STRADS-AP: Simplifying Distributed Machine Learning Programming without Introducing a New Programming Model

一种不使用新编程模型的简化分布式机器学习编程方法

# 现有的问题

现有分布式并行框架在简化分布式并行编程方面取得了重大进展。许多高级框架，例如MapReduce[1]，Spark[2]，Pregel[3]，PowerGraph[4]，GraphX[5]，PyTorch[6]和TensorFlow[7]已经被广泛投入使用。为了自动并行化计算，同时满足诸如容错和负载平衡之类的基本要求，这些框架提供了受限的编程模型和有限的数据抽象。例如，Spark提供的弹性分布式数据集（RDD）无需进行细粒度的写访问。Spark和MapReduce要求程序员使用少量运算符（例如map和reduce）来进行编程模型的设计，而GraphLab则需要采用一种很少使用的以顶点为中心的编程模型。

这些框架的编程模型不同于顺序编程模型。因此，使用框架提供的数据抽象和编程模型重写顺序机器学习代码会花费很多精力，并需要额外知识。此外，框架所提供简单机制通常会导致群集资源的使用不足。这些框架将数据放置，任务映射和通信抽象化，以牺牲对硬件资源的有限访问为代价，并且在有效实施机器学习算法方面面临挑战。研究表明，流行的机器学习算法的单线程[8]或MPI实现[9]比流行框架上的相应代码实现快两个数量级。

我们认为，围绕分布式机器学习编程的复杂性以及执行效率低下是偶然的，而不是固有的。也就是说，许多顺序的机器学习代码可以自动并行化并且可以最佳地利用集群资源。为证明我们的观点，我们提出了STRADS-AP，这是一种新颖的分布式机器学习框架，它仅仅对顺序程序代码进行微调，机械更改的API，便可以使运行时可以自动并行处理任意大小的集群上的代码，同时调整的分布式机器学习程序的性能。

# STRADS-AP自动并行化 (AP)

STRADS-AP是STRADS的发展[10]，它提供了一个框架，可根据用户指定的调度计划并行执行机器学习程序的执行。该计划通常避免数据冲突，从而提高每次迭代的统计进度。 STRADS的挑战在于，用户需要了解代码并手动提出调度计划。STRADS-AP通过自动生成无冲突数据调度计划来应对这一挑战。

STRADS-AP的API使数据科学家摆脱了将顺序机器学习代码转换为框架的编程模型的挑战。为此，STRADS-AP API提供了分布式数据结构（DDS），例如矢量和映射，允许对元素进行细粒度的读/写访问，以及两个熟悉的循环运算符。

# STRADS-AP’s 工作流程

如图1所示，STRADS-AP的工作流程始于数据科学家对顺序代码进行机械更改，如图1中a和b。然后，代码由STRADS-AP的预处理器进行预处理，并由C++编译器编译为二进制代码，如图1（c）。接下来，STRADS-AP的运行时在群集的节点上执行二进制文件，同时隐藏分布式编程的细节，如图1（d）。运行时系统负责：（1）透明地分区存储的DDS。训练数据和模型参数，（2）跨集群并行化计算的各个部分，（3）容错，以及（4）如果需要在共享数据上强制强一致性，或者以宽松的一致性同步部分输出。

