arXiv:2303.01675v1 [cs.DC] 3 Mar 2023

Ada-Grouper：面向微批次的自适应分组调度，在受抢占的网络中加速流水线并行

Siyu Wang Zongyan Cao ChangSi Lansong Diao Jiamang Wang WeiLin

阿里巴巴集团

摘要

流水线并行已被证明是一种卓越的方法，可在异构集群上提升对数十亿参数深度神经网络进行训练的吞吐量。1F1B 调度方案是一种广泛采用的内存和性能优化策略，它交替执行不同微批次的前向传播和反向传播阶段计算。另一方面，使用 1F1B 调度时的一个常见问题是，当网络资源被其他任务抢占时，数据传输会导致阶段计算延迟，即使阶段之间的通信量已降至最低也是如此。在云服务中，无法保证独占访问这些网络资源。我们提出了一种通用调度技术，可通过承受一定程度的内存压力，使流水线并行适应被抢占的网络环境。其核心思想是将 1F1B 调度方案扩展为 kFkB，即将 k 个微批次分为一组，并交替执行k 次前向传播和反向传播计算。我们提出了 Ada-Grouper，这是一种自适应kFkB 调度器，会定期调整组内成员数量k，在内存限制下，随着网络环境变化，在通信效率与计算效率之间维持最佳平衡。实验结果表明，与受抢占网络场景下的 1F1B 相比，我们的设计能保持流水线并行性能稳定，并将性能提升 4% 至 30%。

1 引言

深度神经网络（DNN）的参数数量迅速增长。在自然语言处理领域，GPT-3 [2]拥有超过 1750 亿个参数，而 PaLM [5] 由 5400 亿个参数组成。在多模态预训练领域，M6 [19]拥有超过 1000 亿个参数。近期研究已证明，大型模型在极其复杂的任务中展现出前所未有的性能，这进一步刺激了对更大模型规模的需求。为了有效训练这些大型模型，分布式策略必不可少。这需要将模型或训练数据划分到不同的设备上，并行执行。

目前已有多种用于训练深度神经网络（DNN）的分布式策略，例如流水线并行 [9], ，它对于在互连较弱的异构集群上进行训练尤其有益（例如，通过 NVLink 连接的图形处理器，以及通过网络连接的机器）。这种策略将深度模型的运算划分给不同的工作器（或阶段），并在它们之间进行点对点通信。批次数据被划分为较小的微批次并注入流水线，而流水线并行的调度方案对于最大化吞吐量和节省内存至关重要。PipeDream [22] 和 DAPPLE [6] 采用 1F1B 调度方案，通过交替执行不同微批次的前向传播和反向阶段，在保持峰值内存消耗恒定的同时最大限度地提高设备利用率。与流水线并行不同，单程序多数据并行 [15] [35] 会沿一个或多个维度将运算划分到不同设备上，并插入集合通信以保持数学等价性。现代分布式策略在将大型模型扩展到数千个设备时，通常会结合使用这些并行方式。

在专用集群上训练大型模型时，流水线并行可以实现理想性能。然而，在使用云平台时，采用流水线并行的最先进分布式策略面临一个挑战：由于基于云的网络的特性，阶段之间受抢占的网络资源会显著降低吞吐量。其专用集群不利于充分利用资源，而且难以在软件层面为特定训练任务隔离网络。此外，输入训练数据通过与训练过程本身重叠的网络进行传输，进一步抢占了由流水线并行化引入的通信。因此，即使阶段之间的数据传输量可能很小，在云平台上常见的受抢占环境中，仍很可能出现大量跨阶段通信。

我们分析发现，通过调整调度方案，可以提升抢占式网络环境中的吞吐量。由于前向阶段和反向阶段都将其依赖阶段产生的激活值作为输入，因此每个阶段的开始时间可能受其输入到达时间的影响。默认的 1F1B 调度器交替调度一个前向阶段和一个反向阶段来编排执行。当交替的前向传播或反向传播因输入传输缓慢而停滞时，就会形成瓶颈，从而引入更多气泡时间。为减少这一问题，应扩展 1F1B 计划以适应不稳定网络。通过调度另一个输入已就绪的阶段，而非等待输入受阻的阶段，可以使计算与数据传输重叠，从而提高设备效率。为此，我们提出 kFkB 调度方案，其中，将 k 个微批次的前向阶段和同样数量的反向阶段作为一个调度单元。与 1F1B 计划相比，它交替调度 k 个前向阶段和 k 个反向阶段。不同调度单元之间仍遵循反向优先原则，并且 $k = 1$ 是kFkB 的特殊情况。尽管随着 k 值的增大，在拥塞网络中通信和计算重叠的潜力会提高，但代价是内存负担增加。当将k 增大至微批次数量时，该调度方案等同于 GPipe 的方案，无法在设备上提前释放内存。因此，在微批次大小固定的情况下，选择适当的k 值，可使每个设备保留足够的计算任务以防止通信造成的阻塞，同时在峰值内存使用量与计算性能之间进行权衡，因为 1F1B 有助于提前反向调度。此外，模型用户通常提供全局批次大小，而划分出的微批次大小可以变化。因此，应针对每种微批次大小搜索最优的 k 值。然而，较小的微批次大小可能导致较低的计算效率，尽管较大的 k可能有利于通信和计算重叠，却仍可能降低端到端性能。此外，当网络流量波动时，优异性能可能无法维持。必须根据网络流量的波动定期调整调度方案。

我们提出了 Ada-Grouper，这是一种遵循以下指导原则设计的自适应调度器：

▷ Ada-Grouper 作用于由阶段计算实例组成的任务图。在我们的系统中，每个阶段计算都从 HLO 模块中划分出来 [8],HLO 模块是 XLA 使用的编译器中间表示（IR） [34]。每个阶段计算都会经过编译，并用于创建称为任务节点的执行实例；Ada-Grouper 会调度这些任务节点在集群上执行。

▷ Ada-Grouper 采用 kFkB 调度，将 k 个微批次的前向阶段或反向阶段作为一个调度单元，并在每个阶段计算输出结果后立即触发跨阶段通信。这种设计提供了更多重叠计算与通信的机会，同时通过在内存不再被引用时立即释放其占用空间，降低峰值内存使用量。

▷ Ada-Grouper 会定期自动调整k，并认识到较大的 k 需要更小的微批次大小，这可能导致设备利用不足。考虑到云平台中的计算效率以及网络频繁发生的抢占变化，从头到尾保持固定的 k 并不现实。因此，Ada-Grouper 会针对不同的k 值生成多个版本的调度方案，并在线自动调整以选择最佳版本。

Ada-Grouper 基于 Rhino 实现， [39],Rhino 是一个自动并行化系统。我们的实验表明，该方法能够适应抢占式网络环境，并实现稳定的性能。与 1F1B 调度方案相比，Ada-Grouper 可使 GPT 和 UNet 的训练速度提升 4% 至 30%。

2 背景与动机

2.1 大模型并行化

现代深度学习框架将深度学习计算过程视为数据流图，[1, 8, 25], 模型开发者通过使用细粒度算子构建模型，这些算子表示为节点，并以多维张量作为边相连。为便于优化，一些框架提供了更定制化或通用的中间表示，例如 XLA 中的 HLO、 [34],JaxPR 中的表示 [8] 以及 MLIR， [14],这些表示支持在编译阶段进行通用且系统化的分布式优化。 [6, 15, 35]编译完成后，执行引擎会在集群上执行该计划。

当大型模型因内存限制或性能要求而无法在单个设备上完成训练时，可采用分布策略对模型进行并行化。数据并行 [30]是大多数深度学习框架中最容易实现的模式，它将训练数据拆分到多个设备上，并在更新参数前进行梯度归约。算子并行 [15, 20, 35] 是对数据并行的推广，允许按任意算子进行拆分，并需要通过重分片通信来保证计算正确。数据并行和算子并行都属于算子内并行 [40]。流水线并行 [6, 9, 22] 是一种算子间并行，它将模型图中的操作划分为若干阶段，并将这些阶段部署在不同设备上。所有这些并行化技术都是为了最大限度地提高硬件资源利用率。

2.2 流水线并行

流水线并行在由低带宽网络连接的异构集群上表现出色。在实践中，应找到切分点，在保持各阶段计算均衡的同时，尽可能减少跨阶段通信。在流水线并行中，可通过产生的气泡数量来衡量吞吐性能。对于同步训练，增加微批次数量是减少气泡的常见方法，尽管无法将气泡完全消除。从理论上讲，吞吐量最终将收敛至上界 [6]。内存限制也是一个重要因素，它限制了预热阶段可投入的微批次数量。借助梯度检查点等内存优化技术 [4, 7]，可以向流水线中投入更多微批次，进一步减少气泡。

