arXiv:2303.01778v1 [cs.LG] 3 Mar 2023

FEDML PARROT：通过顺序训练和分层训练中的异构性感知调度实现的可扩展联邦学习系统

Zhenheng Tang [1] **Xiaowen Chu** [1] **Ryan Yide Ran** [2] **Sunwoo Lee** [3] **Shaohuai Shi** [4] **Yonggang Zhang** [1] **Yuxin Wang** [1] **Alex Qiaozhong Liang** [2] **Salman Avestimehr** [2] **Chaoyang He** [2]

摘要

联邦学习（FL）支持客户端之间协作训练机器学习模型，同时保护其数据隐私。现有联邦学习模拟平台从传统分布式训练的角度设计，存在模拟与生产环境之间代码迁移繁琐、效率低、GPU 利用率低、可扩展性差且硬件要求高，以及难以模拟有状态客户端等问题。在本工作中，我们首先阐明了模拟联邦学习所面临的挑战和瓶颈，并设计了一个新的联邦学习系统，命名为 FedML `Parrot`。该系统通过以下方式提升训练效率、显著降低硬件要求，并支持使用有状态客户端开展高效的大规模联邦学习实验：（1）在设备上按顺序训练客户端；（2）将原始聚合分解为分别在设备和服务器上执行的本地聚合与全局聚合；（3）调度任务以缓解掉队问题并提升计算资源利用率；（4）采用分布式客户端状态管理器以支持各种联邦学习算法。此外，基于我们通用的 API 和通信接口，用户无需修改代码即可将模拟无缝转换为真实世界部署。我们通过`Parrot` 在各种联邦学习数据集上训练多种模型的大量实验进行评估，结果表明 `Parrot` 能够在灵活设置 GPU 设备（$4 \sim 32$）和较高 GPU 利用率的情况下，模拟超过 1000 个客户端（有状态或无状态），速度比 FedScale 快 1。$2 \sim 4$ 倍，内存节省达到 FedML 的 $10 \sim 100$ 倍。此外，我们验证了 `Parrot`在三个不同集群中的同构和异构设备上均能良好运行。我们模拟了两种使用有状态客户端的联邦学习算法和四种使用无状态客户端的算法，以验证 `Parrot` 对不同算法的广泛适应性。代码将合并至`https://github.com/FedML-AI/FedML`。

1 引言

联邦学习（FL）是一种分布式机器学习范式，能够在保护数据隐私的前提下，实现多个客户端之间的协同训练（McMahan 等，2017；Kairouz 等，2021）。它支持对隐私问题敏感的组织和个人用户在不相互共享数据的情况下训练出强大的模型。许多应用已从拥有大量客户端的联邦学习中受益，例如词语预测（Hard 等人，2018）、目标检测（Liu 等人，2020；Hsu 等人，2020）和医疗保健（Rieke 等人，2020；Dayan 等人，2020）。与数据中心中的分布式训练（Li 等人，2014）不同，联邦学习中的客户端数量从 $2 \sim 10^{10}$ 不等（Hard 等人，2018；Kairouz 等，2021），这使得联邦学习的扩展极具挑战性。

[1]香港浸会大学 [2]FedML 公司 [3]仁荷大学 [4]香港科技大学。通信作者：Chaoyang He <ch@fedml.ai>。

本研究是在 *FedML* 公司实习期间完成的。

当前联邦学习社区在很大程度上忽视了一个重要的研究领域，具体如下：在真实世界部署之前，新的联邦学习模型和算法通常需要通过模拟进行验证。但是，如何才能在固定的小规模 GPU 集群上，快速而灵活地构建原型，以模拟任意数量的客户端（即大规模场景），同时确保无需更改任何代码即可实现生产部署，并避免繁琐且容易出错的代码迁移?

存在许多联邦学习框架，支持在 GPU/CPU 计算资源上运行不同的联邦学习算法（Caldas 等人，2018；Li 等人，2021b；Lai 等人，2021；Beutel 等人，2020）。然而，这些框架远不能满足上述研究探索需求。它们要么要求设备数量与客户端数量相同，要么仅使用单个设备来模拟所有客户端（Caldas 等人，2018；Li 等人，2021b）。这两种方式都无法支持联邦学习算法的大规模训练。为降低对硬件的高要求，一些联邦学习框架（Lai 等人，2021；Beutel 等人，2020）

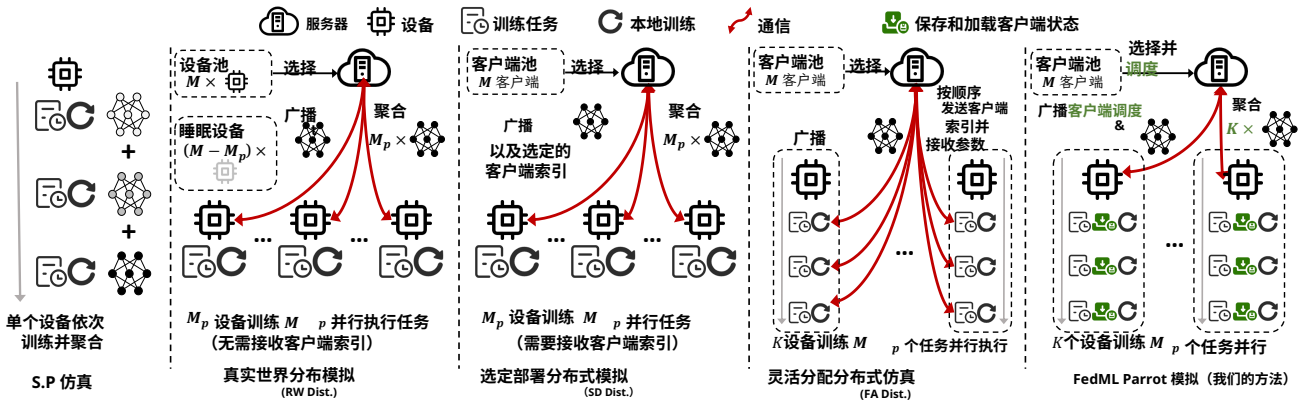


图*1*。 不同联邦学习模拟系统概览。

支持在每个训练轮次中让多个客户端运行于单个执行器上，并且执行器可以在不同轮次中训练来自不同客户端的模型，例如使用 Docker 进行模拟（Xu 等人，2017）。然而，这些系统仍然存在一些主要问题。

首先，客户端与服务器之间的通信开销非常显著，导致计算资源的利用率极低。其次，客户端通常具有很高的数据异质性（例如，数据规模不同），因此不同客户端的训练时间差异很大。这是一个经典的掉队者问题：服务器必须等待最慢的执行器完成其任务，从而导致训练时间过长以及 GPU/张量处理器利用率极低。

第三，这些框架忽略了某些联邦学习算法中的有状态客户端，在这些算法中，执行器需要存储客户端的历史状态，以便每个执行器能够在不同轮次运行不同的客户端。许多联邦学习变体算法（Li 等，2020；Karimireddy 等，2020b；Wang 等，2020；Acar 等，2021；Luo 等，2021；Li 等，2021c；Chen和Chao，2021年；Collins 等，2021）旨在解决数据异质性问题，因为客户端通常具有不同的数据分布和/或不同的数据规模，这使得简单的联邦学习算法（如 FedAvg）难以收敛，并导致较差的泛化性能（Woodworth 等，2020；Acar 等，2021）。这些算法在训练期间交换的内容可能不限于模型参数，还可能包括其他参数，例如中间特征（Collins 等，2021）、模型掩码（Li 等，2021a）、辅助梯度校正（Karimireddy 等，2020b）、第三方数据集（Lin 等，2020；Tang 等，2022）等。此外，许多联邦学习算法要求有状态客户端存储某些客户端状态，例如控制变量（Karimireddy 等，2020b）、旧梯度（Acar 等，2021）、个性化模型或层（Liang 等，2020；Chen和Chao，2021年）、模型掩码（Li 等，2021a）等。

第四，目前社区中的迁移成本很高，需要研究团队和产品团队双方协作；模拟无法快速发现部署中的问题；代码不一致中的一个小错误就会导致训练算法出错，最终导致训练有效模型失败。