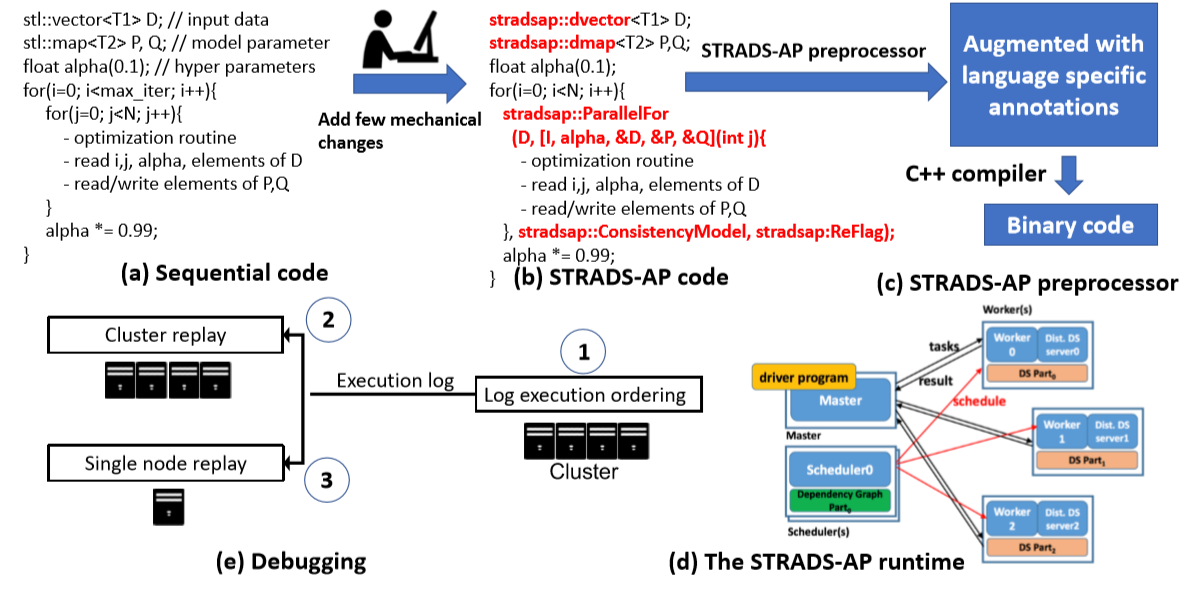
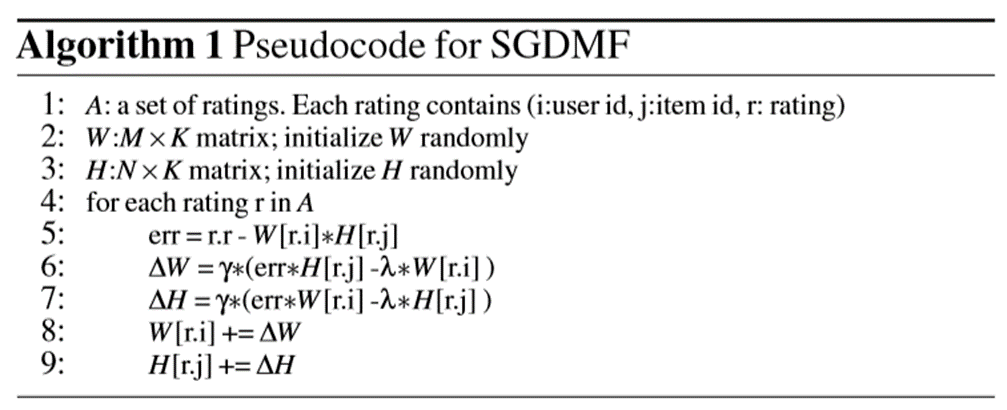


图 1 STRADS-AP工作流程

# 框架的使用代价

文章在本节选择随机梯度下降（SGD）解决矩阵因子分解（MF）的算法——一种流行的推荐系统算法。首先，我们编写使用SGDMF的顺序代码，利用STRADS-AP和Spark将顺序代码转换为三种不同共享内存的并行实现，并比较它们的编写效率。



|  |  |
| --- | --- |
|  |  |
| （a）顺序代码 | （b）OpenMP代码 |
|  |  |
| （c）STRADS-AP代码 | （d）Spark代码 |

图 2 SGDMF（算法1）实现为顺序代码，并使用OpenMP，STRADS-AP和Spark进行并行化

使用OpenMP[11]并行处理图2（a）中的顺序代码，以形成单节点并行化。通过对顺序代码进行两种修改，如图2（b）所示：用parallel-for编译注释批处理循环，以使OpenMP运行时知道要并行化的内容，并添加互斥量以避免在共享矩阵W和H上发生数据争用。内部循环使用fork-join模型进行循环索引，其中线程使用不同的循环索引运行循环主体，并在完成时进行合并。

使用STRADS-AP并行化相同的顺序代码几乎是通过以下方式机械完成的：（1）用STRADS-AP的分布式数据结构替换串行数据结构，以及（2）用STRADS-AP的AsyncFor循环操作符替换内部循环，如图2（C）所示。与OpenMP代码不同，STRADS-AP代码没有显式锁定。运行时负责在分布式设置中执行循环主体时解决矩阵W和H上的数据冲突，从而使用户不必编写容易出错的锁定代码。除了共享内存并行性外，STRADS-AP只需付出很少的努力即可实现有效的分布式并行性。

与STRADS-AP不同，将图2（a）中的顺序代码使用Spark并行化需要大量的编程工作。Spark缺少并发控制原语，且由于SGDMF的内部循环在并行化时会导致数据依赖性，因此我们需要实施计划以正确执行。关于并发控制的推理是特定于应用程序的，通常需要大量的设计工作。对于SGDMF，我们使用Gemulla [12]的Strata调度算法。

