华中科技大学

研 究 生（数 据 中 心 技 术）课 程 论 文（报 告）

题目：云服务器机器学习任务优化综述

学　　号　　M202173757

姓　　名　　王志伟

专　　业　　电子信息

课程指导教师　　施展、童薇

院（系、所）　计算机科学与技术学院

2021 年 12 月 30 日

云服务器机器学习任务优化综述

王志伟 M202173757

华中科技大学 计算机科学与技术学院，武汉市

摘 要 随着人工智能的蓬勃发展，海量的深度学习任务部署在云服务器上，近几年越来越多的学者正在关注如何增大训练任务系统吞吐量及减小推理任务延迟的研究，其成为当前的一个研究热点。就推理任务系统而言，现有的处理方式是通过设置固定参数来形成一个 Batch，但是在面对动态变化的请求流量时，这种方案仍存在一定缺陷，LazyBatching 提出了一种细粒度的解决方案，能根据流量大小动态调节 Batch 大小，从而在不牺牲 SLA 要求的情况下达到较高的系统吞吐量。本文研究了三篇相关技术的论文，对多种加快机器学习任务、增大机器学习任务吞吐量的技术进行了综述，其一为针对 RNN 做的优化：细胞批处理以 RNN 单元的粒度做出批处理决策，并在请求加入和离开系统时动态组装批处理单元以供执行。其二为 PipeDream：这是一个将批间流水线添加到批内并行性以进一步提高并行训练吞吐量的系统，有助于更好地重叠计算与通信并在可能的情况下减少通信量。其三为 GrandSlam：估计通过应用程序中的各个微服务阶段传播的请求的完成时间。然后它利用这个估计来驱动一个运行时系统，该系统在每个微服务上动态批处理和重新排序请求，使各个作业满足各自的目标延迟，同时实现高吞吐量。

关键词 人工智能；深度学习；批处理；吞吐量；服务级别协定

Optimizing Cloud Machine Learning

Zhiwei Wang

College of Computer Science and Technology, Huazhong University of Science and Technology

Abstract With the vigorous development of artificial intelligence, a large number of machine learning tasks are deployed on cloud servers. In recent years, more and more scholars are paying attention to increase the system throughput of training tasks and reduce the delay of inference tasks. For inference systems, the existing solution is to form a batch by setting fixed parameters, but this solution still has certain defects when faced with dynamically changing request traffic. LazyBatching proposes a fine-grained solution which can dynamically adjust the batch size according to the traffic, so as to achieve a higher system throughput without sacrificing SLA requirements. This article has conducted three related technical papers, and reviewed a variety of technologies that speed up machine learning tasks and increase the throughput of machine learning tasks. One is optimization for RNN: cell batch processing is based on the granularity of RNN units. Make batch processing decisions, and dynamically assemble batch processing units for execution when requesting to join and leave the system. The second is PipeDream: This is a system that adds inter-batch pipelines to intra-batch parallelism to further improve parallel training throughput, which helps to better overlap computing and communication and reduce the amount of communication when possible. The third is GrandSlam: Estimate the completion time of requests propagated through the various microservice stages in the application. It then uses this estimate to drive a runtime system that dynamically batches and reorders requests on each microservice so that each job meets its own target delay while achieving high throughput.

Key words Artificial Intelligence; Deep Learning; Batch; Throughput; SLA

1 引言

近年来，深度学习方法从实验研究到现实世界的部署迅速成熟。深度神经网络（DNN）部署的典型生命周期包括两个阶段。在训练阶段，经过多次设计迭代后选择特定的 DNN 模型，并根据训练数据集计算其参数权重。在推理阶段，预训练模型用于使用计算出的权重处理实时应用程序请求。随着 DNN 模型的成熟，推理阶段会消耗最多的计算资源并为性能优化提供最大的收益。

与训练不同，DNN 推理除了良好的吞吐量外，还非常重视低延迟。由于应用程序通常需要实时响应，因此推理延迟对用户体验有很大影响。在现有的 DNN 架构中，面临最大性能挑战的是循环神经网络（RNN）。RNN 不同于其他流行的 DNN 架构，例如多层感知器 （MLP）和卷积神经网络（CNN），因为它代表递归而不是固定计算。因此，当在基于数据流的深度学习系统中表达 RNN 计算时，得到的“展开”数据流图不是固定的，而是根据每个输入而变化。RNN 计算的动态特性使它与最大的性能助推器—批处理不一致。当许多输入的底层计算相同时，它们的批量执行很简单，就像 MLP 和 CNN 的情况一样。相比之下，由于输入影响递归深度，批处理 RNN 计算具有挑战性。

