SlimGC：面向分布式深度学习的梯度压缩优化策略

白 哲 于恩达 董德尊
（国防科技大学计算机学院 长沙 410073）

摘 要 梯度压缩是缓解分布式深度学习中通信瓶颈的关键技术。然而，通过梯度压缩实现显著的性能改进仍然具有挑战性。在实际应用中，梯度压缩面临着以下几个挑战：(1)不能有效优化小规模张量通信的启动开销；(2)压缩操作可能会与张量计算竞争GPU资源，从而延迟梯度传输的启动时机；(3)它可能引发需要谨慎处理的模型精度问题。为了最大限度地发挥梯度压缩的优势并应对这些挑战，本文设计了SlimGC策略来用于通用梯度压缩增强。此外，为了避免对GPU算力和内存资源的争用，SlimGC将压缩操作卸载给CPU，并采用模型备份技术，该技术解除了工作节点间对模型参数的读取依赖，从而隐藏CPU压缩成本和部分通信开销。本文的实验是在一个拥有16个V100 GPU的集群上进行的。实验评估表明，对于典型的分布式深度学习训练任务，SlimGC将1bit和2bit压缩算法的训练吞吐量分别最高提高了74.3%和75.9%。此外，它实现了1.1%～2.3%的收敛精度提高，并减少了10.3%的GPU内存消耗。

关键词 分布式深度学习；梯度压缩；分布式通信优化；压缩卸载；内存消耗
中图法分类号 TP18 **DOI号** 10.11897/SP.J.1016.2025.01168

SlimGC：Gradient Compression Optimization Strategy For Distributed Deep Learning

BAI Zhe YU En-Da DONG De-Zun
(College of Computer Science and Technology, National University of Defense Technology, Changsha 410073)

Abstract Training deep learning models is a primary task in large-scale systems. In recent years, with the dramatic increase in model parameter counts and dataset sizes, the number of nodes participating in parallel training has grown significantly, leading to escalating communication overhead between nodes. Consequently, inter-node communication costs have gradually become a performance bottleneck in distributed training. Gradient compression techniques address this by mapping gradients from floating-point formats to low-bit representations, reducing data transmission overhead and improving training efficiency. These techniques have emerged as a key solution to alleviate communication bottlenecks in distributed deep learning. However, achieving significant performance improvements through gradient compression remains challenging. In practical applications, gradient compression faces the following issues: (1)ineffective optimization of startup overhead for small-scale tensor communication, (2)competition between compression operations and tensor computations for GPU resources, which delays the initiation of gradient transmission, and (3)potential model accuracy degradation requiring careful handling. To

收稿日期：2024-09-05；在线发布日期：2025-03-06。本课题得到国家重点研发计划项目(No. 2022YFB4501702)、国家自然科学基金(U24B20151)资助。**白 哲**，硕士研究生，主要研究领域为分布式训练和通信优化。E-mail：baizheo3o@nudt.edu.cn。**于恩达**，博士研究生，主要研究领域为分布式训练和通信优化。E-mail：yuenda@nudt.edu.cn。**董德尊(通信作者)**，博士，研究员，博士生导师，中国计算机学会(CCF)杰出会员，主要研究领域为高性能网络与架构、数据中心网络、深度学习。E-mail：dong@nudt.edu.cn。

maximize the benefits of gradient compression and address these challenges, we designed the SlimGC strategy for universal gradient compression enhancement to accelerate distributed deep learning training. First, SlimGC employs a compression optimization strategy that constructs a communication threshold table to dynamically determine optimal compression thresholds during training. This strategy exhibits strong adaptability and generalization, enabling real-time adjustments to diverse network environments while maintaining training accuracy and performance. Second, SlimGC adopts a compression offloading strategy that shifts compression-related computations from GPUs to CPUs. This avoids frequent context switching between model computations and gradient compression tasks on GPUs, thereby reducing resource contention and inefficiency. Finally, SlimGC implements a model backup strategy that maintains an auxiliary backup model to achieve high overlap between computation and communication. This technique eliminates read dependencies on model parameters across worker nodes, effectively hiding CPU compression costs and partial communication overhead. The experiments in this study were conducted on a cluster equipped with 16 V100 GPUs, testing typical image recognition and natural language processing models, including evaluations on the CIFAR-10, ImageNet ILSVRC2012, and GLUE SST2 datasets. Experimental results demonstrate the following: (1) Accuracy: SlimGC improves the accuracy of image recognition models by at least 1.1%, and increases the F1 score of the BERT-Base model by at least 0.4%. (2) Parameter Sensitivity: Without requiring hyperparameter tuning for optimality, SlimGC limits the accuracy loss compared to the highest achievable accuracy to within 0.6%. (3) Performance: Under a 100 Gbps network environment, SlimGC enhances the training speed of image recognition and natural language processing models by up to 74.3% and 75.9%, respectively. When compared horizontally with THC, SlimGC achieves up to a 1.55× performance improvement. (4) Training composition, SlimGC reduces the communication cost of ResNet-152 by up to 35.6%, and validates that the model backup strategy effectively optimizes the shortcomings of the compression offloading strategy. (5) Memory Efficiency: SlimGC achieves up to 10.3% reduction in GPU memory consumption. These results highlight SlimGC's ability to balance performance, accuracy, and resource efficiency while maintaining adaptability to diverse network conditions. Furthermore, the findings are extensible to broader distributed computing scenarios, such as federated learning and edge intelligence, demonstrating its potential for real-world deployment.

Keywords　distributed deep learning; gradient compression; distributed communication optimization; compression offloading; memory consumption

1 引　言

训练深度学习模型是大规模系统的主要任务。这种训练可以以各种方式并行化[1-2],其中最主要和最简单的形式是数据并行化。在数据并行训练中,每个节点都复制一份相同的模型。在本地计算梯度后,所有节点都需要将梯度聚合来更新模型参数。近年来,随着模型参数数量和数据集规模的急剧增加,参与并行训练的节点数量不断增加,节点间的通信开销逐渐增大。一方面,通信技术的发展速度远远落后于计算设备的进步,另一方面节点之间更频繁的梯度同步又加剧了网络的压力[3]。节点间的通信开销因此成为了分布式训练的性能瓶颈。为了打破这一瓶颈,研究者们进行了不同方向的探索和创新,包括流水线[4-7],基于优先级的调度[8-11],以及集体通信算法的优化[12-15]。这些工作的主要方向是并行计算和通信操作,以最大限度地提高通信重叠。然而,在大多数训练场景中,通信开销远远超过了计算开销,严重削弱了并行化方法带来的优势。例如,在一个由 16 个 NVIDIA V100 GPU 组成的集群上,使用 100Gbps 以太网进行 ResNet-50 的训练

时,通信时间超过计算时间 20 倍以上。

近些年来有很多方法致力于减少分布式训练的通信开销。在这些方法中,梯度量化[16-20]通过将浮点数的梯度编码为低比特数据表示来减少传输量;梯度稀疏化[21-25]主张每个节点在每次迭代中只传输前 0.1%大小的梯度。梯度量化算法有望缓解通信瓶颈;然而,它们也存在一些明显的局限性。首先,在某些模型的训练过程中会产生大量的小尺度张量通信(图 1)。一方面,与小尺度梯度相关的通信开销主要出现在通信启动阶段,因此压缩小尺度梯度并不一定能提高整体性能(图 2)。另一方面,这些小规模梯度在模型更新过程中可能起到关键作用,如果延迟或遗漏它们的传输,可能会降低模型的收敛精度。其次,梯度压缩方法带来的训练速度优势是有限的,因为压缩操作不仅占用 GPU 资源,而且由于额外的压缩开销,还会延长每个节点传输梯度的时间[26-27]。最后,压缩方法的有效性在很大程度上依赖于梯度压缩的压缩阈值设置[28]。如果梯度分布偏离了这一阈值,可能会导致模型收敛精度显著下降或对性能产生不利影响。

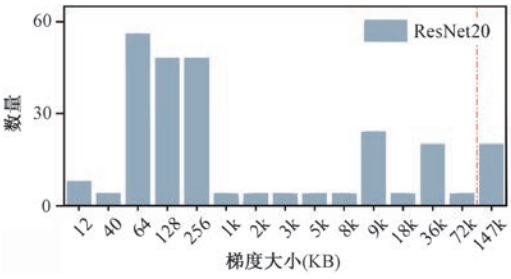


图 1 ResNet20 模型训练迭代中的梯度大小分布

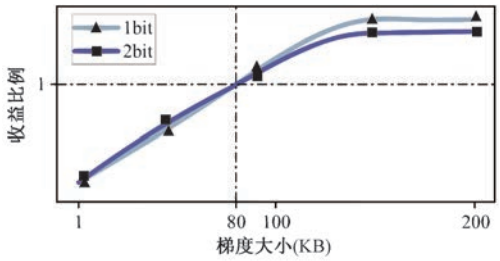


图 2 使用 GPU 执行不同压缩算法的压缩效益比

为了解决上述问题,我们设计出 SlimGC 策略,旨在面向分布式深度学习训练实现通用梯度压缩增强。针对小尺度梯度引起的通信问题,SlimGC 在训练初期通过多轮迭代来评估压缩效益比,并据此确定最优的压缩收益范围,进而指导哪些梯度需要被压缩。此外,SlimGC 将梯度压缩操作卸载到 CPU 上,以解决压缩操作占用 GPU 资源的问题。鉴于这种方法可能会延长每个节点传输梯度的时间,SlimGC 设计了模型备份策略,使节点能够在每次迭代中本地备份全局权重,以此打破对全局权重的读取依赖,并允许后续迭代的计算能够无需等待正在进行的通信阶段。SlimGC 采用了参数服务器(Parameter Server, PS)架构,该架构可以在 GPU/CPU 混合集群中将部分参数聚合任务分配给 CPU,更加充分地利用系统资源。此外,当我们为每个工作节点都配置一个相应的服务器节点时,它本质上就等效 AllReduce 通信[29]。在 16 个 GPU 集群上的实验评估表明,对于典型的 DDL 训练任务,SlimGC 将 1bit 和 2bit 压缩算法的训练吞吐量分别提高了 74.3%和 68.7%。此外,它将收敛精度提高了 1.1% 至 2.3%,并将 GPU 内存消耗降低了 10.3%。综上所述,我们的贡献如下:

(1)设计提出 SlimGC,它能够在无需定制化设计的情况下,在算法级以及系统级两个层面优化分布式训练通信,而且可以普遍提升流行梯度量化算法的收敛精度和训练速度。这里的不需要定制化指的是,在训练不同的模型,使用不同的数据集的时候,SlimGC 不要求对训练框架和压缩方案进行改动而是作为插件的形式为现有压缩方法提供优化。SlimGC 通过覆盖 MXNet 与 PyTorch 对应的 KVStore 和 DDP 通信模块的方式,转而使用 SlimGC 优化后的通信模块。这样 SlimGC 可以简洁地兼容各种优化器与优化算法。此外,SlimGC 只需要按系统步骤训练,不需求额外的超参数,便可以享受到 SlimGC 带来的性能与精度的双重收益。

(2)算法级优化策略:针对当前流行梯度压缩方法对小尺度梯度通信处理的劣势,SlimGC 确定了处理小尺度梯度的最佳解决方案,通过自适应训练环境寻找最优小尺度张量范围,将小尺度张量的压缩通信替换为全精度通信,进而提高了小尺度张量的通信效率,并且帮助梯度压缩算法减少对精度校正方法的依赖。

(3)系统级优化策略一:针对当前流行压缩算法带来的额外计算开销对 GPU 资源的占用而导致训练性能下降的问题,SlimGC 设计出了压缩卸载策略,通过将压缩任务转移到 CPU,从而减轻 GPU 的计算和存储负担。

