计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

分布式机器学习系统网络性能优化研究进展

王帅[1),2)] 李丹[1),2)]

[1)](清华大学计算机科学与技术系 北京 100084)

[2)](北京信息科学与技术国家研究中心 北京 100084)

摘 要 以机器学习为代表的人工智能技术需要对海量数据进行处理，对底层算力要求极高。分布式机器学习通过将计算任务分布式地部署到多个计算节点来加快模型的训练速度，从而将训练任务完成时间降低到可接受范围。由于通信开销对分布式机器学习系统的扩展性具有重要影响，因此，分布式机器学习系统网络性能优化受到各界研究者的广泛关注。本文首先分析了分布式机器学习系统扩展性不足的主要原因，并提出了改善其扩展性的关键思路，然后系统地综述了分布式机器学习系统网络性能优化相关的研究工作，并对这些研究工作从多个角度进行了对比分析。最后，对分布式机器学习系统网络性能优化研究的未来发展趋势进行了展望。

关键词 分布式机器学习系统；网络优化；参数同步；通信调度；网内聚合

中图法分类号 TP18

Research Progress on Network Performance Optimization of Distributed Machine Learning System

WANG Shuai[1),2)] LI Dan[1),2)]

[1)](Department of Computer Science and Technology, Tsinghua University, Beijing 100084)

[2)](Beijing National Research Center for Information Science and Technology, Beijing 100084)

Abstract Artificial intelligence technology, represented by machine learning, enables computers to learn knowledge from massive amounts of data, which costs extremely high computing power by iteratively correcting the learned model. However, the computing power required by today's machine learning model far exceeds the capacity of a single computing hardware, which makes distributed machine learning become the most promising way to accelerates the training speed of the model. Distributed machine learning distributes the computing task to lots of computing nodes, so ideally, it can linearly reduce the task completion time. Unfortunately, the overhead of network communication among computing nodes has an important impact on the scalability of distributed machine learning systems in practice. Therefore, optimizing the network performance of distributed machine learning systems has drawn extensive attention from researchers. This paper starts with an analysis about why the network communication becomes the bottleneck in the scalability of distributed machine learning systems: (1) machine learning models are becoming more and more complicated, and the number of model parameters continues to increase; (2) more and more computing power is required to train the complicated models,

本课题得到重点研发计划项目(2018YFB1800500)、广东省重点领域研发计划项目(2018B010113001)、国家自然科学基金项目(61772305)、清华大学-中国移动联合研究院项目资助. **王 帅**，博士研究生，主要研究领域为数据中心网络、分布式机器学习系统网络性能优化. E-mail: s-wang17@mails.tsinghua.edu.cn. **李 丹**（**通信作者**），博士，教授，博士生导师，主要研究领域为互联网体系架构、数据中心网络、数据驱动网络. E-mail: tolidan@tsinghua.edu.cn.

leading to an increase in the scale of distributed systems; (3) the performance improvement of GPU and other computing hardware are leaving network devices lagging behind. This paper then proposes three fundamental ways to reduce the impact of network communication on the scalability of distributed machine learning systems: (1) reducing communication demand, either in communication frequency or data volume in each communication; (2) improving communication capability, such as high-performance transport technologies and high bisection-bandwidth network topologies; (3) improving communication efficiency. Next, this paper systematically introduces the related research work based on the key ideas of the above-mentioned fundamental methods adopted by them, and compares the similar work that adopt the same fundamental ways to demonstrate their improvements and limitations. Besides, this paper also provides a comprehensive comparison and analysis on the existing work to show their characteristics of optimization methods, improvement effects, effectiveness at large scale, impact on convergence, and requirements for underlying network hardware. In the end, this paper looks forward to the future development trend of the research on network performance optimization of distributed machine learning systems, and proposes four dominant and promising directions on this research field: (1) high-quality compression of models. Today, reducing the volume of the data that is required to be synchronize can significantly accelerate the training speed of distributed machine learning, but at the cost of slowing down the convergence speed of the trained models, due to the limitations of the existing compression technologies. (2) optimization in parallelism manners. In practice, the performance of the newly-proposed parallelism solutions still cannot fully utilize the computing resources, due to the coarse-grained parallelism elements, such as a training sample or a complete model layer. (3) improving the efficiency in multiplexing network resources in multi-job scenarios. The existing network performance optimization schemes are still mainly designed for single-job scenarios, and joint optimization schemes between multiple jobs remains to be studied in depth. (4) designing dedicated network devices and architectures. Designing dedicated network devices and architectures for distributed machine learning, such as switches with ultra-low forwarding delay, integration of GPU and network interface card, will also become a research hotspot in the future.

Key words distributed machine learning system; network optimization; parameter synchronization; communication scheduling; in-network aggregation

1 引言

近年来，以机器学习，尤其是深度学习，为代表的人工智能技术在图像识别[1]、语音识别[2]、机器翻译[3]和自动驾驶[4]等应用领域都取得了突破性进展。其原因可以分为三个方面：首先，互联网、大数据技术的发展，积累了海量的训练数据，在这些数据中蕴含着丰富的信息；其次，机器学习理论的发展使得机器学习算法和模型不断完善，为从训练数据中挖掘有价值的信息创造可能；最后，GPU 等加速器以及云计算等技术大幅提升计算性能，大大加快了从数据中获取信息的速度，使得机器学习技术的应用成为现实。总之，作为第三次人工智能浪潮的“催化剂”，算力的大幅提升直接将人工智能再次推向新的繁荣期。

高性能的机器学习算法往往具有更高的计算需求。据 OpenAI 统计，人工智能训练所需要的算力呈指数级增长，每 3.5 个月翻一倍①。相比之下，近年来计算引擎的发展速度则远远落后于模型计算需求的增长。以 Nvidia GPU 发展为例，表 1 展示了 2012 年以来 Nvidia 的多代 GPU 在训练 ResNet 模型时的性能表现。可以看到近 8 年来，GPU 的计算性能只提高了 16 倍左右，远低于同期模型计算需求的增长。在“后摩尔定律”时代，单个计算引擎的性能提升逐渐进入了瓶颈期。面对日益复杂的计算任务，分布式机器学习被认为是必然的发展趋势，逐渐成为业界的研究热点[5]。

① AI and Compute. https://openai.com/blog/ai-and-compute/ 2018,05,16

表 1 单 GPU 训练 ResNet-269 模型的性能

GPU 型号	发布时间	训练速度（图片/秒）
K520	2012 年	4
K80	2014 年	11
M60	2015 年	14
1080Ti	2017 年	44
V100	2019 年	65

在分布式机器学习训练任务的迭代计算过程中，不同计算节点间需要频繁同步机器学习模型参数，以使得该模型能够遍历完整的数据集，从而保证最终得到的模型与使用单机训练的模型一致。然而，随着计算节点数量的增多，一方面，不同节点间进行参数同步的流量逐渐增加；另一方面，为了避免单轮训练过多数据（即批尺寸过大）带来的模型泛化能力下降问题[6]，每个节点所分配的计算任务会逐渐减少。因此，对于分布式机器学习系统，通信开销和计算开销的比值会随着系统规模的增大而呈现幂增长趋势。这导致通信成为限制大规模分布式机器学习系统扩展效率的主要瓶颈，甚至出现随着节点数量增加，模型训练速度反而下降的情况[7-11]。并且，过多的通信时间会导致 GPU 等昂贵的计算设备大部分时间处于等待参数同步的状态，造成计算资源的浪费。因此，研究如何对分布式机器学习系统的网络性能进行优化，降低通信操作对分布式机器学习系统扩展效率的影响，从而提高机器学习模型的训练速度，具有重要的研究意义和实用价值。

本文将首先介绍分布式机器学习系统的通信特点，并分析网络通信成为分布式机器学习系统扩展性瓶颈的原因，然后提出三种优化网络性能的关键思路，并以这些思路为指导，从通信模式、通信数据量、通信效率以及网络拓扑等方面具体地介绍分布式机器学习系统网络性能优化研究的最新进展，并从加速效果、优化机制、扩展性、对模型收敛性的影响以及是否需要升级硬件设备或更新互联方式等多个角度对这些研究工作进行对比分析，最后讨论分布式机器学习系统中网络性能优化研究的未来发展趋势。

2 背景介绍

2.1 机器学习概述

机器学习是一门交叉学科，涉及概率统计、决策论、计算机科学等多个学科。概括地说，机器学习研究的是如何使计算机模拟人类的学习行为：从历史经验（即，数据）中总结出规律（即，模型），并将该规律应用到新的场景中。从数据中学习到模型的过程称作训练，在新的场景中使用学习到的模型进行预测的过程称作推理。通常，为了提高机器学习模型的质量，需要以海量数据作为模型训练的输入，并且需要反复的学习。相比之下，推理过程则只需要将新场景作为输入，机器学习模型便会输出推理结果。由此可见，训练过程比推理过程更加耗时。

作为机器学习模型的主要代表，人工神经网络是一种模拟生物神经网络的人工智能架构，由大量的人工神经元相互连接而构成。人工神经元是一个接收输入信号并转换为输出信号的基本单元。如图 1 所示，x_i 表示输入信号，由训练数据或其他神经元 i 产生，w_{ij} 表示神经元 i 和神经元 j 之间的连接权重，b_j 表示神经元偏置，y_j 表示输出信号。图 1 对应的数学表达为

$$y_j = \sigma\left(\sum_{i=1}^{n} x_i w_{ij} + b_j\right) \tag{1}$$

其中，$\sigma(\cdot)$ 为非线性激活函数。神经网络中最重要的是神经元之间的连接权重和结构。神经网络的训练过程便是不断地调整权重，使其趋于最佳的过程，因此权重也被称作参数。不同于权重，神经网络结构一般需要人为设计，并且在训练过程中保持不变。神经网络通常有多层神经元组成。图 2 是一个 4 层神经网络示例，包含 1 个输入层、2 个隐含层和 1 个输出层。在实际应用中，神经网络的深度正在不断增加。例如，ResNet 模型[12]的深度高达 152 层。

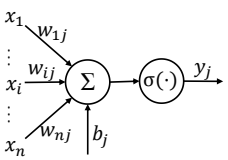


图 1 神经元模型

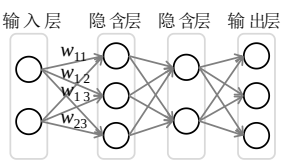


图 2 多层神经网络

模型在训练过程中需要使用优化器对权重进行优化。常用的优化器主要包括随机梯度下降（Stochastic Gradient Descent，SGD）①和标准动

① 现在 SGD 一般指小批量梯度下降（Mini-Batch Gradient Descent， MBGD）。

量优化（Momentum）等。下面以 SGD 优化器为例介绍模型参数更新过程。模型训练分为很多轮，并且每轮只以一小批样本作为输入，这样可以降低参数更新时的方差，使得收敛更稳定。假设第 t 轮训练时模型参数为 θ^t，模型对应的函数为 $J(\theta^t;)$，单轮训练输入的样本数量，即批尺寸（batch size），为 n，第 i 个样本为 x_i，则 SGD 的更新公式如下：

$$\theta^{t+1}=\theta^t-\eta\frac{1}{n}\sum_{i=1}^{n}\nabla J\left(\theta^t;x_i\right) \tag{2}$$

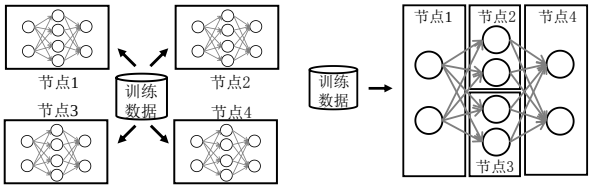
其中，η 表示学习率，$\nabla J(\,)$ 表示新样本对模型参数的影响，即梯度。随着训练的进行，参数会朝着使模型误差（即损失）减小的方向不断更新。

机器学习模型在每轮训练时，需要依次执行两个计算阶段：前向计算（Forward Propagation，FP）和反向计算（Backward Propagation，BP）。前向计算阶段使用当前的模型参数对输入的样本按照从输入层到输出层的顺序逐层计算，得到当前模型对于该样本的预测，然后和预期输出作比较，得到当前模型的损失。反向计算阶段与前向计算阶段方向恰好相反，首先，输出层以损失值为输入，结合前向计算阶段产生的激活值（即各层计算的中间结果），计算出该层参数对应的梯度，然后继续向前传播，直至所有层都计算出梯度。最后，根据公式（2）对模型参数进行更新，从而得到新的模型供下一轮训练使用。

2.2 分布式机器学习

随着信息技术快速发展，全球数据呈现爆发式增长，推动人类社会迈入大数据时代。在大数据时代，机器学习训练任务往往需要对海量的训练数据进行大量的计算，以提高模型的准确度。在单机上执行这样的训练任务，无论是在计算速度还是在数据存储方面都显得十分吃力。例如，使用单块 Nvidia Tesla V100 GPU 训练自然语言处理模型 BERT-large 需要耗时 78 天，这显然是不可接受的。分布式机器学习的目标则是将训练任务分布式地部署到多个计算节点，从而提高模型训练的速度，减少任务耗时。因此，分布式机器学习已经成为机器学习最热门的研究领域之一。

分布式机器学习的并行方式主要包括数据并行（Data Parallelism）和模型并行（Model Parallelism）。如图 3 所示，数据并行是指每个计算节点上均具有同一机器学习模型的副本，但不同计算节点分配到的训练数据是不同的，不同计算节点间需要将各自的模型更新进行同步，以保证机器学习模型的全局一致性。模型并行则是将机器学习模型划分为多个子模型，并分别部署在不同计算节点上，训练数据统一输入，前一节点完成子模型计算后将计算结果传递给后一节点继续对该训练样本进行处理。数据并行由于操作简单，且不同节点的计算负载比较均衡，应用最为广泛。目前，TensorFlow[13]、Pytorch[14]和 MXNet[15]等主流机器学习框架均对数据并行提供了支持，并且具有极好的易用性，但模型并行仍需要用户手动对模型进行划分和分布式部署。



(a) 数据并行　　(b) 模型并行

图 3 典型的分布式机器学习并行方式

假设某分布式机器学习系统共有 m 个计算节点，批尺寸为 n。当采用数据并行时，每个计算节点每轮处理的样本数量为 n/m。各计算节点基于所分配到的训练数据对模型进行更新，然后将不同计算节点更新后的模型进行汇总。汇总后的模型参数如下：

$$\begin{aligned}\theta^{t+1}&=\frac{1}{m}\sum_{w=0}^{m-1}\theta^{t+1,w}\\&=\frac{1}{m}\sum_{w=0}^{m-1}(\theta^t-\eta\frac{1}{n/m}\sum_{i=w\frac{n}{m}+1}^{(w+1)\frac{n}{m}}\nabla J(\theta^t;x_i))\\&=\theta^t-\eta\frac{1}{n}\sum_{i=1}^{n}\nabla J(\theta^t;x_i)\end{aligned} \tag{3}$$

对比公式(2)和公式(3)可知，如果在每一轮训练结束时，将模型参数在所有计算节点间进行同步，则分布式训练时的模型参数变化和单机训练完全相同，即分布式训练可以在不改变模型的收敛性的前提下提高模型的收敛速度。需要说明的是，在实际应用中，一般是对不同计算节点的梯度进行汇总，然后使用汇总后的梯度来更新模型参数，并将新的参数赋给各计算节点的模型副本。

相比于单机训练，分布式训练额外引入了节点之间的数据通信，从而导致分布式训练的速度无法随着计算节点数量的增加而线性提高。衡量

分布式机器学习训练加速效果的指标主要包括加速比（speedup）和扩展效率（scaling efficiency）。加速比是指同一机器学习训练任务在单机训练和分布式训练时所需时间的比值。加速比越大，分布式训练的加速效果越显著，也就是说，可以更快地完成训练任务。需要注意的是，加速比有可能出现小于 1 的情况，此时分布式训练速度反而不及单机训练。扩展效率是指加速比和计算节点数量的比值。扩展效率越高，各计算节点的计算资源利用率也就越高。图 4 展示了利用 Nvidia Tesla V100 GPU 训练 BERT 模型时加速比和扩展效率随 GPU 数量的变化，可以看出加速比和扩展效率的变化趋势并不相同。一般来说，随着节点数量的增多，扩展效率呈下降趋势，而加速比则呈先升后降趋势。加速比在节点数量增多时反而下降，是由于参数同步引入的通信开销抵消了新增节点带来的性能收益。