2.3 提前反向调度

流水线并行中的调度至关重要，必须经过精心设计。以往研究表明，调度会影响吞吐量和峰值内存使用量。GPipe 按顺序注入微批次，这会导致首阶段设备的内存使用量过高，因为每个微批次前向阶段产生的激活值都必须保留，直到其反向阶段完成。近期研究依据尽早反向传播原则，改进了同步 [6] 和异步训练 [22],中的调度，使其形成 1F1B顺序。此举旨在尽早释放各阶段的前向传播激活值，从而容纳更多微批次或更大的微批次大小。研究表明，在稳定且网络独占的环境中，1F1B 可确保气泡数量保持不变，同时在流水线的稳定阶段维持峰值内存使用量不变。然而，网络不稳定时，气泡数量会大幅增加。

2.4 HLO 和任务图

为了支持不同的前端平台和后端硬件，人工智能编译器（例如 XLA [34] 和 TVM [3]）通常会为张量程序引入与平台和硬件无关的中间表示。具体而言，XLA编译器会将用户定义的模型转换为 HLO中间表示，并执行优化遍。在我们先前的工作 [38],中，我们开发了一种自动并行化遍，用于探索包括流水线并行在内的分布策略。该遍通过最小化跨阶段通信成本，分析以 HLO中间表示格式表示的计算图，并将 HLO图分解为阶段计算。我们将每个 HLO 阶段计算视为一个程序，并创建其对应的运行实例，称为任务节点。由不同微批次输入、但属于同一阶段的任务节点共享相同的 HLO 阶段计算，并创建梯度累积任务节点来衔接不同的微批次。所有任务节点均根据数据依赖关系相互连接。任务图会自动构建，用于描述原始深度模型的分布策略。为适配设备分配，必须插入 Send/Recv对。为此，我们构建一种特殊类型的任务节点，并将其自动插入任务图中，用于表示点对点通信中的发送和接收操作。在我们的工作中，调度计划由任务图生成。由于任务图的粒度相对较粗，因此能够显著减少生成调度计划所耗费的时间。

2.5 机遇

为加速同步流水线并行训练，现有方案侧重于提升专用网络中的性能，但由于公有云服务中资源碎片化，这些方案在云平台上不可行。 [6, 26, 31]预留一个网络隔离的专用集群成本高昂，也不适合工业应用。因此，分布式图形处理器任务很可能被分配到具有抢占式网络环境的机器上，其中还包括数据传输或其他非图形处理器分布式任务。这可能导致性能下降，因为网络资源被其他任务占用，从而延迟跨阶段通信。

现有的 1F1B 调度在专用集群上表现出色，但在受抢占的网络环境中其性能极差。由于网络资源被其他任务占用，它会引入更多气泡，导致跨阶段通信变得不可忽略。图 2(a) 表明，由于不可忽略的跨阶段数据传输，每个阶段计算的开始时间可能会延迟，进而导致流水线稳定阶段中的气泡时间增加。这些气泡是由调度尚未就绪、正在等待来自网络输入的阶段所造成的，即使注入更多微批次也无法消除。先前的文献提出，可以将数据传输视为一个独立阶段 [6]，并据此重新规划调度和注入的微批次数量。然而，这种方法并不实用，因为阶段之间的网络资源可能存在很大差异，使得难以通过 1F1B 调度隐藏通信，从而产生更多气泡。

现有方法未能考虑网络资源使用情况会实时变化这一动态特性。因此，固定调度规划无法适应不断变化的环境。例如，如果两个阶段之间的网络资源周期性地被其他任务占用，那么即使将数据传输阶段视为不可忽略且独立的阶段，固定调度方案也可能无法满足需求。此外，不同阶段之间网络资源使用情况的变化使得未来规划更加困难。

如果回到 GPipe 的 1F1B 调度情形，我们推测并观察到，在受抢占的网络中，前向阶段和反向阶段都能更高效地与通信重叠。依次注入前向阶段和反向阶段，可以创造更多重叠机会。例如，对于前向阶段，阶段$n-1$中的微批次s可以与阶段n中微批次$s+1$的输出传送重叠，当$n>1$时。因此，与 1F1B 调度相比，气泡时间得以减少。然而，GPipe 的调度可能会导致更高的峰值内存使用量。

在受抢占的网络环境中，将调度方案调整为kFkB模式时，流水线效率与峰值内存使用量之间存在权衡。因此，我们主张以更通用且自适应的方式调整调度方案，在计算效率与内存使用量之间取得适当平衡。

2.6 挑战

探索机遇也伴随着挑战。

评估候选方案所需的时间。对于kFkB调度，k 个前向传播或反向阶段可作为基本调度单元。必须确定组成员数量k，但在设备内存有限的情况下，该数量也可能受到微批次大小b 的影响。流水线长度应根据阶段执行时间和网络剖析结果进行估算。由于存在许多可行组合，在不断变化的网络环境中，若每个阶段之间都需要进行性能分析，那么评估流水线长度所耗费的时间将无法接受。显然，有必要对候选集进行剪枝以缩短评估时间。为解决这一问题，我们设计了一种高效算法，用于有效剪除帕累托最优前沿上的候选集。

准确预测流水线长度的执行时间存在困难。为了估算流水线长度，我们需要对阶段执行时间和跨阶段通信时间进行性能分析。阶段执行时间可以准确地进行性能分析，而由于主机之间的网络占用率不断变化，且网络占用率对时间敏感，因此跨阶段通信时间更难以预测。此外，稳定的网络环境与数据大小之间并不成比例。进一步说，即使网络稳定，跨阶段通信时间也不会与数据大小成比例。为应对这些挑战，我们提出了一种自适应实用剖析策略，以适应不断变化的网络环境。

3 概述

Ada-Grouper 是基于我们先前的工作 Rhino 开发的独立组件 [38]。如图1所示，Ada-Grouper 的端到端高级工作流包含一个*Ada-Grouper* 处理遍和一个*Ada-Grouper* 调度器。Rhino 的自动并行化遍以用户模型

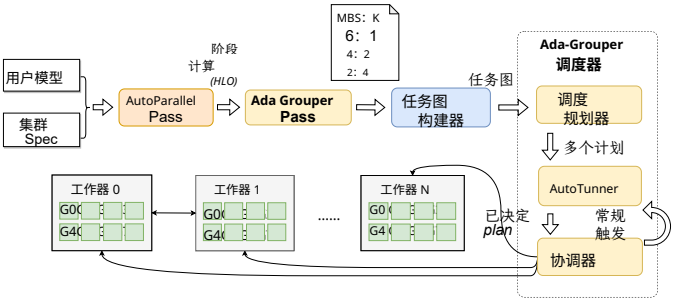


图 1：Ada-Grouper 工作流程

和集群规格作为输入，以探索单程序多数据和流水线并行化空间，并生成一组 HLO 阶段计算，这构成了我们工作的基础。借助 Ada-Grouper，k 个前向传播阶段或反向传播阶段会被组合成一个组，并作为不可分割的单元进行调度。为了在内存限制允许的各种微批次大小下确定组成员数量的最优值k，需要枚举所有可能的解。因此，Ada-Grouper 处理遍会搜索多个微批次大小 (b) 和组成员数量 (k) 的组合。它还提供了一种高效的剪枝算法，用于枚举所有潜在的最优候选项。

接下来，Rhino 中的任务图构建器会编译给定候选集合中的所有阶段计算，生成相应的执行二进制文件，随后根据其指定的组成员数量和微批次大小组合，构建多个版本的任务图。*Ada-Grouper* 调度器 将这些任务图作为输入，并进行自适应调度。具体而言，调度规划器 会根据每个任务图的组成员数量（k）和微批次大小（b），生成kFkB 个调度方案，作为候选计划。自动调优器 会评估所有候选计划，并使用成本模型选出最优方案；该成本模型通过对网络进行性能分析并计算各阶段的执行时间来估算流水线长度。其他候选计划则会保留，以供后续选择。然后，协调器根据已确定的计划协调不同工作器。为了适应不断变化的网络环境，Ada-Grouper 会定期触发自动调优器 重新评估每个候选方案，从而选出未来的最优计划，但其k 值可能不同。计划确定后，在通知策略变更时会立即生效。

3.1 Ada-Grouper 处理遍

Ada-Grouper 处理遍支持在不同的k 值下选择多个候选项，因为在动态网络环境中，每个候选项都有可能被替换。需要注意的是，1F1B 的内存效率最高，因为尽早反向传播会促进相应前向传播激活值的提前释放。