(4)系统级优化策略二:针对当前分布式计算数据并行模式中计算与通信重叠率过低的问题,SlimGC 设计出模型备份策略,以保证 SlimGC 可以

突破与压缩卸载相关的潜在性能瓶颈,并大幅度提高计算与通信重叠率。

本文的其余部分组织如下:第 2 节介绍了分布式训练的研究背景和相关工作;我们在第 3 节中提供了 SlimGC 的详细设计和实现;第 4 节通过实验结果证明了 SlimGC 为不同压缩算法提供的性能、精度优化效果;第 5 节是对本文的总结。

2　背景知识与相关工作

随着深度学习模型和任务规模的不断扩大,传统的单机训练方法已无法满足日益增长的算力需求。因此,跨多个节点实施的分布式深度学习(Distributed Deep Learning, DDL)成为一种主流的解决方案。本章详细阐述了 DDL 的训练流程,包括通信原理以及同步通信所固有的缺陷。为了有效减轻通信负担,我们进一步探讨了梯度压缩技术,同时指出了它在实际应用中遇到的问题和局限性。

2.1　DDL 中的同步通信

在 DDL 中,通信与计算之间存在着紧密的相互依赖关系。要深入理解 DDL 中的通信开销问题,关键在于清晰地把握训练过程中的参数传递机制。

目前,在 DDL 中,同步随机梯度下降(Symmetrical Stochastic Gradient Descent, S-SGD)及其衍生算法是最常用的模型训练方法。为了让读者更加直观地理解 S-SGD 的工作流程,我们以经典的参数服务器(PS)架构为例,通过图 3 展示了节点处理梯度和权重的详细过程。在 PS 架构中,节点根据其承担的任务被抽象为两种角色:工作节点和服务器节点。工作节点负责使用当前模型权重参数 w_i 对样本数据进行训练,生成本地梯度 $g_{i,j}^{loc}$,并将其发送到服务器,而服务器节点负责聚合来自所有工作节点的梯度后并开始更新全局权重(公式(1))。表 1 列出了本文中使用的符号。其中 η 为训练过程中使用的学习率。

$$w_{i+1}=w_i-\frac{\eta}{N}\sum_{j=1}^{N}g_{i,j}^{loc} \tag{1}$$

更新操作完成后,各工作节点从服务器中提取更新后的模型权重 w_{i+1},然后启动下一个迭代。S-SGD 通过训练周期迭代来细化模型,每次迭代中工作节点和服务器节点之间的参数和梯度传递过程构成了 DDL 中的通信。

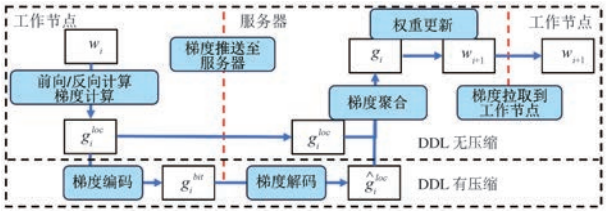


图 3　在分布式深度学习训练中,有无压缩情况下各节点的操作细节(红色虚线表示工作节点与服务器节点间的通信,带有箭头的蓝色曲线则指示参数间的读取或写入依赖关系。)

表 1　文中涉及的变量及其定义

符号	含　义
$w_{i,j}^{loc}$	第 j 个工作节点在第 i 次迭代的本地权重
w_i	服务器第 i 次迭代得到的全局权重
$g_{i,j}^{loc}$	第 j 个工作节点在第 i 次迭代的局部权重
$g_{i,j}^{bit}$	第 j 个工作节点在第 i 次迭代压缩后的局部梯度
$\hat{g}_i^{loc}$	服务器对 $g_{i,j}^{bit}$ 解码后得到的局部梯度
g_i	服务器第 i 次迭代聚合所有局部梯度得到的全局梯度
N	工作节点的个数
η	训练过程中的学习率
t_p	每次迭代的张量计算开销
t_c	每次迭代的未压缩通信时间
t_p^q	每次迭代的梯度压缩计算开销
t_c^q	每次迭代的压缩通信时间

注:为了文章的简洁,部分图或公式可能不需要区分工作节点,便会对符号精简处理。例如 $w_{i,j}^{loc}$ 会简写为 w_i^{loc}。

2.2　DDL 同步通信面临的挑战

DDL 通过多节点合作为模型训练提供强大支持,但也伴随着一些挑战。为避免训练过程陷入局部最优,各节点需同步其局部梯度,以实现模型参数的全局更新。然而,节点间的梯度同步通信引入了显著的通信成本,主要原因如下:(1)频繁的同步需求:在每一轮迭代中,节点间需要进行多次参数的同步通信;(2)庞大的数据同步量:每个节点在每次迭代中必须同步大量的数据;(3)网络带宽限制:随着参与同步的节点数量增加,所有节点生成的梯度数据需同时通过有限的物理链路传输,这在网络带宽上引发了激烈的资源争夺,导致同步通信的成本随着节点数量的增加而指数级上升。

此外,同步协议要求已完成梯度计算的节点等待那些速度较慢的节点,这种等待机制进一步加剧了同步通信的开销。因此,同步通信的延迟不仅成为制约训练速度的瓶颈,也影响了整体的计算效率。

2.3　梯度量化:优化 DDL 通信

正如上文说明的那样,通信开销常常成为分布

式训练过程中的主要瓶颈，尤其是在需要同步大量模型参数和梯度时。为了克服这一挑战，研究者们提出了多种创新方法，其中包括梯度量化技术。这种方法通过将梯度从浮点格式映射到低比特格式来减少数据传输的开销，从而提高训练效率。梯度量化技术的核心是对梯度进行编码和解码操作，并设计补偿机制以减少压缩带来的精度损失。

2.3.1 编码与解码

梯度量化技术的核心机制涉及梯度的编码与解码，通常也被称作梯度压缩与解压缩。在具体的实现过程中，编码步骤在工作节点发送梯度之前执行，而解码步骤则在服务器端对梯度进行聚合之前进行。尽管各种梯度量化技术在梯度压缩的程度上存在差异，但它们的编码和解码流程大体相似。以1bit[16]梯度量化方法为例，该方法通过下面的梯度量化函数来对梯度进行编解码：

$$Q(g)=\begin{cases}0 & g\geqslant 0\\ 1 & g<0\end{cases} \tag{2}$$

$$Q^{-1}(\tilde{g})=\begin{cases}0 & \tilde{g}\geqslant 0\\ 1 & \tilde{g}<0\end{cases} \tag{3}$$

其中，Q 代表着编码方式，其目的是将 32 位浮点数表示的梯度 g 编码为低比特表示的形式。而 Q^{-1} 则代表着对应的解码方式。梯度量化技术通过优化梯度的表示方式，有效降低了分布式训练中节点间传输梯度所需的数据量，从而很大程度上缓解了分布式深度学习中大规模梯度交换带来的问题。

2.3.2 误差补偿机制

在对梯度的编解码过程中可以发现，虽然这种方法极大减少了通信的开销，但对梯度的量化表示会导致模型的权重更新不准确，进而影响模型的迭代与收敛，最终造成精度的损失。为了尽可能减少梯度表示的损失，很多梯度量化方法都会设计一套独特的误差补偿机制。

在1bit[16]算法中，通过将编码前后梯度的误差进行记录，并用于下次的编码中来实现误差补偿：

$$\hat{g}_i^{loc}=Q(g_i^{loc}+Err_{i-1}) \tag{4}$$

$$Err_i=g_i^{loc}-Q^{-1}(\hat{g}_i^{loc}) \tag{5}$$

1bit 算法采用一种异步补偿的机制，将前一步梯度压缩带来的误差 Err_{i-1} 和梯度 g_i^{loc} 一起进行压缩，以此来减少压缩带来的损失。

而在 2bit 梯度量化方法中，为了确保量化过程中模型训练的收敛精度，通常会采用残差累积机制。在这个过程中，梯度与提前设定好的压缩阈值之间的差值会被累积到一个缓存中。当累积残差的绝对值超过阈值时，这些残差会被发送给服务器进行进一步处理。这种方法虽然有效地避免了信息的丢失，但仍然依赖于合理的超参数设置来保证训练的收敛精度。

在早期针对分布式通信优化设计的梯度量化方法中，编码解码策略和误差补偿机制相对基础。随着研究的深入，后续工作在这两个方面进行了显著的改进和优化，在通信量和梯度精度损失之间取得了更好的平衡，详细介绍见第 2.5 节。

2.4 梯度量化弊端的探究

梯度量化同样存在一些弊端。例如，它会增加 GPU 的内存压力和计算开销，因为量化过程中的额外编码和解码开销可能导致在实际模型训练中无法完全实现理论上的收益。

为了研究造成梯度量化方法实际效果不佳的具体原因，我们使用 MXNet-1.6 在由 100Gbps 以太网连接的 4 个节点上(每个节点 4 个 V100GPU)训练了不同的 DNN 模型并记录了各节点在模型训练过程中执行不同操作的开销(表 2)，并据此绘制了使用不同压缩策略的 DDL 示例(图 4)。为了突出主要问题，本文没有深入研究无等待反向传播机制引起的计算和通信之间的重叠[4,7]，因为这不会影响我们的研究结果。

表 2 模型训练时每个操作每轮的总时间消耗 (单位：秒)

模型	t_c	t_c^q	t_p	t_p^q
ResNet50	2 869.6	2 121.5	566.5	951.9
InceptionBn	1 502.9	1 256.4	371.8	563.5
AlexNet	42 108.0	31 389.4	2 566.3	4 466.8
VGG16	158 828.2	120 947.8	27 165.1	42 119.5

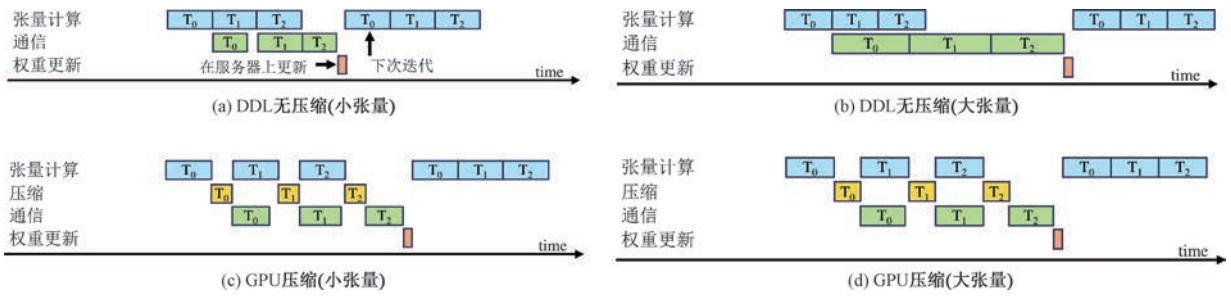


图 4 涉及大规模张量通信与小规模张量通信场景下，各种通信策略时间成本的概念性示意图

根据表 2 的实验结果可以看出:梯度量化的优点:梯度量化通过减少传输的数据量,可以在一定程度上降低通信成本,并最终缩短训练时间。这正是梯度量化设计的初衷,让量化后通信时间 t_c^q 低于原本通信时间 t_c 。根据表 2 的数据,t_c^q 相比于 t_c 节省了 25%到 35%的通信开销,随着分布式训练中参与节点数量的增加,节点间对网络资源的竞争加剧,梯度量化在通信开销优化方面的效果将更加显著。

梯度量化的缺点:(1)压缩操作会推迟通信阶段的开始时间,并且引入了不可忽视的额外开销。通过比较表 2 中的 t_p 和 t_p^q 两列,我们可以发现,压缩操作带来的计算开销有时甚至超过了单轮迭代的计算成本。更糟糕的是,为了快速完成压缩计算,压缩操作通常在 GPU 上执行,这不仅与训练计算争夺有限的 GPU 资源,而且也推迟了通信的启动时机。如图 4(b)与图 4(d)所示,由于对 GPU 资源的竞争,通信阶段(用绿色方块表示的 T_0,T_1,T_2)的启动被推迟,削弱了梯度量化的性能优势。(2)当前的压缩策略没有区分压缩负收益的梯度。如图 1 所示,在梯度尺度较小的情况下,压缩没有带来性能收益。通过对比图 4(a)和图 4(c),我们可以看到,在小尺度梯度的通信中,压缩策略处于性能劣势。

