关于面向机器学习的动态 SSP 分布式训练框架的调研

戴凌飞 狄战元 史好迎 张林隽 钟赟

1 概述

在深度学习中，训练深度神经网络是一项对时间和存储开销要求很高的任务，需要用大量数据训练一个非常复杂的模型，而这往往是一台机器难以做到的。更常见的办法是利用分布式系统，实现参数服务器框架，将参数训练任务分配给多台机器并行完成。

在本次调研中，我们比较了批量同步并行（Bulk Synchronous Parallel，BSP）、异步并行（Asynchronous Parallel，ASP）和延迟同步并行（Stale Synchronous Parallel，SSP）三种经典的参数服务器的框架，并在 SSP 的基础上调研了动态 SSP，后者通过在运行时动态调整延迟同步的阈值，既能在分布式系统中实现较高的一致性和训练结果准确性，又能达到较好的收敛速度，同时又能根据实时运行环境动态调整对于同步的要求，从而达到全局效果最优。

2 参数服务器框架的相关工作

2.1 批量同步并行（Bulk Synchronous Parallel）

批量同步并行 BSP 模型是 Valiant 提出的“桥接模型”，它提供了并行架构和算法领域之间的标准接口。并行机器模型的三个基本参数是 p，l，g，其含义如下：

- p：即处理器/内存组件的个数
- l：连续同步操作之间的最小时间（以基本本地计算步骤衡量）
- g：以基本计算操作计算的整个系统的总吞吐量与以传递信息的单词计算的路由器的总吞吐量之比。

2.1.1 工作原理

在 BSP 模型中，并行机由一组处理器 (每个处理器都有自己的私有内存) 和一个可以在处理器之间路由一定大小数据包的互连网络组成。计算分为超步骤 (supersteps)，在每个超步骤中，处理器可以对本地数据执行操作，发送包和接收包。在一个超步骤中发送的数据包在下一个超步骤开始时被发送到目标处理器。连续的超步骤由所有处理器的全局同步来分隔。用一个简单的代价函数给出了 BSP 模型中算法的通信时间。在一个超步骤中，所有的进程并行执行局部计算。一个超步骤可分为三个阶段：

1) 本地计算阶段，每个处理器只对存储本地内存中的数据进行本地计算。

2) 全局通信阶段，对任何非本地数据进行操作。

3) 栅栏同步阶段，等待所有通信行为的结束。

考虑一个由 S 超步骤组成的 BSP 程序。然后给出超步 i 的执行时间为

$$w_i + gh_i + L$$

其中 w_i 是执行的最大工作量 (本地计算)，h_i 是超步骤期间任何处理器发送或接收的最大数据包数量。整个程序的执行时间定义为

$$W + gH + LS; \tag{1}$$

$$W = \Sigma_{i=0}^{S-1} w_i; H = \Sigma_{i=0}^{S-1} h_i \tag{2}$$

其中 w_i 和 W 为超阶和程序的工作深度。

高效的 BSP 机器编程是基于一个简单的目标。为了最小化公式 (1) 给出的执行时间，程序员必须尝试①最小化程序的工作深度，②最小化任何处理器在每个超步骤中发送或接收的最大数据包数，③最小化程序中的总超步骤数。在实践中，这些目标可能会发生冲突，必须做出权衡。通过考虑底层机器的 g 和 L 参数，可以选择正确的折衷方案。

BSP 模型主要有两个方面。一是 BSP 模型将互连网络视为一个批量路由网络，它可以有效地路由任意均衡的通信模式。该模型忽略了底层机器的特定网络拓扑结构。因此，模型只考虑两个级别的局部性: 本地 (处理器内部) 或远程 (处理器外部)。另一个观察结果是，BSP 模型需要所有处理器之间的完全合作来路由哪怕是一条消息。虽然这似乎是一种不自然的限制，但一般认为这是适当的。

2.1.2 分析

在并行计算时，Valiant 试图也为软件和硬件之间架起一座类似于冯 • 诺伊曼机的桥梁，它论证了 BSP 模型可以起到这样的作用，正是因为如此，BSP 模型也常叫做桥模型。一般而言，分布存储的 MIMD 模型的可编程性比较差，但在 BSP 模型中，如果计算和通信可以合适的平衡（例如 g=1），则它在可编程方面呈现出主要的优点。在 BSP 模型上，曾直接实现了一些重要的算法（如矩阵乘、并行前序运算、FFT 和排序等），他们均避免了自动存储管理的额外开销；另外，它可以有效的在超立方体网络和光交叉开关互连技术上实现，显示出该模型与特定的技术实现无关，只要路由器有一定的通信吞吐率。在 BSP 模型中，超级步的长度必须能够充分的适应任意的 h-relation，这一点是人们最不喜欢的，在超级步开始发送的消息，即使网络延迟时间比超级步的长度短，该消息也只能在下一个超级步才能被使用。此外，BSP 模型中的全局障碍同步假定是用特殊的硬件支持的，但很多并行机中可能没有相应的硬件。