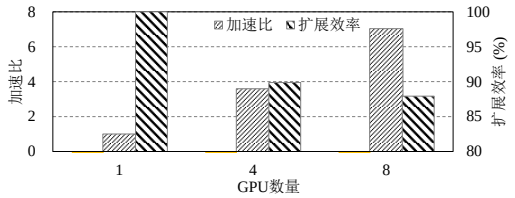


图 4 分布式机器学习系统的扩展性能

具体来说，网络性能之所以会成为分布式机器学习系统扩展性瓶颈的主要原因有以下三点：

1） 机器学习模型越来越复杂，模型参数量不断增加。机器学习算法理论的快速发展催生出各种各样的机器学习模型。例如，OpenAI 最近提出的自然语言处理模型 GPT-3[16]具有 1750 亿参数，而 Krizhevsky 等人在 2012 年提出的 AlexNet[17]模型参数量仅为 0.45 亿。这导致在相同节点规模下，任意两个计算节点间需要同步的参数量随之大幅增加，加剧了分布式机器学习系统中网络通信的压力；

2） 复杂的机器学习模型同时也意味着需要更多的算力，导致分布式系统规模的增大。如前所述，在机器学习模型不变的情况下，分布式系统规模越大，每个节点所承担的计算任务越少，计算耗时越短；与之相反，分布式系统规模越大，每个节点需要通信的对端节点数量越多，通信耗时越长。因此，随着分布式机器学习系统规模的增大，通信开销在整体模型训练开销中的占比越来越高；

3） GPU 等计算设备的性能提升速度快于网络设备的升级。虽然计算设备的性能提升速度远不及模型算力需求的增长，但仍比网络设备的升级速度更快。计算性能和通信性能的差距越来越大，即，计算资源在单位时间内处理的数据需要更长的时间才能被网络资源处理完。这意味着分布式机器学习系统中的网络瓶颈问题将会日益严重。

图 5 分布式机器学习系统网络性能优化工作分类

如图 5 所示，为消除（或缓解）网络性能对分布式机器学习系统扩展性的限制，从本质上来说，有以下三种根本性思路：

1） 降低通信需求。通信需求，即需要通信的数据量和通信次数，对通信耗时具有根本性的影响。显然，数据量越大，或者通信越频繁，通信耗时越长。因此，为了减少通信耗时，可以从机器学习算法层面构建低网络通信需求的训练模型，或采用知识蒸馏和模型剪枝等方式对原模型进行修改以减小模型尺寸。这些方法会造成训练模型的变化，超出了本文的讨论范围，故后文不再作详细描述。参数量化以及参数稀疏化保持训练模型不变，通过降低被传输的参数量来降低通信需求。并行方式的优化通过权衡参数数据量和激活值数据量的大小来切换不同的并行方式：当参数数据量较少时，采用数据并行；反之，采用模型并行。模型一致性协议通过控制参数同步的频率来调节通信需求。参数同步架构对通信次数和每次通信的数据量均会产生影响，通过选取合适的参数同步架构可以有效降低通信需求；

2） 提升通信能力。在通信需求一定时，分布式系统的通信能力越高，通信耗时越短。通信能力的提升主要有两种方式。一种是利用 RDMA、

NVLink 等高性能传输协议实现高带宽、低时延的网络传输，或利用 MLT 等新型机器学习专用传输协议降低丢包对传输性能的影响；另一种是采用高带宽的网络互联拓扑。例如，BCube 和 BiGraph 均采用多网卡服务器架构，不但大幅提高每个计算节点对外通信的能力，甚至可将节点内的通信流量导出到服务器外部，从而绕过 PCIe 瓶颈；

3） 提高通信效率。在通信需求和通信能力均确定的情况下，还可以通过提高通信效率来加速分布式机器学习训练。例如，在 GPU 节点间进行集合通信时，NCCL 通信库由于针对 GPU 设备采取了定制优化，因此具有比传统集合通信库 MPI 更高的性能。网内聚合通过逐跳汇聚参数，增加了单位数据所蕴含的参数信息，从而提高了通信效率。通信调度是在通信需求和通信能力固定的情况下，高效利用网络资源的一种方式，既包括采用小尺寸梯度聚合来降低启动开销的方案，也包括优先传输紧急参数来增加计算和通信重叠程度的方式。

3 参数同步模式优化

在大规模分布式机器学习训练场景中，计算节点间需要频繁地进行参数同步，因此，参数同步模式对整体训练性能具有重要的影响。本节将从模型一致性协议和参数同步架构这两个方面详细介绍对参数同步模式进行优化的相关工作。

3.1 模型一致性协议

在数据并行模式下，每个计算节点都需要保存一份相同的模型副本，然后使用本地的训练数据对模型副本进行更新。因此，在训练过程中，不同计算节点所维护的模型副本会出现差异。为了使得分布式训练能够取得与单机训练相同的效果，需要保证这些模型副本的一致性。接下来，介绍三种典型的模型一致性协议。

整体同步并行（Bulk Synchronous Parallel，BSP）[18]是由著名的计算机科学家 Valiant 于 1990 年提出的，该协议已经被 TensorFlow、PyTorch 和 MXNet 等主流分布式机器学习框架所采用。BSP 的特点是，在所有计算节点完成该轮参数同步前，任何一个计算节点都不会进入下一轮计算过程。具体来说，BSP 协议的同步过程分为以下三个阶段（如图 6(a)所示，阴影部分为通信过程）：

1） 并发计算：各计算节点同时使用自己的模型副本和本地的训练数据进行本地计算，获得本地的模型更新；

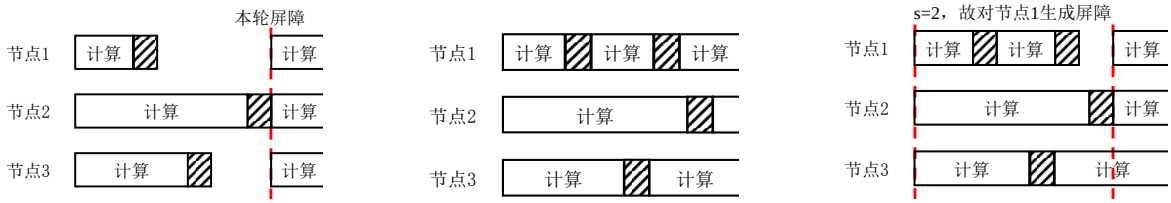
2） 全局通信：不同计算节点间相互交换各自的本地模型更新；

3） 屏障同步：为确保每个计算节点都已经收到了来自其他计算节点的模型更新，计算节点要进入屏障同步阶段，等待所有其他计算节点完成计算和通信；

BSP 协议需要所有计算节点都进入屏障同步后才能开始下一轮计算，在实际应用中存在慢机问题。即，在计算节点的性能存在差异时，整体的计算进度会被最慢的计算节点拖慢。

为解决 BSP 的慢机问题，Recht 等人提出了异步并行（ASynchronous Parallel，ASP）[19]。ASP 的特点是，各计算节点间的计算进度是独立的：快的计算节点在计算完一轮后只需要将本地模型更新推送给全局模型，而无需等待慢的计算节点完成便可以继续计算。相比于 BSP，ASP 协议的同步过程没有屏障同步阶段（如图 6(b)所示）。值得注意的是，ASP 协议虽然能够避免快的计算节点被慢的计算节点拖慢，但是存在过时计算问题，即，由于慢的计算节点基于陈旧的参数计算模型更新，当其计算完成时，全局模型参数可能已经被快的计算节点更新过多次，导致该模型更新过时。如果计算节点间的性能差异非常大，模型可能会无法收敛。由于 ASP 协议实现简单，主流的分布式机器学习框架也均支持 ASP 协议。

延迟同步并行（Stale Synchronous Parallel，SSP）[20]是一种介于 BSP 协议和 ASP 协议之间的有界异步协议。其特点是，允许最快的计算节点和最慢的计算节点间的迭代轮数之差不超过预先设定的阈值 s：当迭代轮数之差不大于 s 时，计算节点间无需等待；否则，最快的计算节点将被暂停以等待最慢的计算节点。图 6(c)展示了 SSP 的同步过程。BSP 和 ASP 可以被认为是 SSP 的特例：当 s=0 时，SSP 特化为 BSP；当 s=∞时，SSP 特化为 ASP。SSP 协议不仅在一定程度上解决了 BSP 协议存在的慢机问题，而且有效地控制了模型更新的陈旧程度。文献[20]和文献[21]对 SSP 在求解凸优化问题时的收敛性进行了理论分析，给出了 SSP 与 BSP 之间的收敛性差异与 s 之间的关系。因此在解决凸优化问题时，SSP 往往能够取得优于 BSP 和 ASP 的表现。但该理论并不适用于非凸优化问题。对于 DNN 等非凸优化问题，SSP 所导



(a) BSP (b) ASP (c) SSP

图 6 模型一致性协议

致的训练轮数的增加甚至超过了单轮训练速度的提升，使得 SSP 的收敛速度甚至慢于 BSP[22]。另外，随着 s 的增大，模型收敛时的准确性也会逐渐下降[23]。目前，支持 SSP 协议的分布式机器学习框架主要包括 SSPTable[20]和 Pettum[24]等。

如表 2 所示，不同的模型一致性协议在训练速度和模型收敛速度方面各有优势。由于 SSP 和 ASP 缓解甚至完全消除了屏障同步，使得计算资源能够被充分利用，从而提高了训练吞吐；但它们往往需要更多轮数的训练才能收敛到和 BSP 相同的水平。文献[19]证明了对于稀疏学习场景，即，每次模型更新只会修改小部分参数，ASP 协议是可以收敛的。文献[20]证明了对于 Lasso 回归这类凸优化问题，SSP 协议是收敛的，并且实验结果表明，相同时间内，使用 SSP 可以获得比 BSP 和 ASP 更好的训练效果。文献[21]的实验结果展示了对于 DNN 这类非凸优化问题，BSP 效果最佳。在实际应用中，BSP 协议由于收敛速度快，是目前最常用的模型一致性协议。

表 2 模型一致性协议比较

协议	训练吞吐	收敛速度	最佳适用场景
BSP	低	快	同构计算；DNN 等非凸优化问题
ASP	高	慢	异构计算；稀疏学习
SSP	中	中	异构计算；Lasso 回归等凸优化问题

3.2 参数同步架构

除模型一致性协议外，参数同步架构对大规模分布式机器学习系统的性能也有至关重要的影响。参数同步架构是指不同计算节点上的模型更新进行汇总，并对模型副本进行更新的方式。按每次模型更新是否推送给所有其他计算节点，参数同步架构可以分为中心化架构和去中心化架构；按模型更新是否在单一逻辑节点汇总，可以分为集中式架构和分布式架构。一般来说，中心化架构既可以是集中式架构，也可以是分布式架构；而去中心化架构一定是分布式架构。目前常用的参数同步架构大多为中心化架构，因此，除特别说明外，下文所提到的参数同步架构均为中心化架构。

3.2.1 扁平化的参数同步架构

参数服务器（Parameter server，PS）架构[25]是一种最为典型的架构设计，目前已经被大部分主流的分布式机器学习框架所支持，包括 TensorFlow、MXNet 和 Caffe 等。如图 7(a)所示，PS 架构将参与计算的所有节点划分为两组，分别是参数服务器组和工作者组。工作者组负责计算模型更新，参数服务器组负责汇总模型更新并维护全局模型。具体来说，PS 架构中，每次参数同步过程可以分为以下 4 个步骤：

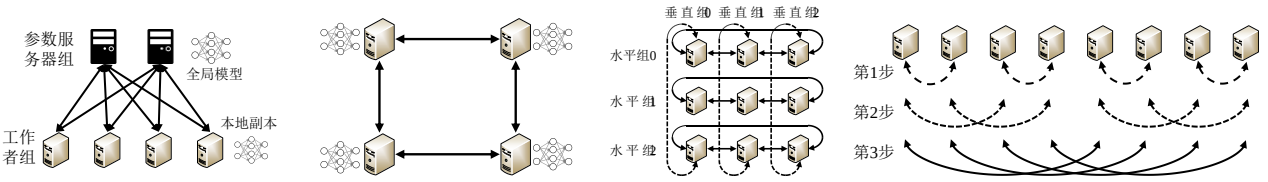
1） 参数拉取。每个工作者独立地从参数服务器上拉取全局模型的参数以更新本地模型副本；

2） 本地计算。不同工作者间并发地进行本地计算：每个工作者基于本地模型副本和训练数据计算模型更新，即，模型梯度；

3） 梯度推送。每个工作者独立地将所计算的模型梯度推送给参数服务器；

4） 汇总更新。参数服务器对来自不同工作者的模型梯度进行汇总，并将汇总后的结果用于更新全局模型。

当参数服务器组由多台参数服务器共同组成时，每台参数服务器只负责维护其中一部分参数的梯度汇总更新。在这种分布式参数服务器架构中，每台工作者需要与所有参数服务器建立连接，并将每个模型梯度推送给相应的参数服务器。可以看到，在参数服务器架构下，工作者之间以及参数服务器之间是不需要通信的，只有工作者和参数服务器之间存在模型数据的交互：工作者将模型梯度推送给参数服务器，并从参数服务器上拉取更新后的全局模型。值得注意的是，参数服



(a) 参数服务器架构 (b) 环规约架构 (c) HiPS 架构 (d) 递归二分和倍增规约架构

图 7 中心化参数同步架构

务器和工作者只是逻辑功能的区分。在实际部署中，同一个物理节点可以同时承担参数服务器和工作者的职能，这样的架构也被称为全连接（Full Mesh）架构。

PS 架构具有诸多优势，如支持各种模型一致性协议、弹性扩展较好、鲁棒性较强、容错性好等。但该架构也存在一些问题，当参数服务器侧总带宽小于工作者侧总带宽时，参数服务器很容易成为网络瓶颈，导致整体的训练速度降低。

环规约（Ring Allreduce）架构是一种分布式的架构设计，广泛应用于高性能计算领域。2017 年，百度硅谷人工智能实验室的一篇博客①提出将 Ring Allreduce 架构引入到分布式机器学习中来，自此，该架构开始受到大规模分布式机器学习网络研究者的关注。如图 7(b)所示，Ring Allreduce 架构将所有的计算节点以逻辑环的形式连接，每个节点只与其前后两个邻居节点进行通信。假设计算节点的数量为 n，Ring Allreduce 架构中，每次参数同步过程可以分为以下四大步骤：

1） 本地计算：各计算节点基于本地的模型副本和训练数据独立地计算模型梯度；

2） 分散-规约（scatter-reduce）：该阶段通信过程分为 n-1 步，并且每个节点将模型梯度分为 n 段。在每一步中，每个节点向后一个邻居节点发送一个梯度段，同时从前一个邻居节点接收一个梯度段。待接收完成后，将接收到的梯度段内容与本地相应的梯度段内容汇总，以用作下一步中发送的内容。这样，任何一个梯度段在经过 n-1 步传输后，都汇总了全部 n 个计算节点上相应梯度段的内容。即，最终，每个计算节点上都拥有一个汇总了全部计算节点上相应梯度段内容的全局梯度段；

3） 全收集（allgather）：该阶段与分散-规约阶段相似：将汇总后的梯度段在环结构中依次传输。不同的是，每个节点不再将接收到的梯度段和本地内容汇总，而是直接用接收到的内容替换掉本地内容。最终，在经过 n-1 步传输后，每个计算节点上拥有了全部的更新后的梯度段；

4） 参数更新：各节点使用汇总后的梯度更新本地模型副本。

相比于 PS 架构的“多对多”通信模式，Ring Allreduce 架构的“一对一”通信模式较为简单，在实际应用中，不容易造成网络拥塞。实际上，如果仅将带宽作为衡量通信成本的唯一因素，则 Ring Allreduce 架构是最优的[26]。但 Ring Allreduce 架构也存在一些问题，如难以支持 ASP 和 SSP 等模型一致性协议以及容错性差等。Horovod[27]是 Ring Allreduce 架构的一种实现，支持以插件的形式为 TensorFlow、PyTorch 和 MXNet 等多种机器学习框架提供参数同步功能。Horovod 已经成为实际部署中广泛使用的 Ring Allreduce 架构实现。

