PaddleDTX Documentation

baidu

2022 年 01 月 25 日

整体介绍

CHAPTER 1

系统介绍

PaddleDTX，是一个基于**去中心化存储**的专注于**分布式机器学习**技术的解决方案，攻克海量隐私数据的安全存储问题，并且实现多方数据的安全交换，助其突破数据孤岛，共同建模，联合发挥数据的最大价值。

1.1 主要特征

PaddleDTX 的主要特征如下：

- 支持多个学习过程**并行**运行的**多方安全计算**框架，集成多种横向联邦学习和纵向联邦学习算法
- **安全存储**高敏感数据，防止隐私泄漏，支持故障容错，抵御存储作弊
- 去中心化管理存储节点，支持**无上限**数据纳管
- 保证多方数据联合建模的**全链路可信**

1.2 架构概览

PaddleDTX 由多方安全计算网络、去中心化存储网络、区块链网络构建而成。

1.2.1 1.1 多方安全计算网络

有预测需求的一方为计算需求节点。可获取样本数据进行模型训练和预测的一方为任务执行节点，多个任务执行节点组成一个 SMPC（多方安全计算）网络。计算需求节点将任务发布到区块链网络，任务执行节点确认后执行任务。数据持有节点对任务执行节点的计算数据做信任背书。

SMPC 是一个支持多个学习过程并行运行的框架，会陆续集成更多纵向联邦学习、横向联邦学习算法。

1.2.2 1.2 去中心化存储网络

数据持有节点将自己的隐私数据进行加密、切分、副本复制后分发到存储节点，存储节点通过应答数据持有节点的挑战证明自己持有数据分片。通过这些机制，实现了在不泄漏隐私的前提下充分且安全地利用存储资源。

训练样本和预测数据集往往是归属于不同机构的隐私数据。这些机构可以作为数据持有节点加入到去中心化存储网络中，通过多方安全计算网络发挥数据的最大价值。

1.2.3 1.3 区块链网络

训练任务和预测任务通过区块链网络广播到任务执行节点，后者继而执行训练任务和预测任务。数据持有节点和存储节点在副本保持证明、健康状态监控过程中，通过区块链网络实现信息交换。

目前，PaddleDTX 底层采用的区块链框架是 XuperChain。

CHAPTER 2

基本概念

下面介绍一些 PaddleDTX 的相关概念，帮助您初步认识系统，了解其基本运行机制，方便后续进一步阅读。

2.1 节点和网络

PaddleDTX 中有五类节点：

- **计算需求节点**（Requester）有训练模型和预测需求。
- **任务执行节点**（Executor）拥有使用数据的权限，参与多方安全计算，进行模型训练和数据预测。
- **数据持有节点**（DataOwner）是数据的归属方，有存储数据的需求。
- **存储节点**（Storage Nodes）有丰富的闲置的存储资源，可以提供存储服务。
- **区块链节点**构成区块链网络 (Blockchain Network)，基于不同的区块链框架，他们有不同的定义。

计算需求节点、任务执行节点，以及数据持有节点组建了 PaddleDTX 的计算层。数据持有节点和存储节点组建了去中心化存储网络，负责数据的存取。区块链网络是去中心化治理和可信 AI 的基石。

2.2 智能合约

智能合约，是运行在区块链上，旨在以信息化方式传播、验证或执行合同的计算机协议 [9] 。PaddleDTX 中的任务广播、节点的去中心化治理、副本保持证明机制都基于智能合约。

2.3 任务

任务，是您使用 PaddleDTX 时用到的最基本概念。

PaddleDTX 中有两类任务：

- **训练任务**以通过训练得到模型为目标；
- **预测任务**以预测数据的目标值为目标。

任务由**计算需求节点**发布到区块链网络，由**数据持有节点**确认数据使用权，由**任务执行节点**最终执行。

2.4 算法

PaddleDTX 中的算法，一般指的是经过分布式改造的机器学习算法，即联邦学习算法。

目前开源了**纵向联邦学习**算法，包括**多元线性回归**和**多元逻辑回归**。

2.5 训练样本和预测数据集

PaddleDTX 中的训练样本和预测数据集都是以文件的形式存储于中心化存储网络，在发布训练任务或者预测任务的时候，由**计算需求节点**指定。

2.6 模型

算法和训练样本确定模型。通过模型可以对预测数据集的标签值进行预测。PaddleDTX 中的模型，以“分片”的形式存储在参与训练的**任务执行节点**的本地，在预测时，**任务执行节点**使用各自的模型进行计算，再汇总得到最终结果。

CHAPTER 3

正在进行中

我们即将支持的主要功能如下:

1. 支持更多的机器学习算法和对应的分布式改造，主要包括神经网络、决策树等；
2. 支持横向联邦学习算法，计划先对多元线性回归和多元逻辑回归进行改造；
3. 提供联邦学习训练参数的评估能力，通过交叉验证等方式评估训练参数的优劣；
4. 优化目前使用的加法同态算法 Paillier 的性能；
5. 去中心化存储服务支持负载均衡策略，根据存储节点剩余资源和以往表现情况，在文件分发时找到最优节点列表。

CHAPTER 4

快速安装

我们为您提供了能够快速拉起 PaddleDTX 测试网络的脚本，在使用前需要您准备如下环境:

- docker, 推荐版本 18.03+ 点击下载安装 docker
- docker-compose, 推荐版本 1.26.0+ 点击下载安装 docker-compose
- 如果使用 Mac 启动服务，Docker Desktop 至少设置为 4GB 运行时内存，参考Docker Desktop for Mac 用户手册

环境准备好之后，可以通过执行脚本快速拉起网络：

```
$ cd PaddleDTX/scripts
$ sh network_up.sh start
```

使用脚本也可以快速销毁网络：

```
$ sh network_up.sh stop
```

关于 PaddleDTX 的使用可以参考命令行工具及相关案例。

CHAPTER 5

编译和安装

5.1 源码编译和安装

5.1.1 环境准备

PaddleDTX 使用 golang 进行开发，当您使用源码进行编译和安装时，首先需要准备源码以及编译的环境：

- 系统环境, 推荐使用 Linux 或 MacOS 操作系统
- golang 编译器, 推荐版本 1.13+ 点击下载安装 golang
- git 源码下载工具 点击下载安装 git
- make 编译工具
  - ubuntu:

```
$ sudo apt-get install -y make
```

  - centos:

```
$ sudo yum install -y make
```

  - macOS: 通过 iTunes 下载安装 xcode

5.1.2 源码编译

1. 下载源码

```
$ git clone https://github.com/PaddlePaddle/PaddleDTX.git
```

2. 编译 XuperDB

```
$ cd PaddleDTX/xdb
$ make
```

编译产出为 output 文件夹，内容为：

```
├── conf
│   ├── config-dataowner.toml
│   └── config-storage.toml
├── xdb                    //数据存储服务启动二进制，
↪根据配置文件的不同有数据持有节点或存储节点两种类型
└── xdb-cli                //数据存储服务操作客户端
```

3. 编译 Distributed AI

```
$ cd ../dai
$ make
```

编译产出为 output 文件夹，内容为：

```
├── conf
│   ├── config-cli.toml     //客户端配置,也可以使用config.toml作为客户端配置
│   └── config.toml         //服务配置
├── executor                //DAI任务执行节点服务启动二进制
├── executor-cli            //DAI任务执行节点操作客户端
└── requester-cli           //计算需求节点操作客户端
```