激活值 [6],，而更大的k 值会消耗更多内存。为确保生成的候选项不超出内存限制，在当前阶段枚举k 时，会估算由组数和微批次大小组合生成的每个计划的k 值总是与更小的b配对。尽管较小的微批次大小可能与较大的组数相结合，从而更好地与通信重叠，但其计算效率并不更高。这一计算效率与运行时流水线吞吐量之间的权衡，由为此处理遍设计的有效搜索算法纳入考量，从而能够生成所有必要的潜在候选项。

3.2 Ada-Grouper 调度器

3.2.1 调度规划器

调度规划器接管任务图构建器构建的所有任务图，并制定kFkB计划。利用尽早反向传播原则，规划器会尽早调度一个反向传播组，以尽可能减少峰值内存使用量。kFkB算法作用于任务图的每个版本，永久保留所有任务图及其对应的调度方案，而不决定执行哪一个计划。

3.2.2 自动调优器与协调器

自动调优器 会定期选定一个组调度计划，以适应不断变化的受抢占网络。计算效率与流水线吞吐量之间的权衡意味着每种方案都有可能成为最优方案。为此，系统设计了一种简单的成本模型，定期评估当前网络条件下的流水线长度。该成本模型需要阶段计算时间和跨阶段通信时间的性能分析数据，这些数据需要多次采集，才能得到更可靠的结果。需要注意的是，由于计算设备不会遭受抢占，训练期间的计算时间开销保持稳定，因此其性能分析数据可以重复使用。然而，网络抢占使得重新评估成为必要。协调器 在运行时被设置为并行执行，通过多线程在各设备上调度任务节点，并将新选定的调度方案分发给所有参与工作节点。自动调优器选出合适的计划后，协调器将修改调度方案并立即使其生效。

4 组调度计划

4.1 kFkB 流水线长度

我们揭示了受抢占的网络环境中kFkB 与 1F1B 调度方案的流水线长度对比。我们作出两个合理的假设：（1）后向阶段的时间开销约为前向阶段的两倍；（2）跨阶段传输时间为前向阶段计算时间的一半。

图 2（a）表明，1F1B 调度计划在这种情况下显著降低了性能。阶段之间数据传输所需的时间不可忽略。从理论上讲，1F1B 计划引入的气泡比例高于独占网络环境中的比例。相比之下，图 2 显示，kFkB 计划在 $k > 1$ 时具有更短的流水线长度。

从分析上看，在 1F1B 场景中，当网络被占用时会产生更多不可忽略的气泡。这种网络环境中的跨阶段通信阻碍了任务调度。只要网络抢占不严重，这种恶化现象就能够得到缓解。然而，在云平台上，实现理想环境的情况十分罕见。当将k 从 1 增加到 2 时，整体吞吐量显著提升。进一步观察kFkB 流水线可以发现，在将两个前向阶段或两个反向阶段合并为一个调度单元后，第二次计算会与第一次计算之后的跨阶段通信重叠，从而通过引入另一个就绪任务，避免每个设备因跨阶段通信而被阻塞。计算与通信重叠的机会越多，消除的气泡就越多。我们可以推测，当将k 从 2 增加到更大值时，气泡比例还会进一步降低。

另一个需要注意的问题是，增大 k 值会对内存使用量产生副作用。如果将 k 设置为 M（微批次的数量），调度方案将恢复为 GPipe [9]的情况。这会给第一阶段所在的设备带来巨大的内存压力，因为其激活值几乎必须在整个训练迭代期间都保存在内存中。缓解这一问题的一种潜在方案是沿批次维度进一步划分微批次。然而，这可能会降低计算效率，因为微批次大小会变小。

4.2 候选集剪枝

为了生成kFkB调度方案，我们应生成所有可用的解候选项。kFkB调度方案与两个变量有关：以组成员数量k衡量的基本调度单元，它为计算和跨阶段通信的重叠提供了机会；以及微批次大小b，它是计算效率的关键。原则上，只要满足内存约束，该解就可用并会被纳入候选集。随后，Ada-Grouper会评估每个调度方案的流水线长度，并选出最佳方案。然而，由于需要进行网络剖析，评估过程非常耗时。例如，微批次大小不同的两个调度方案，其通信性能可能

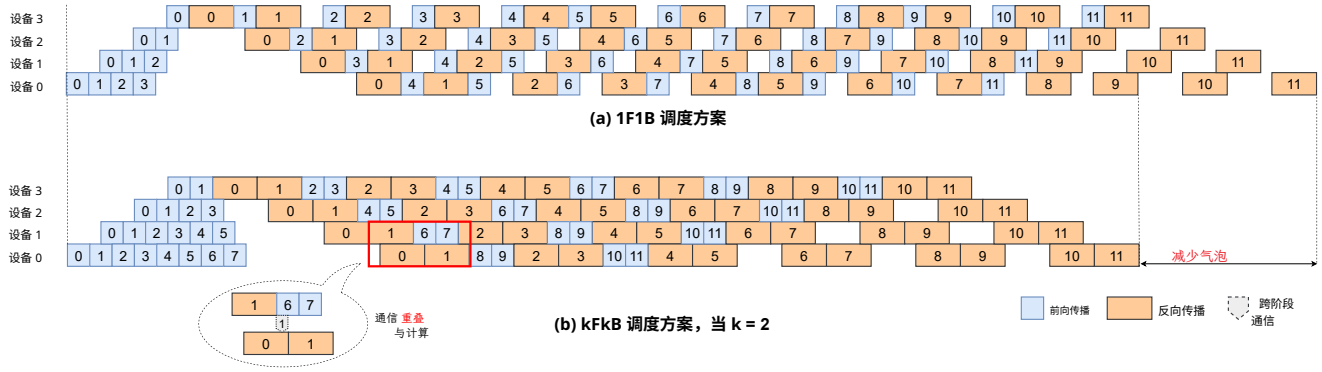


图 2：抢占式网络环境中 1F1B 和 2F2B 调度方案的流水线长度分析。

截然不同。此外，还应在不断变化的网络环境中对通信性能进行剖析。如果评估时间过长，网络环境很可能已经发生变化，导致评估结果失效。为了解决这一问题，我们设计了一种剪枝算法，可大幅缩减候选集，从而减轻运行时的评估压力。

图。3 展示了我们的候选集，本质上它就是帕累托最优前沿。增加组成员数量（k）或微批次大小（b）会消耗更多内存，直到达到设备内存上限。固定 k 后，我们可以轻松获得最大微批次大小，这些组合构成了内存限制曲线。内存限制曲线上或曲线下的所有组合都是可行解，但应剔除内存利用不足的解。内存限制曲线之上的点B 因不满足内存约束而不可选。然而，如果我们选择阴影区域内的点A这样的组合，设备内存就会利用不足。因此，我们只需关注内存限制曲线上的点，例如 C，这有助于大幅缩减候选集。具体来说，我们可以从 1 开始逐步增加组成员数量 k，然后贪心地搜索可接受的最大微批次大小。

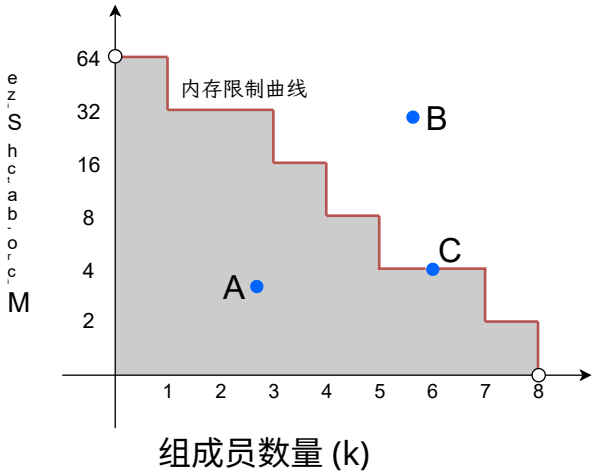


图 3：在内存限制曲线上生成候选项。解 A 所在的阴影区域是可行的，但内存利用率较低。解 B 所在的区域会导致内存溢出。内存限制曲线上所有整数的组合都是我们的候选项。

4.3 成本模型

Ada-Grouper 使用成本模型，该模型需要阶段计算和跨阶段通信的性能分析数据，以估算所有可用调度方案的流水线长度。由于阶段计算专属分配给指定设备，其成本稳定，且在训练过程中无需重新评估。尽管如此，仍应定期监测网络性能，因为它往往会发生波动。因此，应在一段时间内多次进行性能分析，并将这些结果的移动平均值作为最终结果。

