BAFL: A Blockchain-Based Asynchronous Federated Learning Framework 的改进

李英濠

摘要

联邦学习 (FL) 作为一种新兴的分布式机器学习 (ML) 方法，可以通过人工智能 (AI) 模型跨大量设备的协作学习来保护数据隐私。然而，低效率和对中毒攻击的脆弱性降低了 FL 的性能。采用一种新的熵权方法来评价设备 BAFL 中训练的局部模型的参与等级和比例。通过调整局部训练和通信延迟，优化块生成速率，平衡能量消耗和局部模型更新效率。广泛的评估结果表明，与其他分布式机器学习方法相比，所提出的 BAFL 框架在预防中毒攻击方面具有更高的效率和性能，我基于论文逻辑的前提下提出了一种改进方法使得模型总体收敛速度得到提升。

关键词：区块链; 联邦学习; 安全; 异步学习; 学习效率

1 引言

随着机器学习方法的迅速发展，许多新兴的移动应用都采用了这种方法，如自动驾驶、销售预测、视觉安全等，提高了用户的服务体验[1]。尽管机器学习已经显著提高了移动应用程序的性能，但在网络中使用的传统机器学习方法需要将大量包含个人信息的数据存储在中央服务器中，以执行模型训练。这导致中央服务器的计算开销和性能损失急剧上升。与此同时，集中用户数据的做法引发了对隐私和信息滥用的担忧[2]。联邦学习 (FL) 是一种分布式机器学习方法，它的开发就是为了解决这些挑战。联邦学习以去中心化的方式训练全局模型。移动设备对本地模型进行迭代训练，上传本地训练模型并发送到中央服务器。因为没有将原始用户数据[3]传输到中央服务器，联邦学习保护了用户的隐私，并将数据采集、训练和在中央服务器上存储模型的机器学习过程解耦。

然而，传统的联邦学习在网络中的应用存在一些局限性。第一个局限是每个设备学习模型的可靠性。恶意设备可以通过调整本地模型对全局模型产生不利影响。第二，联邦学习完全依赖于中央服务器的可靠性。如果中央服务器出现故障或恶意修改全局模型，则所有后续本地模型的更新精度将大幅降低。另一个问题是滞后效应，这意味着每一轮训练只能以最慢设备的训练速度进行。这使得高计算能力的设备不可能在本地更新完成后及时上传本地模型。

本文提出了一种基于区块链的异步联合学习 (BAFL) 体系结构，该体系结构将区块链与异步联合学习相结合，具有信任和学习效率的优点。本文的主要贡献如下:

- 与传统的联邦学习不同，BAFL 采用异步联邦学习策略，允许每个设备在全局聚合需要快速收敛全局模型时上传本地模型。区块链还可以防止单个中央服务器故障，保证分散和安全的数据存储。此外，区块链通过奖励设备以此激励设备参与到联邦学习中。
- 为保证启用区块链的联邦学习的效率，BAFL 工作流设计了两个互补策略: 一是优化控制块生成率，减少联邦学习的延迟; 二是动态调整异步联邦学习中的训练次数，防止频繁上传本地模型导致区块链事务过载。

• 本文不采用传统的联邦平均 (federal average, FedAvg) 算法，而是对设备 BAFL 训练的局部模型的参与等级和比例进行评估。这些指数通过熵权方法获得，考虑了训练时间、训练样本大小、局部更新相关性和全局更新作弊次数，并记录在区块链中，用于长期信任。

• 在必要的区块链和异步联邦学习过程中，在通信、存储和学习计算中也考虑了 BAFL 中设备的能量消耗。采用帕累托优化策略使设备的能耗最小化，并在不使区块链过载的情况下尽快上传设备的模型。

2 相关工作

已经提出了各种联邦学习方法来提高速度和收敛性，并提高训练过程的准确性。提高用于模型训练的随机梯度下降 (SGD) 优化算法学习效率最简单的方法是保证每个设备并行迭代几轮，提高参数估计，并将数据上传到中央服务器进行聚合。FedAvg 算法使用了这种方法，这是第一个联邦学习算法。随后，Chen 等人[4]在服务器端引入了时间加权聚合策略，并使用先前训练的局部模型来提高中心模型的准确性和收敛性。Wang 等人[5]采用了一种正交方法，该方法识别客户端进行的不相关更新，并防止其被上传，以减少网络使用。此外，参考文献[6]引入了异常检测器，以提高框架对拜占庭设备的鲁棒性。许多研究通过平衡全局延迟和能耗来提高效率并降低能耗。参考文献[7]将无线网络的联邦学习策略视为优化问题。使用无线网络联合学习问题（FEDL）来最小化延迟并降低能耗，同时确保高精度。类似地，Li[8]提出了一种具有分级在线速度的智能电脑控制框架的联邦学习策略，在节省能量的同时平衡了训练时间和模型精度。参考文献[9]和[10]使用神经网络来分配资源以减少延迟。Lu[9]开发了一种优化策略，通过分解联邦学习并使用深度神经网络（DNN）进行通信资源分配，降低了联邦学习通信的成本并提高了整体性能。类似地，Zhan[10]设计了一种基于深度强化学习（DRL）的经验驱动方法，以平衡能量消耗和延迟。在[11]中，Zhou 利用 Lyapunov 优化理论为联邦学习系统独立并同时建立了四个在线控制决策，包括准入控制、负载平衡、数据调度和精度调整。类似地，Luo 使用分层联合边缘学习（HFEL）框架[12]建立了联合计算和通信资源调度模型，以最小化全局成本。然而，这些研究只考虑了同步场景，没有考虑异步场景，系统安全性不高。

恶意篡改模型是联邦学习的一个重大问题。区块链是一个分散的共享账本，它将数据块按时间顺序组合成一个独特的数据结构，并通过密码学确保数据不能被篡改或伪造。区块链内部各节点共同维护和共享数据。区块链与联邦学习组合使用，可以防止恶意设备的攻击。[13]中，所有本地设备都连接到一个区块链上并参与学习。得到学习率、数据大小和局部学习精度的加权平均值。但权重保持不变，且未考虑设备的先验信息。Majeed 等人设计了一个由边缘设备组成的区块链网络组成的联邦学习架构[14]，并在区块链网络中建立了一个独立的通道来学习全局模型。[15]和[16]中，区块链用于隐私保护和激励设备。Martinez 等人使用区块链和区块链外部数据库的组合，并在[15]中提出了可伸缩的梯度记录和奖励工作流。[17]和[18]中提出了分散的联邦学习区块链模型。

同步联邦学习的问题是上传时间只和最慢的设备一样快。在[19]中，为了提高联邦学习的可扩展性和效率，引入了异步联邦学习的概念。在异步联邦学习中，本地模型在完成更新后立即上传。因此，与同步联邦学习不同的是，异步联邦学习不会出现延迟。但是，异步联邦学习容易受到毒化攻击。此外，由于异步联邦学习中上传时间的随机性，设备上传时间的调度是一个复杂的问题。Cong 等人在

随后的研究[20]中对[19]中提出的方法进行了改进，提出了一种新的异步联邦学习优化方法。该方法对强凸和非凸问题以及有限族非凸问题具有全局最优的近线性收敛性。然而，作者没有考虑系统的鲁棒性。在本文中，我们将区块链支持的负载均衡与异步负载均衡相结合，提出了一种新的负载均衡优化方法和设备权重指标。据我们所知，这是第一次将区块链用于异步联邦学习的模型训练。所提出的方法具有令人满意的精度和较高的安全性。

3 本文方法

3.1 初始化

在异步区块链联邦学习中，联邦学习的设备集群记为$D = \{1, 2, \ldots, N_D\}$ 其中$|D| = N_D$ 表示设备总数。第 i 个设备处理它自己的数据集S_i，其中$|S_i| = N_i$。与该设备相关联的区块链中的矿工为M_i，从矿工集群$M = \{1, 2, \ldots, N_M\}$ 中随机选择，其中$|M| = N_M$。为了保证每个设备分配给一个矿工(即$N_M \geqslant N_D$)，需要比设备数量更多的矿工。