现有系统专注于提高训练吞吐量。因此，他们以展开的数据流图的粒度对 RNN 计算进行批处理，我们将其称为图批处理。图批处理收集一批输入，将它们的数据流图组合成单个图，其算子代表原始图中相应算子的批量执行，并将组合图提交给后端执行。图批处理的最常见形式是将输入填充到相同的长度，以便生成的图变得相同并且可以轻松组合。这是在 TensorFlow、MXNet 和 PyTorch 中完成的。图批处理的另一种形式是动态分析一组依赖于输入的数据流图并融合等效运算符以生成聚合图。这种形式的批处理是在 TensorFlow Fold 和 DyNet 中完成的。

图批处理会损害模型推理的延迟和吞吐量。细胞批处理的技术可以显着提高 RNN 推理的延迟和吞吐量。我们的主要见解是认识到递归 RNN 计算是由连接在一起的不同数量的相似计算单元组成的，就像一个有机体由许多细胞组成一样。因此，我们建议在细胞粒度（也就是数据流图中的公共子图）而不是整个生物体（也就是整个数据流图）的粒度上执行批处理和执行，就像在现有系统中所做的那样。

微服务架构通过提供功能目录作为服务，可用作构建应用程序的构建块，从而大大减少了用户采用和维护服务器的工作量。这使数据中心运营商能够以与传统基础设施完全不同的方式来管理数据中心托管微服务。这种范式转变需要重新考虑在这种执行环境中采用的资源管理策略。我们观察到，可以利用微服务执行框架启用的可见性来实现高吞吐量和资源利用率，同时仍满足服务水平协议，尤其是在多租户执行场景中。在本研究中，我们提出了 GrandSLAm，这是一种微服务执行框架，可提高托管微服务的数据中心的利用率。GrandSLAm 估计通过应用程序中的各个微服务阶段传播的请求的完成时间。然后它利用这个估计来驱动一个运行时系统，该系统在每个微服务上动态批处理和重新排序请求，使各个作业满足各自的目标延迟，同时实现高吞吐量。与我们的基线相比，GrandSLAm 将吞吐量显着提高了 3 倍，而且不会违反各种现实世界 AI 和 ML 应用程序的 SLA。

DNN 训练极其耗时，需要高效的多加速器并行化。当前并行化训练的方法主要使用批内并行化，其中一次训练迭代被分配到可用的工作器上，但在工作器数量较多时收益递减。PipeDream 是一个将批间流水线添加到批内并行性以进一步提高并行训练吞吐量的系统，有助于更好地重叠计算与通信并在可能的情况下减少通信量。与传统的流水线不同，DNN 训练是双向的，其中前向通过计算图，然后是后向传递，使用状态和在前向传递期间计算的中间数据。因此，朴素的流水线可能会导致前向和后向传递中使用的状态版本不匹配，或者过多的流水线刷新和较低的硬件效率。为了应对这些挑战，PipeDream 修改了用于数值正确梯度计算的模型参数，并以最小的管道停顿在不同的工作线程上同时调度不同小批量的前向和后向传递。PipeDream 还自动在工作人员之间划分 DNN 层，以平衡工作并最大程度地减少通信。对一系列 DNN 任务、模型和硬件配置的广泛实验表明，PipeDream 训练模型的精度比常用的批内并行技术快 5.3 倍。

2 原理和优势

2.1 细胞批处理

图批处理对于推理效率不高，因为它以粗粒度（数据流图）执行批处理。RNN 的递归特性可以实现更细粒度的批处理—RNN 单元。由于所有未折叠的 RNN 单元共享相同的参数权重，因此在单元级别有充足的批处理机会：请求 X 的每个未折叠单元可以与来自请求 Y 的任何其他未折叠单元进行批处理。这样，RNN 单元类似于生物构成各种生物体的细胞。尽管生物体有多种类型和形状，但它们所拥有的细胞类型的数量要有限得多。此外，无论单元位于何处，相同类型的单元都执行相同的功能（并且可以批量处理）。这一特性使得在细胞级别而不是生物体（数据流图）级别进行批处理更有效。