为了支持大规模联邦学习实验，并灵活扩展和设计算法，亟需一个通用的联邦学习仿真系统。为解决上述问题，我们在本文中作出以下贡献。

贡献。 我们首先系统地揭示了模拟联邦学习算法所面临的挑战和瓶颈（§2）。随后，我们提出了一个名为 `Parrot` （§3§4）的高效且灵活的新型联邦学习系统，具备以下四项特性。1）我们设计了一种分层数据聚合机制：首先在设备上对客户端结果进行本地聚合，然后在服务器上进行全局聚合，从而显著降低通信复杂度。2）我们设计了一个任务调度器，通过估计动态实时工作负载来最小化一轮训练时间，从而缓解掉队者问题并提高 GPU 利用率。3）我们设计了一个客户端状态管理器，以支持有状态客户端算法。4）`Parrot` 提供通用 API 和抽象通信层，使用户能够在模拟中轻松扩展新算法。模拟中经过充分验证的算法和模型可以无缝部署到真实世界的联邦学习环境中。

我们通过广泛的实验验证了`Parrot`的灵活性和效率，包括在不同数据集（FEMNIST（Caldas 等人，2018）、ImageNet（Russakovsky 等人，2015）和 Reddit（Lai 等人，2021）上使用不同数量的设备和客户端（即 1000 和 10000 个客户端）训练多种模型（ResNet18、ResNet50（He 等人，2016）以及 Albert（Lan 等人，2020））。在三个具有同质和异质设备的不同集群中进行的实验验证了`Parrot` 能够良好地适应不同的硬件环境。具体而言，`Parrot` 成功模拟了两种使用有状态客户端的联邦学习算法和四种使用无状态客户端的算法，展现了 `Parrot` 对不同联邦学习算法的高灵活性支持。实验结果还表明，Parrot 实现了近乎

在线性加速，并且随着不同硬件配置下设备数量的增加，

2 背景与挑战

2.1 联邦学习

FedAvg 是联邦学习中一种非常通用的学习算法，它会在多个训练轮次中迭代更新模型参数，直至收敛。每一轮中，中心服务器首先随机选择部分客户端（称为并发客户端），并向这些客户端广播全局参数。随后，选定客户端使用其私有数据进行模型训练，并将更新后的本地参数发送给服务器。之后，服务器聚合本地参数以获得新的全局参数，然后进入下一训练轮次。

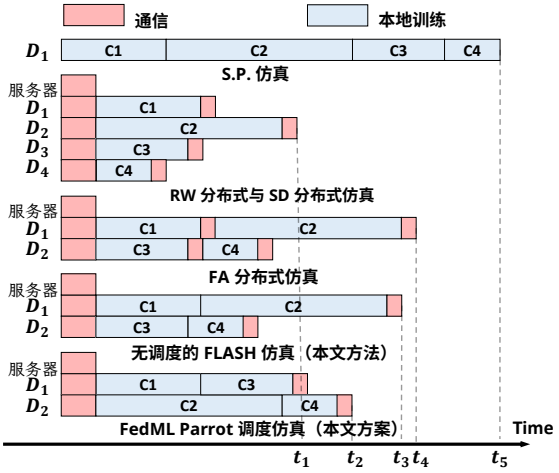


图2. 不同仿真方案的时间线，其中 D_i 表示第 i 个设备。

2.2 联邦学习仿真框架

现有最先进的联邦学习仿真框架（Caldas 等人，2018；Ingerman& Ostrowski，2019；He 等人，2020；Ryffel 等人，2018；Lai 等人，2021；Beutel 等人，2020）可分为两种类型：单进程仿真和分布式仿真。

单进程仿真（SP）。不同客户端的训练任务在 SP 中依次运行于一台计算设备上。代表性系统包括 LEAF（Caldas 等人，2018）和 TFF（Ingerman & Ostrowski，2019）。由于所有模型都存储在单台设备上，如图 2 所示，SP 可以避免在聚合模型时进行通信。然而，联邦学习通常需要模拟许多拥有异构数据集的客户端，因此 SP 中的单台设备需要交替为不同客户端训练模型，这非常耗时。

分布式仿真。分布式仿真还可进一步分为真实世界分布式仿真（RW Dist.）、选定部署分布式仿真（SD Dist.）和灵活分配分布式仿真（FA Dist.）。

在 RW Dist. 中，所有客户端（例如 M 个客户端）都应初始化于 M 台设备上。在训练过程中，它遵循 FedAvg协议。代表性系统包括 Syft（Ryffel，2018）和 FederatedScope（Xie，2022年））。在每个通信轮次中，M_p 个选定客户端或设备进行训练，而其他 $M - M_p$ 台未选定设备处于空闲状态。这会严重降低计算资源的利用率；当扩展到大量客户端时，成本也会变得极高，例如 1000 或 10000 个客户端。

在SD Dist.中，每轮所需的设备数量只需与选定客户端的数量相同，即M_p。每一轮中，每台设备都会收到服务器分配的客户端索引。随后，设备将模拟分配给自己的客户端。这避免了设备闲置导致 CPU 或 GPU 利用率极低的问题。如果设备同质，RW Dist. 和SD Dist. 的时间线如图2 所示，并且在相同环境下，其训练时间在所有方案中最短。然而，由于不同客户端之间存在数据异质性，SD Dist. 和RW Dist. 存在掉队者问题。

FA Dist. 可视为SP 与分布式模拟的结合。代表性系统包括 FedScale（Lai 等人，2021）和 Flower（Beutel 等人，2020）。FADist. 支持使用多台设备进行模拟，并允许设备数量 K小于选定客户端数量 M_p。每台设备会依次模拟多个客户端。模拟完一个客户端后，每台设备会立即将结果返回给服务器。随后，服务器会将当前轮次中需要模拟的新客户端索引分配给该设备。FA Dist. 显著降低了联邦学习模拟的硬件要求。少量 CPU 或 GPU 即可模拟大规模联邦学习。然而，如图2所示，掉队者问题、较大的通信开销以及有状态客户端问题仍然存在。

从模拟到生产部署。许多联邦学习框架旨在集中式计算集群中模拟联邦学习（Caldas 等人，2018；Inger man 和 Os- trowski，2019；Lai 等人，2021；Beutel 等人，2020）。底层通信 API 基于 MPI（Thakur 等人，2005）或 PyTorch-DDP（Paszke 等人，2019）。将模拟中实现的联邦学习算法部署到真实世界中并不容易，可能需要耗费大量精力进行代码迁移。

我们在表中总结了不同方案模拟一个联邦学习轮次的系统开销1 ，涉及设备数量、CPU 或 GPU 内存、磁盘存储、通信数据量和通信往返次数。请注意，我们框架中的客户端状态管理器可以作为辅助

表1。 不同仿真方案的复杂度比较。请注意，**客户端状态管理器**不仅可用于Parrot，也可用于其他方案。 M 表示客户端总数， M_p 表示每轮选定客户端的数量。在RW Dist.和SD Dist.中，每轮参与训练的设备数量为 M_p；而 K 在FA Dist.和FedML Parrot ($K \leq M_p$)中。 s_m 表示模拟单个客户端所需的内存大小。 s_a 表示在服务器上进行平均的参数的平均大小。 s_e 表示服务器应收集而非进行平均的特殊参数大小，这仅适用于部分算法。 s_d 表示客户端状态的大小，这仅适用于采用有状态客户端的算法。

方案	SP	RW Dist.	SD Dist.	FA Dist.	FedML Parrot
设备数（执行器）	1	M	M_p	K	K
内存	$O(s_m M + s_d M)$	$O(s_m M + s_d M)$	$O(s_m M_p + s_d M)$	$O(s_m K + s_d M)$	$O(s_m K + s_d M)$
内存（具有**状态管理器**	$O(s_m + s_d)$	$O(s_m M + s_d M_p)$	$O(s_m M_p + s_d M_p)$	$O(s_m K + s_d K)$	$O(s_m K + s_d K)$
磁盘成本与**状态管理器**	$O(s_m M_p + s_d M)$	$O(s_d M)$	$O(s_d M)$	$O(s_d M)$	$O(s_d M)$
通信大小	$O(0)$	$O\left(s_a M_p + s_e M_p\right)$	$O\left(s_a M_p + s_e M_p\right)$	$O\left(s_a M_p + s_e M_p\right)$	$O\left(s_a K + s_e M_p\right)$
通信轮次	$O(0)$	$O(M_p)$	$O(M_p)$	$O(M_p)$	$O(K)$