区块链系统发布每个设备的最早截止日期，在本研究中，每个设备分配一个矿工。每个设备都必须确定自己的任务相关数据集大小，并将其上传到区块链系统以接收分数。本地数据的质量决定了本地模型的质量。每个设备的本地速度控制器执行本地优化以确定设备资源（例如 CPU）的最佳调度，以确保本地设备节省最大能量并具有最低延迟。

3.2 基于区块链的异步联邦学习的单周期操作

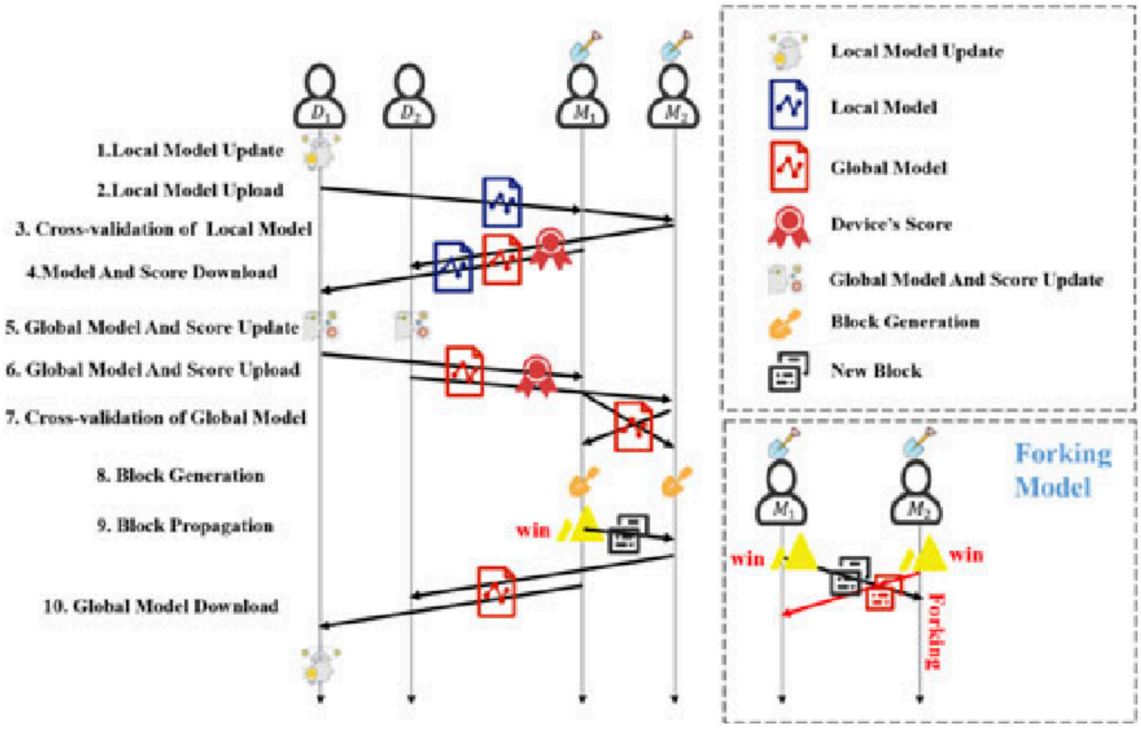

图 1: 操作界面示意

如图 1所示，装置D_i 在第l 轮 epoch 的操作包括以下十个步骤:

步骤 1: 本地模型更新。局部模型的更新过程是对局部模型进行迭代训练，局部求解线性优化问题。设备D_i 有一个本地数据集S_i，其中每个数据样本s_k 用$s_k \{x_k, y_k\}$ 表示; 其中x_k 是高维向量，y_k 是标量。局部训练的目标是最小化损失函数$f(\omega_i; S_i)$，其中ω_i 是设备D_i 的本地模型，S_i 是设备 i 的本地数据集。设备D_i 使用其数据集S_i 和接收到的全局模型ω 来训练局部模型。第l 个全局 epoch 和第h 个

局部迭代的局部模型表示为

$$\omega_i^{(h,l)} = \omega_i^{(h-1,l)} - \gamma \nabla f\left(\omega_i^{(h,l)}; s_i^h\right) \tag{1}$$

其中，γ 为学习率

步骤 2: 本地模型上传。设备D_i 与矿工M_i 随机关联，设备D_i 将本地模型上传到矿工M_i。

步骤 3: 局部模型的交叉验证。一旦矿机M_i 接收到设备D_i 的本地模型，它将该模型写入自己的新块，并在区块链级别广播设备D_i 的本地模型。当其他矿工收到本地模型时，他们验证其准确性，并将其记录到他们的新区块。

步骤 4: 模型和分数下载。每个设备从关联矿机下载当前全局模型、设备D_i 的本地模型、设备D_i 的当前评分和设备D_i 的本地更新持续时间。

步骤 5: 全局模型和分数更新。在此步骤中需要解决两个问题。第一个是全局模型的聚合。在同步联邦学习聚合中，所有本地模型都被聚合，而在异步联邦学习聚合中，一次只能聚合一个本地模型。聚合过程定义如下:

$$\omega^{(l)} = \left(1 - \alpha^{(l)}\right) \omega^{(l-1)} + \alpha^{(l)} \omega_{\text{new}} \tag{2}$$

其中$\alpha^{(l)}$ 为新模型的权重。$\alpha^{(l)}$ 越大，新生成的局部模型在更新中对全局模型的贡献就越大。第二个问题是计算设备D_i 的新评分。每台设备使用设备D_i 的评分、设备D_i 的局部模型和全局模型来计算本轮设备D_i 的评分。

步骤 6: 全局模型和分数上传。每个设备将新的全局模型和设备D_i 的新分数上传到其关联的矿工。

步骤 7: 全局模型的交叉验证。在接收到第 6 步的全局模型和评分后，矿工广播接收到的全局模型和上传模型的设备号 (与矿工关联的设备) 到区块链层。一旦所有设备上传的全局模型都被广播到网络上，所有的矿工就会比较接收到的全局模型之间的一致性，并使用最多的全局模型作为正确的全局模型。其余的全局模型被视为有问题的模型，产生它们的设备被标记为提供恶意全局更新的设备，并被记录下来。最后，所有矿工将正确的全局模型和设备的分数记录到新块中。

步骤 8: 块生成。在现有的 PoW 算法中，每个矿工解决一个基于 SHA256 的密码谜题，这是一个单向哈希函数，用于块生成。每个矿工必须计算一个块哈希进行挖掘。在这一点上，矿工计算哈希值低于某个目标值将生成一个块。挖掘器将 nonce 值递增，直到找到指定的散列目标值。在 PoW 算法中，假设所有矿工具有相同的哈希能力，随机确定块头的哈希结果。在这个步骤中，每个矿机运行一个共识机制 (PoW)，直到它找到所需的 nonce 或接收到生成的块。

步骤 9: 块传播。将$M_o \in M$ 表示为第一时间找到所需要的矿工。它的候选块被生成为一个新的块，并被广播给其他矿工。当区块链系统中的多个矿工同时发现一个 nonce 时，就会发生分叉。为了避免分叉，一旦每个矿工接收到新块，就会发送一个 ACK，以确定是否发生分叉。矿工M_o 生成一个新块，等待由块 ACK 预定义的最大等待时间。如果已生成 fork，则返回步骤 8。区块的验证过程包括哈希加密和非对称加密。哈希算法将任意长度的二进制值映射为更短的固定长度的二进制值，这称为哈希值。哈希是一段数据的唯一且高度紧凑的数字表示。如果对一段明文进行了散列，并且该段中的一个字母发生了更改，则后续的散列将产生不同的值。矿工M_o 首先对新块中的数据执行哈希算法，生成块哈希值H_o，然后用自己的私钥对哈希值进行加密，生成签名信息。然后将此消息广播到区块链 (广播内容 = 新块 + 签名信息)。在矿工 M 预测矿工M_o 的广播内容之后，矿工M_p 使用矿工M_o 公钥对