4. 编译区块链合约

```
$ go build -o paddlempc ./blockchain/xchain/contract
```

编译产出为 paddlempc 合约文件，为安装在 xchain 区块链上的合约文件。

5.1.3 网络部署

1. 部署区块链网络

   PaddleDTX 使用区块链网络支撑计算层和去中心化存储网络，底层依赖可以使用同一个区块链网络。

   这里使用百度超级链 xchain v3.9 作为底层区块链网络，可参考 XuperChain 环境部署 来搭建区块链网络。

   您需要了解如何创建合约账户、部署智能合约，详细参考 部署 native 合约，更多内容请参考XuperChain 官方文档 。

   合约安装过程如下：

```
# 定义合约账户和合约名称
$ export contractAccount='1234567890123456'
$ export contractName='paddlempc'

# 创建区块链账户，并转 token
$ ./xchain-cli account newkeys --strength 1 -o ukeys
$ export address=`cat ./ukeys/address`
$ ./xchain-cli transfer --to $address --amount 1000000000000

# 创建区块链合约账户，并转 token
$ ./xchain-cli account new --account $contractAccount --fee 1000 --keys ./ukeys

# 给合约账户转 token
$ ./xchain-cli transfer --to XC${contractAccount}@xuper --amount 100000000000 --
↪keys ./ukeys

# 安装合约
$ ./xchain-cli native deploy --account XC${contractAccount}@xuper --runtime go -a
↪'{"creator":"XC${contractAccount}@xuper"}' --cname $contractName ./
↪$contractName --fee 19267894 --keys ./ukeys

# 查询合约安装的状态
$ ./xchain-cli contract query $contractName
```

2. 部署 XuperDB

   每一个 XuperDB 的节点都有一对公私钥，用来标识节点的账户，公私钥可以通过如下 XuperDB 的客户端命令进行获取：

```
$ ./xdb-cli key genkey -o ./keys
```

   请妥善保存您创建的公私钥对，在后续的配置及命令行使用时您将会频繁的用到它。

   XuperDB 包含两种类型的节点服务，数据持有节点和存储节点，我们需要分别启动这两种服务。为了方便，我们这里两种类型的服务分别启动一个。首先进入到 xdb 的编译产出 output 文件中：

1. 启动数据存储节点

   修改配置文件，修改内容如下：

```
# vim conf/config-storage.toml
#
type = "storage"
[storage]
name = "storageNode1"
# 修改服务监听端口及对外服务地址
listenAddress = ":8122"
publicAddress = "127.0.0.1:8122"

# genkey创建的私钥
keyPath = "./keys"

[storage.blockchain]
    type = "xchain"
    [storage.blockchain.xchain]
        # 助记词为用户部署区块链网络后，安装合约过程中创建的区块链账户，取值./
↪ukeys/mnemonic
        mnemonic = "充 雄 孔 坝 低 狠 争 短 摸 拜 晨 造"
        contractName = "paddlempc"
        contractAccount = "XC1234567890123456@xuper"
        chainAddress = "127.0.0.1:37101"
        chainName = "xuper"

    [storage.blockchain.fabric]
        configFile = "./config/fabric/config.yaml"
        channelId = "mychannel"
        chaincode = "mycc"
        userName = "Admin"
        orgName = "org1"

[storage.mode]
    type = "local"
    [storage.mode.local]
        rootPath = "./slices"

[storage.monitor]
    challengingSwitch = "on"
    nodemaintainerSwitch = "on"
    fileclearInterval = 24

[log]
```

(下页继续)

(续上页)

```
level = "debug"
path = "./logs"
```

其中，listenAddress 和 publicAddress 指定服务监听的地址及对外暴露的地址，blockchain 配置中使用区块链网络部署时创建的账户助记词、合约账户及合约名，rootPath 指定文件存储的本地路径。

启动服务：

```
$ nohup ./xdb -c conf/config-storage.toml > storage.log &
```

2. 启动数据持有节点

   修改配置文件，修改内容如下：

```
# vim conf/config-dataowner.toml
#
type = "dataOwner"
[dataOwner]
name = "dataOwnerNode1"
# 修改服务监听端口及对外服务地址
listenAddress = ":8121"
publicAddress = "127.0.0.1:8121"

# genkey创建的私钥
keyPath = "./keys"

[dataOwner.slicer]
    type = "simpleSlicer"
    [dataOwner.slicer.simpleSlicer]
        blockSize = 4194304
        queueSize = 4

[dataOwner.encryptor]
    type = "softEncryptor"
    [dataOwner.encryptor.softEncryptor]
        password = "abcdefg"

[dataOwner.challenger]
    type = "pdp"
    [dataOwner.challenger.pdp]
        maxIndexNum = 5
        sk = "W4HyiC7kx+bafMftHrD7Mz4ff2/0Bb103fUIrbRVkFk="
        pk = "AVcT6JO4Ddcq+JjC2Vw/kGZSrCjEeCu1Lu1EFya9C96Nb/
↪HtJkaHEJ4Ni89leAAaCbKu/
↪oEFrfEpb3oAaEd2JqNuyBlWZ0MZk7PmFkaUlYaTVvQRUWRRSmiIZa+iNtJEIYC/
↪AC5C88k1vRoXq3m7VonvJUFP95oLX3CSMMfSiUln"
```

(下页继续)

(续上页)

```
        randu = "AfM3n7CzmkbVEBRPOYV8gH1qpyaQdTNA1MZ7PHYfmWs="
        randv = "TKOt9kE7m5O7fCztoyy1J+WpNugLxKPS3hweeUK+09Y="

    [dataOwner.challenger.merkle]
        leveldbRoot = "./challenger"
        shrinkSize = 500
        segmentSize = 5

[dataOwner.blockchain]
    type = "xchain"
    [dataOwner.blockchain.xchain]
        # 助记词为用户部署区块链网络后，安装合约过程中创建的区块链账户，取值./
↪ukeys/mnemonic
        mnemonic = "充 雄 孔 坝 低 狠 争 短 摸 拜 晨 造"
        contractName = "paddlempc"
        contractAccount = "XC1234567890123456@xuper"
        chainAddress = "127.0.0.1:37101"
        chainName = "xuper"

    [dataOwner.blockchain.fabric]
        configFile = "./config/fabric/config.yaml"
        channelId = "mychannel"
        chaincode = "mycc"
        userName = "Admin"
        orgName = "org1"

[dataOwner.copier]
    type = "random-copier"

[dataOwner.monitor]
    challengingSwitch = "on"
    filemaintainerSwitch = "on"
    filemigrateInterval = 6

[log]
level = "debug"
path = "./logs"
```

其中，listenAddress 和 publicAddress 指定服务监听的地址及对外暴露的地址，blockchain 配置中使用区块链网络部署时创建的账户助记词、合约账户及合约名。

启动服务：

```
$ nohup ./xdb -c conf/config-dataowner.toml > dataowner.log &
```

注意：一般构建 *PaddleDTX* 网络至少需要两方参与，对应两个计算任务执行节点，每个任务执行节点可以从一个或多个数据持有节点获取数据，这里为了说明方便启动一个数据持有节点，您也可以根据实际需求自行启动多个数据存储节点和数据持有节点；配置中的 *keyPath* 参数为节点的身份，不同 *keyPath* 即对应了不同的身份。

3. 查看服务状态

   使用 xdb-cli 客户端执行如下命令，请求数据持有节点查看存储节点的在线状态：

```
$ ./xdb-cli nodes list --host http://127.0.0.1:8121
```

3. 部署 Distributed AI

一般多方安全计算至少由两个任务执行节点，所以这里部署两个任务执行节点。每一个任务执行节点都有一对公私钥，用来标识节点的账户，公私钥可以通过如下 executor-cli 的客户端命令进行获取：

```
$ ./executor-cli key genkey -o ./keys
```

请妥善保存您创建的公私钥对，在后续的配置及命令行使用时您将会频繁的用到它。

注意: 任务执行节点的账户也是通过公私钥对来标明。任务发布后时，任务执行节点会向数据持有节点发起文件授权申请，数据持有节点可通过或拒绝样本文件授权申请。

1. 准备两个任务执行节点的配置

```
$ cd PaddleDTX/dai/
$ cp -r output executor1
$ cp -r output executor2
```