我们不通过测量阶段之间的带宽来估算数据传输的时间开销，而是直接测量跨阶段通信时间。采用这种方法有两个原因：首先，尽管阶段之间传输的数据量相同，但性能可能会因抢占程度不同而

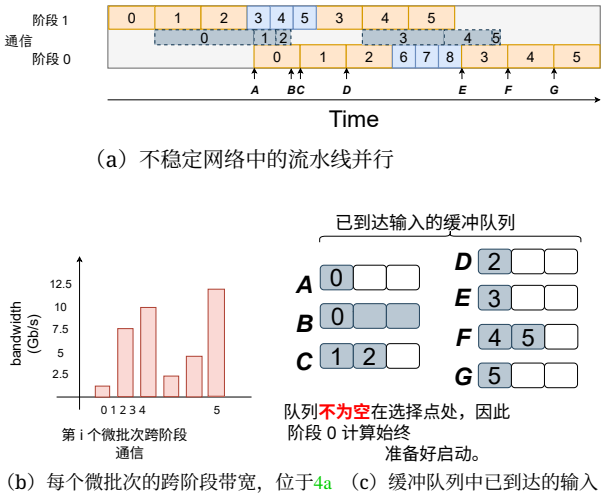
显著变化；其次，不同形状和大小的张量，其带宽利用率也不相同。因此，对端到端跨阶段通信进行性能分析，能够提供更可靠的数据，效率也更高。此外，每个调度计划的性能分析数据都应按一定时间间隔在一个窗口内取平均值。

4.4 不稳定网络中的分析

我们使用 3F3B 调度方案来说明不稳定网络下性能的稳定性。如图 4a所示，3F3B 流水线会受到网络不稳定性的影响。微批次 0、1 和 2 执行反向传播，导致通信从阶段 1 传向阶段 0。阶段 1 与阶段 0 之间带宽的突然波动，导致反向传播计算流程中每个微批次的通信质量较差且不稳定。图 4b 展示了跨阶段

两个阶段之间每个微批次的通信带宽。微批次 0 的带宽较低，延迟了阶段 0 中相应阶段计算的开始时间。然而，微批次 3 的网络状态也较差，但并未对其后续阶段计算产生负面影响。

为了解释这一现象，假设有一个缓冲队列，用于存储从阶段 1 到阶段 0 的跨阶段通信数据。当阶段 0 中的一次计算完成后，其输入的存活性随之结束，因此可以将这些输入从队列中弹出。于是，缓冲队列会存储当前时刻可供后续阶段计算使用的就绪输入。为了确保计算在准备启动时能够立即进行而不被推迟，队列必须非空。无论传输时间多长，输入都会提前发送。图 A至G）中的七个点被选出，用于检查队列状态。除点 B外，所有选定点均对应计算启动的时刻，且其队列状态4c 均非空，这表明所需输入已提前到达。至于点B，由于微批次 0 的计算尚未完成，队列已满，因此无法弹出其输入。最值得关注的是点 E。尽管其输入传输受到网络抢占的影响，导致传输时间变长，但与点 A的输入类似，其计算并未推迟。在 3F3B 调度中，点E 是阶段 0 启动微批次 3 的反向传播计算的最早时机，具体取决于其输入是否已提前到达。我们发现，点 E 对应的队列阶段并非空队列，其输入已经在队列中。这正是网络再次遭到抢占时，流水线未产生额外气泡的根本原因。总体而言，当网络处于高速状态时，更多到达的输入会被推入队列，为网络未来遭到抢占时提供充足的待处理资源。这一优势使流水线系统即使在极不稳定的网络状态下，也能保持良好的吞吐性能。



（a）不稳定网络中的流水线并行

（b）每个微批次的跨阶段带宽，位于4a （c）缓冲队列中已到达的输入

图 4：不稳定网络中的 3F3B 性能分析。

5 实现

我们基于先前的工作 Rhino 设计并实现了 Ada-Grouper。Rhino 可在 HLO 模块上并行化用户模型，以构建任务图并在分布式设备上进行调度。Ada-Grouper 处理遍的核心是，

位于 AutoParallel 过程之后（如图1所示），这是一种有效的枚举算法，可生成充足的候选方案。Ada-Grouper 调度器中的调度规划器由 1F1B 调度器修改而来。我们生成k 份 1F1B 调度序列，并将它们交错排列，构建一个kFkB 计划，将所有可用计划保存下来以供后续选择。成本模型在自动调优器和协调器中发挥着重要作用，它通过可用的分析器网络定期准确评估流水线长度。最后，跨阶段点对点通信应当启动异步启动，且不应阻塞设备或主机。我们为 NCCL 原语实现了专用任务节点（Send/Recv对） [10]，用于处理跨阶段点对点通信。

第5.1 节介绍如何在 Ada-Grouper 处理遍中枚举候选项。第5.2 节讨论成本模型的细节和性能分析策略。第5.3 节展示了我们用于点对点通信的异步设计。第5.4 节简要介绍kFkB 调度和在线调优实现。

5.1 Ada-Grouper 处理遍

我们开发了 Ada-Grouper 处理遍，该处理遍位于 AutoParallel 过程之后，用于枚举并裁剪候选方案集合。作为输入，该处理遍接收已标注批量大小维度并自动分解为各阶段的阶段计算。在枚举过程中，它通过算子拆分推理沿批量大小维度拆分阶段计算，同时利用 XLA 中的缓冲区分配模块准确估算精简 HLO 计算的内存使用量。确定 k 和 b 后，相应的调度计划也随之确定，并使用简单的内存成本模型进行评估，以确保其满足内存限制约束，同时考虑每个阶段计算输出的存活性。

5.2 性能分析

阶段执行时间。 我们针对每个调度方案分别分析阶段执行时间。为确保获得更可靠的分析数据，需要对每个阶段计算进行多次测量并计算平均值。由于所有分配的设备均保证独占，因此无需重新分析所有阶段执行

时间在在线调优阶段。

跨阶段通信时间。 我们针对当前网络环境下的所有调度方案，对各阶段之间的跨阶段通信时间进行分析。与阶段执行时间类似，每项跨阶段通信时间也应进行多次分析并取平均值。不过，由于网络环境可能发生变化，因此在在线调优阶段需要重新分析。为确保分析数据不受我们自身任务的干扰，我们会暂停当前调度任务，并收集每种调度方案的所有性能数据。

5.3 异步点对点通信

该 *k*F*k*B 调度方案受益于跨阶段通信与阶段计算的重叠，因此需要采用异步点对点（p2p）通信。为此，我们使用 NCCL 原语设计并实现了点对点通信，并遵循三项原则。首先，发送内核和接收内核应在独立的流上启动，且这些流应与用于计算的其他所有内核流相互区分。其次，通信双方的发送和接收操作必须在不同设备间正确配对，不得出现不匹配，否则可能导致死锁或不可预测的行为。最后，当不同的参与方配对在同一设备上工作并沿同一方向进行点对点通信时，应复用已创建的通信器。此策略可避免重复创建通信器。图 5 展示了 Ada-Grouper 中的点对点通信设计，其中发送、接收和计算内核分别运行在不同的流上。发送内核与接收内核之间的配对通过各设备上的调度顺序建立。此外，*R*1 在 *R*0 完成后启动，以确保数据传输不对齐；*R*3 和 *R*4 也采用类似方式。

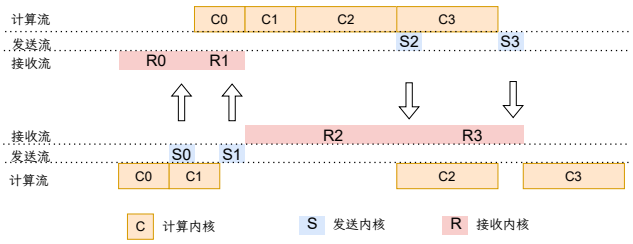


图 5：Ada-Grouper 异步点对点通信

5.4 kFkB 调度与在线调优

kFkB 调度。 实现了启发式 1F1B 调度算法，生成 1F1B 计划的 k 个副本。随后将这 k 个计划交叉合并，构建合并后的计划，

表 1：GPT 模型测试配置。不同配置用于在 1/2/4/8 个工作节点上进行弱扩展测试。“GPT-Medium”用于强扩展和粒度测试。强扩展和弱扩展的全局批次大小设为 64，粒度测试的全局批次大小设为 192。所有测试均采用 float16 精度。

配置名称	N_{params}	N_{layers}	D_{hidden}	D_{ffn}	N_{heads}	D_{head}
GPT-Medium	350M	24	1024	4096	16	64
GPT-Large	760M	24	1536	6144	16	96
GPT-XL	1.3B	24	2048	8192	32	64
GPT-2.7B	2.7B	32	2560	10240	32	80