签名信息执行逆运算，即解密过程，以证明签名信息属于矿工M_o，矿工M_p 对块数据执行哈希运算以获得哈希值H_p，以确定哈希值H_o 是否与哈希值M_p 一致。如果一致，则数据传输正确；否则，所接收的数据已经改变，并且验证不成功；因此事务失败。最后，矿工M_p 将新块中的数据与存储的数据进行比较。如果数据不一致，则假设矿工M_o 篡改了数据。

步骤 10: 全局模型下载。设备D_i 从其关联的矿工下载全局模型。重复这些步骤，直到全局模型达到所需的学习率或收敛为止。

3.3 设备有效性的评估

由于网络结构的复杂性，设备之间的距离较大。还有一些设备在全局聚合期间篡改局部模型或全局模型。选择一种全面评估设备质量的方法是很重要的。熵权法是一种客观加权法。根据指标的相对变化对分数阶系统的影响，通过计算指标的信息熵 (熵是系统无序程度的度量) 来确定指标的权重。相对变化程度越大，指数权重越高。这种方法非常适合于确定权重。许多研究采用熵权法来建立模糊指标。Yang 等人[21]分析了即时消息网络拓扑结构的影响因素。建立评价指标矩阵，采用熵权法确定指标权重。Yu 等人提出了一种分析结构化文本并提取概念[22]的新方法。该方法考虑了文本的结构，并使用熵权来评价概念。作者定量评估了模块的贡献，以确定概念的权重。Chen et al.[23]提出了一种基于熵权法的电力用户信用评估的科学方法。该方法通过考虑基本信用、过去信用、固有属性和调整因子，对客户的信用进行综合评价。在本研究中，我们使用多个指标来评估设备的评分，例如数据大小、全局模型更新时的错误次数和历史评分。我们使用熵权法来估计指标的权重。设备的分数存储在每个矿工的账簿中。评分影响设备在联邦学习中的权重和分配给设备的更新数量。由于神经网络的复杂性，不可能通过比较训练模型的精度来评估模型的更新。因此，我们在 BAFL 中使用模型相关性来解决这个问题。模型训练的本质是更新梯度向量中的参数。因此，我们比较局部模型与全局模型之间各参数的相关性，以确定局部训练性能。具体来说，设$u = \{u_1, u_2, \ldots, u_D\}$ 是局部模型参数，$\bar{u}$ 是全局模型参数。利用欧几里得度量公式确定设备D_i 的局部模型更新与第l 轮中的全局模型更新之间的相关性

$$\mathrm{dis}_{\mathrm{new}}^{(l)}(u, \bar{u}) = \sqrt{\sum_{j=1}^{D} (u_j - \bar{u}_j)^2} \tag{3}$$

越大的$dis(u, \bar{u})$，本地模型和全局模型的相关性就越小。采用熵权法获得设备评分。在本研究中，s_i^k 为设备D_i 的 k 指标的归一化值。然后我们计算s_i^k

$$p_i^k = \frac{s_i^k}{\sum_{i=1}^{N_D} s_i^k}, i = 1, 2, \ldots, N_D, k = 1, 2, ..4 \tag{4}$$

其中$k = 1$ 表示数据大小指标，$k = 2$ 表示设备D_i 的过去得分，$k = 3$ 表示全局模型更新的错误次数，$k = 4$ 表示局部模型更新与全局模型更新的相关性。则指标 k 的熵权为

$$w_k = \frac{1 - e_k}{4 - \sum_{k=1}^{4} e_k}, \sum_{k=1}^{4} w_k = 1 \tag{5}$$

其中$e_k = -\dfrac{1}{\ln(N_D)} \sum_{i=1}^{N_D} p_i^k \ln\left(p_i^k\right)$ 指标的熵权越大，对设备D_i 得分的贡献越大。最后得到设备的评分

$$\tau_i = \sum_{k=1}^{4} w_k \times s_i^k \times 10^{-2} \tag{6}$$

3.4 基于牛顿冷却法的权重法

牛顿冷却定律指出，物体的冷却速度与物体自身与周围环境温度 T0[25] 之间的温度差成正比。当物体的温度高于周围环境时，它会逐渐将热量传递给周围的介质，并逐渐冷却

$$T'(t) = \frac{dT}{dt} = -\theta\left(T(t) - T_0\right), k > 0 \tag{7}$$

其中$T'(t)$ 是温度 T 对时间 T 的导数，即冷却速率。常数 u 是周围温度与冷却速率的比值。不同物质有不同的 u 值。

牛顿冷却定律用于确定联邦学习中的权重 (式 (2))。在异步联邦学习中，每个设备的上传时间是不确定的。局部训练时间过长会导致局部模型过时，降低全局模型的准确性。因此，采用牛顿冷却定律对旧模型在全局模型聚合中的权重进行调整。设备D_i 在时间 t 时的值为

$$R(t) = R_0 \times e^{-\theta(t-t_0)} \tag{8}$$

其中$R0$ 是$t0$ 时刻的新鲜度。θ 为衰减系数，与系统有关。θ 越大，衰减系数越大，训练时间较长的模型在全局模型中所占比例越小。

结合式 (2)，设备D_i 在全局聚合

$$\alpha^{(l)} = \xi R(t)\tau_i^{(l)} \tag{9}$$

为第l 个 epoch 上传的局部模型的权重; 其中ξ 为超参数，表示新局部模型在全局模型中的初始权重。

完成上述步骤后，计算设备的局部模型在全局模型中的权重。我们在全局聚合中调整每个设备的权重，并对时间进行加权，以防止过时的模型影响全局精度。该方法显著提高了全局模型的精度和收敛性。