# STRADS-AP编程接口

STRADS-AP以通用的结构模式针对机器学习代码编写，该结构模式包括两个部分：（1）预训练部分，用于初始化模型和输入数据结构，并执行粗粒度转换。（2）训练部分，使用嵌套循环迭代优化目标函数，其中内部循环执行优化计算。为了实现STRADS-AP程序，用户编写一个简单的驱动程序来声明超参数，调用运算符以创建和转换DDS（图3（a）），然后调用STRADS-AP循环运算符进行优化（图3（b））。

|  |
| --- |
|  |
| （a）预训练部分 |
|  |
| （b）训练部分 |

图 3 STRADS-AP针对的ML应用程序分为两个部分：（a）预培训部分和（b）培训部分。

表1显示了STRADS-AP编程接口的子集。 DDS [T]是一个可变的容器，它在群集上划分类型为T的元素的集合。

STRADS-AP在机器学习程序训练部分中，提供循环运算符以替换的内部循环。

|  |  |  |
| --- | --- | --- |
|  | 类型 | 描述 |
| 分布式数据结构(DDSs) | dvector[T] | 具有按元素读取/写入访问权限的T型元素的分布式Vector |
| dmap[K,V] | 具有按元素读取/写入访问权限的类型为K和V的[K，V]个元素对的分布式Map |
| dmultimap[K,V] | 具有按元素读取/写入访问权限的类型为K和V的[K，V]个元素对的分布式Multimap |
| 循环操作符 | AsyncFor(int64 S, int64 E,   UDF F) | 以隔离的方式使闭包F与索引[S，E]并行化-避免数据冲突 |
| SyncFor(DDS[T] &D, int M,   UDF F, SyncOpt S,  bool RE) | 使用同步选项S以数据并行方式在大小为M的D个minibatch上并行化闭包F。RE指示是否执行侦察执行 |

## 驱动程序的执行

AsyncFor使循环索引上的循环并行化，即使循环体具有共享数据，也可以确保循环体的隔离执行。SyncFor在输入数据上并行化循环。它将输入数据分为P个块，每个块由P个节点并行处理。每个节点处理其数据块，更新模型参数的本地副本。STRADS-AP提供了Map，Reduce，Join，Load和Create运算符，用于在预训练期间加载，存储和创建DDS。

在STRADS-AP驱动程序中，语句分为三类：顺序语句，STRADS-AP数据处理语句和STRADS-AP循环语句。

运行时维护一个状态机，每个类别一个状态，以跟踪要执行的代码类型。

驱动程序节点以顺序状态启动驱动程序，并在本地执行顺序执行，直到首次调用STRADS-AP运算符。在STRADS-AP运算符调用时，状态机切换到相应的状态，运行时在多个节点上将运算符并行化。STRADS-AP运算符完成后，运行时将切换回顺序状态，并继续在本地运行驱动程序。

## 侦察程序执行

第一次调用循环运算符时，运行时将启动侦察执行（RE），这是循环运算符的虚拟执行。RE是只读执行，执行对DDS的所有读取，并发现各个循环体的读取/写入集。

运行时将读/写集用于两个目的：（1）执行依赖关系分析并生成无冲突数据的调度计划，以在AsyncFor运算符中同时执行循环主体。（2）在实际执行期间预取和缓存远程节点上的DDS元素，以实现低延迟访问。

## 分布式数据结构

传统的检查点方法是在检查点期间将整个程序状态转储到存储中，并在恢复期间从最后一个成功的检查点加载状态。由于ML程序可能具有任意数量的非DDS变量（例如超参数），因此传统方法将要求用户编写样板代码以保存和还原这些变量的状态，从而降低生产率并增加引入错误的机会。

因此，STRADS-AP采用了不同的检查点方法，从而消除了对此类样板代码的需求。节点发生故障时，STRADS-AP将以快速重新执行模式重新启动应用程序。在这种模式下，当运行时遇到执行迭代程序i的并行运算符op时，它将首先检查是否存在opi的检查点。如果是，则运行时将跳过opi的执行，并从检查点加载DDS状态。否则，它将继续正常执行。因此，可以快速正确地恢复非DDS变量的状态，而不必强迫用户编写额外的代码。

## 并发控制

STRADS-AP实现了两个并发控制引擎：（1）用于AsyncFor运算符的可序列化引擎。（2）用于SyncFor运算符的数据并行引擎。两个引擎都使用侦察执行程序中的读/写集来预取远程DDS元素，而可序列化引擎还使用它来制定无冲突数据的执行计划。