在图批处理中，系统收集一个 Batch 大小的请求，执行完成该 batch 之后开始处理下一个 batch。相比之下，在细胞批处理中，并没有一个确定的参数来固定 batch 大小，新到

达的请求不断加入到正在执行的任务中，无需等待正在执行的任务完成。下图说明了在处理相同的 8 个请求时，图批处理和细胞批处理的不同处理行为。我们假设一个链式结构的 RNN 模型，并且链中的每个 RNN 单元都需要一个单位时间来执行。每个请求对应一个输入序列，其长度显示在括号中。在图中，每一行显示一个请求的生命周期，从它的到达时间开始。该示例使用的批大小为 4。可以看到细胞批处理不仅降低了每个请求的延迟而且增大了系统的吞吐量。

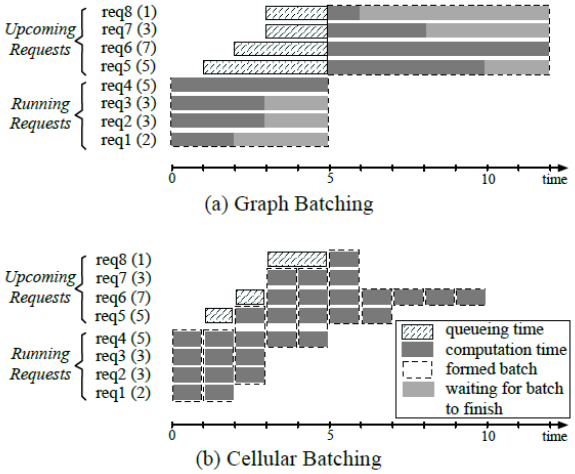


图 1 细胞批处理原理图

2.2 GrandSLAm

GrandSLAm 执行流程的第一步是识别每个作业中存在的微服务管道。为此，我们的系统将用户使用 Python、Scala 等高级语言编写的作业作为输入（下图中的 1）并将其转换为微服务的有向无环图（DAG）（下图中的 2）。这里，每个顶点代表一个微服务，每条边代表两个微服务之间的通信（例如，RPC 调用）。这种基于 DAG 的执行模型已被广泛应用于分布式系统框架，如 Apache Spark、Apache Storm、TensorFlow 等。构建微服务 DAG 是一个离线步骤，需要在 GrandSLAm 的运行时系统开始跨微服务实例分发请求之前执行一次。

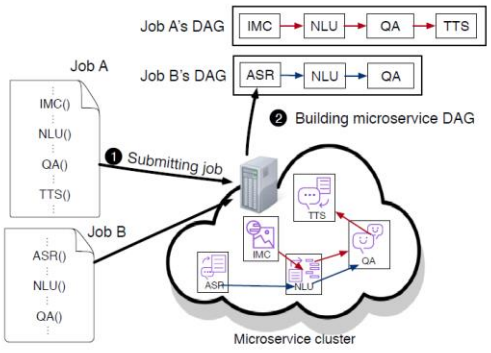


图 2 从微服务集群中的给定作业中提取使用过的微服务

请求的端到端延迟是每个微服务阶段请求完成时间的顶点。因此，为了设计一种为请求提供端到端延迟保证的运行时机制，我们采用了一种分解的方法。我们计算每个请求需要满足的每个微服务阶段的部分截止日期，以便不违反端到端延迟目标。我们将其定义为微服务阶段松弛。换句话说，微服务阶段松弛被定义为请求可以在特定微服务阶段花费的最长时间。在构建微服务 DAG 之后，在 GrandSLAm 运行时系统启动之前，离线分配阶段松弛。