4 改进流程

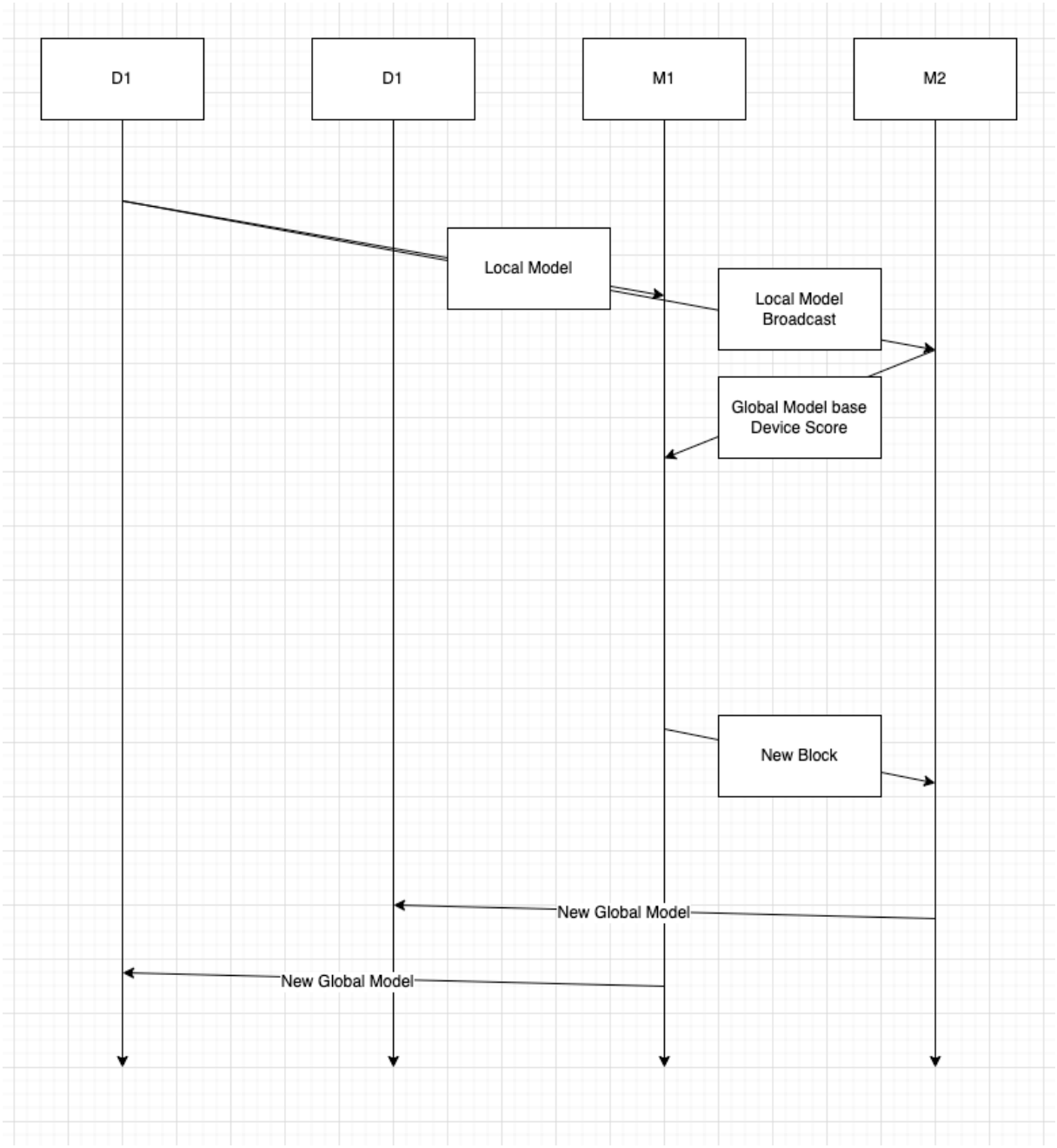

图 2: 改进流程示意

原有的流程是所有训练终端对其他终端提交的局部模型进行打分然后提交，这样的做法是将打分的权利进行分散防止某个恶意的中心化节点掌控打分权力，但是所有的训练终端都要参与整个过程会极大消耗全网的算力资源，因此我提出的改进是不由所有训练终端来进行打分而是由全网的矿工来负责这样子也能够确保打分结果是去中心化的而且也避免了全网所有终端都参与训练消耗全网算力资源的这个弊端。

可以看从图 2可以看到现在的流程被简化为了训练终端只负责产生 LocalModel，矿工负责将收到的 LocalModel 广播给其他矿工其他矿工根据当前的状态上下文结合上文提到的熵权法以及牛顿冷却法来获得学习率最后生成得到全局模型再在矿工之间进行广播，最后挖到区块的矿工用重复次数最多的全局模型作为下一个区块的全局模型。

5 复现细节

5.1 python 客户端实现

5.1.1 初始化数据集

```
if args.dataset == "mnist":
      trans_mnist = transforms.Compose([transforms.ToTensor(), transforms.
         Normalize((0.1307,), (0.3081,))])
      dataset_train = datasets.MNIST('../data/mnist/', train=True, download=
         True, transform=trans_mnist)
      dataset_test = datasets.MNIST('../data/mnist/', train=False, download=
         True, transform=trans_mnist)

      if args.iid:
         dataset_train = iid(dataset_train, args.start_train_index, args.
            end_train_index)
         dataset_test = iid(dataset_test, args.start_test_index, args.
            end_test_index)
      else:
         dataset_train = non_iid(dataset_train, args.user_id, args.num_users)
         dataset_test = non_iid(dataset_test, args.user_id, args.num_users)
   elif args.dataset == "cifar":
      if args.iid == False:
         exit("cifar don't support non_iid")
      trans_cifar = transforms.Compose(
         [transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5,
            0.5, 0.5))])
      dataset_train = datasets.CIFAR10('../data/cifar', train=True, download=
         True, transform=trans_cifar)
      dataset_test = datasets.CIFAR10('../data/cifar', train=False, download=
         True, transform=trans_cifar)
   else:
      exit("don't support dataset")
```

以上代码主要做了如下步骤：

- 判断数据集类型（目前支持 mnist 与 cifar）
- 将数据集进行归一化
- 下载训练集与测试集
- 判断要求的数据集类型是 iid 才是 niid 根据需求切割数据集给训练终端

5.1.2 注册客户端

```
   node = Node(device=device, epoch=args.epoch, dataset_train=dataset_train,
      dataset_test=dataset_test, lr=args.lr)
   register(len(dataset_train.idxs))
   def register(dataSize):
    resp = requests.post(args.rpc_url, json={
       "jsonrpc": "2.0",
       "method": "eth_register",
       "params": [
          args.address,
          dataSize
       ],
       "id": 1
    }, headers={
```

```
        "Content-Type": "application/json; charset=UTF-8"
    })
```

以上代码将实例化一个训练终端实例，并将终端信息通过 HTTP POST 方法调用 GETH 的 RPC 接口注册到终端关联的矿工上

5.1.3 开始训练

```
while True:
      net_glob = getNetGlob().to(device=device)
      node.train(net_glob)
      net_glob.eval()
      acc_test, loss_test = test_img(net_glob, dataset_test, args.epoch, args.
         gpu)
      uploadWeights(net_glob)
      waitNewGlobalModel()

def train(self, net_glob):
   optimizer = torch.optim.SGD(net_glob.parameters(), lr=self.lr)
   for iter in range(self.epoch):
      for batch_idx, (images, labels) in enumerate(self.ldr_train):
         net_glob.train()
         images, labels = images.to(self.device), labels.to(self.device)
         net_glob.zero_grad()
         log_probs = net_glob(images)
         loss = self.loss_func(log_probs, labels)
         loss.backward()
         optimizer.step()
         if batch_idx % 10 == 0:
            print('Update Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
               iter, batch_idx * len(images), len(self.ldr_train.dataset),
                     100. * batch_idx / len(self.ldr_train), loss.item()))
```

以上代码通过一个 while 进行循环，每一轮调用 getNetGlob 来获取当前最新的全局模型然后调用 Node 实例的 train 方法利用本地的数据集进行训练，训练完成后用本地的测试集进行验证得到 acc 与 loss，最后将本地训练后的模型参数上传给矿工等待新的区块生成。

训练主体采用 pytorch 框架进行训练，优化方法为随机梯度下降法。在给定的 epoch 下进行迭代训练并利用损失函数计算出来损失值并进行反向传播。

5.1.4 等待新的全局模型

```
def waitNewGlobalModel():
   global curBlockNumber
   while True:
      data = requests.post(args.rpc_url, json={
         "jsonrpc": "2.0",
         "method": "eth_blockNumber",
         "params": [],
         "id": 83
      }).json()
      result = data['result']
      blockNumber = int(result, 16)
      if blockNumber == curBlockNumber:
         time.sleep(5)
         continue
```

```
        curBlockNumber = blockNumber
        break
```

以上代码通过轮询调用 GETH 获取最新的区块高度判断与上次同步的全局高度是否不同若不同则说明挖到了新的区块则开始新的迭代。

5.2 python 服务端实现

python 服务端采用 Flask 实现的 HTTP 服务端挂载在 GETH 上，GETH 接收到用户的请求后会转发给 python 服务端

5.2.1 处理客户端注册请求

```
@app.post('/register')
def handleRegister():
    data = request.json
    ret = register(data['address'], int(data['data_size']))
    if ret:
        return '1'
    else:
        return '0'
def register(address, dataSize):
    global globalState
    if address in globalState['uid'].keys():
        return False
    globalState['uid'][address] = globalState['n_d']
    globalState['scores'].append(0)
    globalState['t'].append(int(time.time()))
    globalState['n_d'] = globalState['n_d'] + 1
    globalState['s'][0].append(dataSize)
    globalState['s'][1].append(0.5)
    globalState['s'][2].append(1)
    globalState['s'][3].append(0)
    return True
```

以上方法负责接管训练客户端的注册信息，并且初始化训练终端的总数，数据集大小，聚合错误次数，上次设备评分等。

5.2.2 处理客户端获取训练信息请求

```
@app.get("/getTrainInfo")
def getModelType():
    return {
        'model_name': args.model,
        'num_classes': args.num_classes,
        'num_channels': args.num_channels
    }
```