2.2 异步并行（Asynchronous Parallel）

异步并行参数服务器 (Asynchronous Parallel Parameter Server，ASP) 在基础参数服务器架构上做出改进，使得所有 worker 在每次迭代时异步地将计算出的梯度发送到参数服务器，不需要同步。在整个训练过程中，worker 之间不需要相互等待，各自独立运行。ASP 在很多分布式机器学习和数据挖掘框架中都有所应用，例如谷歌的 DistBelief, Hogwild! ，豆瓣的 Paracel 等。在此架构中，数据集和工作负载被分发到各工作节点上，由服务器节点维护全局共享参数，每个工作节点只需计算一部分参数。参数服务器需要对训练模型的所有参数进行维护，首先接收各个工作节点输出的梯度，再对梯度进行累加求平均值，将更新后的梯度值乘以步长，更新全局参数。各工作节点执行的操作主要分为 4 步：

1) 从训练集中随机选择一份样本（Mini-batch ）

2) 向参数服务器申请一个参数子集

3) 利用选择的训练集计算目标函数的梯度

4) 将计算得到的梯度发送给参数服务器

2.2.1 工作原理——DistBelief

谷歌的 DistBelief 的参数服务器架构，是目前的主流分布式并行机器学习架构，主要用于超大规模深度学习网络的训练。如图 1 所示，其中 $\boldsymbol{w}$ 表示梯度，α 表示步长。DistBelief 将巨大的深度学习模型分布存储在全局的参数服务器中，计算节点通过参数服务器进行信息传递，很好地解决了 SGD 和 L-BFGS 算法的分布式训练问题。

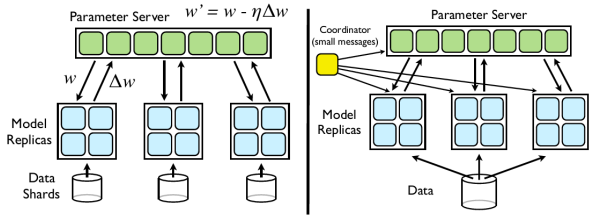

图 1: Asynchronous ParallelParameter Server in DistBelief

DistBelief 把训练数据分为多个子集，然后在每个子集上运行模型的多个副本，模型通过集中式的参数服务器通信，参数服务器存放了模型的全部参数和状态。异步体现在两方面：(1) 模型的副本独立运行；(2) 参数服务器的分片也各自独立运行。模型通过一个集中参数更新服务器通信，使得当前状态的参数模型在许多机器上分片 (例如，如果我们有 10 个参数服务器碎片，每个分片负责存储和更新模型的参数，如图 1)。

在最简单的实现中，每次处理每个 mini-batch 之前，模型副本向参数服务器请求其模型参数的更新副本。因为 DistBelief 参数服务器是跨多台机器划分的，参数服务器的碎片持有与其分区相关的模型参数，所以每台机器只需要与参数服务器碎片的子集进行通信。DistBelief 模型副本收到其参数的更新副本后，使用一个 mini-batch 来计算参数梯度，并将梯度发送给参数服务器，然后用梯度更新模型参数的当前值。

DistBelief 的系统实现主要解决了 straggler 问题，即由于一些软硬件的原因，节点的计算能力往往不尽相同。对于迭代问题来说，每一轮结束时算得快的节点都需等待算得慢的节点算完，再进行下一轮迭代。这种等待在节点数增多时将变得尤为明显，从而拖慢整体的性能。DistBelief 放宽了“每个迭代步都等待”这个约束：当一轮迭代结束时，算得快的节点可以继续下一轮迭代，但不能比最慢的节点领先参数 s 个迭代步。当领先超过 s 个迭代步，DistBelief 才会强制进行等待。这样异步的控制方式既从整体上省去了等待时间，也能间接地帮助慢的节点赶上。从优化问题的角度来看，虽然单迭代步收敛变慢，然而每个迭代步的时间开销变少，总体上收敛也变快。