每个阶段的数学松弛是通过计算请求在每个特定微服务阶段可以利用的端到端延迟的比例来确定的。

$$slack_m = \frac{L_m}{L_a + L_b \cdots + L_m + \ldots} \times SLA$$

其中 Lm 是阶段 m 和 La, Lb…的作业延迟。是同一作业在其他阶段 a, b…的延迟。图 6 说明了应该在每个微服务阶段为不同的批量大小分配的时间比例，对于一个名为 Pose Estimation for Sign Language 的真实世界应用程序。我们可以从图 6 中清楚地看到，请求完成序列学习阶段所需的时间百分比远高于相同请求完成活动姿势阶段所需的时间百分比。使用此观察，请求按比例分配阶段级执行松弛。

2.3 PipeDream

PipeDream 使用管道并行(PP)，这是一种新的并行化策略，它结合了批内并行和批间并行。流水线并行计算涉及将 DNN 模型的层划分为多个阶段，其中每个阶段由模型中的一组连续层组成。每个阶段都映射到一个单独的 GPU，该 GPU 对该阶段中的所有层执行前向传递（和反向传递）。

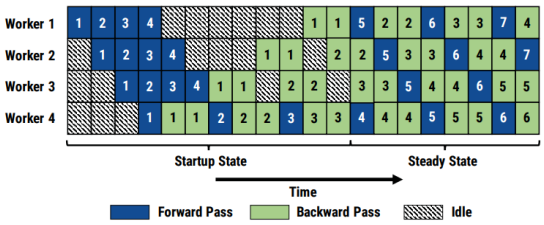


图 3 PipeDream 流水线的一个例子

在最简单的情况下，系统中只有一个 minibatch 处于活动状态，就像在传统的模型并行训练中一样；在此设置中，一次最多有一个 GPU 处于活动状态。理想情况下，我们希望所有 GPU 都处于活动状态。考虑到这一点，我们一个接一个地将多个小批量注入管道。在完成小批量的前向传递后，每个阶段将输出激活异步发送到下一个阶段，同时开始处理另一个小批量。最后一个阶段在正向传递完成后立即开始对小批量的反向传递。在完成其向后传递后，每个阶段将梯度异步发送到前一个阶段，同时开始计算下一个小批量（图 3）。

3 研究进展

当所有输入具有相同的计算图时，批处理很简单。某些 DNN 就是这种情况，例如多层感知器(MLP)和卷积神经网络 (CNN)。然而，对于 RNN，每个输入都有可能不同的递归深度，并导致不同大小的展开图。这种依赖于输入的结构使 RNN 的批处理具有挑战性。现有系统在如何为 RNN 批处理方面分为两个阵营：(1) TensorFlow /MXNet /PyTorch /Theano：这些系统将一批输入序列填充到相同的长度。因此，每个输入都具有相同的计算图，并且可以轻松地批量执行。图 4a 显示了通过填充进行批处理的示例。但是，填充不是通用的解决方案，只能应用于使用链状结构处理顺序数据的 RNN。对于 TreeLSTM 等非链式 RNN，padding 不起作用。(2)TensorFlow-Fold/DyNet：在这两个最近的工作中，系统首先收集一批输入样本，并为每个输入生成数据流图。然后系统将所有这些数据流图合并到一个图中，其中某个操作符可能对应于原始图中操作的批处理执行。图 4b 显示了一个示例。

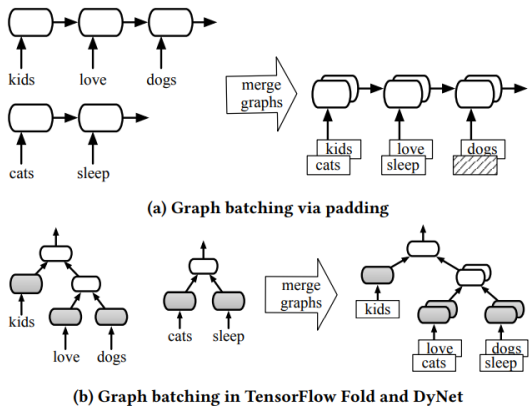


图 4 现有的图批处理系统

上述两种现有策略都试图收集一组输入以形成一个批处理，并找到与批处理中的所有输入兼容的数据流图。因此，我们将这两种策略都称为图形批处理。现有系统使用图批处理进行训练和推理。我们注意到图批处理是 RNN 训练的理想选择。首先，由于所有训练输入在训练开始之前就已存在，因此在收集批次时没有延迟。其次，短输入是否与长输入合并并不重要，因为小批量（同步）SGD 必须等待整个批次完成才能计算参数梯度。

