基于云服务的深度学习推理优化方法综述

朱文清[1)]

[1)](华中科技大学计算机科学与技术学院 武汉 430000)

摘 要 当前云计算拥有十分广阔的应用场景，由于其性能的弹性使得许多工作的效益得到提高。深度学习模型的推理也是一个对服务器性能弹性要求较高的工作，因而目前很多深度学习任务在云服务器上进行。为了提高使用云服务器训练的性能和经济效益，保证训练时的服务质量(QoS)，本文总结了三个方面的优化策略。首先，由于云服务器的实例种类多样化，每种实例的性能与价格也存在较大差异，所以可以基于贝叶斯优化方法从实例池中挑选出能达到服务质量并且花费最小、具有良好性能的实例组合。其次，由于计算硬件如 GPU 的性能快速提升，使得存储层成为深度学习模型推理的瓶颈，根据深度学习作业的 IO 特点，一种分布式的缓存结构被提出，其能提高 GPU 集群中深度学习任务的 IO 效率，也能根据不同深度学习任务从缓存中的收益对这些任务进行优先级排序。最后，目前云服务从单体式服务逐渐转为微服务模式，每个微服务处于一个小的且基本是无状态的容器中，因而许多容器需要被调度到一台物理机上以最大化利用率，当前实际应用中每台物理机上只允许有一个高优先级的延迟敏感的服务和几个低优先级的批处理任务，而使用一个具有服务质量感知的资源控制器，在既利用硬件也利用软件隔离机制的情况下保护服务质量，可以让多个延迟敏感的应用在不违反服务质量的前提下共享一个物理主机。

关键词 云服务；贝叶斯优化；缓存管理；延迟敏感

A survey of deep learning inference optimization methods based on cloud services

Wenqing Zhu[1)]

[1)](School of Computer Science and Technology, Huazhong University of Science and Technology Wuhan 430000)

Abstract Currently cloud computing has a very broad application scenario, because of its elastic performance so that the efficiency of many jobs has been improved. The inference of the deep learning model is also a job that requires high elasticity of server performance, so many deep learning tasks are currently carried out on cloud servers. In order to improve the performance and economic benefits of training using cloud servers and ensure the quality of service (QoS) during training, this paper summarizes the optimization strategies in three aspects. First of all, due to the variety of instance types of ECSs and the differences in the performance and price of each instance, it is possible to select from the instance pool a combination of instances that achieve QoS target and balance the cost and performance based on Bayesian optimization methods. Secondly, due to the rapid improvement of the performance of computing hardware such as GPUs, the storage layer becomes the bottleneck of deep learning model inference, according to the IO characteristics of deep learning jobs, a distributed cache structure is proposed, which can improve the IO efficiency of deep learning tasks in GPU clusters, and deep learning tasks can also be prioritized based on benefits which these tasks get from cache. Finally, the current cloud service from a single service gradually shifted to a microservice model, each microservice is in a small and basically stateless container, so many containers need to be scheduled to a physical machine to maximize utilization, currently each physical machine only allows a high priority latency sensitive service and several low priority batch tasks, but the use of a quality of service awareness resource controller, can ensure the quality of service in the use of both hardware and software isolation

mechanism, also enables multiple latency-critical applications to share a single physical host without violating quality of service.

