

# 面向 CPU-GPU 集群的分布式机器学习资源调度框架研究

朱紫钰, 汤小春, 赵全

(西北工业大学 计算机学院, 陕西 西安 710072)

**摘要:**随着 GPU 硬件设施的广泛应用,越来越多的分布式机器学习应用程序开始使用 CPU-GPU 混合集群资源来提高算法的效率。但是,现有的分布式机器学习调度框架要么只考虑 CPU 资源上的任务调度,要么只考虑 GPU 资源上的任务调度,即使综合考虑 CPU 与 GPU 资源的不同,也很难提高整个系统的资源使用效率,即使用 CPU-GPU 集群进行分布式机器学习作业面临的关键挑战是如何高效地调度作业中的任务。在对现有的方法进行分析后,提出了一种基于不均匀数据分片的策略,利用线性规划的原理,使得 CPU 任务时间与 GPU 任务时间尽可能接近,从而减少分布式机器学习作业的整体执行时间。介绍了 CPU-GPU 混合计算框架的调度结构,这种调度结构针对 CPU 计算能力与 GPU 计算能力的不同特点,将数据分割成大小不等的的数据分片以适应于 CPU 和 GPU 计算资源,给出了 CPU-GPU 混合资源下的任务调度方法,对该方法进行 K-Means 算法验证。使用 CPU-GPU 混合资源计算框架,K-Means 性能平均提高 1.5 倍,且随着 GPU 数量的增加,K-Means 性能能够显著提升。

**关键词:**异构任务;一体化调度;聚类算法;分布式

中图分类号:TP31

文献标志码:A

文章编号:1000-2758(2021)03-0529-10

分布式机器学习<sup>[1-3]</sup>在数据挖掘、医学影像、股票市场分析、计算机视觉、市场分析等领域被大规模使用。但是训练机器学习模型通常需要大量的时间和计算资源。随着数据中心开始大规模使用 GPU 设备,CPU-GPU 混合的异构资源集群已经成为一种通用基础设施<sup>[4-5]</sup>。企业界为了提高分布式机器学习的效率,也逐渐将这类应用移植到异构集群平台上运行。但是,CPU-GPU 集群上面临的一个基本挑战是如何高效地调度分布式机器学习作业中的任务。特别是当 GPU 计算资源较为珍贵时,如何最大程度地利用 CPU-GPU 计算资源进行模型训练已成为异构集群上面临的基本挑战。

早期的机器学习通常运行在单个节点上,其主要流程是不断迭代 2 种类型的操作,一种是梯度计算操作,另外一种误差的聚合操作。随着数据规模的增大,分布式机器学习逐渐成为主流。分布式机器学习采用主-从结构,将梯度计算操作分散到

从节点上,即集群工作节点并行运行有关误差计算的操作,具体表示为: $g_i^k \leftarrow \sum_{i \in I_k} \partial f_i(w_i)$ ,其含义是在第  $i$  次迭代过程中,在第  $k$  工作点训练数据  $I_k$  时产生的梯度误差  $g_i^k$ 。误差聚合操作在主节点上进行,主节点通常也称为参数服务器(PS),具体表示为: $g_i = \sum_{m=1}^k g_m^k$ ,其含义是将各个工作节点得到的梯度误差进行聚集运算。

分布式机器学习初期,由于硬件限制,主要采用 CPU 集群来进行,针对一批训练数据,计算框架把数据分成很多个片段,然后根据集群节点上 CPU 的数量,将每一个片段作为一个单独的任务分配到计算节点的不同 CPU 上,进行局部梯度计算。当所有的任务完成计算后,参数服务器收集各个任务的梯度误差进行汇总,然后发送新的参数用于下一次迭代。由于数据分片大小相同,CPU 性能相同,所以每个任务在 CPU 上的执行时间大致相同,这能够保

证每一次迭代的同步。

近几年来,随着 GPU 硬件设施的广泛应用,分布式机器学习开始使用 GPU 计算资源。在分布式机器学习的训练过程中,一部分任务可能运行在 GPU 上,一部分任务可能运行在 CPU 上。GPU 用于单纯的梯度计算,CPU 用于初始化、节点之间通信、数据加载、主机内存到设备内存之间拷贝以及启动 GPU 的 kernel 函数等。

面对同构的集群环境,分布式机器学习采用大小相同的数据分片策略,这样可以保证任务在 CPU 或者 GPU 上执行时间相同。面对异构集群环境,分布式机器学习存在学习过程中不能保证 CPU 和 GPU 资源合理利用的问题:如果只考虑使用异构集群中的 CPU 计算资源,则没有考虑到 GPU 计算资源带来的计算优势;如果只考虑使用异构集群中的 GPU 计算资源,则在 GPU 设备执行任务时,CPU 资源往往被闲置。无论是在异构集群中只使用 CPU 资源还是只使用 GPU 资源,都无法充分利用异构集群资源。因此,一种合理使用异构集群计算资源的方式为:当 CPU 资源存在空闲时,将任务分配到 CPU 上执行,当 GPU 资源存在空闲时,将任务分配到 GPU 上运行。这种资源分配方案的优点是最大限度地利用了 CPU 计算资源和 GPU 计算资源。但是上述任务分配方式的最大挑战在于:(1)由于 GPU 和 CPU 具有不同的运算能力(FLOPS),导致同样的任务在 GPU 上执行比在 CPU 上执行消耗的时间更短;(2)类似于聚类分析的迭代过程,只有当全部工作节点上的任务完成后,误差聚合操作才能进行参数的更新,并发布下一次的迭代过程。因此,如果把每一次迭代过程中的计算任务均匀地分配给 CPU 和 GPU 资源,就会造成 CPU 任务严重滞后,迭代过程不能同步,从而导致误差聚合操作的推迟,影响下一次迭代的开始时间。