不幸的是，图批处理远非 RNN 推理的理想选择，并且会对延迟和吞吐量产生负面影响。由于不必要的同步，图形批处理会导致额外的延迟，因为除非当前批处理中的所有请求都已完成，否则输入无法开始执行。当输入具有不同的长度时，这在实践中会进一步加剧，导致一些长输入延迟整个批次的完成。图批处理也可能导致次优吞吐量，这是由于对填充执行无用的计算或未能在合并数据流图中的所有运算符的最佳级别进行批处理。

RNN 的独特之处在于它包含许多相互连接的相同计算单元。细胞批处理利用此功能来 1)在 RNN 单元级别而不是整个数据流图进行批处理，以及 2)让新请求加入当前请求的执行，并让请求在完成后立即返回给用户。

在细胞批处理中，调度器需要决定哪些节点应该一起批处理以形成一个任务，以及哪些任务要推送给哪些 worker。设计必须考虑三个因素，局部性、优先级和多个 GPU 的利用率，这三个因素经常相互冲突。

局部性是指以下偏好：1）如果要执行相同的节点序列，则相同的一组请求应该一起批处理，以及 2）该节点序列的执行应该坚持在同一个 GPU 上。1)和 2)的原因都是为了避免内存复制。在执行之前，单元格的批量输入必须布置在连续的 GPU 内存中。由于执行的批处理输出也存储在连续的内存中，因此在同一 GPU 上执行同一组请求时，无需在每个单独的单元执行之前进行内存复制。相反，如果这批请求在两个连续的单元执行之间发生变化，则必须进行内存复制，称为“收集”，以确保连续输入。此外，如果连续单元的执行从一个 GPU 切换到另一个 GPU，则必须将数据从一个 GPU 复制到另一个 GPU。

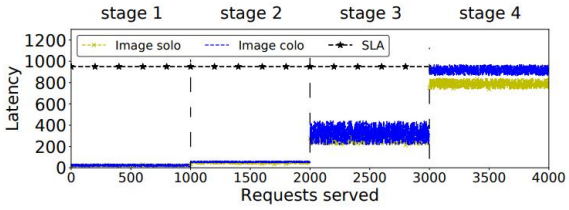
优先级是指优先执行一种类型的单元格而不是另一种类型的能力。许多实用的 RNN 模型具有多种类型的细胞。例如，如图 2 所示，TreeLSTM 有叶子单元和内部单元。流行的基于 RNN 的 Seq2Seq 模型具有编码器单元和解码器单元。对于这些模型，可以通过优先执行计算图中稍后出现的 DNN 类型来实现更好的延迟。因此，在 TreeLSTM 中，内部节点应该优先于叶节点。在 Seq2Seq 模型中，解码器节点应该优先于编码器节点。

我们设计了一个简单的调度策略来在多个 GPU 的局部性、优先级和良好利用率之间进行权衡。我们通过构建和调度包含多个节点调用而不是单个节点调用的批处理任务来支持位置偏好。为了实现这一点，请求处理器会分析请求的单元图，以找到要传递给调度程序的子图。子图包含单个节点或多个连接的节点，其特性是已满足对图其他部分的所有外部依赖关系。此外，子图的所有节点必须是相同的单元类型。例如，在 Seq2Seq 的情况下，一系列编码器单元形成一个子图，而解码器单元序列形成另一个子图。

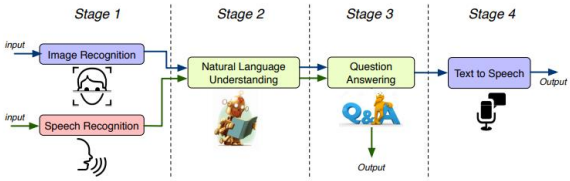
微服务架构在软件工程师中越来越受欢迎，因为它可以更轻松地开发和部署应用程序，而不必担心底层硬件和软件要求。微服务类似于执行特定功能的定义明确的库，可以通过简单的 API 向消费者（即应用程序开发人员）公开。使用微服务范式方法，软件工程师不是从头开始编写应用程序，而是利用这些微服务作为构建块来构建端到端应用程

