节省内存使用量并允许并行化，PBG 将相邻矩阵的块分解为 N 个 buckets，一次训练一个 bucket 的边。对图进行分区可以减少 88%的内存消耗，同时不会降低 embeding 质量，在 8 台机器上进行分布式训练可以将训练速度提高 4 倍。

LightNE 结合了两种先进的 graph embeding 算法 NetSMF 和 ProNE，在九个数据集上实现了当年最先进的性能。通过结合稀疏并行图形处理技术和其他并行算法技术，LightNE 能够在几个小时内学习具有数千亿条边图的 embeddding。

GraphVite 是一种高性能 CPU-GPU 混合系统。它将现有的节点嵌入方法扩展到 GPU，并显著加快了在单机上训练节点嵌入的速度。GraphVite 高效地利用 CPU 线程。还使用了一种降低 CPU 和 GPU

之间的同步成本的策略。

参 考 文 献

[1] Qiu J , Dhulipala L , Tang J , et al. LightNE: A Lightweight Graph Processing System for Network Embedding[C]// SIGMOD/PODS '21: International Conference on Management of Data. 2021.

[2] Lerer A , Wu L , Shen J , et al. PyTorch-BigGraph: A Large-scale Graph Embedding System[J]. 2019.

[3] Zhu Z , Xu S , Qu M , et al. GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding:, 10.1145/3308558.3313508[P]. 2019.

[4] Mohoney J , Waleffe R , Venkataraman S , et al. Marius: Learning Massive Graph Embeddings on a Single Machine:, 2021.

[5] Qiu J , Dhulipala L , Tang J , et al. LightNE: A Lightweight Graph Processing System for Network Embedding[C]// SIGMOD/PODS '21: International Conference on Management of Data. 2021.