手稿，以防万一

- 老师，师兄师姐，大家晚上好。今天我介绍的这篇论文题目是Blink: Fast and Generic Collectives for Distributed ML (分布式机器学习的快速通用集合通信库)。作者是来自加州伯克利分校的RISELab的博士生王冠华，研究领域：分布式机器学习，计算机网络，区块链；二作来自威斯康星大学麦迪逊分校；后面三位来自微软实验室；最后一位是作者的导师斯托卡，同时是riselab的leader。Blink是微软项目的一部分，于2018年申请专利，2020年被MLSys（Machine Learning and Sysytems，机器学习与系统会议）录用。
- 首先来介绍一下背景。DNNs为许多不同应用提供了最先进的结果。在图像分类，语音识别，机器人，游戏博弈等等领域，都取得了不错的效果。但是深度神经网络模型的训练是一个反复迭代且耗时的过程。针对此，分布式深度学习训练应运而生。
- 分布式深度学习最常用的方法是数据并行化，增加训练所用的节点数量，能够有效地减少训练时间。上图显示了在ImageNet数据集上，随着训练所用的节点数量增加，训练时间极大地减少了。（翻页）在数据并行化中，每个节点仅读取和处理唯一的数据子集，并在训练期间更新本地模型。然后，将这些本地模型参数与其他节点同步以计算全局参数。
- 尽管进行了许多性能优化，但是模型同步是云服务器上数据并行训练中的一大开销。该图显示的是：使用tensorflow进行分布式训练，理想速度和实际训练速度的对比。结果显示：通信开销往往占到50%。
- 这张图显示的是：使用Pytorch进行分布式训练，对比不同的网络在对GPU上训练的通信开销。在通信密集型网络如AlexNet,VGG,GNMT，通信开销达到90%。这个问题反映了一个事实：GPU的算力变得越来越快，模型变得越来越大，使得通信开销变成了瓶颈。
- 为了缓解通信瓶颈，最近硬件和软件都有了很大的改进。
- 在硬件方面，NVIDIA DGX-1 和 DGX-2 是最先进的多GPU服务器。GPU之间通过NVLink互连，提高20-25GBps（G比特每秒）的吞吐量。
- 软件：NVIDIA的NCCL，Uber的horovod，facebook的gloo，还有百度的Ring AllReduce。这些都是环形聚合通信协议。
- 这些硬件和软件的改进能否缓解数据并行训练中的通信瓶颈? (翻页)并没有
- 即使使用NVLink 和 NCCL，通信开销依然挺大的。在DGX1-V100中运行时，跨GPU通信以总epoch时间的百分比来衡量。以4个GPU为例，通信开销所占比例

的变化还是挺大的。

- 即使对于相同的网络，最高能到45%，最低能到10%。
- 对于不同数量的GPU和不同类别的神经网络来说，高通信开销是普遍的。如该红线所示。
- 我们需要更快的集合通信协议
- 接下来的介绍分成一下四个部分，首先介绍一下Motivation，然后介绍实现更快的聚合通信所遇到的挑战，然后是总体架构设计，最后是实验评估
- 第一个挑战是：不同服务器的拓扑结构。DGX1-P100是第一代NVLink，拓扑结构如左图所示，DGX1-V100是第2代NVLink，拓扑结构如右图所示。对比二者的拓扑结构，红色虚线是改变的地方，速度从18提升到23GB/s。因此，通信协议需要了解拓扑结构才能有效使用硬件连接。
- 第二个挑战是：链路的异构性。DGX-1 中既有诸如 NVLink的 GPU 点对点互连，也有诸如 PCIe的共享互连。如左图所示：PCIe 通过 PCIe 交换机层次结构将多个 GPU 相互连接到一台计算机内，并连接到 CPU 和 I/O 设备。
- 第三个挑战是：集群中的碎片化。如图所示，在单个8-GPU服务器上分配3、5、6或7个gpu任务是很常见的，尽管多gpu任务更多地要求2的幂次gpu。（翻页）为什么碎片化？（不了解拓扑和作业迁移）（碎片化：一个任务可以被分成多个）许多群集调度程序不了解拓扑；如果没有对有效作业迁移的支持，DNN作业必须拥抱碎片化以避免排队延迟。（翻页）其次，不规则的拓扑将导致非环形结构如果现有解决方案（NCCL）无法形成NVLink环，则会退回到PCIe。（这图上画一下）
- 作者就思考：我们能够做的更好，来应对以上三个挑战（不同的拓扑结构、链路的异构性、多集群中碎片化支持），（翻页）作者提出了Blink。
- 下面将介绍Blink的设计
- 给定拓扑，Blink 的主要方法是动态地生成适当的集合通信原语。通过寻找最优生成树结构来实现高利用率，使用能够最大化传输速率的算法，同时最小化所使用的树的数量。最后，通过在双向链路的每个方向上执行多对一和一对多的操作来实现像 AllReduce 这样的多对多算法。
  Blink的工作流程：
  （1）给定深度学习任务，一旦安排并分配了一组 GPU，Blink 就能够探测机器的拓扑结构，并通过分配的 GPU 推断互连拓扑结构。（翻页）（2）给定拓扑，将集体通信操作建模为有向图上的流，并计算生成树。此步骤表示为图中的 TreeGen，此步骤输出一组生成树和对应于通过它们发送多少数据的权重。（3）CodeGen 解析生成树并生成 CUDA 代码。生成的代码与 NCCL 提供的 API 匹