2.5　其他相关工作

2.5.1　梯度稀疏化

梯度稀疏化是一种在分布式深度学习训练中减少通信开销的技术。它通过选择性地关注梯度向量中的重要值,即那些数值较大的元素,来实现高效的数据压缩。这种方法的核心挑战在于如何有效地从梯度向量中选择出这些有效值,从而将稠密更新转换为稀疏更新。

DGC[30]要求工作节点在每次迭代中只交换前 0.1%梯度,并在缓冲区中积累额外的梯度,直到它们超过压缩阈值。为了防止精度显著下降,DGC 采用动量校正和动量因子掩蔽机制来调整权重更新时的动量方向。LGC[31]允许最快的节点将所有梯度发送到服务器,而其他节点只发送较大的梯度。这种方法解决了参与权重更新的小梯度滞后问题。然而,SIDCo[32]已经证明,大多数压缩方法依赖于合适的压缩阈值来保证精度,这需要用户拥有相关的先验知识或动态调整阈值。然而,AdaComp[33]、Top-k Allreduce[34]和 Redsync[35]等能够动态调整压缩阈值的方法会带来昂贵的阈值计算开销。

2.5.2　梯度量化

在前文第 2.3 节中已经详细介绍了梯度量化的主要概念和设计思路,本节将重点讨论相关研究工作以及它们存在的局限性。

Q-SGD[36]和 Terngrad[37]作为早期经典的梯度量化方法,其梯度编码方式要比 1bit[16]更为复杂且精准,而且实现了梯度的无偏量化。其中,Q-SGD 通过下面的方法进行梯度的编码操作:

$$Q(v_i)=\|v\|_2\cdot \mathrm{sgn}(v_i)\cdot \xi_i(v,s) \tag{6}$$

其中,v_i 代表着梯度 v 的第 i 个维度,$\mathrm{sgn}(x)$ 为符号函数,大于等于 0 时返回+1,小于 0 时返回−1。s 为提前设置的用于控制压缩精度的整数,如果 $s=5$,那么梯度就会被压缩为[0, 0.25, 0.5, 0.75, 1]这 5 个值(不考虑符号)。

$$\xi_i(v,s)=\begin{cases}\dfrac{l}{s}, & 1-p\left(\dfrac{|v_i|}{\|v\|_2},s\right)\\ \dfrac{l+1}{s}, & \text{others}\end{cases} \tag{7}$$

其中,$0<l\leqslant s$ 且 l 为整数使得 $\dfrac{|v_i|}{\|v\|_2}\in\left[\dfrac{l}{s},\dfrac{l+1}{s}\right]$。$p(a,s)=as-l, a\in[0,1]$,当 $v=0$,则 $p(v,s)=0$。

Q-SGD 将梯度值距离压缩端点的距离,以对应的概率将梯度压缩为两端的值。这种映射方式保证了在大规模梯度通信的情况下,编码后梯度的期望值与编码前的梯度值相等,实现了无损编码。但是,这种编码方式会增加节点间梯度方差,影响参数更新的准确性,进而妨碍模型收敛和降低精度。

为了减轻 Q-SGD 造成的梯度方差过大的问题,自适应梯度量化方法[38]引入 ALQ 和 AMQ 两种自适应梯度量化策略,提升了模型训练的精度;ECQSGD[39]引入了权重更新的校正机制。在量化梯度传输时,将误差额外存储在内存中,并与随后的梯度集成以进行权重更新。以防止量化引起的信息丢失,这种策略被后来的大多数压缩算法所采用。但却造成了内存压力的增加。

随着研究的深入,GRACE[26]关注到了梯度压缩会产生显著的压缩开销,但它没有将梯度压缩应用于 DDL。OMGS-SGD[40]和 Cupcake[23]努力整合小梯度压缩以减少压缩操作的频率,但没有解决压缩操作垄断 GPU 资源的问题,也没有关注压缩算法本身的精度性能缺点。OSP[41]对梯度进行了分类,通过量化梯度的重要性来决定是否进行梯度压缩,并以此来保证模型训练的精度。但是,该方法没有重点关注性能,计算与通信的并行性较低。

SlimGC 优化策略能够以梯度压缩优化插件的形式捆绑在各个分布式训练框架中，首次通过对梯度的细粒度筛选实现了最大化压缩收益。SlimGC 模型备份策略的设计突破了传统梯度压缩技术的局限，为分布式训练环境中的通信优化提供了新的系统级解决策略。其优势在于能够与现有压缩技术协同工作，保证模型训练的高效率和高准确性。

3 SlimGC 策略的设计

3.1 SlimGC 设计过程中的挑战

根据第 2.4 节中提到的现有压缩方法的缺点，我们总结了在设计 SlimGC 的时候面临的三个挑战：

挑战 1：使用梯度量化时，模型的收敛精度极大程度地受到梯度编码阈值的影响，一旦该阈值的设置偏离梯度分布则会严重损失收敛精度甚至导致模型不收敛。然而，我们在图 5 中展示了不同模型的梯度分布，发现它们并不具备统计规律。这意味着在训练不同模型时，需要开发人员具有丰富的先验知识来选择合适的超参数。当然，部分方法通过动量修正或者设立多个阈值来解决精度问题，但是这些方法本身就引入了额外的超参数。

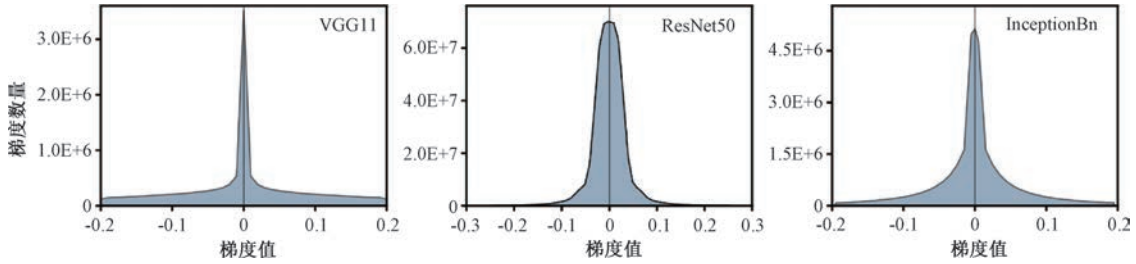


图 5 在不同模型训练过程中，各节点每轮生成的梯度值分布及梯度数量的情况

挑战 2：梯度量化并不总是带来正向的通信优化收益，这一现象在节点传输小规模梯度时尤为明显。这种情况下，大部分通信开销被启动开销占据，此时将梯度编码为低比特格式无法带来有效的性能收益，反而引入了额外的压缩开销。此外，在不同训练环境下，对于小规模梯度的划分不同，因此需要在不同条件下设定一个统一的标准。

挑战 3：在 GPU 上执行梯度量化会占用部分算力资源，延迟了张量计算的时机。如果将梯度量化卸载到 CPU 上，则会导致更大的压缩开销。需要找到一个策略——既能享受到卸载对 GPU 资源的释放，又能够消除因 CPU 算力低而带来的时间开销。

针对上述三点挑战，我们设计如下解决策略：首先，以优化压缩操作性能收益为前提，缩小参与压缩的梯度范围以降低压缩算法对压缩阈值的敏感性；并且在权重更新方面，降低因为梯度精度损失以及残差滞后参与权重更新而带来对模型训练精度的影响（第 3.3 节）。其次，应该统一所有梯度使用梯度压缩的时机，以确保每次压缩操作能带来正向的性能收益（第 3.3.2 节）。最后，应该规避压缩操作对 GPU 的资源竞争，保证训练阶段能够充分使用 GPU 的资源（第 3.4 节）。在此基础上，解决压缩开销导致通信延迟启动的问题。更大程度上提高通信与计算的重叠率，来获取更高的训练速度，实现模型训练性能的提升（第 3.5 节）。

3.2 SlimGC 的工作流程

为了解决与梯度量化方法相关的上述挑战，我们设计了 SlimGC，旨在通过通信阈值表（如图 6）最小化不必要的压缩操作，将压缩任务转移到 CPU 上执行，并解除节点间通信任务对计算操作的制约。

通信阈值表

梯度大小	t_c	t_c^q	t_p^q	t^q	压缩收益比
1MB	28.5ms	24.3ms	6.3ms	30.6ms	0.93
1.6MB	30.5ms	25.3ms	6.5ms	31.8ms	0.96
2.2MB	40.2ms	28.3ms	6.8ms	35.1ms	1.15
4MB	75.5ms	37.3ms	8.5ms	45.8ms	1.65
……	……	……	……	……	……

图 6 通信阈值表的示意图

图 7 展示了 SlimGC 的工作阶段的抽象视图，并详细说明了在训练阶段中，工作节点（worker）和服务器节点（server）的操作过程。

预训练阶段：为了确定压缩对哪些梯度能带来显著效益，SlimGC 将记录多个训练轮次中梯度在压缩前后的通信开销，并将数据存储在通信阈值表（如图 6）中。随后，SlimGC 将从这个表中得出通信阈值，来确定真正需要压缩的梯度范围，并指导之后的训练。

正式训练阶段：工作节点使用本地备份的模型进行梯度计算（步骤 1 至步骤 3），然后使用压缩优

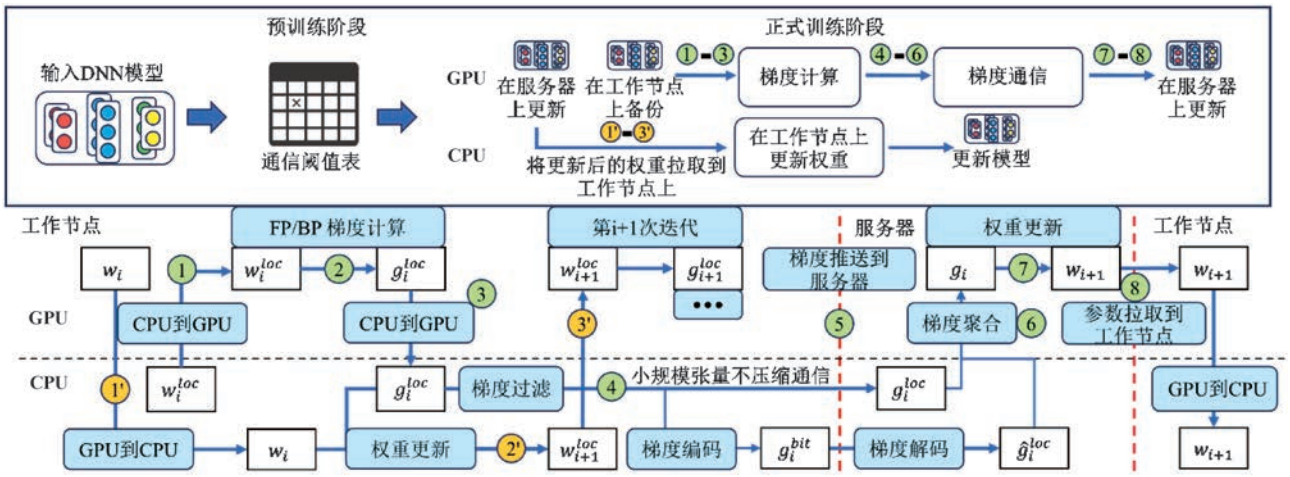


图 7　SlimGC 总览(包含用于确定压缩策略的预训练阶段;使用压缩优化与模型备份等优化技术的正式训练阶段。下方的正式训练过程由训练涉及的关键参数以及相关操作来描述。圆圈中的数字描述了各操作的相对先后顺序,绿色代表训练过程,黄色代表模型备份阶段。)

化策略,对每个梯度做出最优的调度,并传送给服务器节点(步骤 4 至步骤 6)。在服务器获取到梯度之后,进行聚合,得到全局的梯度值,并依次计算得到全局权重并将其与各个工作节点同步,工作节点获取权重之后便准备进行下一次的训练(步骤 5 至步骤 8)

