MLSYS分布式系统

解决单节点的算例和内存不足

1. 并行方法：
    1. 数据并行：一个批次内N个数据，使用M个并行设备训练，每个设备需要拷贝一份网络参数，各自计算出梯度后，由集合通信的AllReduce进行梯度聚合
    2. 模型并行：分为算子内并行（一个算子分配到不同的设备上），算子间并行（不同算子分配到不同的设备上）
    3. 混合并行：数据并行+模型并行
2. 流水线并行：
    1. 模型并行的劣势：一个设备在计算时另一个设备会空闲
    2. 流水线改进：将一个小批次进一步拆分为多个**微批次**，如下图，提高并行训练效率。核心：微批次大小
        1. 过大：每个微批次样本过少，无法充分利用硬件加速
        2. 过小：更长时间的流水线气泡

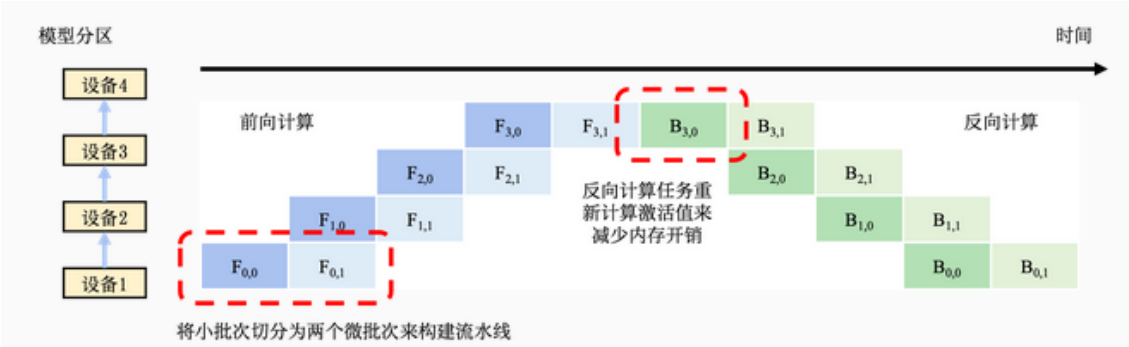


图11.3.1 流水线并行训练系统（Pipeline Parallel Training System）

3. 机器学习的集群架构：
    1. 计算集群：机器学习模型分布训练的整套设备。多台服务器放置在一个机柜内，由架顶交换机管理，多台架顶交换机间可增加骨干交换机。即通常由树状的拓扑结构构建，叶子节点是服务器，上层是交换机。

