西北工業大學

本科毕业设计（论文）

题 目 多专家系统模型分布式训练优化

专业名称 计算机科学与技术专业

学生姓名 杨益滔

指导教师 陈亚兴，徐宏

完成时间 2023 年 6 月

摘　要

本文工作致力于提出一种面向 MoE（混合专家）模型的高效分布式训练框架，旨在加速和优化大规模模型的训练过程。MoE 模型是一种强大的深度学习模型，但其训练过程常常受限于计算和通信的瓶颈。为了解决这些问题，在 Deepspeed 分布式深度学习框架的基础上，提出了两个创新点：动态路由的数据分派方式和基于网络拓扑的负载均衡策略。

首先，引入了动态路由的数据分派方式，旨在优化数据在 MoE 模型中的分配过程。传统的 Top-1 和 Top-2 Gating 策略在数据分派中存在一定的局限性，无法充分利用专家模型的容量和多样性。我们提出了一种动态路由策略，根据数据的特征和模型的权重动态地选择适合的专家进行计算，从而实现更好的数据分派和利用。

其次，设计了基于网络拓扑的负载均衡策略，以优化训练阶段全局通信与后续计算并行性较差的问题。在传统的 MoE 并行训练中，需要通过全局的 All-to-All 通信不同 GPU 上的数据，这一过程过程常常成为性能瓶颈，尤其是在大规模模型和分布式环境中。我们通过分析 GPU 集群的拓扑结构，提出了一种智能的负载均衡策略，根据网络拓扑信息调整专家的分配和数据的路由，以减少通信开销并提高整体性能。

进行了一系列实验来评估我们的设计。我们在中小规模 GPU 集群中开展真实实验。在前向和反向传播阶段，我们设计的训练系统显至多可以提升 4 倍训练速度，并且可以有效利用了数据的多样性。然而，在梯度同步阶段，由于更复杂的通信模式，性能有所下降。整体端到端的实验结果表明我们设计的策略能够更均衡地分配计算负载，减少通信瓶颈，提高整体训练性能。

综上, 本文主要贡献有

1. 分析现有 MoE 训练框架的不足
2. 提出了基于动态路由的数据分派方式
3. 设计了基于网络拓扑的负载均衡策略
4. 开展了真实系统的实验，验证提出方案的性能

关键词:　混合专家模型，深度学习分布式训练系统, 动态负载均衡算法

Abstract

Our work aims to propose a distributed training system for MoE (Mixture of Experts) models, with the goal of accelerating and optimizing the training process of large-scale models. The MoE model is a powerful deep learning model, but its training process is often constrained by computational and communication bottlenecks. To address these issues, we introduce two innovative points based on the Deepspeed distributed deep learning framework: dynamic routing for data allocation and network topology-based load balancing strategy.

Firstly, we introduce dynamic routing for data allocation to optimize the allocation process within the MoE model. Traditional Top-1 and Top-2 gating strategies have limitations in data allocation, as they cannot fully leverage the expert capacity and diversity of the model. We propose a dynamic routing strategy that dynamically selects suitable experts for computation based on the data features and model weights, achieving better data allocation and utilization.

Secondly, we design a load balancing strategy for experts based on network topology to address the performance degradation during the All-to-All communication phase in each layer. In traditional parallel MoE training, computation must wait for All-to-All communication finish, which becomes a performance bottleneck, especially in large-scale models and distributed environments. By analyzing the GPU cluster's topology, we propose an intelligent load balancing strategy that adjusts the expert allocation and data routing based on the network topology information to reduce communication overhead and improve overall performance.

We conducted a series of experiments to evaluate our designs. Real experiments were conducted on medium-sized GPU clusters. In the forward and backward propagation stages, our designed training system achieved up to a 4x speedup and effectively utilized the diversity of data. However, in the gradient synchronization phase, there was a slight performance drop due to the more complex communication patterns. Overall, the end-to-end experimental results demonstrate that our designed strategies can balance the computational load, reduce communication bottlenecks, and improve overall training performance.

To sum up, the contribution of our works can be summarized as follow:

1. Analyzing the limitations of existing MoE training frameworks
2. Proposing a data allocation approach based on dynamic routing
3. Designing a load balancing strategy based on network topology
4. Conducting real-system experiments to validate the performance of the proposed

KEY WORDS: Mixture of Experts model, distributed deep learning training system, dynamic load balancing algorithm

目 录

第一章　绪论

1.1 研究背景及意义

(1) 选题背景

预训练模型是已经用广泛的样本训练过的模型。它已经在一个大型数据集上针对特定任务进行了训练。这个数据集可以是多种形式的，例如图像、文本或音频等。预训练模型的训练过程会产生一种通用的特征表示, 这些特征可以被用来执行类似任务的新数据。从头开始训练一个深度学习模型可能需要花费数周或数月的时间, 特别是在缺乏大量数据集的情况下（如 BERT [1], GPT-3 [2]）。在像 ImageNet 这样的大型数据集上训练神经网络，该数据集包含 1000 个类别的 1400 多万张图片，在这样的数据集上重新训练这样的模型是一种很大的开销。使用预训练模型可以作为模型的起点, 这意味着，当为一个新任务微调预训练的模型时，我们不必从随机权重开始。而是可以使用预训练的权重作为初始化，然后只训练后面的特定于新任务的层。这需要更少的数据和训练时间。这可以节省大量的时间和精力。此外预训练模型有其他方面的优势。它们可以将知识从一般领域转移到特定领域。神经网络的浅层倾向于学习一般的特征，如边缘和形状，而后深则学习更具体的特征。因此，在一般图像上训练的预训练模型可以提供一般的低层次特征，然后你只需要为你的具体任务重新训练后面的层。这就是所谓的迁移学习。近年来, 学术界和工业界都对开发精度更高, 参数量更大的预训练模型感兴趣, 因为采用较大的模型会带来更高的准确性。如图1-1所示, 近年机器学习模型参数量近几年呈现爆炸式增长。远远超出了常用 GPU 的显存限制（通常只有 40-80GB，如 Nvidia A100[3]），这也对我们如何快速，高效的训练模型提出了全新的挑战。

研究人员表明，较大的模型会带来更高的准确性。从过去几年深度神经网络 (DNN) 驱动的机器学习技术的快速发展来看，研究者们发现加入更多 DNN 模型参数是最直接但不太复杂的方法之一提高 ML 算法的性能 [4]。然而，DNN 模型容量通常受到计算资源和能源成本的限制 [5]。根本原因是 DNN 的密集架构，其中计算成本通常与参数数量成线性比例。为了解决这种问题，混合专家 (MoE) [6] 在 DNN 中被广泛使用，它通过使用多个并行子模型 (混合专家) 引入了稀疏架构，其中每个输入经过门网络，动态选择并转发给少数专

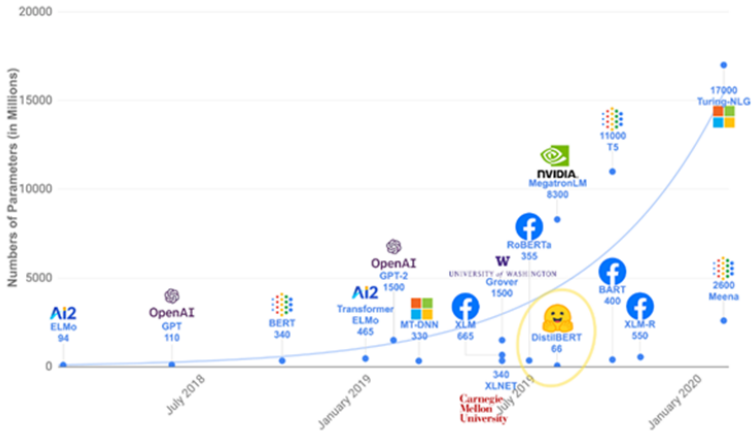

图 1-1　深度学习模型参数显著增长

家处理。专家混合似乎有望将模型扩展到极端尺寸。如图1-2 所示，与直接将小模型缩放为大密集模型不同，MoE[6] 模型由许多小模型组成，即专家。训练样本被送入不同的专家，由轻量级可训练门网络动态选择。在 MoE 中，由于稀疏激活专家，节省了大量额外的计算量，与传统的密集型 DNN 相比，可以显着增加同一时间段内训练的样本数，提高模型精度。MoE 技术是如今将 DNN 扩展到万亿参数的流行方法之一。

MoE 混合专家系统是一种稀疏模型，因此其训练过程不同于传统的密集型 DNN 模型。主要有以下三点挑战：

- **动态激活特性：** MoE 模型的稀疏激活特性使得它在 GPU 集群分布式训练时与现有的静态并行策略不匹配。因为 MoE 模型的每个样本只会被激活一个专家（也就是一个子模型），而其他的专家则不会被激活。这导致静态并行策略无法充分利用计算资源，因为只有部分计算节点被激活，而其他节点则处于空闲状态。

- **额外通信开销：** MoE 模型引入了 GPU 集群节点间额外的 All-to-All 通信，这种通信方式需要在所有计算节点之间进行数据传输和同步，由于 All-to-All 通信是一种同步通信方式，因此它会严重影响训练速度和效率。但是这种通信方式在 MoE 模型中是必需的，因为它需要将每个计算节点计算得到的结果进行汇总和组合，以得到最终的预测结果。

- **节点负载不均衡：** 由于 MoE 模型中的 Gating 在训练过程中不断变化，因此数据可能被分配到不同的专家上，导致负载分配不平衡的问题。如

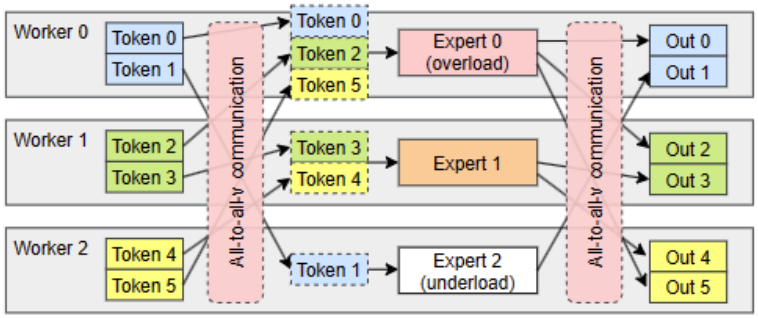

图 1-2　分布式 MoE 模型训练过程（图源 [9]）

果某些专家的负载过高，而其他专家的负载过低，就会导致训练时间的延长和模型性能的下降。因此，需要根据每个专家的激活情况和计算负载进行动态的数据分配和负载均衡，以确保每个专家的计算负载均衡，并最大限度地利用计算资源。

现有的分布式训练框架 [7, 8] 对于其稀疏性的结构没有很好的支持，因此本次毕业设置拟设计一种更加高效的 MoE 训练框架，加速其分布式训练的过程, 从而降低训练大规模的 MoE 模型架构的成本。

(2) 选题依据

随着人工智能技术的不断发展，MoE 模型在各个领域都具有广泛的应用前景。它可以将多个不同的模型集成起来，以提高模型的性能和泛化能力。目前 MoE 模型在语音识别、自然语言处理、计算机视觉等领域都取得了很好的效果。因此，研究 MoE 模型的分布式训练和优化策略对于进一步提高模型的训练效率和性能具有重要意义。通过并行化和分布式计算等技术手段，可以加快 MoE 模型的训练过程，同时提高模型在大规模数据集上的效果。这将有助于应对深度学习任务和数据集日益复杂和庞大的挑战。此外，研究 MoE 模型的训练和优化策略不仅对于深度学习领域的研究具有重要意义，还能为各个领域的应用提供更高效和精确的解决方案。通过将多个专家的知识和能力结合起来，MoE 模型能够更好地应对现实世界中的复杂问题，并提供更加准确和可靠的预测结果。因此，对 MoE 模型的分布式训练和优化策略进行深入研究，还能够在各个应用领域中取得更大的突破，为人工智能技术的进步做出重要贡献。

1.2 深度学习训练系统及优化方法概述

在本节中，我们从算法的角度演示了如何减少分布式 DNN 训练中的通信开销。算法优化包括减少通信轮次和通信量，以及增加计算-通信重叠率。这些优化与底层网络兼容，其中大多数可以在各种网络基础设施和协议之上运行。

1.2.1 分布式深度学习训练系统概述

分布式深度学习训练是一种将一个任务划分为较小的子任务并在多个处理器或设备上同时运行的技术 [10]。这种方法可以加快任务的执行速度，特别是当子任务可以在不同的处理器或设备上并行执行时。在分布式深度学习训练中，我们通过将训练数据和模型参数分布在多个处理器或设备上，然后在多个 GPU 上同时执行子任务来提高训练速度。这使我们能够突破单个 GPU 的内存限制，训练出比单个处理器或设备上所能训练的更大的深度学习模型。通过将训练过程在多个 GPU 上并行化，我们有效地增加了可用于训练模型的计算能力。每个 GPU 在训练数据的一个子集和模型参数的一部分上工作。然后通过汇总各个 GPU 的结果来建立整体模型。随着深度学习模型的规模和复杂性不断增加，分布式训练的好处变得更加明显。更大的神经网络需要更多的数据和计算来优化大量的权重和参数。通过利用多个 GPU，我们可以扩大可用资源的规模，以满足这些大规模模型的需求。