3.2.2 层次化的参数同步架构

PS 架构和 Ring Allreduce 架构虽然在参数规约的具体实现方面存在差异，但是具有一个共同的特点，即扁平化。当节点规模很大时，扁平化的参数同步架构会带来诸多问题，如 Ring Allreduce 架构下会有每个梯度段过小以及传输次数过多等问题；PS 架构会有跨机架流量大幅增多以及 incast 等等问题。层次化的参数同步架构可以有效地缓解这些问题。

Geng 等人提出了层次化的参数同步框架 HiPS[28]。HiPS 提出将计算节点按照层次化的方式进行组织，在每一层次中将所有节点分组，同一节点在不同层次中可以属于不同分组。不同层次可以独立地使用不同的参数同步架构。这里以两层 Ring Allreduce 为例进行说明。如图 7(c)所示，将 n 个节点编排为水平方向和垂直方向各 $\sqrt{n}$ 组，其中每组包括 $\sqrt{n}$ 个节点。第一阶段，每个水平组内对全部参数进行分散-规约；第二阶段，每个垂直组内对上一步规约后的结果继续进行分散-

① Bringing HPC Techniques to Deep Learning. https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/. 2017,02,21

规约；第三阶段，每个垂直组内对上一步规约后的结果进行全收集；第四阶段，每个水平组内将上一步全收集的结果继续进行全收集。值得注意的是，以上描述中，同一时刻仅有水平或垂直方向的通信，因此，可以使用另一进程按照“垂直组-水平组-水平组-垂直组”的顺序进行同步。由于两个方向的并行同步，每个方向的同步数据量可以减半。

HiPS 框架在索尼公司的工作[29]中也得到了具体应用，并被命名为 2D-Torus 算法。HiPS 框架的另一个实例化应用是 BML[30]。BML 以 BCube 网络拓扑为基础，通过将 HiPS 算法与其层次化的互连结构相结合，实现高效的参数同步。

递归二分和倍增（Recursive Halving-Doubling，HD）[31]是一种典型的树型规约方式（如图 7(d)所示）。同 Ring Allreduce 架构一样，HD Allreduce 架构最早也是应用在高性能计算领域，作为 MPI 在多进程之间实现高效数据归约的方式。最近，HD Allreduce 架构也被引入到分布式机器学习场景中来[32]。HD Allreduce 架构中，每次参数同步的过程也可以分为分散-规约和全收集这两个部分。不同于 Ring Allreduce 架构通过环状连接实现相应功能的方式，HD Allreduce 架构采用树状方式。具体来说，假设节点依次被编号 1~n，并且任意两节点间的距离定义为其编号的差值，则分散-规约过程可以分为 $\log_2 n$ 步：第 1 步时，距离为 1 的两个节点，如 1 和 2、3 和 4 等等，相互交换一半各自的梯度，即，发送一半梯度给对方，同时从对方接收另一半梯度，并汇总接收到的梯度和本地相应的内容；第 2 步时，距离为 2 的两个节点，如 1 和 3、2 和 4 等等，相互交换一半各自节点上拥有的上一步中汇总后的梯度。后续步骤依次将上一步汇总梯度中的一半与距离倍增的节点进行交换。最终，每个节点上都会拥有一个汇总了全部节点上相应梯度段内容的全局梯度段。全收集过程相当于分散-规约的逆过程，在此不再赘述。

HD Allreduce 架构中，每个计算节点一共需要建立 $\log_2 n$ 条连接，但每一步中仅有一条连接会被使用。故其也是“一对一”的流量模式。相比于 Ring Allreduce 需要 2*(n-1)次通信来说，HD Allreduce 仅需要 2*$\log_2 n$ 步，这在节点规模很大时，可以显著降低通信的次数。然而，HD Allreduce 架构中，不同步骤的通信需要在不同的节点间建立连接，而非像 Ring Allreduce 架构一样在不同的步骤中使用相同连接，这就极大地增加了产生拥塞的可能性。

3.2.3 去中心化的参数同步架构

近来，去中心化的参数同步架构也开始受到研究者的关注[33-34]。在去中心化的参数同步架构下，每个节点只与自己的邻居交换信息，当交换完成后，便可以继续下一轮计算过程。具体来说，首先，每个节点使用本地模型和训练数据计算本地梯度；然后，每个节点和其邻居交换参数信息，并将本地参数和邻居的参数信息汇总后更新本地模型；最后，使用本地梯度进一步更新本地模型。可以看到，任一节点都会利用到邻居的计算结果，同时其计算结果也都会被其邻居所利用。最重要的是，通过邻居之间的多次传递，任一节点的计算结果最终都会传播到其他所有节点上。因此，在一些文献中该算法也被叫做 gossip 算法[35]。相比于中心化架构，去中心化架构的主要优势在于有效地减少了通信流量。因此，当网络带宽较低时，采用去中心化架构的训练速度要快于中心化架构。然而，目前去中心化架构的应用场景仍然较少，主要应用于凸优化领域[36-37]。去中心化参数同步架构与异步训练具有相同的本质，即，某个计算节点的模型更新在多轮以后才会被传递到另一个计算节点。因此，在实际训练中，去中心化的参数同步架构仍会影响非凸优化问题的收敛速度。

3.2.4 参数同步架构小结

参数同步架构对通信的流量模式、通信次数、通信量等具有决定性影响。

表 3 参数同步架构性能比较

架构	通信次数	理论同步时间
PS	2	$2a+\frac{2(n-1)}{nB}$
Ring Allreduce	2(n-1)	$2a(n-1)+\frac{2(n-1)}{nB}$
HD Allreduce	$2\log_2 n$	$2a\log_2 n+\frac{2(n-1)}{nB}$
2D-Torus	4*($\sqrt{n}-1$)	$2a(\sqrt{n}-1)+\frac{2(n-1)}{nB}$
BML	4*($\sqrt{n}-1$)	$4a+\frac{2(n-1)}{nB}$

表 3 总结了不同参数同步架构的特点，其中

n 为计算节点数量，参数总量为单位 1，B 为计算节点的总通信带宽，a 为两节点间进行一次通信的时延开销。这些架构的通信量均为 $2*(n-1)/n$，其中 BML 假设为 2 层结构。可以看到，PS 架构中，每个节点的通信次数为常数，Ring Allreduce 的通信次数最多，层次化的参数同步架构可以大幅减少通信次数。从理论同步时间上来看，这些参数同步架构的差别主要在于时延项的不同。其中，Ring Allreduce 架构由于需要串行地通信 $2*(n-1)$次，其时延项占比较大。这也从理论上解释了为何 Ring Allreduce 架构在节点规模很大时，性能会有所下降。

4 通信效率优化

在实际部署中，通信的性能和效率也会对参数同步过程产生重要的影响。即使采用相同的参数同步模式，不同的通信方式也会对整体的训练性能造成很大的差异。因此，为了提高分布式机器学习训练时的通信效率，研究人员在以下方面进行了深入研究。

4.1 传输协议和通信库

传统的 TCP 协议虽然在广域网中得到了广泛的应用，但是在数据中心场景中却无法满足日益增长的流量需求。为弥补 TCP 在数据中心环境下的不足，研究人员提出了一些 TCP 的变体[38-42]。其中，Alizadeh 等人提出的 DCTCP[39]针对数据中心场景对 TCP 进行了优化，大大降低了传输时延。传统的 TCP 协议将丢包作为拥塞信号，当发送端检测到丢包时，才会减小发送窗口。这就使得大量的数据包在交换机队列中被缓存，从而导致排队时延大幅增加。DCTCP 通过交换机队列长度判断当前网络的拥塞程度，当交换机队列长度达到一定阈值时，交换机便会在数据包中加上 ECN 标记。接收端收到 ECN 标记的数据包后，在回传给发送端的 ACK 包中同样加以标记。发送端根据收到的 ACK 包中带有 ECN 标记的比例来调节发送窗口，从而实现更加及时且更加准确的拥塞控制。由于交换机队列使用率始终维持在一个比较低的水平，因此，数据包所经历的排队时延将会大大降低。更重要的是，由于交换机队列长度被控制在一个较低的水平，DCTCP 能够在一定程度上缓解 PS 架构下产生的 incast 问题。然而，当发送端数量很多时，DCTCP 仍无法避免交换机缓冲区溢出的问题，导致尾部流完成时间增加。

Xia 等人提出的 MLT 传输协议[43]是为分布式机器学习场景设计的专用传输协议。作者发现，机器学习算法具有有限的丢包容忍性，即，当网络随机丢包率小于一定阈值时，模型的收敛性基本不受影响。基于这一观察，作者设计了一种新的具备有限丢包容忍性的传输协议 MLT。不同于 TCP 的完全可靠传输和 UDP 的不可靠传输，MLT 提供了部分可靠传输特性，即保证预先设定的比例的数据包会被可靠地交付给接收端，而只为其他的数据包提供尽力而为的传输。当丢包发生时，采取如下方式处理：对可以被快速检测到的丢包仍会重传；对超时重传这类耗时较长才能被检测到的丢包，且丢包比例未超过预设阈值时，则直接丢弃。作者的观察结果显示，对 RNN 等模型，丢包容忍阈值一般在 10%~35%之间。因此，通过避免大量的超时重传，MLT 可以有效地降低尾部流完成时间。仿真结果表明，相比于 TCP，基于 MLT 的分布式机器学习训练速度可提升 10%~120%。需要强调的是，虽然 MLT 在所测试的模型和数据集上可以提高模型的训练速度，但是其在不同模型和数据集上的普适性尚未被理论证明。因此，其他模型基于 MLT 协议训练时的收敛性仍无法保证。

除对 TCP 协议本身进行优化外，近年来，远程直接内存访问（Remote Direct Memory Access，RDMA）技术也被引入到分布式机器学习训练中来，并成为大规模分布式机器学习训练的常见配置。RDMA 是一种硬件技术，它可以在无需 CPU 干预的情况下直接访问远程主机内存。具体来说，相比于 TCP，RDMA 具有以下优点：（1）零拷贝。数据直接从网卡传输到应用程序的缓冲区，避免了内存拷贝和系统调用开销；（2）内核旁路。通过将网络协议栈卸载到硬件中，绕过内核，RDMA 大大减少了总体延迟。（3）无需 CPU 参与。RDMA 可以访问远程内存，而无需占用 CPU 时间，从而节省了更多的 CPU 资源用于模型训练。因此，与现有的 TCP 方案相比，RDMA 可以实现更高的吞吐量和更低的延迟。目前，业界存在三种具体的 RDMA 技术标准，分别是 InfiniBand（IB）、RDMA over Converged Ethernet（RoCE）和 internet Wide-area RDMA Protocol（iWARP）。这三种技术都需要硬件支持，其中，IB 网络使用专用的网络协议栈，从硬件级别保证可靠传输，因此部署成

本非常高，广泛应用于高性能计算领域；RoCE 和 iWARP 则是基于以太网的 RDMA 技术，可以使用普通的以太网交换机，但需要使用专用的 RDMA 网卡。在实际应用中，受到底层 TCP 协议的限制，iWARP 的性能不及 RoCE①，因此，为了兼顾成本和性能，数据中心内使用的 RDMA 技术大多为 RoCE 协议。

目前，分布式机器学习框架中使用 RDMA 通信的方式可以分为两种，IB Verbs 和 GPU Direct RDMA（GDR）。对于 GPU 通信，前者需要先将数据从 GPU 显存拷贝到主机内存，然后再使用 RDMA 技术传输到对端主机内存，对端 GPU 则需要将数据从主机内存拷贝到 GPU 显存。相比之下，GDR 则为 GPU 提供了直接访问对端 GPU 显存的能力，从而减少了 GPU 通信过程中数据拷贝的次数，使得通信延迟进一步降低。Yi 等人的工作[44]表明，在包含 16 块 GPU 卡的实验床上，相比于 TensorFlow 中原生的基于 TCP 的通信方式和 Yahoo 公司提出的基于 IB Verbs 的通信方式②，GDR 可以分别将端到端训练速度提高 2.43 倍和 1.21 倍。

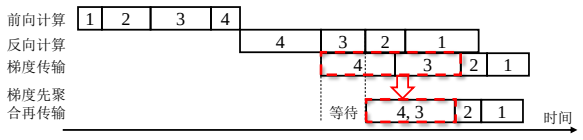


图 8 MG-WFBP 聚合算法示意图

Shi 等人发现参数同步过程中小尺寸梯度的频繁传输会导致传输效率的降低，因此需要尽量避免传输小尺寸梯度[45]。一般来说，当传输大小为 M 的数据，其通信开销为

$$T(M)=a+bM \tag{4}$$

其中，a 为时延项，b 为传输单位大小数据的传输时间。可以发现，M 越小，时延项所占比例越大，即通信效率越低。如图 8 所示，将多个小尺寸梯度聚合后再进行传输虽然会提高通信效率，但也会造成梯度更新后需要经过更长的时间才会被传输，从而降低计算和通信的重叠度。作者基于以上通信开销模型对分布式机器学习训练的单次迭代过程进行建模，并提出 MG-WFBP 算法来求解最优聚合方案。实验结果表明，在 8 节点环境下，相比于无需等待的反向传播 WFBP 和反向计算完成后将所有梯度聚合再传输这两种方案，MG-WFBP 可将端到端训练速度提升 1.2~1.36 倍。

NVLink③是 GPU 厂商 Nvidia 推出的一种总线及其通信协议，它能够在多 GPU 之间以及 GPU 与 CPU 之间实现高速互联。2016 年，Nvidia 发布了首款搭载 NVLink 的 P100 GPU，双向带宽达到 160GB/s，相当于 PCIe Gen3 * 16 带宽的 5 倍。2017 年，Nvidia 发布的 V100 GPU 上所搭载的 NVLink 2.0 可以将 GPU 双向带宽进一步提升至 300GB/s。2020 年，Nvidia 发布的 A100 GPU 上所搭载的 NVLink 3.0 将通道数从 6 提升至 12，总带宽也达到 600GB/s，几乎相当于 PCIe Gen4 * 16 带宽的 10 倍。NVLink 虽然可以提供更高的互联速度，却需要 GPU 和 CPU 等硬件的支持。目前，仅 IBM Power 系列的部分 CPU 支持 NVLink。

Nvidia 还开发了集合通信库 Nvidia Collective Communications Library（NCCL）④，可以在多 GPU 以及多主机间实现高效的集合通信。NCCL 高度优化和兼容了 MPI，提供了 allgather、reduce、broadcast 等集合通信原语，并且具有 GPU 互联拓扑感知能力，可以在 PCIe、NVLink、RDMA 上实现较高的通信速度。NCCL 1.0 版本仅支持单机多卡通信，GPU 卡之间可以通过 PCIe、NVLink 通信；NCCL 2.0 版本扩展了多机多卡通信能力，不同主机间可以通过 TCP 或 RDMA 来通信。相比于 MPI，NCCL 是为 Nvidia GPU 硬件量身打造的集合通信库，对数据搬移、任务下发、内存访问等细节均有很好的处理，因此可以大幅提高通信性能。Nvidia 官方提供的数据⑤显示，在 32 节点规模下，使用 NCCL 时的端到端训练性能是 MPI 的 1.95 倍。

在使用环规约架构时，需要保证所有节点均按照相同的顺序对模型参数进行同步，否则可能造成死锁。Horovod[27]采用“主从模式”来保证参数同步顺序的一致性：如图 9(a)所示，从进程将

① RoCE vs. iWARP Competitive Analysis. https://www.mellanox.com/related-docs/whitepapers/WP_RoCE_vs_iWARP.pdf. 2017,2

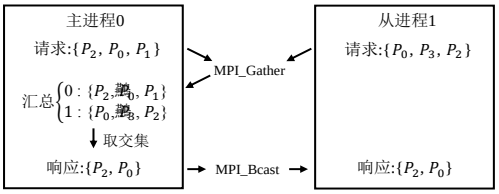
② TensorFlow on Spark. https://github.com/yahoo/TensorFlowOnSpark. 2017,2,5

③ NVLink and NVSwitch. https://www.nvidia.com/en-us/data-center/nvlink/. 2020

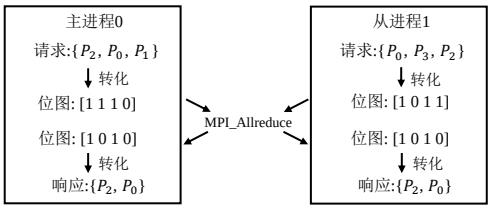
④ Nvidia NCCL. https://developer.nvidia.com/nccl. 2020

⑤ NCCL 2.0 report. https://on-demand.gputechconf.com/gtc/2017/presentation/s7155-jeaugey-nccl.pdf. 2017