Key words Cloud service; Bayesian Optimization;cache management;latency-critical

1 引言

深度学习(DL)近年来在各种应用领域得到了广泛的应用。科学计算、计算机视觉和个性化推荐等领域广泛采用的模型显示了研究深度学习模型的重要性。虽然模型训练关注的是模型的准确性，并且没有严格的时间要求，但是在服务质量（QoS）约束下，模型推理通常在实时环境中进行。随着 DL 模型的广泛使用，推理服务提供商更有可能利用云计算资源为其客户提供服务，不同的云计算资源其成本也有差异。因此，随着 QoS 目标的实现，成本效益正成为主要的优化目标。云计算平台提供了多样的处理器体系结构，如何正确地利用各种可用资源来满足 QoS 目标并最大限度地降低成本成为关键的挑战。RIBBON 则通过在云计算实例池中查找最优实例搭配实现成本优化和 QoS 感知的功能。根据图 1 中不同实例在不同 Batch Size 下的性能表现和成本效益可以看出，具有高性能表现的实例成本效益低，低性能的反而成本效益较高，如果使用多样化实例池能提高平衡性能和开销的能力，例如便宜的实例能在保证 QoS 的同时减少开销。但快速查找最优化的实例池配置有三个难点。第一，随着实例种类的增加，搜索空间的维数也随之增加；第二，实例池配置和 QoS 达标率的关系难以用数学化的公式表达，只能耗费时间和成本测试每个实例池配置；第三，搜寻最佳配置时，没有明显有区别的实验结果，有时甚至反常规。例如花费相似的配置可能存在有显著差别的服务质量达标率，花费有显著差别的配置却有相似的 QoS 达标率。

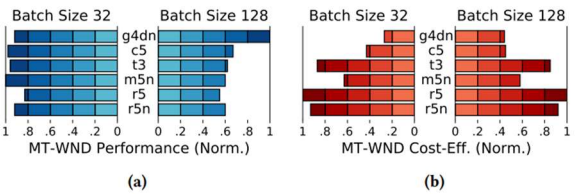

图 1 不同实例在不同 Batch Size 下的性能表现和成本效益

同时，提高深度学习工作的 IO 性能也是在云平台中进行深度学习推理的优化方向。深度学习的一些 IO 特性，使得分布式缓存的优点得以体现。首先，深度学习的 IO 存在可共享性，作业在作业内部和作业之间执行的 IO 有高度的重叠。在一个作业中，由于每个作业都要对相同的输入训练数据进行多次传递(即多个 epoch)，因此可以缓存数据以供以后的 epoch 使用。更重要的是，还有广泛的内部作业共享。其次，深度学习工作的数据是可替换的，深度作业的 epoch 只需要保持两个属性，其一是每个输入数据项必须恰好接触一次，其二是必须为每个小批选择随机的输入样本。数据项的确切顺序与作业的正确性或准确性无关，这意味着 IO 是可替换的，有了可替换性，即使是容纳 20%训练数据的小缓存也能提供良好的缓存性能，因为如果一个项不在缓存中，可以从缓存中返回一个保留随机性和唯一性属性的替代项。最后，深度学习作业的另一个有利特性是跨小批的可预见性，因为每个小批的时间是预先知道的，可以预测每个作业对 IO 性能有多敏感，这反过来又可以允许缓存的放置和回收为从缓存中获益最多的作业提供更高的优先级。Quiver 则基于这些 IO 特性设计了一种在 GPU 集群中用于深度学习训练(DLT)工作的知情存储缓存，显著提高深度学习工作负载的吞吐量。

随着云计算的发展，云应用逐渐从批处理转变成低延迟的服务。比如传统的以吞吐率为主的图像处理和大数据的应用在采用 Spark 和 X-Stream 框架后进入内存计算的阶段，这让任务执行延迟变为几毫秒或秒。云应用逐渐从单体式服务 Monolith 转变为成百上千个松耦合的微服务。尽管一个大规模的端到端的服务的延迟依旧保持在几毫秒或秒的粒度，每个微服务也必须满足更加严格的延迟约束，通常是几百微秒的级别。每个微服务处于一个小的且基本是无状态的容器中，这意味着许多容器需要被调度到一台物理机上以最大化利用率。当前的技术下每台物理机上只允许有一个高优先级的延迟敏感（LC）的服务与几个低优先级的批处理任务，并不能满足新的场景。在 PARTIES 设计之前，首先验证了“资源置换性（fungibility）”这个概念。之后提出 PARTIES，第一个针对多个延迟敏感（LC）应用的 QoS 有感资源管理器，它不需要应用的先验知识，通过动态监测，利用系统或硬件层面可划分共享资源。此外，PARTIES 适用于动态变化的负载，并利用用资源替换性来快速实现收敛。