模型备份阶段:在分布式训练参数服务器架构中,SlimGC 解决了传统压缩策略中的延迟问题,其核心在于让工作节点在每次迭代开始时,使用本地备份的模型而非依赖服务器上的全局模型。在这一过程中,工作节点利用本地计算得到的梯度 g_i^{loc} 对备份模型进行更新,并在完成这些更新后立即开始下一次迭代(从步骤 1′至步骤 3′)。与此同时,本地计算得到的梯度也会同步地传给服务器节点,以保证工作节点能够及时获取到最新的全局权重来进行下一次的迭代。

模型备份阶段本质上属于正式训练过程的一部分,将其单独讨论是为了突出其在方法论中的创新性,并帮助读者更好地理解其对整体训练策略的贡献。通过这种区分,我们能够更清晰地展示模型备份在提高训练效率和模型性能方面的重要作用。

3.3　压缩优化策略

SlimGC 的压缩优化策略基本原则是避免对小规模梯度进行压缩,以此来指导正式训练,确保精度提升并加快训练速度。

本论文中压缩方法的优化策略设计依据以下几个关键原则:第一,通信开销由两部分构成:通信操作开销及数据传输开销。第二,对于小规模张量,通信开销的主要组成部分是通信操作的启动成本,且在某一数据区间内(例如 1B 至 1KB),数据的通信开销趋于一致。第三,对于大规模张量,数据传输开销与数据量之间存在正相关关系,即数据量越大,传输开销越高。第四,压缩操作会引入额外的编码与解码开销,应一并考虑进通信开销。

3.3.1　设计思路

SlimGC 的压缩优化策略主要分为两部分,预训练构建通信阈值表(Communication Threshold Table, CTTab)并尽可能消除网络波动带来的影响;正式训练阶段,SlimGC 通过通信阈值表获取通信阈值,并将其统一至各个节点来指导正式训练。

算法 1. 通信阈值表构建方法 *CTTab*

输入:训练模型 *model*, 训练使用的优化器 *optimizer*.

输出:通信阈值表 *CTTab*

1. 训练阶段初始化之后获取预热阶段所需的迭代次数 *pre_steps*, 并初始化通信阈值表 *CTTab*.*init*().
2. FOREACH *step* = 1 TO *pre_steps*
3. 模型预热阶段计算 *model*.*train*()期间得到梯度 *grad*
4. 依据 *step* 的奇偶性针对梯度进行不同的同步处理,并将得到的时间计入通信阈值表中.
5. IF (*step* 是偶数)
6. 对梯度压缩后同步 *CompressPush*(*grad*)
7. ELSE
8. 直接进行梯度同步 *DefaultPush*(*grad*)
9. 服务器同步完之后拉取全局梯度 *Pull*(*grad*)
10. 将梯度传输的开销同步到 *CTTab* 中
11. *CTTab*.*avg*()
12. 计算得到图 6 所示的压缩收益比
13. RETURN *CTTab*

在预训练阶段，如算法 1 所示，SlimGC 会记录每个梯度在压缩及非压缩状态下的通信相关开销，并依据梯度大小，将这些记录从小到大排序，整合于通信阈值表中（图 6 所示）。为减小网络波动带来的影响，工作节点会对多轮迭代产生的数据进行平均处理（算法 1，第 11 行）。

算法 2. SlimGC 利用通信阈值表指导训练

输入：训练模型 *model*，训练使用的优化器 *optimizer*，用户设定的训练相关的超参数如 *epoch*

1. 通过通信阈值表得到通信阈值 *BestCT*
2. 向其余工作节点广播通信阈值 *Broadcast*(*BestCT*)
3. FOREACH $i = 0$ TO *epoch*
4. 　工作节点上模型计算 *model.train*()
5. 　FOREACH *grad* IN *optimizer*
6. 　　IF (*grad.size*＜*BestCT*)
7. 　　小于通信阈值 *DefaultPush*(*grad*)
8. 　　ELSE　*CompressPush*(*grad*)
9. 　　从服务器上拉取更新后的全局梯度 *Pull*(*grad*)
10. 　　获取全局梯度后对模型进行更新 *model.step*()

算法 2 则是描述了通信阈值表指导训练的具体过程。预训练结束后，工作节点将自上而下顺序遍历通信阈值表，寻找压缩收益（t_c/t^q）大于 1 的梯度大小（如图 6 为 2.2MB），并将其作为通信阈值（算法 2，第 1 行）。这个通信阈值将被广播到所有工作节点，用于指导后续正式训练阶段（步骤 4 至 6）中对特定梯度是否进行压缩。

在通信阈值表中，若其压缩通信成本（t_c^q）与压缩处理成本（t_p^q）之和超过未压缩的通信成本（t_c），则认定该梯度为小规模梯度，相反则是大规模梯度。小规模梯度将直接传输至服务器节点；大规模梯度则会被编码为低比特格式（g_i^{bit}），并在服务器节点上解码回浮点格式（$\hat{g}_i^{loc}$）后再参与梯度聚合。

3.3.2 压缩优化策略的优点

优点 1：在不同模型或网络条件下，压缩优化策略有着很强的适应性与泛化性。由于不同模型梯度规模大小分布不同，通信阈值表和通信阈值也会有所变化；对于不同网络带宽，梯度压缩带来压缩收益的范围不同，通信阈值也会不同。在面向深度学习独占式集群环境中，压缩优化策略得到的压缩阈值表切实有效地反映了当前训练条件下梯度压缩的收益范围。然而，对于实时网络波动较大的应用场景，比如边缘计算集群，以及那些不是以模型训练为主且资源非独占的集群环境，并不在我们的考虑范围之内。因为这些场景下的任务并不追求在最短时间内完成模型训练，而是可能更关注于任务的实时性和资源的共享使用。因此，SlimGC 的设计初衷是专注于那些资源独占、追求训练效率的场景，而不是涵盖所有可能的应用环境。

优点 2：压缩优化策略对模型训练的精度与性能两方面都有提升。精度提升方面：采用压缩优化策略可以降低原有压缩方法对压缩阈值的敏感性，并且小规模梯度的全精度通信使得模型训练的精度高于一般的压缩方法，这一点将在第 1.1 节展示。性能提升方面：算法的预训练部分与模型本身的预热阶段[42]是重合的，不会因为预训练构建通信阈值表而导致模型本身的性能降低。同时，预训练中记录通信开销的任务由独立于训练的线程执行，不会引入显著的额外开销。从图 8(c)与(a)的比较中可以推断：正式训练阶段，由于部分梯度取消了压缩操作使得编码成本降低，小规模梯度的通信可以更早启动，从而减少了前后迭代计算阶段之间的等待时间，下一轮迭代可以更快开始。

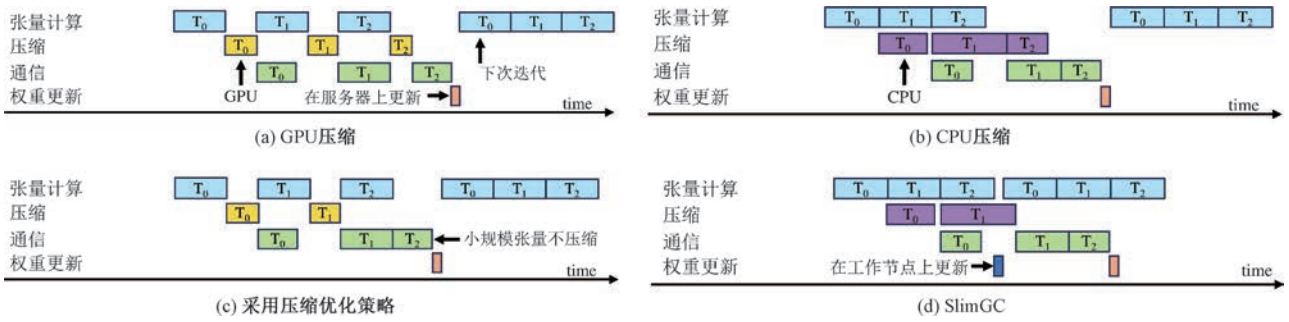


图 8 使用多种压缩策略进行分布式深度学习的时间成本示意图

3.4 压缩卸载策略

在模型训练的前向和后向传播过程中，GPU 以并行方式高效执行张量计算任务。而梯度的压缩操作并不严格要求并行处理。实际上，每一层的梯度计算完成之后就会立即进行压缩操作。但是，梯度压缩操作一般是在 GPU 上完成的，在 GPU 上执行

会占用宝贵的计算资源,从而影响张量计算的效率。

因此,SlimGC 提出了压缩卸载策略,即将压缩相关的计算任务从 GPU 卸载到 CPU 上。这样可以避免 GPU 在模型计算与梯度压缩任务之间频繁地切换,造成不必要的资源浪费。并且,梯度数据的存储将被同步迁移至 CPU,这有助于减少 GPU 的显存开销,并在第 4.6 节进行了实验验证。

尽管压缩卸载策略能够释放 GPU 资源,但它也带来了新的挑战,下面列举了存在的挑战以及我们的解决方案:1)数据从 GPU 到 CPU 的传输引入了额外的开销,可能导致 CPU 对梯度压缩的处理延迟。为了缓解数据复制带来的影响,SlimGC 引入了一种非阻塞数据传输机制。该机制不依赖于 GPU 以特定顺序传输数据,而是依据每个张量的键值来确认数据传输的成功。在此过程中,我们特别关注了那些不应被压缩成本所掩盖的数据复制开销(t_p^q)。2)由于 CPU 的算力低于 GPU,因此 CPU 在执行压缩操作时的速度通常慢于 GPU。在 CPU 上执行的压缩操作成本可能接近甚至超过梯度计算的成本,这会显著延迟通信启动的时间(参见图 8(a)与图 8(b)),进而影响模型训练的速度。因此,SlimGC 还需要采取额外措施(详见第 3.5 节)来解决工作节点对最新权重读取的依赖问题,以充分利用压缩卸载的优势。

3.5　模型备份策略

3.5.1　设计思路与原则

模型训练的核心在于利用训练迭代产生的反馈信息梯度来不断调整模型参数。因此,工作节点必须获取最新的模型参数以启动新的迭代。基于这一需求,SlimGC 在每个迭代周期中引入了模型参数的备份机制。具体而言,SlimGC 备份服务器节点在前一个迭代中同步更新的模型参数。这些备份参数并不直接用于启动新的训练迭代,而是作为工作节点执行局部权重更新的基准,从而在下一次迭代中使用,而不是直接采用服务器新生成的模型参数(公式 1)。公式 8 表示工作节点上的权重更新。

$$w_{i+1,j}^{loc} = w_i - \eta * g_{i,j}^{loc} \tag{8}$$

模型备份的实现基于一个简单而有效的原则:工作节点在利用本地梯度信息更新其备份的模型参数后,这些局部梯度信息最终会在服务器节点上进行聚合。理解策略的核心在于通过反馈信息的时间点来区分迭代的启动方式。具体来说,使用备份模型的工作节点会在全局梯度聚合完成之前,就将这些局部梯度更新融入迭代过程中。相比之下,依赖全局模型的服务器节点则在聚合完成后才开始新的迭代周期。简而言之,备份模型的策略使得工作节点能够在全局聚合完成之前先行一步,从而显著加速了整个训练过程。

3.5.2　模型备份的优点

优点 1:计算通信重叠率增高。图 8(d)展示了在 SlimGC 中使用模型备份的作用,它使得工作节点能够在执行权重更新后直接开始下一次迭代,从而避免了由于通信导致的长时间等待。这样一来,工作节点的计算操作与通信操作可以并行进行,减少了因压缩操作而增加的计算延迟。通过减少迭代间的等待时间,模型训练的整体效率得以提升。需要强调的是,模型备份机制仅仅打破了相邻迭代之间计算行为对通信行为的参数读依赖,具体而言,第 $i+1$ 次迭代的计算可以不依赖于第 i 次迭代的同步通信结果而独立启动,只要工作节点已经基于第 $i-1$ 次迭代完成了全局权重更新,获得了 w_i,第 $i+1$ 次迭代就可以启动(参见公式(8))。这种设计确保了所有工作节点在对备份权重进行本地更新之前,都使用相同的全局权重,而非异步更新的结果。这一策略不仅提高了训练效率,还保证了模型更新的一致性和准确性。