可序列化引擎实现了一个调度程序模块，该模块从RE获取读/写集，分析数据依存关系，生成依存关系图，并生成避免数据冲突的并行执行计划。 为了增加并行度，假定循环体执行的任何串行顺序是可接受的，则可序列化引擎可以更改任务的执行顺序。调度程序使用一种算法，将循环体分为N个任务组，其中N比群集中节点的数量大得多。该算法允许任务组内的依赖关系，但确保没有任务组之间的依赖关系。在运行时，调度程序将任务组放置在节点上，每个节点都保留一个任务组池。为了平衡负载，可序列化引擎运行贪心算法，该算法按大小降序对任务组进行排序，并将任务组分配给迄今为止负载最小的节点。每个节点创建用户指定数量的线程，并将任务组从任务池分派到线程。节点上的所有线程都无需锁定就可以访问每个节点的DDS缓存，这是因为每个线程都按顺序执行任务组，并且调度算法保证了整个任务组之间不会发生数据冲突。

在数据并行引擎中，任务被定义为循环主体，循环主体拥有D个大小为M的mini-batch，其中D和M为SyncFor参数。因此，单个SyncFor调用将生成具有不同mini-batch的多个任务。引擎将这些任务放在多个拥有关联的mini-batch的节点上，这些节点形成任务池。与可序列化引擎不同，线程包含每个线程缓存，并且不允许访问每个节点缓存，因为在这种情况下，不能保证无冲突的数据访问（不同任务中mini-batch的数据可能会产生冲突）。当节点将任务从任务池分派到线程时，它将参数值从每个节点缓存复制到每个线程缓存。

## 调试模式

STRADS-AP 支持两种调试模式：(1) cluster replay mode, and (2) single-node replay mode。STRADS-AP记录执行顺序，包括锁授予顺序和消息顺序，并允许用户重播日志。

为此，STRADS-AP实现了记录/重放模块。 记录模块将每个节点中的锁授予顺序记录下来，并将消息到达DDS键值存储中的顺序记录到持久性存储中。重播日志时，每个节点强制执行相同的部分顺序。

# 框架的评价

## 性能评估

在矢量大小=100，窗口=5，负样本数=10的10亿个单词数据集[13]上，Word2Vec进行10次迭代的时间如图4所示。可见STRADS-AP框架下，集群中核心越多，运行时间越短，且和MPI、OpenMP等并行框架性能差距较小。

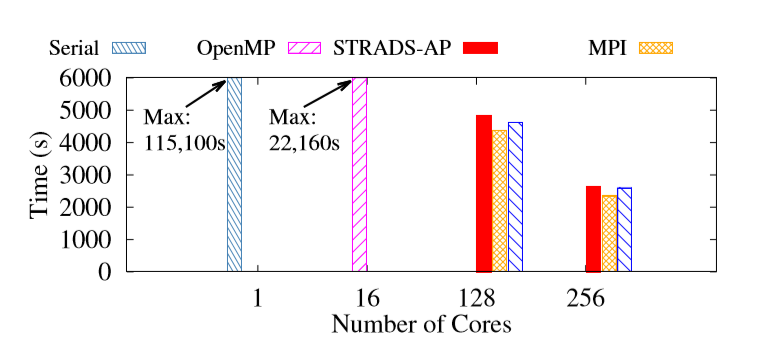


图 4 Word2Vec模型在不同框架下运行时间对比

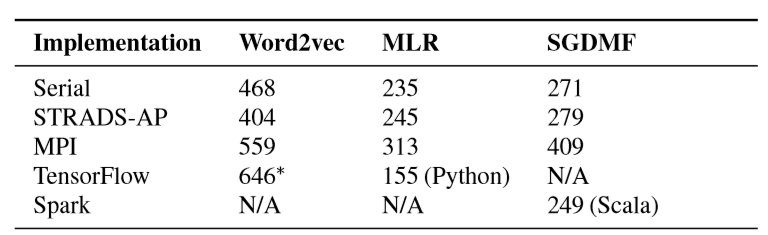


图 5 不同模型在不同框架下实现需要的行数

图5在第一栏中显示了Word2vec实现的行数。与串行实现相比，STRADS-AP实现的行数减少了15％，这主要是由于编码风格和STRADS-AP内置文本解析库的使用所致。 如果我们将比较的重点放在程序的核心（训练例程）上，那么这两种实现都有大约100行，因为STRADS-AP实现采用了串行代码并对其进行了一些简单的更改。

# 存在的问题

当任务需要并行处理大规模数据时，例如训练一个大型的神经网络，需要每个节点有足够的外存（checkpoint机制），且每个节点的I/O访问时间会很长。

由于ML程序可能具有任意数量的非DDS变量（例如超参数），传统checkpoint方法将要求用户编写样板代码以保存和还原这些变量的状态，从而降低生产率并增加引入错误的机会。是否可以将所有变量都以DDS变量的形式进行暂存和读取？

DDS种类不够丰富，不能够实现矩阵运算。

框架尚未开源，许多细节没有公开。