目前，分布式深度学习训练中使用的并行性主要有三种类型：

- **数据并行性 (Data parallel)[10]：**在数据并行要求整个模型能够装入每个处理器或设备的内存中，这种方案将大规模的数据集分成多个小批次，分别在不同的处理器或设备上并行计算，以提高深度学习模型的训练速度和效率。在每个训练迭代或历时结束时，模型参数在所有处理器或设备上同步。更具体地说，每个处理器或设备在其训练数据部分的一批训练样本上工作。它在这个本地批次上执行前向和后向传播，以计算梯度并更新其模型副本中的权重和偏差。计算完成之后需要执行同步步骤，来自每个处理器或设备的模型参数被平均到一起（太频繁的同步会因为通信开销而降低训练速度，而太不频繁的同步则会导致独立的模型副本之间出现分歧）。通过数据并行和模型参数的定期同步，训练数据和计算可以有效地分布在多个处理器或设备上，以加快深度学习模型的训练。总的来说，数据并行难度相对较低，只需要并行化已有代码，但是

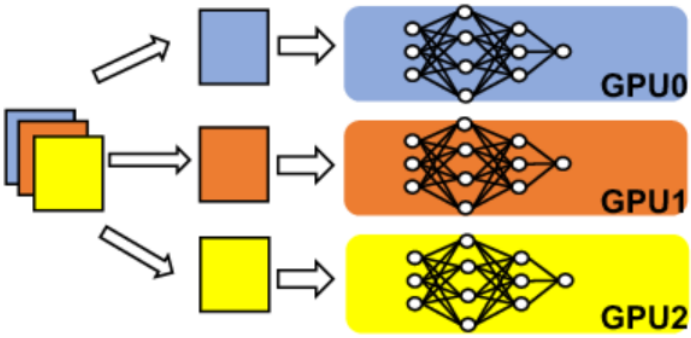

图 1-3　分布式训练-数据并行

其额外的通信开销随着 GPU 数量增加而增加，大规模训练性能较差。

Horovod[11] 和 Pytorchddp[12] 是两个通常采用的数据并行训练系统，可使用 All-Reduce 同步梯度。Byteps[13] 统一了全减少和参数服务器，并利用数据中心簇中的异质资源。

- **模型并行性 (Model parallel)[14]：**模型并行可用于训练太大而无法放入单个处理器或设备的内存中的模型，但是需要处理器或设备之间进行更多通信模型的参数分割到多个 GPU 卡或服务器上训练。这种方法将复杂的深度学习模型分割成独立的子模型，并分配到多张 GPU 卡或服务器上，从而减少单个设备的计算负担。在每张 GPU 计算完成对应部分之后，需要将中间的激活值通过网络传输（NCCL[15]）的方式发送给其他 GPU 参与后续阶段参与计算。使用模型并行需要对模型进行重构以分割参数，因此相应的并行度比较高。这种方式可以利用更多的计算资源来加速非常大模型的训练, 但是参数分割和同步亦需要投入额外工作并存在较高的通信开销。总的来说, 模型并行适合那些模块化清晰且参数易于分割的深度学习模型，并且可以更好地随处理器数量线性缩放。

- **流水线并行 (Pipeline parallel)[16]：**流水线并行是一种通过将深度学习模型划分为多个独立阶段, 并行执行不同阶段来加速训练模型的技术。具体操作包括: 首先将模型分割成多个连续的阶段, 如 Embedding 层、卷积层、池化层等并将一个 Batch 的数据划分为多个 Macro-Batch 分配给不同的阶段进行计算。当一个 Macro-Batch 通过一个阶段后, 将结果传递给下个阶段, 直至所有的 Macro-batch 通过流水线。不同阶段的计算可以同时执行, 形成 computaional pipeline。整个 Batch 全部通过后, 损失函数和梯度计算在最后一个阶段完成。流水线并行适用于很大的模型, 超

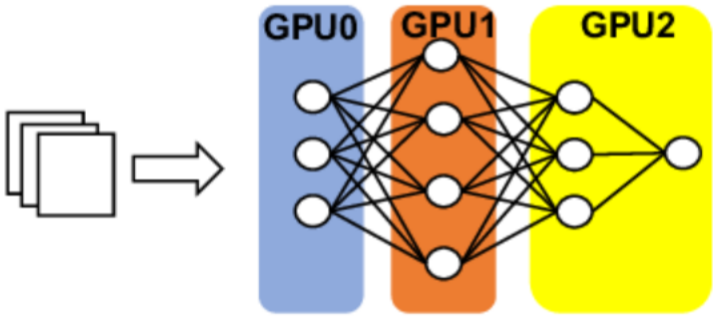

图 1-4　分布式训练-模型并行

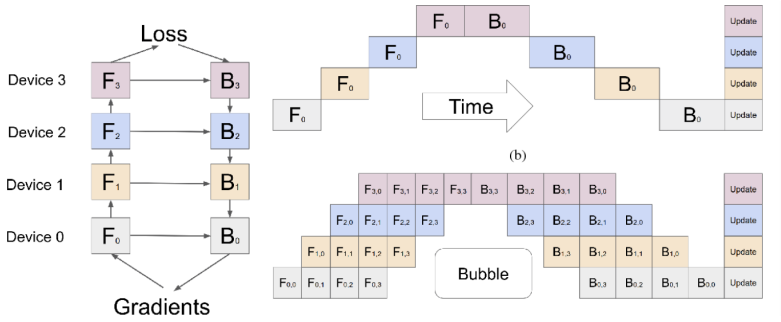

图 1-5　分布式训练-流水线并行（图源 [16]）

出单个 GPU 容量。它可以利用多个 GPU 来加速训练。流水线并行的优点是可以扩展到多个 GPU 来利用更多计算资源, 利用计算资源更有效率。但是它主要挑战是需要更严格的同步不同阶段的计算, 部分阶段可能存在空转, 需要通过调整 Macro-Batch 大小来缓解。通常模型并行与流水线并行可能一起使用，以更好的平衡流水线并行中各个阶段的计算负载与数据通信开销。

- **自动并行（auto parallelism search）**[17] 最近，研究者们提出了自动化寻找并行的策略的方式，即在深度学习分布式训练中，自动化并行通常指的是在给定 GPU 集群拓扑的情况下，通过自动化方法来选择和配置数据/模型/流水线并行方案。首先，自动化寻找并行策略的方式需要考虑许多因素，例如 GPU 数量、网络拓扑结构、数据集大小、模型结构和超参数等。这些因素之间的相互作用非常复杂，因此需要使用高级算法来解决优化问题。然而，这种算法的计算复杂度非常高，需要大量的计算资源和时间。其次，自动化并行还需要考虑不同的并行方案之间的竞争和冲突。例如，多个 GPU 同时访问同一数据集可能会导致争用和延

迟，从而影响训练的速度和质量。因此，需要使用先进的调度算法来解决这些问题。最后，自动化并行的方式还需要考虑可扩展性和通用性。不同的深度学习任务可能需要不同的并行方案，因此需要使用通用的方法来解决这些问题。此外，随着 GPU 集群规模的增加，自动化并行的方式需要支持更大规模的计算，因此需要具有高度的可扩展性。

并行训练模型时，我们需要通过合理的任务划分和调度来优化深度学习训练的效率和可扩展性，并根据深度学习模型本身的特点和可用的硬件资源选择最佳的并行方案。从而在保证模型精度的同时，实现训练速度提升，支持更大规模的训练，以及提高可靠性和容错性。这种方式可以更充分地利用计算资源，提高深度学习模型的训练效率和可扩展性，从而适应越来越复杂和庞大的深度学习任务和数据集的需求。

1.2.2 算法层面训练优化

当使用 SGD 训练深度神经网络时，整个训练过程通常由多个时期和迭代组成。关于并行化，节点通常在每次迭代结束时交换数据。减少通信开销的一种直观方法是减少数据交换或通信轮次的数量。节点的轮次与批量大小和通信周期有关。更大的批量和更长的通信周期都可以减少数据交换轮次。

调整批量大小：批量大小是一个重要的超参数，用于控制节点在每次迭代中读取的数据量。较大的批量通常比较小的批量更接近输入数据的分布，并且在梯度估计中引入较少的方差。然而，使用大批量处理数据会导致更长的处理时间和较少的模型参数更新。这一发现主要是由于批量大小、迭代次数和训练数据大小之间的关系。

大批量的使用会导致迭代次数的减少，从而减少了参数更新的频率。在具有数据并行方案的分布式训练环境中，批大小是每个节点的本地批大小的总和。在传统的分布式深度学习中，节点在每次迭代结束时交换梯度和模型参数。因为参数仅取决于深度神经网络（DNN）本身，单次迭代传递的消息大小保持不变，即使改变了批量大小，消息的形状和大小也不会改变。因此，增加批量大小可以减少迭代次数和通信轮次 [18]。

然而，在实践中，通过直接使用巨大的批量进行并行随机梯度下降（SGD）训练时，与训练小批量相比，可能会出现泛化能力下降的问题。这是因为较大的批量可能会导致模型过度依赖于批量内部的统计特性，而忽略了数据集的整体分布。较大的批量也可能会使模型更难以逃离局部极小值，从

而限制了其学习能力[19]。

因此，在确定批量大小时需要进行权衡。较小的批量可以提供更多的参数更新和更快的收敛速度，但估计的梯度可能会受到较大的方差影响。较大的批量可以减少方差，但会导致更少的参数更新和更长的训练时间，并可能对模型的泛化能力产生负面影响。选择合适的批量大小需要综合考虑数据集的规模、计算资源、训练时间和模型的泛化需求。

因此，在实际应用中，可以通过调整批量大小并进行实验评估来确定最佳的超参数配置，以获得更好的模型性能和泛化能力。

周期性通信：当使用传统的分布式随机梯度下降（SGD）训练深度神经网络（DNN）时，通信通常在每次迭代的末尾进行。然而，许多研究建议减少梯度和参数的交换频率，以降低通信开销并提高训练效率。

在梯度同步的过程中，每个局部节点上训练的模型参数以一定周期 r 进行平均梯度。传统的梯度同步通常在每轮训练结束时进行，即 r=1。然而，一些研究工作[20]提出将梯度平均的周期设置为 $1 < r < T$，其中 T 是总的训练轮数。通过将梯度平均的周期设置为介于 1 和 T 之间的值，可以减少模型训练过程中的通信开销。

通过减少梯度和参数的交换频率，可以降低通信带宽的要求，减少网络传输的延迟，并提高分布式训练的效率。通过在一定周期内对梯度进行平均，可以减少通信的次数，从而减少训练过程中的通信开销。

然而，选择合适的梯度平均周期 r 也需要进行权衡。较大的 r 值可以降低通信频率，但可能会导致模型更新的延迟和不准确性。较小的 r 值可以提高梯度的准确性，但会增加通信的次数和开销。因此，需要根据具体的应用场景和系统约束来选择适当的梯度平均周期。

综上所述，通过调整梯度平均的周期，可以在分布式训练中减少通信开销，提高训练效率，并在一定程度上平衡模型更新的准确性和延迟。这些技术的应用可以帮助加快深度神经网络的训练速度，提高分布式训练的可扩展性和性能。

模型梯度压缩：模型梯度压缩是一种用于减少分布式深度学习通信量的技术。在分布式深度学习中，多个计算节点共同训练一个模型，这些节点需要在每个训练步骤中共享模型参数的更新。这种通信需要传输大量的数据，通常会成为分布式训练的瓶颈[21]。

模型梯度压缩通过减少传输的数据量来缓解这种问题。具体而言，该技

术通过对模型梯度进行压缩来减少通信量。压缩的方式可以是量化、稀疏化、低秩分解等。

量化是一种常见的压缩方法，它将浮点数转换为较小的整数或定点数。这样可以大大减少传输的数据量，但也会导致精度损失 [22]。为了减轻精度损失，研究人员开发了一些基于量化的方法，如误差反向传播（Error Feedback Backpropagation，EFB）[23] 和 Top-k[24] 选择。

稀疏化是另一种常见的压缩方法，它将模型梯度中的某些元素设置为零。这样可以减少传输的数据量，但也会导致一些信息的丢失。为了保留尽可能多的信息，研究人员开发了一些基于稀疏化的方法，如稀疏梯度蒸馏, 模型剪枝选择 [25]。

低秩分解是一种将原始矩阵分解为多个低秩矩阵的方法，从而减少矩阵的大小。这种方法已经被应用于分布式深度学习中，例如矩阵分解（Matrix Factorization，MF）和低秩近似 [26].