2.2.2 分析

ASP 中各个 worker 之间的计算相互独立，无需等待其他 worker 计算的梯度即可更新整体梯度，接着进行下一轮梯度计算。一方面，异步更新策略增加了分布式系统整体的吞吐量，减少节点因互相等待其它节点而造成的时间浪费。另一方面，异步并行策略也带来了一些问题。在这种情况下，一些较慢的 worker 会用延迟或过时的梯度来更新参数服务器上的全局共享权重，从而给参数迭代带来误差，

延长训练模型的收敛速度，甚至当过时的更新来自非常旧的迭代时，会偏离模型的学习。在没有任何同步的情况下，每个 worker 可以在迭代结束时从参数服务器获得不同版本的全局权值。从系统的角度来看，全局权值在一开始对所有 worker 是不一致的。解决这一问题的办法是：分布式训练中的参数服务器可以配置不同比例的参数服务器和工作节点，然而难以确定合适的比例。如果参数服务器数量过少，那么它可能会成为网络计算的性能瓶颈；如果过多，则可能导致网络互连饱和。

2.3 延迟同步并行（Stale Synchronous Parallel）

2.3.1 工作原理

SSP 参数服务器是 BSP 和 ASP 两者的结合，通过限制最快 worker 和最慢 worker 之间迭代次数差值在一个阈值 s 以内，保证各个 worker 间计算的进度不会差太多，因此只要该差值保持在 s 以内，各 worker 之间就无需等待，系统可以异步地进行；一旦差值超过了阈值，最快 worker 会被迫同步，等待系统中最慢的 worker。尽管在 SSP 的系统中，每次迭代开始后系统中分布的参数就是不一致的，但是延迟同步的阈值将这种不同步限定在一定步数之内，该机器学习模型仍然可以收敛。

在 SSP 系统中，假设有 P 个 worker，即 P 个负责参数计算的线程，每一次迭代都对系统共享的参数 x 作出更新 $x \leftarrow x + u$，代表该机器学习算法中某一计算步骤，每个 worker 都有自己的时钟 c，worker 只有在时钟周期的末尾提交最新的计算结果，提交的更新也不一定会对其他想要读 x 的线程可见，也就是说工作线程只能看见其他所有工作线程的更新的一个过时的子集。因为容忍一定程度的延迟，工作线程可以从本地机器的缓存取回数据而不是每次都要通过通信来向参数服务器请求数据。

用户给定一个可以容忍的延迟阈值 $s \geq 0$，SSP 系统自然满足关于该有界最大延迟的条件（参考图 2）：

- 最快和最慢工作线程的时钟差必须不超过 s，否则最快的工作线程需要等待最慢的线程。
- 当一个工作线程在时钟周期 c 提交一个更新 u 时，u 应该带上时间戳 c。
- 当一个工作线程在时钟周期 c 想要读 x 时，它必须能够读到所有时间戳不超过 $c - s - 1$ 的更新 u 在 x 上产生效果后的结果，同时也可以包括一些时间戳在 $c - s - 1$ 之后的更改 u。
- 保证写后读一致：一个工作线程 p 任何时候看到的都是自己在完成写操作 u_p 后的结果。

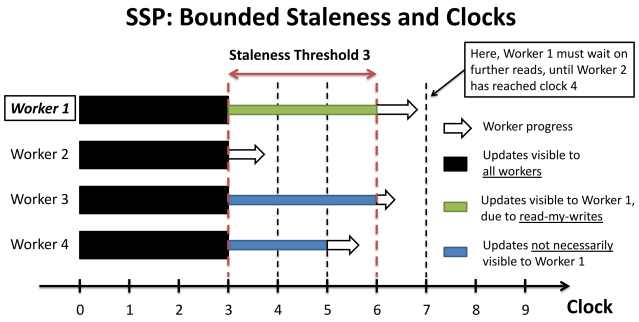

图 2: Bounded Staleness under the SSP Model

由于最快和最慢的工作线程之间相差不超过 s 步，一个线程在时钟 c 的时候读 x，一定能够看到时间戳在 $[0, c - s - 1]$ 范围内更新，有可能能看到 $[c - s, c + s - 1]$ 范围内的更新。

2.3.2 具体实现——SSPtable

这一部分介绍 SSPtable 的实现。SSPtable 是一种基于 SSP 模型的参数服务器，并且可以并行地运行在多态服务器上。如下图所示，SSPtable 遵循客户-服务器的分布式架构，客户为了访问共享的训练参数，通过调用相应的库访问进程缓存和（可选）每个线程的局部缓存，后者由于减少了单个机器上的线程间同步能够一定程度上提高性能。在 SSPtable 上的编程在读写参数 x 时采用一些简单的 API 即可：