本地待同步的参数 ID 集合发送给主进程，主进程收集到所有从进程发来的参数同步请求时，先筛选出所有请求的交集，然后将交集中包含的参数 ID 广播给所有从进程。这样，每个从进程按照只需要按照收到的交集中的参数顺序进行同步就可以保证全局一致性。以上过程分两个段：请求收集阶段使用 MPI_Gather 集合通信原语，交集广播阶段使用 MPI_Bcast 集合通信原语。随着节点规模的增大，这种“两阶段协商”机制的性能逐渐下降。为了改善这一问题，Laanait 等人提出了“单阶段协商”机制 -- bitAllReduce[46]。如图 9(b)所示，该机制首先将每个节点上的待同步参数 ID 转化为位图，然后利用 MPI_Allreduce 集合通信原语在所有的从进程间对不同节点上的位图进行规约，从而做到仅需一步便可完成参数同步顺序协调。作者在 1024 节点规模下测试发现，使用 bitAllReduce 时的扩展效率约为 NCCL 的 2 倍。目前 bitAllReduce 机制已经被集成到 Horovod 通信库中。



(a) “两阶段协商”机制



(b) “单阶段协商”机制

图 9 参数同步顺序协商

虽然 GPU 卡间可以存在多种通信通道，如 PCIe、NVLink 以及 RDMA 等，但是 NCCL 在任意两块 GPU 卡间仅会采用一种最高效的通信通道，无法充分利用所有的通道。除无法利用异构的通信通道外，任务调度也会造成同一任务使用的 GPU 之间使用异构的通信通道。

针对这一问题，Wang 等人提出了可以充分利用 GPU 间所有的同构/异构数据传输通道的数据聚合方案 Blink[47]。首先，Blink 打破了业界广泛采用的环规约架构，使用生成树来构建参数同步架构，从而避免环规约架构中存在的某一条低带宽链路限制整体通信速度的问题；其次，针对异构通信通道，Blink 可以根据通道的带宽差异，对不同通信通道上的数据传输比例进行相应的划分，从而达到均衡利用所有通信通道的目的。文中对使用 NVLink 和 PCIe 进行异构连接的 GPU 间通信吞吐进行测试，由于 NVLink 未能在 GPU 间形成环式连接，NCCL 2.0 仅能使用 PCIe 通道，而 Blink 则可以充分利用所有通道，从而将吞吐从 4.8GB/s 提升至 26.4GB/s。

4.2 网内聚合

随着可编程交换机的发展，研究人员开始探索如何将参数汇总功能从服务器转移到交换机，从而大幅提高参数汇总节点的进出口带宽，或者达到减少网络传输数据量的目的[48-51]。虽然网内聚合在概念上比较直观，但是在真实的可编程交换机上实现网内聚合功能却面临着以下挑战：

1） 交换机的计算能力有限。参数聚合功能的数学本质是计算若干数据的平均值。机器学习模型参数是以浮点型存储的，然而，目前可编程交换机仅支持整型运算和逻辑运算。此外，可编程交换机也不支持乘除计算。因此，可编程交换机无法直接计算模型参数的平均值。

2） 交换机的存储空间有限。一方面，从发展趋势上来看，机器学习模型的参数量在不断增大。例如，当采用数据并行方式时，BERT 模型的参数同步数据量高达 1.3GB①。相比之下，可编程交换机的存储空间仅数十 MB [52]。另一方面，存储需求还会随着计算节点的增多而增大。因此，可编程交换机无法将来自众多计算节点的模型参数先存储到本地再进行聚合。

3） 无可靠传输保证。从协议分层的角度看，可编程交换机不具备四层功能，即，无法保证数据包的可靠传输。当发生丢包时，不会触发重传机制，使得接收端陷入等待接收数据状态无法继续下一轮计算。

Sapio 等人在 2017 年发表的一篇短文[48]分析了实现网内聚合面临的挑战，并针对 MapReduce 应用实现了概念验证原型系统。继而，在 2019 年提出了 SwitchML 架构[50]，通过对交换机侧和端侧进行协同设计，将参数汇总过程分解，由端侧辅助完成复杂操作，在一定程度上解决了以上问题。

① BertLarge 分布式训练（流水并行）. https://www.alibabacloud.com/help/zh/doc-detail/194800.htm. 2021,1,19

首先，为了保证可编程交换机的线速转发能力，SwitchML 在端侧将浮点数转换为整型数。考虑到数据类型转换导致的精度损失，端侧先将原始模型参数进行适当伸缩，再将伸缩后的数据转换为整型数传输到交换机上。这样，当收到交换机汇总的结果后，只需要再通过逆操作便可以得到精度更高的汇总结果；其次，针对可编程交换机存储空间不足的问题，SwitchML 采用了基于聚合器池的流式处理协议。交换机侧具有由若干（设为 k）整形聚合器构成的聚合器池，端侧将模型参数切分成固定大小，并指定该参数切片使用的聚合器。端侧采用自同步的方式发送切片，即，初始时发送 k 个切片，随后每收到一个聚合结果便发送一个新的切片，从而保证交换机侧处理的数据量不会过载。图 10 展示了 SwitchML 对两个计算节点执行网内聚合的过程，其中每个节点各 5 个参数切片，并保存在 G_0 或 G_1 中，参数汇总后的结果保存到 A_0 或 A_1 中。最后，SwitchML 设计了重传机制，以避免陷入死锁状态。若端侧在一定时间内未收到聚合结果，则判定为发生了丢包，触发端侧重传。交换机侧收到重传报文后，若此前未收到该报文，则将该报文所携带的数据交给指定的聚合器，否则，则判定为聚合结果丢失，需要重传该结果。因此，聚合结果需要在交换机侧保留一定时间。SwitchML 采用“阴影复制”的方法，在交换机侧提供两个互为主从的聚合器池，端侧交替使用两者。由于自同步机制只有在收到聚合结果后，才会发送新的切片来使用该聚合器，因此，当交换机的池 A 中聚合器 i 聚合完来自所有端侧的切片时，可以获知由池 B 中聚合器 i 聚合的结果已经被所有的端侧接收到，池 B 的聚合结果可以被释放掉以接收新的切片。

实验结果表明，在 16 节点计算规模下，相比于 NCCL+RDMA（或 NCCL+TCP），SwitchML 可将 DNN 模型的端到端训练速度提升高达 2.2 倍（或 5.5 倍）。具体到通信性能，相比于 NCCL+RDMA（或 NCCL+TCP），SwitchML 可将通信性能提高 2.9 倍（或 9.1 倍）。虽然 SwitchML 在简单场景下取得了较大的性能提升，但如何将其扩展到多机架、多任务场景，仍需要更进一步的设计，并且部署成本很高。最重要的是，该方案不支持 SSP 和 ASP 等异步训练方式。

Mellanox 公司提出的可扩展分层聚合和缩减协议（Scalable Hierarchical Aggregation Protocol，SHARP）技术[①]通过将集合通信需要的计算操作从 CPU 卸载至交换机改进了 MPI 操作的性能。SHARP 技术使用专用的片上浮点运算器（FPU）完成计算卸载。相比于 SHARPv1，SHARPv2 采用流式聚合方式，提高了计算处理能力。SHARP 技术是基于 IB 网络设计的，因此，流控等机制均依靠底层网络，无法直接工作在以太网。此外，将计算操作固化到 ASIC 芯片虽然能够提高处理速度，但升级迭代也因此受到了限制。

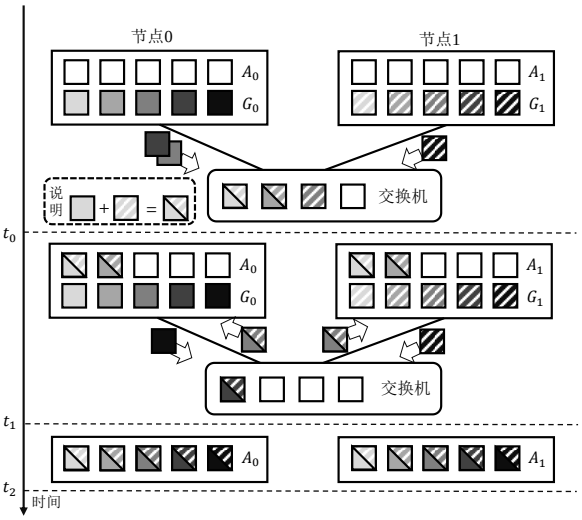


图 10 SwitchML 网内聚合过程

除了将参数聚合的计算操作卸载到交换机外，还可以提高参数服务器的网络通信能力。Parameter Hub[53]使用 PBox 架构为机架规模的参数服务器提供高吞吐、低时延的通信性能，同时兼顾集群的部署成本。PBox 的目标是匹配 IO 带宽和内存带宽。如作者使用的原型系统中，参数服务器的内存双向带宽为 120GB/s，因此可以使用 10 张 56Gbps 的网卡将该参数服务器与交换机进行连接，从而平衡 IO 带宽和内存带宽。除了硬件匹配外，还需要应用能够充分利用底层硬件的通信能力。为此，PBox 采用 chunk-to-core 的映射机制（见图 11），将数据块与网卡和 CPU 核绑定，任一网卡的通信上下文（如 RDMA 通信的 completion queue 和 queue pair）只会被一个 CPU 核读写，从而保证数据局部性，避免跨核流量降低通信性能。PBox 还针对更大规模部署进行了扩展设计。当在多个机架上执行训练任务时，PBox 采用分层规约算法。该算法分为以下三步：

① Mellanox SHARP. https://cn.mellanox.com/products/sharp. 2020

1） 每个机架内的 PBox 将该机架内所有工作者的梯度进行汇总；

2） PBox 节点将部分汇总结果在机架间进行汇总，并计算全局梯度更新；

3） 每个机架内的 PBox 将该全局梯度更新作用于模型参数，并将更新后的模型参数广播给该机架内的所有工作者。

作者对部署成本建模发现，当 ToR 交换机的超额订购比为 2：1 时，相比于分布式参数服务器架构，单位成本下 Parameter Hub 可以将训练吞吐提高 26%。同 SwitchML 一样，Parameter Hub 没有针对 SSP 和 ASP 等异步通信方式进行完善的设计。虽然 Parameter Hub 需要尽量保证不同工作者发送的梯度尽量同时到达 PBox 节点，以充分利用缓存性能，但是 Parameter Hub 中 PBox 服务器有更大的内存来存储收到的梯度。因此，相比于 SwitchML 和 SHARP 来说，Parameter Hub 扩展到 SSP 和 ASP 等异步训练场景的难度相对较低。

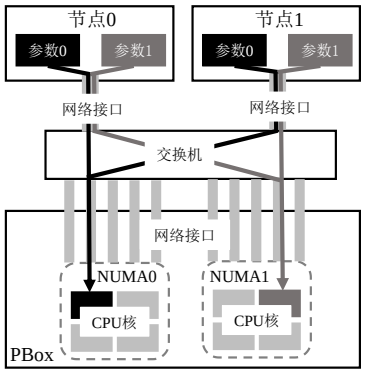
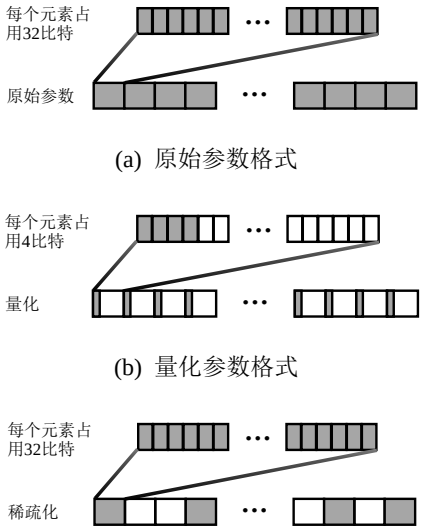


图 11 PBox 架构

4.3 模型压缩

模型压缩是指将梯度或参数压缩后再进行传输的方法。该方法虽然可以大大减少传输的数据量，但数据精度在压缩后也会降低，即数据压缩是有损的。这就导致模型在训练时的收敛性会受到影响，因此，模型压缩方法是以更多的训练轮数为代价来提高每一轮的训练速度的。图 12 展示了两种常见的模型压缩方式：量化（Quantization）和稀疏化（Sparsification）。

一般来说，模型参数或梯度采用 32 比特的全精度来表示每一个元素，而量化方法则通过使用更少的比特数来表示每一个元素，而传输的元素个数仍与原始数据相同。例如，对一个形状为[1024, 1024, 1024]的参数，使用全精度表示时，其大小为 4GBytes，而使用 16 比特的半精度来表示，其大小可以被压缩一半。文献[54]证明了在假设优化问题具有凸且稀疏的性质时，这种量化压缩方式是可以保证模型收敛的。



(a) 原始参数格式

(b) 量化参数格式

(c) 稀疏化参数格式

图 12 模型压缩方式对比

1-Bit SGD[55]将所有的梯度均进行了 1 比特量化，大大减少了参数同步过程传输的数据量。为了减少量化引入的误差，受 Delta-Sigma 调制的启发，1-Bit SGD 在将梯度进行量化时会保存此次量化引入的误差，并将该误差和下一轮计算得到的梯度相加后再将新的梯度进行量化：

$$G_i^q(\theta^t)=Q(G_i(\theta^t)+\Delta_i(\theta^{t-1})) \tag{5}$$

$$\Delta_i(\theta^t)=G_i(\theta^t)-Q^{-1}(G_i^q(\theta^t)) \tag{6}$$

$$\theta^{t+1}=\theta^t+\eta\frac{1}{n}\sum_{i=1}^{n}G_i^q(\theta^t) \tag{7}$$

其中，$Q(\cdot)$为对工作者节点 i 上的梯度 G 进行量化的函数，θ^t 为 t 时刻的模型参数。在经过量化压缩后，$G_i^q(\theta^t)$ 的大小只有 $G_i(\theta^t)$ 的 1/32，因此可以将网络传输的数据量大大减小。实验结果表明，在经典的 Switchboard 语音识别系统上，1-Bit SGD 可将训练速度提升 10 倍。值得说明的是，由于误差 Δ 会在随后的计算过程中不断更新并对梯度进行补偿，直至其累计量足够大时才会被聚合，因此，1-Bit SGD 也可以被认为是另一种形式的异步训练方式。1-Bit SGD 在其他场景下是否能够收敛仍未有理论证明。

Alistarh 等人分析了通信数据压缩程度和模

型收敛速度之间的关系，并基于此提出了Quantized SGD (QSGD) [56]。QSGD 不只包括一种量化方式，而是一系列量化压缩算法。使用该算法可以权衡每次迭代所传输数据的比特数。QSGD 的量化算法如下：给定非 0 参数向量θ，对其中的任一元素θ_i，其量化之后的值为

$$Q_s(\theta_i)=\|\theta\|_2\ \mathrm{sgn}(\theta_i)\xi_i(\theta,s) \tag{8}$$

其中，$Q_s(\cdot)$表示量化级别为 s 的压缩函数，$\mathrm{sgn}(\cdot)$为符号函数，$\xi_i(\theta,s)$为随机变量，其定义如下。假设$0\le l<s$，且 l 为整数，那么$|\theta_i|/\|\theta\|_2\in[l/s,(l+1)/s]$。$[l/s,(l+1)/s]$就是$|\theta_i|/\|\theta\|_2$的量化区间，其具体取值如下：

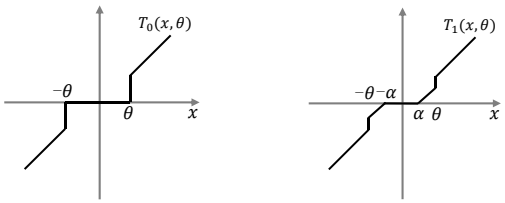
$$\xi_i(\theta,s)=\begin{cases} l/s, & \text{概率}1-p(\dfrac{|\theta_i|}{\|\theta\|_2},s) \\ (l+1)s, & \text{否则} \end{cases} \tag{9}$$

这里，$|\theta_i|$表示θ_i的长度，$p(a,s)=as-l$，$a\in[0,1]$。由此量化算法压缩之后的结果满足无偏性和方差有界性，并且压缩后参数向量的期望满足非 0 元素的数量不超过$s(s+\sqrt{n})$)，其中，n为参数向量中元素的总数量。