2 原理和优势

2.1 平衡云上深度学习的成本效率和服务质量

为了解决云上进行深度学习模型推理的最佳实例池配置问题。RIBBON 通过构建搜索空间以探索确定最佳的异构的多样化池配置。在探索过程开始时，它以实例类型和 QoS 目标为输入。目标是确定每种实例类型应选择多少个实例，以便以最低成本满足 QoS 目标。最具挑战性的部分是快速确定最佳实例池配置，同时，参与评估的配置的数量也应该尽可能的小。RIBBON 使用贝叶斯优化 (BO) 为核心来解决这一问题。选择 BO 的原因是它是一种轻量级在线学习模型，不需要昂贵的培训、先验知识或数据。通过后续的评价可以得出，与其他竞争的搜索空间内最优查找方法相比，RIBBON 的 BO 引擎是高效的。

由于 RIBBON 优化的两个目标是相互竞争的，即高 QoS 目标和高成本效益，所以为了平衡两个目标，需要设计一个目标函数作为已给出的实例池配置优劣的评估标准，同时也能提供下一步优化的方向。RIBBON 目标方程的主要原则是，当配置满足 QoS 要求，目标方程应引导评估向花费更低的方向；当配置未满足 QoS 要求，目标方程应引导评估向 QoS 达标率更高的方向。

为了保证最优化过程的收敛性，RIBBON 使用了两种方法来规避这种情况，首先对不满足 QoS 要求的配置，仍然在目标函数中考虑其性能，而不是简单的在函数中返回 0 值；其次是在 BO 搜索时按配置中实例数量逐次增加的顺序，保证配置的平滑过渡。

2.2 提高云上深度学习任务吞吐量

云服务中为了保证深度学习工作的高吞吐量，Quiver 与 DLT 框架(如 PyTorch 或 Tensorflow)紧密耦合形成了协同设计缓存。通过修改 DLT 框架，缓存客户端深入集成到 DLT 作业的 I/O 访问中，并与缓存服务器共享更丰富的信息。Quiver 的架构如图 2，其中缓存服务器的运行在所有虚拟机上，但在它们自己的容器中。Quiver 客户端运行在 DLT 作业的地址空间中，并包含对 DLT 框架的更改。用户的输入训练数据在缓冲未命中时从云存储中获取。每个作业的数据集被分片到多个缓存服务器上，并使用内容哈希进行查找。

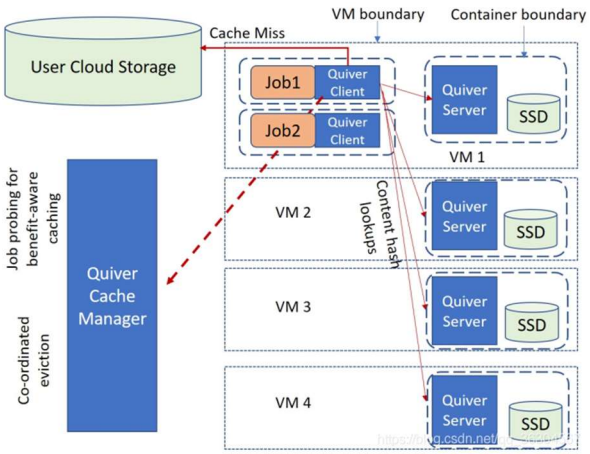

图 2 Quiver 的架构

使用内容哈希寻址的好处是，多个副本上的相同的数据项(比如 ImageNet 数据在不同用户的不同存储帐户中的副本)可以映射到相同的散列，从而允许跨用户重用，这也保证了用户数据的隔离。对于用户拥有的每个数据集，用户计算每个数据项的内容哈希，并将这些哈希存储在摘要文件中。摘要文件包含形式为<content_hash: file_location>的条目，其中 file_location 表示特定数据项驻留在该特定用户的云存储帐户中的路径和偏移量。