• 参数表的组织：SSPtable 用表的形式来存储参数 x，对表的数量不设限，这些表需要划分成行，每一行需要进一步划分成元素。

• `read_row(table,row,s)`：取回某一行，延迟阈值为 s，用户可以用来访问特定的行或元素。

• `inc(table,row,el,val)`：在某张表的某一行某个元素上加 `val`，`val` 可以是负值，但这些更新只有在下次调用 `clock()` 的时候才传给服务器。

• `clock()`：告诉所有服务器当前线程已经完成了一个时钟周期的运算，并提交所有未提交的 `inc()` 写操作。

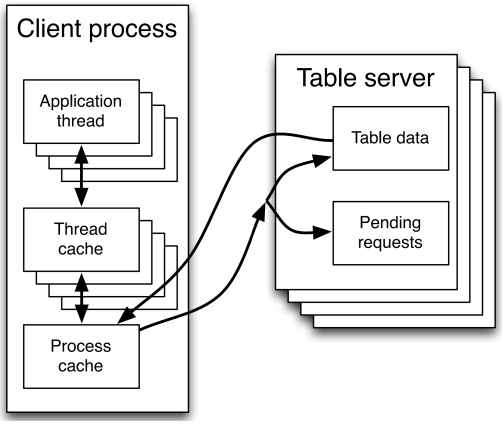

图 3: Cache structure of SSPtable, with multiple server shards

为了保证最快和最慢的线程之间最多不相差 s 步，SSPtable 会让最快的线程在调用 `read_row()` 时阻塞住，知道最慢的线程赶上来。为了保证写后读的一致性，缓存采用了写回策略：所有写操作会写到线程局部 cache，等到调用 `clock()` 的时候再写回进程 cache 和服务器。

为了保证有界延迟的同时最小化 `read_row()` 的阻塞时间，SSPtable 采用如下 cache 协议：线程或进程 cache 上的每一张表的每一行都附有相应的时钟 r_{thread} 和 r_{proc}；每个工作线程都有一个时钟 c，c 就等于该线程调用 `clock()` 的次数；并由此定义服务器时钟 c_{server}，它等于所有线程时钟 c 的最小值。当一个线程在时间 c 请求某一行时，首先检查线程私有 cache，如果其时间戳满足 $r_{thread} \geq c - s$ 就直接读线程 cache；否则，检查进程 cache，如果进程 cache 的这一行满足 $r_{proc} \geq c - s$ 就读进程 cache。目前为止都只需要访问本地的缓存。如果本地缓存都没有命中，就向服务器发送请求，工作线程此时被阻塞，直到服务器返回相应的行及相应的时间戳 c_{server}。取回数据后，线程需要将最新的数据及时间戳 c_{server} 写入线程和进程 cache，并更新 r_{thread} 和 r_{proc}。

2.3.3 分析

SSP 的 cache 一致性协议的优点是最慢的工作线程只需要每 s 个时钟周期向服务器发送一次这种需要阻塞的读请求，而较快的线程向服务器发送读请求会相对频繁一些，最坏情况也只有最快线程每个时钟周期都需要和最慢线程同步。这是和 BSP 不同的地方，BSP 需要每个周期所有线程都同步。因此 SSP 不仅能够减少和远程服务器的同步，减少网络通信开销，还能让运行比较慢的线程避免在一些迭代中读服务器，较慢的线程也可以渐渐跟上，同时运行较快的线程可以在一定程度上异步地计算而不需要阻塞同步。因此整体来说，SSP 最大化了每台机器上的计算资源的利用率，减少了阻塞同步带来的开销。

3 动态 DSSP（Dynamic Stale Sychronous Parallel）

3.1 研究背景

SSP 是介于 BSP 和 ASP 之间的一种中间方案。它的速度比 BSP 快，而且保证了收敛性，从而得到比 ASP 更准确的模型。然而，在 SSP 中，用户指定的陈旧度阈值是固定的，这导致了两个问题。首先，用户通常很难指定一个好的单阈值，因为用户不知道哪个值是最好的。选择一个好的阈值可能需要在一个整数范围内通过无数次试验手动搜索。此外，DNN 模型还涉及许多其他超参数（如层数和每层的节点数）。当这些参数发生变化时，必须再次重复相同的搜索试验。其次，单一的固定值可能不适合整个训练过程，一个不明确的值可能会导致最快的 worker 等待的时间比必要的时间长。