配，并打包到共享库 libblink.so 中。（4）设置 LD_PRELOAD 标志，以便在调用主程序时动态加载 Blink 实现。这确保了现有程序可以在没有任何修改的情况下运行。

- 下面来介绍如何寻找最优的生成树结构。对于一个6个GPU的拓扑结构，可以用（生成树和环形)两种方式建模。
- 假设从GPU3开始进行广播，可以使用两个NCCL，有一部分链路从始至终没有被使用到，就是这些红色的叉号。
- 也可以使用3个Blink的生成树结构。这用所有的链路都可以被充分利用，从理论上来说，生成树的方法会比ring-based方法好。
- TreeGen的作用是寻找最优的生成树结构。（翻页）优化目标是：给定根节点，在指定的拓扑结构下，最大化所有链路的带宽使用总和。（这要某一条链路的带宽使用量没有超过限制）
  约束条件如下：如果在该生成树，该条链路被保留，则k=1，否则k=0。对所有生成树的k×w求和得到的值不能超过该条边的容量。在包装生成树时，所有带宽的使用量不能超过该链路的容量（一条边不好在太多的生成树里使用）(一条边一条边考虑)（翻页）对于8-GPU的DGX-1，能得到181个生成树，因此需要减少生成树数量。因为生成树数量太多，分配到每个生成树的数据太小，无法完全利用链接带宽。
- 下面这页ppt价绍如何减少生成树的数量。优化目标没有变，给定根节点，在指定的拓扑结构下，最大化所有链路的带宽使用总和。给定根节点，在指定的拓扑结构下，最大化所有链路的带宽使用总和。（翻页）这里作者将问题简化，使用乘法权重更新算法。使用一个容量和一个权重来初始化每一个边，这个权重用来标记已经使用了多少容量。构造了一个整数线性规划问题（integer linear program，ILP），每个权重被限制为 0 或 1：w等于0或1是因为：生成树要么使用链路的所有带宽，要么不使用它。约束条件：在包装生成树时，所有带宽的使用量不能超过该链路的容量。（翻页）最后，181个生成树近似得到6个生成树（其他）运行一个迭代方法，每次迭代都会找到给定当前分配的最小权值生成树。然后，将所选树上的权重增加一个 ε 因子，并相应地更新图上的权重。给定 T1, T2, T3, … Tk，为了最小化生成树的数目，运行一个迭代方法，每次迭代都会找到给定当前分配的最小权值生成树。然后，将所选树上的权重增加一个 ε 因子，并相应地更新图上的权重。给定 T1, T2, T3, … Tk，为了最小化生成树的数目，
- 给定拓扑结构，生成最优的生成树结构
  前面介绍的能够支持一对多，多对一的通信原语，如Reduce和Broadcast
  也能支持多对多的通信原语。选定根节点，先从根节点reduce，然后根节点再朝相反方向broadcast

- 下面介绍Code generate部分
- 将TreeGen输出（生成树）转换为真实的数据传输命令
  Code generate 优化策略：
  将数据块流水线化以减少延迟
  数据块大小多少合适？太小不能充分利用带宽；太大导致高延迟
  作者提出的策略是自动选择数据块大小：MIAD（乘性增长，加性减少）以1，2，4，8的方式增长数据块大小，当throughput(吞吐量)不再增加时，就停止。
  论文中作者说，如果增长数据块大小throughput减少后，就以加性的方法减少数据块大小
- 下面介绍实验评估部分：
- 这是DGX-1V100的拓扑结构，实验结果显示Blink在AllReduce方面，速度明显优于NCCL2
- 这是NCCL的拓扑结构，形成两个环型结构。
- 这是Blink的结构，得到三个生成树。Blink能够充分利用服务器上的拓扑结构。
- Blink和NCCL2在DGX-1v100上的对比效果。Broadcast操作最高能够提高8倍速度，平均提高2倍速度。AllReduce操作最高能够提高6倍速度，平均提高2倍速度。
- 在PyTorch上应用Blink，测试四种经典网络（ResNet18,ResNet50,AlexNet,VGG16)，通信时间最高减少87%，平均31%
- 同时能够最高减少40%的训练迭代时间
- 以上是这篇论文的基本内容。首先因为拓扑异构性导致链路利用不足，本文提出的Blink协议能够通过寻找最优生成树的结构优化链路利用。自动生成one-to-all, all-to-one, all-to-all的通信原语。实验显示，Blink明显优于NCCL。