针对分布式机器学习作业如何提高资源利用率的问题,有相当多的研究者进行了研究。文献[6]从分布式机器学习训练过程出发,详细说明了训练过程中各个阶段使用 CPU 和 GPU 资源带来的性能提升,同时也说明了使用 GPU 设备的瓶颈。文献[7-14]主要针对 GPU 计算资源的调度策略,忽略 CPU 计算资源的调度。文献[7]介绍了跨作业共享 GPU 资源带来的一些挑战性问题。文献[8]描述了一种大规模快速梯度下降算法的应用场景,提出了大规模数据集在多个 GPU 之间的调度策略。文献

[9]提供了与分布式机器学习相关的资源管理模型和以及任务调度策略。文献[10]提供了一种任务执行过程的控制算法,保证机器学习作业的质量。文献[11]提出了一种针对异构资源的混合 DRF 算法,能够有效提高集群资源利用率。文献[12]提出了一种支持向量机的任务分类算法,任务根据类型划分并投入相应的设备进行计算,可以改善集群资源利用情况,但是该算法获得数据存在一定的时间延迟,并不能准确获得集群资源变化。这些研究在最大化资源利用率、最小化作业的平均完成时间以及任务调度质量等方面都有一定的借鉴意义。但是,上述文献对 CPU 和 GPU 搭配使用的研究不全面,即要么只考虑 CPU 资源,要么只考虑 GPU 资源,不考虑 CPU 资源与 GPU 资源混合分配和调度的价值,也不关注 CPU 和 GPU 资源混合调度的利用率。文献[13-14]主要任务是解决多个机器学习作业共享 GPU 集群时,如何分配资源,减少迭代过程的延迟。它解决了不同作业之间的资源共享问题,但没有解决 CPU-GPU 资源之间的合理分配问题。文献[15]介绍了一种通用的 CPU-GPU 计算资源管理系统,它不涉及具体的应用,而是根据任务类型(例如工作节点程序,参数服务器程序)执行启发式静态资源分配,因此无法提高机器学习中 CPU-GPU 资源的利用率。文献[16]通过预估任务的执行时间来调度资源,但是它的核心在于确定参数服务器与计算节点之间的分配关系,不太关注 CPU 资源与 GPU 资源分配问题。文献[17]介绍了一种通过 MapReduce 来加速 K-Means 的算法,其按照 CPU 处理能力和 GPU 处理能力将 Map 任务分成 2 个部分,这种方式可以利用 CPU 资源和 GPU 资源,但是它对 CPU 任务和 GPU 任务的不敏感,导致迭代过程中 CPU 任务的延迟,增加了整个作业的响应时间。文献[18]中的理论框架产生了一个最佳整数规划解决方案,根据 CPU 数量、CPU 与 GPU 之间计算能力的不同,少部分任务由 CPU 承担,大部分任务由 GPU 承担,提高了 CPU 和 GPU 的利用率,但是由于 CPU 任务的数据大小与 GPU 任务的数据大小相同,当数据分片较大时,存在任务延迟问题。

为了提高 CPU 和 GPU 资源利用率,同时也减少每次迭代中的任务延迟,本文提出了一种不均匀数据分片方法。在每一次迭代过程中,根据集群系统可以使用的 CPU 核数量和 GPU 数量,以及 CPU 和 GPU 的计算能力,将数据块分解成大小不同的分

片,小分片数据应用于 CPU 资源,大分片数据应用于 GPU 资源,在所有 GPU 任务执行结束前,保证 CPU 任务也能够结束,从而减少 2 类不同任务之间的同步等待问题,进而降低分布式学习的训练时间,且提高系统的资源使用率。

论文的主要创新点是提出了一种面向 CPU-GPU 混合集群的分布式机器学习调度框架,即在机器学习模型参数的训练过程中,对于每个任务,系统都包含 2 个版本——GPU 版本和 CPU 版本。在每一次迭代操作开始前,首先得到可以使用的 CPU 设备数量  $m$  以及 GPU 设备数量  $n$ ,再根据任务的历史统计数据,确定 CPU 和 GPU 上运行时间的比值(这个比值相当于 CPU 和 GPU 执行效率比),根据比值可将任务分解为  $p$  和  $q$  个;然后将 GPU 任务提交到 GPU 计算资源,CPU 任务提交到 CPU 计算资源。最后,采用线性规划的思想,保证分配的 CPU 任务与 GPU 任务之间的同步执行,即 CPU 任务与 GPU 任务之间不存在严重滞后的情况。

## 1 分布式机器学习的资源调度框架

### 1.1 CPU-GPU 资源分配模型

通常情况下,当使用 GPU 计算资源计算不同数据分片的梯度时,CPU 被用来初始化参数,启动 kernel 函数的运行。在 GPU 的 kernel 函数执行过程中,CPU 就不再被使用。这会导致 CPU 处于闲置状态,降低系统资源利用效率。但是,如果所有的 CPU 资源都处于“忙”状态时,也会造成 GPU 任务等待 CPU 资源而不能被迅速执行的情况,这会浪费 GPU 资源。

在机器学习训练中,要想 CPU 资源和 GPU 资源充分被利用,就需要考虑为 GPU 绑定一定的 CPU 资源,方便 GPU 任务的启动。在面向 CPU-GPU 的异构集群系统中,图 1 给出了 CPU 与 GPU 之间的资源分配关系。图中的 c0、c1 等代表 CPU 设备,GPU1、GPU2 等代表 GPU 设备。在每次的迭代过程中,将 CPU 和 GPU 进行捆绑,即为每个 GPU 搭配一定数量的 CPU。训练开始后,对于 GPU 任务,绑定的 CPU 只负责加载参数以及启动 GPU 的 kernel 函数,一旦开始在 GPU 上进行误差的计算,CPU 会处于闲置状态,直到完成所有的局部误差计算。之后在指定的 CPU 或者 GPU 上进行误差聚合运算,产生新的模型参数。上述过程相当于完成一次迭