为了解决这些问题，接下来介绍动态 SSP 的研究。DSSP 提出在训练过程中，根据分布式计算资源实时处理速度的统计，从给定范围中动态选择一个阈值。它允许阈值随时间变化，也允许不同的 worker 节点有不同的阈值，以适应运行时的环境。

3.2 工作原理

定义 S_L 和 S_U 为 staleness 的上界与下界，DSSP 要寻找的是整数 $r^* \in [0, r_{max}]$，$r_{max} = S_U - S_L$，且 $r^* = s^* - S_L$，其中 $s^* \in [S_L, S_U]$.

以随机梯度下降为例，算法分为 worker、server 和同步控制器三部分：

• worker 部分：首先，每个 worker 上都有 model 的副本以及被分配的一部分 data，每次迭代时在本地使用随机梯度下降（SGD）计算参数，随后上传至 server。等待 server 发回 OK 信号后，继续从 server 端拉取一部分更新迭代过的参数，继续使用下一批 mini-batch 做 SGD 更新本地参数，如此循环直至训练结束。

• server 部分：server 收到某个 worker p 上传的梯度参数后，立即更新全局参数。然后开始判断是否给 worker p 发送 OK 信号：server 存储了从每个 worker 收到的上传次数，并找到最慢的 worker，若 worker p 的上传次数与最慢的 worker 差距不超过 s_L 次迭代，则向 worker p 发送 OK 信号，令其继续迭代；否则，若 worker p 是当前最快的 worker 节点，server 要通过调用同步控制器（synchronization controller）来决定是否允许 worker p 继续迭代。

• 同步控制器（synchronization controller）部分：在表中存储所有 worker 节点最近两次上传请求的时间戳，使用该表的信息来模拟 worker p 和最慢的 worker 之间的 r_{max}，同步控制器要在 $[0\ r_{max}]$ 区间找到一个时间点 r^*，使得 worker p 的等待时间最小化。并将这个时间点 r^* 发送给 worker p，让 worker p 继续迭代到时间点 r^*。

4 总结

参数训练由快到慢依次是：BSP→SSP→DSSP→ASP。

DNN 中有无全连接层对训练迭代有着很大的影响。在有全连接层的 DNN 训练中，BSP 有着最高的精度，DSSP 有着最快的收敛速度且精度比 SSP 和 ASP 高；而在无全连接层的 DNN 训练中，BSP 的收敛速度最慢，精度也比较低，ASP 收敛速度最快，DSSP 在阈值 $[3, 15]$ 的区间中表现略高于 SSP 的平均水平。

造成这种影响的原因是：与使用共享参数的卷积层相比，全连接层需要更多的参数，有全连接层的 DNN 有大量的模型参数需要在 worker 和 server 之间传输更新。卷积层的矩阵点积运算的计算时间开销更大，带全连接层的计算时间开销较小，传输时间开销更大。因此，DSSP、SSP 和 ASP 在 DNN 上花费的训练时间较少，而 BSP 在 CNN 上花费的训练时间最少。

参考文献

[1] Gerbessiotis A V, Valiant L G. Direct bulk-synchronous parallel algorithms[J]. Journal of parallel and distributed computing, 1994, 22(2): 251-267.

[2] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010: 135-146.

[3] Goudreau M, Lang K, Rao S, et al. Towards efficiency and portability: Programming with the BSP model[C]//Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures. 1996: 1-12.

[4] Dean J, Corrado G, Monga R, et al. Large scale distributed deep networks[J]. Advances in neural information processing systems, 2012, 25: 1223-1231.

[5] Recht B, Re C, Wright S, et al. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent[J]. Advances in neural information processing systems, 2011, 24: 693-701.

[6] Chilimbi T, Suzue Y, Apacible J, et al. Project adam: Building an efficient and scalable deep learning training system[C]//11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). 2014: 571-582.

[7] Ho Q, Cipar J, Cui H, et al. More effective distributed ml via a stale synchronous parallel parameter server[J]. Advances in neural information processing systems, 2013, 26: 1223-1231.

[8] Cui H, Cipar J, Ho Q, et al. Exploiting bounded staleness to speed up big data analytics[C]//2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014: 37-48.

[9] Zhao X, An A, Liu J, et al. Dynamic stale synchronous parallel distributed training for deep learning[C]//2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019: 1507-1517.