作者在 ImageNet[57]和 CIFAR-10[58]等数据集上评估了 QSGD 对图片分类模型的加速效果。实验结果表明，在使用 16 块 GPU 训练 AlexNet 模型时，使用 QSGD 可以减少 75%的通信时间，同时模型收敛时间可降低 60%。此外，作者也对语音识别场景下 QSGD 的加速效果进行了评测，模型收敛速度可提高 2.7 倍。和 1-Bit SGD 相比，QSGD 的性能更优，尤其是在训练 ResNet 和 Inception 等卷积层较多的神经网络时，QSGD 的优势更加明显。此外，相比于 1-Bit SGD，QSGD 无需为每个元素维护量化误差，更加节省内存空间。

不同于量化方式，稀疏化则是通过过滤掉不重要的参数或梯度来减少传输的元素个数，未被过滤掉的元素与原始数据保持一致。一般来说，随着模型趋于收敛，会有越来越多的梯度值将逐渐趋于 0，这些足够小的梯度值对模型的收敛速度影响远不及大的梯度值。因此，不必每一轮训练都传输这些小梯度。仍以上述形状为[1024, 1024, 1024]的参数为例，若其中有 90%的元素值足够小，则采用稀疏化方式可以将传输数据量压缩至 0.4GBytes。由此可见，相比于量化方式，稀疏化方式更加激进。

直观上，为了将梯度进行稀疏化处理，只需要设定一个合适的阈值θ：若某一维度的梯度值小于θ，则将其舍弃；否则，按原值处理[59]。这种简单截断的方法虽然简单，但梯度值较小也有可能是由于当前样本对该维度的训练不足导致的，将其简单截断可能导致该维度特征的丢失。Langford 等人在 2009 年提出了截断梯度法（Truncated Gradient）[60]，以改进简单截断的不足。截断梯度法在截断梯度时更加缓和，除要与阈值θ比较外，还要另一个阈值α比较：若梯度值与 0 的差值大于α与 0 的差值，则以梯度值与α的差值来更新，否则以 0 来更新。这样，截断梯度法使用两个参数来控制稀疏程度：两个值越大，模型越稀疏。简单截断法和截断梯度法的区别如图 13 所示：



(a) 简单截断法　　(b) 截断梯度法

图 13 固定阈值稀疏化方法

这种采用固定阈值的稀疏化方法的缺点在于需要选取合适的阈值。然而，由于模型和训练环境都具有多样性，在实际训练时，很难针对特定的模型和训练环境选取到合适的阈值。除固定阈值外，还可以采用动态阈值的方式[61-65]，即 Top-k 稀疏算法。

Dreden 等人依据预先设定的传输比例来确定正阈值和负阈值[64]。在每一轮迭代中，大于正阈值或者小于负阈值的梯度将会被传输，未被传输的梯度将会在随后的迭代中逐渐积累直至超过阈值被传输。该算法可以保证压缩比率恒定。Aji 等人对该算法进行了改进[65]，用一个绝对值阈值来代替正负两个阈值：如果某个梯度的绝对值大于该阈值，则被传输；否则，将被丢弃。此外，针对不同梯度的尺度不同的问题，Aji 等人也提出使用层归一化（layer normalization）方式来归一化不同的梯度，从而优化全局阈值的效果。

除梯度可被压缩外，参数也可以被压缩。然而，Top-k 稀疏算法中，不同工作者独立地选取需要传输的梯度，容易导致不同工作者选取的梯度

差别很大。这样，在每一轮迭代中，被更新的参数数量几乎与工作者数量成正比。假如有 m 个工作者，每个工作者独立地选取 k 个梯度，且任意两个工作者选取的梯度均完全不同，则会有 $k*m$ 个参数会被更新。由此可见，在大规模训练时，几乎所有的参数都会被更新，并被传输到工作者上以供下一轮计算使用。由此可知，参数并未被有效地压缩。Shi 等人提出 gTop-k 算法[66]利用树结构在梯度聚合过程中逐轮稀疏化，从而将参数服务器到工作者的流量从 O($k*m$) 降低到 O($k*\log m$)。

量化和稀疏化方式也可以结合起来进一步压缩传输数据量。SBC（Sparse Binary Compression）算法[67]首先利用传输比例 p 筛选出绝对值较大的梯度，然后将这些梯度分为正梯度值和负梯度值，并分别求得正平均值 μ^+ 和负平均值 μ^-。若正平均值 μ^+ 大于负平均值 μ^-，则将所有负梯度值置为 0，所有正梯度值置为 μ^+；否则，将所有正梯度值置为 0，所有负梯度值置为 μ^-。这样，只需要传输一个梯度值和非 0 梯度的索引即可。

总的来说，模型压缩，尤其是稀疏化，可以大幅减少分布式机器学习训练过程的通信数据量。以 SBC 算法[67]为例，梯度稀疏化的压缩率可达到 1/2363。然而，对于任意的机器学习模型，这些压缩算法是否会影响模型收敛性尚需充分评估[54]。

4.4 通信调度

分布式机器学习系统中每轮迭代的时间除与计算耗费时间和通信耗费时间相关外，还受计算和通信之间的重叠程度影响。从计算的角度看，机器学习模型具有层次化结构：在每轮迭代中，前向计算过程按照从输入层到输出层的方向逐层计算，并将第 i 层的计算结果作为第 i+1 层的输入；反向计算过程则按照从输出层到输入层的方向逐层计算，并基于前向计算的中间结果和第 i+1 层产生的模型误差来计算第 i 层参数所对应的梯度。从通信的角度看，现有的分布式机器学习框架大多采用无需等待的反向计算方式（Wait-Free Back-Propagation， WFBP）[68]，即，第 i 层梯度计算完成后可以立即被同步，而无需等待所有梯度计算完。综合计算过程和通信过程来看，参数同步的顺序和前向计算消耗的参数顺序恰好相反，比如，前向计算首先消耗第 1 层参数，而该参数在反向计算的最后才会被同步，由此导致下一轮迭代时必须要等待所有参数都同步完才能开始新的计算过程。近年来，为了优化模型计算和网络通信之间的重叠程度，一些相关工作[8-9][69-71]提出对参数同步过程进行更加合理的调度，从而有效地利用网络资源。

2019 年，Hashemi 等人 **Error! Reference source not found.**通过大量实验注意到 TensorFlow 等主流的分布式机器学习框架在同步参数时未考虑计算和通信之间的重叠，参数的传输顺序具有随机性。随机的参数传输顺序为迭代时间带来了不确定性，不但容易导致计算资源的浪费，而且不同工作者获取到参数的顺序不一致，也会导致落后者效应，即，获取到参数的顺序最差的工作者成为落后者，其他工作者不得不等待其计算完成后才能开始新一轮计算。

进一步地，Hashemi 等人通过理论分析发现参数调度问题为 NP 难问题，并提出两种启发式算法—时间无关的通信调度算法 TIC 和时间相关的通信调度算法 TAC，并统称为 TicTac。TIC 算法以机器学习模型对应的计算图为唯一输入，通过对计算图进行遍历，得到各通信操作符所依赖的通信次数，然后按照依赖的通信次数由少到多的顺序依次降低通信操作符的优先级。TAC 算法的输入除计算图外，还包括操作符消耗的时间。基于更丰富的信息，TAC 算法可以进一步优化参数传输顺序。如图 14，A 和 B 两个通信操作符所依赖的通信次数相同，TIC 算法将会随机分配优先级，但 TAC 算法则会通过比较计算操作符 1 与接收操作符 B 的重叠程度和计算操作符 2 与接收操作符 A 的重叠程度来决定 A 和 B 的优先级。

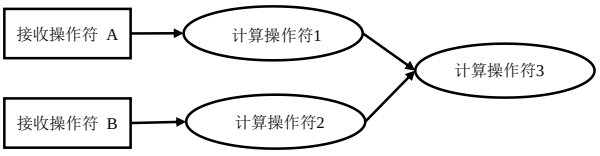


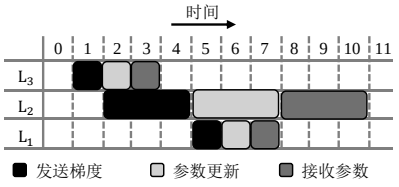
图 14 TIC 和 TAC 区别

TicTac 为每个通信操作符分配一个唯一的优先级编号 p，表示发送端在执行该通信操作符前需要先完成 p 个通信操作符。实验结果表明，对于 ResNet-101 等典型的 DNN 模型，TicTac 可以将训练和推理速度分别提升 20%和 37%。然而，相比于 TIC 算法，TAC 算法并未表现出明显的性能优势，这说明对于大多数机器学习模型来说，计算图已经可以提供足够的信息来实现接近最优

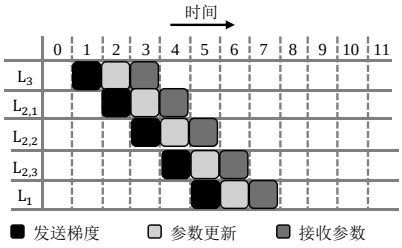
的参数调度。此外，TicTac 所使用的优先级调度方式为非抢占式调度，即，高优先级参数未被发送前，低优先级参数即使已经就绪，也无法被发送，反而造成网络资源的浪费。

与 TicTac 同时，Jayarajan 等人也注意到参数同步的顺序不但要考虑梯度生成的顺序，更要考虑参数消耗的顺序，并提出了基于优先级的参数调度算法 P3[Error! Reference source not found.]。P3 认为层粒度的参数同步过于激进，会导致资源利用率低下。图 15(a)展示了一个具有 3 层参数的 DNN 模型的参数同步过程。层粒度的参数同步使得在参数同步时很难同时利用所有资源，反观使用细粒度的参数切片①时，在第 3 秒开始便可以同时利用计算资源和双向带宽资源，从而降低参数同步时间。

基于参数切片的思想，P3 为每个参数切片分配一个优先级编号：被前向计算消耗的越早，优先级越高，并且同一层的参数切片具有相同优先级。待传输的参数切片在计算完成后进入优先级队列，P3 从中选取优先级最高的参数切片进行同步。由于传输的粒度从层级别细化为切片级别，这样，即使更高优先级的参数切片进入优先级队列，最多也只需要等待一个切片的传输时间，从而既保证了优先传输高优先级参数，又可以在无高优先级参数时传输低优先级参数，即抢占式调度。



(a) 层粒度



(b) 切片粒度

图 15 参数同步耗时

在 4 节点集群上的实验结果表明，对于 VGG-19 模型，P3 通过对参数切片进行调度可将训练速度提升 49%，并且当互联带宽从 30Gbps 降低到 15Gbps 后，P3 和标准分布式机器学习训练之间的性能差异进一步拉大到 66%。需要说明的是，切片的大小对于训练速度有着重要的影响。随着切片粒度变小，高优先级参数可以更及时地被传输，网络资源也可以被更加充分地利用。然而，过小的参数切片会导致参数的切分和聚合开销提高，甚至消除参数切片带来的性能增益。作者通过实验得到最优切片大小的经验值为 50000。

P3 采用“停止-等待”的方式实现抢占式调度：每次只发送最高优先级的参数切片，便停止发送直至收到该次传输的确认才会继续发送下一个参数切片。该方式虽然可以保证高优先级参数切片及时被传输，但是会造成网络带宽的浪费。针对这一问题，ByteScheduler[8]提出了基于 credit 的抢占式调度。类似滑动窗口，credit 允许发送多个参数切片而无需等待收到其确认，同时又对未被确认的参数切片数量进行了限制。例如，有 4 个参数切片 $p_1 \sim p_4$，其优先级依次升高。当 p_1 在被传输时，p_2、p_3、p_4 依次就绪。若 credit 为 2，则当 p_2 就绪时，即使 p_1 仍在传输，p_2 也可以立刻被传输，即，传输顺序为 p_1 -> p_2 -> p_4 -> p_3；若 credit 为 1，即“停止-等待”方式，则只有当 p_1 传输结束时才会传输其他参数，因此，传输顺序为 p_1 -> p_4 -> p_3 -> p_2。由此可见，较大的 credit 虽然可以提高带宽利用率，但会降低高优先级抢占的及时性。ByteScheduler 将 credit 和切片大小作为两个系统参数，并使用贝叶斯优化算法对其进行优化，以获得最优的训练性能。此外，ByteScheduler 是第一项同时兼容多种机器学习框架和参数同步架构以及传输协议的工作。实验结果表明，在相同环境条件下，相比于 P3，ByteScheduler 对 VGG-16 等模型的训练性能提升可多达 43%。

以上通信调度方案均工作在端主机侧，通过控制分布式机器学习应用向主机内的通信协议栈传递的参数顺序来实现通信调度。然而，在分布式参数服务器场景下，这些调度方案无法对不同参数服务器上的参数传输进行调度。假设有 2 个不同优先级的参数，且位于不同的参数服务器节点，这些端侧调度方案由于只能控制每个节点内的参数传输顺序，因此无法保证工作者尽早地接收到高优先级参数。为了保证高优先级参数在网

① L_{ij} 表示第 i 层参数对应的第 j 个切片。

络中可以被优先传输，Wang 等人提出了全网级别的参数调度方案 Geryon[9]。传统的分布式机器学习框架在任意两个节点间仅建立一条连接，Geryon 打破了这一限制，提出在任意两节点间可以建立多条连接，并为这些连接分配不同的优先级，然后使用不同优先级的连接传输参数。具体来说，Geryon 首先根据参数所处的位置为该参数分配一个紧急度。假设该模型共 L 层，则第 l 层参数的紧急度为 $1-l/L$。然后，通过紧急度阈值为参数分配相应的优先级。比如，使用两个优先级连接，且紧急度阈值为 0.6，则紧急度为 0.7 的参数将会被分配高优先级，即，该参数/梯度的传输均使用高优先级连接。

Geryon 是第一个将参数调度转化为流调度的工作。流调度方案具有以下特点：

1） 抢占式。大多数网络设备已支持严格优先级调度（strict priority），既可以提高带宽利用率（work-conservation），又可以保证高优先级参数可以被及时转发；

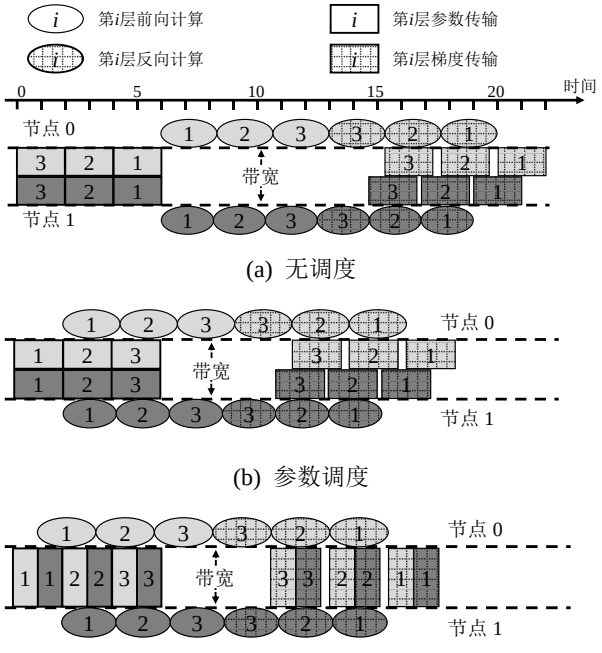
2） 细粒度。流调度通常以数据包（1KB）作为调度的最小粒度，远小于参数切片大小（200KB）。更重要的是，将参数拆分为多个数据包或是将多个数据包还原为参数的操作可以通过 LSO/LRO 等硬件卸载技术完成，大大提高了网络吞吐；

3） 全网规模。优先级信息以标签的形式插入到每个数据包的包头，因此，无论是在端侧还是在交换机侧，均可以基于优先级标签对数据包进行调度，从而避免了端侧调度方案中参数进入网络中后无法被调度的问题。

在 8 台 GPU 服务器上的实验结果表明，在 10G 网络下，Geryon 可将分布式训练的扩展效率提高至 95%。相比之下，标准的 TensorFlow 分布式训练仅有不足 40%的扩展效率。另外，与端侧调度方案的对比实验展示了 Geryon 的性能几乎不受切片大小的影响，而端侧调度方案的性能则随着切片尺寸的减小逐渐降低。