序。端到端应用程序由一系列微服务组成，其中许多由数据中心服务提供商提供。基于微服务的软件架构加快了部署周期，通过提供一组丰富的原语来促进应用程序级创新，并提高可维护性和可扩展性，对于那些往往在许多应用程序上下文中使用相同构建块的应用程序类。

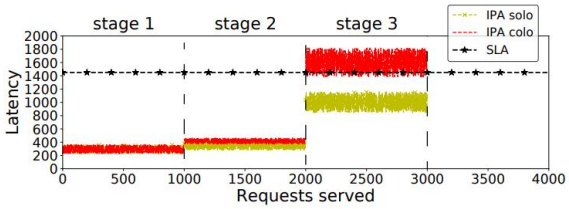
传统的多层架构根据服务的性质将应用程序阶段划分为不同的层。在大多数情况下，应用程序阶段属于专注于用户界面的表示层、实际应用程序执行发生的应用程序处理层以及存储属于应用程序的数据和元数据的数据管理层。这与微服务架构有着根本的不同。微服务，在多阶段应用程序的每个阶段，在大型应用程序中执行部分处理。换句话说，可以想象一串微服务来构成应用处理层。



(a) Image Querying: SLA is not violated



(b) Sharing NLU and QA microservices



(c) IPA: SLA is violated

图 5：在图像查询和智能个人助理之间共享两个微服务

尽管微服务的使用和部署与传统的数据中心应用有着根本的不同，但即使在包含微服务的数据中心中，资源利用率低的问题仍然存在。为了遏制这种情况，数据中心服务提供商可能允许跨多个作业共享通用微服务，如图 5b 所示。但是，这些面向用户的应用程序类别需要满足严格的服务级别协议 (SLA) 保证。因此，共享微服务可能会产生争用，从而违反面向用户的单个应用程序的端到端延迟，从而违反 SLA。这类似于传统数据中心，在那里倾向于主动避免多个面向用户的应用程序并置，导致在优化峰值性能时过度配置底层资源。

GrandSLAm 需要准确估计每个微服务阶段的执行时间。为此，确定影响微服务执行时间的因素变得至关重要。这促使我们基于一组基于我们考虑的应用程序空间的因素开发执行时间估计(ETC)模型。在本研究中，我们分析了 AI 和 ML 相关微服务的性能特征，因为这些应用程序非常适合托管在微服务架构上。在这种情况下，我们在 AI 和 ML 空间中观察到了两个截然不同的特征。首先，在这些微服务中广泛使用将多个请求批处理为一个大请求，以提高计算设备的资源利用率。为此，这些微服务对输入进行预处理（例如，在图像分类中调整图像大小、在语音识别中拆分语音输入、在自然语言处理中对单词进行分块）以适合单个批处理以同时执行。其次，许多 AI 应用程序展示了微服务的流水线执行。图像识别，来自 AWS Step Functions 的应用就是一个这样的例子。这种简单的线性管道使得设计基于松弛的优化变得更加容易。

但是，GrandSLAm 不能直接应用于 AI 和 ML 空间以外的微服务类型。例如，我们提出的一个简单模型对于其他类型的微服务来说是不够的，这些微服务不批处理属于不同应用程序的查询。例如，执行 SQL 范围查询的微服务的执行时间，对输入查询和输出结果都很敏感。换句话说，在不同数据集上执行的类似查询可能具有不同的执行时间。在这种情况下，需要对应用程序类型进行更详细的分析和调查，以构建更复杂的模型。除此之外，本研究未考虑复杂的微服务拓扑，例如通用图和条件执行。对于现有形式的 GrandSLAm 来说，在不同请求在运行时采用不同路径或需要并行执行几个微服务的情况下，计算空闲时间是一项挑战。这些是我们计划在不久的将来调查的 GrandSLAm 的一些局限性。

近几年训练 DNN 模型最常见的方法是批内并行化，其中将单个训练迭代拆分到可用的工作器中。在数据并行中，输入在工作人员之间进行分区。每个工作人员维护模型权重的本地副本，并在其自己的输入分区上进行训练，同时使用诸如 all_reduce 或参数服务器之类的集体通信原语定期与其他工作人员同步权重。传递的数据量与模型权重的数量和参与培训的工人数量成正比。