代,如果误差未达到模型精度要求,则会进行重复的迭代过程,直到满足精度要求。

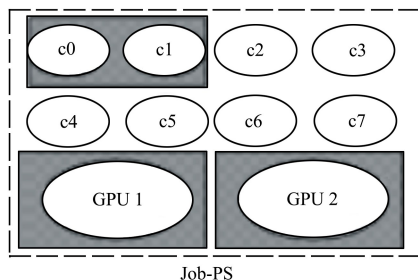


图 1 CPU-GPU 异构集群系统资源分配方式

### 1.2 GPU 任务与 CPU 任务的资源需求

CPU-GPU 异构集群要求任务的二进制程序既能够在 CPU 上运行,也能够在 GPU 上运行。当一个机器学习作业准备执行时,可以指定哪些任务运行在 CPU 设备上,哪些任务运行在 GPU 设备上。任务调度器会查看集群资源中是否有设备处于空闲状态,如果有 GPU 空闲,则提交 GPU 任务,如果有 CPU 空闲,则提交 CPU 任务。任务调度器对任务进行资源分配时,如果需要使用 GPU 资源,那么任务就与 GPU 二进制程序绑定;如果需要使用 CPU 资源,那么任务就与 CPU 二进制程序绑定。之后,任务调度器会将任务及信息发送给工作节点的 CPU 执行器或 GPU 执行器,由执行器启动并执行任务。在设计 CPU 二进制程序和 GPU 二进制程序时,任务的输入数据格式和输出数据格式要求完全一样。这样一来,2 种任务的输入输出能够正确获得下一次迭代时新的训练参数,且不会影响机器学习作业的参数更新效果。

### 1.3 数据分片策略

数据分片策略包含 2 个目的:①面向分布式机器学习,通过数据分片,可以并行执行多个梯度计算过程,从而减少机器学习的训练时间,快速地完成模型的训练;②面向 CPU-GPU 异构集群环境,进行数据分片的目的是充分利用 CPU 资源和 GPU 资源。

为了达到上述目的,依据 CPU 任务和 GPU 任务的运行时间,对数据进行大小不等的分片。通过统计 CPU 任务和 GPU 任务在各自设备上的运行时间,得到任务运行效率的比值,从而确定数据分片的大小。由于机器学习聚合操作更新参数前要求本批次数据全部计算完成,因此分片大小决定了 CPU 任务和 GPU 任务的同步快慢问题,这是本文算法的关

键所在。

### 1.4 分片方法

减少机器学习迭代过程中运行时间的核心是保证 CPU 任务与 GPU 任务的同步完成。假设  $N$  是整个数据规模的大小,  $n$  是 CPU 核数量,  $m$  是 GPU 设备数量, 对于同样大小的一块数据  $B$ , CPU 运行时间与 GPU 运行时间的比值是  $\alpha$ , 称  $\alpha$  为加速系数。其含义是如果使用一个 GPU 设备完成数据的计算需要时间为  $t$ , 那么使用一个 CPU 完成数据的计算需要  $\alpha \cdot t$  的时间。如果可以将数据分割成  $B/\alpha$  的大小, 那么一个 CPU 完成  $B/\alpha$  数据的计算就只需要时间  $t$ , 这样对于 GPU 任务和 CPU 任务的计算就不存在执行时间差别的问题, 很容易通过调度满足迭代过程的同步。

对于机器学习中的数据  $N$ , 现在需要进行数据的分片, 分片后的每一个数据块对应一个任务。为了保证计算性能的无差异, 根据数据分片的大小决定任务由 CPU 执行还是由 GPU 执行。假设数据被分割后, 有  $x$  个 CPU 任务和  $y$  个 GPU 任务。在  $n$  个 CPU 核上运行  $x$  个任务的完成时间记为  $t_c$ , 在  $m$  个 GPU 上完成  $y$  任务的时间为  $t_g$ , 理想状态下, 数据分割计划的目的是  $|t_c - t_g| = 0$ 。因此, 任务同步时间差如下所示

$$f(x, y) = \frac{x}{n} \cdot \alpha \cdot t - \frac{y}{m} \cdot t \quad (1)$$

使用以上参数, 需要确定 CPU 任务的个数  $x$  和 GPU 任务的个数  $y$ , 目的是要减少 CPU 任务的执行结束时间与 GPU 任务执行结束时间的差值。所以给出以下的规划条件, 其目标是

$$\min |f(x, y)| \quad (2)$$

同时要满足条件

$$x \cdot B_1 + y \cdot B_2 \geq N \quad (3)$$

$$|B_2 - \alpha B_1| < \eta \quad (4)$$

$$B_1, B_2, x, y \geq 0 \quad (5)$$

公式(2)给出了目标函数, 即减少 CPU 任务与 GPU 任务之间的不同步。其物理意义是, 假设  $t_p$  代表  $p$  个任务中第一个任务开始到最后一个任务结束的时间区间, 而  $t_q$  代表  $q$  个任务中第一个任务开始到最后一个任务结束的时间区间。调度的目的是满足  $|t_p - t_q| < \varepsilon, \varepsilon \in \forall (0, 1)$ , 即 GPU 任务的完成时间和 CPU 任务的完成时间相当。公式(3)是按数据分片要求, 将 CPU 任务的数据分片大小为  $B_1$ , GPU 任务的数据分片大小为  $B_2$ , 公式(3)要求  $x + y$