由于 Quiver 是一个分布式缓存，它需要协调收回和放置决策，以便所有缓存服务器大致同意缓存数据集的哪些部分。Quiver 中的缓存管理器与 Quiver 客户机和 Quiver 服务器交互，以协调这些决策。缓存管理器还负责通过探测 DLT 作业来衡量每个作业从缓存中可能获得的好处。它通过指示缓存服务器临时返回由 DLT 作业在几个小批量中读取的所有数据的缓存未命中来实现这一点。然后，它将这个执行时间与使用缓存的正常操作期间的时间进行比较，并使用这个时间来确定缓存放置的优先级。

2.3 确保云上共享物理主机的服务质量

PARTIES 由一个检测组件和一个资源分配组件组成。前者检测每个应用的尾延迟、内存容量、网络带宽使用量，后者使用检测结果来决定恰当的资源分配，并强制对它们隔离。

PARTIES 是基于反馈的控制器，它利用细粒度的检测与资源划分动态地调整混布的具有延迟敏感(LC)的应用间的资源分配，以满足所有应用的 QoS 为目标。其设计有四个原则，第一，资源的分配决策是动态的且是细粒度的。LC 应用对资源分配很敏感，次优的决策即便很少也能导致 QoS 违规。细粒度的检测可侦测出这源需求的激增，并阻

止这些激增的发生。第二，不需要应用的先验知识或和关于它的描述。对所有可能的应用混布中创建离线的描述，即便可行的其代价也会极其高昂。此外，获得此项信息也不总是可行的，尤其在公有云环境中。相反地，资源置换性使 PARTIES 能够在线地找出可行的分配，而不依赖经验为每个应用调参。第三，控制器从错误的决策中快速地恢复。因为 PARTIES 在线地探索了分配的空间，它的一些决策可能会适得其反。通过利用细粒度的在线检测，PARTIES 可以快速地侦测出这些事件并恢复。第四，万不得已时才迁移。当混布应用的资源需求总额超过了机器的总容量，就无法满足所有服务的 QoS，负载迁移就是唯一的补救措施。由于迁移的高昂开销，PARTIES 会选择那些性能受迁移影响最小的应用去做，因为这个应用要么是无状态的，要么有一个松散的 QoS 目标。

3 研究进展

3.1 RIBBON优化实例池配置的方法

3.1.1 RIBBON 的 BO 引擎

RIBBON 的 BO 引擎在初始情况下并未获知未知目标函数的性质和分布。BO 通过其代理模型维持了一种先验的信念。该代理模型作为真实目标函数的代理。BO 的成功与否取决于在优化过程中代理模型能否逐渐逼近真实目标函数。RIBBON 选择高斯过程（GP）作为其代理模型，因为它适应地表示任何随机过程，并且不会因为输入或目标而受到限制，这使得 RIBBON 能够处理各种不同的云实例中不断变化的请求数据流。

当 RIBBON 开始采样实例池中的具体配置时，它收集这些配置对应的真实目标函数的值。RIBBON 使用这些真实值更新代理模型在采样配置处的值。代理模型使用其协方差核对尚未采样的配置的函数值进行外推。该外推值受两种因素的影响，其一是已采样配置的协方差，其二是已采样配置与尚未采样配置的差距。RIBBON 选择 Matern 5/2 协方差核来确保平滑度，同样也能保证相似配置能有相近的目标函数值，假设配置的目标函数值遵循单调或特定类型的多项式分布，则可以选择的协方差核也包括点积和有理二次型，根据实际目标函数值的分布，这两种协方差核不适用于 RIBBON。

在评估每个配置样本后，RIBBON 的代理模型获得更新，代理模型将置信度值与未采样的配置相关联。RIBBON 需要智能采样，以实现最佳配置，为了实现智能采样的功能，需要利用 BO 中一个关联的采样函数，它能帮助 RIBBON 确定下一个样本。该采样函数考虑了代理模型中尚未采样的配置的置信度，更高的置信度意味着代理模型表示真实目标函数的可能性更大。RIBBON 使用期望改进（EI）作为其采样函数。对于每个未探索的配置，EI 使用其 GP 均值和方差作为输入，并计算相对于当前最佳配置的预期改进，BO 引擎将取样拥有最高提升的配置作为新的样本。以上所描述贝叶斯优化过程如图 3。