表 2 . 不同联邦学习框架的比较 架。

	LEAF	TFF	Syft	FedScale	Flower	FederatedScope	FedML 跨机构	FedML Parrot
SP	✓	✓	✓	✓	✓	✓	✓	✓
RW Dist.				✓		✓	✓	✓
SD Dist.					✓		✓	✓
FA Dist.				✓	✓			✓
可扩展性		✓		✓	✓	✓	✓	✓
灵活硬件配置	✓			✓	✓			✓
真实世界部署				✓	✓	✓	✓	✓
任务调度								✓
客户端状态管理器								✓

作为增强其他方案的组件。通信往返指设备与服务器之间的通信轮次数，这是影响通信时间的重要因素（Sarvotham 等人，2001；Shi 等人，2019；Huba 等人，2022年）。我们在第4.5节提供了详细的复杂度分析。

2.3 挑战

Parrot 旨在为用户提供高效的联邦学习系统，以实现：(1) 显著加速模拟；(2) 降低大规模联邦学习实验模拟对大量边缘设备的需求，例如 $1000 \sim 10000$ 客户端；(3) 支持广泛的联邦学习算法，包括有状态联邦优化器；(4) 提供从研究模拟到真实世界联邦学习部署的无缝迁移，从而在统一框架中应对以下四项挑战。

• **异构数据**。客户端具有不同的数据分布和数据集大小（Kairouz 等人，2021）。这会导致掉队者问题，即服务器必须等待最慢的客户端。在传统分布式训练中也会出现这一问题（Zhang 等人，2016），但仅发生在设备具有异构性的情况下。

• **在小型集群上模拟海量客户端**。联邦学习中客户端的数量从 2 个到超过 10_{10} (Kairouz 等人，2021）。为了实现快速、灵活的原型开发，联邦学习社区缺乏一种成熟的系统设计，无法将任意数量的客户端扩展到固定的小型 GPU 集群。

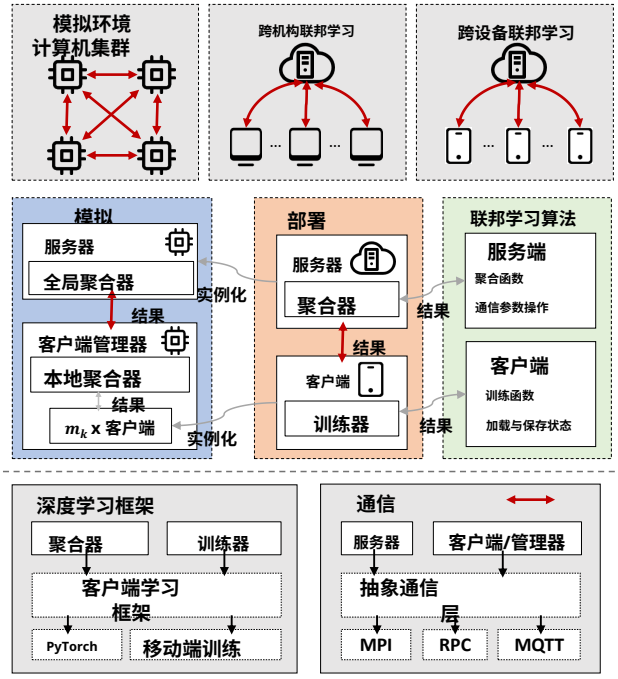
• **部分参与和有状态客户端**。每一轮中，只需模拟选定客户端中的一部分。通常会利用这一特性来降低硬件要求：执行器只需加载全局参数，即可模拟选定客户端。然而，当客户端是有状态的时，这一优势便会受到限制。具体而言，要求每台设备存储所有历史客户端状态，并将其传输给其他设备以获取客户端状态并不现实，因为这需要占用大量额外内存并带来巨大的通信开销。

• **从模拟迁移到生产环境的复杂性**。 在当前社区中，迁移成本很高，需要研究团队和产品团队双方的协作；模拟无法快速发现部署中的问题；代码不一致中的一个小错误就会导致训练算法出错，并最终导致无法训练出有效模型。

3 Parrot 系统设计

3.1 总体架构

联邦学习算法可以概括为算法1。具体而言，在几乎所有联邦学习算法中，服务器都需要向客户端传递一些全局参数 Θ^r，并收集客户端结果 $\{\mathcal{C}^r_{m,E-1} | m \in \mathcal{M}^r\}$，然后执行一些更新。请注意， Θ^r 不仅可以是机器学习



*Fig*图 *3* . FedML Pa 系统设计概览 rrot。

模型参数，也可以包括联邦学习算法所需的其他参数。$\mathcal{C}^r_{m,\ j}$ 表示将返回给服务器的通用客户端结果。$\mathcal{S}^r_{m,\ j}$ 表示某些联邦学习算法所需的客户端状态。与传统的分布式训练方案不同，我们希望允许设备在每个通信轮次中执行多个任务，从而避免将每个客户端部署到单台设备上所需的高硬件要求。

算法2和图 3 展示了我们的整体系统设计。算法 客户端执行 在算法 1 中被视为算法2中的一项任务。在每一轮中，服务器管理器从原始联邦学习服务器获取选定客户端，然后进行调度，将任务合理分配给设备（详见第4.4节）。设备收到分配的任务后，执行任务、在本地聚合客户端结果 $\mathcal{C}^r_{m,\ j}$ 并保存客户端状态 $\mathcal{S}^r_{m,\ j}$ 逐一完成。所有任务完成后，设备将预处理后的客户端结果 $\mathcal{G}^r_k$ 返回给服务器。服务器对所有 $\{\mathcal{G}^r_k|k\in\mathcal{K}\}$ 进行全局聚合。该本地-全局聚合方案能够保证获得与原始联邦学习聚合相同的结果，同时降低通信开销。这种本地-全局分层聚合将在第 4.2节具体介绍。

3.2 轻松迁移：从模拟到部署

为实现便捷的联邦学习部署，我们设计了通用 API接口，以适应模拟和真实世界的分布式部署。设备执行和通信的细节对用户隐藏，使用户能够

算法 1联邦学习的一般流程。

服务器输入： 初始 Θ^0，最大通信轮次 R
客户端 m**的输入：** 本地训练轮数 E
服务器执行：
对于每一轮 $r=0\ ,1,\cdots,R$ **do**
服务器选择一组客户端 $\mathcal{M}^r\subseteq\{1,...,M\}$。
服务器**通信** Θ^r 并向所有客户端 $m\in\mathcal{M}^r$。
对每个客户端 $m\in\mathcal{M}^r$ **并行执行**
$\mathcal{C}^{r+1}_{m,\ E-1}\leftarrow$ 客户端执行（$m,\ \Theta^r$）。
结束循环
$\Theta^{r+1}\leftarrow$ 聚合（$\Theta^r,\ \{\mathcal{C}^{r+1}_{m,E-1}|m\in\mathcal{M}^r\}$）。
结束循环

客户端执行($m,\ \Theta^r$**):**
对每个本地步骤 j 使用 $j=0,\ \cdots\ ,E-1$ **执行**
采样原始数据 $\xi_{m,\ j}\sim\mathcal{D}_m$
$\mathcal{C}^{r+1}_{m,\ j},\ \mathcal{S}^{r+1}_{m,\ j}\leftarrow$ 训练 $(m,\ \mathcal{C}^r_{\xi_m,\ j},\ \mathcal{S}^r_{m,\ j},\ \Theta^r)$
结束循环
返回 $\mathcal{C}^{r+1}_{m,E-1}$ 到服务器。

算法2 Parrot 模拟方案。

服务器输入： 初始 Θ^0，最大通信轮次 R
客户端 m **的输入：** 本地训练轮数 E
服务器执行：
对于每一轮 $r=0\ ,1,\cdots,R$ **do**
服务器选择一组客户端 $\mathcal{M}^r\subseteq\{1,...,M\}$。
$\{\mathcal{M}^r_k|k\in\mathcal{K}\}\leftarrow$ **任务调度**$(r,\ \mathcal{M}^r,\ \mathcal{T}^r)$。
服务器**向** Θ^r 每台设备 $k\in\mathcal{K}$**发送** $\mathcal{M}^r_k$。
并行地对每台设备 $k\in\mathcal{K}$ **执行 do do**
$\mathcal{G}^{r+1}_k\leftarrow$ 设备执行$(\mathcal{M}^r_k,\ \Theta^r)$。
结束循环
$\Theta^{r+1}\leftarrow$ 全局聚合$(\Theta^r,\{\mathcal{G}^{r+1}_k|k\in\mathcal{K}\})$.
结束循环