近年来，异构计算在机器学习领域已变得非常普遍。然而，异构硬件计算平台为分布式机器学习集群的迭代升级带来了成本问题。以 GPU 为例，对大规模分布式机器学习集群来说，整体升级成本非常大，因此当新一代 GPU 面世时，往往通过在原有集群的基础上增设部分新一代 GPU 进行系统扩容，从而导致具有不同计算性能的多代 GPU 共存的现象在大规模机器学习集群中十分常见。当使用不同性能的计算设备来进行分布式机器学习训练时，低性能的节点成为整体训练的瓶颈。针对这一问题，Wang 等人提出了面向迭代同步应用的计算高效的流调度方案 CEFS[71]。

一般来说，传统的流调度方案通常以最小化平均流完成时间或者最大化有效完成流数量为优化目标。然而，在以分布式机器学习训练为代表的迭代同步应用中，优化这些指标并不意味着训练性能的提升。CEFS 指出，对于分布式机器学习训练，流调度的目标应该是最小化计算资源的闲置时间，从而最大化整体的训练性能。如图 16 所示，一方面，CEFS 优先调度紧急参数，使每个工作者都可以尽早地开始计算；另一方面，CEFS 优先调度低性能节点，使低性能节点的计算被阻塞的时间最小化，从而有效缓解其对整体性能的影响。这样，对于任一参数，其优先级除受其紧急度影响外，还取决于其目的节点的性能。CEFS 采用贝叶斯优化算法为每个参数分配优先级，并采用二维保序规则来过滤掉不符合要求的分配结果。二维保序规则是指：1）对同一个节点，分配给较高紧急度参数的优先级不低于较低紧急度参数的优先级；2）对同一个参数，分配给较低性能节点的优先级不低于较高性能节点的优先级。



(a) 无调度

(b) 参数调度

(c)参数调度和节点调度

图 16 CEFS 调度效果示意图

在 16 台异构 GPU 服务器上的实验结果表明，相比于标准的 TensorFlow 和 ByteScheduler，CEFS

可分别将训练吞吐提升多达 253%和 47%。CEFS 之所以能够达到更高的性能，主要在于其能够更好地保证参数传输顺序。例如，训练相同的模型，CEFS 的参数传输乱序度[①]为 57，而 TensorFlow 和 ByteScheduler 则分别为 300 和 126。此外，对于 VGG-19 等经典 DNN 模型，贝叶斯优化算法在经过 10 轮左右迭代后，可以使模型的训练速度收敛。

不同于其他优化方案，虽然参数调度优化可以在既不影响模型收敛性，又无需额外的硬件升级的前提下加快分布式机器学习训练速度，但需要强调的是，以上通过参数调度来加速分布式机器学习训练的方案均对模型的计算/通信比有一定的要求。当计算/通信比过高或者过低时，即使对参数进行有效的调度，也无法显著提高计算和通信的重叠程度，因此无法改善分布式训练性能。例如，在网络互联带宽为 1Gbps 时，P3 和标准的 MXNet 分布式几乎拥有相同的性能，因为此时通信时间已远大于计算时间。由此可见，通信调度优化的应用场景还是受到一定限制的。

5 并行方式优化

如上文所述，数据并行和模型并行是两种经典的分布式机器学习训练方式。对于数据并行来说，通信开销主要来自不同计算节点间的参数同步；对于模型并行来说，当某个计算节点的输入来自另一个计算节点的输出时，便会产生通信开销。当模型参数量小于中间计算结果的数据量时，数据并行带来的通信开销较小；反之，模型并行的通信开销更小。然而，对模型整体使用某一种并行方式，可能无法达到最优的训练性能。因此，一些工作[68-74]提出使用混合并行、流水并行等方式，通过细粒度的并行优化来提升分布式训练性能。

卷积神经网络主要由卷积层和全连接层构成，但这两种功能层却呈现不同的特点。卷积层参数仅占全部参数的很少部分（通常低于 10%），却需要大量的计算（通常高于 90%），而全连接层则恰好相反。基于这一观察，Stanza[72]提出将卷积层和全连接层的训练解偶（见图 17）：使用大部分计算节点以数据并行的方式训练卷积层，剩余的小部分计算节点以数据并行的方式训练全连接层。这样使得原本产生大部分通信流量的全连接层参数同步仅发生在少数的计算节点间，有效地减少了参数同步带来的通信流量。同时，卷积层和全连接层构成模型并行。一般来说，最后的卷积层输出数据量较小，在卷积层计算节点和全连接层计算节点间传输中间计算结果时，即使使用多对一/一对多的通信方式，引入的通信流量也比较小。此外，卷积层参数同步可以和全连接层参数同步重叠起来，进一步缩短参数同步时间。为了确定负责卷积层和全连接层的计算节点数量，Stanza 建立了性能模型，以最大化训练吞吐为优化目标，对计算节点分配问题进行求解。

作者使用 Nvidia Tesla V100 GPU 在 10G 带宽环境下测试发现，随着计算节点数量的增加，Stanza 相比于参数服务器架构的性能提升越来越显著。在 10 节点规模下，Stanza 相对于参数服务器架构的性能提升高达 13.9 倍。

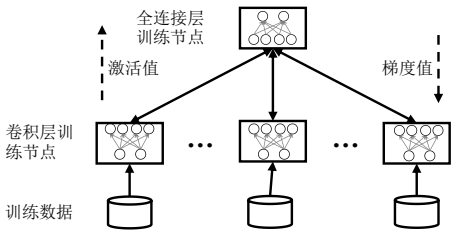


图 17 Stanza 训练示意图

Geng 等人从同步开销、GPU 利用率、负载均衡、落后者消除以及 I/O 扩展性等 5 个方面对数据并行和模型并行进行比较[73]，得出了以下结论：在负载均衡和 I/O 扩展性方面，数据并行更具优势；在其他方面，模型并行则更有优势。具体来说，由于数据并行下每个工作者的计算量基本相同，因此负载均衡性能较好。相比之下，模型并行划分的多个子模型之间的计算量差异则相对较大，导致不同工作者的负载相对不均。此外，数据并行可以减小每个工作者节点的 I/O 数据量，因此具有较好的 I/O 扩展性。基于该结论，作者提出了混合并行训练架构 Hove。Hove 首先将计算节点分组，组内使用模型并行，组间使用数据并行，从而保证 I/O 扩展性；然后采用“先组内汇总，后组间同步”的方式提高参数同步效率；最后，在负载不均时，通过自动调节机制对模型进行动态划分，以消除落后者。

PipeDream[74]在混合并行的基础上进一步引

① CEFS 提出乱序度来衡量参数到达工作者的顺序和最优顺序之间的差异。乱序度为违反计算图依赖关系的参数之间的距离之和。

入流水方式来优化分布式机器学习训练性能。混合并行中使用模型并行划分计算任务虽然比较简单，但却面临一个重大挑战：机器学习训练为双向过程，反向计算需要基于前向计算的中间结果来更新参数。因此，当某个子模型完成前向计算后，该计算节点需要保存计算结果以保证反向计算的正常执行。如下图所示，这种简单的模型并行方式会极大地浪费计算资源。为解决这一问题，PipeDream 提出了新的任务调度算法 1F1B。如图 18 所示，1F1B 以流水线的方式处理多个训练批次：当某个计算节点完成其所分配的子模型的计算，便将计算结果传给后一计算节点，同时该计算节点开始对下一个批次执行前向计算。当最后一个节点完成前向计算后立刻对该批次执行反向计算，并在反向计算完成后把梯度计算结果传给前一节点继续执行反向计算过程，同时，该计算节点开始对下一个批次执行前向计算。这样，当进入稳定状态时，每个节点均以前向计算和反向计算交替的方式进行计算，并且不存在计算资源浪费的情况。

基于模型 profiling 结果，PipeDream 的模型划分算法可以输出如何划分模型、各子模型需要的计算节点数量以及使得流水线饱和的最优并发批次数量。然而，流水并行带来的新问题是模型更新作用的参数版本和前向计算使用的参数版本不一致。为便于说明，定义 θ_l^t 为被 t 个批次更新过的子模型 l 的参数。如图 18 所示，如果在每个节点上只维护一个参数版本，那么对于工作者节点 0 来说，批次 5 的前向计算使用的参数为 θ_0^1，但是，被批次 5 所计算的梯度作用的参数却是 θ_0^4。PipeDream 使用参数暂存（weight stashing）技术来避免这一问题。参数暂存会在每个工作者节点上为每一个活跃的批次维护一个参数版本，前向计算时使用最新版本的参数，反向计算时使用的参数版本和前向计算保持一致。如图 18，工作者节点 0 会将批次 5 的梯度作用在 θ_0^1 上，而非 θ_0^4。虽然不同计算节点使用的参数版本仍不一致，但实验结果展示了参数暂存技术可以有效地保证模型收敛性。

为了避免模型并行导致的计算资源浪费问题，GPipe[75]也提出了微批次（micro-batch）流水并行策略。如图 19 所示，在前向计算时，GPipe 首先将每个批次划分为多个微批次，然后以流水线方式处理多个微批次。当所有微批次完成反向计算时，不同微批次产生的梯度会被汇总，并用来更新模型。由于所有流水线操作均发生在一次迭代内，因此，不同于 PipeDream，GPipe 不存在参数版本过时的问题。但相比于 PipeDream，GPipe 的计算过程仍存在“气泡”，即计算资源闲置，的情况。文献[74]的对比实验结果表明，PipeDream 的训练性能是 GPipe 等简单流水并行方式的 1.7 倍。

图 18 PipeDream 流水并行和参数暂存示意图

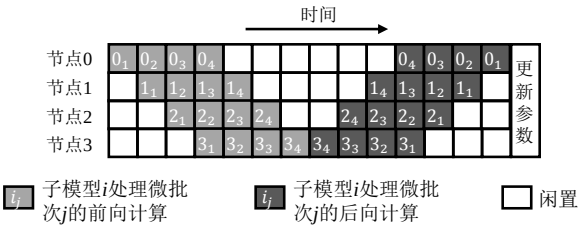


图 19 GPipe 流水并行示意图

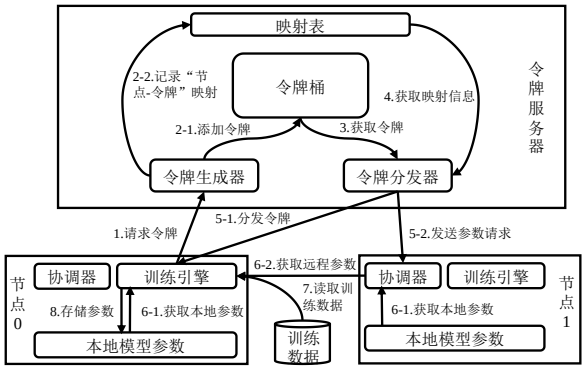


图 20 Fela 架构

Fela[76]采用更细粒度的灵活并行策略。更细粒度是指将模型逐层划分，并对不同层使用不同的 batch size。这是因为作者发现，不同层在计算时达到饱和所需的 batch size 不同。当使用统一的 batch size 训练不同层时，由于显存限制，无法使用足够大的 batch size 以使所有层均能达到计算饱和状态。因此，Fela 将模型训练拆分成多个子任务，并采用令牌分发的方式协调不同子任务（见图 20）。Fela 将服务器分为令牌服务器和工作者两个角色。不同于参数服务器，令牌服务器只负责令牌的生成和分发，并不维护模型参数。在每次迭代开始时，工作者首先向令牌服务器发送请求，

由令牌服务器返回令牌；然后工作者根据令牌信息训练相应的子模型，并将新的参数保存在本地；最后，工作者向令牌服务器发送任务完成通知，并请求新的令牌。当令牌服务器收到其他工作者的任务完成通知后，将通知所有工作者互相同步子模型参数。此外，需要强调的是，由于模型计算是逐层进行的，因此令牌服务器在生成令牌时需要等前层的计算结果就绪时才会生成后层的令牌。

6 网络拓扑优化

除以上优化方案外，分布式机器学习系统领域的研究者也对分布式训练集群所使用的底层物理网络拓扑提出了优化方案。

Wang 等人通过理论分析和仿真实验发现基于 Fat-Tree 网络拓扑（见图 21）同步参数时，同步性能要低于 BCube 拓扑[77]。如图 22 所示，BCube 是以服务器为中心的拓扑结构，每个服务器节点均有多个网卡，因此 BCube 提高了每个节点的互联带宽。这样，即使对于同样的参数同步架构，相比于 Fat-Tree，基于 BCube 进行参数同步也可以有效减少理论参数同步时间。以 HiPS 算法为例，若使用 k 级 BCube 拓扑，即每个服务器节点上有 k 块网卡并分别连接到不同的交换机上，其理论参数同步时间仅为 Fat-Tree 拓扑的 $1/k$。仿真实验结果表明，当在 Fat-Tree 拓扑中使用 RDMA 协议同步参数时，存在流量负载不均衡，PFC 暂停帧蔓延等问题。需要注意的是，全局参数同步时间主要取决于所有节点间最晚完成的参数传输，因此流量负载不均会导致不同节点间参数传输时间出现很大的差异，从而增大全局参数同步时间。除性能优势外，BCube 也具有成本优势：构建相同规模的集群，BCube 所需要的交换机数量仅为 Fat-Tree 的 $k/5$。基于以上分析，作者认为分布式机器学习训练集群的底层物理网络拓扑应使用 BCube，而非传统数据中心中常用的 Fat-Tree。

阿里巴巴自建的分布式机器学习训练集群 EFLOPS[32]也注意到在传统的分布式训练方案中，服务器上一般会配置多个 GPU 但只配备一张网卡，导致多 GPU 访问网络时，唯一的网卡成为限制系统性能的瓶颈。为了提高服务器的带宽，EFLOPS 为每个 GPU 配备专用的网卡。即使同一节点内的 GPU 之间通信也会被导出到节点外，从而绕过 PCIe 带宽瓶颈。

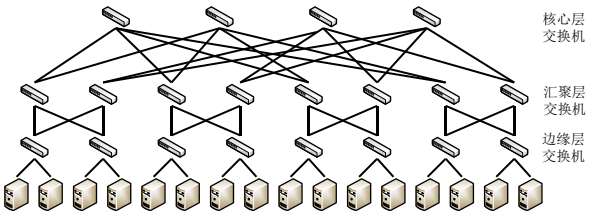


图 21 Fat-Tree 拓扑

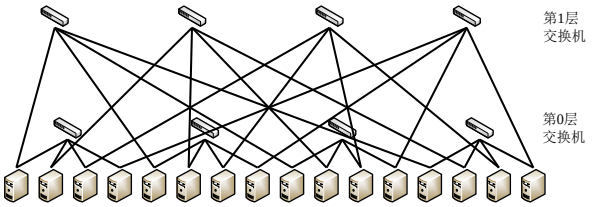


图 22 BCube 拓扑

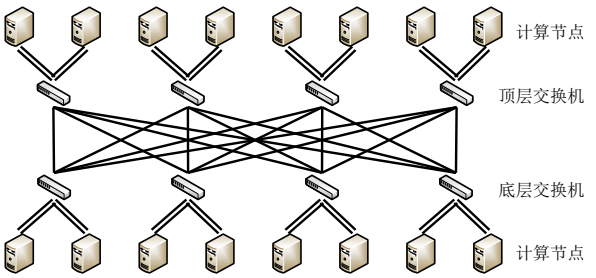


图 23 BiGraph 拓扑

基于多网卡服务器结构，EFLOPS 提出了 BiGraph 网络拓扑。如图 23 所示，BiGraph 将网络分为上下两层，两部分之间通过 Clos 架构互连。顶层交换机和底层交换机均可以连接服务器，且服务器与同一交换机之间存在多条链路。这样，任一顶层服务器和底层服务器通信均只需要 3 跳，且最短路径具有唯一性。这一特性为应用控制流量路径提供了支持，即，当通信的两端确定时，流量路径也是确定的。因此，利用该特性，EFLOPS 提出了带编号映射的递归二分和倍增（Halving-Doubling with Rank-Mapping，HDRM）算法。HDRM 算法在 HD Allreduce 的基础上，通过将逻辑连接和物理链路进行一一映射，使得通信仅发生在不同部分的服务器之间，且在每一步中每条链路上仅存在一条数据流，以避免不同连接之间的链路争用问题，从而彻底解决网络拥塞问题。作者通过对比实验展示了，相比于 HD Allreduce，HDRM 算法可将通信延迟降低 40%-50%，同时将网络吞吐提高一倍。