1.2.3 系统层面训练优化

加速 DNN 模型训练: 通信优化在分布式深度学习训练系统中起着至关重要的作用，它可以减少通信开销、降低训练时间，并提高系统的可扩展性。下面介绍几种常见的通信优化方式:

- 使用 RDMA[27] 或 NCCL[15] 来加速单个消息：RDMA（Remote Direct Memory Access）和 NCCL（NVIDIA Collective Communications Library）是一些常用的通信优化技术。RDMA 技术可以直接在计算节点之间进行内存数据传输，绕过 CPU 的参与，从而减少通信的延迟。NCCL 是一种专门为 GPU 集群设计的高性能通信库，它能够充分利用 GPU 的计算能力来加速通信操作。

- 压缩数据传输，例如梯度量化 [24] 和稀疏参数同步 [28]：梯度量化和稀疏参数同步是常用的数据传输压缩技术。梯度量化将浮点数梯度转换为较小的整数或定点数，从而减少传输的数据量。稀疏参数同步则将模型参数中的部分元素设置为零，只传输非零元素的索引和值，从而减少传输的数据量。

- 优化通信方法，例如对于 PS 和不同的 All-reduce 算法研究者们根据架构信息提出了不同的优化加速方式 [29][13]。例如，对于参数服务器

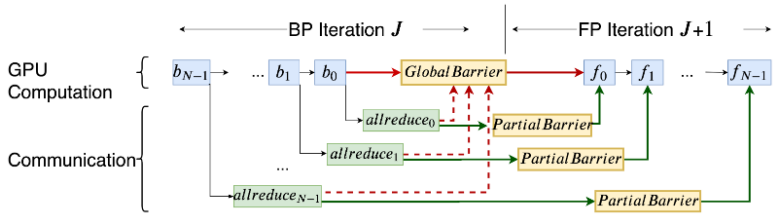

图 1-6　通信计算在 Forward/Backward 过程中重叠（图源 [13]）

（Parameter Server）可以根据深度学习模型的特点，为每一层设计不同的通信方式，调整同步策略，以减少通信开销。而 All-reduce 中可以采用异步通信中维持全局模型一致性的方法优化通信，它可以减少通信的同步开销。

- 通过使用流控制（flow control）[30] 或 Coflow 调度 [31]，最大程度地减少网络流量完成时间：流控制和 Coflow 调度是一些优化网络流量的技术。流控制通过调整发送和接收的速率来避免网络拥塞，减少数据传输的延迟。Coflow 计划则将相关的数据流组合在一起，进行优化调度，以最大程度地减少数据传输的时间。

将通信与计算重叠: 大多数深度神经网络（DNN）框架，如 Tensorflow、PyTorch、MXNet 和 Posei-Don [39]，都支持使用反向传播时的重叠通信。其中，P3[32] 进一步探索了如何通过对 MXNet 的参数服务器体系结构进行图层分区和调度来实现正向传播的重叠。类似地，TIC-TAC[33] 提出了一个类似的思想，但其基于 Tensorflow 的参数服务器（PS）的训练速度较慢（小于 20%）。Bytescheduler[13] 结合模型具体特征调整训练方式，尽可能最大化通信与计算的重叠，从而提升计算速度。并且，Bytescheduler 在多个 DNN 框架中，多种通信方式（RDMA/TCP/IP）都是通用的。作者对已有调度算法进行了分析，克服了它们的缺点，例如不适应系统开销和全局同步的限制，并实现了更高的加速度。Bytescheduler 提供了在多个 DNN 框架中使用的通用优化策略，能够克服现有方法的缺点。该方法充分考虑了系统开销以及全局同步所带来的限制，并采用了有效的调度算法。Bytescheduler 实现了更高的训练加速度，为深度学习研究和应用提供了重要的贡献。SYNDICATE[34] 在 bytescheduler 的基础上更进一步，将网络拓扑与模型架构联合考虑从而实现通信-计算，通信-通信的重叠，从而极大程度的提高了模型训练速度。SYNDICATE 设计了一种有关通信原语代数表示，将原本的通信原语分解为很小的 motifs，以

motifs 为力度调度，从而实现了更加细粒度的调度策略，提升了模型训练系统吞吐量。

内存优化： Zero Redundancy Optimizer（零冗余优化器）[35, 36] 是一种用于深度神经网络训练的优化方法，旨在减少参数服务器（PS）之间的通信冗余，从而提高训练效率。该方法的核心思想是通过在每个参数服务器上只存储和更新模型的一部分参数来减少通信开销。

传统的分布式训练中，每个参数服务器都存储完整的模型参数，并在每个训练步骤中进行同步通信以更新参数。这种全局同步的方式会导致较高的通信开销，尤其是当模型规模较大时。Zero Redundancy Optimizer 则采用了一种新的策略，将模型参数分割为多个部分，并将每个部分分配给不同的参数服务器。这样，每个参数服务器只需要存储和处理分配给它的参数部分，而不再需要与其他服务器进行通信。这样，每个 GPU 只需要存储和处理自己所负责的参数部分，而不需要存储整个模型参数。通过减少每个 GPU 显存中所存储的参数量，从而有效地降低了显存的使用。

Zero Redundancy Optimizer 的关键优势在于减少了参数服务器之间的通信量，从而显著降低了通信开销，提高了训练效率。它允许每个参数服务器独立地更新自己所负责的参数部分，而不需要与其他服务器同步。这种方式可以充分利用计算资源，加速模型训练过程。

然而，Zero Redundancy Optimizer 也引入了一些挑战和限制。首先，由于参数被分割到不同的服务器上，模型更新可能不再是全局同步的，这可能导致一些收敛性问题。因此，需要仔细设计合适的同步策略，以确保模型能够收敛到良好的结果。其次，参数分割的方式需要根据具体的网络结构和训练需求进行设计，这对于大规模模型的优化来说可能是一项复杂的任务。

1.3 国内外研究现状

1.3.1 MoE 训练系统

随着 MoE 训练范式的普及，许多科研机构和企业都开源了 MoE 训练框架和系统。DeepSpeed-MoE 利用多种分布式并行方法结合 MoE 并行性，包括数据并行、张量切片 [37]、Zero 内存优化 [35] 来训练更大的模型。至于 MoE 的推理，DeepSpeed[38] 设计了一种名为 PR-MoE 的新型稀疏激活模型和模型压缩技术来减小 MoE 模型的大小，以及一种有效的通信方法来优化延迟。FastMoE[9] 是一个分布式 MoE 训练系统，它提供了一个分层接口和简单的

机构，说明如何基于数据并行性和张量切片并行性使用 Megatron-LM[37] 和 Transformer-XL[39]。与 DeepSpeed 的实施不同，FastMoE 使用复杂的优化方法来减少网络流量。Fairseq-MoE[40] 是一个序列建模框架，用于训练用于摘要、翻译和语言建模的自定义模型。而 Tutel[41] 在通信和计算方面进一步优化了 Fairseq 系统，其性能提升了约 40%。Tutel 中的优化已集成到 DeepSpeed 中，以促进 MoE 模型训练。

1.3.2 MoE 数据分派策略

MoE 的核心问题之一是如何设计 gating 策略，即如何根据输入分配不同的专家网络。不同的 gating 策略会影响模型的性能、稀疏性、均衡性和公平性。

- **Softmax gating**[9]：这是最简单的一种 gating 策略，它使用一个 softmax 层来为每个输入分配一个概率分布，表示每个专家网络的权重。这种方法可以看作是多个专家网络合作来产生输出，但是也会导致所有的专家网络都被激活，从而增加计算量和内存消耗。

- **Top-k gating**[41]：这种 gating 策略只选择概率最高的 k 个专家网络来处理输入，其他的专家网络则被忽略。这种方法可以实现稀疏性，即只有少数的专家网络被激活，从而节省计算量和内存消耗。但是这种方法也会带来一些问题，比如如何确定 k 的值，以及如何保证每个专家网络都能被充分利用。

- **Noisy softmax gating**[42]：这种 gating 策略在 softmax gating 的基础上增加了一个可学习的噪声权重，用来提高不同专家网络的 gating 均衡性。这种方法可以防止某些专家网络被过度使用或者被忽略，从而提高模型的公平性和泛化能力。

- **Hierarchical softmax gating**[42]：这种 gating 策略将多个专家网络组织成一个层次结构，每一层都有一个 softmax gating 来决定下一层的激活。这种方法可以减少 softmax gating 的计算复杂度，从而提高模型的效率。

- **Hash layer**[43]：这种 gating 策略使用哈希函数来为每个输入分配一个或多个专家网络，而不需要学习任何参数或者使用额外的损失函数。这种方法可以实现极高的稀疏性和效率，同时保持或者提升模型的性能。

- **Topology-aware gating**[44]: TA-MoE 中提出了一种拓扑感知的路由策略，它能够根据网络拓扑的变化动态地调整 MoE 的数据分派的调度策略。通过基于通信建模的方法，TA-MoE 将调度问题抽象为一个优化目标，并得到了适用于不同拓扑结构的近似调度模式。他们设计了一种拓扑感知辅助损失函数，它可以自适应地根据底层拓扑调整数据分派策略，而不会牺牲模型的准确性。

1.4 研究目标

本课题针对 MoE 模型带来以上挑战，拟分析现有的 MoE 模型分布式训练系统存在的缺陷，并设计一套全新的 MoE 模型训练系统。

MoE 模型在分布式训练中面临几个挑战。首先，MoE 模型的稀疏激活特性与现有的静态并行策略不匹配，导致计算资源无法充分利用。其次，MoE 模型引入了全局所有 GPU 之间同步的 All-to-All 通信，严重影响训练速度和效率。最后，由于 Gating 策略的动态变化，节点负载可能不均衡，导致训练时间延长和模型性能下降。因此，需要采用动态并行策略、优化通信开销和实现负载均衡，以克服这些挑战，提高 MoE 模型的训练效率和性能。

如图1-7所示，我们提出了一种全新的 MoE 训练系统，通过在算法层面（gating policy）和系统层面（Expert placement scheduler）提出创新的训练解决方案，旨在克服 MoE 模型训练过程中的瓶颈，并更好地适应复杂而庞大的深度学习任务需求。

我们的训练系统综合考虑了多个因素，包括任务特性、训练系统的拓扑结构、训练数据的分布以及每个专家的能力。通过算法与系统的协同设计，我们致力于实现能够动态优化专家的分布式并行策略，使每个专家都能够发挥最大潜力，并在整个系统中平衡负载和资源利用率。

在算法层面上，我们对 MoE 模型的数据分派关键部分，即 gating 策略，进行了改进。通过动态调整数据分派策略，我们实现了更快的收敛速度，并同时控制了全局通信开销。我们采用了 All-to-All 通信原语中的 unequal 模式，进一步减少由填充（padding）引起的额外数据发送量，从而降低全局通信的成本。

同时，在系统层面上，我们提出了一种基于网络拓扑的自动负载均衡策略。我们根据训练系统实际的网络拓扑结构，研究了不同拓扑结构下的数据传输和通信开销，并针对性地设计了一种优化策略。通过充分利用数据的局

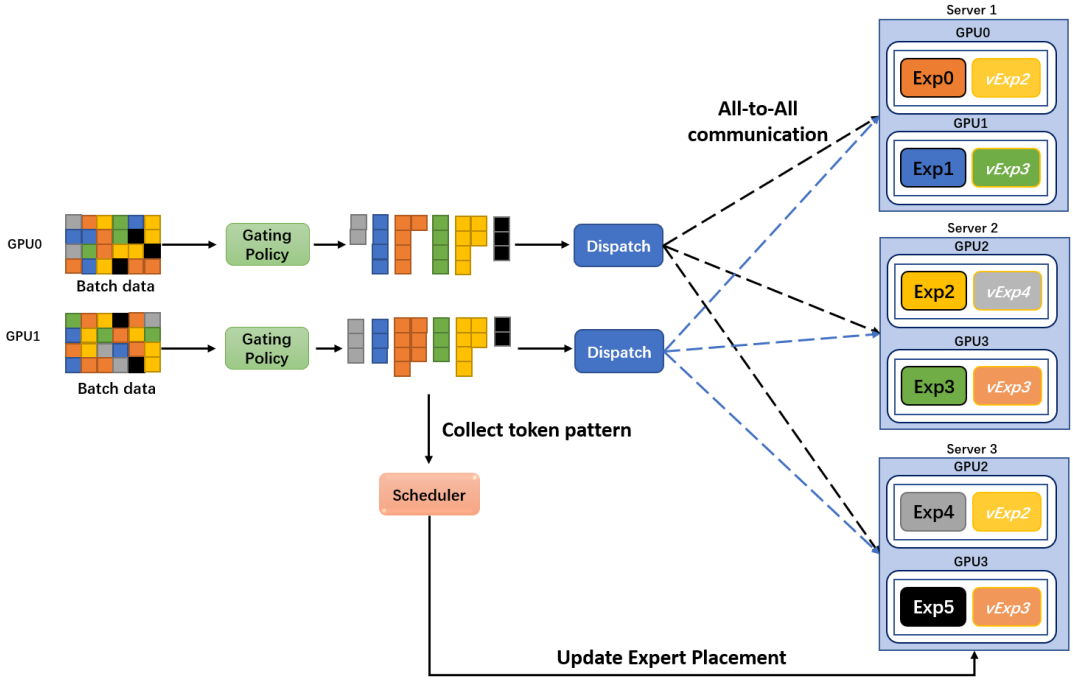

图 1-7　设计的 MoE 训练框架

部性和专家之间的并行计算能力，我们能够减少通信开销，提高整体训练效率。我们的设计在 offline 的过程中，首先根据系统的网络拓扑结构，分析训练系统中任意两点进行通信操作的开销，并建立数学模型。在 online 过程中，我们设计了一种调度器（Expert placement scheduler），它能够收集并监测每一个 MoE 层中所有专家的任务负载。当检测到专家的负载发生一定变化时，调度器会根据当前负载情况和专家的放置情况，自动寻找最合适的专家并行放置策略，从而解决 MoE 模型在训练过程中由于负载动态变化而导致的训练效率低下的问题。

这种全新的训练解决方案使得 MoE 模型能够更好地应对复杂和庞大的深度学习任务需求。通过优化算法和系统设计，我们能够充分发挥 MoE 模型的潜力，提高模型的准确性和泛化能力，为解决现实世界中的复杂问题提供有力工具。我们的研究对推动 MoE 技术的发展和应用具有重要意义，也对分布式训练系统的进一步发展起到推动作用。

1.4.1 基于动态路由的数据分派策略

如图1-8所示在传统的 MoE 模型中，采用固定的 Top1 和 Top2 Gating 策略，即将数据发送到分数最高（次高）的 expert 进行处理。然而，采用 Top1