需要分别修改对应的 conf/config.toml 文件：

```
# executor1
listenAddress = ":8184"
publicAddress = "127.0.0.1:8184"
# genkey创建的私钥
keyPath = "./keys"

[executor.storage]
    # 定义模型存储的路径
    localModelStoragePath = "./models"
    #␣
↪定义预测结果存储的方式，默认本地存储，如果用户采取XuperDB方式存储，则需提前生成数据持有节
↪/ukeys并授权，同时创建预测结果存储的命名空间
    type = 'Local'
    [executor.storage.XuperDB]
```

(下页继续)

(续上页)

```
        host = "http://127.0.0.1:8121"
        keyPath = "./ukeys"
        namespace = "mpc"
    [executor.storage.Local]
        localPredictStoragePath = "./predictions"
[executor.blockchain]
    [executor.blockchain.xchain]
        # 助记词为用户部署区块链网络后，安装合约过程中创建的区块链账户，取值./
↪ukeys/mnemonic
        mnemonic = "充 雄 孔 坝 低 狠 争 短 摸 拜 晨 造"
        contractName = "paddlempc"
        contractAccount = "XC1234567890123456@xuper"
        chainAddress = "127.0.0.1:37101"
        chainName = "xuper"
```

```
# executor2
listenAddress = ":8185"
publicAddress = "127.0.0.1:8185"
# genkey创建的私钥
keyPath = "./keys"

[executor.storage]
    # 定义模型存储的路径
    localModelStoragePath = "./models"
    #␣
↪定义预测结果存储的方式，默认本地存储，如果用户采取XuperDB方式存储，则需提前生成数据持有节
↪/ukeys并授权，同时创建预测结果存储的命名空间
    type = 'Local'
    [executor.storage.XuperDB]
        host = "http://127.0.0.1:8121"
        keyPath = "./ukeys"
        namespace = "mpc"
    [executor.storage.Local]
        localPredictStoragePath = "./predictions"
[executor.blockchain]
    [executor.blockchain.xchain]
        # 助记词为用户部署区块链网络后，安装合约过程中创建的区块链账户，取值./
↪ukeys/mnemonic
        mnemonic = "充 雄 孔 坝 低 狠 争 短 摸 拜 晨 造"
        contractName = "paddlempc"
        contractAccount = "XC1234567890123456@xuper"
        chainAddress = "127.0.0.1:37101"
        chainName = "xuper"
```

2. 启动服务

   分别在对应文件夹下执行如下命令，启动任务执行节点：

```
$ nohup ./executor &
```

   通过命令查看有两个 executor 的进程，则启动成功：

```
$ ps -aux | grep executor
```

5.2 通过 docker 安装

5.2.1 环境准备

PaddleDTX 也支持使用 docker 进行编译、安装和使用，您需要准备如下环境：

- docker, 推荐版本 18.03+ 点击下载安装 docker
- docker-compose, 推荐版本 1.26.0+ 点击下载安装 docker-compose

5.2.2 制作镜像

1. 制作 XuperDB 镜像

```
$ cd PaddleDTX/xdb
$ sh build_image.sh
```

   产出镜像名和版本号为 *registry.baidubce.com/paddledtx/paddledtx-dai:1.0* ，可以通过修改 *build_image.sh* 脚本来修改镜像名和版本号。

2. 制作 Distributed AI 镜像

```
$ cd PaddleDTX/dai
$ sh build_image.sh
```

   产出为计算需求节点和任务执行节点两个镜像, 镜像名和版本号分别为：

   - registry.baidubce.com/paddledtx/xdb-storage:1.0
   - registry.baidubce.com/paddledtx/xdb-dataowner:1.0 实际上他们是使用不同镜像名的同一个镜像，可以通过修改 *build_image.sh* 脚本来修改镜像名和版本号。

3. 编译合约

```
$ export contractName='paddlempc'
$ docker run -it --rm \
  -v $(dirname ${PWD}):/workspace \
  -v ~/.ssh:/root/.ssh \
  -w /workspace \
  -e GONOSUMDB=* \
  -e GO111MODULE=on \
  golang:1.13.4 sh -c "cd dai && go build -o ../testdata/blockchain/contract/
↪$contractName ./blockchain/xchain/contract"
```

5.2.3 网络部署

1. 部署区块链网络

```
$ cd PaddleDTX/testdata/blockchain
$ docker-compose -f docker-compose.yml up -d
```

搭建了三个节点的区块链网络，对应的区块链相关配置在文件夹 blockchain/xchain1/conf、blockchain/xchain2/conf、blockchain/xchain3/conf 下，需要调整配置时在网络拉起前进行修改。

可以通过容器中的 xchain-cli 客户端进行区块链上的一些操作，例如创建合约账户及安装智能合约。

```
# 定义合约账户和合约名称
$ export contractAccount='1234567890123456'
$ export contractName='paddlempc'

# 创建区块链账户，并转 token
# 这里也可以使用已有的区块链账户，文章后面的修改区块链配置可以不操作
$ docker exec -it xchain1.node.com sh -c "./xchain-cli account newkeys --strength␣
↪1  -o user"
$ export address=`cat user/address`
$ docker exec -it xchain1.node.com sh -c "./xchain-cli transfer --to $address --
↪amount 1000000000000"

# 创建合约账户
$ docker exec -it xchain1.node.com sh -c "./xchain-cli account new --account
↪$contractAccount --fee 1000 --keys ./user"

# 给合约账户转 token
$ docker exec -it xchain1.node.com sh -c "./xchain-cli transfer --to XC$
↪{contractAccount}@xuper --amount 100000000000 --keys ./user"

# 将合约拷贝到容器中
$ docker cp ./contract/$contractName xchain1.node.com:/home/work/xchain/
↪$contractName
```

(下页继续)

(续上页)

```
# 安装合约
$ docker exec -it xchain1.node.com sh -c "./xchain-cli native deploy --account XC$
↪{contractAccount}@xuper --runtime go -a '{\"creator\":\"XC${contractAccount}
↪@xuper\"}' --cname $contractName ./contract/$contractName --fee 19267894 --keys␣
↪./user --fee 19597986"

# 查询合约安装的状态
$ docker exec -it xchain1.node.com sh -c "./xchain-cli contract query
↪$contractName"
```

2. 部署 XuperDB

   数据持有节点将自己的隐私数据进行加密、切分、副本复制后分发到存储节点，存储节点是数据存储的物理节点。这里部署三个存储节点和两个数据持有节点，两个数据节点模拟分别提供部分数据的两方。

   修改配置文件：

```
$ vim PaddleDTX/testdata/xdb/data1/conf/config.toml
$ vim PaddleDTX/testdata/xdb/data2/conf/config.toml

# 使用在区块链部署时创建的合约账户、合约以及助记词
[dataOwner.blockchain]
type = "xchain"
[dataOwner.blockchain.xchain]
    mnemonic = "提 现 详 责 腐 贪 沉 回 涨 谓 献 即"
    contractName = "paddlempc"
    contractAccount = "XC1234567890123456@xuper"
    chainAddress = "xchain1.node.com:37101"
    chainName = "xuper"
```

```
$ vim xdb/storage1/conf/config.toml
$ vim xdb/storage2/conf/config.toml
$ vim xdb/storage3/conf/config.toml

# 使用在区块链部署时创建的合约账户、合约以及助记词
[storage.blockchain]
type = "xchain"
[storage.blockchain.xchain]
    mnemonic = "提 现 详 责 腐 贪 沉 回 涨 谓 献 即"
    contractName = "paddlempc"
    contractAccount = "XC1234567890123456@xuper"
    chainAddress = "xchain1.node.com:37101"
    chainName = "xuper"
```

启动服务：

```
$ cd PaddleDTX/testdata/xdb
$ docker-compose -f docker-compose.yml up -d
```