优点 2:梯度量化算法中的权重精细化更新。在第 2.3 节中,我们讨论了梯度经过编解码处理后,与原梯度之间存在着较大的偏差。为了减轻梯度量化引发的数据损失,误差补偿机制将梯度压缩与原始梯度之间的差异暂存于缓存中,仅当累积残差超过预设阈值时,才会将其发送至服务器参与权重更新。这种处理方式可能导致小数值梯度无法及时反映在权重更新中。尤其是在模型训练的后期,随着模型逐渐收敛,梯度数值趋向于更小,残差累积机制可能会阻碍小数值梯度对模型进行精细化更新。然而,工作节点在每次迭代开始时所使用的权重是基于本地非压缩梯度对备份权重的更新结果(参见公式(8)),这意味着模型备份机制为梯度量化算法提供了更为精细的权重更新方式。

在实验部分(见第 1.1 节),我们通过实验研究了模型备份机制对模型精度的影响,结果表明采用模型备份机制能够在一定程度上提高模型的收敛精度。这一发现进一步证实了模型备份机制在提升梯度量化算法性能方面的重要作用。

3.6 SlimGC 在框架中的实现

SlimGC 可以作为插件嵌入各训练框架负责分布式通信的模块，如 MXNet 的 KVStore 模块与 PyTorch 的 DDP 模块。它不干涉原训练框架的训练过程，即前向传播和后向传播过程。当节点产出梯度时，SlimGC 提供了名为 *grad_buf* 的缓存区来将梯度存放于 CPU。之后，工作节点根据通信阈值表提供的通信阈值对 *grad_buf* 中的梯度进行筛选，小于阈值的梯度被拷贝进 *fp_buf* 缓存区，而大于阈值的梯度经过训练框架提供的压缩器压缩后存放于 *bit_buf* 缓存区。通信阈值表是根据计时器监测的通信开销生成的。计时器并没有额外构造新的时间测量工具，而是在识别所用的训练框架后调用框架内置的时间开销测量组件（如 mx.profiler 和 torch.autograd.profiler）。}

fp_buf 和 *bit_buf* 中的数据被传入通信接口后，训练框架提供的同步通信与权重更新模块开始正常工作。权重处理方面，SlimGC 提供了 *glo_buf* 用于接收每次迭代新产出的权重，并由 *loc_buf* 进行权重备份。*loc_buf* 附带了一个状态符号，用于标记其中的备份权重是否已经被本地更新。本地更新操作由本地更新器执行，它封装了训练框架原有的更新器，要求用户为其设置独立的超参数，比如本地更新学习率 *loc_lr* 、本地更新算法 *loc_optimizer*（通常与全局更新器所用参数保持一致即可）。本地更新产生的新权重会覆盖 *loc_buf* 原有的数据，并将 *loc_buf* 的状态符号由 0 变为 1，此后 *loc_buf* 中的数据将被训练器读取以启动下一次迭代训练。每次 *glo_buf* 的数据写入 *loc_buf* 前会查看其状态符号是否为 1，在权重备份后将 *loc_buf* 的状态符号由 1 变为 0，这种机制的设计可以以避免训练过程中出现权重读写错误。

4 实验验证

在本节中，我们评估 SlimGC 及其各个策略对精度、性能、超参数敏感性、GPU 内存优化四个方面的提升。精度方面，验证 SlimGC 的压缩优化策略和模型备份策略对模型训练的精度提升。超参数敏感性方面，探究压缩优化策略是否可以降低对一些精度相关超参数的敏感性。性能方面，验证三种策略对训练速度的提升，并进一步通过分析训练开销组成来探究性能提升的原因。内存方面，验证压缩卸载策略对 GPU 内存的优化。

4.1 实验环境设置

集群配置：我们在一个配备了 8 个 Intel Xeon Gold 6230R 处理器的集群上进行了实验。每台机器兼容 PCIe 3.0 x16 协议，配备 2 个 V100 GPU。此外，每个节点配有两种类型的以太网卡，分别提供 10Gbps 和 100Gbps 的最大带宽。操作环境为 CentOS 7.6、CUDA10.2 和 cuDNN7.4.1。

数据集：我们在计算机视觉和自然语言处理任务两方面进行了实验评估。

准确率测试：在 CIFAR-10[43] 上训练 ResNet-50[44]；在 ImageNet ILSVRC2012[45] 上训练 VGG-16[46]；在 SQuAD[47] 上训练 BERT-base[48]。

超参数灵敏度测试：在 CIFAR-10 上训练 ResNet152 和 Inception-Bn[49]。

性能测试：在 CIFAR-10 上训练 ResNet-152；在 ImageNet ILSVRC2012 上训练 VGG-16；在 SQuAD 上训练 BERT-base。并在 ImageNet ILSVRC2012 上训练 VGG16，VGG19 和 GLUE SST2[50] 任务上训练 BERT-Base 和 GPT-2[51] 来做横向对比。

分析训练开销构成的实验：在 CIFAR-10 上训练 ResNet-152 和在 ImageNet ILSVRC2012 上训练 Inception-Bn。

内存占用分析实验：在 ImageNet 上训练 Inception-Bn、AlexNet[52] 和 VGG-16。

训练框架：SlimGC 支持 MXNet 以及 PyTorch 两种训练框架。并通过第 3.6 节介绍的方式，将 SlimGC 覆盖 MXNet 和 PyTorch 的通信模块来实现与框架的结合。实验采用参数服务器架构，我们为每个工作节点都配置一个相应的服务器节点，在执行通信操作时，它们本质上起着 AllReduce 的作用。

基准线：我们使用 1bit 量化[19] 和 2bit 量化[53] 压缩算法作为 SlimGC 的对比。在 GPU 上执行压缩操作时，以 1bit 量化的训练速度作为性能测试的基准。采用误差反馈[39] 来保持模型精度。此外，在第 4.4.3 小节将 SlimGC 与主流压缩方法 Terngrad[37]、TopK[34]、DGC[30] 和最新的压缩优化类最优方法 THC[54] 进行了横向性能对比。

性能指标：准确性测试：使用 Top-1 精度和 F1 分数作为评估指标。性能测试：基于计算机视觉模型训练中每秒处理的图像数量和 NLP 模型训练中每秒处理的 token 数量来评估。为了防止因性能指

标的显著差异而造成的误解,我们根据基线结果对数据进行了归一化。横向测试中使用平均后每次迭代的训练速度(Samples/sec)作为标准。训练开销组成测试:测量每次迭代中各种操作的时间开销。内存占用测试:评估每次迭代中 GPU 内存消耗的变化。

4.2　模型收敛精度实验分析

实验包括 6 个组成对象:1bit 和 2bit 压缩算法,使用 SlimGC 优化压缩策略的 1bit 和 2bit 压缩算法(CG Opt.),以及使用所有 SlimGC 优化策略的 1bit 和 2bit 压缩算法(SlimGC)。

各实验对象采用相同的参数设置和实验配置,所有超参数(批大小、压缩阈值、学习率)均按照对应数据集和模型的官方推荐设置。各方法的精度变化曲线如图 9(a)所示,从中可以观察到以下现象:

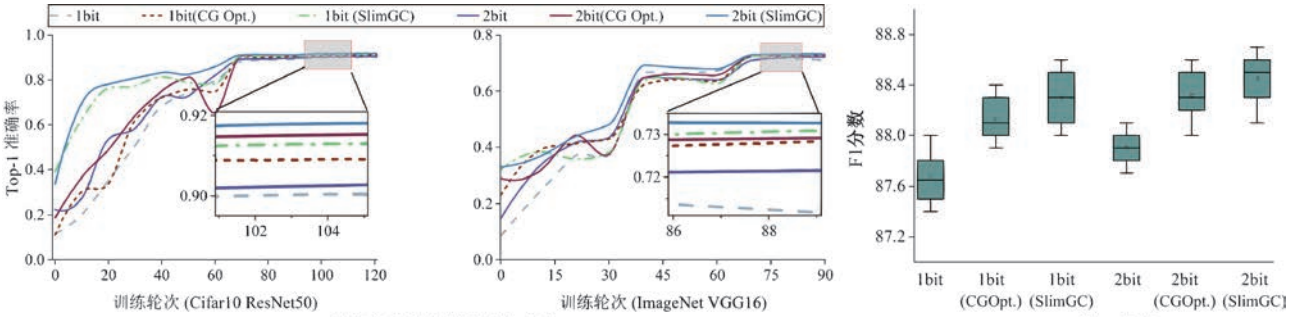
首先,在不使用压缩优化或使用相同的压缩优化技术的情况下,由于能够表达更丰富的信息,2bit 压缩算法收敛精度比 1bit 压缩算法高 0.2%～1.1%。

其次,在相同训练次数的情况下,优化后的 SlimGC 压缩策略有效地提高了压缩算法的收敛精度。1bit(CG Opt.)和 2bit(CG Opt.)可以帮助 1bit 和 2bit 压缩算法分别实现 0.7%～1.7%和 0.7%～1.6%的精度提升。

第三,在采用了所有优化技术后,SlimGC 的 1bit 和 2bit 压缩算法的准确率分别提高了 1.1%～2.3%和 1.1%～1.6%。

第四,由于小尺度梯度以全精度方式传输,并且工作节点上的备份模型以全精度梯度更新,因此,SlimGC 可以帮助压缩算法在模型训练的早期更快地寻找合适的权值,从而帮助节点在收敛阶段找到最优解。这种现象在 ResNet-50 等模型的训练中更明显,因为这些模型包含更多的小尺度梯度。

我们对 SQuAD 上的问答任务进行了两个 epoch 的 BERT-base 微调,并重复了 6 次实验,比较了各种方法的 F1 分数。从图 9(b)我们可以看到,在 NLP 模型训练任务中,1bit 压缩算法和 2bit 压缩算法的准确率差异比较小,在 0.3%以内。此外,优化后的压缩策略是提高 1bit 和 2bit 压缩算法精度的关键,可以将精度提高 0.4%～0.5%。基于 2bit(CG Opt.),结合局部全精度更新技术,2bit(SlimGC)达到了全精度训练的标准准确率(88.5%)。



(a) 图像分类任务的准确率曲线　　(b) NLP的*F*1分数

图 9　模型训练过程中,采用不同策略的压缩算法的准确率表现(一个 epoch 代表对整个训练数据集的一次完整遍历)

4.3　参数敏感性实验分析

在面对多样化的数据集和模型以追求最优化的训练精度时,框架开发人员在经过一系列迭代训练后,能够确定一套最优的超参数配置,从而为框架的最终用户省去了繁琐的参数调整工作。然而,由于每位框架使用人员的工作配置环境存在差异,对超参数进行微调仍然是不可避免的。SlimGC 在精度上做了针对性优化,因此在一些对精度影响很大的超参数(如 batch-size,压缩需要的压缩阈值等)难以确定的时候,SlimGC 可以使得不同超参数设置下对模型的精度影响最小。

4.3.1　压缩阈值分析

在探索影响 SlimGC 压缩算法精度的关键因素时,我们研究了三个超参数:压缩阈值、通信阈值和学习率。由于在 DDL 训练过程中,工作节点的数量与学习率息息相关。在增大节点数量时,需要将 batch-size 和学习率进行合适的缩放,这里选用了两种常用的缩放方式:线性缩放和平方根缩放。图 10 显示了利用各种压缩阈值时每种压缩算法可实现的最高精度,可以观察到:

首先,当压缩阈值为 0.5 时,1bit 和 2bit 压缩算法的精度都最高。这个阈值更符合大多数模型的梯度分布,并且是官方所推荐的。

其次,1bit 压缩算法在采用线性缩放原理(学习率与批大小成比例增加)调整学习率时,无法实现 ResNet-152 的收敛;然而,当使用平方根缩放策略

来调整学习率时，它实现了收敛。在不使用其他精度优化方法的情况下，该方法的收敛性依赖于合理的学习率设置。

第三，当使用不合适的压缩阈值时，两种压缩算法使得模型的收敛性变得不可预测。此外，偏离理想阈值(0.5)的大小与模型收敛性退化之间没有相关性。

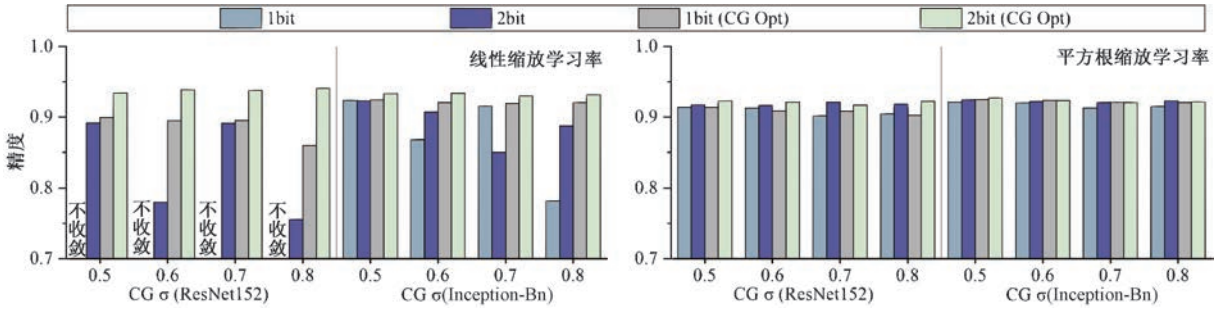


图 10 在不同压缩策略下，使用不同的压缩阈值训练 ResNet-152 与 Inception-Bn 模型时的精度值(CG：压缩阈值)

最后，从现有算法中设定的压缩阈值所积累的经验不一定会为其他算法的优化提供有价值的指导。例如，在 Inception-Bn 的训练过程中，1bit 压缩算法的压缩阈值为 0.7，优于 0.6，而 2bit 压缩算法表现出相反的行为。

采用压缩优化策略可能导致压缩算法的最优压缩阈值发生偏移。因此，当在相同的模型上训练时，不同的算法可能会导致不同的最佳压缩阈值，甚至相同的算法，也可能在不同的模型训练场景中也表现出不同的最佳压缩阈值。这种现象产生的原因是不同模型之间张量分布的差异，以及不同算法的代价特征的差异。压缩阈值的差异导致不同比例的梯度被压缩，从而会影响 SlimGC 动态确定的通信阈值。在这种情况下，SlimGC 避免追求最佳的压缩阈值设置，因为它认识到这会给用户带来额外的负担。与采用最佳压缩阈值设置相比，SlimGC 将压缩阈值统一设置为 0.5，并可以有效地将精度损失控制在 0.6%以内。

4.3.2 通信阈值分析

经典的 1bit 和 2bit 压缩算法可以看作是通信阈值为零的 1bit(CG Opt)和 2bit(CG Opt)算法。因此，验证最佳通信阈值设置以提高压缩算法的准确性至关重要。图 11 给出了不同通信阈值下各算法的最小损失值，压缩阈值固定为 0.5。观察到，当数据量小于 1500B 的张量不进行压缩通信时，压缩算法的精度得到了显著提高。而且，随着通信阈值的增大，压缩算法的精度效益也在逐步提高。

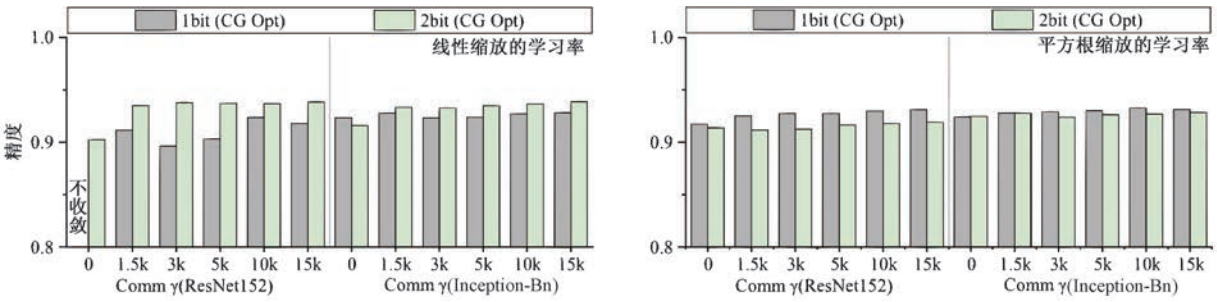


图 11 在不同压缩策略下，使用不同通信阈值训练 ResNet-152 与 Inception-Bn 模型时的精度值(Comm：通信阈值)

这一结果具有积极的研究意义，因为图 1 中的数据表明，在传统的计算能力和网络配置(V100 GPU 和 100Gbps 以太网)下，1bit 和 2bit 压缩算法只有在压缩张量为 80KB 或更高时才能实现真正的性能优势，通信阈值远远超过 1500B。理论上，当通信阈值设置为无穷大时，1bit(CG Opt)和 2bit(CG Opt)可视为全精度梯度同步通信算法。在实际的模型训练场景中，达到一定的通信阈值可以产生最优的效益。例如，当采用线性缩放学习率策略时，将通信阈值设置为 15KB，可以使 2bit 压缩算法达到全精度训练的标准精度：ResNet-152 和 Inception-Bn 分别达到 93.9%和 92.9%的 Top-1 精度。

4.4 模型加速效果实验分析

为了探究 SlimGC 在通信与计算任务量占比不同的环境下的表现，我们选取了三个不同类型模型进行性能训练，来验证在不同环境下 SlimGC 的泛

化性。我们选择了 ResNet-152、VGG-16 和 BERT-base 等典型模型,以评估 SlimGC 在压缩算法性能优化方面的效果。ResNet-152 由于其密集的小规模张量卷积计算,被归类为计算密集型。VGG-16 则因全连接层的大规模梯度通信成本高,被归类为通信密集型,BERT-base 则融合了复杂的计算和频繁的通信需求。

4.4.1　10Gbps 网络下的训练性能

图 12(a)展示了在 10Gbps 网络环境下,模型采用不同算法的训练性能。以 1bit 压缩算法的速度为标准,将其他方法的训练速度进行归一化,直观地比较了不同训练方法的性能差异。观察结果如下:

第一,基准线的两个压缩算法与模型本身的计算与通信的占比关系紧密。对于 ResNet-152 这类计算密集型模型,1bit 压缩算法并未显著超越 2bit 压缩,因为在高通信压缩率的实现上存在挑战。而在 VGG-16 和 BERT-base 这类通信操作更为频繁的模型训练中,1bit 压缩算法则显示出更优的性能,尤其在增大批处理大小时,其优势更为显著。

第二,基准线在分别使用 CPU 与 GPU 压缩时在不同模型下展示了不同的特性。在 ResNet-152 这类计算密集型模型训练中,GPU 压缩与 CPU 压缩策略的性能差异不大,且这种差异不受批处理大小的影响。在 VGG-16 的大批量训练中,CPU 压缩反而能带来更佳性能。但对于 BERT-base,CPU 和 GPU 压缩策略的相对性能则受批处理大小和压缩算法类型的影响,没有统一规律。

第三,在低带宽环境下,对于模型 ResNet-152,SlimGC 可以为 1bit 和 2bit 压缩分别带来至多 28% 和 32% 的性能提升。在训练 ResNet-152 时,其通信任务集中于大量小规模梯度的交换,因此 SlimGC 可以充分发挥压缩优化策略的优势,有效降低通信开销。对于其他模型,SlimGC 也能为压缩算法带来 4%到 16%的性能提升。这些模型有着十分复杂且频繁的梯度交换,由于网络带宽的限制,计算成本和通信成本的不平衡进一步加剧,使得通信屏蔽更加困难。特别是对于 VGG-16 等模型,主要的通信开销并不来源于小规模梯度,而 SlimGC 在低带宽环境下对于 VGG-16 等模型倾向于使用较小的通信阈值,这限制了可优化梯度通信的范围,使得压缩优化策略的优势很难发挥。

4.4.2　100Gbps 网络下的训练性能

为了消除低带宽网络带来的干扰,将 10Gbps 以太网替换为 100Gbps 以太网,并在相同配置下进行测试。如图 12(b)所示,当使用更高带宽的网络进行训练时,各算法的性能都发生了一些变化:

第一,在解除带宽限制条件后,CPU 压缩普遍展示出比 GPU 更好的性能。这是由于在高带宽环

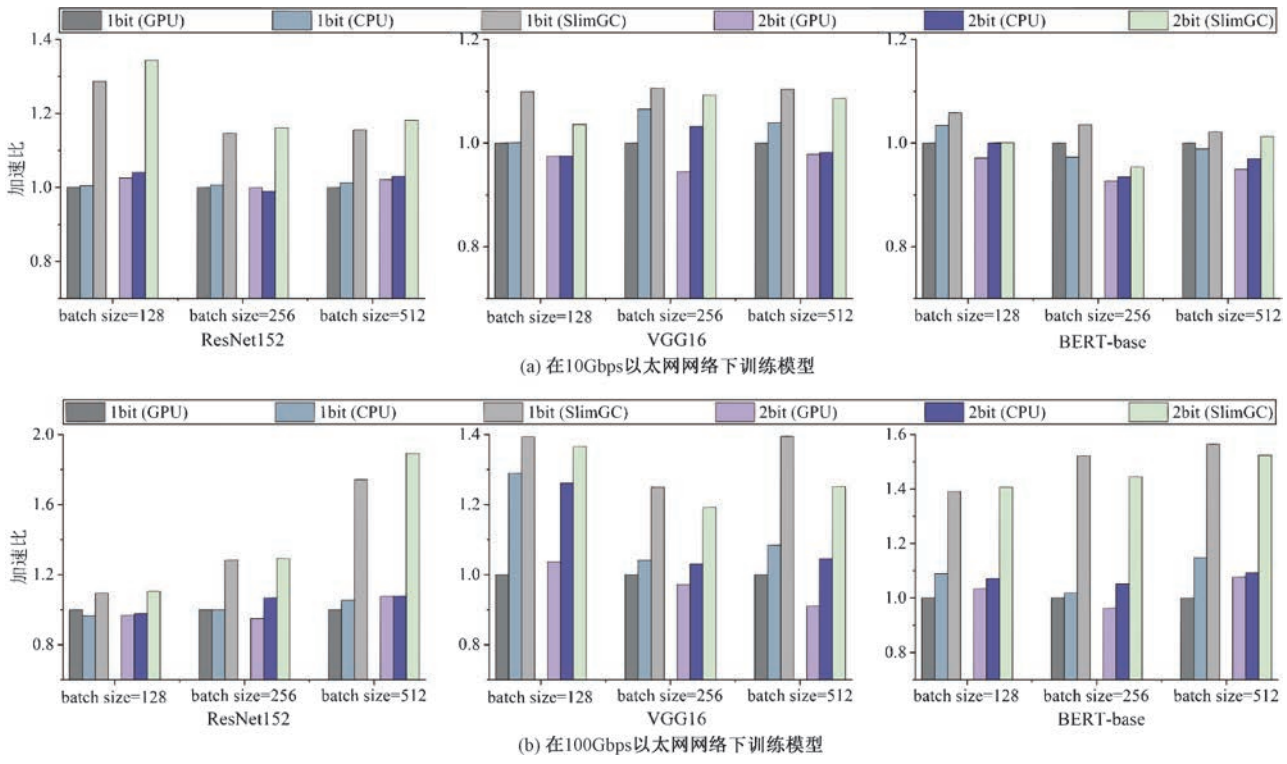


图 12　在不同网络带宽环境下,采用多种压缩策略训练模型时的归一化训练速度。

境下,GPU 上的张量计算操作对压缩操作的延迟变得更加敏感,这使得 CPU 压缩策略在多个场景下展现出性能优势,而 SlimGC 的压缩卸载策略(见第 3.4 节)也更有可能带来性能提升。