设备执行$(\mathcal{M}^r_k,\Theta^r)$:
对每个 $m\in\mathcal{M}^r_k$ **执行**
$\mathcal{S}^r_{m,0}\leftarrow$加载状态$(m)$.
$\mathcal{C}^{r+1}_{m\ ,E-1},\mathcal{S}^{r+1}_{m\ ,E-1}\leftarrow$ 客户端执行$(m,\Theta^r,\mathcal{S}^r_{m,0})$。
保存状态$(m,\mathcal{S}^{r+1}_{m\ ,E-1})$。
结束循环
$\mathcal{G}^{r+1}_k\leftarrow$ 本地聚合$(\mathcal{C}^{r+1}_{m\ ,E-1})$ **返回** $\mathcal{G}^{r+1}_k$ 至服务器。

专注于联邦学习算法和模型的设计。

如图3所示，用户在 `Parrot` 中实现联邦学习算法时，只需额外完成以下工作：(1) 指定通信参数的操作（OP），例如加权平均、求和、简单平均或不经平均直接收集。`Parrot` 将根据定义的 OP 执行聚合；(2) 指定需要存储的参数（如有需要，即有状态客户端）。用户还可以自定义服务器端和客户端函数，例如客户端选择、本地训练和服务器更新等。随后，`Parrot` 即可自动将客户端任务调度至设备并执行。

3.3 顺序分布式训练

训练每个已分配的客户端时，设备会先从状态管理器加载客户端状态，并将当前客户端的参数设为初始状态。随后，设备根据用户定义的函数执行本地训练。训练完成后，返回的本地结果$\mathcal{C}^r_{m,\ k}$将按照用户定义的 OP 在本地进行聚合。然后，设备 k 的新结果 $\mathcal{G}^r_k$ 将返回服务器，以便进一步聚合。

3.4 有状态联邦学习客户端的状态管理

当算法包含有状态客户端时，模拟大规模联邦学习需要存储大量客户端状态。假设客户端状态的大小为 s_d，则客户端状态总共占用 $s_d M$ 内存。由于无法将其加载到 CPU 或 GPU 内存中，因此我们构建了一个状态管理器，用于自动从磁盘加载客户端状态并将其保存到磁盘。当设备开始模拟客户端 k 时，可以从磁盘加载相应的客户端状态。训练完成后，该客户端会更新状态，状态管理器则会自动将其保存到磁盘。

4 Parrot 系统优化

本节将介绍 Parrot 的系统性能优化。我们的系统优化旨在减少通信开销，并通过调度客户端来最大限度地缩短总训练时间。

4.1 调度的必要性

在联邦学习中，客户端的数据集大小存在很大差异，因此不同客户端的训练时间也大不相同。这会导致众所周知的掉队者问题：服务器必须等待速度最慢的执行器完成其任务，从而导致训练时间过长以及 GPU 利用率极低。此外，训练设备可能并不具有同质性。用户可能使用配备不同 CPU 或 GPU 的机器。或者，设备可能还被用于执行其他任务，导致计算性能不稳定。这些因素都可能引发掉队者问题。

4.2 分层聚合

许多联邦学习算法会对来自 M_p客户端的参数、梯度或机器学习模型差异进行平均（McMahan 等人，2017；Li 等人，2020；Wang 等人，2020；Reddi 等人，2021；Acar 等人，2021；Chen & Chao，2021）。假设需要进行平均的参数（平均参数）的大小为 s_a，则一轮中的总通信大小为 $s_a M_p$。服务器需要对平均参数进行 $M_p - 1$ 次求和。

我们注意到，这一平均过程可以分解为本地平均和全局平均，从而减少通信数据量和服务器的聚合成本。接着，我们将联邦学习服务器上的聚合操作改为本地-全局分层聚合方案。具体而言，设备将在本地对 $|\mathcal{M}^r_k|$平均参数求平均，服务器则对来自设备的 K 结果进行全局聚合。因此，通信成本从 $s_a M_p$ 降至 $s_a K$，通信往返次数从 M_p 减少至 K，服务器只需执行 $K-1$ 次求和操作。

然而，某些算法（Li 等人，2021a；Collins 等人，2021；Lin 等人，2020）需要通信一些应在服务器端收集但不进行平均的参数（特殊参数）。对于大小为 s_e的这些参数，设备会将它们封装成一条消息并发送给服务器。服务器以与联邦学习服务器相同的方式收集这些参数。在这种情况下，我们只能减少通信往返次数，而不能减少总通信数据量 $s_e M_p$。幸运的是，包含大量特殊参数的联邦学习算法并不多，因为服务器需要额外的 $s_e M_p$ 或 $s_e M$ 内存来存储和使用这些参数。当 M_p 和 M 较大时，会给服务器带来过多的系统开销。

4.3 量化每个客户端的运行时间

掉队者问题，即等待速度最慢的设备，会严重降低系统效率。因此，我们希望在设备之间平衡工作负载（训练时间），尽量使设备几乎同时完成各自的任务。为此，首先应准确估计设备上各项任务的工作负载。

工作负载模型。由于我们希望平衡各设备的运行时间，因此将设备 k 上任务 π_m 的工作负载定义为 $T_{m,\ k}$。客户端 k 上的本地训练主要包括以下几个部分：(1) 加载数据集；(2) 前向传播；(3) 反向传播；(4) 更新模型。总运行时间主要由 (2) 和 (3) 决定，这两部分通常会随着数据集大小近乎线性地增长。因此，我们建立了一个简单的线性模型来估算运行时间，该模型广泛用于分布式训练：

$$T_m = N_m * t^{sample} + b, \tag{1}$$

其中，N_m 表示客户端 m上数据集的大小 $\mathcal{D}_m$，t^{sample} 表示处理一个数据样本所需的时间，b 表示设备 k执行一个任务所需的固定时间。

t^{sample} 包括每个样本在 (1)~(4) 中的所有处理时间。b 则是固定的，因为某些操作的耗时与数据集无关，例如在 CPU 内存和 GPU 内存之间移动模型、将客户端模型参数设为服务器模型参数等。

异构硬件资源。 考虑到计算设备的异构性，处理时间

算法 3任务调度。

输入： 当前轮次 r，选定客户端 $\mathcal{M}^r$，历史运行时间 $\mathcal{T}^r$

如果 $r \leq R_w$ **则**
  将 $\mathcal{M}^r$ 均匀划分为 $\{\mathcal{M}_k^r | k \in \mathcal{K}\}$，使其具有相似的 $|\mathcal{M}_k^r|$
else
  t_k^{sample}，$b_k \leftarrow$ **估算工作负载($\mathcal{T}^r$)** 初始化任务集 $\{\mathcal{M}_k^r = \emptyset | k \in \mathcal{K}\}$。
  初始化 $\{w_k = 0 | k \in \mathcal{K}\}$。
  $\mathcal{M}_{sort}^s \leftarrow$**排序**$\{N_m | m \in \mathcal{M}^r\}$
  对每个客户端 $m \in \mathcal{M}_{sort}^r$ **do**
    查找 $k^\star$ 依据公式4。
    将 m放入 $\mathcal{M}_{k^\star}^r$。
    $w_{k^\star} = w_{k^\star} + N_m * t_{k^\star}^{sample} + b_{k^\star}$。
  结束循环
结束条件判断
返回 $\mathcal{M}_1^r, ..., \mathcal{M}_K^r$.

