

机器学习

第10章 分布式机器学习

章节介绍

- 机器学习方法是计算机利用已有的数据生成某种模型，并利用此模型预测的一种方法。在确定模型结构之后，根据已知模型寻找模型参数的过程就是训练，训练过程中不断依据训练数据来迭代调整模型的参数值，从而使模型的预测结果更为准确。在现实应用中，要达到好的效果，训练数据集可能很大，模型参数量剧增，会带来很多性能和算法设计问题，单台机器难以胜任，需要分布式的机器学习架构。本章主要介绍分布式机器学习基础知识，并介绍主流的分布式机器学习框架，结合实例介绍一些机器学习算法。

章节结构

- 分布式机器学习基础
 - 参数服务器
 - 分布式并行计算类型
- 分布式机器学习框架
 - MapReduce 编程模型
 - Hadoop MapReduce 框架
 - Spark
 - PMLS
 - TensorFlow
- 并行决策树
- 并行k-均值算法

分布式机器学习基础

- 分布式机器学习的一些核心问题：
 - 如何提高各分布式任务节点之间的网络传输效率；
 - 如何解决参数同步问题，传统训练模型是采用同步方法，如果机器性能不统一，必然会产生训练任务之间的协作；
 - 分布式环境下如何提高容错能力，需要避免单点故障，并能合理处理异常，训练子节点出错不影响全局任务。

参数服务器

- 应用传统的大数据处理框架训练大型的机器学习模型时，由于数据量比较大并且训练方法多样，存在着一致性、扩展性、稳定性的问题。较大的模型也意味着参数较多，因而需要实现分布式并行训练，参数服务器是分布式并行训练框架之一，存放着模型的参数和状态。参数服务器具有如下特点：
 - 高效通信
 - 宽松一致性
 - 灵活可扩展
 - 容错能力强
 - 易用

高效通信

- 参数服务器采用异步通信的方式，模型训练和参数存储各自独立，使各迭代之间并行训练，大大减少了延时。

宽松一致性

- 传统的梯度下降由于梯度下降方向是在一次完整迭代（可能是**mini-batch**）之后才确定，而并行训练从多个子训练集中随机选择梯度方向，对一致性要求较宽松。

灵活可扩展

- 训练过程中支持动态扩展节点，不需要重启训练任务就可以动态插入新节点到集合中，这一特性无疑有利于那些训练周期较长（长达数天或数周）的机器学习项目，可节省大量训练时间。

容错能力强

- 在大型服务器集群中，由于节点较多，小概率故障往日常态化，所以需要节点的恢复（状态清理、任务重启）时间要短，而且不能中断训练过程，这就要求并行化系统具有较强的容错能力。

易用

- 目前机器学习项目开发数量较少，为了减少学习难度，需要尽可能的使用常用语言或将参数表示成通用的形式，如向量、矩阵等，并与现有机器学习框架无缝拼接。

分布式并行计算框架

- 分布式并行计算的类型一般分为三种：
 - 模型并行
 - 数据并行
 - 混合同行

模型并行

- 模型并行是指将模型按照其结构放在不同的分布式机器上进行训练，一般用在那些内存要求较高的机器学习项目，例如，单机训练一个**1000**层的**DNN**网络，内存容易溢出，而使用模型并行，用不同的机器负责不同的层进行训练，通过维护各层间参数同步实现整个**DNN**网络的并行训练。

数据并行

- 数据并行是指各机器上的模型相同，对训练数据进行分割，并分配到各机器上，最后将计算结果按照某种方式合并。该方法主要应用在海量训练数据的情况，数据以并行化方式训练，训练过程中组合各工作节点的结果，实现模型参数的更新。参数并行常用的方法有参数平均和异步梯度下降的方法。

参数平均

- 参数平均是在每次训练迭代完成后计算各节点各模型参数平均值，这一方法操作简单，主要依赖网络同步更新，如果更新频率较慢会导致参数差别较大，平均之后的模型参数的局部差异化被抵消，效果较差，影响模型的精确性，反之，如果更新较快，对网络压力较大，通信和同步的成本较高，所以在应用中需要结合模型复杂度和优化方法进行平衡。

异步梯度下降

- 异步梯度下降是一种基于更新的数据并行化，它传递的是模型训练过程中的梯度、动量等信息，而没有直接传递参数值，这样一方面可以减少传输数据量，提高网络传输效率，另一方面不同计算节点通过共享梯度，可以提高模型收敛速度。该方法的不足之处在于会随着引入参数数量的增多出现梯度值过时的问题。

混合并行

- 混合并行的方式是指综合应用模型并行和数据并行，在训练集群的设计中，将上述两种方式进行合并，各取所长，形成互补。例如，可以在同一台机器上采用模型并行化，在GPU和CPU之间使用模型并行，然后在机器之间采用数据并行化，将数据分配在不同的机器上，既实现了计算资源利用的最大化，也减少了数据分发的压力。