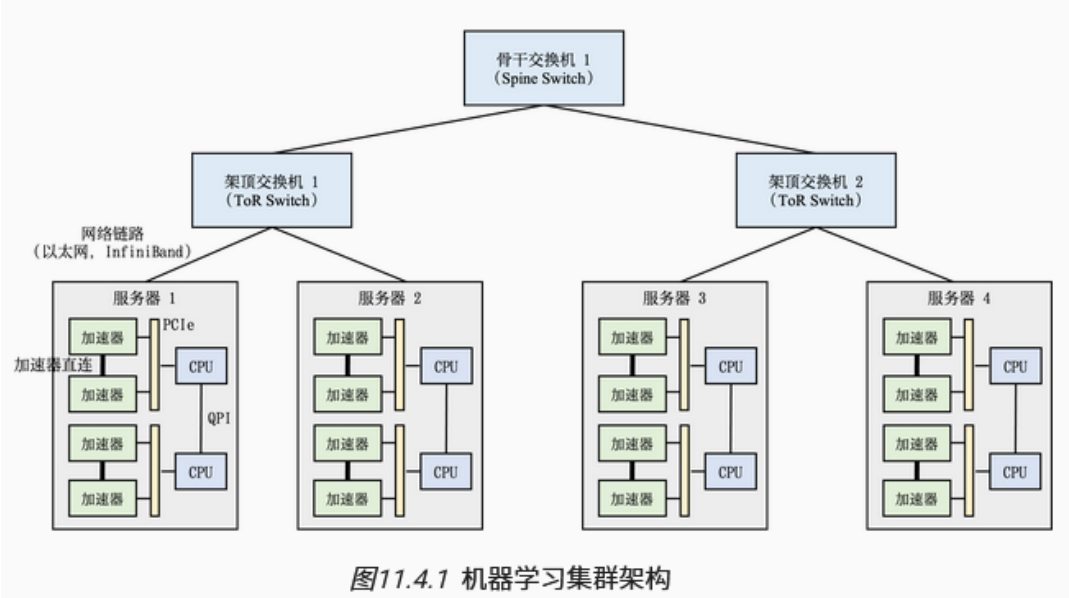


图11.4.1 机器学习集群架构

    2. 核心设计需求：跨机柜通信会产生网络带宽超额认购，应将网络通信限制在机柜内

3. 训练流程（共有N个模型副本分布在N个设备）：
    1. 各自计算梯度后，N-1个模型副本的梯度同步到主模型中
    2. 主模型计算平均梯度
    3. 将平均梯度广播到其余N-1个副本，避免偏离主模型参数
4. 服务器间的网络通信：以前使用以太网，现在使用InfiniBand链路
5. 服务器内部的网络通信：异构网络，如上图，两个CPU间通过QuickPathInterconnect（QPI）进行通信，硬件加速器和CPU通过PCIe总线通信，但是PCIe带宽较小，往往硬件加速器间实现高速互连，以绕过PCIe

4. **集合通信**：
    1. 通信模型：参考计网

    > 假定在一个分布式机器学习集群中，存在p个计算设备，并由一个网络来连接所有的设备。每个设备有自己的独立内存，并且所有设备间的通信都通过该网络传输。同时，每个设备都有一个编号i，其中i的范围从1到p。设备之间的点对点（Point-to-Point, P2P）通信由全双工传输（Full-Duplex Transmission)实现。该通信模型的基本行为可以定义如下：
    >
    > - 每次通信有且仅有一个发送者（Sender）和一个接收者（Receiver）。在某个特定时刻，每个设备仅能至多发送或接收一个消息（Message）。每个设备可以同时发送一个消息和接收一个消息。一个网络中可以同时传输多个来自于不同设备的消息。
    > - 传输一个长度为l个字节（Byte）的消息会花费$a + b \times l$的时间，其中a代表延迟（Latency），即一个字节通过网络从一个设备出发到达另一个设备所需的时间；b代表传输延迟（Transmission Delay），即传输一个具有l个字节的消息所需的全部时间。前者取决于两个设备间的物理距离（如跨设备、跨机器、跨集群等），后者取决于通信网络的带宽。需要注意的是，这里简化了传输延迟的定义，其并不考虑在真实网络传输中会出现的丢失的消息（Dropped Message）和损坏的消息（Corrupted Message）的情况。

    2. 各类通信算子：
        1. Broadcast：从编号为i的设备发送长度为l字节的消息给剩余的p-1个设备。

            实现：采用分治的思想，**1传2，2传4...**，使用多设备并行广播，时间复杂度为$(a + b * l) * \log p$。

        2. Reduce：将不同设备上的计算结果聚合，通常由全部设备共同发起，最终聚合结果存在编号为i的设备上，聚合函数包括加和，乘积，最大值和最小值。

            实现：同样可采用分治的思想，即把1到$p/2 - 1$的聚合结果存到设备1，$p/2$到p的聚合结果存到$p/2$，最后把$p/2$的结果发送到1，再聚合，得到最终结果。时间复杂度为$(a + b * l) * \log p$。

            分析：这种算法的复杂度并非最低，但是若所有数据同步全部涌入i号设备，会导致严重的带宽不足而产生网络拥塞

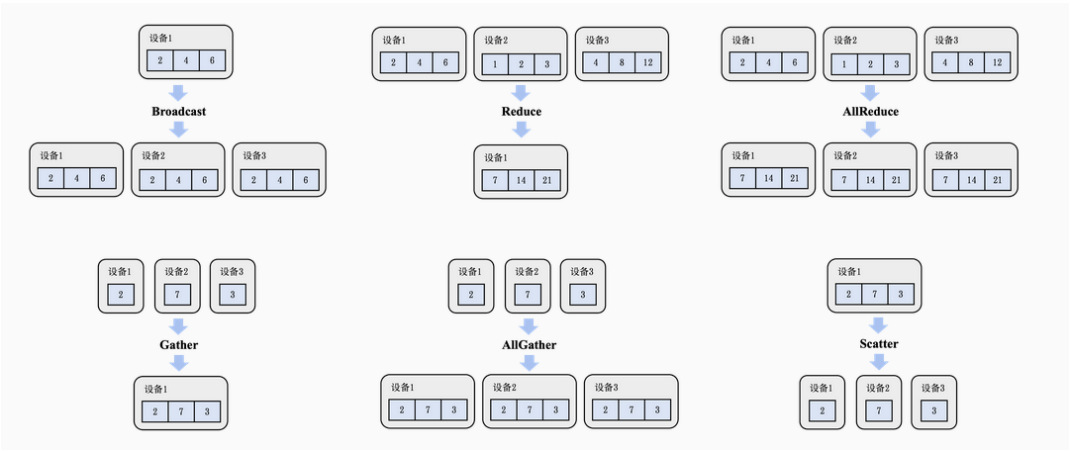
        3. **AllReduce**：Reduce + Broadcast
            1. 基于AllReduce的梯度平均算法：将网络和算力开销均摊给全部节点。
            2. 具体方案：M个设备，模型N个参数，将N个参数分M个区，每个区N/M个参数，即每个设备作为一个区的主设备，将该区的参数聚合并广播。
            3. **网络优化：** 全部设备都同时在接收和发送数据，利用起了每个设备的入口（Ingress）和出口（Egress）带宽。