表 2：U-Net 模型测试配置。两种配置均用于弱扩展（按样本数）测试。所有测试均使用单精度浮点。误差线显示不同步骤的性能变化范围。

配置名称	N_{params}	N_{dims}	$D_{image\ size}$
UNet-Base	32M	64	32
UNet-Medium	768M	320	32

并进行调整以确保发送/接收任务节点配对。随后，Ada-Grouper 处理遍会生成一组可用的 *k*F*k*B 调度方案，这些方案保存在 Ada-Grouper 调度器中，并通过在线调优定期重新评估。

在线调优。在线调优过程由环境变量控制，并按固定间隔触发；它会重新分析跨阶段通信时间，并评估每种调度方案的流水线长度，以确定最优方案。由于微批次大小或组成员数量的变化不会影响模型参数，因此在不同调度方案间切换时，无需将可变缓冲区转存并在新方案中恢复，从而将切换开销降至最低。在线调优作出决策后，协调器会立即分发最佳调度方案。

6 评估

Ada-Grouper 旨在通过*k*F*k*B 流水线方案实现自适应微批次调度，从而在生产环境中的网络状况变化时，使阶段数据传输更加顺畅。因此，我们选择了三个用途各异的平台，来测试 Ada-Grouper 对流水线并行模型训练工作负载的影响。事实上，要定量、精确地展示实时网络状况并不容易，因此多次迭代步骤的用时预计会在一定范围内波动。如果测试恰好在平台相对繁忙或空闲时进行，性能也会受到影响。

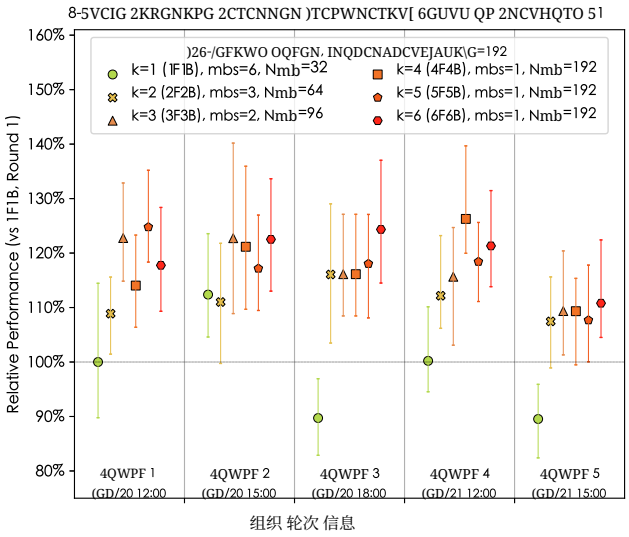


图 6：在采用 GPT-Medium 配置的 Plat- form S1 的 8 个工作节点上进行的流水线粒度测试。所有测试均使用固定的全局批次大小 192，微批次大小（mbs）和数量（N_{mb}）则根据不同的组调度数量而变化k （从 1 到 6），kFkB。误差棒显示了不同步骤的性能变化范围。

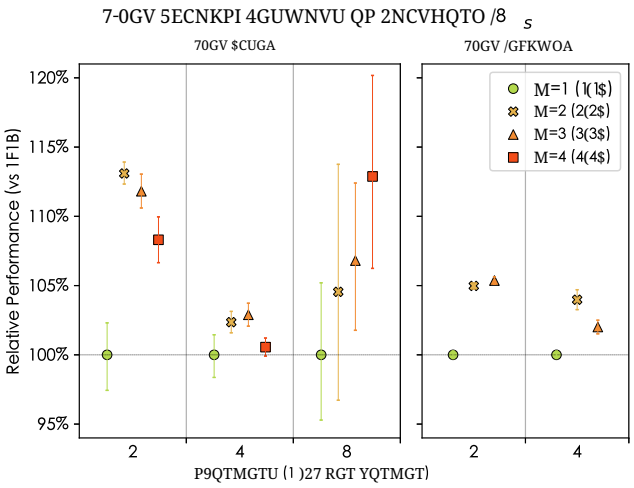


图 7：U-Net 模型上不同kFkB 的相对性能比较，在平台 M8$_s$ 上按全局批次大小进行弱扩展。测试中，全局批次大小设置为 $N_{workers}$*128。误差线表示不同步骤的性能变化范围。

6.1 平台与模型

测试平台。这三个平台均配备了典型的商用硬件配置。需要注意的是：1）S1 和 M8$_s$ 中的节点连接到同一网络，该网络承载着实际生产作业；2）C$_{1x}$ 和 M8$_s$ 上的工作器可能会与其他工作器共享相同的物理机器资源，这些工作器属于其他作业，也可能偶尔属于同一作业。这意味着有时会出现无法预料的资源使用冲突或网络争用，这显然会影响测试性能。不过，我们认为这些情况能让测试结果更贴近真实世界的作业。

- **平台 C1$_x$：**：一个用于内部测试的云资源池。配置：（每个实例）1 个 V100-SMX2-32GB GPU、12 个 Xeon（Skylake）2.7GHz 虚拟 CPU 核心、96GB DDR4-2666 虚拟内存、25Gb 虚拟以太网。
- **平台 S1：** 一个专用于在线开发的平台。配置：（每台机器）1 个 V100S-PCIE-32GB GPU、2 个 Xeon（Cascade Lake）26C 2.5GHz 处理器、512GB DDR4-2666、1 个跨越生产环境交换机的 100Gb RoCE 互连。
- **平台 M8$_s$：** 一个在线预生产平台。配置：（每台机器）8 块带 NVLink2 的 V100-SMX2-32GB GPU、2 颗 24 核 2.5GHz Xeon（Skylake）CPU、768GB DDR4-2666，以及 1 个通过生产环境交换机连接的 100Gb RoCE 互连。

所有机器均安装了 Linux 操作系统，内核版本为 4.19.91，Nvidia 驱动程序版本为 470.82，NCCL 库版本为 2.8。测试在配备 CUDA 10.1 运行时和 Rhino 并行系统的开发用 Docker 中运行 [38]。

基准模型。Ada-Grouper 评估实验选用了两个当前常用的基础模型。第一个是用于自然语言处理任务的 GPT [2] 模型，另一个是用于 [28] 文本到图像扩散模型的 U-Net。两个模型的评估配置列于表1 和表2。

基线。实验基于相同的并行训练系统 Rhino [38]，在上述平台与模型上将 Ada-Grouper 与 1F1B 流水线调度结果进行比较。所提出的测试重点考察每个工作器使用 1 个图形处理器的情况，这与云环境或多任务在线生产系统的情形更为相似；在这些场景中，通常会有零散资源可供分配给任务。

6.2 结果

6.2.1 流水线粒度测试

在平台 S1 的 8 个工作节点上采用流水线并行，对 GPT-Medium 模型配置进行了流水线粒度测试。对于 Ada-Grouper 的kFkB 组调度计划，我们认为每个训练迭代步骤中应激活 k 倍数量的微批次。因此，这些粒度测试专为内存受限模型设计，以验证是否可以采用更细粒度的微批次拆分，使kFkB 保持最大内存使用量。