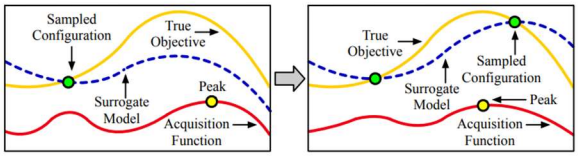

图 3 RIBBON 的贝叶斯优化过程

3.1.2 RIBBON 目标方程的设计及收敛性保证

由于 RIBBON 的优化性能依赖于目标函数的取值，因此设计合适的目标函数来进行 RIBBON 优化的度量是非常重要的。由于 RIBBON 的目标是能在满足 QoS 目标的前提下最大化成本收益，而这两个目标并不能如同使用传统的优化方程一样找到评估的最大值或最小值。因此需要设计一种新的目标函数。

RIBBON 所使用的目标函数如公式(1)所示，其核心思想是在满足 QoS 目标的前提下尽可能降低云服务花费。式中的x表示各类实例数量的集合，$R_{sat}(x)$是当前选择的配置样本所达到的 QoS 达标率，常量T_{qos}为 QoS 目标，而常量p_i和m_i分别为单位时间价格和i类型的实例的数量上限。

$$f(x)=\begin{cases}\frac{1}{2}\cdot\frac{R_{sat}(x)}{T_{qos}} & \text{if violates QoS,}\\ \frac{1}{2}+\frac{1}{2}\cdot\left(1-\frac{\sum_{i=1}^{n}p_i\cdot x_i}{\sum_{i=1}^{n}p_i\cdot m_i}\right) & \text{otherwise.}\end{cases}$$

在默认情况下，贝叶斯优化所处理的变量都具有连续的值，然而在 RIBBON 所处理的样本中，所处理的数据为每种实例的数量，其数值类型为整数，为解决这个问题，RIBBON 对 BO 底层的有 GP 内核中的连续值进行取整，如下公式(2)所示。

$$k'(x_i,x_j)=k(R(x_i),R(x_j))$$

此处$R(x)$为取整函数，其将变量x转化为离它最近的整数，这保证了 GP 的函数图像更好的匹配

真实目标函数。

以上关于目标函数的设计和类型变量的处理使得优化过程是可行且收敛的。如果使用非光滑目标函数，BO 的取样函数优化器在 35%的情况下失败，并返回无效值。同样地，如果 RIBBON 放弃上述的类型变量处理方法，它将对落在先前采样配置的相同整数范围内的配置执行重复采样，从而产生比穷尽搜索还要多 30%的总样本。使用 RIBBON 的设计，在所有情况下都能快速收敛。

3.2 Quiver的缓存管理

3.2.1 可替换命中

当只有数据集的一部分(比如 10%)被缓存时，Quiver 会在单个 epoch 内对数据集的排列索引列表进行多次传递。为了在第二次传递期间获得良好的命中率，必须在第二次传递期间缓存数据集的不同部分。在多个 DLT 作业(例如，进行超参数探测的多个作业)访问同一数据集的场景中，这是很棘手的，因为不同的作业可能会在不同的时间耗尽它们对排列索引列表的第一次传递。

Quiver 通过为数据集的两个块分配缓存空间来处理这个问题，并使用类似于双缓冲的技术。首先，表示完整数据集的摘要文件被划分为固定数量的块，使得每个数据块占数据集的 10%。数据集的分块必须智能地完成，以确保每个数据块内输入数据的随机性。一些数据集如 LibriSpeech 按序列长度对数据项排序;按照逻辑字节顺序对它们进行分块将导致第一个块完全由短序列组成，从而影响随机性。递归神经网络(RNN)要求小批内的所有输入具有相同的序列长度;如果一个小批包含不同序列长度的输入(如，随机选择的输入)，它们填充所有输入以匹配小批中最长输入的长度。因此，为了提高计算效率，让小批中的所有输入长度大致相同是有意义的。为了在小批中高效地存储输入，我们将块定义为条带分区;让我们将输入数据集的每一个连续的 10%作为一个分区。每个分区被分成 10 个条带单元;逻辑块就是将相应的条带单元拼接到每个分区中而形成的完整条带。为了保证序列长度的同质性，同时也确保输入的均匀分布，尽可能地，一个小批完全由单个条带单元的输入组成。