个任务要处理完全部的训练数据。公式(4)描述了 CPU 和 GPU 任务数据规模之间的关系, 其数据要满足加速系数要求,  $\eta$  表示数据分割误差。公式(5)对所有变量的取值范围进行了约束。

对于每一个数据分片, 都会产生一个任务, 分片小的数据绑定 CPU 任务, 分片大的数据绑定 GPU 任务。整个数据分片完成后, 将任务及数据信息分别插入到 CPU 任务队列和 GPU 任务队列中, 等待调度器调度。根据数据分片产生的任务, 只能用于机器学习中的局部梯度计算, 即  $g_i$  值的计算。基于 CPU 任务和 GPU 任务的数量, 调度器安排 CPU 资源以及 GPU 资源执行任务。

### 1.5 算法分析

迭代计算过程中, CPU 负责内存到设备内存的输入和输出, GPU 负责计算。当 GPU 计算时, CPU 空闲, 因此, 利用这些 CPU 时间是可行的, 但是前提是不能影响 GPU 执行前后对 CPU 的需求; 另外, GPU 数量一般较少, CPU 核数量众多, 部分 CPU 核就可以承担 GPU 计算前后的输入和输出工作, 多余的 CPU 核就可以参与计算, 因此 CPU-GPU 混合的算法是可行的。

为了利用这些 CPU 核, 就需要对数据进行合理的分片, 避免部分任务滞后导致的作业响应时间延迟。所以, 论文采用整数规划算法来计算分片的数量和分片的大小。

在时间复杂度方面, 在公式(3)中, 定义了 2 类变量  $B_1$  和  $B_2$ , 设其可取最大值为  $N$ , 由公式(4)可知 2 类变量满足比例关系  $\alpha$ 。设  $x, y$  和  $N$  可取最大值为  $a$ , 那么分片规模循环次数为  $a$ , 任务数作为内层循环也为  $a$ , 则公式(3)时间复杂度为  $O(a^2)$ , 该时间复杂度为算法时间复杂度, 因此本算法可行。在空间复杂度方面, 算法使用的主要变量共 8 个, 涉及的数量较少, 因此占用内存较少, 算法具有较好的空间复杂度。

### 1.6 机器学习作业的任务调度框架

机器学习作业的任务调度框架由 3 个主要部分组成(ML 框架、工作节点和数据分片)。ML 框架是主控节点, 工作节点是由 CPU-GPU 资源组成的集群节点, 数据分片是对整个训练数据的划分, 其分片大小与计算资源的计算能力相关。整个框架如图 2 所示。

ML 框架: ML 框架主要功能有①聚合各个工作节点执行器上发送的局部梯度值; ②根据梯度值更

新模型参数;③对训练数据进行分片,并依据分片创建CPU任务和GPU任务的描述;④对上述任务进行调度。

**工作节点:**工作节点的主要功能是依据CPU资源和GPU资源的不同,启动CPU执行器以及GPU执行器。CPU执行器接收CPU任务描述,使用CPU资源来计算指定分片数据的局部梯度值;GPU执行器接收GPU任务描述,采用GPU资源来计算指定分片数据的局部梯度值。

**数据分片:**数据分片是按照CPU资源和GPU资源的数量和计算能力,将整个训练数据分割成大小不等的CPU数据块以及GPU数据块。

整个作业任务调度框架的运行过程按照以下步骤完成(以下的步骤序号与图2中序号对应)。

①ML框架执行梯度值的聚集运算。如果是第一次迭代,随机给出梯度值;如果不是第一次迭代,当所有的执行器全部将局部梯度值发送到ML框架后,ML框架再进行聚集计算;

②根据聚集的梯度值,ML框架计算出新的权重系数,即模型的参数值;

③ML框架将新的模型参数发布到各个工作节点的CPU执行器和GPU执行器;

④ML框架根据训练数据的大小以及CPU和GPU资源数量,按照第1.4节的方式进行数据分片,将数据分为2类,CPU数据分片和GPU数据分片。每个分片包含训练数据的编号、开始位置、数据块大小和数据所在的IP地址信息等。每个数据分片按照一定数据格式构成一个任务描述,并添加到各自的排队队列中;

⑤ML框架的任务调度器从CPU队列和GPU队列中取得任务描述,并根据任务描述及调度算法,确定每个任务将要发送的执行器;

⑥调度器将任务发送到各个工作节点的CPU执行器和GPU执行器队列中;

⑦各个工作节点的CPU执行器以及GPU执行器开始按照FIFO的方式执行机器学习中的局部训练任务,计算局部梯度值;

⑧CPU执行器以及GPU执行器执行计算前,根据任务描述中的数据分片信息,从训练数据中获取采样数据;

⑨各个工作节点的执行器完成任务的运行后,将所有计算出的局部梯度值发送到ML框架;

ML框架收到全部的梯度值后,检查误差是否

达到训练模型的精度要求或者是否满足迭代次数的要求。如果满足要求,停止训练;否则,继续执行①,进行新一轮的迭代计算。这样不断地进行迭代训练过程,最终得到模型参数;