查看存储节点列表：

```
$ docker exec -it dataowner1.node.com sh -c "./xdb-cli nodes list --host http://
↪dataowner1.node.com:80"
```

注意：如果用户想启动基于 *fabric* 的 *XuperDB* 服务，可参考*XuperDB* 服务启动和命令使用说明。

3. 部署 Distributed AI

部署两个任务执行节点，模拟由两方组成的多方安全计算网络，两个任务执行节点分别对应不同的数据持有节点。

修改配置文件：

```
$ vim PaddleDTX/testdata/executor/node1/conf/config.toml
$ vim PaddleDTX/testdata/executor/node2/conf/config.toml
# 使用在区块链部署时创建的合约账户、合约以及助记词
[executor.blockchain]
type = 'xchain'
[executor.blockchain.xchain]
    mnemonic = "提 现 详 责 腐 贪 沉 回 涨 谓 献 即"
    contractName = "paddlempc"
    contractAccount = "XC1234567890123456@xuper"
    chainAddress = "xchain1.node.com:37101"
    chainName = "xuper"
```

启动服务：

```
$ cd PaddleDTX/testdata/executor
$ docker-compose -f docker-compose.yml up -d
```

CHAPTER 6

客户端工具

PaddleDTX 各方使用的客户端工具详细使用说明如下：

- Distributed AI 计算需求方命令使用说明
- Distributed AI 任务执行方命令使用说明
- XuperDB 客户端命令使用说明

本章重点介绍使用 PaddleDTX 时的几个常用命令。

6.1 操作 XuperDB

6.1.1 创建客户端帐号

XuperDB 数据持有节点服务的访问依赖于授权的客户端公私钥，可通过如下命令创建客户端公私钥：

```
$ ./xdb-cli key genkey -o ./ukeys
```

6.1.2 授权客户端帐号

生成数据持有节点客户端公私钥后，需要服务端对客户端进行授权：

```
$ ./xdb-cli key addukey -o ./authkeys -u␣
↪339524f35fb86a85bc3f9eed2b6ffd976de08b2cd47953b6640912f16e6863f2123f057cfef1f7132072602255a5a39bf25
```

该命令会将客户端公钥添加到服务端授权白名单，只有授权通过的客户端才允许请求数据持有节点进行文件上传下载。授权记录存储在服务端的 authkeys 文件夹中，-u 取值为客户端公钥。

6.1.3 创建命名空间

使用 XuperDB 服务的第一步是在每一个数据持有节点创建文件存储的命名空间，使用如下命令：

```
$ ./xdb-cli files addns  --host http://127.0.0.1:8121 --keyPath './ukeys' -n␣
↪paddlempc  -r 2
```

可以通过替换 host 来实现请求不同的数据持有节点；–keyPath 默认取值’./ukeys’，从该文件夹中读取客户端的私钥; -n 参数为命名空间的名称；-r 参数为副本数，一般取大于 1 的数。

如果您使用 docker-compose 来部署网络，需要进入 docker 后执行命令，也可以使用 docker exec 命令，例如

```
$ docker exec -it dataowner1.node.com sh -c "./xdb-cli files addns  --host http://
↪dataowner1.node.com:80 -n paddlempc -r 2 --keyPath ./ukeys"
```

使用 listns 命令可以查看已有的命名空间：

```
$ ./xdb-cli files listns  --host http://127.0.0.1:8122
```

–keyPath 默认取值’./ukeys’，该命令从该文件夹中读取客户端的公钥

6.1.4 上传文件

执行训练任务和预测任务之前都需要上传对应的文件，文件上传需要请求对应的数据持有节点进行上传。

对于用户部署的环境，需要分别上传两方的样本文件和预测文件。

```
$ ./xdb-cli --host http://127.0.0.1:8121 files upload -n paddlempc -m train_dataA4.
↪csv -i ./train_dataA.csv --ext '{"FileType":"csv","Features":"id,CRIM,ZN,INDUS,CHAS,
↪NOX,RM", "TotalRows":457}'  -e '2021-12-10 12:00:00' -d 'train_dataA4' --keyPath ./
↪ukeys
# 命令返回
FileID: 01edba10-ef04-4096-a984-c81191262d03
```

通过修改 host 来指定不同的数据持有节点；–keyPath 默认取值’./ukeys’，从该文件夹中读取客户端的私钥；-n 为命名空间的名称；-m 为文件名称；-i 指定了上传的文件；–ext 指定了样本或者预测文件中的标签；-e 为文件在 XuperDB 中的过期时间；-d 为文件描述。

上传文件后，可以使用 getbyid 命令进行文件的查询：

```
$ ./xdb-cli files getbyid -i 01edba10-ef04-4096-a984-c81191262d03 --host http://127.0.
↪0.1:8121
```

相同的，如果使用 docker 的话，需要进入 docker 后执行命令，也可以使用 docker exec 命令。

6.1.5 授权查询与确认

计算需求方发布训练或预测任务后，任务执行节点可获取到需自身参与计算的任务，自动向样本文件持有方发起文件授权使用申请，此时数据持有节点可通过如下命令查询授权申请列表：

```
$ ./xdb-cli --host http://localhost:8121 files listauth -a␣
↪6a5ba56bccf843c591a3a32baa5aa76deebffe2695d48521799b77fb0a32e286ae493143560d1f548dd494bf266d4df393
↪-o␣
↪71c516458ef075609be6a7ebaeca23dc42a3ff3aa0597d0abd3843253da09ee5bcdc292d517617bb7eb610a5351ae92240
```

通过修改 host 来指定不同的数据持有节点；-a 为任务执行节点公钥；-o 为文件持有方公钥；

查询文件授权申请列表后，可以通过 confirmauth 命令进行文件授权使用确认：

```
$ ./xdb-cli files confirmauth --host http://127.0.0.1:8121 -e '2022-08-08 15:15:04' -
↪i b87b588f-2e46-4ee5-8128-888592ada4fd --keyPath ./ukeys
```

6.2 操作 Distributed AI

Distributed AI 的操作方分为两个角色，计算需求方和任务执行方，分别通过 requester-cli 和 executor-cli 两个命令行客户端进行操作。

6.2.1 创建计算需求方账户

```
$ ./requester-cli key genkey -o ./keys
```

6.2.2 查询任务执行节点列表

发布训练或预测任务时，计算需求方需指定任务执行节点，如下命令可以查询区块链网络上的任务执行节点：

```
$ ./requester-cli nodes list
```

6.2.3 发布训练任务

训练任务由计算需求方发起：

```
$ ./requester-cli task publish -a "linear-vl" -l "MEDV" --keyPath './keys' -t "train"␣
↪-n "房价预测任务v3" -d "hahahha" -p "id,id" --conf ./testdata/executor/node1/conf/
↪config.toml -f "01edba10-ef04-4096-a984-c81191262d03,21e5b591-9126-4df8-8b84-
↪72a682a46fc1" -e "executor1,executor2"
# 命令行返回
TaskID: fdc5b7e1-fc87-4e4b-86ee-b139a7721391
```

命令行各参数说明如下：

- -a: 训练使用的算法, 可选线性回归‘linear-vl’或逻辑回归‘logistic-vl’
- -l: 训练的目标特征
- –keyPath: 默认取值’./keys’, 从该文件夹中读取私钥, 计算需求方的私钥, 表明了计算需求方的身份, 可以用-k 参数直接指定私钥
- -t: 任务类型, 可选训练任务’train’或预测任务‘predict’
- -n: 任务名称
- -d: 任务描述
- -p: PSI 求交时使用的标签
- –conf: 使用的配置文件
- -f: 训练使用的文件 ID, 这里是一个列表, 指明了各个任务执行方需要使用的文件
- -e: 任务执行节点名称, 和-f 一一对应, 执行节点执行任务时, 分别取对应位置的样本文件