最常用的数据并行形式，称为批量同步并行或 BSP，要求每个工作人员等待来自其他工作人员的梯度。尽管进行了诸如无等待反向传播之类的优化，权重梯度在可用时立即发送（在现代框架中很常见），但对于大型模型而言，通信停顿有时是不可避免的，因为跨工作器同步梯度所需的时间可以支配计算时间。

我们专注于四个关键要点。首先，尽管使用多 GPU 服务器和 NCCL 等最先进的通信库，但其中许多模型的通信开销很高。数据并行性适用于 ResNet-50 等模型，这些模型具有大量具有紧凑权重表示的卷积层，但对于其他具有 LSTM 或全连接层的模型（具有更密集的权重表示），其扩

展性较差。其次，分布在多 GPU 服务器上的应用程序受到较慢的服务器间链接的瓶颈，正如当训练扩展到多台服务器时通信开销激增然后平稳所证明的那样。这种分层网络的数据并行性可能不太合适，因为在高带宽和低带宽通道上发送的字节数相同。第三，随着数据并行工作者数量的增加，所有模型的通信开销都会增加，即使是在使用 NVLink 的多 GPU 实例上执行训练也是如此。科尔曼等人显示了类似的结果。第四，随着 GPU 计算速度的提高（1080Tis 到 V100s），所有模型的通信开销也会增加。

异步并行训练(ASP)允许每个工作人员在接收来自前一个 minibatch 的梯度之前继续处理下一个输入 minibatch。这种方法通过重叠计算与通信提高了 BSP 上的硬件效率（每次迭代所需的时间），但也引入了陈旧性并降低了统计效率（达到特定目标精度所需的迭代次数）。

塞德等人研究了量化梯度以减少需要通过网络进行通信的数据量。这种近似策略对有限场景有效，但缺乏通用性；它不会损害某些语音模型的收敛性，但尚未证明对其他类型的模型有效。其他人已经探索了 HPC 文献中的技术来减少通信开销，通常使用高度专业化的网络硬件。我们的工作是对这些技术的补充，主要侧重于在使用公共云中可用的商品加速器和互连时提高并行 DNN 训练的性能。

最近的工作表明，使用大的 minibatches 对训练 ResNet-50 是有效的，尤其是与逐层自适应速率缩放(LARS)结合使用时。大的小批量通过不那么频繁地交换参数来减少通信开销；然而，我们的实验表明，这些技术在 ResNet-50 之外缺乏通用性，并且管道并行可以胜过最快的 LARS 数据并行选项。

模型并行性是一种批内并行性方法，其中 DNN 模型中的算子在可用的工作人员之间进行分区，每个工作人员仅对所有输入的模型参数的一个子集进行评估和更新。通信的数据量是需要跨工作人员发送的中间输出（和相应的梯度）的大小。

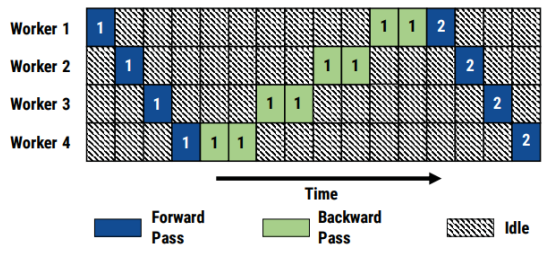


图 6 使用四个进程并行训练模型

尽管模型并行可以训练非常大的模型，但普通模型并行很少用于加速 DNN 训练，因为它有两个主要限制。首先，模型并行训练导致计算资源利用不足，如图 6 所示。每个工作人员负责一组连续的层；在这种情况下，这些组之间的中间输出是唯一需要在工人之间进行交流的数据。

模型并行训练的第二个限制是跨多个 GPU 划分模型的负担留给了程序员，从而产生了点解决方案。最近的工作探索了使用强化学习来自动确定模型并行性的设备放置。然而，这些技术是时间和资源密集型的，并且没有利用 DNN 训练可以被认为是由连续层组组成的计算管道这一事实。