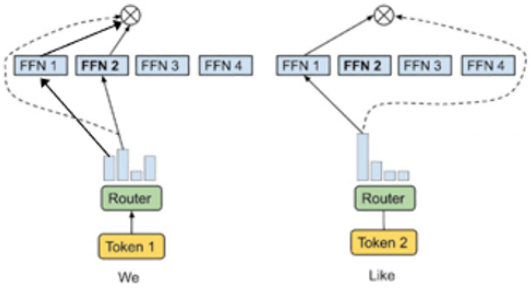

图 1-8　传统的 Top1 和 Top2 Gating 策略

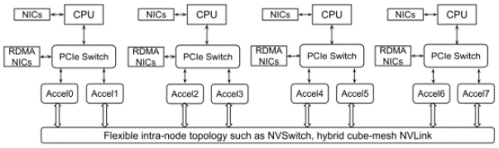

图 1-9　GPU 集群内网络拓扑连接图（图源 [34]）

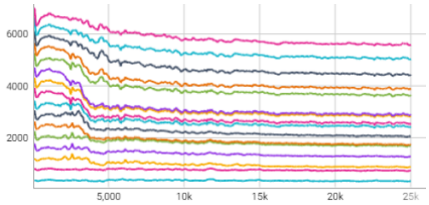

图 1-10　8 层 128 专家的 Transformer-XL[39] 的 MoE 模型, 第 1 层 16 个专家负载变化情况

Gating 时，由于只选择一个最高分数的 expert，可能会错过其他有价值的信息，导致模型收敛速度较慢；而采用 Top2 Gating 时，虽然可以选择两个最高分数的 expert，但每轮训练时间较长，因为需要进行两次 All-to-All 通信。因此我们能否将 Top1 和 Top2 Gating 策略结合起来，以一种动态的方式选择合适的数据分派策略, 从而实现较快的收敛速度和较短的训练时间。一种简单的方法是，将每个数据按照 Top1 和 Top2 的分数进行排序，然后将它们分别分配给 Top1 和 Top2 的 expert 进行处理。这种方法可以利用 Top1 和 Top2 的优点，避免错过其他有价值的信息，同时减少通信开销，提高训练效率。

1.4.2 基于网络拓扑的自动负载均衡策略

虽然基于 MoE 的算法开辟了一个巨大扩展模型参数量机会，但它也对训练系统带来了新的挑战，而这些挑战在之前的密集型 DNN 训练算法和系统中从未见过。根本原因是动态专家选择和灵活的 MoE 结构。具体来说，每个 MoE 层由一定数量的并行专家组成，这些专家分布在加速器（本工作中的 GPU）上，其中每个 GPU 根据智能门函数将每个输入数据分配给几个最适合的专家并取回相应的输出以将它们组合起来。这意味着每个专家的工作量基本上是不确定的，取决于输入数据和门函数。在图1-10中，我们使用了一个 8

层的 Transformer-xl MoE 模型，分析了第 1 层 16 个专家负载变化情况。

我们发现，在 MoE 模型中，不同专家的负载在每次迭代中都会发生变化。这种现象是由于 MoE 的稀疏性动态数据分配所导致的，导致每个专家获取的数据不均匀，从而产生了负载不均衡的问题。这会导致一些节点或 expert 的负载过重，从而影响整个模型的训练速度和效果。因此，需要采取适当的负载均衡策略来缓解这个问题。此外在 MoE 模型中所有专家都需要从其他 GPU 上那里获得输入，这引入了 GPU 集群所有节点间额外的 All-to-All 通信，且 All-to-All 通信与后续计算是完全同步关系，并行较差。而 GPU 集群内部节点之间的网络带宽并不相同，节点通信效率存在差异。因而 All-to-All 通信也成为了大规模 MoE 训练中最耗时的操作之一。它通常实现为具有可变消息大小的同步 All-to-All 操作。考虑到动态特性的数据分派会导致计算和通信的严重不平衡，这样的方法会导致严重的开销。

因此，为了解决负载不均衡问题，需要采取一些适当的负载均衡策略，我们需要结合 GPU 集群内部节点之间的网络带宽不相同的情况，选择最佳的负载均衡策略，以提高整个模型的训练效率和性能。

1.5 本文组织结构

本文的组织结构如下:

本文的第二章是 MoE 训练过程的概述。该章主要介绍了 MoE 混合专家模型在分布式深度学习训练中的主要过程, 分析了其正向反向传播的计算过程, 并解释了为什么 MoE 模型与现有传统集中式训练系统的不匹配。

本文的第三章是基于动态路由的数据分派策略的设计方式。该章首先分析了现有的数据分派策略的主要方式, 并总结了现有设计的不足之处。进而更进一步，给出了我们设计的基于动态路由的数据分派策略具体的方案架构, 包括方案的详细流程和性能分析。

本文的第四章为基于网络拓扑的自动负载均衡策略方案设计。该章首先分析了现有 GPU 数据中心网络拓扑。并结合每个专家在训练过程的负载变化现象, 建立模型寻找最佳的负载均衡方案设计。

本文的第五章为系统实现以及实验结果展示。该章给出了整体的设计方案以及实验结果展示。

本文的第六章为总结与展望。该章总结了本文的主要贡献，以及对未来工作的展望。

1.6 本章小结

本章首先对论文选题的背景和研究意义进行讨论，提出了稀疏混合专家 MoE 模型对于现有深度学习训练系统的挑战。随后对国内外研究现状进行了简要的介绍。最后我们介绍了本工作的研究内容和研究目标。我们分析了 MoE 模型在分布式训练系统中的挑战，并介绍了设计的全新 MoE 模型训练系统。该系统在算法层面和系统层面提出了创新的训练解决方案，旨在克服 MoE 模型训练过程中的瓶颈，并更好地适应复杂而庞大的深度学习任务需求。

第二章　MoE 模型训练过程概述

MoE 模型在原本 Transformer block 的 FFN 中通过添加门控函数以及一系列相互并列的专家组成。在训练过程中门控函数主要将输入数据分配给不同的专家模型参与前向/反向计算，这一过程主要通过 All-to-All 通信实现。待每个专家计算完成相对应的数据之后，我们需要通过一次额外的 All-to-All 通信，将计算完成的激活值返回原本对应的 GPU 上，并参与后续的计算过程。

2.1 MoE 模型的分布式训练

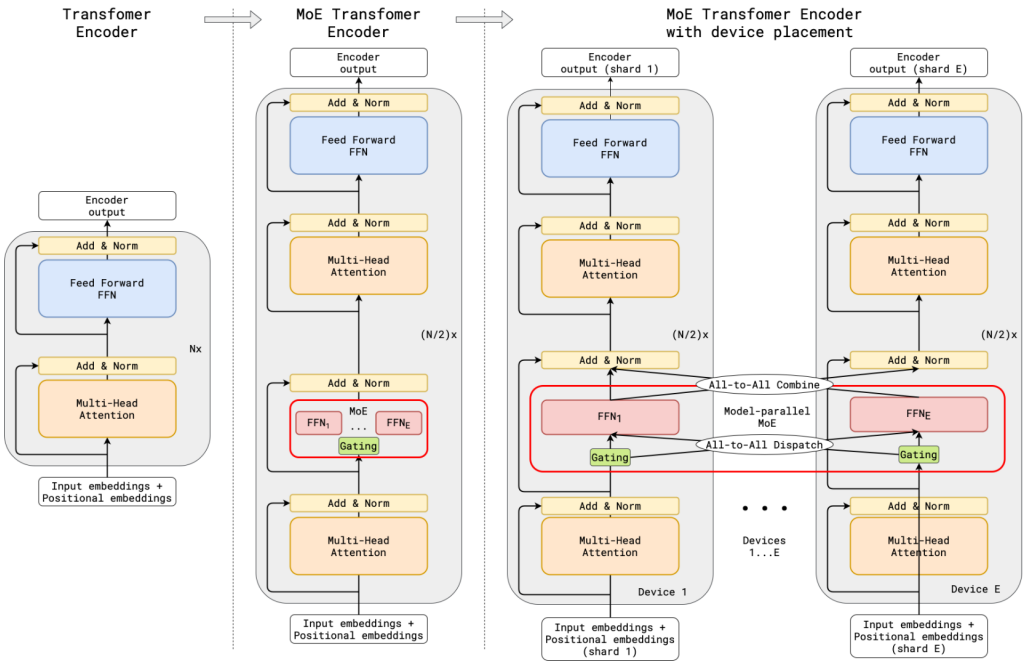

图 2-1　MoE 模型训练过程

图2-1展示了传统的集中式的 Transformer 模型训练过程转变到分布式的 Transformer-MoE 训练过程的示意图。绿色的部分是增加的门控函数，用于决定将数据分派至哪一个 expert 参与计算，红色部分的 $\mathbf{FFN_1}$ – $\mathbf{FFN_E}$ 表示该层中具有 E 个互相独立的专家个数。由于 GPU 的显存有限，不能将所有的专家的参数都在一张 GPU 中保存下。因此 GShard[6] 的研究者们提出了专家并行的概念，将每个专家的参数单独放置在 GPU 上，其他部分采用数据并行的架构，从而实现了一种混合的分布式训练过程。在训练过程结束之后，通常需

要额外的 All-reduce 用于同步数据并行模块的梯度，最后根据每个可训练参数的梯度，调用优化器更新参数。

整体的训练流程总结如下:

- **1. 数据划分**

  每个 GPU 将训练数据随机筛选，确保每一轮训练中每个 GPU 上包含的数据子集是互不重复的。每个子集分配到不同的 GPU 或计算节点上进行训练。

- **2. 数据并行 & 专家并行**

  在 MoE 模型训练中，数据并行和专家并行通常结合使用，以实现更高效的分布式训练。具体而言，可以将训练数据划分为多个子集，并将每个子集分配到不同的 GPU 或计算节点上。在每个 GPU 或计算节点上，可以使用专家并行的方式对每个专家模型进行训练，同时使用数据并行的方式对整个 MoE 模型进行训练。这样可以将计算负载和数据负载都分散到多个 GPU 或计算节点上，从而加速整个 MoE 模型的训练过程。

- **3. 前向传播**

  每一层的前向传播主要在以下两个步骤区别与传统的密集型 Transformer 计算。(数据分派和专家计算)。

  在数据分派阶段，MoE 模型使用 Gating 门函数对每个输入数据进行加权打分，以决定每个专家模型的贡献。这个 Gating 函数由一层 MLP 和 softmax 组成。具体而言，MoE 模型将输入 x 输入到门控模型中，得到一个 E 维的得分，$f = [f_1, f_2, \cdots, f_E]$，$f_i$ 表示第 i 个专家模型的权重得分。门控向量的每个元素都是非负的，并且它们的和等于 1，即 $\sum_{i=1}^{K} g_i = 1$。最后通过全局所有 GPU 的 All-to-All 通信，将对应数据发送给相应的 GPU，参与后续的计算。

  专家计算阶段，每个 GPU 上接收到其他 GPU 传输来的数据后（假设有 d 维），将其输入相应的专家模型参与前向计算。得到 d 维的输出向量 r。之后通过反向的 All-to-All 通信，将激活值返回对应的 GPU 上，得到每个 GPU 上原本数据的输出向量 $z = [z_1, z_2, \cdots, z_E]$。然后，MoE 模型将每个输出向量 z_i 与对应的门控向量 g_i 进行按元素乘法，得到一个

加权输出向量 w，其中 $w_i = g_i z_i$。最后，MoE 模型将所有加权输出向量 w_i 进行累加，得到最终的输出向量 y，其中 $y = \sum_{i=1}^{E} w_i$。

- **4. 反向传播**

  是用于计算 MoE 模型的梯度，其过程类似于传统的神经网络模型。在完成前向传播计算之后，使用 PyTorch 的自动求导机制（autograd）建立计算图，并自动计算每个参数的梯度。具体而言，可以使用 loss 函数对模型的输出进行评估，并计算输出和目标值之间的误差。然后，使用误差及其对模型参数的导数，计算每个参数的梯度。反向传播过程可以使用链式法则（chain rule）实现，即将误差从输出层向输入层传播，并依次计算每个参数的梯度。

- **5. 梯度计算**

  是将每个模型的梯度进行同步，以便在参数更新时使用。由于 MoE 模型的分布式训练过程涉及多个 GPU 或计算节点的并行计算，因此需要使用 all-reduce 等方法将所有模型的梯度进行同步。

- **6. 参数更新**

  使用优化器对模型参数进行更新，以最小化损失函数。在 MoE 模型训练中，可以使用常见的优化器，如 Adam、SGD 等，对每个模型的参数进行更新。通常，参数更新的速率会受到学习率（learning rate）等超参数的控制，以平衡模型的收敛速度和稳定性。

- **7. 重复迭代**

  将以上步骤重复多次，直到模型收敛为止。在每次迭代中，可以使用不同的训练数据子集，防止模型陷入局部最优解。

2.2 MoE 训练瓶颈分析

在 GPU 集群中分布式训练 MoE 模型时，我们首先测量分析了一些已有训练系统存在的缺陷和问题，这些挑战会极大地影响训练效率。

All-to-All 通信效率低 MoE 模型的训练瓶颈之一是 All-to-All 通信，这是由于模型中多个专家模型之间需要进行信息交换所导致的。之前的工作已经确定了 All-to-All 通信是 MoE 模型训练的一个瓶颈 [9, 41, 45]。表 2-1 显示了在我们的 GPU 集群中各种语言模型的训练步骤时间以及 All-to-All 操作的完成时间。