与 XuperDB 的使用方法一致，当使用 docker-compose 部署时需要进入容器执行命令或者使用 docker exec 命令，后续命令将不再赘述。

6.2.4 授权确认

计算需求方发布任务之后，各个任务执行节点会自动向数据持有节点发起文件授权使用申请，此时需要数据持有节点查询授权申请并进行确认，命令参考./xdb-cli files confirmauth

6.2.5 启动训练任务

当所有的任务执行节点对任务进行确认后，需要计算需求方触发启动命令的执行，训练任务的执行结果是产出一个预测模型。

```
$ ./requester-cli task start --id fdc5b7e1-fc87-4e4b-86ee-b139a7721391 --keyPath './
↪keys' --conf ./testdata/executor/node1/conf/config.toml
```

各参数说明如下：

- –id: 任务 id
- –keyPath: 默认取值’./keys’，从该文件夹中读取私钥，计算需求方的私钥，表明了计算需求方的身份，可以用-k 参数直接指定私钥
- –conf: 使用的配置文件

6.2.6 发布预测任务

训练任务执行完成后产出预测模型，计算需求方可以提交预测任务，为预测数据计算出预测结果。

```
$ ./requester-cli task publish -a "linear-vl" --keyPath './keys' -t "predict" -n
↪"房价任务v3" -d "hahahha" -p "id,id" --conf ./testdata/executor/node1/conf/config.
↪toml -f "01d3b812-4dd7-4deb-a48d-4437312a164a,e02b27a6-0057-4673-b7ec-408ad060c952"␣
↪-i fdc5b7e1-fc87-4e4b-86ee-b139a7721391  -e "executor1,executor2"
TaskID: a7dfac43-fa51-423e-bd05-8e0965c708a8
```

参数说明与发布训练任务差别在一个参数：

- -i: 指定训练任务的 ID，使用训练任务的产出

6.2.7 授权确认

计算需求方发布任务之后，各个任务执行节点会自动向数据持有节点发起文件授权使用申请，此时需要数据持有节点查询授权申请并进行确认，命令参考./xdb-cli files confirmauth

6.2.8 启动预测任务

任务被各任务执行节点确认后，由计算需求方启动预测任务。

```
$ ./requester-cli task start --id a7dfac43-fa51-423e-bd05-8e0965c708a8 --keyPath './
↪keys' --conf ./testdata/executor/node1/conf/config.toml
```

6.2.9 获取预测结果

预测任务执行成功后，计算需求方可以获取到预测的结果。

```
$ ./requester-cli task result --id a7dfac43-fa51-423e-bd05-8e0965c708a8 --keyPath './
↪keys' --conf ./testdata/executor/node1/conf/config.toml  -o ./output.csv
```

各参数说明如下：

- –id: 预测任务的 id
- –keyPath: 默认取值’./keys’, 从该文件夹中读取私钥, 计算需求方的私钥, 表明了计算需求方的身份, 可以用-k 参数直接指定私钥
- –conf: 指定使用的配置文件
- -o: 预测结果的导出文件

CHAPTER 7

案例应用-线性回归算法测试

在本节中，我们使用 PaddleDTX 解决波士顿房价预测问题，帮助您更好的理解 PaddleDTX。

您可以参考 快速安装 来准备 PaddleDTX 的环境。

7.1 案例简介

本案例中我们使用了来自 UCI 机器学习数据库中的波士顿房屋信息数据。该数据集统计了波士顿郊区不动产税、城镇人均犯罪率等共计 13 个特征指标和平均房价，我们通过机器学习找到特征指标和房价之间的关系，进而预测该地区房价，这是一个典型线性回归计算案例。

以下是数据集中的字段含义, 特征变量为:

- CRIM: 城镇人均犯罪率
- ZN: 住宅用地超过 25000 sq.ft. 的比例
- INDUS: 城镇非零售商用土地的比例
- CHAS: 边界是河流为 1，否则 0
- NOX: 一氧化氮浓度
- RM: 住宅平均房间数
- AGE: 1940 年之前建成的自用房屋比例
- DIS: 到波士顿 5 个中心区域的加权距离
- RAD: 辐射性公路的靠近指数

- TAX: 每 10000 美元的全值财产税率
- PTRATIO: 城镇师生比例
- B: 城镇中黑人比例
- LSTAT: 人口中地位低下者的比例

目标变量 (也称为标签变量) 为:

- MEDV: 房价中位数

我们从数据集中随机选取了部分数据作为测试集，其余为训练集，训练集为模型训练使用的样本数据，测试集用来验证我们的模型。同时, 我们又将数据集纵向拆分为两部分，每部分包含不同的特征变量，分别由不同的数据持有方 A 和 B 进行持有，通过 id 来标识同一条样本。本案例我们模拟分别持有部分特征变量数据的两方进行联合训练和预测。

具体样本文件说明如下:

1. 训练任务：任务执行节点 A 参与模型训练样本文件 train_dataA.csv ，任务执行节点 B 参与模型训练样本文件 train_dataB.csv
2. 预测任务：任务执行节点 A 参与模型预测样本文件 predict_dataA.csv ，任务执行节点 B 参与模型预测样本文件 predict_dataB.csv

7.2 测试脚本说明

本案例采用 paddledtx_test.sh 演示:

```
Usage:
  ./paddledtx_test.sh <mode> [-f <sample files>] [-m <model task id>] [-i <task id>]
    <mode> - one of 'upload_sample_files', 'start_vl_linear_train', 'start_vl_linear_
↪predict', 'start_vl_logistic_train'
         'start_vl_logistic_predict', 'tasklist', 'gettaskbyid'
      - 'upload_sample_files' - save linear and logistic sample files into XuperDB
      - 'start_vl_linear_train' - start vertical linear training task
      - 'start_vl_linear_predict' - start vertical linear prediction task
      - 'start_vl_logistic_train' - start vertical logistic training task
      - 'start_vl_logistic_predict' - start vertical logistic prediction task
      - 'tasklist' - list task in PaddleDTX
      - 'gettaskbyid' - get task by id from PaddleDTX
    -f <sample files> - linear or logistic sample files
    -m <model task id> - finished train task ID from which obtain the model, required␣
↪for predict task
    -i <task id> - training or prediction task id

  ./paddledtx_test.sh -h (print this message), e.g.:
```

(下页继续)

(续上页)

```
  ./paddledtx_test.sh upload_sample_files
  ./paddledtx_test.sh start_vl_linear_train -f 1ffc4504-6a62-45be-a7e3-191c708b901f,
↪f8439128-bebb-47c2-a04d-1121dbc087a4
  ./paddledtx_test.sh start_vl_linear_predict -f cb40b8ad-db08-447f-a9d9-628b69d01660,
↪2a8a45ab-3c5d-482e-b945-bc45b7e28bf9 -m 9b3ff4be-bfcd-4520-a23b-4aa6ea4d59f1
  ./paddledtx_test.sh start_vl_logistic_train -f b31f53a5-0f8b-4f57-a7ea-956f1c7f7991,
↪f3dddade-1f52-4b9e-9253-835e9fc81901
  ./paddledtx_test.sh start_vl_logistic_predict -f 1e97d684-722f-4798-aaf0-
↪dffe955a94ba,b51a927c-f73e-4b8f-a81c-491b9e938b4d -m d8c8865c-a837-41fd-802b-
↪8bd754b648eb
  ./paddledtx_test.sh gettaskbyid -i 9b3ff4be-bfcd-4520-a23b-4aa6ea4d59f1
  ./paddledtx_test.sh tasklist
```