以上方法负责返回当前训练的模型名 (CNN...)、训练分类数、训练通道数

5.2.3 处理客户端新局部模型请求

```
@app.post('/newLocalModel/<address>')
def newLocalModel(address):
    data = request.json
    localStateDict = hexToStateDict(data['local_model_hex'])
```

```
    global net_glob
    globalStateDict = net_glob.state_dict()
    uid = globalState['uid'][address]
    globalState['s'][3][uid] = getDis(globalStateDict, localStateDict)
    # 进行聚合

    res = merge(uid, address, {
        "local_state_dict": localStateDict,
        "global_state_dict": globalStateDict,
        "s": globalState['s'],
        "n_d": globalState['n_d'],
        "uid": uid,
        "t0": globalState['t'][uid]
    })
    requests.post(args.eth_rpc, json={
        "jsonrpc": "2.0",
        "method": "eth_newGlobalModel",
        "params": [
            res['address'],
            res['model_state_hex']
        ],
        "id": 1
    })
    globalState['t'][uid] = time.time()
    globalState['s'][1][uid] = float(res['score'])
    return '1'
def merge(uid, address, data):
    print("getAlpha参数: ", 1, int(time.time()), data['t0'], 0.003, 1, data['uid
        '], data['n_d'], data['s'])
    alpha = getAlpha(1, int(time.time()), data['t0'], 0.003, 1, data['uid'],
        data['n_d'], data['s'])
    print("此次更新率: " + str(alpha))
    if alpha == 0:
        return
    localStateDict = data['local_state_dict']
    globStateDict = data['global_state_dict']

    for k in globStateDict.keys():
        globStateDict[k] = (1 - alpha) * globStateDict[k] + alpha *
            localStateDict[k]

    stateDictHex = stateDictToHex(globStateDict)
    s = normalization(data['s'])
    print("getTauI参数: ", uid, data['n_d'], s)
    score = getTauI(uid, data['n_d'], s)
    return {
        'model_state_hex': stateDictHex,
        'score': score,
        'address': address,
        'cur_global_state_dict': globStateDict
    }

def getDis(globalW, w):
    sumDis = 0
    w_avg = copy.deepcopy(w)
    for i in w_avg.keys():
        sumDis += torch.norm(w[i] - globalW[i], 2)

    return pow(float(sumDis), 0.5)
```

```
def getP(s_k, s_k_i):
    if s_k_i == 0:
        return 0
    sum_s_k = 0
    for cur_s_k_i in s_k:
        sum_s_k += cur_s_k_i
    return s_k_i / sum_s_k


def getEk(N_D, s_k):
    sum = 0
    for i in range(N_D):
        p = getP(s_k, s_k[i])
        if p == 0:
            continue
        sum += p * math.log(p)
    return -1.0 * (1 / math.log(N_D)) * sum


# 根据熵权法取得当前指标的权重
def getWk(N_D, s, s_i):
    sum = 0
    for i in range(len(s)):
        sum += getEk(N_D, s[i])
    return (1 - getEk(N_D, s_i)) / (len(s) - sum)


def getTauI(i, N_D, s):
    sum = 0
    for k in range(len(s)):
        sum += getWk(N_D, s, s[k]) * s[k][i]
    return sum


# 根据牛顿冷却法取得当前模型的权重
def getR(t, t0, theta, R0):
    return R0 * pow(math.e, -1 * (theta * (t - t0)))


def normalization(s):
    res = []
    for k in range(len(s)):
        res.append([])
        sum = 0
        for i in range(len(s[k])):
            sum += s[k][i]
        if sum == 0:
            return 0
        for i in range(len(s[k])):
            res[k].append(s[k][i] / sum)
    return res


def getAlpha(kexi, t, t0, theta, R0, i, N_D, s):
    # 归一化解决时间戳数值过大导致的熵权过小的问题。
    s = normalization(s)
    return kexi * getR(t, t0, theta, R0) * getTauI(i, N_D, s)
```

以上代码负责接收用户发出的新局部模型请求（聚合）、merge 方法负责根据用户的地址再结合熵权法和牛顿冷却法得到当前此用户的学习率、最后进行聚合。

5.3 GETH 改动

5.3.1 与 python 服务端的联动

```
package star

import (
   "bytes"
   "encoding/json"
   "fmt"
   "github.com/ethereum/go-ethereum/common"
   "io/ioutil"
   "net/http"
)

type Backend struct {
   host string
}

func NewBackend(host string) *Backend {
   return &Backend{host: host}
}

func (b *Backend) request(path string, param map[string]interface{}) error {
   bs, err := json.Marshal(param)
   if err != nil {
      return fmt.Errorf("json.Marshal err: %v", err)
   }
   reader := bytes.NewReader(bs)
   req, err := http.NewRequest(http.MethodPost, fmt.Sprintf("%s/%s", b.host,
      path), reader)
   if err != nil {
      return fmt.Errorf("http.NewRequest err: %v", err)
   }
   req.Header.Set("Content-Type", "application/json")

   resp, err := http.DefaultClient.Do(req)
   if err != nil {
      return fmt.Errorf("http.DefaultClient.Do err: %v", err)
   }

   if resp.StatusCode != http.StatusOK {
      return fmt.Errorf("http.statusCode == %d", resp.StatusCode)
   }

   return nil
}

func (b *Backend) Register(addr *common.Address, dataSize uint) error {
   s := map[string]interface{}{
      "address": addr.Hex(),
      "data_size": dataSize,
   }
   err := b.request("register", s)
   if err != nil {
      return fmt.Errorf("b.request err: %v", err)
   }

   return nil
}
```

```
func (b *Backend) NewLocalModel(addr *common.Address, modelStateHex string)
   error {
   s := map[string]interface{}{
      "local_model_hex": modelStateHex,
   }

   err := b.request("newLocalModel/"+addr.Hex(), s)
   if err != nil {
      return fmt.Errorf("b.requestStarBackend err: %v", err)
   }

   return nil
}

func (b *Backend) GetTrainInfo() (map[string]interface{}, error) {
   req, err := http.NewRequest(http.MethodGet, fmt.Sprintf("%s/%s", b.host, "
      getTrainInfo"), nil)
   if err != nil {
      return nil, fmt.Errorf("http.NewRequest err: %v", err)
   }
   resp, err := http.DefaultClient.Do(req)
   if err != nil {
      return nil, fmt.Errorf("http.DefaultClient.Do err: %v", err)
   }
   bs, err := ioutil.ReadAll(resp.Body)
   defer resp.Body.Close()
   if err != nil {
      return nil, fmt.Errorf("ioutil.ReadAll err: %v", err)
   }
   data := make(map[string]interface{})
   err = json.Unmarshal(bs, &data)
   if err != nil {
      return nil, fmt.Errorf("json.Unmarshal err: %v", err)
   }

   return data, nil
}
```

以上代码为转发 GETH 收到的请求给 python 服务端

5.3.2 GETH 增加 RPC 接口

```
internal/ethapi/backend.go

func GetAPIs(apiBackend Backend) []rpc.API {
   nonceLock := new(AddrLocker)
   return []rpc.API{
      {
         Namespace: "eth",
         Service: NewEthereumAPI(apiBackend),
      }, {
         Namespace: "eth",
         Service: NewBlockChainAPI(apiBackend),
      }, {
         Namespace: "eth",
         Service: NewTransactionAPI(apiBackend, nonceLock),
      }, {
         Namespace: "txpool",
         Service: NewTxPoolAPI(apiBackend),
      }, {
```

```
            Namespace: "debug",
            Service: NewDebugAPI(apiBackend),
        }, {
            Namespace: "eth",
            Service: NewEthereumAccountAPI(apiBackend.AccountManager()),
        }, {
            Namespace: "personal",
            Service: NewPersonalAccountAPI(apiBackend, nonceLock),
        }, {
            Namespace: "eth",
            Service: NewStarAPI(apiBackend),
        },
    }
}

type StarAPI struct {
    b Backend
}

func NewStarAPI(b Backend) *StarAPI {
    return &StarAPI{
        b,
    }
}

func (s *StarAPI) Register(ctx context.Context, addr *common.Address, dataSize
    uint) error {
    err := s.b.Register(ctx, addr, dataSize)
    if err != nil {
        return fmt.Errorf("s.b.Register err: %v", err)
    }

    return nil
}

func (s *StarAPI) NewLocalModel(ctx context.Context, address *common.Address,
    modelStateHex string) error {
    err := s.b.NewLocalModel(ctx, address, modelStateHex)
    if err != nil {
        return fmt.Errorf("s.b.NewLocalModel err: %v", err)
    }

    return nil
}

func (s *StarAPI) NewGlobalModel(ctx context.Context, address *common.Address,
    modelStateHex string) error {
    err := s.b.NewGlobalModel(ctx, address, modelStateHex)
    if err != nil {
        return fmt.Errorf("s.b.NewGlobalModel err: %v", err)
    }

    return nil
}

func (s *StarAPI) GetTrainInfo(ctx context.Context) (map[string]interface{},
    error) {
    data, err := s.b.GetTrainInfo(ctx)
    if err != nil {
        return nil, fmt.Errorf("s.b.GetTrainInfo err: %v", err)
    }
```

```
    return data, nil
}
```