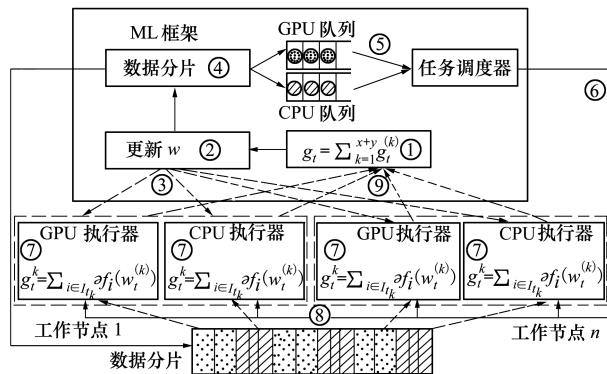


图2 面向CPU-GPU集群的机器学习调度框架

## 2 调度算法的实现

机器学习是一个不断迭代的过程,每一次迭代都需要计算梯度误差的变化,然后收集这些变化,为下一次的迭代做准备。分布式机器学习中,采用数据并行的计算模型,各个计算节点上只计算整个数据的一个分片,然后将计算结果发送到PS服务器进行聚合。如果数据分片大小不合适,可能会出现某个工作节点严重滞后的情况,导致下一次迭代的延迟,因此数据分片是本调度算法的核心。对于CPU-GPU混合的集群系统,根据CPU执行时间和GPU执行时间来决定分片的数据大小,使一个分片对应一个计算任务,从而产生大量的任务。这些任务在执行过程中,会根据各自的分片信息,读取相应的数据,按照各自的类别在相应的计算资源上进行计算。其计算过程如算法1所示。

ML\_FrameWork是ML框架的主函数,也是PS服务器所在的节点。ML\_FrameWork首先得到CPU-GPU集群中的CPU资源数量和GPU资源数量(s1),然后进入迭代过程中(s2~s8)。(s3)获得各个数据分片任务计算的梯度值,进行聚合计算;(s4)根据CPU和GPU数量、训练数据大小以及CPU任务与GPU任务之间的加速系数,将数据划分为大小不同的数据块;(s5)对于较小的数据块,产生CPU任务描述,放入CPU执行队列;(s6)对于较大的数据块,产生GPU任务描述,放入GPU执行队列;(s7)是ML\_FrameWork等待CPU任务队列和

GPU 任务队列的执行同步。

WorkerIterate 运行在各个工作节点的执行器上,是计算局部梯度的函数。它可以使用 CPU 资源或 GPU 资源进行局部梯度计算。(s1)更新并保存来自 PS 服务器的新的模型参数;(s2)计算梯度值,其中最为关键的是获得数据的分片信息,然后读取对应的训练数据,通过分片信息中包含的计算资源信息,使用不同的计算资源计算出新的局部梯度值;(s3)将计算出的局部梯度值按照一定的标准格式组织起来并返回给 ML\_FrameWork。

**算法 1** 梯度优化的分布式机器学习框架算法

Begin ML\_FrameWork:

(s1)  $x = \text{cpus}, y = \text{gpus}$

(s2) for iteration  $t = 0, 1, \dots, T$  do

(s3)  $g_t = \sum_{k=1}^{x+y} g_t^{(k)}$

(s4) partition  $I_t = \cup_{k=1}^x I_{t_k} \cup_{k=x+1}^{x+y} I_{t_k}$

(s5)  $\text{cpu\_queue.insert}(\cup_{k=1}^x I_{t_k})$

(s6)  $\text{gpu\_queue.insert}(\cup_{k=x+1}^{x+y} I_{t_k})$

(s7) sync

(s8) end for

End ML\_FrameWork

Begin WorkerIterate(t):

(s1) pull  $w_i^k$  from ML\_Main

(s2) compute  $g_i^k = \sum_{i \in I_{t_k}} \partial f_i(w_i^k)$

(s3) push  $g_i^k$  to ML\_FrameWork

End WorkerIterate

Schedule 函数描述从队列中取出任务并提交到工作节点的过程,如算法 2 所示。任务调度器从队列取出一个任务描述,根据任务的类型,如果是 GPU 任务,执行(s1),根据任务描述中的 IP 地址,将任务的分片 ID、数据分片的起始位置和数据长度发送到对应工作节点的 GPU 执行器(s2),请求 GPU 执行器执行;反之,执行(s3),发送任务到 CPU 执行器(s4)。

**算法 2** 任务调度算法

Begin Schedule

(s1) while(Task =  $\text{gpu\_queue.deque}()$ )

(s2) launchTask to GPU

(s3) while(Task =  $\text{cpu\_queue.deque}()$ )

(s4) launchTask to CPU

End Schedule

### 3 实验评价

系统在一个集群上进行实验,集群中包含 6 台 NF5468M5 服务器,作为计算节点;1 台中科曙光服务器 620/420,作为参数服务器节点。每个服务器节点包含 2 颗 Xeon2.1 处理器,每个处理器包含 8 个核(相当于一个服务器节点具有 16 个 CPU 核),32 GB DDR4 内存,2 块 RTX2080TI GPU 卡,10 GB 显存。集群包含 1 台 AS2150G2 磁盘阵列。服务器操作系统为 Ubuntu 7.5.0, CUDA 版本为 10.1.105,采用 C++11 作为编程语言,软件编写采用 NVIDIA CUDA 工具和 C++。

#### 3.1 实验任务

考虑使用一种通用的数据分析算法 K-Means 进行实验验证和分析。K-Means 算法是一个迭代求解的聚类算法,是实验分析中常用算法,且具有代表性。实现了一个 CPU 版本和 GPU 版本的 K-Means 程序,使用 C++ 语言来编写程序的 CPU 版本, NVIDIA CUDA 来编写程序的 GPU 版本。调度框架和执行器使用 C++ 语言编写。调度框架和执行器通过实现上述算法,完成对 K-Means 分布式机器学习作业的执行。