表 2-1 使用 Transformer-xl [39] 语言模型再不同模型设计下的 All-to-All 通信时间与每个训练阶段完成时间。每个 FFN 层都使用 MoE 替代，且专家的数量等于 GPU 的数量。

#Experts	Model	All-to-All	Step Time	Ratio
4	8L + 94M	259	722	35.80%
	12L + 117M	589	1684	34.90%
	24L + 233M	1479	3894	37.80%

平均而言，All-to-All 操作占用了 37.4% 的步进时间，这是很大一部分。在下文中，我们通过剖析 MoE 中 All-to-All 通信成本的主要原因来引入我们的工作。我们的分析基于专家数量与 (GPU) 设备数量相同的常见场景。

首先，All-to-All 操作是同步操作，会阻塞计算过程。图2-1显示了集群中 MoE 模型训练前向传递的流程图。我们观察到专家的 FFN 计算和组合操作仅在 All-to-All 操作完成时发生。在此期间，GPU 大部分处于空闲状态。我们使用 PyTorch Profiler 对表 2-1 中每个实验中 20 个步骤的 GPU 活动进行了分析，并发现这种空闲状态很明显。在 MoE 模型的前向传递中，会使用两个 All-to-All 操作，一个用于将标记从之前的 Add & Norm 层输出结果路由到选择的专家，另一个用于在专家计算后将它们发送回其原始 GPU。因此，一个 MoE 层的完整训练步骤将涉及四个 All-to-All 操作，其中两个来自反向传递。这加剧了 MoE 模型训练的低效率问题。

其次 MoE 训练过程瓶颈的第二个原因是其庞大的数据传输规模。由于专家的 FFN 架构确保其输入数据大小与输出数据大小相同，因此前向传递的两个 All-to-All 操作中的数据传输具有相同的大小。数据传输的大小由每个 GPU 的 *batch_size*、专家（或 GPU）的数量 N、序列长度 *seq_len*, 如 top-k 中的 k(所选专家的数量) 以及编码器, 解码器输入 *d_model* 中的特征大小决定。

2.3 本章小结

本章我们主要分析了现有的 MoE 模型训练过程，MoE 模型的训练过程通常比传统的神经网络模型更为复杂和耗时，需要充分利用分布式计算和并行计算等技术，以提高训练速度和效率。同时，MoE 模型的设计和调整也需要考虑多个因素，如门控模型的设计、专家模型的选择和训练方式等，以提高模型的性能和泛化能力。

第三章　基于动态路由的数据分派策略

在 MoE 模型的每一层前向/反向传播中，都有一个计算各个专家权重的门网络，他的作用主要是根据每个输入样本的特征来预测每个专家的权重或者分配的系数。但是门网络的数据分派方式又会影响到全局通信量，因此如何设计合适的数据分派方式，在保证模型收敛速度的同时，不会带来巨大的系统总通信量，使值得深入研究的问题。

3.1 MoE 数据分派方式概述

门网络的架构通常是 MLP+Softmax，通过学习可调节的 MLP 权重参数，预测各个计算专家的权重，从而实现对输入样本的有效处理。

3.1.1 数据分派流程

在 MoE 模型的训练过程中，他的过程如下：

- **专家容量设置**：在 MoE 模型训练开始之前，人为地为每个专家设定一个容量的限制。这个容量限制可以根据每个专家的计算资源、存储能力或其他约束条件来确定。当数据量超过专家的容量限制时，系统会对数据进行强制截断，以确保专家在其容量范围内进行计算。通过设置专家容量限制，可以控制每个专家参与计算的数据量，从而平衡计算资源的使用和模型的性能。

- **输入特征**：：输入样本的特征被提供给门网络作为输入。通过学习可调节的权重参数，门网络预测每个专家的权重。这些权重参数在训练过程中通过优化算法进行调整，以最佳地预测专家权重。

- **权重计算**: 具体而言，MoE 模型将输入 x 输入到门网络的 MLP 层，之后通过 Softmax 函数规范化得到一个 E 维的得分，$f = [f_1, f_2, \cdots, f_E]$，其中 f_i 表示第 i 个专家模型的权重得分。且权重之和等于 1，即 $\sum_{i=1}^{K} f_i = 1$。

- **Top-k 分派**: 根据门网络的输出，通常采用 Top-k gating 的方式将数据分派给具体的专家进行计算。一种常见的方式是使用 Top-1/Top-2 Gating，

根据专家权重得分从高到低对专家进行排序，并将数据发送到选择的前 k 个专家中参与后续计算。

- **系统开销**: 由于采用 Top-k Gating，整个系统的通信数据量是 Top-1 All-to-All 通信量的 k 倍。因为 All-to-All 通信是全局的通信操作，需要权衡模型的收敛速度和系统的通信量，选择适当的 gating 策略。

这些步骤组成了 MoE 混合专家系统中数据分派和权重计算的基本过程，使得不同专家能够根据其权重参与输入样本的处理和计算。

3.2 动态路由的数据分派协议设计与实现

动态路由是一种用于数据分派的系统设计和算法，它可以根据数据特征和专家模型的状态动态地分配数据到合适的专家进行计算。如图3-1所示，我们展示了设计的动态路由的数据分派协议示意图，主要我们引入了一个 Judge 模块，用于比较 Router 模块打分的结果，并最终决定将数据分派至哪一（几）个 expert 参与计算。

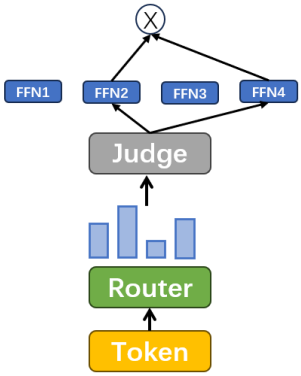

图 3-1　动态路由的数据分派协议设计

3.2.1 协议设计

主要设计包含以下步骤：

1. 初始阶段：类似于传统的 gating 策略，系统在计算开始之前需要进行一些初始设置。这包括设定专家容量，确定输入特征，并通过 MLP 和 Softmax 层计算权重（如图3-1中的输入 Token 以及 Router 计算过程）。这些权重代表每个专家在当前数据上的重要程度或贡献度。

2. 动态数据分派：与传统的固定分派模式不同，动态路由的数据分派是根据门网络的打分结果进行动态决策。在分派数据时，根据权重得分 $f = [f_1, f_2, \cdots, f_E]$ 从高到低排列，选择得分最高的两个专家，记为 f_{k1} 和 f_{k2}。如果 $f_{k1} - f_{k2}$ 小于预先设定的阈值 Threshold，系统将采用 Top-2 Gating 的方式将数据发送给这两个专家参与计算。否则，系统按照 Top-1 Gating 的形式选择得分最高的专家，将数据发送给该专家进行计算。即系统动态地选择使用 Top-k（k=1 或 2）的专家参与计算，根据权重得分的差异性来确定采用哪种分派模式。

3. 自适应数据发送: 在发送数据时，使用 All2All Communication 的 unequal 模式，这样可以自适应有效数据长度，从而减少发送不必要的数据（即用于额外 padding 的 0），减少通信量，提高通信效率。

3.2.2 系统实现

对于系统实现而言，采用动态路由的方式确实会带来一些困难，我们在传统的 Top-k 路由的基础上做出了一些修改以支持动态路由的设计。

(1) Judge 模块实现

Judge 模块接受来自路由器的每个专家（expert）的评分结果，并根据一定的规则进行筛选和判定。该模块的目标是选择最合适的专家进行进一步的决策。

具体而言，Judge 模块首先从所有专家的评分结果中选出排名最高的两个专家（即 top1 和 top2），并获取对应的评分分值。然后，它会比较 top1 和 top2 的分值差异，判断它们之间是否足够显著。在比较过程中，Judge 模块会使用预先定义的阈值进行判断。如果 top1 和 top2 的分值之间的差异小于该阈值，说明它们的评分非常接近，此时将选择 top2 gating 策略。这意味着在进一步的决策中，会考虑 top2 专家的意见和贡献。相反，如果 top1 和 top2 的分值之间的差异大于预先定义的阈值，说明它们的评分有较大的差异，此时将选择 top1 gating 策略。这意味着在进一步的决策中，会主要考虑 top1 专家的意见和贡献。

通过这样的判定过程，Judge 模块能够动态地根据专家评分的差异性选择适当的策略，以最大程度地利用专家的知识和经验，从而对问题进行更准确的决策。

(2) Unequal All-to-All 通信实现

在传统的 Top-k Gating 中，每个 GPU 在 All-to-All 通信时发送的数据量是均等的，因此在系统实现时比较容易，可以直接调用 PyTorch（NCCL）的 All-to-All 通信实现来完成通信过程。

然而，在采用动态路由后，每个 GPU 上发送的数据量是不均等的。如果仍然按照传统的 Top-2 Gating 的方式发送数据，实际上并未减少通信量。为了解决这个问题，我们采用了 All-to-All 通信的 unequal 模式，该模式不会发送额外的用于填充的 0，从而达到减少数据量的目的。

通过实现 unequal 模式的 All-to-All 通信，我们可以根据动态路由的结果，灵活地分派数据并减少通信量。这在系统实现上可能会带来一些挑战，在实现过程中，需要对通信模块进行适当的调整，以支持 unequal 模式的数据传输。

通过在算法上实现动态路由的数据分派模式，在系统上实现 unequal All-to-All 通信模式。我们可以提高系统的通信效率和计算性能。这样，系统能够根据实际的数据分派需求，灵活地选择数据发送的方式，并减少不必要的通信开销。

3.3 本章小结

MoE 模型中的数据分派方式对训练系统的通信量和模型的收敛速度有重要影响。传统的 Top-k 分派方式在选择专家进行计算时可能会导致不均衡的通信量。为了解决这个问题，我们采用动态路由的数据分派算法设计。该设计通过动态决策，根据门网络的权重得分选择合适的分派方式。具体而言，我们的 Gating 模块根据权重得分的差异性选择 Top-1 或 Top-2 分派方式，从而灵活地分配数据给专家进行计算。在系统实现上，可以采用 unequal All-to-All 通信模式，减少通信量。通过这样的设计和实现，可以提高系统的通信效率和模型的训练速度。

第四章　基于网络拓扑的自动负载均衡策略

在 MoE（混合专家）训练模型中，专家的计算负载不均衡问题可能会对模型的训练效果产生影响。如果某些专家的计算负载过高，可能会导致模型训练变慢，甚至出现不收敛的情况。为了避免这种情况的发生，需要设计一种负载均衡算法来实现专家之间的计算负载均衡，实现较好的系统整体性能。

当设计负载均衡算法时，除了考虑专家的负载变化情况，还需要考虑训练系统的实际拓扑连接情况。不同的拓扑结构可能对负载均衡策略有不同的影响。在分布式拓扑中，不同的专家可能分布在多个节点上，通过网络进行通信。在这种情况下，负载均衡的策略需要考虑节点之间的通信开销和计算资源分配的平衡。

因此我们需要综合考虑专家负载、拓扑结构和通信开销之间的关系，选择合适的策略来实现负载均衡。这样可以提高训练系统的效率，减少通信开销，并加快模型的收敛速度。

4.1 负载均衡算法概述

4.1.1 GPU 集群网络拓扑结构

Spine-leaf 架构是一种常见的数据中心网络拓扑结构，用于构建高性能的 GPU 集群。它将数据中心网络划分为两个层级：Spine 层和 Leaf 层。在该拓扑结构中，节点之间的通信是基于 TCP/IP 协议的以太网通信。

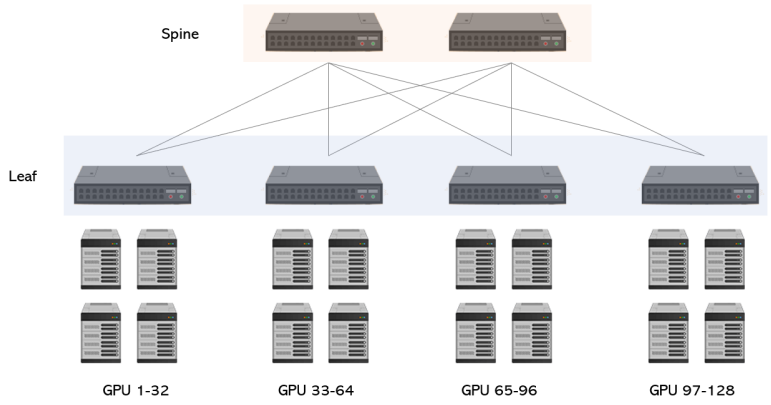

图 4-1　Spine-leaf GPU 数据中心架构

Spine-leaf 架构中，Spine 层和 Leaf 层之间通过高速物理链路连接。每个 Leaf 节点连接多个 GPU 设备 (一般为 ToR 交换机)，而每个 Spine 节点则连接多个 Leaf 节点。通信过程中，数据包从源节点的 GPU 设备开始，通过本地的交换机和链路，经过目标节点的交换机，最后到达目标节点的 GPU 设备，以完成通信。

在 Spine-leaf 架构 GPU 拓扑结构中，通信可以分为两种类型：内部通信和外部通信。

- **内部通信**：指的是同一台 Leaf 节点内部的 GPU 设备之间进行的通信（在同一机架内），这种通信只需要经过本地的交换机和链路即可完成。
- **外部通信**：是指不同 Leaf 节点之间的 GPU 设备之间进行的通信（跨机架 GPU 通信），这种通信需要经过 Spine 节点和目标 Leaf 节点的交换机和链路，这增加了传输的延迟和可能的网络拥塞。通常会采用高速的物理链路和交换机来连接 Spine 节点和 Leaf 节点，以提供相对较高的传输带宽以确保数据包能够到达目标 GPU 设备，但通信延迟和带宽可能相对较低。