估算工作负载($\mathcal{T}^r$):
若使用近期运行时间 **则**
  $t_k^{sample}, b_k \leftarrow$ 使用方程2 进行线性回归（$\mathcal{T}_\tau^r$）。
else
  t_k^{sample}，$b_k \leftarrow$ 关于方程2 的线性回归（$\mathcal{T}^r$）。
结束 if
返回 t_k^{sample}，b_k。

在不同设备上可能差异很大。为了更准确地建模工作负载，我们将公式1修改为：

$$T_{m,k} = N_m * t_k^{sample} + b_k. \tag{2}$$

现在，工作负载模型会考虑设备异构性，以便更好地将任务调度给客户端。

估算。根据硬件资源、数据集、模型架构以及批量大小和迭代次数等超参数来预测运行时间十分困难，要实现准确估算也需要耗费大量时间（）。因此，我们倾向于使用历史运行时间来拟合工作负载模型。

在训练过程中，每个设备 k 都会记录执行任务 $\left\{\hat{T}_{m,k}^r | m \in \mathcal{M}_k^r\right\}$ 在第 r 轮次的运行时间，并将其与结果 $\left\{\mathcal{C}_{m,E-1}^r | m \in \mathcal{M}_k^r\right\}$一同发送回服务器。

在第 r轮进行调度时，服务器将使用所有 $\mathcal{T}^r = \left\{\hat{T}_{m,k}^i | m \in \mathcal{M}_k^r, \ k \in \mathcal{K}, \ i \in [0, ..., r-1]\right\}$ 来拟合工作负载模型，即公式2，从而获得估计参数 t_k^{sample} 和 b_k。

4.4 高效任务调度与负载均衡

任务调度的优化目标是最小化总训练时间。由于设备并行进行模拟，总训练时间由训练时间最长的设备决定。这里将设备 k 的训练时间表示为工作负载 w_k。忽略任务之间额外的切换时间，采用公式 2 的工作负载模型，我们得到累积工作负载 $w_k = \sum_{m \in \mathcal{M}_k} T_{m,k}$。因此，一轮的估计训练时间为 $\max(w_k | k \in \mathcal{K})$。由此可将优化问题表述为：

$$\min_{\mathcal{M}_1, ..., \mathcal{M}_K} \max\left(\left\{\sum_{m \in \mathcal{M}_k} T_{m,k} | k \in \mathcal{K}\right\}\right). \tag{3}$$

我们在算法3中设计了一种高效的任务调度方法，其计算复杂度为 $O(KM_p)$。我们首先按照降序对 $\{N_m | m \in \mathcal{M}^r\}$ 进行排序，得到 $\mathcal{M}_{sort}^r$。然后，我们遍历 $m \in \mathcal{M}_{sort}^r$，逐步寻找解 $\mathcal{M}_1$，...，$\mathcal{M}_K$。在调度新任务 m时，我们将其分配给设备 $k^\star$ 以满足：

$$k^\star = \arg\min_{k \in \mathcal{K}} \max\left(\left\{w_k + N_m * t_k^{sample} + b_k | k \in \mathcal{K}\right\}\right). \tag{4}$$

用于模拟客户端 m 的任务将被放入 $\mathcal{M}_{k^\star}^r$中。遍历 $\mathcal{M}_{sort}^r$ 后，服务器可以获得 $\mathcal{M}_1$，...，$\mathcal{M}_K$并将它们发送给设备。

应对动态硬件环境。通常，研究人员和工程师会与其他用户同时使用计算资源。这种不稳定的环境会使 t_k^{sample} 和 b_k 在训练过程中发生变化。因此，基于所有历史运行时间的估计会变得不准确。

为此，假设硬件环境在近期的短时间内相对稳定，我们提出丢弃早期记录的信息，仅利用最近一个时间窗口内的运行时间进行估计。用户可以根据其环境的不稳定性定义时间阈值 τ。然后，$\mathcal{T}_\tau^r = \left\{\hat{T}_{m,k}^i | m \in \mathcal{M}_k^r, \ k \in \mathcal{K}, \ i \in [r-\tau, ..., r-1]\right\}$ 将用于估计。我们将这种调度称为时间窗口调度。

4.5 复杂度分析

本节将分析不同仿真方案对硬件的要求。符号的定义以及不同仿真方案的复杂度总结于表1中。

CPU/GPU 内存与磁盘成本。SP仿真只使用一个设备来模拟联邦学习。将所有模型和客户端状态存储在该设备上需要 $O(s_m M + s_d M)$ 大小的内存。从磁盘加载客户端模型和状态可以将内存成本降低至 $O(s_m + s_d)$，但会增加分布式成本$O(s_m M_p + s_d M)$。

如图 1所示，`RW Dist.` 需要部署所有客户端，因此总共需要 $O(s_m M + s_d M)$ 大小的

内存来存储和训练客户端模型。通过管理状态，未选定的客户端无需从磁盘加载状态，因此内存减少了 $O(s_m M + s_d(M - M_p))$，相应地，磁盘成本增加到 $O(s_d M)$。

在 SD Dist. 中，设备只需加载全局模型并对选定客户端进行模拟，而无需部署所有客户端。这部分需要 $O(s_m M_p)$ 的空间。由于无法丢弃未选定客户端的客户端状态，设备需要 $O(s_m M)$ 大小的内存来存储这些状态。与 RW Dist. 类似，将客户端状态加载到该磁盘中可以将内存成本降低至 $O(s_m M_p + s_d M_p)$。

SD Dist. 和 Parrot 都将任务分配给 K 设- 备，以逐个模拟客户端。因此，它们只需要 $O(s_m K)$ 大小的内存来训练客户端模型，并需要 $O(s_d M)$ 来存储客户端状态。借助状态管理器， K 设备可以在需要时加载客户端状态。因此，总内存成本降低至 $O(s_m K + s_d K)$。

请注意，对于所有方案，客户端状态的磁盘成本都无法进一步降低。因此，磁盘大小可能会成为使用有状态客户端进行大规模联邦学习模拟的瓶颈。

通信成本。如图1 和2所示，`RW Dist.` 和`SD Dist.` 中的服务器需要向 M_p 客户端发送参数，并接收它们返回的 M_p 结果。因此，通信往返次数为 $O(M_p)$，通信数据量为 $O(s_a M_p + s_e M_p)$。

对于 FA Dist.，服务器只会在客户端可用时向其分配一个任务和参数。因此，FA Dist. 的通信往返次数和通信数据量实际上与 SD Dist. 相同。

得益于 Parrot 的分层聚合，设备只需发送一次结果，而不必多次发送。因此，通信往返次数仅为 $O(K)$，通信数据量为 $O(s_a K + s_e M_p)$。此处，由于这些特殊参数旨在由服务器收集而非求平均，因此通信数据量 $O(s_e M_p)$ 无法进一步优化。

工作负载估算的时间成本。如果使用所有运行时间历史数据来拟合工作负载模型2。对于设备 k的每个工作负载模型，在通信轮次 rM_p/K 时，数据点数量为 r。由于每个工作负载模型只有2 个特征，即 t_k^{sample} 和 b_k，因此对每个模型进行线性回归的复杂度为 $O(4(rM_p)^2/K^2)$。如果需要长期开展大规模联邦学习，可以像第 4.3 节中的时间窗口调度那样，舍弃较早的历史数据以降低计算复杂度。

调度的时间成本。 在我们的调度算法中，需要遍历 $\mathcal{M}^r$。对于每个 $m \in \mathcal{M}^r$，我们需要找到一个 $k^\star$，以使累积工作负载达到最小最大值，这需要搜索 $O(K)$ 次。因此，计算复杂度为 $O(KM_p)$。我们将在实验中表明，工作负载估计和调度的时间远少于训练时间。

5 实验

5.1 实验设置

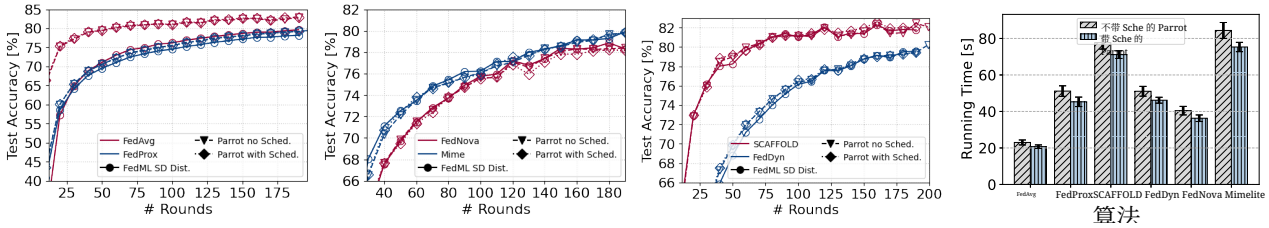
数据集和模型。 为验证 `Parrot` 能否在各种数据集和模型上良好运行，我们训练了三种不同的模型，包括 ResNet-18、ResNet-50（He 等人，2016）和 Albert-V2（Lan 等人，2020）），并在 FEMNIST（Caldas 等人，2018）、ImageNet（Russakovsky 等人，2015）和 Reddit（Lai 等人，2021）上进行训练。数据集、模型和超参数的详细信息列于附录中。