数据集分块允许跨多个作业协调访问缓存。当作业对第一个块进行操作时，第二个块被带入缓存，以便在作业切换到下一个传递时准备好。一个重要的问题是何时从缓存中取出第一个块。如果太早被赶出，仍然在第一次执行并访问该块的作业子集将不命中，而如果它在缓存中停留太长时间，则无法预加载下一个(第三个)块。Quiver 使用两步流程来处理驱逐。当数据集的另一个数据块完全加载到缓存中时，一个数据块被标记为替换;现在所有的新工作都只会从最近一个块命中。然而，仍然在运行第一个块的现有作业将继续从第一个块上命中。当所有现有的作业都用尽了它们对第一个块的传递(并通知缓存服务器)时，第一个块才会实际上被驱逐。此时，可以开始对数据集的第三个块进行预加载。

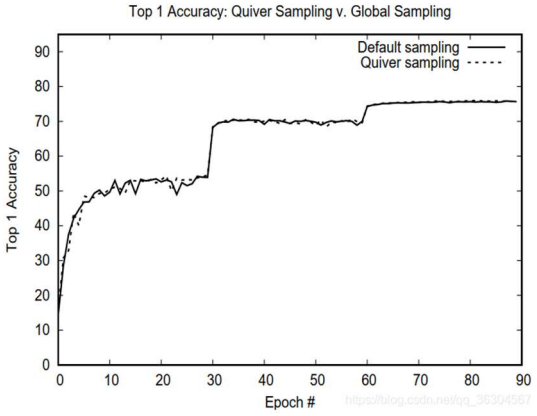

图 4 默认取样方法和 Quiver 取样方法的 Top-1 精度变化

替换命中自然会产生一个问题，那就是它是否会影响训练的准确性。图 4 所示的实验结果说明了 Quiver 中的可替换缓存(即，将 shuffle 限制为数据集的一部分而不是整个数据集)对准确性没有影响。从图中可以看出，默认采样方法下和 Quiver 采样方法下 top-1 的精度曲线非常吻合。表 1 显示了两种配置的最终 top-1 和 top-5 精度，Quiver 采样实现了与全局随机采样相同的精度。

表 1 两种采样方式的 Top-1 精度和 Top-5 精度

Config	Top-1 Acc. (%)	Top-5 Acc. (%)
Baseline sampling	75.87	92.82
Quiver sampling	75.89	92.76

3.2.2 合作缓存未命中处理

一种对存储带宽要求很高的常见工作负载是多作业，其中 DLT 用户在同一数据集上为同一模型运行数十个或数百个作业，但使用不同的超参数配置。如果没有 Quiver，这每一个作业会从远程存储读取同一数据，导致远程存储成为瓶颈，导致每个作业的 IO 吞吐量很低。Quiver 使用合作未命中处理，它跨多个作业对缓存获取进行分片，以避免多个作业对同一数据项进行多次获取。这种分片是通过简单地随机化未命中文件的获取顺序来隐式完

成的，从而避免了独立的作业之间的直接协调。因此，每个作业首先检查缓存中是否存在一组数据项，然后读取这些数据项的随机子集，并将读数据项添加到缓存中。添加之后，它将执行另一个缓存查找，但这一次，它不仅会得到所添加的数据项，而且还会得到由执行类似随机获取的其他作业同时添加的其他数据项(大部分不重叠)。因此，即使在冷缓存的情况下，或者如果整个数据集不能放入缓存，Quiver 也可以通过保留远程存储带宽提供好处，在单个 epoch 内跨多个作业读取大多数数据项。Quiver 所使用的可替换缓存命中和协同未命中处理算法如算法 1 所示。