Spine-leaf 架构 GPU 拓扑结构的优势在于提供了高性能、低延迟的通信，同时具有良好的可扩展性。该拓扑结构能够满足大规模 GPU 集群的通信需求，并支持高吞吐量的数据传输。通过合理设计和配置 Spine-leaf 架构，可以实现有效的负载均衡和网络流量管理，以提高 GPU 集群的整体性能和可靠性。

4.1.2 GPU 节点通信模型设计

我们在 GPU 集群的架构上，使用了 Alpha-Beta 模型建立集群节点通信模型，Alpha-Beta 模型是一种常见的建模方法，它可以有效地描述集群节点之间的通信行为，并为集群性能优化提供重要的参考依据。这样的建模方式，通过估计节点间通信成本，可以表示出任务的数据通信需求和节点的资源状况，从而优化任务调度和负载均衡策略，以提高整个集群的性能和效率。

(1) Alpha/Beta 的含义

- **Alpha**: 表示数据包在发送过程中的传输时间，即数据包从发送节点到达接收节点的时间延迟。它可以涉及网络传输延迟、路由器处理时间、链路拥塞等因素。

- **Beta**: Beta 表示数据包的传输带宽或吞吐量，即在单位时间内可以传输的数据量。它通常与网络带宽、链路容量以及节点处理能力相关。

(2) Alpha/Beta 如何测量

在建立集群节点通信模型时，可以使用 Alpha-Beta 模型来估计节点之间数据包发送的成本。这个成本可以通过以下因素进行建模：通常我们可以连续发送 k 次大小为 M 的数据包，测量总共的发送时间 T_1, 此时 $T_1 = k * (alpha + \frac{M}{Beta})$。之后发送一次大小为 K*M 的数据包，测量发送时间 T_2, 这里的 $T_2 = alpha + \frac{k*M}{Beta}$, 之后联立以上两个方程，即可求解出任意两个节点的 alpha/beta 值。

此后任意两节点发送数据的大小即可使用求解出的 alpha/beta 进行表示。其中相同节点内的 GPU 之间具有较小的 alpha/beta 值，而节点之间的 GPU 传输具有较大的 alpha/beta 值。

4.1.3 MoE 训练系统专家负载变化情况

在 MoE 训练中，每个子模型都是一个专家，需要处理一定比例的输入数据。不同的子模型可能具有不同的负载，即它们需要处理不同数量的输入数据。这是因为在 MoE 模型中，每个子模型负责处理不同类型的输入数据，因此在训练过程中，每个专家的负载可能会根据输入数据发生变化。

此外，同一专家的负载在不同时刻也会发生变化。这是因为在 MoE 训练中，子模型的权重和负载是动态调整的，以适应不同的输入数据分布和模型性能。因此，在训练过程中，同一专家的负载可能会根据模型的状态和数据的分布发生变化。

我们使用 Transformer-xl 模型 [39], 将其原本的模型拓展为 MoE 架构，并对其第一层 16 个专家的负载情况做了统计。如图1-10所示，我们发现随着时间的变化 MoE 模型的负载也随之变化，并不是静态不变的，有部分专家的负载甚至接近 0，因此我们需要根据负载的变化情况，寻找最合适的策略。

为了解决 MoE 训练中的负载不均衡问题，可以采用一些负载均衡技术，例如动态调整权重、分配数据等，以确保每个子模型都能够得到足够的训练和优化。此外，还可以通过调整 MoE 模型的结构和参数来平衡不同专家之间的负载，以提高模型的训练效率和准确度。

以下我们举一个例子来说明我们的自动负载均衡设计的原因：假设我们有一个由 4 个节点和每个节点上有 2 块 GPU 组成的 GPU 集群。初始状态下，

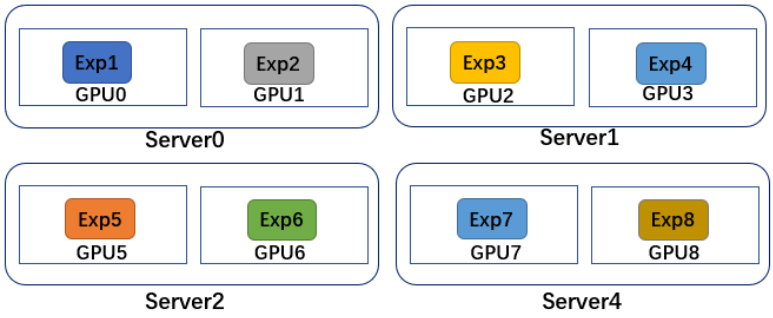

图 4-2　4 节点 8 卡 MoE 模型分布式训练架构（Baseline 训练策略）

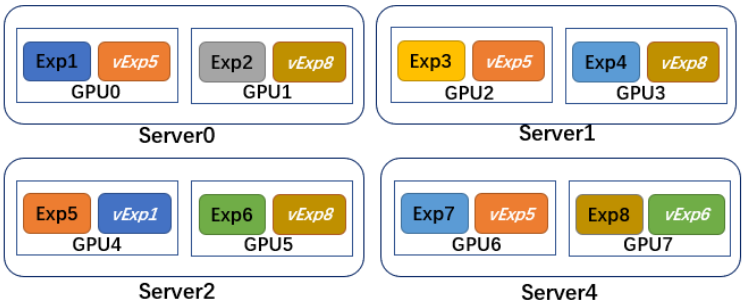

图 4-3　4 节点 8 卡 MoE 模型分布式训练架构（使用负载均衡策略）

我们将每个 GPU 上放置一个专家（Expert），如图4-2所示。现假设 Experts1-8 的负载如下：210、312、200、198、415、150、189、250。我们分析跨服务器通信的开销。

在没有应用任何负载均衡策略的情况下（即图4-2所示的放置方式），在一轮迭代中跨服务器通信的开销为 $(210+312+\cdots+250)*6=11544$。

为了实现负载均衡，我们引入了虚拟专家（virtual expert）的概念，将虚拟专家同样放置在 GPU 服务器上，以平衡负载。如果在同一台服务器上存在专家或虚拟专家，我们首先选择将数据发送到该专家/虚拟专家（即服务器内部通信）。假设我们采用了一种负载均衡策略，如图4-3所示，现在每块 GPU 上除了原有的专家参数外，还有虚拟专家。我们计算应用该策略后，跨服务器的 GPU 通信开销为 $1474+1772+1798+1840=6834$。与原始通信量相比，通过应用负载均衡策略，我们可以将跨服务器通信量减少了 40

通过这个例子，我们可以看出应用负载均衡策略可以显著减少跨服务器的通信量。然而，同时也带来了一个问题：不同的策略（虚拟专家的放置方式）可以带来显著的性能改进，但如何选择最佳策略成为一个挑战。

因此，我们的自动负载均衡设计的目标是通过自动搜索和选择最佳的负载均衡方案，以最大程度地减少跨服务器通信开销，并提高深度学习分布式训练的效率和性能。

4.2 自动负载均衡机制设计与实现

4.2.1 机制设计

基于以上考虑，我们设计了一种基于网络拓扑的自动负载均衡策略，我们将 MoE 训练中涉及到的 All-to-All 通信过程建立成 alpha-beta 数学模型。由于 MoE 模型训练中通信是主要的开销，因此我们将训练的问题建模为组合优化问题，通过求解器求解基于当前负载情况的最合适的负载均衡策略。

我们对 MoE 模型训练的每一层都建立有一个优化问题。下面以一层为例，介绍优化模型的主要内容。优化模型的目标函数为全局节点之间的 All-to-All 成本最小。

(1) 优化问题的输入数据

优化模型的主要输入信息为:

1. 专家总数为 E
2. 设备总数为 P
3. 一张 GPU 上一个 MoE 层最多可以放置的专家数目为 L
4. 节点 i 与节点 z 之间的建立的网络模型的 alpha 值 α_{ij}，用于表示任意两点网络连接的时间
5. 节点 i 与节点 z 之间网络带宽的 beta 值 β_{ij}，表示网络传输速率;
6. 当前系统每个专家的负载情况，即可表示为 GPU i 到专家 e 总发送数据比例 $C_{i,e}$

(2) 优化问题的求解参数

优化问题的求解参数主要是每张 GPU 上可以放置哪些 expert，以及每张 GPU 上原本拥有的数据需要发送到对应目标 GPU 上的比例，用字母表示出来分别是:

1. 放置矩阵 R，表示每个专家应该放置在哪些 GPU 上。其大小为 $P \times E$，表示有 P 个设备和 E 个专家。

$$R \in \mathbb{Z}^{P \times E}$$

2. 发送数据矩阵 W，表示数据从一个 GPU 到另一个 GPU 的比例。其大小为 $P \times P \times E$，表示有 P 个设备，每个设备发送数据到其他 $P-1$ 个设备中的 E 个专家。

$$W \in \mathbb{R}^{+P \times P \times E}$$

(3) **优化问题的约束条件**

1. 每个 GPU 上放置的专家数量不超过 L:

$$\sum_{k=1}^{E} R_{j,k} \leq L, \quad \forall j = 1, 2, \ldots, P$$

其中 $\sum_{k=1}^{E} R_{j,k} \leq L$ 表示对矩阵 R 按行求和不能超过 2。

2. 每种类型的专家必须存在:

$$\sum_{j=1}^{P} R_{j,k} \geq 1, \quad \forall k = 1, 2, \ldots, E$$

其中 $R_{j,k}$ 表示专家 k 是否被放置在 GPU j 上，其取值为 0 或 1。每种专家 k 必须至少被一个 GPU 放置，即 $\sum_{j=1}^{P} R_{j,k} \geq 1$，应该对所有的专家 k 进行约束。

3. 数据发送比例的和为 1:

$$\sum_{k} W_{i,j,k} = 1$$

其中 $W_{i,j,k}$ 表示从 GPU i 发送到 GPU j 的数据比例中，专家 k 所占的比例。

4. 当不存在专家时，数据发送比例必须为 0:

$$R_{j,k} = 0 \Rightarrow W_{i,j,k} = 0$$

其中 $R_{j,k}$ 表示专家 k 放置在 GPU j 上，$W_{i,j,k}$ 表示从 GPU i 发送到 GPU j 的数据比例中，专家 k 所占的比例。

(4) 优化问题目标函数

$$\text{MIN}\left(\sum_{i}\sum_{j}\left(a_{i,j}+\frac{V_{i,j}}{b_{i,j}}\right)\right)$$

其中 MIN 表示对所有可能的解求最小值，$a_{i,j}$ 表示建立从 GPU i 到 GPU j 的网络连接需要的时间，$b_{i,j}$ 表示从 GPU i 到 GPU j 的网络传输速率的倒数，$V_{i,j}$ 表示从 GPU i 发送到 GPU j 的数据量，其计算如下：

$$V_{i,j}=\sum_{k=1}^{E}R_{j,k}\cdot W_{i,j,k}\cdot C_{i,k}\cdot(1-R_{i,k})$$

$R_{j,k}$ 表示专家 k 是否被放置在 GPU j 上，$W_{i,j,k}$ 表示从 GPU i 发送到 GPU j 的数据比例中，专家 k 所占的比例，$C_{i,k}$ 表示从 GPU i 发送到专家 k 的数据量。该式表示在数据从 GPU i 发送到 GPU j 的过程中，需要发送所有的数据量。

4.2.2 系统实现

我们在已有的 MoE 训练框架上进行了相关改进，我们主要采用了 Deepspeed-MoE 训练框架。DeepSpeed-MoE[38] 是一种基于 DeepSpeed 分布式深度学习框架的 Mixture of Experts (MoE) 模型实现，它为用户提供了一个可扩展且高性能的 MoE 训练框架。通过在 Deepspeed 的源码上做出修改，我们可以实现高性能和高效的分布式训练解决方案。具体而言，我们在 Deepspeed 框架的基础上实现了负载均衡算法。对原本的 DeepSpeed-MoE 分布式深度学习框架训练过程中对系统的负载进行优化和均衡。Deepspeed 也是我们训练系统在性能比较时的一个 baseline。

1. 专家并行初始化：我们引入了专家-数据并行组（expert-data-parallel-group）的概念。这个组的建立旨在实现对专家和数据的并行处理。通过这一修改，我们能够同时在不同设备上进行专家和数据的计算，从而提高训练速度和效率。

2. 训练阶段 1：我们移除了可以在本地计算的 All-to-All 参数，并将其替换为全零。这个调整旨在减少通信开销，避免不必要的数据传输。在这一步中我们主要完成每个节点内部的 GPU 之间相互交换数据。

3. 训练阶段 2：在第二轮训练开始时，我们进行本地计算。这意味着一部分计算可以在每个设备上独立完成，并进行全局的 GPU 数据交换，这意味着，我们可以将计算过程与通信重叠，并进一步提高训练的速度和效率。

4. 训练阶段 3：在每轮训练的结束时，使用 reduce 原语计算负载跟踪（load trace）。这一修改有助于监控每个设备的负载情况，从而了解训练过程中的负载分布。

5. 训在训练结束时，我们将 MoE 参数通过 All-Reduce 操作同步。这一步骤确保了 MoE 参数的正确同步，以获得准确的模型结果。通过正确的同步，我们可以确保模型的一致性和准确性。