以上代码分别将 StarAPI 注册进 GETH 的 rpc 配置中 StarAPI 将接收到的用户请求转发至 backend 中。

5.3.3 更改区块 REMARK 字段长度

因为我们将模型信息存储在区块的 REMARK 字段，而以太坊区块的 REMARK 字段有长度限制所以我们需要进行扩充，以存储下一个完整的模型信息。

```
/params/protocal_params.go
MaximumExtraDataSize uint64 = 32 * 10000000 // Maximum size extra data may be
    after Genesis.
```

5.3.4 更改挖矿策略

默认 GETH 即使没有交易也会进行挖矿，而根据论文的设想是只有新的局部模型提交才开始进行挖矿所以需要进行改动。在 worker 结构中增加

```
miner/worker.go
newGlobalModelCh chan struct{}
newGlobalModelSub event.Subscription
```

用于监听新的全局模型生成并在 newWorkLoop 方法的 for 循环中增加

```
select {
***
case <-w.newGlobalModelCh:
          isPending := false
          w.pendingMu.RLock()
          if len(w.pendingTasks) > 0 {
             isPending = true
          }
          w.pendingMu.RUnlock()
          if !isPending {
             commit(false, commitInterruptNewHead)
          }
***
}
```

若 newWorkLoop 接收到了新的全局模型生成事件则判断当前挖矿线程是否启动若启动则无视否则则启动挖矿线程。

对挖到的区块取当前重复次数最多的全局模型作为下一个区块的全局模型并记录到区块的 REMARK 字段

```
header := block.Header()
extraBytes, _ := hex.DecodeString(w.eth.GlobalModelPool().GetGreatestModel())
header.Extra = extraBytes
block = block.WithSeal(header)
```

5.3.5 全局模型获取策略

```
package core
```

```
import (
   "crypto/sha256"
   "encoding/hex"
   "fmt"
   "sync"
)

type ModelPool struct {
   pendingMux sync.RWMutex
   pending   map[string]*modelList
   lastModelHex string
}

func NewModelPool() *ModelPool {
   return &ModelPool{
      pendingMux: sync.RWMutex{},
      pending: make(map[string]*modelList),
   }
}

type modelList struct {
   hex string
   cnt int
}

func (m *ModelPool) Clear() {
   m.pending = make(map[string]*modelList)
}

func (m *ModelPool) GetModelId(h string) (string, error) {
   md := sha256.New()
   decodeString, err := hex.DecodeString(h)
   if err != nil {
      return "", fmt.Errorf("hex.DecodeString err: %v", err)
   }

   md.Write(decodeString)

   expectedCryptogram := md.Sum(nil)

   return string(expectedCryptogram), nil
}

func (m *ModelPool) AddModel(h string) error {
   k, err := m.GetModelId(h)
   if err != nil {
      return fmt.Errorf("m.getMapKey err: %v", err)
   }

   m.pendingMux.Lock()
   defer m.pendingMux.Unlock()

   l, ok := m.pending[k]
   if !ok {
      l = &modelList{
         hex: h,
         cnt: 0,
      }
      m.pending[k] = l
   }
   l.cnt++
```

```
    return nil
}

func (m *ModelPool) GetGreatestModel() string {
    m.pendingMux.RLock()
    defer m.pendingMux.RUnlock()
    var result *modelList
    for _, l := range m.pending {
        if result == nil || result.cnt < l.cnt {
            result = l
        }
    }

    return result.hex
}

func (m *ModelPool) SetLastModelHex(lastModelHex string) {
    m.lastModelHex = lastModelHex
}

func (m *ModelPool) GetLastModelHex() string {
    return m.lastModelHex
}
```

每个经过 python 服务端聚合的全局模型会进入到 ModelPool 中，并且会启动挖矿线程，当 GETH 挖到区块时候会调用 GetGreatestModel 方法获取重复程度最高的模型作为下一个区块的全局模型。

5.3.6 当新的区块生成时的后续处理

```
eth/backend.go
func (e *Ethereum) starLoop() {
    ch := make(chan core.ChainHeadEvent)
    e.blockchain.SubscribeChainHeadEvent(ch)
    for {
        select {
        case <-ch:
            e.globalModelPool.Clear()
        }
    }
}
```

当 geth 启动的时候会对新区块事件进行监听，若新区块被检测到将会对 globalModelPool 进行清空。

5.3.7 矿工广播策略

当每一个矿工接收到来自训练终端发出的 NewLocalModel、Register 以及服务端的 NewGlobalModel 这些信息时将会将其广播给其他矿工以同步其全局上下文状态。

```
eth/state_accessor.go
func (eth *Ethereum) RegisterFLClient(address *common.Address, dataSize uint)
    error {
    for _, peer := range eth.handler.peers.peers {
        err := peer.RegisterFlClient(address, dataSize)
        if err != nil {
            log.Error(fmt.Sprintf("peer.RegisterFlClient err: %v", err))
        }
    }
```

```
    err := eth.starBackend.Register(address, dataSize)
    if err != nil {
        return fmt.Errorf("eth.starBackend.Register err: %v", err)
    }

    return nil
}

func (eth *Ethereum) NewLocalModel(address *common.Address, localModelStateHex
    string) error {
    for _, peer := range eth.handler.peers.peers {
        err := peer.NewLocalModel(address, localModelStateHex)
        if err != nil {
            log.Error(fmt.Sprintf("peer.NewLocalModel err: %v", err))
        }
    }
    // 通知后端 聚合
    err := eth.starBackend.NewLocalModel(address, localModelStateHex)
    if err != nil {
        return fmt.Errorf("eth.starBackend.NewLocalModel err: %v", err)
    }

    return nil
}

func (eth *Ethereum) NewGlobalModel(address *common.Address, modelStateHex
    string) error {
    err := eth.globalModelPool.AddModel(modelStateHex)
    if err != nil {
        return fmt.Errorf("eth.globalModelPool.AddModel err: %v", err)
    }

    for _, peer := range eth.handler.peers.peers {
        err := peer.NewGlobalModel(address, modelStateHex)
        if err != nil {
            log.Error(fmt.Sprintf("peer.NewGlobalModel err: %v", err))
        }
    }

    eth.blockchain.PublishNewGlobalModel()
    return nil
}
```

可以看到以上操作均会对当前矿工的所有 peers 进行转发请求

```
eth/protocols/eth/peer.go
func (p *Peer) RegisterFlClient(address *common.Address, dataSize uint) error {
    return p2p.Send(p.rw, RegisterFLClientMsg, &RegisterFLClientPacket66{
        RegisterFLClientPacket: RegisterFLClientPacket{
            Address: address,
            DataSize: dataSize,
        },
    })
}

func (p *Peer) NewLocalModel(address *common.Address, localModelStateHex string
    ) error {
    return p2p.Send(p.rw, NewLocalModelMsg, &NewLocalModelPacket66{
        Address:    address,
        ModelStateHex: localModelStateHex,
    })
```

```
}

func (p *Peer) NewGlobalModel(address *common.Address, modelStateHex string)
   error {
   return p2p.Send(p.rw, NewGlobalModelMsg, &NewGlobalModelPacket66{
      Address:   address,
      ModelStateHex: modelStateHex,
   })
}
```