                算力优化： 全部设备的处理器都参与了梯度相加的计算。因此在AllReduce算法的过程中，可利用的处理器是$M * P$，其中M是节点数量，P是单个设备的处理器数量，从而让系统实现计算上的可扩展性。

                负载均衡： 由于数据分区是平均划分的，因此每次设备分摊到的通信和计算开销是相等的。

4. Gather：将全部设备的数据全部收集到编号为i的设备上。

   实现：将其看作一种特殊的Reduce函数，即拼接。采用分治的思想并行收集，但由于数据长度随收集阶段而指数级上升，因此复杂度为$a * \log p + (p-1) * b * l$



5. AllGather：Gather+Boardcast，可使用超立方体算法优化
6. Scatter：Gather的逆运算，把一个存在于编号为i的设备上，长度为$p * l$的链式数据分散到每个设备上。

   实现：分治，数据平均切分为两个子链分散到不同设备中，子问题递归，时间复杂度同Gather。

3. 集合通信优化技术（涉及模型并行）：
   1. 几个概念：节点（服务器），模型（包含多个模型副本），模型副本（包含多个切片）
   2. 单一节点（服务器）的参数存储：已知节点内部的通信比节点之间的通信更高效，模型本身前向和反向计算时需要在不同切片之间进行的通信远小于不同模型副本梯度平均的通信量。

      结论（书上好像有问题）：将单一模型的全部切片存储到同一节点内部。
   3. **基于AllGather算子的前向计算**：对所有包含模型参数的加速器进行一次AllGather计算。（大致算法）对于某一层，接收上一层的输出，抛弃其余层传过来的数据，用于计算当前层的输出，并传给其余所有层。
   4. **基于ReduceScatter算子的梯度平均**：在**反向计算**时我们只需要后一层的参数来计算本层的激活值和梯度，只需要再次使用AllGather来完成每个加速器上的梯度计算。在**梯度更新**时，由于每个加速器只需要对应的一部分参数的梯度，因此无需使用AllReduce算子，而可以使用ReduceScatter算子直接把相应的梯度存到编号为i的加速器上。
4. 集合通信技术（数据并行）：MLSYS提供两个级别的抽象
   1. 与硬件耦合，调用集合通信算子

```
# 基于pygloo底层接口实现AllReduce算法

@ray.remote(num_cpus=1)
def gloo_allreduce(rank, world_size):
    context = pygloo.rendezvous.Context(rank, world_size)
    ...

    Sendbuf = np.array([[1,2,3],[1,2,3]], dtype=np.float32)
    recvbuf = np.zeros_like(Sendbuf, dtype=np.float32)
    Sendptr = Sendbuf.ctypes.data
    recvptr = recvbuf.ctypes.data

    # 标明发送者和接收者并直接调用AllReduce算法
    pygloo.allreduce(context, Sendptr, recvptr,
```

```
                    Sendbuf.size, pygloo.glooDataType_t.glooFloat32,
                    pygloo.ReduceOp.SUM,
pygloo.allreduceAlgorithm.RING)
```

@ray.remote(num_cpus=1)：表示这个函数将作为一个 Ray 任务在分布式环境中运行，并且需要分配 1 个 CPU 资源

2. 偏向神经网络实现，打包到更上层（如pytorch用的Data Parallel）

```
ddp_model = torch.nn.parallel.DistributedDataParallel(model,
device_ids=[rank])
```

5. 集合通信技术（混合并行）：介绍亲爹华为的MindSpore，用户可自行定义各个算子的切分方法，同时框架会为算子间自动插入适当的集合通信算子，如下图所示：算子1数据并行后Y_i分布在不同的设备上，而下一个算子是模型并行，需要Y的全量数据，此时需要插入AllGather算子将Y_i收集。