测试将全局批次大小设为固定值 192，并进行调整

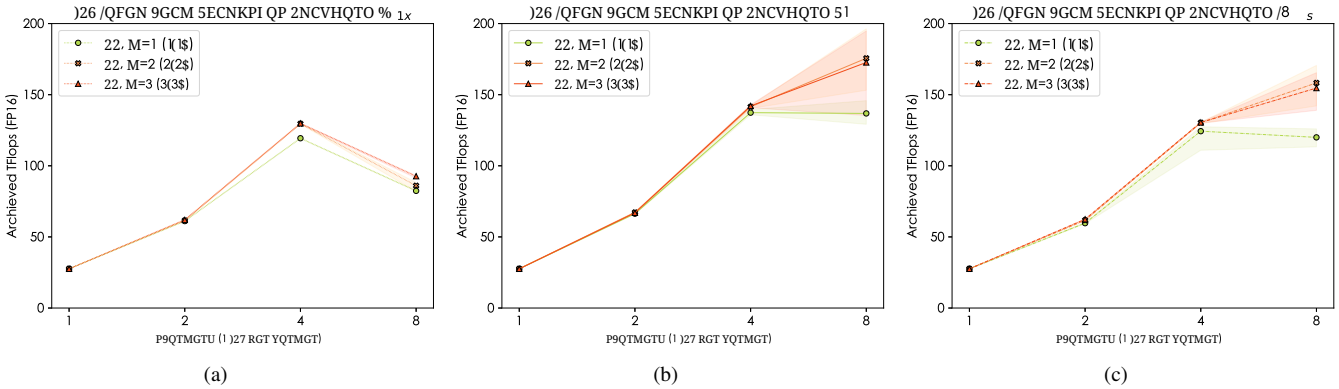


图 8：GPT 模型流水线并行弱扩展（按参数）性能。阴影区域表示不同步骤的性能变化范围。

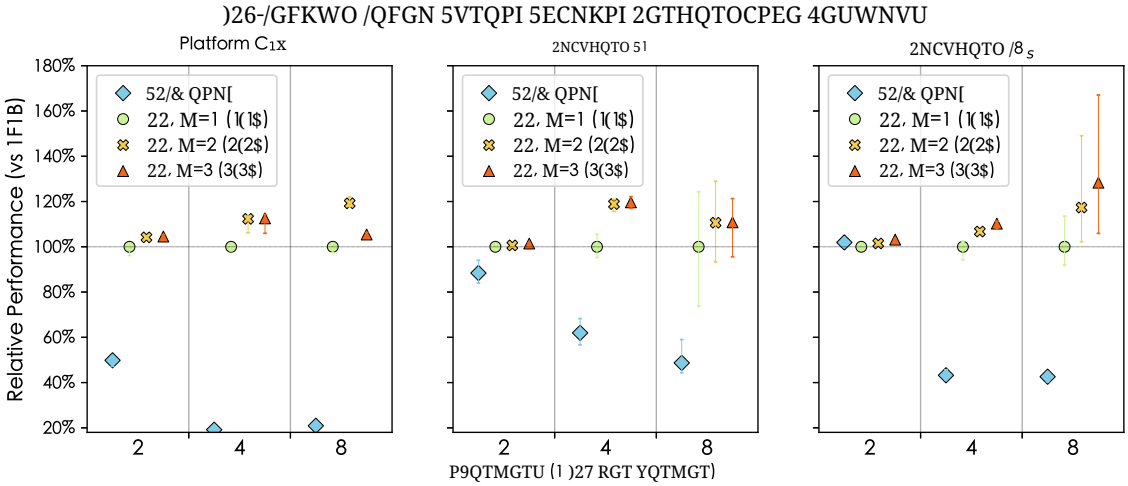


图 9：GPT-Medium 在三个目标平台上采用不同流水线调度方法的强扩展性能结果。还测试了仅采用单程序多数据并行的情况，以供比较。误差线表示不同步骤的性能变化范围。

k 从 1 变为 6，并将微批次大小调整为 6//k。测试安排在不同时间段分 5 轮进行，以便在不同程度的整体集群网络负载下获得更全面的测试结果。相较于第1轮的 1F1B，性能相对变化及其记录如图6所示。

结果显然表明，与 1F1B 计划相比，Ada-Grouper 的性能提升了 10%–25%。还可以推测，网络在第3轮和第5轮测试期间处于繁忙状态，因为 1F1B 的耗时降至第1轮的 90%。然而，$k > 1$ 类情况的性能结果仍保持在相当稳定的水平，比 1F1B 的结果高出 20% 以上，尽管性能也略有下降。当 $k \geq 3$ 时，随着 k 增大，性能并未进一步确定性地提升。这可能是因为：1）在如此高的流水线粒度下不再有更多的重叠；2）计算更小的微批次会导致计算效率降低。

综上所述，kFkB 调度可能更稳定地实现更优的计算和更好的通信重叠，从而以更细的粒度提升并行效率，同时不会增加内存需求。

6.2.2 弱扩展测试

在 U-Net 和 GPT 模型上进行了弱扩展测试，最多使用 8 个流水线工作进程。对于 U-Net，全局数据批次大小按工作进程数量进行缩放。我们在平台 M8$_s$上分别测试了两种配置，图7 展示了 Ada-Grouper 相较于 1F1B 的相对性能。由于内存溢出，UNet-Medium 没有 $k = 4$ 或 $N_{workers}$=8 的结果。与 BERT 或 GPT 等基于层的语言模型相比，U-Net 结构在划分后的流水线阶段之间可能需要更多张量通信。尽管如此，最多达到

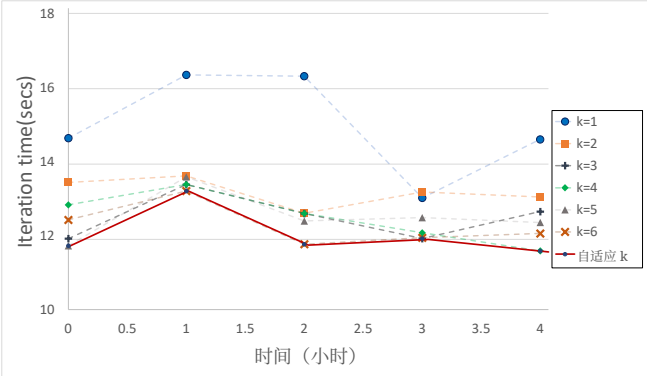


图 10：GPT 自适应调优测试。每小时触发一次调优，并从具有不同k值的计划中选择最优计划，从而适应不断变化的网络。

当$k \geq 2$时，UNet-Base 仍可观察到 2–14% 的性能提升，UNet-Medium 则可观察到 4–5% 的性能提升。

对于 GPT，按照表 1中的说明，随着工作器数量增加，进行了模型参数数量扩展。所有测试的全局批次大小均设为 64。我们还根据 [23] 中的方法，计算了测试期间达到的实际浮点运算次数。为更好地了解不同平台环境条件的影响，我们在全部三个平台上进行了测试。结果如图 8 所示。

在 GPT 弱扩展测试中，Ada-Grouper 的kFkB 调度计划始终优于 1F1B，尤其是在 8 个工作器的情况下。聚焦于三个不同平台在这一规模下的表现，可以发现平台 C_{1x}上的性能未能扩展，主要原因是其带宽有限，无法满足 8 个阶段带来的更高通信需求。据推测，在平台 $M8_s$ 上，训练期间共享作业可能会对实际通信延迟和带宽产生轻微影响。因此，平台 S1 上测得的浮点运算次数略高于平台 $M8_s$，尽管两者连接在同一生产网络环境中。这也可能是工作器数量增加时，弱扩展结果中出现更大性能波动的原因之一；其他所有测试也呈现出相同趋势。

6.2.3 强扩展测试

在 GPT-Medium 模型上进行强扩展测试，不仅可以解释kFkB 性能，我们还尝试在相同工作负载和平台条件下，将其与同等规模的纯单程序多数据并行训练进行比较。这些测试也在全部三个平台上进行，结果如图 9 所示。所有测试的全局批次大小均设为 64。流水线并行测试的微批次大小为 1，纯单程序多数据测试的微批次大小为 8。

与前几节介绍的测试类似，kFkB 的性能比 1F1B 提升了最高 20%。此外，结果还表明，对于这一 GPT-Medium 模型配置，流水线在这些平台上的生产工作负载中几乎总是能获得更高的性能。

对于所有单程序多数据并行结果，我们都检查了 Rhino 自动并行系统推导出的并行策略。这些单程序多数据策略非常接近数据并行，在每个微批次计算期间需要传输约 0.7–1.4 GB 的数据，这与我们的认知相符。但流水线并行中的这一数值降低了 2–5 倍。因此，不难理解在这种情况下，流水线并行为何应是一种更好的并行方法。

6.2.4 自适应调优测试

图 .10展示了 Ada-Grouper 在网络资源受到抢占的 Plat- form S1 上的调优结果。Ada-Grouper 生成了六种不同的调度方案，其 k 值从 1 到 6。在线调优触发条件设为每小时触发一次。我们在实验中采样了四个小时。全局批次大小保持为 192。通过对跨阶段通信和阶段执行时间进行性能分析，估算每个方案的性能。虚线上的点表示各方案在评估时间的估算性能，而实线上的点表示同一评估时间下所有方案中的最佳性能。$k = 1$ 对应原始的 1F1B 计划；由于不稳定网络中气泡数量增加，该计划在每个评估时间的表现都逊于几乎所有其他方案。其他方案的 k 值从 2 到 6 不等，性能也各不相同，但随着 k 值增大，性能提升不一定会更加显著。这是因为随着 k 值增大，需要采用更小的微批次大小，导致计算效率未能充分利用，并受到内存限制。因此，计算效率与受抢占网络中的通信重叠能力之间的权衡，使得估算整体性能成为必要。