6. 发现负载出现变化：在负载发送变化之后，我们引入了 Gurobi 优化算法，用于计算负载均衡策略。这一修改允许我们在特定轮次后通过优化算法调整专家的放置位置，以进一步优化模型的性能。通过优化放置策略，我们可以更好地利用集群资源，提高训练效率和模型性能。

7. 更新专家放置以适应当前负载：在达到指定步骤后，我们进行专家参数的交换。这一修改涉及在不同设备之间传输和同步专家参数，以更好地利用各设备的计算资源。通过根据当前负载情况调整专家的放置位置，我们可以实现负载的动态均衡，进一步提高训练的效率和性能。

8. 重复迭代直到模型收敛。我们的负载均衡算法系统会在模型收敛之前反复执行上述步骤，以确保模型能够充分训练和收敛。通过迭代的过程，我们能够不断优化负载均衡策略，提高系统的整体性能和训练效果。

在训练的过程中，我们还设计了一个 scheduler 用于监控并收集专家的负载情况，该调度器是负载均衡算法系统中的关键组件，通过定期收集和分析专家的负载信息，能够及时地检测到系统中的负载变化。它会综合考虑整个集群中各个专家的负载状态，识别出负载较重或负载较轻的专家，并根据负载不均衡的程度进行优化调度。

通过以上一系列的定制修改，我们对 DeepSpeed 框架进行了个性化的定制，以满足特定需求。这些调整的目标在于提高训练速度、优化资源利用、调整专家的放置策略，并确保模型参数的正确同步。这些改进对于 MoE 模型的训练过程具有重要意义，有望提升性能和效果。

4.3 本章小结

总结起来，通过我们对 DeepSpeed 框架的修改和补充，以及对 MoE 训练过程的优化，我们能够实现负载均衡算法系统，监控和调度专家的负载情况，从而提高训练效率和模型性能。这些改进措施包括并行化处理、减少通信开销、本地计算、负载跟踪、参数同步、放置策略优化和专家参数交换等。通过持续迭代和优化，我们能够达到更好的负载均衡效果，并在训练过程中实现模型的收敛。这些改进为深度学习任务的分布式训练提供了更好的性能和效果，为研究者和工程师提供了有力的工具和方法。

第五章　系统测试与分析

5.1 系统测试环境

我们的系统基于 Microsoft 开源的 Deepspeed 框架 [8] 进行开发，主要使用 Python 语言。我们使用了一个 8 层的 Transformer-xl 模型 [39]，并在其基础上拓展成为 MoE 架构。在数据集的选择上，我们主要使用了 enwik8 数据集 [46] 进行测试。

我们使用了私有集群进行测试，其中包括两种不同的 GPU 集群配置，用于验证基于网络拓扑的自动负载均衡策略设计。我们主要使用了 2 节点 8 卡的 Nvidia Titan XP GPU 集群以及 4 节点 8 卡的 Nvidia Titan XP GPU 集群。集群节点之间的网络通信通过 Ethernet，其带宽约为 10Gb/s。

5.2 系统性能指标

我们主要关注两个性能指标进行测试。第一个指标是模型的收敛速度，即 loss 收敛速度。第二个指标是每个 step 的完成时间，即模型训练过程中每个 batch 所需要的 forward + backward + gradient synchronization 所需要的总时间。我们将对这两个指标进行详细的测试评估，并根据测试结果进行优化和调整。

5.3 实验结果

5.3.1 动态路由的数据分派策略实验结果

我们进行了动态路由的数据分派策略实验，并将我们的设计与传统的 Top-1 和 Top-2 Gating 策略进行了对比。实验的对象是 Transformer-XL 模型，我们关注了每轮训练中损失的变化情况，并将其可视化为图表。

对比传统的 Top-1 和 Top-2 Gating 策略，我们的设计采用了动态路由的数据分派策略。通过实验结果的对比分析，我们发现我们的设计在损失的降低速度和最终的收敛效果上表现出了显著的优势。图5-1清晰地展示了每轮训练中损失的变化趋势，我们的设计在较早的轮次就取得了更快的收敛速度，并且在后续的训练过程中持续保持较低的损失值。

这些实验结果表明，我们的动态路由数据分派策略在 Transformer-XL 模

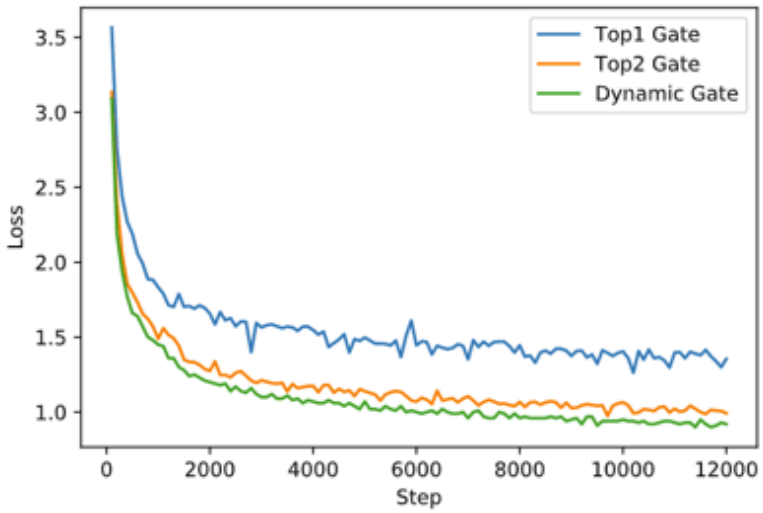

图 5-1　训练 Loss 变化图

型上的有效性和优越性。相比传统的 Top-1 和 Top-2 Gating 策略，我们的设计能够更好地调度和分配数据，使得模型能够更快地收敛并获得更好的训练效果。这为提升模型性能和训练效率提供了有力的实证支持。

综上所述，我们的动态路由数据分派策略在 Transformer-XL 模型上的实验结果显示出了明显的优势，为改进模型训练和优化数据分派提供了一种有效的方法。这些发现对于深度学习研究和实践具有重要意义，并为未来的工作和改进方向提供了有益的启示。

5.3.2 基于网络拓扑的自动负载均衡实验结果

(1) 端到端训练性能分析

我们对基于网络拓扑的自动负载均衡设计在 GPU 集群上进行了测试，我们使用了 2 个节点，每个节点配备了 8 个 Nvidia TitanX GPU。网络带宽为 10Gbps，训练框架采用了 Deepspeed，MoE 模型为 Transformer-xl MoE，数据集为 enwik8，模型参数量为 45M。

根据我们的实验数据5-2，我们的设计在前向传播阶段获得了约 4.09 倍的加速比。这表明我们基于网络拓扑的自动负载均衡设计在模型的前向计算过程中取得了显著的性能提升。通过有效地分配计算任务和利用 GPU 集群的并行能力，我们能够加快前向传播的速度，从而加速整体训练过程。

在后向传播阶段，我们的设计获得了约 3.08 倍的加速比。这进一步证明了我们的自动负载均衡设计在处理梯度计算和参数更新时的优越性。通过合

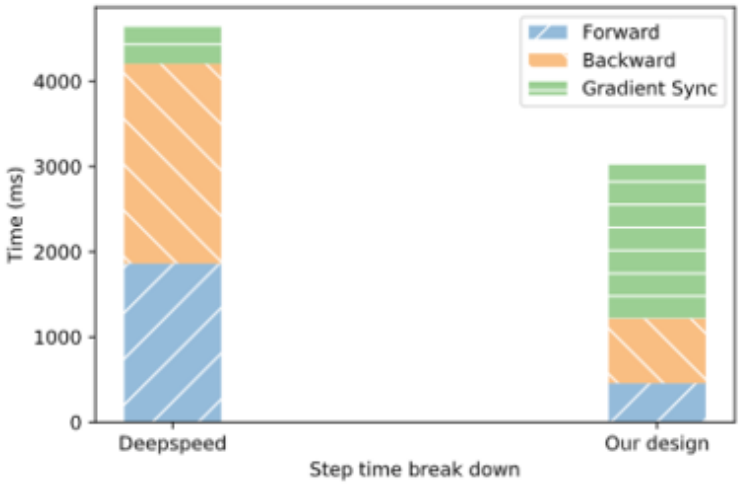

图 5-2　参数量为 45M 时，每个 step 时间步分解

表 5-1　在参数量为 45M 的 Trasnformer-xl MoE 模型训练中，我们的设计与 baseline(Deepspeed) 的 Step Time 的拆分

Stages	Baseline	Our design	**Acc**
Fwd_MoE (ms)	1860	455	**4.09x**
Bwd_Moe (ms)	2347	761	**3.08x**
Gradient Sync (ms)	430	1805	**0.24x**
Overall (ms)	5200	3576	**1.45x**

理地分配计算任务和优化通信流程，我们能够有效地利用 GPU 集群的计算资源，提高后向传播的效率。

然而，在梯度同步阶段，由于更负载的并行策略和更复杂的通信流程，我们的设计可能会遇到一定程度的性能下降。这是因为梯度同步过程中的通信开销增加，可能导致整体训练速度的一定减缓。

综合而言，基于我们的实验结果，我们的基于网络拓扑的自动负载均衡设计在前向传播和后向传播阶段分别获得了约 4.09 倍和 3.08 倍的加速比。虽然在梯度同步阶段可能存在一定的性能下降，但最终整体端到端的训练速度仍然提升了约 1.24 倍。这证明了我们设计的有效性和优越性，能够加速训练过程并提高模型训练的效率。在未来的工作中，我们将继续优化和改进设计，以进一步提高整体训练速度并减少梯度同步阶段的性能下降。

8 GPUs 2 Nodes	model: transformer-xl		layer:8			batch_size: 24					
#Params	Fwd_MoE(ms)			Bwd_inner(ms)			Bwd_Allreduce (ms)			Overall (ms)		
	Baseline	Our design	Acc	Baseline	Our design	Acc	Baseline	Our design	Acc	Baseline	Our design	Acc
45M	1860	455	**4.09x**	2347	761	**3.08x**	430	1805	**0.24x**	5200	3576	**1.45x**
94M	1929	463	**4.17x**	2386	815	**2.93x**	430	5350	**0.08x**	5161	7005	**0.74x**

表 5-2　更改模型参数，分析系统训练性能

8 GPUs 2 Nodes	**model: transformer-xl**			**layer:8**			**batch_size: 40**					
#Params	Fwd_MoE (ms)			Bwd_inner (ms)			Bwd_Allreduce (ms)			Overall (ms)		
	Baseline	Our design	Acc	Baseline	Our design	Acc	Baseline	Our design	Acc	Baseline	Our design	Acc
45M	3233	527	**6.13x**	4171	1146	**3.64x**	430	1813	**0.24x**	8365	4674	**1.79x**
94M	3276	612	**5.35x**	3945	1250	**3.16x**	430	5350	**0.08x**	8168	7590	**1.08x**
145M	3212	2145	**1.5x**	3935	2258	**1.74x**	480	3150	**0.15x**	8000	8430	**0.95x**

表 5-3　更改训练 batchsize，分析系统训练性能

8 GPUs 4 Nodes	**model: transformer-xl**		**layer:8**		**batch_size: 40**		**#Params: 94M**	
#Replica	Fwd_MoE		Bwd_inner		Bwd_Allreduce (ms)		Overall (ms)	
	time (ms)	Acc	time (ms)	Acc	time (ms)	Acc	time (ms)	Acc
Baseline (0 replica)	4251	**1x**	3935	**1x**	911	**1x**	10656	**1.0x**
1 replica	3221	**1.32x**	3416	**1.15x**	5851	**0.16x**	12501	**0.85x**
2 replica	1928	**2.2x**	2235	**1.76x**	9278	**0.1x**	14737	**0.72x**

表 5-4　使用 4 节点 8GPU，分析系统训练性能

(2) **自动负载均衡策略深入分析**

除了之前的实验结果，我们对基于网络拓扑的自动负载均衡设计进行了进一步分析。我们通过改变模型参数（如表 6-1 所示）和训练批量大小（如表5-2所示），并调整训练系统的 GPU 集群拓扑（如表5-3所示），来评估设计在不同条件下的性能表现。实验结果与之前的发现相一致，并得出了一些重要的结论。

首先，我们观察到在前向传播和后向传播阶段，我们的系统能够显著提升模型性能，获得了可观的加速比。这证实了基于网络拓扑的自动负载均衡设计在处理不同模型参数和训练批量大小时的有效性和鲁棒性。该设计能够充分利用 GPU 集群的计算资源，实现高效的计算过程。

然而，我们也观察到在梯度同步阶段，由于通信模式的复杂性增加，系统性能有所下降。这是由于梯度同步阶段涉及大量的通信和同步操作，而这些操作可能受限于网络带宽和通信延迟。为了进一步提升系统性能，我们需要考虑额外的优化策略，例如减少通信量、优化通信模式或调整网络拓扑。这将是未来工作的一个关键方向，以实现在梯度同步阶段的高效性能。

综上所述，我们的实验结果与之前的研究结论一致，验证了基于网络拓扑的自动负载均衡设计在前向传播和后向传播阶段的有效性。然而，在梯度同步阶段的性能下降提示我们需要进一步优化设计以提高整体训练效率。这些研究结果为深入探索和改进负载均衡设计提供了重要的参考，并凸显了在大规模深度学习训练中负载均衡的关键作用。在未来的工作中，我们将致力于进一步优化梯度同步阶段的性能，以实现更高效的训练过程。