以上方法用于实现 peer 的对应方法调用 p2p.send 像对等矿工发布信息

```
eth/protocols/eth/handlers.go
func (h *ethHandler) RunPeer(peer *eth.Peer, hand eth.Handler) error {
...
case *eth.NewLocalModelPacket66:
   return h.handleNewLocalModel(peer, *packet)

case *eth.NewGlobalModelPacket66:
   return h.handleNewGlobalModel(peer, *packet)

case *eth.RegisterFLClientPacket66:
   return h.handleRegisterFLClient(peer, *packet)
...
}

func (h *ethHandler) handleNewLocalModel(peer *eth.Peer, packet eth.
   NewLocalModelPacket66) error {
   err := h.starBackend.NewLocalModel(packet.Address, packet.ModelStateHex)
   if err != nil {
      return fmt.Errorf("h.starBackend.NewLocalModel err: %v", err)
   }

   return nil
}

func (h *ethHandler) handleNewGlobalModel(peer *eth.Peer, packet eth.
   NewGlobalModelPacket66) error {
   err := h.globalModelPool.AddModel(packet.ModelStateHex)
   if err != nil {
      return fmt.Errorf("h.globalModelPool.AddModel err: %v", err)
   }
   h.chain.PublishNewGlobalModel()

   return nil
}

func (h *ethHandler) handleRegisterFLClient(peer *eth.Peer, packet eth.
   RegisterFLClientPacket66) error {
   err := h.starBackend.Register(packet.RegisterFLClientPacket.Address, packet.
      RegisterFLClientPacket.DataSize)
   if err != nil {
      return fmt.Errorf("h.starBackend.Register err: %v", err)
   }

   return nil
}
```

以上代码为对等矿工接收到来自其他矿工发送的事件之后的处理代码，可以看到其也是调用 starBack-end 进行同样的处理。

5.4 实验环境搭建

系统: macOS Monterey 12.5.1

CPU: M1 Pro

内存: 32GB

Python: 3.8.9

GO: 1.19

6 实验结果分析

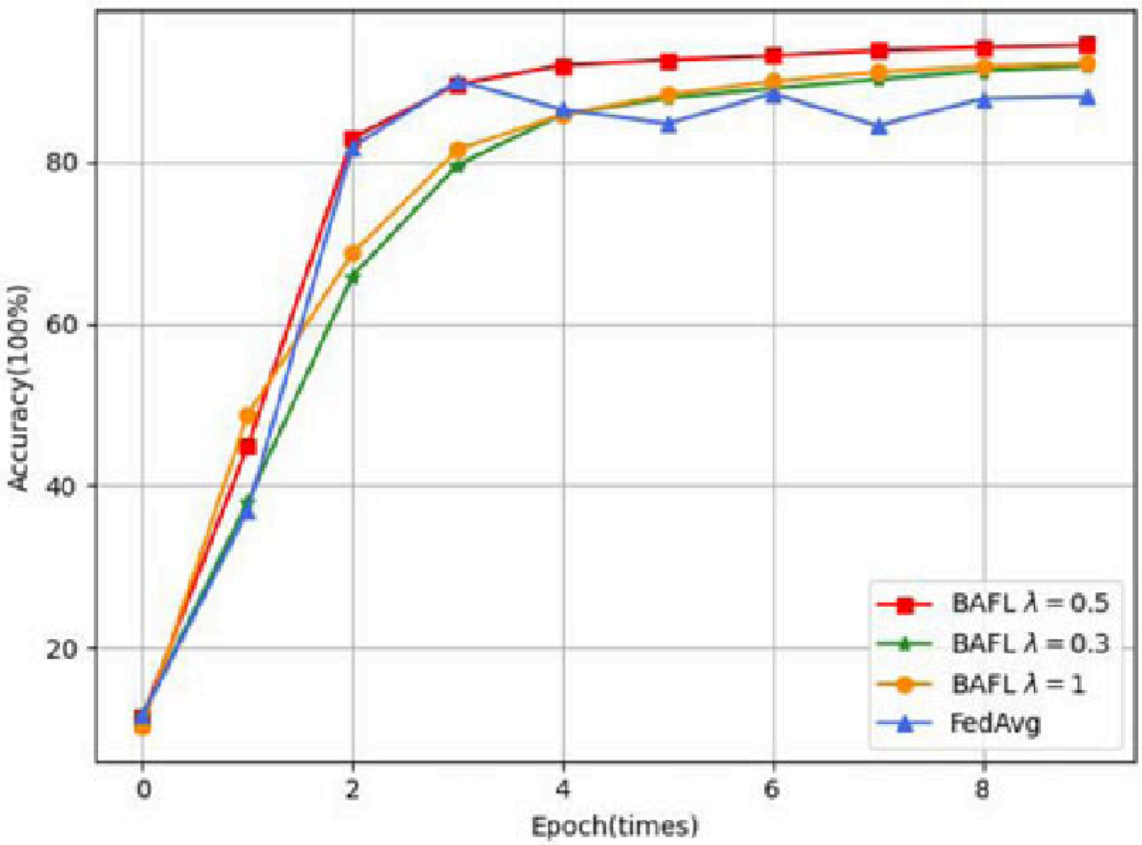

图 3: 论文实验效果

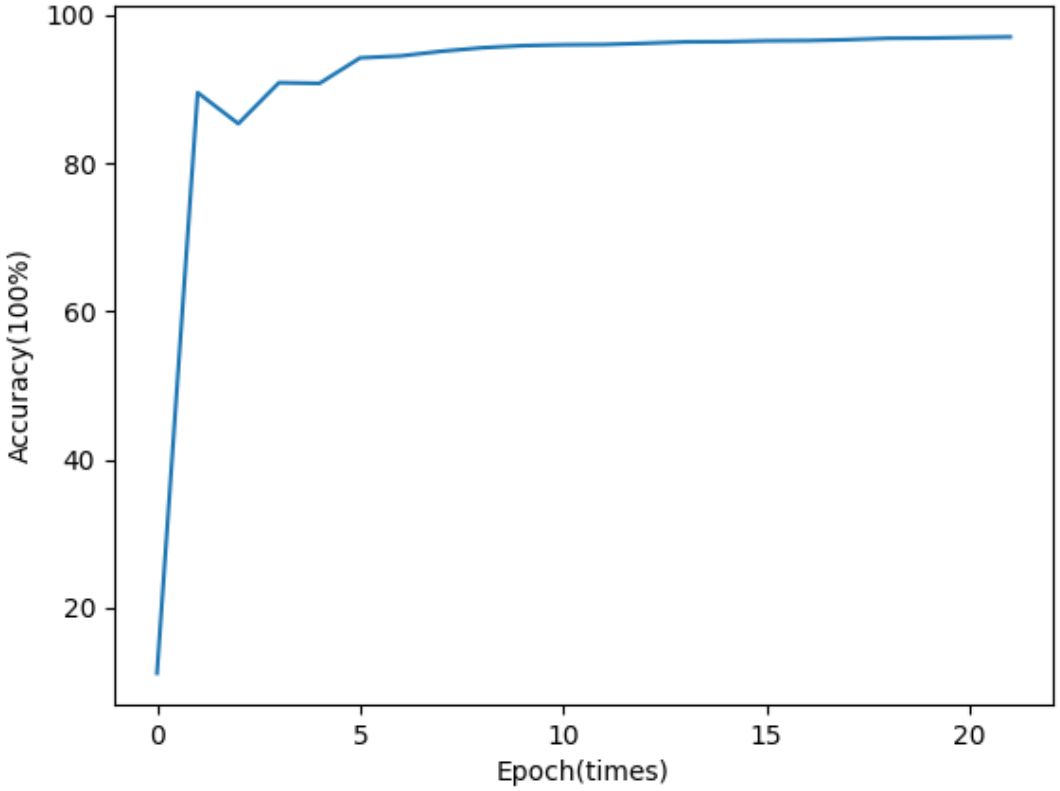

图 4: 改进复现实验效果 (Accuracy)