在我们的实验中，Ada-Grouper 最初选择了 $k = 5$，随后在接下来的两次评估时间选择了 $k = 6$。尽管采用 $k = 5$ 和 $k = 6$ 的计划，其估算性能相近，但两者的表现都比 1F1B 计划高出 20% 以上。第三小时的网络抢占似乎有所缓解，因为与前两小时相比，1F1B 的性能有所提升，且所有其他计划的性能都非常接近。此时，Ada-Grouper 切换到了采用 $k = 3$ 的计划。最后一小时，网络再次变得不稳定，导致不同计划之间的性能差距扩大。采用 $k = 4$ 的计划在所有计划中表现最佳，比 1F1B 高出约 21%，这表明 Ada-Grouper 能够在存在抢占的环境中提供稳定且令人满意的性能。用户可以通过环境变量调整触发频率，从而适应

频繁变化的网络环境。

7 相关工作

7.1 流水线并行

GPipe [9]提出了采用流水线并行训练大型深度模型的方法，并指出了峰值内存使用量和气泡问题。它使用梯度检查点 [7] [4] 来缓解内存压力。PipeDream [22] 首次提出 1F1B 调度方案，并将这一技术用于异步训练，以减少气泡时间。Pipemare [36] 为流水线并行提供了一种稳健的异步训练方法，在不牺牲内存使用量和利用率的情况下保持模型质量。DAPPLE [6] 将 1F1B 调度方案扩展至同步训练，并指出峰值内存使用量可以保持恒定。Pipe-Torch [37] 通过结合数据并行和模型并行，将流水线并行改进为混合并行方法，以加速异构网络环境中的训练。Chimera [17] 采用双向流水线并行，进一步提升流水线效率。OOO [24] 提出了用于反向传播的乱序调度，从而促进分布式训练。Ter- aPipe [18] 提出了一种新颖的流水线方法，通过对语言模型的令牌级维度进行划分来实现。

7.2 深度学习编译器

HLO [34] 是 XLA 中用于张量计算的基于单赋值的中间表示（IR）。MLIR [14]是可复用且可扩展的编译器基础设施，它标准化了基于静态单赋值的 IR 数据结构，并提供了用于定义中间表示方言的声明式系统。GShard [15]和 GSPMD [35] 在 HLO 中间表示中引入了集合通信原语，并提供便捷的 API 来对大型模型进行分片。Relay [27] 提出了一种编译器框架，用于统一并泛化现有框架中的 IR。TVM [3] 是一种编译器，可提供图级和算子级优化，从而使深度学习工作负载能够在各种硬件后端之间实现性能可移植性。

7.3 通信隐藏策略

PyTorchDDP [16] 在 PyTorch 上实现数据并行，并在反向传播过程中重叠执行梯度归约。Horovod [29] 通过采用张量融合技术，使梯度全归约能够在多个设备之间高效交换，同时优化通信隐藏效果和网络利用率。HiDup [39]将深度神经网络模型复制为两个互不依赖的副本，并交错执行，使一个副本的计算与另一个副本的通信重叠。IPart [33] 通过以不同的分块大小划分通信和计算，使梯度通信与反向计算以及参数通信重叠。

7.4 并行框架

Alpa [40]构建了一个用于并行化的分层模型，将算子内并行建模为整数线性规划问题，并采用动态规划方法探索算子内并行和算子间并行。Unity [32] 扩展了 TASO [12]，引入并行计算图，并通过随机图替换同时优化代数变换和并行化。FlexFlow [20] 定义了“S OAP”搜索空间，并利用随机搜索研究并行化。Whale [11]根据计算均衡原则对模型进行划分，以适应异构计算设备，并允许用户借助并行化原语指定并行化策略。Galvatron [21] 提出了面向类 Transfor mer 模型的自动并行优化方法。Rhino [38]是我们之前的工作，通过结合整数线性规划、动态规划和数据驱动启发式方法，自动探索单程序多数据和流水线并行，为用户在搜索时间和策略质量之间提供灵活的权衡。BytePS [13] 利用 GPU 集群中的闲置 CPU 和带宽资源，通过参数服务器加速分布式训练。Megatron [23] 支持大规模训练 Transformer 模型，其专家设计的并行化策略结合了数据并行、张量分片并行和流水线并行。DeepSpeed [26] 是一个高性能框架，采用专家设计的、涉及数据并行和模型并行的策略。

8 结论

我们提出了 Ada-Grouper，这是一种自适应的 kFkB 调度器，可定期调整调度组成员数 k，以适应流水线并行中的抢占式网络环境。首先，我们分析了拥塞网络中 1F1B 调度方案的性能影响，这种情况在云服务中十分常见。相比之下，由于气泡更少，kFkB 方案会导致阶段计算停顿更少，并缩短流水线长度。随后，我们分析了其在不稳定网络环境中保持更稳定且更优性能的能力。此外，我们引入 Ada-Grouper 处理遍，在给定固定全局批次大小的情况下，生成具有不同组成员数（k）和微批次大小（b）的所有潜在候选方案。为剪枝候选集，我们设计了一种高效的枚举算法，用于过滤掉所有超过内存限制或未充分利用设备内存的方案，从而仅保留位于帕累托最优前沿上的潜在候选方案。最后，为适应不断变化的网络环境，Ada-Grouper 通过评估每个方案的性能，提供了一种自动调优机制，可在候选方案之间切换。我们的

实验表明，与 1F1B 相比，Ada-Grouper 在 GPT 和 UNet 的不同平台上实现了更优的性能，提升幅度介于 4% 至 30% 之间，同时能够在不断变化的网络环境中保持稳定的性能。

参考文献

[1] Martín Abadi、Paul Barham、Jianmin Chen、Zhifeng Chen、Andy Davis、Jeffrey Dean、Matthieu Devin、San-jay Ghemawat、Geoffrey Irving、Michael Isard 等。TensorFlow：一个用于大规模机器学习的系统。载于第 *12* 届 {*USENIX*} 操作系统设计与实现研讨会（{*OSDI*} *16*），2016 年。

[2] Tom B Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared Kaplan、Prafulla Dhariwal、Arvind Nee- lakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。语言模型是少样本学习者。*arXiv* 预印本 *arXiv:2005.14165*，2020。[3] Tianqi Chen、Thierry Moreau、Ziheng Jiang、Lianmin Zheng、Eddie Yan、Meghan Cowan、Haichen Shen、Leyuan Wang、Yuwei Hu、Luis Ceze 等。Tvm：一种用于深度学习的自动化端到端优化编译器。*arXiv* 预印本 *arXiv:1802.04799*，2018。[4] Tianqi Chen、Bing Xu、Chiyuan Zhang 和 Carlos Guestrin。以亚线性内存开销训练深度神经网络。*arXiv* 预印本 *arXiv:1604.06174*，2016。[5] Aakanksha Chowdhery、Sharan Narang、Jacob Devlin、Maarten Bosma、Gaurav Mishra、Adam Roberts、Paul Barham、Hyung Won Chung、Charles Sutton、Sebastian Gehrmann 等。PaLM：借助 Pathways 扩展语言建模。*arXiv* 预印本 *arXiv:2204.02311*，2022。[6] S. Fan、Y. Rong、C. Meng、Z. Cao 和 W. Lin。DAPPLE：一种用于训练大型模型的流水线式数据并行方法。发表于*PPoPP '21*：第 *26* 届 *ACM SIGPLAN* 并行编程原理与实践研讨会，2021。[7] Jianwei Feng 和 Dong Huang。任意计算图的最优梯度检查点搜索。发表于*IEEE/CVF*计算机视觉与模式识别会议论文集，第 11433–11442 页，2021。[8] Roy Frostig、Matthew James Johnson 和 Chris Leary。通过高级跟踪编译机器学习程序。机器学习系统，4(9)，2018。[9] Yanping Huang、Youlong Cheng、Ankur Bapna、Orhan Firat、Dehao Chen、Mia Chen、HyoukJoong Lee、Jiquan Ngiam、Quoc V Le、Yonghui Wu 等。GPipe：使用流水线高效训练巨型神经网络并行。神经信息处理系统进展，32：103–112，2019。

[10] Sylvain Jeaugey。NCCL 2.0。载于图形处理器技术大会（*GTC*），第 2 卷，2017 年。

[11] Xianyan Jia、Le Jiang、Ang Wang、Wencong Xiao、Ziji Shi、Jie Zhang、Xinyuan Li、Langshi Chen、Yong Li、Zhen Zheng 等。Whale：在异构 {图形处理器} 上进行高效的大模型训练。载于*2022 USENIX* 年度技术会议（*USENIX ATC 22*），第 673–688 页，2022 年。