7.3 上传样本文件

任务的发布与执行离不开样本文件，故在计算需求方发布任务之前，需确保数据持有方已上传各自所拥有的样本文件。

```
# 上传样本文件
$ sh paddledtx_test.sh upload_sample_files

# 命令返回
Vertical linear train sample files: 688e4a1b-e9bf-4bfe-a13c-23ebb1d82850,19d4d284-
↪6b1e-4a62-b421-40fdb6b7e787
Vertical linear prediction sample files: 9196f040-0743-4ae6-a1aa-b37f08c9bd3b,
↪6d34fa49-5aac-409f-8973-a648d9309378
Vertical logistic train sample files: 9fb28896-eb6c-48f2-b356-2ab342a2aa6d,8b79fddd-
↪3370-402c-ba9b-1f239156ec51
Vertical logistic prediction sample files: 96140537-8c7a-46cb-b2d3-0540e8cadc0e,
↪abacaded-afdd-419d-bc52-0d90b5641aa2
```

命令执行说明：

- **upload_sample_files** 命令会自动化执行如下 2 个步骤：
    1. 为数据持有方 A 与 B 分别创建文件存储所需的命名空间
    2. 上传数据持有方 A 与 B 拥有的波士顿房价预测和鸢尾花数据分类所需的训练及预测样本文件

每个步骤对应的客户端命令详情参考 操作 XuperDB 。

样本上传执行结果说明：

- Vertical linear train sample files 值为数据持有方 A 上传 train_dataA.csv 和数据持有方 B 上传 train_dataB.csv 后所得的波士顿房价训练样本文件 ID
- Vertical linear prediction sample files 值为数据持有方 A 上传 predict_dataA.csv 和数据持有方 B 上传 predict_dataB.csv 所得的波士顿房价预测样本文件 ID
- upload_files.csv 存储了波士顿房价训练和预测所需的样本文件 ID

查看 XuperDB 中的样本文件：

```
# 数据持有方A查询train_dataA.csv文件：
$ docker exec -it dataowner1.node.com sh -c './xdb-cli files getbyid -i 688e4a1b-e9bf-
↪4bfe-a13c-23ebb1d82850 --host http://127.0.0.1:80'

# 数据持有方B查询train_dataB.csv文件：
$ docker exec -it dataowner2.node.com sh -c './xdb-cli files getbyid -i 19d4d284-6b1e-
↪4a62-b421-40fdb6b7e787 --host http://127.0.0.1:80'

# 数据持有方A查询predict_dataA.csv文件：
$ docker exec -it dataowner1.node.com sh -c './xdb-cli files getbyid -i 9196f040-0743-
↪4ae6-a1aa-b37f08c9bd3b --host http://127.0.0.1:80'

# 数据持有方B查询predict_dataB.csv文件：
$ docker exec -it dataowner2.node.com sh -c './xdb-cli files getbyid -i 6d34fa49-5aac-
↪409f-8973-a648d9309378 --host http://127.0.0.1:80'
```

7.4 训练任务

发布和启动训练任务：

```
# -f 取值样本上传upload_sample_files命令返回的Vertical linear train sample files
$ sh paddledtx_test.sh start_vl_linear_train -f 688e4a1b-e9bf-4bfe-a13c-23ebb1d82850,
↪19d4d284-6b1e-4a62-b421-40fdb6b7e787

# 命令返回
Requester published linear train task: TaskID: 91d9c0b7-996b-4954-86e8-95048e91a3b8
```

命令执行说明：

- **start_vl_linear_train** 命令会自动化执行如下 3 个步骤：
  1. 计算需求方发布波士顿房价预测训练任务
  2. 数据持有方 A/B 授权任务执行节点 A/B 确认任务
  3. 计算需求方启动任务

每个步骤对应的客户端命令详情参考 操作 Distributed AI 。

查看训练任务：

```
$ sh paddledtx_test.sh gettaskbyid -i 91d9c0b7-996b-4954-86e8-95048e91a3b8
```

7.5 预测任务

发布和启动波士顿房价预测任务：

```
# -f 取值样本上传upload_sample_files命令返回的Vertical linear prediction sample␣
↪files, -m 取值为训练任务返回的TaskID
$ sh paddledtx_test.sh start_vl_linear_predict -f 9196f040-0743-4ae6-a1aa-
↪b37f08c9bd3b,6d34fa49-5aac-409f-8973-a648d9309378 -m 91d9c0b7-996b-4954-86e8-
↪95048e91a3b8

# 命令返回
Requester published linear prediction task: TaskID: 44e1cc47-4cb4-4d9c-a27e-
↪81949182d2a4
Root mean square error of Boston house price prediction is: 4.568173732971698
```

命令执行说明：

- **start_vl_linear_predict** 自动化执行如下 5 个步骤：
    1. 计算需求方发布波士顿房价预测任务
    2. 数据持有方 A/B 授权任务执行节点 A/B 确认任务
    3. 计算需求方启动任务
    4. 为计算需求方下载预测结果
    5. 计算模型的均方根误差

每个步骤对应的客户端命令详情参考 操作 Distributed AI 。

7.6 模型评估

通过预测任务，获得了模型的预测结果，我们通过计算预测值与真实值的均方根误差来评估模型优劣。

预测任务执行完成后，同时输出了波士顿房价预测模型的均方根误差，为 4.568173732971698。

CHAPTER 8

案例应用-逻辑回归算法测试

在本节中，我们使用 PaddleDTX 解决鸢尾花的分类问题，帮助您更好的理解 PaddleDTX。

您可以参考 快速安装 来准备 PaddleDTX 的环境。

8.1 案例简介

鸢尾花卉数据集是常用的分类实验数据集，也来源于 UCI 机器学习数据库，其中鸢尾花的种类分为三种：

- 山鸢尾 (Iris-setosa)
- 变色鸢尾 (Iris-versicolor)
- 维吉尼亚鸢尾 (Iris-virginica)

在每一条样本数据中包含了四项特征值和一个标签值，标签值为鸢尾花的种类，特征值包括：

- 花瓣长度 (Petal length)
- 花瓣宽度 (Petal width)
- 花萼长度 (Sepal length)
- 花萼宽度 (Sepal width)

鸢尾花的分类问题是二分类逻辑回归的经典案例。我们利用逻辑回归的算法构建模型，根据鸢尾花的花萼和花瓣大小来区分鸢尾花的品种。

我们从数据集中随机选取了部分数据作为测试集，其余为训练集，训练集为模型训练使用的样本数据，测试集用来验证我们的模型。同时，我们又将数据集纵向拆分为两部分，每部分包含不同的特征变量，分别由不

同的数据持有方 A 和 B 进行持有，通过 id 来标识同一条样本。本案例我们模拟分别持有部分特征变量数据的两方进行联合训练和预测。

具体样本文件说明如下：

1. 训练任务：任务执行节点 A 参与模型训练样本文件 train_dataA.csv ，任务执行节点 B 参与模型训练样本文件 train_dataB.csv
2. 预测任务：任务执行节点 A 参与模型预测样本文件 predict_dataA.csv ，任务执行节点 B 参与模型预测样本文件 predict_dataB.csv

8.2 测试脚本说明

参考 案例应用-线性回归算法测试-测试脚本说明

8.3 上传样本文件

参考 案例应用-线性回归算法测试-上传样本文件

样本上传执行结果说明：

- Vertical logistic train sample files 值为数据持有方 A 上传 train_dataA.csv 和数据持有方 B 上传 train_dataB.csv 所得的鸢尾花品种分类样本文件 ID
- Vertical logistic prediction sample files 值为数据持有方 A 上传 predict_dataA.csv 和数据持有方 B 上传 predict_dataB.csv 所得的鸢尾花品种分类的样本文件 ID
- upload_files.csv 存储了鸢尾花训练和预测所需的样本文件 ID