Algorithm 1 Substitutable hits & Co-operative miss handling

```
global gChunkIndex = -1
▷ Returns: List of indices of data items to be fetched for current mini-batch
function GETBATCH(SIZE)
    ▷ Try to randomly sample 10 x size unused elements
    pendingIndices = getPendingIndices(size * 10)
    cacheHits = cacheClient.lookup(pendingIndices)
    if len(cacheHits) >= size then
        return pickAndMarkUsed(cacheHits, size)
    end if
    ▷ Not enough cache hits, perform co-operative
    ▷ cache miss handling
    result = List()
    result.addAll(
        pickAndMarkUsed(cacheHits, len(cacheHits)))
    if gChunkIndex < 0 then
        ▷ cacheManager returns 0 if no chunk is cached
        gChunkIndex =
          cacheManager.getCurrentChunk(datasetId)
    end if
    chunksChecked=0
    while chunksChecked < totalChunks do
        ▷ Tell cache servers that I am using this chunk
        ▷ (if not done already)
        informServers(jobId, datasetId, gChunkIndex)
        unusedIndices = getRandomUnusedIndices (
            gChunkIndex, size - len(result))
        if len(unusedIndices) == 0 then
            informServersDoneUsingChunk(
                jobId, datasetId, gChunkIndex )
        end if
        result.append(unusedIndices)
        if len(result) == size then
            return result
        end if
        gChunkIndex =
            (gChunkIndex + 1) % totalChunks
        ++chunksChecked
    end while
end function
```


3.2.3 收益感知的缓存设计

当缓存总空间受到限制时，Quiver 利用作业异构性优先将缓存空间分配给从缓存中获益最多的作业。DLT 任务同时执行计算(在 GPU 上)和 IO。直观上，如果计算时间高于从远程存储读取的 IO 时间，那么 IO 时间可以重叠，无论从缓存读取还是从远程存储读取，作业性能都是相同的。然而，这是一个复杂的现象，因为它取决于作业的并行度(即它运行的 GPU 数量)，模型有多大，模型是否以流水线计算和 IO 的方式编写，等等。

而与 DLT 框架的紧密集成使 Quiver 能够智能地探测和测量有缓存和没有缓存的作业性能。当一个新的作业请求将条目添加到缓存中时，缓存管理器将挑选作业进行探测。探测分为两个步骤。在第一步中，缓存管理器指示所有缓存服务器拒绝该作业的所有缓存查找，从而迫使作业从远程存储中获取数据。在这个探测阶段的最后，例如，100 个小批，缓存管理器从缓存客户端(作为 DLT 作业的一部分运行)获得总的运行时间。然后，缓存管理器会使用默认缓存策略定期监视作业的性能。如果使用默认缓存策略和不使用缓存的时间相差不大，那么它就会得出结论，该作业在远程 IO 带宽上没有遇到瓶颈，并决定关闭该作业的缓存。因此，只与此类作业接触的数据集将永远不会进入缓存，从而为其他有助于作业性能的数据集释放空间。Quiver 不仅在作业开始时运行探测阶段，而且是周期性地，因为有效 I/O 吞吐量有可能会因为远程存储的负载增加（例如，新的作业从同一存储读取）而减少，因此使得作业对 IO 性能更加敏感，反之亦然。

设t_h^i为作业i在缓存命中下每小批的平均时间，t_m^i为作业i在缓存未命中下对应的时间。则作业i的缓存收益为$b^i = t_m^i/t_h^i$。假设n^i为作业i占用的 GPU 数量，则缓存作业i的数据集为作业i节省的 GPU 资源为$g^i = b^i * n^i$。对于每个数据集D^k，集群中可能有多个作业访问同一数据集。由于缓存是由所有这些作业共享的，所以如果有 N 个作业访问D^k，缓存数据集所节省的 GPU 资源总数为$G_{D^k} = \sum_{i=0}^{N} g^i$。有趣的是,缓存管理器必须只在三个选项中为每个数据集做决定，第一，完全缓存（空间成本为数据集的整个大小）；第二，通过缓存一个固定大小的块(如，15G)或 10%的数据集(双缓冲技术的成本是 2 个块)中较小的一个使能合作未命中；第三，没有缓存(零成本)。其中缓存的中间大小是没有用的，因为考虑到 Quiver 中可替换的缓存命中，缓存两个块的好处是一样的。给定整个集群范围内的缓存空间预算 S，缓存管理器使用贪婪算法优先将缓存空间分配给效益/成本比率最高的数据集或数据集块。