第二,在大批量训练模式下,SlimGC 的优势更加明显。大批量训练增加了每次迭代的计算成本,但由于梯度的规模与模型类型固定,通信成本并未改变。SlimGC 在大批量下的优势原因有二:一、通过压缩卸载,将 GPU 资源尽可能多的用于计算上,使得通信阶段不会因为压缩对 GPU 资源的占用而导致性能的下降。二、通过模型备份,增强了计算与通信操作的并行性,从而提升了整体性能。

第三,SlimGC 有助于缩小不同梯度量化压缩算法之间的性能差异。例如,在 ResNet-152 中,经过 SlimGC 优化后,2bit 压缩算法的性能接近 1bit 压缩算法。这种提升主要得益于对小梯度压缩的优化。理论上,当通信阈值无限大时,压缩算法间将无性能差异。

SlimGC 在高带宽环境中通过设置更高的通信阈值,并通过模型备份实现更高的通信掩蔽率,进一步扩大了性能优势。在 ResNet-152、VGG16 和 BERT-base 模型的训练中,SlimGC 显著提升了 1bit 和 2bit 压缩算法的性能。具体来说,对于 1bit 压缩算法,性能提升分别为 74.3%、39.5%、29.9% 和 56.6%;对于 2bit 压缩算法,性能提升分别为 75.9%、37.4%、68.7%和 50.1%。

4.4.3 横向性能评估

如表 3 所示,在 ImageNet ILSVRC2012 和 GLUE SST2 数据集上,SlimGC 与其他主流的通信优化算法进行了性能对比。SlimGC 是 THC 速度的 1.53~1.55 倍,而与 SlimGC 性能相近的 Terngrad 由于采用了三值量化机制,机制更简洁,没有引入额外的动量修正等计算开销,并且压缩比例高因此获得很好的性能收益。与 SlimGC 类似,THC 在梯度量化方面的目标都是尽可能让梯度实现无损量化,并同时提高训练的性能。但是,THC 实现的方法是通过构建大量的表通过压缩符号将梯度重新映射回高精度的浮点数,因此,THC 的性能受限于表格的反复读取。而 SlimGC 维护的通信阈值表则更为轻量,可以在显存中快速读取。与此同时,SlimGC 在系统级层面也通过模型备份技术实现了更高的计算与通信的重叠率,进而更进一步地提升了模型训练的性能,而这正是 THC 所忽略的。

表 3 SlimGC 与主流压缩方法在 100Gbps 网络环境下的模型训练速度对比(单位:Samples/s)

模型	Terngrad	TopK10%	DGC10%	THC	SlimGC
VGG16	92.25	52.17	37.33	69.3	107.78
VGG19	80.32	49.92	35.86	64.1	98.40
Bert-Base	123.75	102.20	92.44	118.08	143.15
GPT-2	110.73	95.50	84.12	106.29	127.98

对于 GPT 和 DiT 等这一类在不同层之间张量大小相对一致的模型而言,SlimGC 相较于 THC 也有着 1.20 倍的性能提升。这是由于这些模型也会产生并同步很多不同类型的小张量(例如注意力机制中的查询、键、值矩阵等)。SlimGC 的压缩优化策略可以通过避免对这些小张量进行压缩,直接以全精度传输,从而减少压缩和解压缩的开销,实现这类模型训练性能提升。

4.5 训练开销组成实验分析

为了进一步研究 SlimGC 的各个策略对训练开销组成的影响,我们使用 MXNet 的内置定时器记录了在 CIFAR-10 上 ResNet-152 和 ImageNet 上 Inception-Bn 训练期间节点执行的时间成本,并对每个模型进行了 5 次训练,每次训练 20 个 epoch,计算每个 epoch 中每个节点的平均计算、通信、压缩和重叠成本。如图 13 所示,得出以下结论:

首先,模型备份能够有效提高模型训练中的计算与通信的重叠率。训练计算密集型的 ResNet-152 时,SlimGC 能够将重叠的时间成本扩大到 6.4 倍以上;对 Inception-Bn 模型也有至少 3.5 倍的提升。

其次,模型备份技术能够有效解决 CPU 压缩带来的额外压缩操作开销。CPU 压缩相较于 GPU 压缩在 ResNet-152 和 Inception-Bn 训练中的开销分别高出 35.6%和 9.5%。但是,由于模型备份技术可以使大部分压缩操作与模型计算同步进行,因此未被重叠的压缩开销在 ResNet-152 和 Inception-Bn 训练时分别降低到原来的 21.8%~23.3%和 33.9%~41.4%,从而保证了本地模型的及时更新。

最后,SlimGC 在降低压缩成本方面表现突出,特别是在 ResNet-152 模型训练中效果显著。这得益于该模型生成的大量小尺度梯度,SlimGC 通过避免对这些张量进行压缩,同时优化通信过程,有效减少了开销。SlimGC 使得 1bit 压缩算法的通信成本降低了 26.5%至 35.6%,而 2bit 压缩算法的通信成本降低了 29.8%至 33.9%。

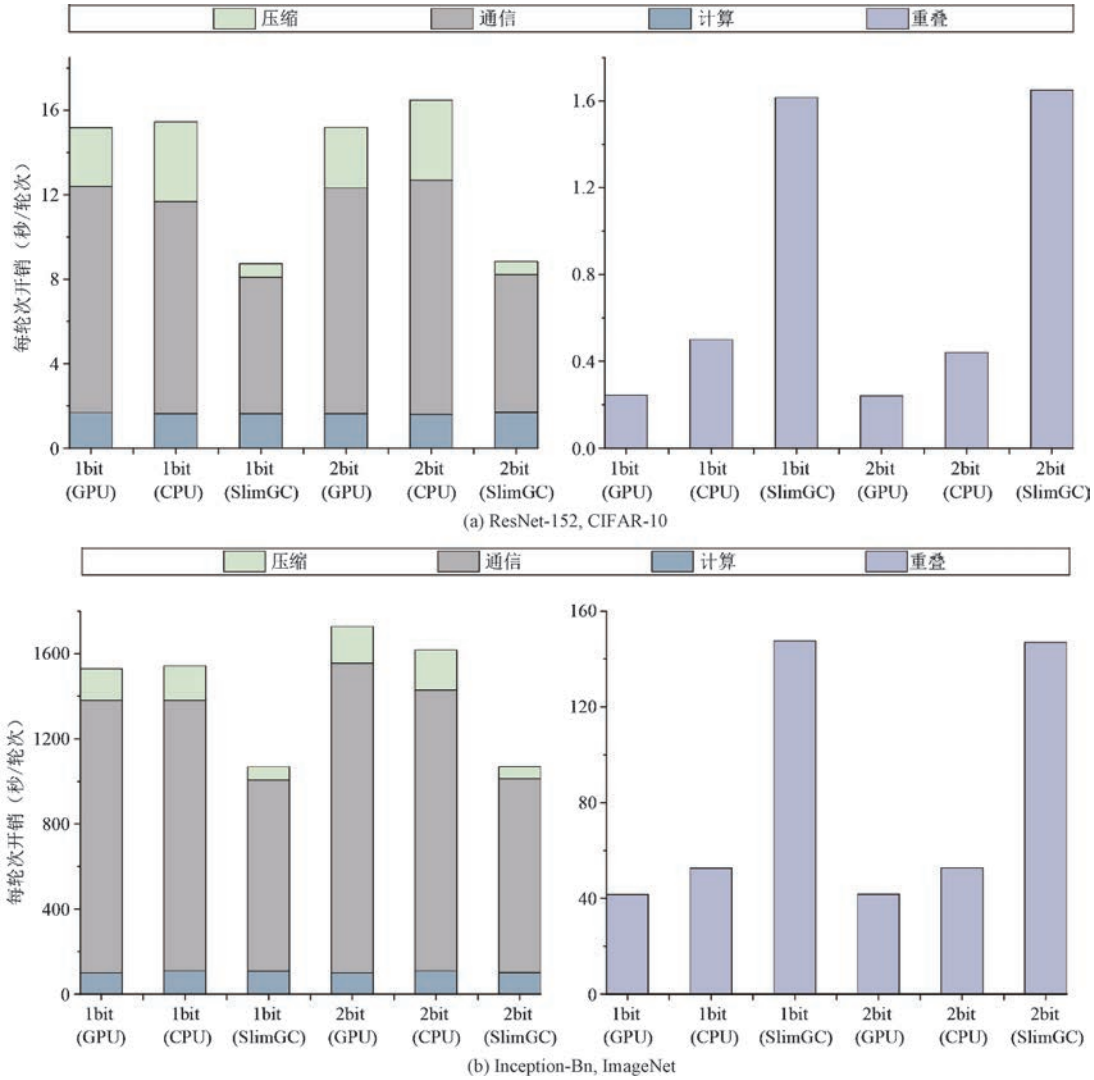


图 13　在 100Gbps 带宽网络下,训练 ResNet-152 和 Inception-Bn 模型时,节点在每个周期执行各种操作的时间开销

4.6　内存消耗情况实验分析

我们通过 NVIDIA 系统接口记录了在不同压缩策略下的 GPU 内存消耗情况,以此评估 CPU 压缩策略与 GPU 压缩策略在内存优化方面的性能差异。根据记录结果,绘制了 GPU 内存消耗图。

如图 14 所示,在使用 2bit 压缩算法时,SlimGC 与使用 CPU 压缩的方案的内存消耗相近,并且明显低于使用 GPU 压缩的方案所消耗的内存。具体来说,在 VGG-16、Inception-Bn 和 AlexNet 模型的训练中,SlimGC 相较于使用 GPU 压缩的方案分别减少了 8.9%、4.1%和 10.3%的内存开销。SlimGC 受益于压缩卸载策略可以在一定程度上减少对 GPU 内存的占用,并缓解压缩算法带来的残差累积机制所造成的内存压力。

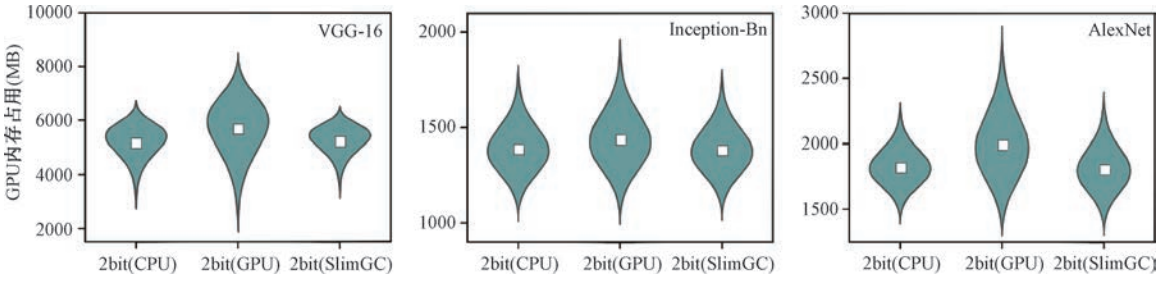


图 14　使用不同压缩策略训练各种模型时的内存消耗情况

5 总 结

我们面向分布式深度学习设计了 SlimGC 优化策略。它可以作为插件集成到主流训练框架中，并显著提升了多种梯度量化算法的收敛精度和训练速度。压缩优化策略利用小规模梯度的全精度通信，从而确保最佳的收敛精度和压缩收益。压缩卸载策略通过将压缩任务卸载到 CPU，有效地缓解了 GPU 上压缩操作和张量计算之间对计算资源的竞争，并缓解了残差累积机制导致的 GPU 内存压力。模型备份策略解决了工作节点对模型参数的读取依赖问题，从而使压缩卸载策略能够实现切实的性能优势。