样本文件查看：

```
# 数据持有方A查询train_dataA.csv文件：
$ docker exec -it dataowner1.node.com sh -c './xdb-cli files getbyid -i 9fb28896-eb6c-
↪48f2-b356-2ab342a2aa6d --host http://127.0.0.1:80'

# 数据持有方B查询train_dataB.csv文件：
$ docker exec -it dataowner2.node.com sh -c './xdb-cli files getbyid -i 8b79fddd-3370-
↪402c-ba9b-1f239156ec51 --host http://127.0.0.1:80'

# 数据持有方A查询predict_dataA.csv文件：
$ docker exec -it dataowner1.node.com sh -c './xdb-cli files getbyid -i 96140537-8c7a-
↪46cb-b2d3-0540e8cadc0e --host http://127.0.0.1:80'

# 数据持有方B查询predict_dataB.csv文件：
$ docker exec -it dataowner2.node.com sh -c './xdb-cli files getbyid -i abacaded-afdd-
↪419d-bc52-0d90b5641aa2 --host http://127.0.0.1:80'
```

8.4 训练任务

发布和启动训练任务：

```
# -f 取值样本上传upload_sample_files命令返回的Vertical logistic train sample files
$ sh paddledtx_test.sh start_vl_logistic_train -f 9fb28896-eb6c-48f2-b356-
↪2ab342a2aa6d,8b79fddd-3370-402c-ba9b-1f239156ec51

# 命令返回
Requester published logistic train task: TaskID: 95104913-c6cc-4520-bab1-2be814f0d81e
```

命令执行说明

- **start_vl_logistic_train** 命令会自动化执行如下 3 个步骤:
    1. 计算需求方发布鸢尾花品种分类训练任务
    2. 数据持有方 A/B 授权任务执行节点 A/B 确认任务
    3. 计算需求方启动任务

每个步骤对应的客户端命令详情参考 操作 Distributed AI 。

查看训练任务：

```
$ sh paddledtx_test.sh gettaskbyid -i 95104913-c6cc-4520-bab1-2be814f0d81e
```

8.5 预测任务

发布和启动预测任务：

```
# -f 取值样本上传upload_sample_files命令返回的Vertical logistic prediction sample␣
↪files, -m 取值为训练任务返回的TaskID
$ sh paddledtx_test.sh start_vl_logistic_predict -f 96140537-8c7a-46cb-b2d3-
↪0540e8cadc0e,abacaded-afdd-419d-bc52-0d90b5641aa2 -m 95104913-c6cc-4520-bab1-
↪2be814f0d81e

# 命令返回
Requester published logistic prediction task: TaskID: f86ff72b-fb94-4f8a-9797-
↪9b42ae3ade84
Accuracy of Iris plants prediction is: 1.00
```

命令执行说明：

- **start_vl_logistic_predict** 自动化执行如下 6 个步骤:
    1. 计算需求方发布鸢尾花品种分类预测任务

2. 数据持有方 A/B 授权任务执行节点 A/B 确认任务
3. 计算需求方启动任务
4. 计算需求方下载预测结果
5. 计算模型的均方根误差
6. 计算预测的准确率

每个步骤对应的客户端命令详情参考 操作 Distributed AI 。

8.6 模型评估

通过预测任务，获得了模型的预测结果，我们将置信度设置为 0.5，通过计算预测值与真实值的分类准确率来评估模型优劣。

预测任务执行完成后，同时输出了鸢尾花品种分类模型的预测结果，在测试集上准确率为 100%。

CHAPTER 9

部署架构

PaddleDTX 主要由计算需求方、任务执行节点、数据持有节点、存储节点和区块链节点组成，部署架构如下图所示：

9.1 计算需求方 (Requester)

计算需求方是机器学习模型或预测结果的需求方，需求方可能没有存储能力、没有数据资源和计算资源。需求方利用 PaddleDTX 网络中的资源，完成自己所需的计算并获得最终结果。

9.2 任务执行节点 (Executor Node)

任务执行节点进行模型训练和预测任务的具体计算。在计算过程中，执行节点会自动向数据持有节点发起文件授权使用申请，只有数据持有节点确认授权，执行节点才拥有数据的使用权限，并利用授权的数据和计算资源参与多方计算。一个任务执行节点可向一个或多个数据持有节点发起数据使用申请。

9.3 数据持有节点 (DataOwner Node)

数据持有节点是数据的归属方，可以为使用方做数据的可信性背书。数据持有节点通常将数据存储于远程节点，通过数据加密切分和分发、数据迁移、网络健康状态监控等机制，保障己方数据的私密性、安全性和高可用。

9.4 存储节点 (Storage Node)

存储节点拥有丰富的存储资源，为数据持有方提供数据存储服务。存储节点通过健康状态更新、副本保持证明的挑战与应答等机制，证明自身的可靠性。

9.5 区块链节点 (Blockchain Node)

区块链节点记录 PaddleDTX 网络中流转的关键信息，包括数据摘要、网络节点的去中心化治理、计算任务等信息，保证数据的安全可信。

从海量隐私数据存储到多方数据联合建模，PaddleDTX 提供了全链路可信方案。在实际应用中，各方机构根据自身的需求和能力，部署某类服务节点，共建去中心化可信网络。

CHAPTER 10

Distributed AI

Distributed AI 是 PaddleDTX 的计算层。一方面，它实现了可信的多方安全计算网络（SMPC），支持多个学习过程并行运行。另一方面，作为一个可扩展框架，它可以持续集成多种联邦学习算法。

10.1 服务组件

Distributed AI 包含以下服务：

- Requester，计算需求节点，向区块链网络发布计算任务，从任务执行节点获取任务执行结果，并做可信性验证。
- Executor，任务执行节点，从区块链上获取要参与执行的任务，确认数据的使用权限后执行任务。做权限确认的过程也是数据持有节点对数据的可信性做背书的过程。

10.2 多方安全计算框架

PaddleDTX 实现的多方安全计算框架，具备以下特征：

- 任务维度动态构建计算网络
- 支持多个学习过程并行执行
- 可扩展，方便集成各种联邦学习算法
- 以区块链、隐私计算、ACL 技术为支撑，保证数据、模型的隐私性和可信性

10.3 可信联邦学习

PaddleDTX 中，联邦学习分为训练过程和预测过程。计算需求方通过发布训练任务，任务执行节点会向数据持有节点做数据可信性背书，继而触发训练过程，最终得到满足条件的模型。如果有预测需求，计算需求方发布预测任务，任务执行节点会向数据持有节点做数据可信性背书，继而触发预测过程，最终得到预测结果。目前已集成的算法及其原理和实现，在 crypto 部分有更多体现。

10.4 接口与消息定义

任务与相关接口定义

联邦学习过程接口定义

10.5 配置说明

Requester 配置

Executor 配置

10.6 命令行工具

Requester 命令使用说明

Executor 命令使用说明

CHAPTER 11

XuperDB

PaddleDTX 的 xdb 模块实现了基于区块链的去中心化存储服务 (XuperDB)，为分布式计算网络提供了数据存储支撑。

XuperDB 专注于解决海量的重要、高价值、隐私数据的安全存储问题，可以作为存储服务独立于 PaddleDTX 使用。

11.1 背景和目标

敏感数据存储、使用和监管方面存在如下问题：

- 敏感数据需要分散存储，防止泄露，当单 IDC 被入侵，攻击者也无法恢复原始数据；
- 敏感数据常用来做某些特定的分析和计算，在使用时不能泄露原始数据，支持数据可用不可见；
- 敏感数据使用需要得到严格的授权或监管，以备后续审计。

因此，亟需一个去中心化存储系统解决敏感数据的隐私保护、安全使用和监管审计问题。

11.2 特点和优势

XuperDB 具备高安全、高可用、可审计的特点：

- **高安全**：数据加密分片存储，有权限的用户才能恢复原始数据，且可以抵御存储节点串谋和单副本攻击；
- **高可用**：通过副本保持证明机制保证数据被安全存储，通过健康监控和文件迁移机制保证文件随时可恢复，且可以抵御单节点故障；
- **可审计**：数据和存储节点详情记录到区块链上，保证数据完整性，支持审计功能。