此外，Liu 等人提出的 PSNet 拓扑[78]也通过

增加参数服务器和交换机互连链路的数量来提高

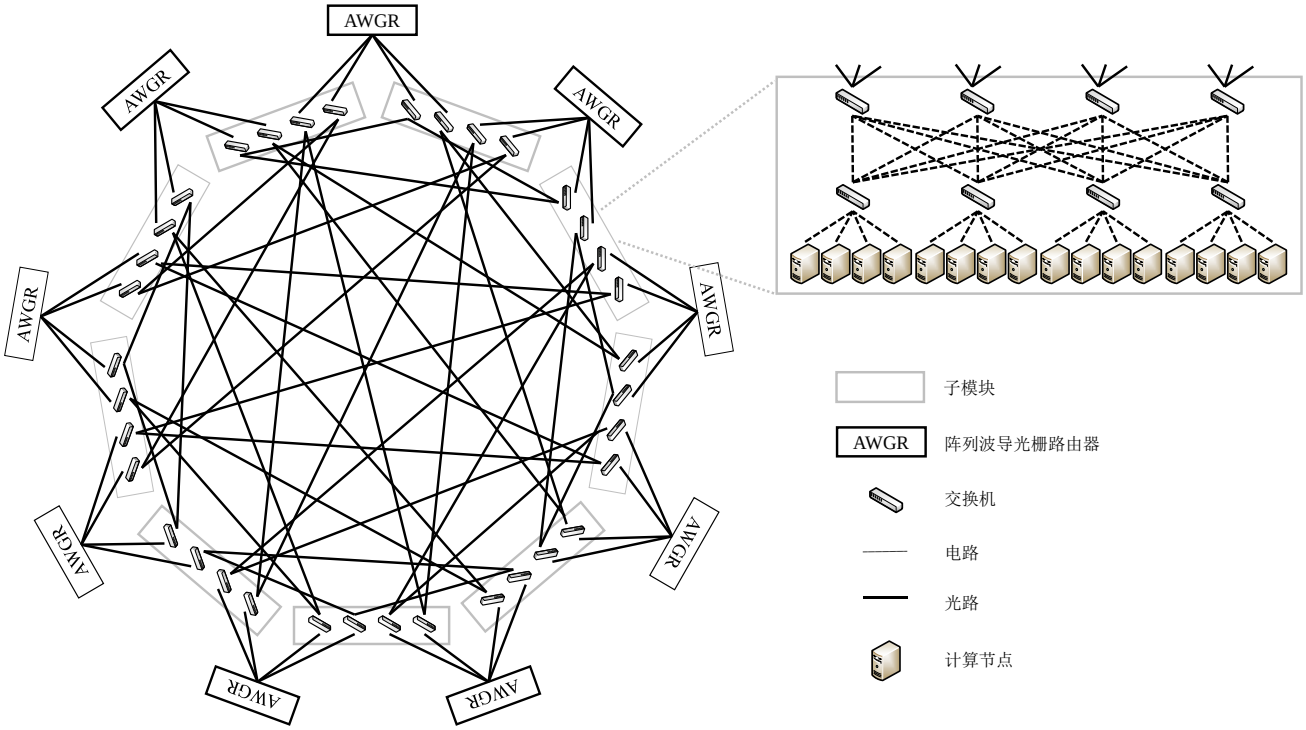


图 24 Lotus 拓扑

网络带宽和系统容错性。Lu 等人针对分布式机器学习集群的流量特点为其专门设计了基于阵列波导光栅路由器（Arrayed Waveguide Grating Router, AWGR）的网络互联拓扑 Lotus[79]。如图 24 所示，Lotus 在每个子模块内部采用 Clos 架构提供丰富的链路资源，同时在不同子模块间也创建多条直连链路，从而缩短路径长度，并且提高对分带宽。阵列波导光栅路由器为无源光交换设备，并且可以在任意端口间实现快速无阻塞的通信，因此，使用阵列波导光栅路由器不但可以提高网络设备的转发能力，并且可以降低网络设备功耗。

总的来说，针对分布式机器学习训练的场景，对底层物理网络拓扑进行适配，可以极大地提高通信能力，减少数据流之间的带宽资源竞争，从而大幅降低通信时间。然而，该类方案涉及整个集群互联方式的大规模调整，灵活性较低，因此比较适用于构建新的训练集群。此外，现有网络拓扑优化工作均面向单训练任务专用集群进行设计，很难扩展到多训练任务共享集群。以 BiGraph 拓扑为例，当某个训练任务仅占用部分计算节点时，这些计算节点可能无法构成小规模的 BiGraph 拓扑，导致 HDRM 算法无法将逻辑连接和物理链路进行一一映射，从而使得网络拓扑优化无法达到预期效果。

7 优化方案总结与对比

表 4 综合对比了近年来研究人员所提出的分布式机器学习系统网络性能优化研究相关工作。对比的主要指标包括优化机制、训练加速效果、节点扩展性、对模型收敛性的影响以及是否需要更换硬件设备或者互联方式等。这些工作从多个层面对分布式机器学习系统的网络性能进行优化，不同机制之间各有优劣。

从训练加速效果来看，ASP 这一模型一致性协议将通信开销从模型训练的核心路径上移除，使得网络通信不会阻塞训练过程，加速效果非常好；模型压缩或并行方式优化等方案，有效地减少了各计算节点通信的数据量，而网内聚合方案则逐跳减少了网络中的流量，因此，这些方案具有非常好的加速效果；传输协议和通信库优化类方案提高了点到点通信性能，网络拓扑优化类方案提高了通信节点之间的互联带宽，这些方案的加速效果也很好；虽然不同参数同步架构的理论参数同步时间之间的差距主要来自于时延开销，但在实际中，负载均衡、多流竞争等都会影响不同参数同步架构的实际参数同步时间，总体来说，参数同步架构类方案的加速效果不如前面几种方

案；通信调度类方案的加速效果与训练模型的通

表 4 分布式机器学习网络性能优化方案的综合比较

方案类型	优化机制	训练加速效果	节点扩展性	是否影响模型收敛性	是否需要更换硬件设备或互联方式	相关工作
模型一致性	通过松弛模型一致性来缓解通信对计算造成的阻塞	高	高	BSP 无影响；SSP 和 ASP 有影响	否	[18][19][20][21][23][54][80][81]
参数同步架构	通过设计合理的参数同步架构以避免流量热点，甚至减少网络流量	中	中	中心化架构无影响；去中心化架构有影响	否	[25][27][28][29][30][31][32][33][34][35]
传输协议和通信库	从底层硬件、拥塞控制协议、应用层逻辑等多个层面优化单次传输的性能	高	中	否	RDMA 和 NVLink 等方案需要；其他方案不需要	[27][39][43][44][45][46][47]
网内聚合	借助可编程交换机或专用交换机将参数汇总操作卸载到网络中，从而减少网络流量	高	低	否	是	[48][49][50][51][53]
模型压缩	通过对参数/梯度进行稀疏化或量化处理来大幅减小通信数据量	高	高	是	否	[54][55][56][60][61][62][63][64][65][66][67]
通信调度	通过优先传输对计算阻塞程度较高的通信操作，以达到隐藏通信开销的目的	低	低	否	否	[6][9]**Error! Reference source not found.Error! Reference source not found.** [71]
并行方式	扩大并行策略的探索空间，利用更优的策略来提升并行效率，降低通信需求	高	中	否	否	[72][73][74][75][76][82] [83]
网络拓扑	通过设计对分带宽更高的网络互联拓扑，提高节点间通信能力，减少通信耗时	高	中	否	是	[77][78][79][84]

信/计算比高度相关，相比其他方案来说，加速效果比较有限。

从节点扩展性来看，网内聚合类方案受限于交换机硬件计算能力和存储空间限制，通常应用于单机架规模的训练集群，扩展性较差；随着计算节点数量的增多，通信/计算比越来越高，导致通信调度类方案在节点数量较多时的扩展性较差；由于交换机端口数量、布线难度等因素的限制，底层物理网络拓扑的规模往往不能无限增大，如 BCube 适用于集装箱规模的数据中心，故网络拓扑类方案的扩展性一般；虽然传输协议和通信库类方案可将通信性

能提高数倍，暂缓网络瓶颈出现的时间，但随着节点数量的增多，通信操作又将成为系统瓶颈，故该

类方案的扩展性也一般；并行方式优化类方案的模型并行粒度不能无限切分，因此在节点规模很大时，仍会出现大量节点使用数据并行的情况，并且对大量节点求解最优并行方式的算法复杂度也非常高，如 PipeDream 的求解时间与计算节点数的二次方和模型层数的三次方成正比，以上因素导致并行方式优化类方案的扩展性也一般；节点规模很大时，环规约架构的通信时间被时延开销所主导，参数服务器架构的连接数量也会大大增加，从而导致传输性能的降低，故参数同步架构类方案的扩展性一般；模型一致性协议从核心路径上移除了通信，模型压缩类方案可将通信量降低数十乃至上百倍，因此这两类方案的扩展性较高。

从对模型收敛性的影响来看，模型压缩会导致参数同步时信息量的丢失，从而影响模型收敛性；SSP 和 ASP 以及去中心化参数同步架构引入了陈旧参数对全局模型的更新，也会对模型收敛性产生一定影响；其他类方案不涉及通信内容的改变，故不影响模型收敛性；

从对硬件的依赖性来看，RDMA 和 NVLink 需要专用的硬件设备，故依赖于硬件设备的更新升级；网内聚合类方案依赖于可编程交换机或专用交换机来实现在网络内部对参数进行聚合的目的，故该类方案也需要底层硬件设备的支持；网络拓扑类方案涉及对整个集群互联方式的修改，比较适用于新训练集群的搭建，在现有集群上的部署难度较大；其他类方案均为软件层方案，对底层硬件环境无特殊要求，因此部署难度较低，具有非常好的通用性。

2016 年，Google 提出了一种分布式机器学习新形式—联邦学习[85]。 本质上，联邦学习是一种加密的分布式机器学习框架，允许各参与方在不共享本地数据的条件下与其他各方共建模型。不同于传统分布式机器学习，联邦学习面临四个新的问题：客户端中数据非独立同分布问题、差分隐私问题、通信开销问题和客户端无状态问题。本文仅关注联邦学习中的通信开销问题。在联邦学习中，通信开销远大于计算开销，这主要是由于客户端与中央服务器之间的网络带宽有限，且连接质量较差，同时不同客户端的连接质量参差不齐造成的。

虽然联邦学习是一种分布式机器学习框架，但有些针对传统分布式机器学习的网络性能优化方案却不适用于联邦学习。例如，客户端可能通过无线方式接入网络，故无法对这些客户端之间的互联方式进行改善；一般来说，联邦学习的网络瓶颈点基本在客户端侧，因此，网内聚合的方式不能解决联邦学习场景下的网络传输痛点；联邦学习中各客户端均要使用本地数据进行训练，并且不会将本地数据传输给其他客户端，因此联邦学习只能使用数据并行方式，无法通过并行方式优化的方式来提高训练速度。

联邦学习场景下的网络性能优化主要依赖对通信内容的压缩来实现。一般来说，客户端上行链路的带宽比下行链路带宽更小，因此，一些工作[85-87]最早尝试通过多种梯度压缩方式，如量化、稀疏化、下采样、矩阵分解、计数草图（count sketch）和周期平均等，来减小客户端的上行通信压力。随后，一些工作[88]通过压缩参数的方式降低下行通信成本。文献[88]采用 Federated Dropout 的方式对神经元进行随机丢弃，这样客户端可以只训练一个更小的子模型，从而既减小了中央服务器到客户端的通信数据量，又能更加高效地完成本地计算。

虽然模型压缩会减缓模型的收敛速度，但受限于网络连接质量，联邦学习不得不通过压缩通信内容的方式来降低通信成本，提高训练速度。相比之下，传统分布式机器学习训练集群的互联带宽非常高，并且连接可靠性极高，因此，模型压缩在传统分布式机器学习训练中往往作为可选方案，需要充分权衡收敛性和训练速度来决定是否需要对模型进行压缩，以及使用何种压缩方式和压缩比例。

8 研究趋势展望

分布式机器学习系统性能优化作为分布式机器学习领域最为热门的研究方向之一，正在吸引越来越多学术界和工业界研究人员的关注。由于分布式机器学习系统网络性能优化研究与工业界结合紧密，具有重要的实践价值，可以预计在未来数年内相关研究还将持续成为焦点。

当前，国内学术界和工业界关于分布式机器学习系统网络性能优化的研究基本与国际水平处于并跑状态。因此，在国家大力发展新基建的背景下，加强分布式机器学习系统网络性能优化研究，不但能够为人工智能的发展提供内生动力，并且可以为依托人工智能实现外部赋能创造条件，对于推动传

统行业信息化、数字化、智能化转型升级具有非常重要的意义。

从网络通信的角度看，我们认为未来的分布式机器学习系统性能优化研究主要包括以下四个方向：

（1）模型的高质量压缩。分布式机器学习训练的通信数据量对通信耗时具有决定性影响。从机器学习算法的发展趋势来看，越来越大的机器学习模型已经成为必然[16][89]。因此，如何对训练超大模型时的通信数据进行高质量的压缩，既能大幅降低通信数据量，又不会造成训练信息的大量丢失，是未来缓解甚至彻底消除网络瓶颈的重要方向。当前，模型压缩程度仍然受到相关理论发展的限制，通信数据的压缩是以更多的通信次数为代价的。除相关压缩理论的突破外，未来可能的发展方向还包括细粒度的模型压缩方式，如不同层乃至不同算子采用不同的压缩方式、压缩比例，不同训练轮数采用不同的压缩方式、压缩比例，从而避免最差压缩比例限制整体的压缩效果。另一个可能的方向是综合考虑时空相关性的模型压缩方式，当前的压缩算法大多将每个参数值作为单独个体来处理，部分算法引入时间序列相关性以将相邻两轮训练间的结果相关联，从而降低随时间累积的压缩误差。然而，参数张量的空间相关性尚未得到充分重视。视频压缩领域中，基于时空相关性的视频帧间压缩方法已得到广泛应用。因此，模型压缩可以借鉴视频压缩领域的相关经验，综合考虑参数张量的时空相关性对模型参数采取进一步的有效压缩。

（2）并行方式优化。除压缩通信内容外，改进多节点之间的并行训练方式也是降低通信开销的重要途径。分布式机器学习训练通过将训练数据、训练模型分布到多个计算节点来达到并行训练的目的。即使对同一训练模型和相同训练数据而言，不同的并行方式也会产生完全不同的流量模式和通信数据量。现有方案大多在训练数据、模型不同层等维度对训练任务进行并行化分解，最近一些工作又引入了流水并行来提高计算资源利用率。但这些方案仍远未成熟。对流水并行来说，由于层间计算依赖关系的存在，这些方案或者无法完全消除“气泡”，或者需要占用大量显存来存储多个模型版本。如何提高流水并行的效率，同时最小化硬件资源占用，对流水并行的应用前景至关重要。对模型并行来说，更细粒度的操作符拆分，使其能够并行化计算，从而提高单个操作符的执行速度上限，也是未来值得探索的重要方向。另外，在大规模分布式机器学习训练场景下，如何快速求解最优并行化方式，将大量的计算节点合理地进行编排，也是该类方案将来能否得到广泛应用的重要基础。

（3）多任务场景下的网络资源复用。现有的网络性能优化方案仍主要针对单任务场景而设计，对多任务之间的联合优化方案仍有待研究。但在实际训练场景中，计算设备往往被单一训练任务所独占，但网络设备却被很多训练任务共享，导致不同训练任务由于彼此竞争网络资源造成性能的互相影响。对于分布式机器学习训练任务来说，流量具有明显的周期特征，即，从宏观结构来看，平均流量并不高，但缩放到毫秒粒度，则会出现链路利用率在满载和空载之间频繁切换的情况。当多个训练任务同时使用网络资源时，所有训练任务的通信时间都会被拉长，导致训练速度的下降。因此，未来一个可能的发展方向便是通过使不同训练任务分时复用网络资源，尽量减小每个训练任务所花费的通信时间，从而提升整体的训练速度。

（4）专用网络设备和架构。网络硬件技术的提升，对于分布式机器学习系统性能的提升具有显著的效果。当前，分布式机器学习训练任务和其他业务一样运行在通用网络硬件设备之上。但是分布式机器学习训练任务具有自己的特点，如流量矩阵的确定性以及数据传输的周期性等。因此，针对分布式机器学习业务设计专用的网络设备和架构，如超低转发时延交换机、GPU 与网卡的一体化设计等，也将成为未来的研究热点。此外，光电互联技术的出现也使得数据中心网络的带宽和容量大幅提高。可以预料，未来在如何合理地利用这些新型网络设备来提升网络传输性能方面也将会产生更多的研究成果。

9 总结

当前，以机器学习为代表的人工智能技术蓬勃发展，但是随着训练数据量的不断增长和机器学习模型的日趋复杂，对算力的要求也越来越高。由于单一计算设备的性能限制，单机训练的时效性非常差，因此，分布式机器学习成为加速人工智能发展的重要基石。然而，在大规模分布式机器学习训练场景下，计算节点之间的数据通信开销非常高，并且通信时间的增长会导致计算资源的浪费，降低分布式机器学习系统的扩展性。

本文分析了网络通信之所以会成为分布式机器学习系统扩展性瓶颈的主要原因，并提出了三种关键思路用以缓解甚至消除网络性能对分布式机器学习系统的限制，然后以这三种思路为基础，全面概述了当前的分布式机器学习系统网络性能优化工作，并通过归纳总结的方式从多个角度对比了不同方案的性能。在此基础上，对分布式机器学习系统网络性能优化研究未来的发展趋势进行了展望，以期为后续的相关研究提供重要参考。

致 谢 衷心感谢评审专家和编辑对本文提出的宝贵意见和建议！

参 考 文 献

[1] Qiao S, Zhang Z, Shen W, et al. Gradually updated neural networks for large-scale image recognition//Proceedings of the International Conference on Machine Learning (ICML). Stockholm, Sweden, 2018: 4188-4197.