最近的工作建议在多个维度之间拆分优化算法的单个迭代。OWT 手工分割当时流行的 AlexNet 模型，对权重参数少、输出大的卷积层使用数据并行，同时选择不复制权重参数多、输出小的全连接层。输出。OWT 不使用流水线。FlexFlow 建议沿样本、运算符、属性和参数拆分单个迭代，并描述了一种算法来确定如何以自动化方式执行此拆分。但是，FlexFlow 不执行流水线操作，实验表明这会省下多达 90%的性能。

Chen 等人简要探讨了流水线小批量在模型并行训练中的潜在好处，但没有解决适用于大型现实世界模型的良好统计效率、规模和通用性的条件。霍等人探索了在训练期间并行化反向传播。

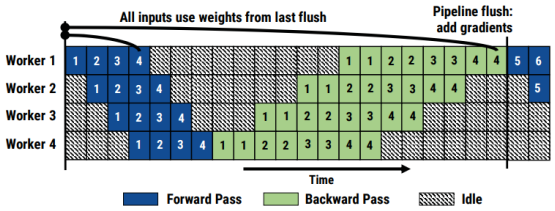


图 7 GPipe 的批间并行方法

GPipe（与早期 PipeDream 预印本的并行工作）在非常大模型的模型并行训练环境中使用流水线。GPipe 没有指定用于划分模型的算法，而是假设一个分区模型作为输入。GPipe 进一步将一个 minibatch 拆分为 m 个微批次，并对这 m 个微批次执行前向传递和后向传递（见图 7，m = 4）。GPipe 专注于训练像 AmoebaNet 这样的大型模型，针对内存效率进行了优化；它使用现有技术，例如权重梯度聚合，并通过丢弃前向和后向传递之间的激活存储来交换内存计算，而不是选择在后向传递中需要时重新计算它们。因此，如果 m 很小，则由于重新计算开销和频繁的管道刷新，它可能会降低硬件效率。

相比之下，PipeDream 解决了先前工作中忽略的关键问题，提供了一种通用解决方案，可以让工作人员得到充分利用，以有原则的方式将流水线与批内并行性相结合，同时还可以在可用工作人员之间自动划分模型。流水线并行 DNN 训练有助于减少可能导致批内并行性瓶颈的通信开销。PipeDream 自动跨工作器划分 DNN 训练，将批次间流水线与批次内并行性相结合，以更好地重叠计算与通信，同时最大限度地减少通信的数据量。

4 总结与展望

细胞批处理以实现对循环神经网络模型的低延迟推理。其以 RNN 单元为粒度批量执行推理请求。这样做允许新请求加入一批现有请求的执行，并在其计算完成后立即返回，而无需等待批次中的其他请求完成。细胞批处理只对 RNN 推理有益。它不会提高训练的性能，因为与推理不同，所有训练输入同时准备就绪，权重更新算法通常需要等待一批中的所有输入完成。

微服务执行框架正在迅速改变数据中心的运营方式。与单体应用程序相比，它为底层应用程序执行提供了更高的透明度。这种可见性是在同一系统上共置多个延迟关键应用程序并仍然满足 SLA 的关键推动因素。面对这种可见性和不断变化的机会，显然需要重新考虑运行时系统和框架。GrandSLAm 是一个运行时系统，它利用这种可见性并识别不同应用程序的单个查询中的松弛。GrandSLAm 使多个租户能够满足他们的 SLA，同时实现高吞吐量和利用率，无需性能开销或程序员支持。

此外，流水线并行 DNN 训练有助于减少可能导致批内并行性瓶颈的通信开销。PipeDream 自动跨工作器划分 DNN 训练，将批次间流水线与批次内并行性相结合，以更好地重叠计算与通信，同时最大限度地减少通信的数据量。与最先进的方法相比，PipeDream 在一系列 DNN 和硬件配置中以高达 5.3 倍的速度完成训练。

参 考 文 献

[1] Gao P, Yu L, Wu Y, et al. Low latency rnn inference with cellular batching[C]//Proceedings of the Thirteenth EuroSys Conference. 2018: 1-15.

[2] Kannan R S, Subramanian L, Raju A, et al. Grandslam: Guaranteeing slas for jobs in microservices execution frameworks[C]//Proceedings of the Fourteenth EuroSys Conference 2019. 2019: 1-16.

[3] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: generalized pipeline parallelism for DNN training[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019: 1-15.