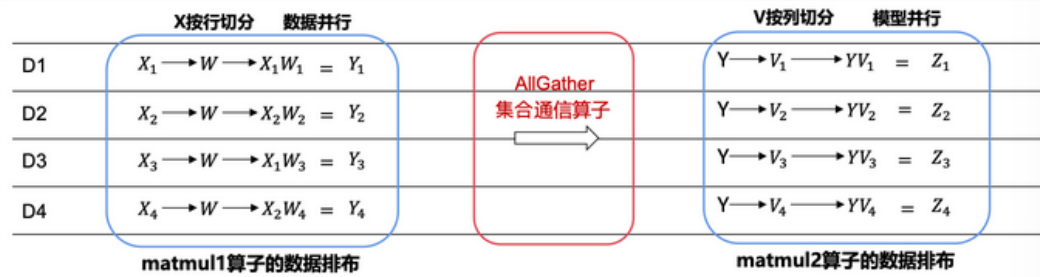
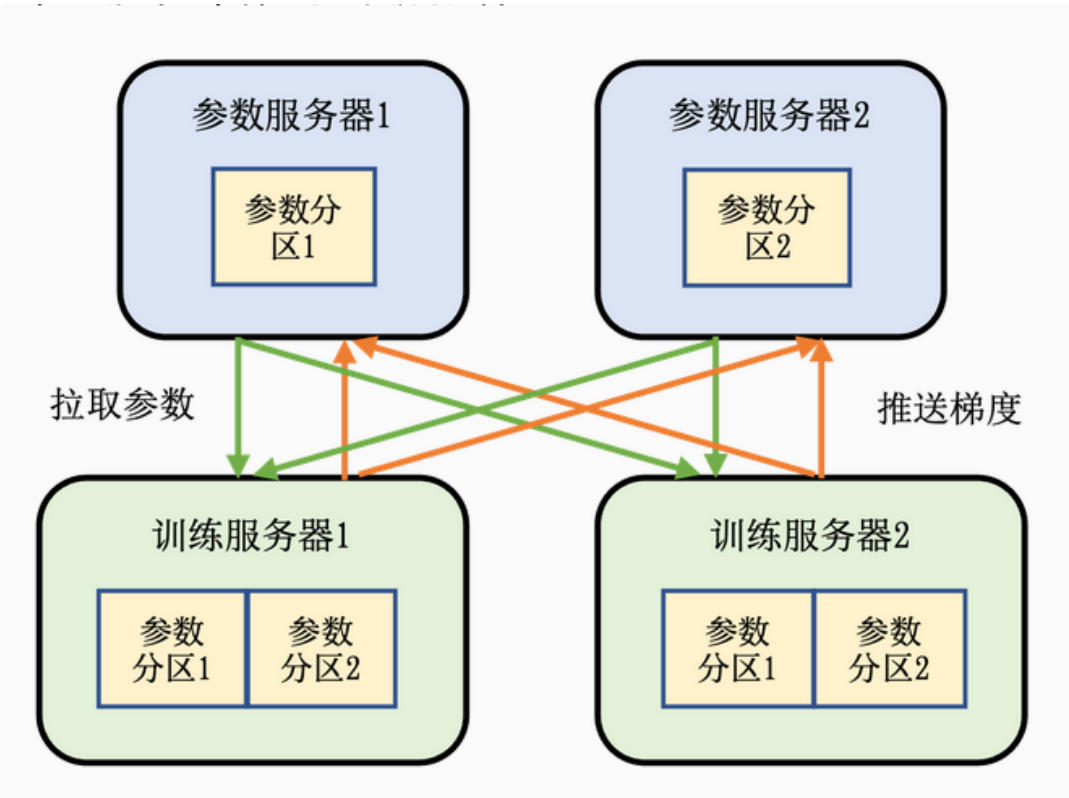


图11.5.3 相邻算子之间插入集合通信算子举例

5. 参数服务器系统：与集合通信并列的分布式系统的实现
    1. 系统架构：服务器有两种角色，参数服务器与训练服务器
        1. 参数服务器：提供内存资源与通信资源
        2. 训练服务器：提供计算资源
        3. 流程：模型参数分配到不同的参数服务器上负责同步，训练服务器拥有完整模型，用本地数据集训练出梯度后推送到各自的参数服务器，参数服务器等待训练服务器都完成梯度推送后，更新参数，后通知训练服务器来拉取最新的参数，开始下一轮迭代。
    2. 异步训练：解决某些节点的“落后者现象”，即正常的AllReduce很可能被个别性能差的节点显著影响整体的训练速度。使用参数服务器，有两种解决办法
        1. 参数服务器等到一定的训练服务器的梯度推送后，便更新参数，不用等落后者
        2. 异步训练：每有训练服务器推送梯度后，立刻用于更新参数，并通知训练服务器立刻拉拉去最新参数。这会导致不同的训练服务器中最新参数不同步，牺牲精度但是显著提高效率。
    3. 数据副本：解决参数服务器故障问题，一份参数在多个机器上拥有副本，区分主副本和从副本，主副本更新后向所有从副本同步。


参数服务器1
参数分区1
参数服务器2
参数分区2
拉取参数
推送梯度
训练服务器1
参数分区1
参数分区2
训练服务器2
参数分区1
参数分区2