[12] Zhihao Jia、Oded Padon、James Thomas、Todd Warsza- wski、Matei Zaharia 和 Alex Aiken。TASO：通过自动生成图替换来优化深度学习计算。载于第 *27* 届 *ACM* 操作系统原理研讨会论文集，第 47–62 页，2019 年。[13] Yimin Jiang、Yibo Zhu、Chang Lan、Bairen Yi、Yong Cui 和 Chuanxiong Guo。用于加速异构 GPU/CPU 集群中的分布式 DNN 训练的统一架构。载于第 *14* 届 *USENIX* 操作系统设计与实现会议论文集，第 463–479 页，2020 年。[14] Chris Lattner、Mehdi Amini、Uday Bondhugula、Albert Cohen、Andy Davis、Jacques Pienaar、River Riddle、Ta- tiana Shpeisman、Nicolas Vasilache 和 Oleksandr Zi- nenko。MLIR：面向摩尔定律终结的编译器基础设施。*arXiv* 预印本 *arXiv:2002.11054*，2020 年。[15] Dmitry Lepikhin、HyoukJoong Lee、Yuanzhong Xu、De- hao Chen、Orhan Firat、Yanping Huang、Maxim Krikun、Noam Shazeer 和 Zhifeng Chen。GShard：通过条件计算和自动分片扩展巨型模型。*arXiv* 预印本 *arXiv:2006.16668*，2020 年。[16] Shen Li、Yanli Zhao、Rohan Varma、Omkar Salpekar、Pieter Noordhuis、Teng Li、Adam Paszke、Jeff Smith、Brian Vaughan、Pritam Damania 等。PyTorch 分布式：加速数据并行训练的经验。*arXiv* 预印本 *arXiv:2006.15704*，2020 年。[17] Shigang Li 和 Torsten Hoefler。Chimera：利用双向流水线高效训练大规模神经网络。载于国际高性能计算、网络、存储与分析会议论文集，第 1–14 页，2021 年。[18] Zhuohan Li、Siyuan Zhuang、Shiyuan Guo、Danyang Zhuo、Hao Zhang、Dawn Song 和 Ion Stoica。TeraPipe：用于训练大规模语言模型的令牌级流水线并行。载于国际机器学习会议，第 6543–6552 页。PMLR，2021 年。

[19] Junyang Lin、Rui Men、An Yang、Chang Zhou、Ming Ding、Yichang Zhang、Peng Wang、Ang Wang、Le Jiang、Xianyan Jia 等。M6：一种中文多模态预训练模型。*arXiv* 预印本 *arXiv:2103.00823*，2021 年。

[20] Wenyan Lu、Guihai Yan、Jiajun Li、Shijun Gong、Yinhe Han 和 Xiaowei Li。Flexflow：一种用于卷积神经网络的灵活数据流加速器架构。发表于 *2017* 年 *IEEE* 高性能计算机体系结构国际研讨会（*HPCA*），第 553–564 页。IEEE，2017 年。

[21] Xupeng Miao、Yujie Wang、Youhe Jiang、Chunan Shi、Xiaonan Nie、Hailin Zhang 和 Bin Cui。Galvatron：通过自动并行实现多图形处理器上的高效 Transformer 训练。*arXiv* 预印本 *arXiv:2211.13878*，2022 年。

[22] Deepak Narayanan、Aaron Harlap、Amar Phanishayee、Vivek Seshadri、Nikhil R Devanur、Gregory R Ganger、Phillip B Gibbons 和 Matei Zaharia。PipeDream：用于 DNN训练 的通用流水线并行。在第 *27* 届 *ACM* 操作系统原理研讨会论文集，第 1–15 页，2019 年。[23] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Korthikanti、Dmitri Vainbrand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro、Amar Phanishayee 和 Matei Zaharia。使用 Megatron-LM 在 GPU 集群上高效训练大规模语言模型。在国际高性能计算、网络、存储与分析会议论文集，SC '21，2021 年。[24] Hyungjun Oh、Junyeol Lee、Hyeongju Kim 和 Jiwon Seo。乱序反向传播：一种有效的深度学习调度技术。在第十七届欧洲计算机系统会议论文集，第 435–452 页，2022 年。[25] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。PyTorch：一种命令式风格的高性能深度学习库。神经信息处理系统进展，32：8026–8037，2019 年。[26] Jeff Rasley、Samyam Rajbhandari、Olatunji Ruwase 和 Yuxiong He。DeepSpeed：系统优化使训练参数量超过 1000 亿的深度学习模型成为可能。在第 *26* 届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集，第 3505–3506 页，2020 年。[27] Jared Roesch、Steven Lyubomirsky、Marisa Kirisame、Logan Weber、Josh Pollock、Luis Vega、Ziheng Jiang、Tianqi Chen、Thierry Moreau 和 Zachary Tatlock。Relay：一种用于深度学习的高级编译器。*arXiv* 预印本 *arXiv:1904.08368*，2019 年。

[28] Olaf Ronneberger、Philipp Fischer 和 Thomas Brox。U-Net：用于生物医学图像分割的卷积网络。发表于医学图像计算与计算机辅助干预国际会议，第 234–241 页。Springer，2015 年。

[29] Alexander Sergeev 和 Mike Del Balso。Horovod：在 TensorFlow 中实现快速、简便的分布式深度学习。*arXiv* 预印本 *arXiv:1802.05799*，2018 年。

[30] Alexander Sergeev 和 Mike Del Balso。Horovod：在 tensorflow 中实现快速、简便的分布式深度学习。*arXiv* 预印本 *arXiv:1802.05799*，2018 年。

[31] Mohammad Shoeybi、Mostofa Patwary、Raul Puri、Patrick LeGresley、Jared Casper 和 Bryan Catanzaro。《Megatron-lm：使用模型并行训练数十亿参数的语言模型》。*arXiv* 预印本 *arXiv:1909.08053*，2019 年。[32] Colin Unger、Zhihao Jia、Wei Wu、Sina Lin、Mandeep Baines、Carlos Efrain Quintero Narvaez、Vinay Ramakrishnaiah、Nirmal Prajapati、Pat McCormick、Jamaludin Mohd-Yusof 等。《Unity：通过代数变换与并行化的联合优化加速 {DNN} 训练》。载于第 *16* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 22*），第 267–284 页，2022 年。[33] Shaoqi Wang、Aidi Pi、Xiaobo Zhou、Jun Wang 和 Cheng-Zhong Xu。《在参数服务器中重叠通信与计算，以实现可扩展的深度学习训练》。*IEEE*并行与分布式系统汇刊，32(9):2144–2159，2021 年。[34] XLA 和 TensorFlow 团队。《Xla - tensorflow，已编译》。2017 年。[35] Yuanzhong Xu、HyoukJoong Lee、Dehao Chen、Blake Hechtman、Yanping Huang、Rahul Joshi、Maxim Krikun、Dmitry Lepikhin、Andy Ly、Marcello Maggioni 等。《Gspmd：机器学习计算图的通用可扩展并行化》。*arXiv* 预印本 *arXiv:2105.04663*，2021 年。[36] Bowen Yang、Jian Zhang、Jonathan Li、Christopher Ré、Christopher Aberger 和 Christopher De Sa。《Pipemare：异步流水线并行DNN训练》。机器学习与系统会议论文集，3:269–296，2021 年。[37] Jun Zhan 和 Jinghui Zhang。《Pipe-torch：在异构网络的图形处理器集群中实现基于流水线的分布式深度学习》。载于 2019 年第七届国际

高级云与大数据会议（*CBD*），第 55–60 页。IEEE, 2019 年。

[38] Shiwei Zhang、Lansong Diao、Siyu Wang、Zongyan Cao、Yiliang Gu、Chang Si、Ziji Shi、Zhen Zheng、Chuan Wu 和 Wei Lin。借助 Rhino 实现大型模型自动并行化：生产级人工智能平台上的系统化方法。*arXiv* 预印本 *arXiv:2302.08141*，2023 年。

[39] Shiwei Zhang、Lansong Diao、Chuan Wu、Siyu Wang 和 Wei Lin。利用 SPMD 并行加速大规模分布式神经网络训练。载于第 *13* 届云计算研讨会论文集，第 403–418 页，2022 年。

[40] Lianmin Zheng、Zhuohan Li、Hao Zhang、Yonghao Zhuang、Zhifeng Chen、Yanping Huang、Yida Wang、Yuanzhong Xu、Danyang Zhuo、Eric P Xing 等。Alpa：自动化分布式深度学习的算子间和 {算子内} 并行。载于第 *16* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 22*），第 559–578 页，2022 年。