联邦学习设置。 为模拟不同的联邦学习场景，我们针对 100 个和 1000 个并发客户端开展实验。此外，我们还在 ImageNet 上尝试了不同的联邦学习划分方法，以观察 Parrot 在数量异质性不同的情况下系统性能的变化[1]。

硬件配置。 我们在三个不同的计算机集群 A、B 和 C 上开展实验，以验证 Parrot 在不同硬件环境下的表现。硬件和软件环境的详细信息列于附录中。我们还调整设备数量（$4 \sim 32$），以观察 Parrot 能获得多大的加速效果。此外，我们还模拟异构 GPU 环境（异构 GPU ）和不稳定计算环境（动态 GPU ），以评估调度能否适应异构且不稳定的设备。异构 GPU 环境和不稳定设备的模拟方法见附录。请注意，集群 C 本身是一个异构计算集群，用于验证真实场景中的设备异质性，而非模拟情况。

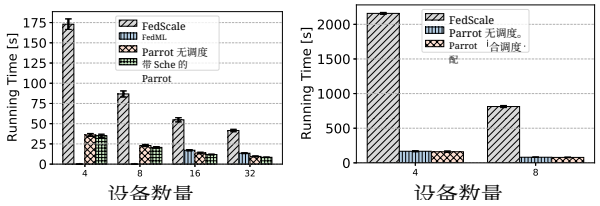
联邦学习算法。 为考察`Parrot` 能否支持多种联邦学习算法，我们选择了 6 种经典和先进的联邦学习算法进行模拟。在这些算法中，FedAvg (McMa- han 等人, 2017) 和 FedProx (Li 等人,2020) 只需通信机器学习模型参数。除模型参数外，FedNova (Wang 等人, 2020) 还需通信聚合权重；Mime (Karimireddy 等人, 2020a) 需要通信本地批次梯度和服务器优化器状态；SCAFFOLD (Karimireddy 等人,2020b) 需要通信控制变量。此外，Feddyn (Acar 等人, 2021) 需要在客户端存储本地梯度，SCAFFOLD (Karimireddy 等人, 2020b) 需要在客户端存储本地控制变量。我们将证明，`Parrot` 不会改变这些算法的性能，并能为它们提供便捷支持，以进行大规模模拟。

[1]请注意，只有数量偏斜（Kairouz 等人，2021）会影响系统性能

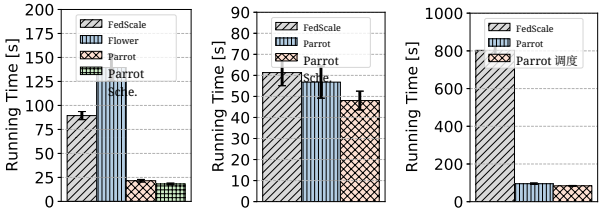


(a) 不含特殊参数和有状态客户端的算法。(b) 含特殊参数但不含有状态客户端的算法。(c) 同时含特殊参数和有状态客户端的算法。(d) 每轮运行时间。客户端。

图 *4.* 不同联邦学习算法的测试准确率和每轮运行时间。



(a) 集群 A 上的 FEMNIST。(b) 集群 A 上的 ImageNet(a)。



(c) FEMNIST on 集群 B。 (d) 集群 B 上的 Reddit 集群 B。 (e) ImageNet(b) 上的集群 A。

图 *5*。 不同联邦学习框架在不同设备数量下的每轮运行时间。

5.2 实验结果

比较不同框架我们将 Parrot与不同的联邦学习仿真框架进行比较，包括 FedML (He 等人,2020)、FedScale (Lai 等人, 2021) 和 Flower (Beutel 等人, 2020)，以展示 Parrot的性能效率。图5展示了不同框架在不同集群中的运行时间。 Parrot在所有情况下的加速比约为 1。 2 ∼ 10 × 倍，高于 FedScale、Flower 和 FedML。请注意，FedML 的仿真方案 SD.Dist 无法部署 $M_p = 100$ FEMNIST 联邦学习到集群 A 中的 4 台和 8 台设备上，这是由于 SD. Dist 的内存需求过高。

实现联邦学习算法我们通过 FedML 和 Parrot 实现了多种联邦学习算法。需要注意的是，对于需要有状态客户端的 SCAFFOLD 和 FedDyn，我们在 FedML 中添加了客户端状态管理器来实现它们。不同算法的收敛结果如图 4所示，结果表明，Parrot实现的算法能够达到与 SDDist. 方案相近或更高的测试准确率。图4(b) 显示 Parrot 能够很好地支持需要通信特殊参数的算法。图4(c)显示 Parrot 能够通过大规模实验实现有状态客户端算法。

不同联邦学习算法的平均运行时间如图4(d)所示。这表明，Parrot 的调度有助于加快所有选定联邦学习算法的运行速度。对于一些需要额外计算开销的算法，运行时间缩短幅度更大。

表 *3.* 不同联邦学习任务的 GPU 内存开销。

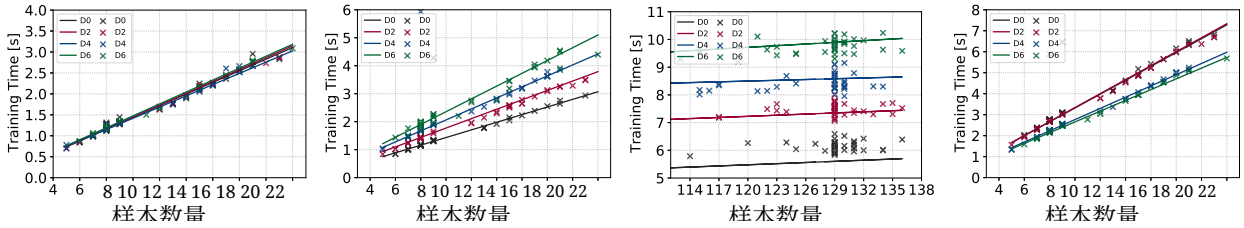
			GPU 内存开销（MB）		
数据集	M_p	K	SP	SD Dist.	FA Dist. & Parrot
FEMNIST	100	8	1,122	112,200	8,976
FEMNIST	100	16	1,122	112,200	17,952
ImageNet	1000	8	3,305	3305,000	26,440
ImageNet	1000	16	3,305	3305,000	52,880

设备数量不同 我们改变设备数量，以验证Parrot 的可扩展性，如图7所示。结果表明，Parrot 能够随着客户端数量增加而良好扩展。图8 所示的工作负载估计和调度时间几乎随设备数量线性增长。与运行时间相比，调度耗时很少。

并发客户端数量不同 图10展示了模拟不同并发客户端数量（即 100 和 1000）下联邦学习的结果。我们的调度方案在不同规模下均有优势。

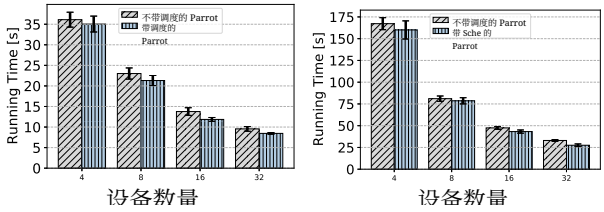
GPU 内存开销 表3 展示了模拟不同联邦学习实验的 GPU 内存开销。结果表明，SD Dist. 需要大量内存来模拟大规模实验， $M_p = 1000$。此外，模拟包含 $M_p = 100$的实验也需要大量内存。

异构计算设备 图 6 显示，Parrot 能够准确估计任务在同构或异构设备上的运行时间。因此，如图 9所示，Parrot 进行调度时的运行时间远低于不进行 Parrot 调度时的运行时间。