#### 3.2 算法执行效率评价分析

使用 3 种资源分配策略来比较 K-Means 算法的梯度计算的过程,图 3 为使用 3 种策略的作业执行时间比较。

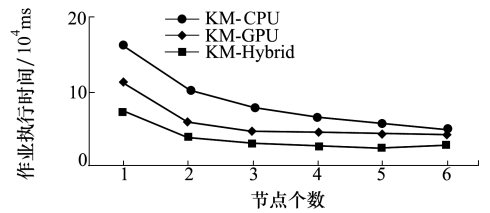


图 3 使用不同调度算法的 K-Means 作业执行时间

1) 每个计算节点采用 16 个 CPU 核来计算数据的梯度,在任务调度过程中,数据均匀分片,1 个任务绑定 1 个 CPU 核,即当 1 个计算节点有 1 个 CPU 核处于空闲状态时,分配 1 个数据分片对应的任务到该空闲 CPU 核。称该算法为 CPU 算法(KM-CPU)。

2) 每个计算节点使用 1 个 CPU 核以及 1 个 GPU 设备, GPU 设备绑定该 CPU 核。这时称被绑定的 CPU 核处于非自由状态,反之为自由状态。在

任务调度过程中,数据均匀分片,1个任务绑定1个GPU设备,即当该节点GPU设备空闲时,分配1个数据分片对应的任务到该空闲GPU设备,称该算法为GPU算法(KM-GPU)。

3) 每个计算节点使用16个CPU核及1个GPU设备,将每个节点中的GPU设备绑定1个CPU核。这时,1个计算节点上拥有1个非自由状态的CPU核和15个自由状态的CPU核。ML框架按照本文提出的算法进行数据的不均匀分片,1个分片对应的任务绑定1个自由状态的CPU核或者1个GPU设备,当自由状态的CPU核空闲时,提交1个具有较小的数据分片的任务,当GPU设备空闲时,提交1个具有较大的数据分片的任务。该算法称为混合资源调度算法(KM-Hybrid)。

图3包含3组数据,分别代表使用KM-CPU算法、KM-GPU算法和KM-Hybrid算法对应的作业执行时间。每个节点测试3次,取平均值进行统计。其中,横坐标代表集群节点数目,纵坐标代表使用各算法的作业执行时间。从实验中可以得到2个结论:①使用KM-GPU和KM-Hybrid算法的作业执行时间明显少于KM-CPU算法,这是因为GPU设备的运算能力远高于CPU设备,说明使用GPU计算资源可以明显降低K-Means作业的执行时间;②随着节点增加,计算设备增加,3种算法用时逐渐减少,性能提升逐渐变慢,这是由于每一个分片包含的数据量变少,且当节点数目过多时,节点通信开销成为影响算法性能提升的重要因素。因此,使用本文提出的算法执行分布式机器学习作业,能够加快作业的执行,且能够有效提高CPU-GPU集群中CPU计算资源的利用率,不再只是将CPU计算资源闲置。

考虑到上述实验并未说明KM-Hybrid相较于KM-GPU算法的具体优势,下面探究随着节点增加,KM-GPU和KM-Hybrid算法加速比。加速比是衡量算法性能常用指标,表示为 $s(n,p) = T_{\text{serial}}(n)/T_{\text{parallel}}(n,p)$ ,其中 $n$ 表示数据处理规模, $T_{\text{serial}}(n)$ 表示作业串行计算用时, $T_{\text{parallel}}(n,p)$ 表示作业并行计算用时。图4为KM-GPU和KM-Hybrid加速比对比情况。其中横坐标代表节点个数,纵坐标代表加速比。比值越大,说明并行计算加速效果越好。在图4中可以得出结论:随着节点数目增加,2种算法均表现出较好的加速;但随着节点数目增加,KM-GPU加速比增长逐渐缓慢,KM-Hybrid加速比增长缓慢并出现回落现象,分析后得出原因:随着

节点增加,通信延迟和调度处理开销增加,2种算法的加速比增长缓慢,且KM-Hybrid算法对通信和调度延迟更敏感。

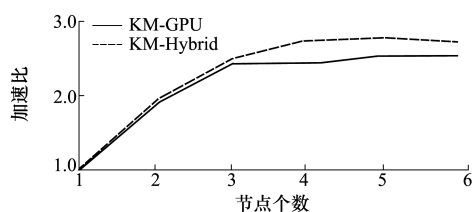


图4 KM-GPU和KM-Hybrid加速比

为了进一步探究本文中提出的不均匀数据分片策略为分布式机器学习作业带来的性能提升情况,现对KM-GPU和KM-Hybrid算法单次梯度计算时间进行了统计,其结果如图5所示。在图5中,横坐标为集群节点数目,纵坐标为单次梯度计算平均值。随着计算节点增加,2种算法单次梯度计算任务的平均时间均呈现非线性减少,主要原因是:①数据分片规模对梯度计算用时有很大的影响。随着节点增加,并行任务数目增加,单个分片数据量变少,计算设备计算时间减少;②节点的增加会造成分布式机器学习中I/O通信开销增加,特别是通过网络传输的代价会增加,导致单次梯度计算用时减少的速度越来越缓慢。基于以上两点,KM-GPU算法和KM-Hybrid算法单次梯度计算的平均时间会呈非线性减少。

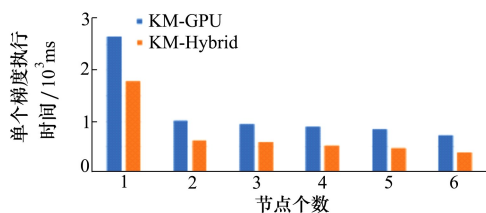


图5 梯度计算任务平均时间

上述实验过程主要探究了集群内CPU计算设备和GPU计算设备之间对作业调度的影响,但是还没有探究集群中不同数量GPU对作业调度的影响。为了更加明确GPU数量对分布式机器学习作业的影响,针对不同数量的GPU进行了K-Means算法附加实验,实验共测试4种不同情况:

1) 每个节点使用1个GPU设备和1个CPU核,GPU设备绑定该CPU核,该算法记为KM-1-GPU,相当于上述KM-GPU算法。

2) 每个节点使用 2 个 GPU 设备和 2 个 CPU 核,1 个 GPU 设备绑定 1 个 CPU 核。该算法记为 KM-2-GPU,其任务分配方式与步骤 1) 中相同,但计算资源量是步骤 1) 的 2 倍。

3) 每个节点使用 1 个 GPU 设备和 16 个 CPU 核,16 个 CPU 核中有 1 个非自由状态核和 15 个自由状态核,数据按照自由状态的 CPU 核以及 GPU 处理能力来分片,将较大的数据分片分配给 GPU,较小的数据分片分给 CPU。该算法记为 KM-1-Hybrid,相当于上述 KM-Hybrid 算法。

4) 每个节点使用 2 个 GPU 设备和 16 个 CPU 核,16 个 CPU 核中包含 2 个非自由状态核和 14 个自由状态核。任务分配方式同步步骤 3)。该算法记为 KM-2-Hybrid。

图 6 为不同 GPU 数量下 K-Means 算法作业执行时间的测试结果。可以得到如下结论:①由于节点数目增加,CPU 和 GPU 计算资源增多,4 种算法性能均有提升。②KM-2-GPU 比 KM-1-GPU 算法作业执行时间少,KM-2-Hybrid 比 KM-1-Hybrid 算法作业执行时间少,即 GPU 设备越多,作业执行越快,算法性能越好。③KM-1-Hybrid 比 KM-2-GPU 算法作业执行时间少,说明集群内单纯增加 GPU 计算资源带来的作业性能提升不如将集群中已有的闲置 CPU 利用起来对作业的性能提升大,因此可以考虑使用不均匀数据分片策略,将集群中空闲或部分空闲的 CPU 利用起来。

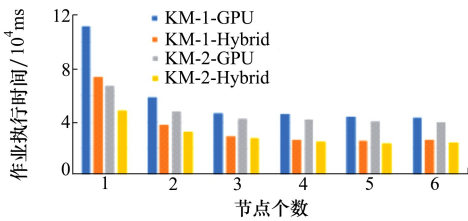


图 6 不同 GPU 数量时的作业执行时间对比

为了进一步验证论文提出的不均匀数据分片策略对分布式学习带来的性能提升,使用 Rodinia<sup>[19]</sup> 基准测试套件验证实验效果,该套件包含 huffman、K\_Means 等单节点基准程序。图 7 为 Rodinia K\_Means 基准测试用时和 KM-Hybrid 用时对比情况,横坐标表示实验次数,纵坐标表示作业运行时间。如图所示,R-16-CPU 表示在单节点上使用 16 个线程进行 K\_Means 计算,R-1-GPU 表示在单节点上使用 1 块 GPU 进行 K\_Means 计算,KM-Hybrid 表示在

单节点上使用 1 块 GPU,15 块 CPU 进行 K\_Means 计算。数据规模为 819 200×100,3 组数据平均值分别为:33.87,29.83,23.8 s,方差分别为 1.079,0.224,3.443。可以得到如下结论:①使用 GPU 进行加速的效果比使用多线程加速的效果好;②KM-Hybrid 算法比上述 2 种算法效果更好,这是由于在节点上既使用了 GPU 进行计算加速,又利用了闲置的 CPU;③KM-Hybrid 算法的稳定性不如 Rodinia 基准测试,这和数据规模及程序运行时 CPU、GPU 设备的性能相关,可以对其进一步探究。

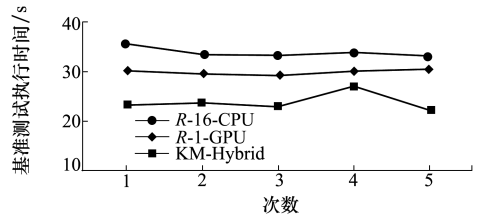


图 7 基准测试用时对比

### 3.3 算法执行质量评价分析

上述实验可以得出,本文提出的算法可以实现对作业的加速并提高异构集群资源利用率,但是其执行效果受到 CPU 和 GPU 实时性能的影响,稳定性较差。

为了进一步分析算法对作业质量的影响,使用轮廓系数<sup>[20]</sup>作为衡量 KM-Hybrid 作业质量的标准。数据规模为 8 192×100,每项测试进行 10 次,取平均值作为作业质量评估值。轮廓系数越接近 1,算法执行质量越好。经计算,R-16-CPU 算法轮廓系数均值为 0.567 996,R-1-GPU 算法轮廓系数均值为 0.540 185,KM-Hybrid 算法轮廓系数均值为 0.538 466,3 种算法轮廓系数相近,由此可以得出结论:本文基于不均匀数据分片策略的机器学习方法对作业质量没有显著影响。

## 4 结 论

通过 CPU-GPU 混合异构集群进行分布式机器学习训练已经成为一个趋势,但是在这个过程中经常存在集群资源得不到充分利用的现象。为此,论文提出了一种不均匀分片的资源调度算法,并证实该算法能够加快分布式机器学习训练,提高集群系统资源利用率。但是,考虑到算法受制于 CPU 和 GPU 资源的任务分配方式,如何更加精确合理地