[2] Xiong W, Wu L, Alleva F, et al. The Microsoft 2017 conversational speech recognition system//Proceedings of the IEEE international conference on acoustics, speech and signal processing (ICASSP). Calgary, Canada, 2018: 5934-5938.

[3] He D, Lu H, Xia Y, et al. Decoding with value networks for neural machine translation//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Long Beach, USA, 2017: 178-187.

[4] Bojarski M, Del Testa D, Dworakowski D, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

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

[6] Keskar N S, Mudigere D, Nocedal J, et al. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836, 2016.

[7] Geng J, Li D, Wang S. ElasticPipe: an efficient and dynamic model-parallel solution to DNN training//Proceedings of the 10th Workshop on Scientific Cloud Computing (ScienceCloud). Phoenix, USA, 2019: 5-9.

[8] Peng Y, Zhu Y, Chen Y, et al. A generic communication scheduler for distributed dnn training acceleration//Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). Huntsville, Canada, 2019: 16-29.

[9] Wang S, Li D, Geng J. Geryon: Accelerating distributed cnn training by network-level flow scheduling//Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). Virtual Conference, 2020: 1678-1687.

[10] Tang Z, Shi S, Chu X, et al. Communication-efficient distributed deep learning: A comprehensive survey. arXiv preprint arXiv:2003.06307, 2020.

[11] Zhang Z, Chang C, Lin H, et al. Is network the bottleneck of distributed training?//Proceedings of the Workshop on Network Meets AI & ML. New York, USA, 2020: 8-13.

[12] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition//Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). Las Vegas, USA, 2016: 770-778.

[13] Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning//Proceedings of the USENIX symposium on operating systems design and implementation (OSDI). Savannah, USA, 2016: 265-283.

[14] Paszke A, Gross S, Massa F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Vancouver, Canada, 2019: 8026-8037.

[15] Chen T, Li M, Li Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274, 2015.

[16] Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.

[17] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Nevada, USA, 2012: 1097-1105.

[18] Valiant L G. A bridging model for parallel computation. Communications of the ACM, 1990, 33(8): 103-111.

[19] Recht B, Re C, Wright S, et al. Hogwild: A lock-free approach to parallelizing stochastic gradient descent//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Granada, Spain, 2011: 693-701.

[20] Ho Q, Cipar J, Cui H, et al. More effective distributed ml via a stale synchronous parallel parameter server//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Nevada, USA, 2013: 1223-1231.

[21] Xing E P, Ho Q, Xie P, et al. Strategies and principles of distributed machine learning on big data[J]. Engineering, 2016, 2(2): 179-195.

[22] Cui H, Zhang H, Ganger G R, et al. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server//Proceedings of the Eleventh European Conference on Computer Systems (EuroSys). London, UK, 2016: 1-16.

[23] Chen J, Pan X, Monga R, et al. Revisiting distributed synchronous SGD[J]. arXiv preprint arXiv:1604.00981, 2016.

[24] Xing E P, Ho Q, Dai W, et al. Petuum: A New Platform for Distributed Machine Learning on Big Data//Proceedings of the ACM

SIGKDD Conference on Knowledge Discovery and Data Mining (KDD). Sydney, Australia, 2015: 1335-1344.

[25] Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI). Broomfield, USA, 2014: 583-598.

[26] Patarasuk P, Yuan X. Bandwidth optimal all-reduce algorithms for clusters of workstations. Journal of Parallel and Distributed Computing, 2009, 69(2): 117-124.

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

[28] Geng J, Li D, Cheng Y, et al. HiPS: Hierarchical parameter synchronization in large-scale distributed machine learning//Proceedings of the Workshop on Network Meets AI & ML. Budapest, Hungary, 2018: 1-7.

[29] Mikami H, Suganuma H, Tanaka Y, et al. Massively distributed SGD: ImageNet/ResNet-50 training in a flash. arXiv preprint arXiv:1811.05233, 2018.

[30] Wang S, Li D, Cheng Y, et al. Bml: A high-performance, low-cost gradient synchronization algorithm for dml training//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Montréal, Canada, 2018: 4238-4248.

[31] Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. The International Journal of High Performance Computing Applications, 2005, 19(1): 49-66.

[32] Dong J, Cao Z, Zhang T, et al. EFLOPS: Algorithm and System Co-Design for a High Performance Distributed Training Platform// Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA). San Diego, USA, 2020: 610-622.

[33] Omidshafiei S, Pazis J, Amato C, et al. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. arXiv preprint arXiv:1703.06182, 2017.

[34] Lian X, Zhang C, Zhang H, et al. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS). Long Beach, USA, 2017: 5330-5340.

[35] Colin I, Bellet A, Salmon J, et al. Gossip dual averaging for decentralized optimization of pairwise functions. arXiv preprint arXiv:1606.02421, 2016.

[36] Lan G, Lee S, Zhou Y. Communication-efficient algorithms for decentralized and stochastic optimization. Mathematical Programming, 2020, 180(1): 237-284.

[37] Shi W, Ling Q, Wu G, et al. A proximal gradient algorithm for decentralized composite optimization. IEEE Transactions on Signal Processing, 2015, 63(22): 6013-6023.

[38] Alizadeh M, Kabbani A, Edsall T, et al. Less is more: trading a little bandwidth for ultra-low latency in the data center//Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI). Lombard, USA, 2012: 253-266.

[39] Alizadeh M, Greenberg A, Maltz D A, et al. Data center tcp (dctcp)//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM). New Delhi, India, 2010: 63-74.

[40] Hu S, Bai W, Zeng G, et al. Aeolus: A Building Block for Proactive Transport in Datacenters//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM). New York, USA, 2020: 422-434.

[41] Bai W, Hu S, Chen K, et al. One More Config is Enough: Saving (DC) TCP for High-speed Extremely Shallow-buffered Datacenters//Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). Virtual Conference, 2020: 2007-2016.

[42] Wu H, Feng Z, Guo C, et al. ICTCP: Incast congestion control for TCP in data-center networks. IEEE/ACM transactions on networking, 2012, 21(2): 345-358.

[43] Xia J, Zeng G, Zhang J, et al. Rethinking transport layer design for distributed machine learning//Proceedings of the 3rd Asia-Pacific Workshop on Networking (APNet). Beijing, China, 2019: 22-28.

[44] Yi B, Xia J, Chen L, et al. Towards zero copy dataflows using rdma//Proceedings of the ACM SIGCOMM Posters and Demos. Los Angeles, USA, 2017: 28-30.

[45] Shi S, Chu X, Li B. MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms//Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). Paris, France, 2019: 172-180.

[46] Laanait N, Romero J, Yin J, et al. Exascale deep learning for scientific inverse problems. arXiv preprint arXiv:1909.11150, 2019.

[47] Wang G, Venkataraman S, Phanishayee A, et al. Blink: Fast and generic collectives for distributed ml//Proceedings of Machine Learning and Systems (MLSys), Austin, USA, 2020: 172-186.

[48] Sapio A, Abdelaziz I, Aldilaijan A, et al. In-network computation is a dumb idea whose time has come//Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets). Palo Alto, USA, 2017: 150-156.

[49] Luo L, Liu M, Nelson J, et al. Motivating in-network aggregation for distributed deep neural network training//Workshop on Approximate Computing Across the Stack (WAX). Xi'an, China, 2017.

[50] Sapio A, Canini M, Ho C Y, et al. Scaling distributed machine learning with in-network aggregation. arXiv preprint arXiv:1903.06701, 2019.

[51] Mai L, Hong C, Costa P. Optimizing network performance in distributed machine learning//Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud). Denver, USA, 2015: 1-2.

[52] Kim D, Liu Z, Zhu Y, et al. Tea: Enabling state-intensive network functions on programmable switches//Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM). New York, USA, 2020: 90-106.

[53] Luo L, Nelson J, Ceze L, et al. Parameter hub: a rack-scale parameter server for distributed deep neural network training//Proceedings of the ACM Symposium on Cloud Computing (SoCC). Carlsbad, USA, 2018: 41-54.

[54] De Sa C M, Zhang C, Olukotun K, et al. Taming the wild: A unified analysis of hogwild-style algorithms//Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Montreal, Canada, 2015: 2674-2682.

[55] Seide F, Fu H, Droppo J, et al. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns//Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH). Singapore, 2014: 1058-1062.

[56] Alistarh D, Li J, Tomioka R, et al. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. arXiv preprint arXiv:1610.02132, 2016.

[57] Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database//Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). Miami, USA, 2009: 248-255.

[58] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Toronto: University of Toronto, 2009.

[59] Strom N. Scalable distributed DNN training using commodity GPU cloud computing//Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH). Dresden, Germany, 2015: 1488-1492.

[60] Langford J, Li L, Zhang T. Sparse Online Learning via Truncated Gradient. Journal of Machine Learning Research, 2009, 10(3): 777-801.

[61] Stich S U, Cordonnier J B, Jaggi M. Sparsified SGD with memory. arXiv preprint arXiv:1809.07599, 2018.

[62] Alistarh D, Hoefler T, Johansson M, et al. The convergence of sparsified gradient methods. arXiv preprint arXiv:1809.10505, 2018.

[63] Ba J L, Kiros J R, Hinton G E. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[64] Dryden N, Moon T, Jacobs S A, et al. Communication quantization for data-parallel training of deep neural networks//Proceedings of the 2nd Workshop on Machine Learning in HPC Environments (MLHPC). Salt Lake City, USA, 2016: 1-8.

[65] Aji A F, Heafield K. Sparse communication for distributed gradient descent. arXiv preprint arXiv:1704.05021, 2017.

[66] Shi S, Wang Q, Zhao K, et al. A distributed synchronous SGD algorithm with global Top-k Sparsification for low bandwidth networks//Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS). Dallas, USA, 2019: 2238-2247.

[67] Sattler F, Wiedemann S, Müller K R, et al. Sparse binary compression: Towards distributed deep learning with minimal communication//Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN). Budapest, Hungary, 2019: 1-8.

[68] Zhang H, Zheng Z, Xu S, et al. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters//Proceedings of the USENIX Annual Technical Conference (ATC). Santa Clara, USA, 2017: 181-193.

[69] Hadi Hashemi S, Abdu Jyothi S, Campbell R H. TicTac: Accelerating Distributed Deep Learning with Communication Scheduling. arXiv preprint arXiv: 1803.03288, 2018.

[70] Jayarajan A, Wei J, Gibson G, et al. Priority-based parameter propagation for distributed DNN training. arXiv preprint arXiv: 1905.03960, 2019.

[71] Wang S, Li D, Zhang J, et al. CEFS: compute-efficient flow scheduling for iterative synchronous applications//Proceedings of the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT). Barcelona, Spain, 2020: 136-148.

[72] Wu X, Xu H, Li B, et al. Stanza: Layer separation for distributed training in deep learning. IEEE Transactions on Services Computing, 2020: 1-12.

[73] Geng J, Li D, Wang S. Horizontal or vertical? a hybrid approach to large-scale distributed machine learning//Proceedings of the 10th Workshop on Scientific Cloud Computing (ScienceCloud). Phoenix, USA, 2019: 1-4.

[74] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: generalized pipeline parallelism for DNN training//Proceedings of the ACM Symposium on Operating Systems Principles (SOSP). Huntsville, Canada, 2019: 1-15.

[75] Huang Y, Cheng Y, Bapna A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1811.06965, 2018.

[76] Geng J, Li D, Wang S. Fela: Incorporating Flexible Parallelism and Elastic Tuning to Accelerate Large-Scale DML//Proceedings of the IEEE International Conference on Data Engineering (ICDE). Dallas, USA, 2020: 1393-1404.

[77] Wang S, Li D, Geng J, et al. Impact of network topology on the performance of DML: Theoretical analysis and practical factors//Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). Paris, France, 2019: 1729-1737.

[78] Liu L, Jin Q, Wang D, et al. PSNet: Reconfigurable network topology design for accelerating parameter server architecture based distributed machine learning. Future Generation Computer Systems, 2020, 106: 320-332.

[79] Lu Y, Gu H, Yu X, et al. Lotus: A New Topology for Large-scale Distributed Machine Learning. ACM Journal on Emerging Technologies in Computing Systems, 2020, 17(1): 1-21.

[80] Lian X, Huang Y, Li Y, et al. Asynchronous parallel stochastic gra-

dient for nonconvex optimization. arXiv preprint arXiv:1506.08272, 2015.

[81] Zhang W, Gupta S, Lian X, et al. Staleness-aware async-sgd for distributed deep learning. arXiv preprint arXiv:1511.05950, 2015.

[82] Jia Z, Zaharia M, Aiken A. Beyond data and model parallelism for deep neural networks. arXiv preprint arXiv:1807.05358, 2018.

[83] Fan S, Rong Y, Meng C, et al. DAPPLE: A Pipelined Data Parallel Approach for Training Large Models//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). Virtual Conference, 2021: 431–445.

[84] Deng Y, Guo M, Ramos A F, et al. Optimal low-latency network topologies for cluster performance enhancement. The Journal of Supercomputing, 2020, 76(12): 9558-9584.

[85] Konečný J, McMahan H B, Yu F X, et al. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492, 2016.

[86] Rothchild D, Panda A, Ullah E, et al. Fetchsgd: Communication-efficient federated learning with sketching//International Conference on Machine Learning (ICML). Virtual Conference, 2020: 8253-8265.

[87] Reisizadeh A, Mokhtari A, Hassani H, et al. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization//International Conference on Artificial Intelligence and Statistics (AISTATS). Virtual Conference, 2020: 2021-2031.

[88] Caldas S, Konečny J, McMahan H B, et al. Expanding the reach of federated learning by reducing client resource requirements. arXiv preprint arXiv:1812.07210, 2018.

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

WANG Shuai, Ph.D. candidate. His research interest includes data center network, network performance optimization of distributed machine learning system.

LI Dan, Ph.D., professor, Ph.D. supervisor. His research interest includes Internet architecture, data center network, data-driven network.

Background

With the great success made by machine learning in a wide range of applications, distributed machine learning training has become an increasingly important workload in data centers recently. Distributed machine learning makes it possible to train complex machine learning models with big data in an acceptable time by distributing a training job to multiple computing nodes. Unfortunately, distributed machine learning comes with a price. The high volume of communication among computing nodes can make the scalability of a distributed machine learning system very poor. There have been a large body of research work on performance optimization of distributed machine learning system both in academia and in industry. This paper conducts a survey study on the research progress of on the network performance optimization of distributed machine learning system, including parameter synchronization algorithms optimization, communication efficiency improvement, parallelism architecture and network topology design, etc. Then a comparison of these proposed schemes is made. Finally, the future research trends in this area are discussed.

The work was supported by the National Key Research and Development Program of China (2018YFB1800500), the Research and Development Program in Key Areas of Guangdong Province (2018B010113001), the National Science Foundation of China (61772305) and Tsinghua University-China Mobile Communications Group Co.,Ltd. Joint Institute.