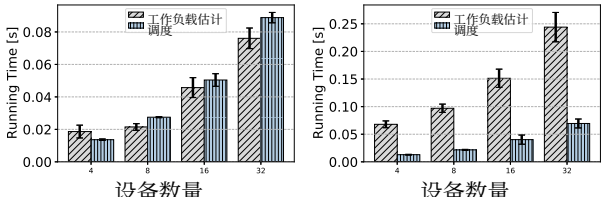
(a) (b) (c) (d)

图 *6.* 在不同环境下使用 8 个设备训练 FEMNIST 的运行时间。我们展示了估计的工作负载模型和一些采样的运行时间。为使可视化效果更加清晰，这里仅选择 4 个设备进行展示。(a) 在同构 GPU 集群 A 上。(b) FEMNIST， $K = 8$，同构 GPU 集群 A，模拟异构 GPU。(c) ImageNet， $K = 8$，同构 GPU 集群 A，模拟异构 GPU。(d) FEMNIST， $K = 8$，异构 GPU 集群 C。



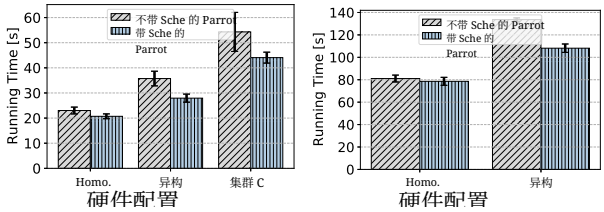
(a) 使用 $M_p = 100$ 的 FEMNIST。(b) 使用 $M_p = 100$ 的 ImageNet。

图 *7.* 不同设备数量下的每轮运行时间。



(a) 使用 $M_p = 100$ 的 FEMNIST。(b) 使用 $M_p = 100$ 的 ImageNet。

图 *8.*不同设备数量下每轮工作负载估计和调度的平均时间。

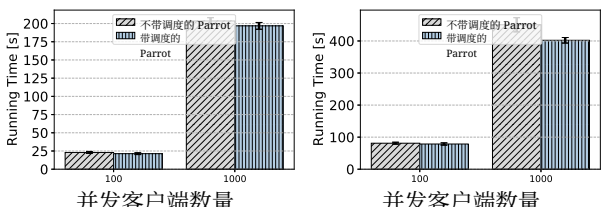


(a) 使用 $M_p = 100$ 的 FEMNIST。(b) 使用 $M_p = 100$ 的 ImageNet。

图*9*。不同硬件配置下的每轮运行时间。

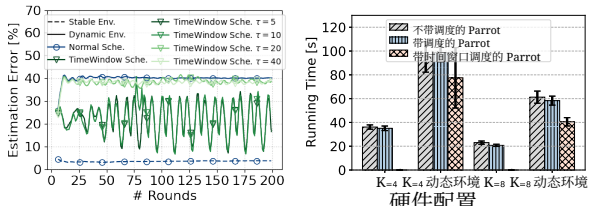
ing.

动态环境 在动态环境中，由于设备的计算性能不稳定，相同任务的运行时间可能会持续变化。因此，很难利用所有历史数据来拟合我们的工作负载模型 2。如图 11（a）所示，使用所有历史数据进行调度无法估计动态环境中的工作负载。因此，其运行时间与



（a）FEMNIST。 （b）ImageNet。

图*10*。不同并发客户端数量下的每轮运行时间。



（a）估计误差。 （b）每轮运行时间。

图 *11*。不同环境和调度算法的每轮运行时间及估计误差。

`Parrot` 未进行调度，如图 11(b)所示。时间窗口调度能够更准确地估计工作负载，从而缩短运行时间。

6 结论

在本工作中，我们揭示了模拟联邦学习所面临的挑战与瓶颈，并设计了一个名为 Parrot 的新型联邦学习系统。Parrot 提供高效的联邦学习模拟器，大幅降低硬件要求。借助状态管理器，Parrot 为配备有状态客户端的各种联邦学习算法提供友好支持。Parrot 的调度组件显著加速了联邦学习模拟。通过通用且易用的 API，用户无需修改代码，即可将经过验证的联邦学习算法和模型从模拟环境无缝迁移到真实世界部署中。

参考文献

Acar、D. A. E.、Zhao、Y.、Matas、R.、Mattina、M.、What- mough、P. 和 Saligrama、V. 基于动态正则化的联邦学习。发表于 *ICLR*，2021 年。

Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Parcollet, T., 和 Lane, N. D. Flower：一个友好的联邦学习研究框架。*arXiv* 预印本 *arXiv:2007.14390*，2020。

Caldas, S., Wu, P., Li, T., Koneˇcn`y, J., McMahan, H. B., Smith, V., 和 Talwalkar, A. LEAF：联邦设置的基准测试。*arXiv* 预印本 *arXiv:1812.01097*，2018。Chen, H.-Y. 和 Chao, W.-L. 关于弥合图像分类中通用联邦学习与个性化联邦学习之间的差距。载于国际学习表征会议，2021。Collins, L., Hassani, H., Mokhtari, A., 和 Shakkottai, S. 利用共享表征实现个性化联邦学习。载于 Meila, M. 和 Zhang, T.（编），第*38*届国际机器学习会议论文集，第139卷，机器学习研究论文集，第2089–2099页。PMLR，2021年7月18–24日。Dayan, I., Roth, H., 和 A. Zhong 等。利用联邦学习预测新冠患者的临床结局。*npjDigitalMedicine*，3(1)，2020年12月。ISSN 2398-6352。doi: 10.1038/s41746-020-00323-1。Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., 和 Ramage, D. 用于移动键盘预测的联邦学习。*arXiv* 预印本 *arXiv:1811.03604*，2018。He, C., Li, S., So, J., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., Shen, L., Zhao, P., Kang, Y., Liu, Y., Raskar, R., Yang, Q., Annavaram, M., 和 Avestimehr, S. FedML：用于联邦机器学习的研究库和基准测试。*arXiv* 预印本 *arXiv:2007.13518*，2020。He, K., Zhang, X., Ren, S., 和 Sun, J. 用于图像识别的深度残差学习。载于 *CVPR*，2016。Hsu, T.-M. H., Qi, H., 和 Brown, M. 基于真实世界数据分布的联邦视觉分类。载于*ECCV*，2020。Huba, D., Nguyen, J., Malik, K., Zhu, R., Rabbat, M., Yousefpour, A., Wu, C.-J., Zhan, H., Ustinov, P., Srinivas, H., Wang, K., Shoumikhin, A., Min, J., 和 Malek, M. Papaya：实用、隐私保护且可扩展的联邦学习。载于 Marculescu, D., Chi, Y., 和 Wu, C.（编），机器学习与系统论文集，第4卷，第814–832页，2022年。网址 https://proceeding s.mlsys.org/paper/2022/file/f340f1b1f65b6df5b5e3f94d95b11daf-Paper.pdf.

Ingerman, A. 和 Ostrowski, K. *TensorFlow* 联邦学习，2019年。网址 https://medium.com/tensorflow/introducing-tensorflow-federated-a4147aa20041.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R. 等。联邦学习的进展与开放问题。 *Found. Trends Mach. Learn.*，14(1-2): 1–210，2021年。Karimireddy, S. P., Jaggi, M., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U. 和 Suresh, A. T. Mime：在联邦学习中模仿集中式随机算法。2020a年。Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S. 和 Suresh, A. T. SCAFFOLD：用于联邦学习的随机控制平均。发表于*ICML*，2020b年。Lai, F., Dai, Y., Zhu, X., Madhyastha, H. V. 和 Chowdhury, M. Fedscale：联邦学习模型与系统性能基准测试。发表于第一届可靠与安全联邦学习系统挑战研讨会论文集，ResilientFL '21，第1–3页，纽约，NY，美国，2021年。计算机协会。ISBN 9781450387088。doi: 10.1145/3477114.3488760。网址 https://doi.org/10.1145/3477114.3488760。Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P. 和 Soricut, R. Albert：用于自监督语言表示学习的轻量级 BERT。发表于国际学习表征会议，2020年。网址https://openreview.net/forum?id=H1eA7AEtvS。Li, A., Sun, J., Zeng, X., Zhang, M., Li, H. 和 Chen, Y. Fedmask：通过异构掩码实现计算与通信高效的个性化联邦学习。发表于第*19*届 *ACM* 嵌入式网络传感器系统会议论文集，SenSys '21，第42–55页，纽约，NY，美国，2021a年。计算机协会。ISBN 9781450390972。Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J. 和 Su, B.-Y. 使用参数服务器扩展分布式机器学习。发表于第*11*届 {*USENIX*} 操作系统设计与实现研讨会（{*OSDI*} *14*），第583–598页，2014年。Li, Q., Diao, Y., Chen, Q. 和 He, B. 非独立同分布数据孤岛上的联邦学习：一项实验研究。*arXiv* 预印本 *arXiv:2102.02079*，2021b年。