SlimGC 的三种策略是相辅相成的。模型备份策略在压缩场景下可以发挥出比非压缩场景下更多的优势。因此，压缩场景下的压缩优化策略和压缩卸载策略才能够充分利用模型备份策略对模型训练进行精细化调整。我们对几种计算机视觉和语言模型的评估表明，SlimGC 比最先进的 1bit 量化方法高出 74.3%，同时在可比配置环境下，梯度量化方法的准确率也提高了 1.1%至 2.3%。

作者贡献声明 白哲、于恩达为共同第一作者。

参 考 文 献

[1] Tal Ben-Nun, Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. ACM Computing Surveys, 2018, 52(4): 1-43

[2] Enda Yu, Dezun Dong, Xiangke Liao. Communication optimization algorithms for distributed deep learning systems: A survey. IEEE Transactions on Parallel Distributed Systems, 2023, 34(12): 3294-3308

[3] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, et al. In-datacenter performance analysis of a tensor processing unit//Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto, Canada, 2017: 1-12

[4] Shaohuai Shi, Xiaowen Chu, Bo Li. MG-WFBP: Merging gradients wisely for efficient communication in distributed deep learning. IEEE Transactions on Parallel Distributed Systems, 2021, 32(8): 1903-1917

[5] Shaohuai Shi, Xiaowen Chu, Bo Li. Exploiting simultaneous communications to accelerate data parallel distributed deep learning//Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications, 2021: 1-10

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:.01274, 2015

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, et al. Pytorch: An imperative style, high-performance deep learning library//Proceedings of the 33rd International Conference on Neural Information Processing Systems Vancouver, Canada, 2019

[8] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, Gennady Pekhimenko. Priority-based parameter propagation for distributed DNN training//Proceedings of Machine Learning Systems. Stanford, USA, 2019: 132-145

[9] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, et al. A generic communication scheduler for distributed DNN training acceleration//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville, USA, 2019: 16-29

[10] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, Roy Campbell. Tictac: Accelerating distributed deep learning with communication scheduling//Proceedings of Machine Learning Systems. California, USA, 2019: 418-430

[11] Shuai Wang, Dan Li, Jinkun Geng. Geryon: Accelerating distributed CNN training by network-level flow scheduling//Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications. 2020: 1678-1687

[12] Jiayi Huang, Pritam Majumder, Sungkeun Kim, Abdullah Muzahid, Ki Hwan Yum, Eun Jung Kim. Communication algorithm-architecture co-design for distributed deep learning//Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture Valencia, Spain, 2021: 181-194

[13] Sanghun Cho, Hyojun Son, John Kim. Logical/physical topology-aware collective communication in deep learning training//Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture Montreal, Canada, 2023: 56-68

[14] Shuai Wang, Jinkun Geng, Dan Li. Impact of synchronization topology on DML performance: Both logical topology and physical topology. IEEE/ACM Transactions on Networking, 2021, 30(2): 572-585

[15] Zhaoyi Li, Jiawei Huang, Yijun Li, Aikun Xu, Shengwen Zhou, Jingling Liu, Jianxin Wang. A2tp: Aggregator-aware in-network aggregation for multi-tenant learning//Proceedings of the Eighteenth European Conference on Computer Systems. Prague, Czech Republic, 2023: 639-653

[16] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs//Proceedings of the Interspeech. Singapore, 2014: 1058-1062

[17] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, Animashree Anandkumar. signSGD: Compressed optimi-

sation for non-convex problems//Proceedings of the International Conference on Machine Learning, Stockholmsmässan, Stockholm SWEDEN. 2018: 560-569

[18] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, et al. Gradient compression supercharged high-performance data parallel dnn training//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. New York, USA, 2021: 359-375

[19] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes//Proceedings of the International Conference on Machine Learning. Long Beach, USA, 2019: 3252-3261

[20] Hyeontaek Lim, David G Andersen, Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning//Proceedings of Machine Learning Systems. Long Beach, USA, 2019: 53-64

[21] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, Amedeo Sapio. Efficient sparse collective communication and its application to accelerate distributed deep learning//Proceedings of the 2021 ACM SIGCOMM 2021 Conference. New York, USA, 2021: 676-691

[22] Zhuang Wang, Haibin Lin, Yibo Zhu, Ts Eugene Ng. Hi-speed dnn training with espresso: Unleashing the full potential of gradient compression with near-optimal usage strategies//Proceedings of the Eighteenth European Conference on Computer Systems. Prague, Czech Republic, 2023: 867-882

[23] Zhuang Wang, Xinyu Wu, Zhaozhuo Xu, Ts Ng. Cupcake: A compression scheduler for scalable communication-efficient distributed training//Proceedings of Machine Learning Systems. Miami, USA, 2023: 373-386

[24] Linnan Wang, Wei Wu, Junyu Zhang, Hang Liu, George Bosilca, Maurice Herlihy, Rodrigo Fonseca. FFT-based gradient sparsification for the distributed training of deep neural networks//Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing. 2020: 113-124

[25] Jianqiao Wangni, Jialei Wang, Ji Liu, Tong Zhang. Gradient sparsification for communication-efficient distributed optimization//Proceedings of the 32nd International Conference on Neural Information Processing Systems. Montréal, Canada, 2018: 1299-1309

[26] Hang Xu, Chen-Yu Ho, Ahmed M Abdelmoniem, Aritra Dutta, El Houcine Bergou, Konstantinos Karatsenidis, et al. Grace: A compressed communication framework for distributed machine learning//Proceedings of the 2021 IEEE 41st international conference on distributed computing systems. Washington, USA, 2021: 561-572

[27] Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, Dimitris Papailiopoulos. On the utility of gradient compression in distributed training systems//Proceedings of the Machine Learning and Systems. Santa Clara, USA, 2022: 652-672

[28] Zeqin Wang, Ming Wen, Yuedong Xu, Yipeng Zhou, Jessie Hui Wang, Liang Zhang. Communication compression techniques in distributed deep learning: A survey. Journal of Systems Architecture, 2023, 142: 102927

[29] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters//Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA, 2020: 463-479

[30] Yujun Lin, Song Han, Huizi Mao, Yu Wang, William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:.01887, 2017

[31] Lusine Abrahamyan, Yiming Chen, Giannis Bekoulis, Nikos Deligiannis. Learned gradient compression for distributed deep learning. IEEE Transactions on Neural Networks and Learning Systems, 2021, 33(12): 7330-7344

[32] Ahmed M Abdelmoniem, Ahmed Elzanaty, Mohamed-Slim Alouini, Marco Canini. An efficient statistical-based gradient compression technique for distributed training systems//Proceedings of Machine Learning Systems. 2021: 297-322

[33] Chia-Yu Chen, Jungwook Choi, Daniel Brand, Ankur Agrawal, Wei Zhang, Kailash Gopalakrishnan. Adacomp: Adaptive residual gradient compression for data-parallel distributed training//Proceedings of the AAAI conference on artificial intelligence. New Orleans, USA, 2018: 2827-2835

[34] Shigang Li, Torsten Hoefler. Near-optimal sparse allreduce for distributed deep learning//Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Seoul, Republic of Korea, 2022: 135-149

[35] Jiarui Fang, Haohuan Fu, Guangwen Yang, Cho-Jui Hsieh. RedSync: reducing synchronization bandwidth for distributed deep learning training system. Journal of Parallel Distributed Computing, 2019, 133: 30-39

[36] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding//Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach, USA, 2017: 1709-1720

[37] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning//Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach, USA, 2017: 1509-1519

[38] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel M Roy, Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd//Proceedings of the 34th International Conference on Neural Information Processing Systems. New Orleans, USA, 2020: 3174-3185

[39] Jiaxiang Wu, Weidong Huang, Junzhou Huang, Tong Zhang. Error compensated quantized SGD and its applications to large-scale distributed opti-mization//Proceedings of the International Conference on Machine Learning. Stockholm,

Sweden, 2018: 5325-5333

[40] Shaohuai Shi, Qiang Wang, Xiaowen Chu, Bo Li, Yang Qin, Ruihao Liu, Xinxiao Zhao. Communication-efficient distributed deep learning with merged gradient sparsification on GPUs//Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications. 2020: 406-415

[41] Zixuan Chen, Lei Shi, Xuandong Liu, Jiahui Li, Sen Liu, Yang Xu. OSP: Boosting Distributed Model Training with 2-stage Synchronization//Proceedings of the 52nd International Conference on Parallel Processing. New York, USA, 2023: 102-111

[42] Dayal Singh Kalra, Maissam Barkeshli. Why warmup the learning rate? underlying mechanisms and improve-ments//Proceedings of the 38th International Conference on Neural Information Processing Systems. Vancouver, Canada, 2024: 111760-111801

[43] Alex Krizhevsky, Geoffrey Hinton. Learning multiple layers of features from tiny images. Toronto, Canada: University of Toronto, Technical Report: TR-2009, 2009

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, USA, 2016: 770-778

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei. Imagenet: A large-scale hierarchical image database//Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition. Miami, USA, 2009: 248-255

[46] Karen Simonyan. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:.01274, 2014

[47] Pranav Rajpurkar, Robin Jia, Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. arXiv preprint arXiv:.03822, 2018

[48] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:.04805, 2018

[49] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:.03167, 2015

[50] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank//Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. Seattle, USA, 2013: 1631-1642

[51] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever. Language models are un-supervised multitask learners, Openai Blog, 2019, 1(8): 9

[52] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks 25//Proceedings of the 26th International Conference on Neural Information Processing Systems. Lake Tahoe, USA, 2012: 1097-1105

[53] Enda Yu, Dezun Dong, Yemao Xu, Shuo Ouyang, Xiangke Liao. CP-SGD: Distributed stochastic gradient descent with compression and periodic compensation. Journal of Parallel Distributed Computing, 2022, 169: 42-57

[54] Minghao Li, Ran Ben Basat, Shay Vargaftik, Chonlam Lao, Kevin Xu, Michael Mitzenmacher, Minlan Yu. {THC}: Accelerating Distributed Deep Learning Using Tensor Homomorphic Compression//Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation. Santa Clara, USA, 2024: 1191-1211

BAI Zhe, M. S. His research interests include distributed deep learning system and communication optimization.

YU En-Da, Ph. D. His research interests include distributed deep learning system and communication optimization.

DONG De-Zun, Ph. D. His research interests focus on high-performance network and architecture, datacenter and deep learning systems.

Background

The research presented in this paper falls within the domain of distributed deep learning, specifically addressing the challenge of communication bottlenecks in large-scale model training. In the era of big data and complex neural network architectures, the efficient synchronization of gradi-

ents across distributed systems has become a critical performance bottleneck. While significant strides have been made in optimizing collective communication operations, such as allreduce, the problem remains an active area of research with continuous advancements in both industry and academia.

This paper contributes to the field by designing a non-customized distributed deep learning train system, SlimGC, it which significantly enhances the training throughput of distributed deep learning(DDL) tasks We demonstrate that SlimGC can improve the train speed by up to 74.3% for 1-bit compression and 68.7% for 2-bit compression, while also achieving a convergence accuracy increase of 1.1% to 2.3% and reducing GPU memory consumption by 10.3%. These improvements are not only theoretical; they have practical implications for the deployment of DDL in resource-constrained environments.

Our work is part of a larger project aimed at optimizing distributed computing systems, which is funded by the National Key Research and Development Program of China (Grant No. 2022YFB4501702) and National Natural Science Foundation of China (NSFC) under Grant Agreements U24B20151. The significance of this project lies in its potential to democratize access to high-performance computing resources, enabling a broader range of institutions and businesses to engage in data-intensive tasks without the need for excessive infrastructure investment.

Besides SlimGC, the research group contributing to this study has a history of impressive work in the field of distributed systems and high-performance computing. Our previous research has explored various aspects of distributed training, including communication scheduling, priority-based parameter propagation, optimization of gradient compression and the development of generic communication libraries for deep neural networks. The outcomes of this paper are part of a comprehensive research plan that aims to advance the state-of-the-art in distributed deep learning. By improving gradient compression techniques, we are addressing one of the core issues that limit the scalability and efficiency of distributed training systems. We believe that our findings will contribute to the broader goals of the project and pave the way for more effective and accessible distributed computing solutions.