分布式机器学习框架

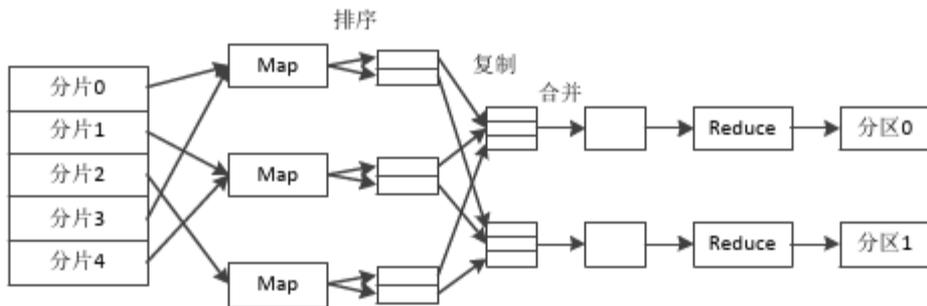
- 分布式机器学习是机器学习领域的一大主要研究方向，其中**MapReduce**适合做离线计算，**Storm**适合做流式计算，**Spark**是内存计算框架，能快速得到计算结果。分布式机器学习平台归类为三种基本设计方法：基本数据流、参数服务器模型以及高级数据流。基于这三种方法来介绍分布式机器学习框架。

MapReduce编程模型

- **MapReduce**是一个能处理和生成超大数据集的计算模型，该架构能够在大量硬件配置不高的计算机上实现并行化处理，这一编程模型结合用户自定义的**Map**和**Reduce**函数。**Map**函数处理一个输入的基于<Key,value>对的集合，输出中间基于<Key,value>对的集合，**Reduce**函数是将所有具有相同key值的value值进行合并，将数据集合进行压缩。

MapReduce编程模型

- 一个典型的MapReduce程序的执行流程如下图所示。



Hadoop MapReduce 框架

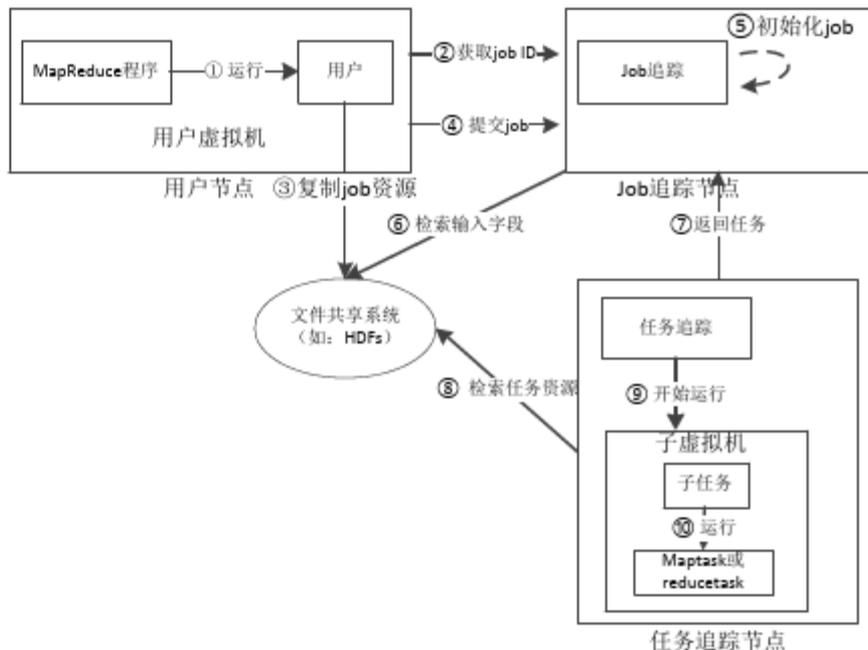
- Hadoop MapReduce是Hadoop三大组件之一，包括JobTracker和一定数量的TaskTracker。JobTracker负责任务分配和调度，一个MapReduce作业通常会把输入的数据集切分为若干独立的数据块，由Map任务以并行方式处理它们，框架会对Map的输出先进行排序，然后把结果输入到Reduce任务中。通常作业和输出都会被存储在文件系统HDFS中，由JobTracker负责任务的调度和监控，以及重新执行已经失败的任务。

Hadoop MapReduce 框架

- Hadoop MapReduce 框架由一个单独的主 JobTracker 和每个集群节点对应一个备 TaskTracker 组成。JobTracker 负责调度作业的所有任务，并监控它们的执行，这些任务分布在不同的备 TaskTracker 上。如果 TaskTracker 上的任务执行失败，还会调度其重新执行。而 TaskTracker 仅负责执行指派的任务。