CPU 与 GPU 任务进行资源分配,是今后的研究方向。

## 参考文献:

- [1] CHEN T, LI M, LI Y, et al. Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems[J/OL]. (2015-12-03)[2015-12-07]. <https://arxiv.org/abs/1512.01274>
- [2] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: convolutional architecture for fast feature embedding[C]//Proceedings of the 22nd ACM International Conference on Multimedia, 2014: 675-678
- [3] XING E P, HO Q, DAI W, et al. Petuum: a new platform for distributed machine learning on big data[J]. IEEE Trans on Big Data, 2015, 1: 49-67
- [4] CHEN L, HUO X, AGRAWAL G. Accelerating mapreduce on a coupled CPU-GPU architecture[C]//Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012: 1-11
- [5] RAVI V T, BECCHI M, JIANG W, et al. Scheduling concurrent applications on a cluster of CPU-GPU nodes[C]//2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012: 140-147
- [6] GU J, LIU H, ZHOU Y, et al. Deepprof: performance analysis for deep learning applications via mining GPU execution patterns [J/OL]. (2017-07-12)[2017-07-13]. <https://arxiv.org/abs/1707.03750>
- [7] RHU M, GIMELSHEIN N, CLEMONS J, et al. vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design[C]//2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, 2016: 1-13
- [8] GOYAL P, DOLLÁR P, GIRSHICK R, et al. Accurate, large minibatch SGD: training imagenet in 1 hour[J/OL]. (2018-04-30)[2018-05-02]. <https://arxiv.org/abs/1706.02677>
- [9] VAVILAPALLI V K, MURTHY A C, DOUGLAS C, et al. Apache hadoop yarn: yet another resource negotiator[C]//Proceedings of the 4th Annual Symposium on Cloud Computing, 2013: 1-16
- [10] ZHANG H, STAFMAN L, OR A, et al. Slaq: quality-driven scheduling for distributed machine learning[C]//Proceedings of the 2017 Symposium on Cloud Computing, 2017: 390-404
- [11] 汤小春,符莹,樊雪枫. 数据中心上异构资源的细粒度分配算法研究[J]. 西北工业大学学报,2020,38: 589-595  
TANG Xiaochun, FU Ying, FAN Xuefeng. Research on fine-grained allocation algorithm of heterogeneous resources in data center[J]. Journal of Northwestern Polytechnical University, 2020, 38: 589-595 (in Chinese)
- [12] 王彦华,乔建忠,林树宽,等. 基于 SVM 的 CPU-GPU 异构系统任务分配模型[J]. 东北大学学报,2016,37: 1089-1094  
WANG Yanhua, QIAO Jianzhong, LIN Shukuan, et al. SVM-based task allocation model of CPU-GPU heterogeneous system [J]. Journal of Northeastern University, 2016, 37: 1089-1094 (in Chinese)
- [13] XIAO W, BHARDWAJ R, RAMJEE R, et al. Gandiva: introspective cluster scheduling for deep learning[C]//13th Symposium on Operating Systems Design and Implementation, 2018: 595-610
- [14] GU J, CHOWDHURY M, SHIN K G, et al. Tiresias: a {GPU} cluster manager for distributed deep learning[C]//16th Symposium on Networked Systems Design and Implementation, 2019: 485-500
- [15] PENG Y, BAO Y, CHEN Y, et al. Optimus: an efficient dynamic resource scheduler for deep learning clusters[C]//Proceedings of the Thirteenth EuroSys Conference, 2018: 1-14
- [16] JEON M, VENKATARAMAN S, QIAN J, et al. Multi-tenant GPU clusters for deep learning workloads: analysis and implications [J/OL]. (2018-05-13)[2018-06-16]. <https://www.microsoft.com/en-us/research/publication/multi-tenant-gpu-clusters-deep-learning-workloads-analysis-implications-tr>
- [17] SHIRAHATA K, SATO H, MATSUOKA S. Hybrid map task scheduling for GPU-based heterogeneous clusters[C]//2010 IEEE Second International Conference on Cloud Computing Technology and Science, 2010: 733-740
- [18] ZHOU H, LIU C. Task mapping in heterogeneous embedded systems for fast completion time[C]//2014 International Conference on Embedded Software, 2014: 1-10
- [19] CHE S, BOYER M, MENG J, et al. Rodinia: a benchmark suite for heterogeneous computing[C]//2009 IEEE International Symposium on Workload Characterization, 2009: 44-54
- [20] ROUSSEEUW P J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis[J]. Journal of Computational and Applied Mathematics, 1987, 20: 53-65

# A unified schedule policy of distributed machine learning framework for CPU-GPU cluster

ZHU Ziyu, TANG Xiaochun, ZHAO Quan

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

**Abstract:** With the widespread using of GPU hardware facilities, more and more distributed machine learning applications have begun to use CPU-GPU hybrid cluster resources to improve the efficiency of algorithms. However, the existing distributed machine learning scheduling framework either only considers task scheduling on CPU resources or only considers task scheduling on GPU resources. Even considering the difference between CPU and GPU resources, it is difficult to improve the resource usage of the entire system. In other words, the key challenge in using CPU-GPU clusters for distributed machine learning jobs is how to efficiently schedule tasks in the job. In the full paper, we propose a CPU-GPU hybrid cluster schedule framework in detail. First, according to the different characteristics of the computing power of the CPU and the computing power of the GPU, the data is divided into data fragments of different sizes to adapt to CPU and GPU computing resources. Second, the paper introduces the task scheduling method under the CPU-GPU hybrid. Finally, the proposed method is verified at the end of the paper. After our verification for K-Means, using the CPU-GPU hybrid computing framework can increase the performance of K-Means by about 1.5 times. As the number of GPUs increases, the performance of K-Means can be significantly improved.

**Keywords:** CPU-GPU tasks; unified scheduler; clustering algorithm; distribution

**引用格式:** 朱紫钰, 汤小春, 赵全. 面向 CPU-GPU 集群的分布式机器学习资源调度框架研究[J]. 西北工业大学学报, 2021, 39(3): 529-538

ZHU Ziyu, TANG Xiaochun, ZHAO Quan. A unified schedule policy of distributed machine learning framework for CPU-GPU cluster[J]. *Journal of Northwestern Polytechnical University*, 2021, 39(3): 529-538 (in Chinese)