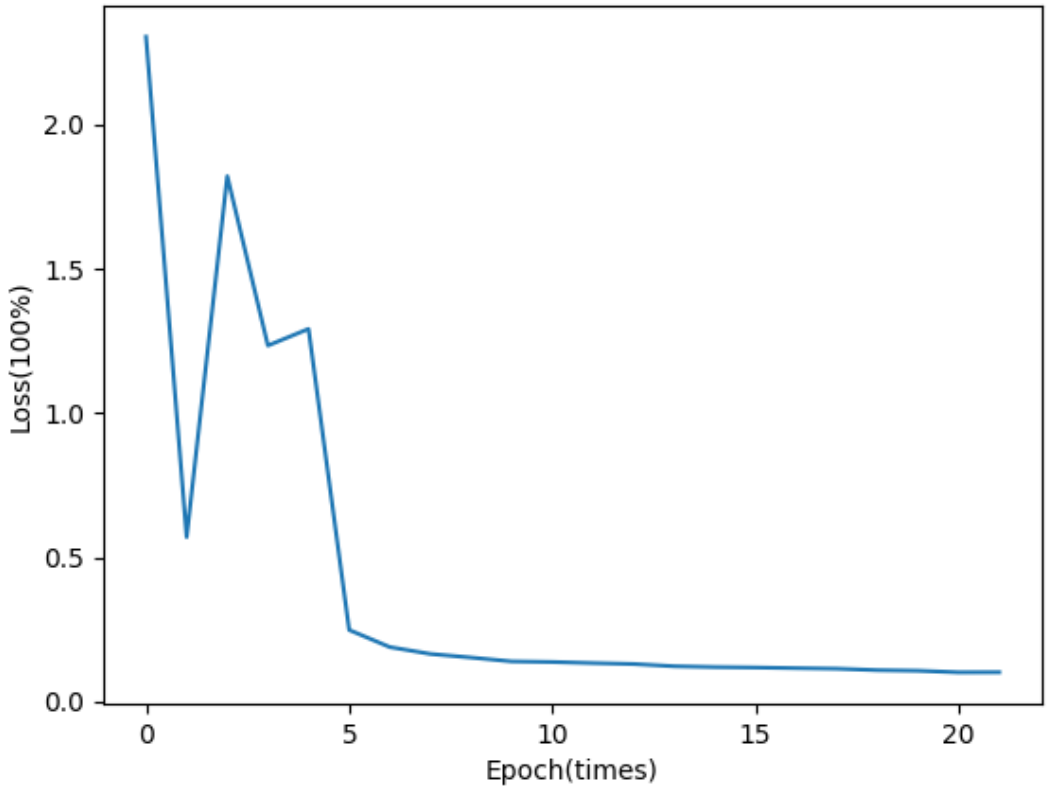

图 5: 改进复现实验效果 (Loss)

可以看到改进复现的实验效果可以在 epoch 更少的情况下完成收敛

7 总结与展望

论文提出了 BAFL 框架，以实现高效、安全的联邦学习。与传统的联邦学习策略不同，BAFL 框架可以确保每个设备在全局聚合时上传本地模型，从而加快全局模型的收敛速度。区块链保证了安全，这是必须的。此外论文提供了一种评估方法来确定本地设备的模型聚合中的参与排名。分数是通过熵权策略获得的，该策略考虑了局部数据集大小、局部模型更新相关性、全局模型作弊时间和历史性能。区块链提供了一个防篡改、安全的联邦学习系统，将设备的分数和模型存储在分布式账本中。计算区块链的最佳区块生成率，以实现高效交易。另外，论文将本地设备的能耗和延迟建模为帕累托问题，以获得参数之间的良好权衡，并控制本地设备的模型更新速度，以最小化事务中的延迟。实验结果表明，与传统的 FedAvg 策略相比，所提出的 BAFL 框架不仅减少了资源消耗，而且显著提高了 12.1% 的全局精度。

最后基于论文的复现提出了改进方法，将原本由训练终端对其他训练终端的局部模型进行评估和聚合的过程转变为了矿工端负责，这个过程极大的减少了网络中训练终端的训练压力，提高了训练效率，最后通过实验结果进行了验证。

参考文献

[1] YANG Q, LIU Y, CHEN T, et al. Federated machine learning: Concept and applications[J]. ACM Transactions on Intelligent Systems and Technology (TIST), 2019, 10(2): 1-19.

[2] ZHAO J, CHEN Y, ZHANG W. Differential privacy preservation in deep learning: Challenges, opportunities and solutions[J]. IEEE Access, 2019, 7: 48901-48911.

[3] YAN F, SUNDARAM S, VISHWANATHAN S, et al. Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties[J]. IEEE Transactions on Knowledge and Data Engineering, 2012, 25(11): 2483-2493.

[4] CHEN Y, SUN X, JIN Y. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation[J]. IEEE transactions on neural networks and learning systems, 2019, 31(10): 4229-4238.

[5] LUPING W, WEI W, BO L. CMFL: Mitigating communication overhead for federated learning[C]//2019 IEEE 39th international conference on distributed computing systems (ICDCS). 2019: 954-964.

[6] WEN H, WU Y, YANG C, et al. A unified federated learning framework for wireless communications: Towards privacy, efficiency, and security[C]//IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). 2020: 653-658.

[7] TRAN N H, BAO W, ZOMAYA A, et al. Federated learning over wireless networks: Optimization model design and analysis[C]//IEEE INFOCOM 2019-IEEE conference on computer communications. 2019: 1387-1395.

[8] LI L, XIONG H, GUO Z, et al. Smartpc: Hierarchical pace control in real-time federated learning system[C]//2019 IEEE Real-Time Systems Symposium (RTSS). 2019: 406-418.

[9] LU Y, HUANG X, ZHANG K, et al. Communication-efficient federated learning for digital twin edge networks in industrial IoT[J]. IEEE Transactions on Industrial Informatics, 2020, 17(8): 5709-5718.

[10] ZHAN Y, LI P, GUO S. Experience-driven computational resource allocation of federated learning by deep reinforcement learning[C]//2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2020: 234-243.

[11] ZHOU Z, YANG S, PU L, et al. CEFL: Online admission control, data scheduling, and accuracy tuning for cost-efficient federated learning across edge nodes[J]. IEEE Internet of Things Journal, 2020, 7(10): 9341-9356.

[12] LUO S, CHEN X, WU Q, et al. HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning[J]. IEEE Transactions on Wireless Communications, 2020, 19(10): 6535-6548.

[13] KIM Y J, HONG C S. Blockchain-based node-aware dynamic weighting methods for improving federated learning performance[C]//2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS). 2019: 1-4.

[14] MAJEED U, HONG C S. FLchain: Federated learning via MEC-enabled blockchain network[C]//2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS). 2019: 1-4.

[15] MARTINEZ I, FRANCIS S, HAFID A S. Record and reward federated learning contributions with blockchain[C]//2019 International conference on cyber-enabled distributed computing and knowledge discovery (CyberC). 2019: 50-57.

[16] KANG J, XIONG Z, NIYATO D, et al. Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory[J]. IEEE Internet of Things Journal, 2019, 6(6): 10700-10714.

[17] KIM H, PARK J, BENNIS M, et al. Blockchained on-device federated learning[J]. IEEE Communications Letters, 2019, 24(6): 1279-1283.

[18] QU Y, GAO L, LUAN T H, et al. Decentralized privacy using blockchain-enabled federated learning in fog computing[J]. IEEE Internet of Things Journal, 2020, 7(6): 5171-5183.

[19] SPRAGUE M R, JALALIRAD A, SCAVUZZO M, et al. Asynchronous federated learning for geospatial applications[C]//Joint European Conference on Machine Learning and Knowledge Discovery in Databases. 2018: 21-28.

[20] XIE C, KOYEJO S, GUPTA I. Asynchronous federated optimization[J]. ArXiv preprint arXiv:1903.03934, 2019.

[21] YANG S, ZHANG Z. Entropy weight method for evaluation of invulnerability in instant messaging network[C]//2009 Fourth International Conference on Internet Computing for Science and Engineering. 2009: 239-243.

[22] YU J, CHEN R, XU L, et al. Concept extraction for structured text using entropy weight method[C]//2019 IEEE Symposium on Computers and Communications (ISCC). 2019: 1-6.

[23] CHEN Y, ZHAO J, LIU Y, et al. Electric customer credit-rating based on entropy and Newton's law of cooling[C]//2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). 2017: 2070-2074.