5.4 本章小节

在动态路由的数据分派方式实验中，我们设计了一种新的数据分派策略，并将其与传统的 Top-1 和 Top-2 Gating 策略进行了对比。我们使用了 Transformer-xl MoE 模型和 enwik8 数据集进行实验。实验结果表明我们设计的动态路由的策略可以以更高的效率实现收敛，同时相比于传统的 Top-1 和 Top-2 Gating 策略减少了通信量，因此是一种有效的方法。

在基于网络拓扑的负载均衡策略实验中，我们提出了一种基于网络拓扑的自动负载均衡方法，并对其性能进行了评估。我们使用了多种 GPU 集群配置，网络带宽为 10Gbps，采用了 Deepspeed 训练框架和 Transformer-xl MoE 模型。实验结果显示，在参数量为 45M 的模型上，我们的系统在前向传播和

反向传播阶段，我们的设计实现了约 4.09 倍和 3.08 倍的加速。然而，在梯度同步阶段，由于负载的并行策略和复杂的通信流程，我们的设计性能有所下降。最终，整体端到端的训练速度提高了约 1.24 倍。这些实验结果为我们深入理解基于网络拓扑的负载均衡策略的优缺点提供了重要见解，并为进一步优化和改进提供了指导方向。这些收获和不足将指导我们未来的工作。

第六章　总结与展望

6.1 本文工作总结

本文我们研究了如何设计高效的 MoE 混合专家模型训练系统，主要深入研究了算法和系统层面的优化，在算法上，我们设计了一种基于动态路由的数据分派策略；在系统设计上，我们设计了一种基于网络拓扑的自动负载均衡策略。我们在真实的 GPU 集群中测试了我们的设计，实验结果表明，我们的设计可以加速 MoE 混合专家模型的训练过程，但是在梯度同步阶段仍然存在继续优化的空间。

6.2 未来工作展望

未来的工作将重点关注如何有效解决由于更复杂的通信模式导致的梯度同步阶段性能下降问题，以进一步提高训练 MoE 混合专家模型的效率。为此，我们可以探索以下几个方向来解决这一问题。

首先，我们可以进一步优化梯度同步算法，以减少通信开销并提高性能。可以考虑使用更高效的同步机制，如基于稀疏梯度的同步方法，以减少传输的数据量。此外，可以采用异步梯度聚合技术，允许不同设备的计算和通信重叠，从而加速梯度同步过程。

此外，我们还可以考虑引入硬件加速技术来加速梯度同步过程。例如，利用高速网络和专用加速器（如 GPU-Direct RDMA）进行高效的设备间通信，以降低通信延迟和带宽瓶颈。

最后，我们应该继续进行实验和性能分析，以评估提出的解决方案的有效性和可扩展性。通过细致的实验设计和性能测量，可以全面了解各种因素对系统性能的影响，并对改进策略进行验证和优化。

参考文献

[1] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[2] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33:1877-1901.

[3] Nvidia a100 gpu[EB/OL]. https://www.nvidia.com/en-us/data-center/a100/.

[4] Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models[J]. arXiv preprint arXiv:2001.08361, 2020.

[5] Sharir O, Peleg B, Shoham Y. The cost of training nlp models: A concise overview[J]. arXiv preprint arXiv:2004.08900, 2020.

[6] Lepikhin D, Lee H, Xu Y, et al. Gshard: Scaling giant models with conditional computation and automatic sharding[J]. arXiv preprint arXiv:2006.16668, 2020.

[7] Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library[J]. Advances in neural information processing systems, 2019, 32.

[8] Deepspeed[EB/OL]. https://www.deepspeed.ai/.

[9] He J, Qiu J, Zeng A, et al. Fastmoe: A fast mixture-of-expert training system[J]. arXiv preprint arXiv:2103.13262, 2021.

[10] Ben-Nun T, Hoefler T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis[J]. ACM Computing Surveys (CSUR), 2019, 52(4):1-43.

[11] Sergeev A, Del Balso M. Horovod: fast and easy distributed deep learning in tensorflow[J]. arXiv preprint arXiv:1802.05799, 2018.

[12] Li S, Zhao Y, Varma R, et al. Pytorch distributed: Experiences on accelerating data parallel training[J]. arXiv preprint arXiv:2006.15704, 2020.

[13] Peng Y, Zhu Y, Chen Y, et al. A generic communication scheduler for distributed dnn training acceleration[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019: 16-29.

[14] Shazeer N, Cheng Y, Parmar N, et al. Mesh-tensorflow: Deep learning for supercomputers[J]. Advances in neural information processing systems, 2018, 31.

[15] Nccl[EB/OL]. https://github.com/NVIDIA/nccl.

[16] Huang Y, Cheng Y, Bapna A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism[J]. Advances in neural information processing systems, 2019, 32.

[17] Zheng L, Li Z, Zhang H, et al. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning[C]//16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022: 559-578.

[18] Ouyang S, Dong D, Xu Y, et al. Communication optimization strategies for distributed deep neural network training: A survey[J]. J. Parallel Distributed Comput., 2021, 149:52-65.

[19] You Y, Zhang Z, Hsieh C J, et al. Fast deep neural network training on distributed systems and cloud tpus[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(11):2449-2462.

[20] Stich S U. Local sgd converges fast and communicates little[C]//ICLR 2019-International Conference on Learning Representations: CONF. 2019.

[21] Cheng Y, Wang D, Zhou P, et al. A survey of model compression and acceleration for deep neural networks[J]. arXiv preprint arXiv:1710.09282, 2017.

[22] Zhou Y, Moosavi-Dezfooli S M, Cheung N M, et al. Adaptive quantization for deep neural network[C]//Proceedings of the AAAI Conference on Artificial Intelligence: volume 32. 2018.

[23] Jacob B, Kligys S, Chen B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 2704-2713.

[24] Pham H, Guan M, Zoph B, et al. Efficient neural architecture search via parameters sharing [C]//International conference on machine learning. PMLR, 2018: 4095-4104.

[25] Liu Z, Sun M, Zhou T, et al. Rethinking the value of network pruning[J]. arXiv preprint arXiv:1810.05270, 2018.

[26] Hu E J, Shen Y, Wallis P, et al. Lora: Low-rank adaptation of large language models[J]. arXiv preprint arXiv:2106.09685, 2021.

[27] Nvidia gpu direct rdma[EB/OL]. https://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[28] Fei J, Ho C Y, Sahu A N, et al. Efficient sparse collective communication and its application to accelerate distributed deep learning[C]//Proceedings of the 2021 ACM SIGCOMM 2021 Conference. 2021: 676-691.

[29] Daily J, Vishnu A, Siegel C, et al. Gossipgrad: Scalable deep learning using gossip communication based asynchronous gradient descent[J]. arXiv preprint arXiv:1803.05880, 2018.

[30] Mai L, Hong C, Costa P. Optimizing network performance in distributed machine learning. [C]//HotCloud. 2015.

[31] Chowdhury M, Zhong Y, Stoica I. Efficient coflow scheduling with varys[C]//Proceedings of the 2014 ACM conference on SIGCOMM. 2014: 443-454.

[32] Jayarajan A, Wei J, Gibson G, et al. Priority-based parameter propagation for distributed dnn training[J]. Proceedings of Machine Learning and Systems, 2019, 1:132-145.

[33] Hashemi S H, Abdu Jyothi S, Campbell R. Tictac: Accelerating distributed deep learning with communication scheduling[J]. Proceedings of Machine Learning and Systems, 2019, 1: 418-430.

[34] Mahajan K, Chu C H, Sridharan S, et al. Better together: Jointly optimizing {ML} collective scheduling and execution planning using {SYNDICATE}[C]//20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023: 809-824.

[35] Rajbhandari S, Rasley J, Ruwase O, et al. Zero: Memory optimizations toward training trillion parameter models[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020: 1-16.

[36] Ren J, Rajbhandari S, Aminabadi R Y, et al. Zero-offload: Democratizing billion-scale model training.[C]//USENIX Annual Technical Conference. 2021: 551-564.

[37] Shoeybi M, Patwary M, Puri R, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism[J]. arXiv preprint arXiv:1909.08053, 2019.

[38] Rajbhandari S, Li C, Yao Z, et al. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale[C]//International Conference on Machine Learning. PMLR, 2022: 18332-18346.

[39] Dai Z, Yang Z, Yang Y, et al. Transformer-xl: Attentive language models beyond a fixed-length context[J]. arXiv preprint arXiv:1901.02860, 2019.

[40] Ott M, Edunov S, Baevski A, et al. fairseq: A fast, extensible toolkit for sequence modeling [J]. arXiv preprint arXiv:1904.01038, 2019.

[41] Hwang C, Cui W, Xiong Y, et al. Tutel: Adaptive mixture-of-experts at scale[J]. arXiv preprint arXiv:2206.03382, 2022.

[42] Xu C, McAuley J. A survey on dynamic neural networks for natural language processing[J]. arXiv preprint arXiv:2202.07101, 2022.

[43] Roller S, Sukhbaatar S, Weston J, et al. Hash layers for large sparse models[J]. Advances in Neural Information Processing Systems, 2021, 34:17555-17566.

[44] Chen C, Li M, Wu Z, et al. Ta-moe: Topology-aware large scale mixture-of-expert training[J]. Advances in Neural Information Processing Systems, 2022, 35:22173-22186.

[45] Li J, Jiang Y, Zhu Y, et al. Accelerating distributed moe training and inference with lina[C]// Proceedings of the USENIX Annual Technical Conference. 2023.

[46] Enwik8 dataset[EB/OL]. https://huggingface.co/datasets/enwik8.

致　谢

在完成本篇论文的过程中，我要首先感谢学校组织的海外毕设项目，让我有幸前往香港中文大学，加入徐宏教授的研究组，参与了为期 7 个月的海外毕设项目。这个宝贵的机会让我能够在国际化的研究环境中学习和成长，拓宽了我的学术视野。

此外，我还要感谢计算机学院对我的支持和培养。特别要感谢学院设置的康继昌智能系统班，作为第一届康班的成员，我有幸结识了很多优秀的同学，我们一起努力学习和进步。在康班的学习生活中，我找到了自己真正热爱的系统网络方向，并在老师的指导下开展了一系列具有挑战性的竞赛和科研项目，收获了宝贵的经验和成长。

其次，我要衷心感谢我的毕设导师崔禾磊教授和陈亚兴教授，以及外校指导老师徐宏教授。他们在整个研究过程中给予了我专业的指导和深入的思路启发。他们的悉心指导使我能够深入理解研究领域的核心问题，并提供了宝贵的建议和意见，使我能够顺利地完成研究工作。

此外，我要感谢我的朋友们对我无私的支持和帮助。特别要感谢我的好朋友张博日常与我讨论，让我有了清晰的研究目标。还有我的舍友张瀚文、张利军和戴柯迪，我们一起度过了在云天苑 306A 的宝贵时光。他们在整个研究过程中给予了我宝贵的建议和鼓励。我们进行了许多深入的讨论和思考，共同推动了我的研究工作的进展。感谢当时龙芯杯的队友魏天昊，江嘉熙，申世东，与你们一起参加龙芯杯，一起通宵的经历是我大学以来最难忘的回忆。还有康班的其他同学们，向你们学到了很多宝贵的技术知识。

我还要特别感谢我的师兄谭昕和师姐李嘉敏。他们在我的研究方向上提供了许多有益的讨论和指导，帮助我克服了许多困难和挑战。他们的经验和见解对我的研究工作有着重要的影响，使我能够更加深入地探索问题并取得进展。

最后，我要衷心感谢所有帮助和指导过我的人。没有你们的支持和鼓励，我无法完成这篇论文。你们的知识和经验对我产生了深远的影响，激励着我不断学习和进步。

再次向所有帮助过我的人表示衷心的感谢！我将倍加珍惜这段宝贵的学术经历，并将继续努力追求知识的深度和广度，发扬康继昌先生的奉献精神，为解决卡脖子问题做出自己的贡献。

毕业设计小结

毕业论文是大学四年的最后一份相对完整的科研工作，通过完成本科毕设，我掌握了科研的初步要领，并且也成功完成了毕业论文，这标志着我的本科生涯正式画上了一个句号。

回顾大学四年的生活，我意识到学习一直占据了我主要的精力。常常感受到学业压力和内卷竞争的困扰。然而，我也明白在后续的求学生涯中，我需要找到学习与生活的平衡点。我希望在未来的学习和研究中，能够更好地处理学业和生活的关系，拥有更充实而有意义的大学生活。

同时，我决心坚定地在系统与网络的研究道路上继续前行。我深深地热爱这个领域，它充满了挑战和机遇。我会继续努力学习和钻研，不断提升自己的专业知识和研究能力。我希望在未来的研究中能够做出更多有意义的贡献，为推动系统与网络领域的发展做出自己的努力。

对于未来的道路，我充满期待和信心。我相信通过持续不断的努力和奋斗，我将能够迎接更多的机遇和成就更大的突破。我将牢记学术研究的初心，继续追求知识的深度和广度，为建设科技强国贡献自己的力量。

本科期间研究成果产出

在投论文

- **[IEEE TDSC]**: Fighting Fake News Spread in Online Social Networks: A Privacy-Preserving Approach (In submission)

专利

- 一种隐私保护的社交媒体假消息检测方法 (已受理)
  发明人：崔禾磊，杨益滔，丁亚三，邱晨，郭斌，於志文
  申请号：202210615749.1

竞赛获奖

- 第五届全国大学生系统能力大赛 CPU 设计赛道（龙芯杯）
  团队赛全国一等奖

附　录

源代码参见：https://github.com/yyyyyt123/dynamic-moe-deepspeed