3.3 延迟敏感应用的资源分配

PARTIES 中检测组件和资源分配组件的主要操作是控制器从公平的分配开始，即每个应用收到相等的划分，且所有处理器运行在标称频率上。初始化后，尾延迟及资源利用率每 500 毫秒采样一次，并基于测量值和每个应用的尾延迟的松弛度来调整资源，包含三个主要操作：第一，如果至少有一

个应用几乎没有或有负的松弛度，即 QoS 被违反或将被违反，PARTIES 从有最小松弛度的应用 S 开始，分派更多的资源给这个应用，记为 upsize()。第二，当所有应用轻易地满足了他们的 QoS，PARTIES 将减少有最高尾延迟松弛度的应用 L 所分到的资源。回收过量的资源可降低功耗，或者空出资源给混布的 job，这提升了机器效能，为 downsize()。第三，另有一个计时器以记录 QoS 违反持续了多久，若满足 QoS，计时器被重置；若在 1 分钟内无法找出满足所有应用 QoS 的分配，迁移会被触发以降低服务器负载并阻止性能衰减。

而迁移的过程如下，首先选择一个待迁移的应用，之后在一个低负载的机器上创建该应用的实例，将请求从当前实例重定向到新的实例，最后终止之前的实例。PARTIES 会选择迁移一个产生最少迁移开销的应用，无状态的服务（比有状态的）不需要迁移内存中的数据，这引入了较低的迁移开销，当所有混布任务是有状态的，选择迁移有最松散 QoS 目标的。

4 总结和展望

根据前述的各种有关云服务上的深度学习任务优化方法，首先，RIBBON 第一个提出了采用构建云计算实例池的方式来显著提高深度学习模型推理的成本效益和性能效率，其基于贝叶斯优化的方法来查找最佳实例配置，平衡了两个相互抵触的目标，即满足 QoS 目标和最小化成本。经过实验评估，RIBBON 同样也可以快速适应负载变化。其次，当下的深度学习系统主要关注于提高计算效率和网络效率，存储层主要由特别的解决方案来处理，例如将数据手动分段到本地 SSD，这有很大的局限性。借助 Quiver 提供了一种自动缓存机制，在深度学习不断增长的计算能力面前，帮助弥补存储性能的差距。Quiver 通过与深度学习工作流和框架紧密集成来实现缓存管理，并利用 IO 可替换性等特性来确保高效缓存。最后，对于 PARTIES 所提出的资源置换性有很重要的意义，其实际上反映出一种等价性，对于某个目标(类似于 QoS)，如果也受到多重因素的影响，并且在调整不同核心数、频率等参数后仍维持在一个恒定的值，那么就能实现更灵活的资源划分机制。

参 考 文 献

[1] Baolin Li, Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, Karen Gettings, and Devesh Tiwari. 2021. RIBBON: cost-effective and qos-aware deep learning model inference using a diverse pool of cloud computing instances. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21). Association for Computing Machinery, New York, NY, USA, Article 24, 1–13. DOI:https://doi.org/10.1145/3458817.3476168

[2] Abhishek Vijaya Kumar, & Muthian Sivathanu (2020). Quiver: An Informed Storage Cache for Deep Learning. In 18th USENIX Conference on File and Storage Technologies (FAST 20) (pp. 283–296). USENIX Association.

[3] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). Association for Computing Machinery, New York, NY, USA, 107–120. DOI:https://doi.org/10.1145/3297858.3304005