Hadoop MapReduce框架

- 作业的提交和执行的过程如右图所示。



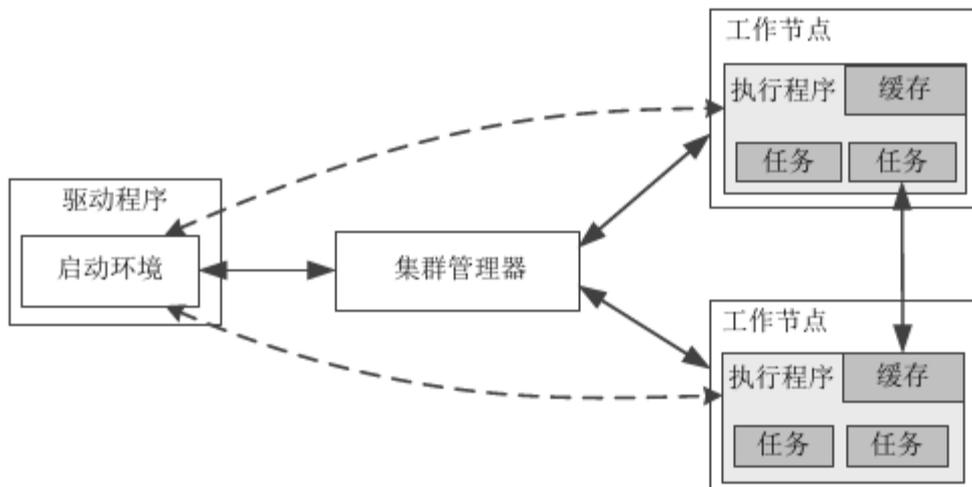
Hadoop MapReduce 框架

- Hadoop MapReduce 的作业提交执行过程如下：
 - MapReduce 程序通过 `runJob()` 方法新建一个 `JobClient` 实例；
 - 通过 `JobTracker` 的 `getNewJobId()` 获取新 job ID；
 - 检查作业输入输出的合法性，如果验证失败，作业将不会被提交，错误返回给 `MapReduce`。
 - 通过调用 `JobTracker` 的 `submitJob()` 方法提交作业；
 - 将提交的作业放到队列中，交由作业调度器进行调度，并对其进行初始化。
 - 查询数据分片，并创建 `Map`、`Reduce` 任务：数据分片个数 `Map` 个数，`Reduce` 任务的数量由 `JobConf` 的 `mapred.reduce.tasks` 属性决定；
 - `TaskTracker` 定期发送心跳（`heartbeat`）给 `JobTracker` 报告状态；
 - `TaskTracker` 从文件系统提取任务需要的资源文件；
 - 开始执行 `TaskTracker` 相关任务
 - 运行 `TaskTracker` 中的 `Map`、`Reduce` 方法并输出结果。

- 与Hadoop MapReduce相比，Spark的优势在于处理迭代计算的机器学习任务，尤其是内存要求小的应用，性能提升很大，Spark还可以进行批处理、实时数据处理、机器学习以及图算法等计算模块。使用Spark平台无需关心分布式并行计算的细节，可以智能地进行数据切分、算法复制、分布执行、结果合并，以支持数据分析人员快速开发分布式应用。

Spark

- Spark的基本框架如下图所示。



Spark

- **Spark**应用核心由启动环境和执行程序两部分组成，其中执行程序负责执行任务，运行执行程序的机器是工作节点，而启动环境由用户程序启动，通过集群管理器与各个执行程序进行通信。集群管理器主要负责集群的资源管理和调度，目前支持**Standalone**、**Apache Mesos**和**YARN**三种类型的管理器。
 -

- Spark使用弹性分布式数据集（RDD）抽象分布式计算，RDD是Spark并行数据处理的基础，它是一种只读的分区记录的集合，用户可以通过RDD对数据显示地控制存储位置和选择数据的分区。RDD主要通过转换和动作操作来进行分布式计算，转换是根据现有数据集创建新数据集，动作是在数据集上进行计算后返回值给Driver程序。使用RDD可以用基本一致的方式应对不同的大数据处理场景，还能够提高分布式计算的容错性。

Spark

- **Spark**是一种粗粒度、基于数据集的并行计算框架。其计算范式是数据集上的计算，在使用**Spark**的时候，要按照这一范式编写算法。所谓的数据集操作，就是成堆的数据，如果源数据集是按行存储的话，就需要对其进行适配，将若干记录组成一个集合。因此在提交给**Spark**任务时，需要先构建数据集，然后通过数据集的操作，实现目标任务。