Li, Q., He, B. 和 Song, D. 模型对比联邦学习。发表于 *CVPR*，2021c年。

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., 和 Smith, V. 异构网络中的联邦优化。发表于 *MLSys*，2020。

Liang, P. P., Liu, T., Ziyin, L., Allen, N. B., Auerbach, R. P., Brent, D., Salakhutdinov, R., 和 Morency, L.-P. 思考于本地，行动于全局：具有局部与全局表示的联邦学习。*arXiv*预印本 *arXiv:2001.01523*，2020。

Lin, T., Kong, L., Stich, S. U., 和 Jaggi, M. 用于联邦学习中鲁棒模型融合的集成蒸馏。在*NeurIPS*，2020。

Liu, Y., Huang, A., Luo, Y., Huang, H., Liu, Y., Chen, Y., Feng, L., Chen, T., Yu, H., 和 Yang, Q. Fedvision：由联邦学习驱动的在线视觉目标检测平台。发表于 *AAAI*人工智能会议论文集，第34卷，第13172–13179页，2020年。

Luo, M., Chen, F., Hu, D., Zhang, Y., Liang, J. 和 Feng, J. 不惧异质性：面向非独立同分布数据联邦学习的分类器校准。载于*NeurIPS*，2021 年。

McMahan, B., Moore, E., Ramage, D., Hampson, S. 和 y Arcas, B. A. 基于去中心化数据的深度网络通信高效学习。载于 *AISTATS*，2017 年。

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. 等。PyTorch：一种命令式风格的高性能深度学习库。载于 *NeurIPS*，第 8024–8035 页，2019 年。

Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Koneˇcn´y, J., Kumar, S. 和 McMahan, H. B. 自适应联邦优化。载于*ICLR*，2021 年。

Rieke, N., Hancox, J., Li, W., Milletarı, F., Roth, H., Al-barqouni, S., Bakas, S., Galtier, M., Landman, B., Maier-Hein, K., Ourselin, S., Sheller, M., Summers, R., Trask, A., Xu, D., Baust, M., and Cardoso, M. 借助联邦学习实现数字健康的未来。*npjDigitalMedicine*, 3(1), 2020年12月。ISSN 2398-6352。doi: 10.1038/s41746-020-00323-1。

Russakovsky, O.、Deng, J.、Su, H.、Krause, J.、Satheesh, S.、Ma, S.、Huang, Z.、Karpathy, A.、Khosla, A.、Bernstein, M. 等。《ImageNet 大规模视觉识别挑战赛》。国际计算机视觉杂志，115(3)：211–252，2015。Ryffel, T.、Trask, A.、Dahl, M.、Wagner, B.、Mancuso, J.、Rueckert, D. 和 Passerat-Palmbach, J.《隐私保护深度学习的通用框架》。*arXiv* 预印本 *arXiv:1811.04017*，2018。

Sarvotham, S.、Riedi, R. 和 Baraniuk, R.《网络流量的连接级分析与建模》。载于第 *1* 届 *ACM SIGCOMM* 互联网测量研讨会论文集，第 99–103 页，美国纽约州纽约市，2001 年。

Shi, S., Chu, X., and Li, B. MG-WFBP：面向分布式同步 SGD 算法的高效数据通信。发表于 *IEEE INFOCOM 2019 - IEEE*计算机通信会议，第 172–180 页，2019 年。doi: 10.1109/INFOCOM.2019.8737367。

Tang, Z., Zhang, Y., Shi, S., He, X., Han, B., 和 Chu, X. 虚拟同质性学习：抵御联邦学习中的数据异质性。载于 Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., 和 Sabato, S.（编），第 *39* 届国际机器学习会议论文集，第 162 卷，机器学习研究论文集，第 21111–21132 页。PMLR，2022年7月17日至23日。

Thakur, R., Rabenseifner, R., 和 Gropp, W. MPICH 中集体通信操作的优化。国际高性能计算应用杂志，2005。

Wang, J., Liu, Q., Liang, H., Joshi, G., 和 Poor, H. V. 解决异构联邦优化中的目标不一致问题。载于*NeurIPS*，2020。

Woodworth, B. E., Patel, K. K., 和 Srebro, N. 异构分布式学习中的小批量与本地随机梯度下降。神经信息处理系统进展，33：6281–6292，2020。

Xie, Y., Wang, Z., Chen, D., Gao, D., Yao, L., Kuang, W., Li, Y., Ding, B., 和 Zhou, J. FederatedScope：面向异构性的灵活联邦学习平台。*arXiv* 预印本 *arXiv:2204.05011*，2022年。

Xu, P., Shi, S., 和 Chu, X. Docker 容器中深度学习工具的性能评估。载于*2017*年第三届大数据计算与通信国际会议（*BIGCOM*），第395–403页，2017年。doi：10.1109/BIGCOM.2017.32。

Zhang, W., Gupta, S., Lian, X. 和 Liu, J. 面向分布式深度学习的陈旧性感知异步随机梯度下降。载于 第二十五届国际人工智能联合会议论文集，IJCAI'16，第 2350–2356 页。AAAI出版社，2016。ISBN 9781577357704。

附录

A 实验配置

本节将展示详细的实验配置和更多结果。

表*4.* 数据集和超参数详情。

数据集	划分	# 客户端	M_p	模型	模型参数	学习率	批量大小	本地训练轮数
FEMNIST	自然	3,400	100	ResNet-18	11M	0.05	20	10
FEMNIST	自然	3,400	1000	ResNet-18	11M	0.05	20	10
ImageNet(a)	狄利克雷(0.1)	10,000	100	ResNet-50	23M	0.05	20	2
ImageNet(a)	狄利克雷(0.1)	10,000	1000	ResNet-50	23M	0.05	20	1
ImageNet(b)	数量偏斜(5.0)	10,000	100	ResNet-50	23M	0.05	20	2
Reddit	自然	1,660,820	100	Albert-Base-v2	11M	0.0005	20	5

表*5*。 硬件配置。

Name	CPU	GPU	网络带宽	操作系统	CUDA	PyTorch
集群 A	Intel(R) Xeon(R) Gold 5115 处理器 @ 2.40GHz	4× RTX 2080 Ti 在 8 个节点上	10 Gbps 或 100 Gbps	Ubuntu 18.04.6	V11.3	V1.12.1
集群 B	Intel(R) Xeon(R) Gold 5220R 处理器 @ 2.20GHz	8 ×Quadro RTX 5000	10 Gbps	Ubuntu 18.04.5	V11.2	V1.12.1
集群 C	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz	节点1: 4×Tesla K80 ，节点2: 2×Tesla P40, 节点3: 2×Tesla P40	10 Gbps 或 100 Gbps	Oracle Linux 服务器 8.6	V11.4	V1.10.1

异构 GPU 环境模拟。 为了测试 Parrot 在不同硬件环境中的适应能力，我们对异构 GPU 环境（异构 GPU）和不稳定设备（动态 GPU）进行模拟。由于异构 GPU 执行相同任务所需的计算时间不同，我们预先为设备分配不同的异构比率 $\{\eta_k|k \in \mathcal{K}\}$ 。在设备 k完成任务 m的本地训练后，耗时为 $T^r_{m,\ k}$，设备将休眠一段时间 $\eta_k T^r_{m,\ k}$。因此，同构 GPU 将呈现不同的计算时间。 η_k 越低，表示 GPU 性能越强。现在，服务器将获取运行时间 $\eta_k T^r_{m,\ k}$，以进行工作负载估计，而不是使用 $T^r_{m,\ k}$。

不稳定设备模拟。 与模拟异构 GPU 环境类似，我们利用不同的休眠时间来模拟不稳定设备。具体而言，我们生成不同客户端在不同通信轮次中的休眠时间比率，即 $(1 + cos(3.14r/R + k))$。因此，在整个训练过程中，设备在不同轮次的计算性能各不相同。