- **PMLS**是面向机器学习任务设计的，核心解决非均匀收敛、并行训练中容错能力和动态结构依赖等问题。非均匀收敛采用模型并行的方式减少各工作器对模型参数的依赖。**PMLS**的架构中主要包括参数服务器、调度器和工作节点组成，通过引入参数服务器实现结构的动态扩展和快速迭代训练，参数服务器用于分布式内存键值参数的存储和更新。调度器通过向参数交换通道发送参数子集实现模型的并行化调度。工作节点从参数服务器上获取的参数子集对分区数据集进行处理，每个工作节点上都有参数更新客户端和调度客户端，前者通过参数交换通道收发参数值，后者通过调度控制通道收发调度命令。

- **PMLS**为了提高参数传输效率，采用过时同步并行策略提高容错能力。在动态结构依赖方面，**PMLS**在运行时分析参数间的依赖关系，使用调度器实现参数子集的并行独立更新。在非均匀收敛方面，**PMLS**对不同参数的更新优先级进行量化，减少优先级较低的参数的更新频率，避免资源浪费，从而提高了每次迭代的收敛性能。

TensorFlow

- TensorFlow为用户封装了底层的分布式操作，使其可以专注于编写机器学习代码。使用数据流图进行数值计算，用有向图中的节点表示，节点的状态是可变的，边是张量，对应为多维数组。TensorFlow中数据并行化的方式由In-graph、Between-graph、异步训练、同步训练几种方式，通过将模型训练分配给不同的工作节点，并使用参数服务器共享参数。

In-graph模式

- 这一模式与单机多GPU模式相比，是由一个数据节点和多机多计算节点组成，通过数据节点实现训练数据分发，一般采用gRPC协议，即Google通过对RPC协议进行封装，简化了网络调用方式。计算节点通过使用join操作公开一个网络接口，等待接收计算任务，客户端在使用时将任务指定到某一个计算节点上进行计算，调用方法与单机模式下指定GPU是一样的。

Between-graph模式

- 这一模式的训练参数保存在参数服务器，数据分片包存在各个计算节点上，各计算节点算完之后将最新的参数提交给参数服务器，由其进行更新保存。这一模式的优点在于无需训练数据的分发，特别是处理TB级数据量时，节省大量数据传输时间，所以在大数据深度学习应用中推荐使用此模式。

异步和同步训练

- 在In-graph以及Between-graph模式下，参数更新都支持同步和异步两种方式。其中，同步更新是指每次梯度更新都要等所有节点计算完成后返回结果，结果处理完成后统一更新参数。这种方式训练过程稳定，不会抖动，但模型处理速度存在木桶效应，计算较慢的节点会对整体训练过程有拖累。异步更新的方式不存在这种问题，计算性能强，所以模型训练速度快，但是各计算节点的更新不同步导致梯度下降不稳定，抖动较大。

并行决策树

- 随着大数据时代的到来，算法需要处理的数据量急剧增加，仅依靠原始的决策树算法进行分类无论在效率上还是准确性上都不足以满足需求。高效出色的在大数据量下使用决策树算法，需要将决策树算法并行化。

并行决策树

- 并行决策树算法基于**MapReduce**框架，核心思想是分而治之的策略。**MapReduce**通过将海量数据集分割成多个小数据集交给多台不同计算机进行处理，实现并行化数据处理。应用到决策树算法上，通过**MapReduce**将决策树算法并行处理，将耗时的属性相似度计算的步骤并行执行。**Map**阶段，以单元组形式分解数据，计算属性相似度，以<属性名，相似度>形式输出。**Reduce**阶段，汇总所有局部结果，找到最大相似度属性名，以这个属性作为测试节点，若是叶子节点，则返回，否则执行分裂，将其录入待计算数据库中进行存储。不断重复上述过程完成决策树的构建。

并行k-均值算法

- k-均值算法是应用最广泛的聚类算法之一，随着大数据的发展，在实际使用过程中如何提升该算法的性能成为了一个有挑战性的任务。可以基于**Map Reduce**实现k-均值算法，在**Hadoop**环境中并行运行，能够高效且廉价的处理大型数据集。

并行k-均值算法

- 在具体实现该算法时，将输入数据集存储在分布式文件系统HDFS中，作为<key,value>的序列文件，每个键值对代表数据集的一条记录，其中key记录的是数据文件距离起始位置的偏移量，value是该条记录的内容。将迭代后或初始化后的k个聚类中心放到Configuration中，然后在Mapper的setUp计算读取这k个聚类中心。Mapper会将同一类的数据发送至同一个Reducer。在Reducer中，只需要根据数据重新计算聚类中心即可。

并行k-均值算法

- 使用MapReduce框架实现k-均值聚类算法时，需要将每一次迭代作为一个MapReduce Job进行计算，通过多次运行该Job达到迭代的效果，最终得到k个聚类中心。基于MapReduce的并行k-均值算法，可以在廉价机器上有效处理大型数据集。