11.3 架构设计

XuperDB 系统架构如下图所示：

XuperDB 网络由三类节点构成：

- **数据持有节点** (DataOwner Node)：数据的归属方，有存储需求，将自己的隐私数据进行加密、切分、副本制作后分发到存储节点；
- **存储节点** (Storage Node)：有丰富的闲置的存储资源，可以提供存储服务，通过应答数据持有节点的挑战证明自己持有数据分片；
- **区块链节点** (Blockchain Node)：构成区块链网络，支撑整个存储网络的去中心化治理，数据持有节点和存储节点在副本保持证明、健康状态监控过程中，通过区块链网络实现信息交换。

11.4 功能介绍

XuperDB 系统目前主要具备文件上传下载、链上存证、副本保持证明、系统健康状态监控和文件迁移、资源访问控制等功能。我们将持续完善和新增更多功能。

11.4.1 1 文件多副本分片存储

为保证数据的安全高可用，一个原始文件在系统中常采用多副本存储，用户可根据文件的安全级别为其设置副本数量。

文件经过持有方加密、切分和二次加密等操作，分散并乱序分发给不同存储节点，且同一文件分片在不同存储节点中的存储内容是不同的，以此抵御单副本攻击，防止节点串谋。

11.4.2 2 文件详情上链

为便于网络监管和审计、保证数据完整性，文件成功上传后，需要将文件详情记录到区块链中，主要包括文件名称、文件 ID、文件所有者身份、文件结构信息密文 (切片顺序和哈希)、每个切片所在的存储节点、文件发布时间和过期时间等。

11.4.3 3 文件原文恢复

系统支持数据持有方从多个存储节点恢复原始数据，主要步骤包括：从链上获取文件结构加密信息、解密结构信息得到文件对应的切片顺序和所在存储节点列表、分别从相应存储节点拉取切片、按顺序组装切片获得原始数据。

11.4.4 4 副本保持证明

系统采用了副本保持证明挑战和应答机制，保证文件切片被安全存储。目前支持两种副本保持证明协议：基于梅克尔树的协议和基于双线性对映射的协议。前者空间复杂度高、计算复杂度低，后者空间复杂度低、计算复杂度高。

11.4.5 5 健康状态监控

系统支持存储节点和文件的健康状态监控：

- 存储节点健康状态根据节点的活跃度和副本保持证明应答成功比例来衡量，文件分发时会优先选择健康的存储节点；
- 文件健康状态根据每个切片的健康状态来衡量，每个切片的健康状态由其所在的存储节点健康状态决定。

11.4.6 6 文件迁移

为保证文件的安全和高可用，数据持有方会定期检查己方文件的健康状态，并将非健康的文件切片从不健康的存储节点迁移到健康节点，保证每个文件都处于健康且可随时恢复的状态。

11.4.7 7 资源访问控制

系统设计了一套统一的数据访问授权方案，将数据使用需求方与持有方解偶。各需求方向数据持有节点发起使用授权申请，持有节点可选择通过或拒绝，授权通过的需求方可直接从存储节点获取数据并解密恢复。整个授权流程基于智能合约，最小化授权粒度，保证在数据持有方安全可信授权的基础上，各需求方便捷的使用数据。

11.5 如何使用

XuperDB 的使用方法详见接口使用部分 。

CHAPTER 12

Crypto

PaddleDTX 的 crypto 模块实现了若干机器学习算法和对应的分布式改造，为分布式计算网络提供了算法支撑。

目前开源了纵向联邦学习算法，包括多元线性回归和多元逻辑回归。同时支持秘密分享、不经意传输、加法同态加密、隐私求交等联邦学习依赖的工具。

12.1 数据隐私保护

随着大数据和机器学习技术地不断发展，人工智能在各个领域实现了商业落地。然而，数据孤岛和数据隐私保护问题普遍存在。企业间渴望通过协作计算发挥各自数据的价值，又担心泄露用户敏感数据和企业机密信息。

为实现数据的跨企业协同计算，同时保护数据的隐私，联邦学习技术应运而生。联邦学习支持多个参与方利用己方数据共同建模和预测，数据在各方本地进行处理，通过中间参数的加密传输和计算，获得最终的模型或预测结果。在这个过程中，各方原始数据的隐私得到了保护，真正实现了数据可用不可见。

联邦学习分为横向联邦学习、纵向联邦学习和联邦迁移学习：

- **横向联邦学习**：参与方的特征重叠较多，而样本重叠较少。将样本按照横向切分，取出各方特征相同而样本不同的那部分数据进行训练；
- **纵向联邦学习**：参与方的样本重叠较多，而特征重叠较少。将样本按照纵向切分，取出各方样本相同而特征不同的那部分数据进行训练；
- **联邦迁移学习**：参与方样本与特征重叠都较少，该场景下不对数据进行切分，而是利用迁移学习来克服数据或标签不足的情况。

12.2 机器学习算法

PaddleDTX 目前已经开源多元线性回归和多元逻辑回归算法，决策树、神经网络等更丰富的机器学习算法即将开源。

12.2.1 1.1 多元线性回归

多元线性回归用来描述一个变量受多个因素影响，且他们的关系可以用多元线性方程表示的场景。如房屋价格受房屋大小、楼层数、周边环境等因素影响。

线性回归的模型可以用如下表达式描述：

$y = \theta 0 + \theta 1X1 + \theta 2X2 + \cdots + \theta nXn$

其中，目标特征值由 n 个变量乘以对应系数再加上常数项计算得到。学习过程就是通过迭代找到合适的系数，使得模型在训练集合的误差尽量小。

12.2.2 1.2 多元逻辑回归

不同于多元线性回归，多元逻辑回归的目标特征值是离散的，通常定义为 {1,0}，分别表示目标特征是否为指定值。如利用鸢尾花卉数据集，可以训练模型来判断给定的样本是否为山鸢尾。

逻辑回归的模型可以用如下表达式描述 (Sigmoid 函数)：

$y = 1 / (1 + e\text{-}\theta X)$

该模型是基于线性回归模型变化得到的，模型连续可导，且可以保证目标特征是 (0,1) 之间的数值，越接近 1 表明样本是指定值的概率越大。学习过程就是通过迭代找到合适的参数 θ，使得模型在训练集合的误差尽量小。

12.3 纵向联邦学习

PaddleDTX 目前已经开源两方的纵向联邦学习算法，包括多元线性回归和多元逻辑回归。多方横向联邦学习和多方纵向联邦学习相关算法即将开源，敬请期待。

纵向联邦训练和预测步骤如下：

12.3.1 2.1 数据准备

计算任务会指定参与方的样本数据，数据存在去中心化存储系统 (XuperDB) 中。任务启动前，任务计算方 (即数据持有方) 需要从 XuperDB 获取自己的样本数据。

12.3.2 2.2 样本对齐

纵向的模型训练或预测任务，都需要对数据进行样本对齐，即：根据各自样本 ID 找到多个参与方的样本交集，用交集样本进行训练或预测。

项目采用了 PSI(隐私求交) 技术，可以在不泄露各方样本 ID 的前提下，实现样本求交的功能。

12.3.3 2.3 训练过程

模型训练是多次迭代和交互的过程，依赖于两方数据的协同计算，需要双方不断传递中间参数来计算出各自的模型。

为保护样本数据的隐私，迭代过程的模型中间参数采用 Paillier 同态算法进行加密，Paillier 支持密文加法和数乘操作。

12.3.4 2.4 预测过程

预测任务需要指定模型，因此在预测任务启动前，指定的模型训练任务必须已经成功完成。模型分别存储在训练双方的本地，在预测时分别利用各自的模型进行计算，并汇总得到最终结果。

对于线性回归算法，预测结果汇总后需要进行逆标准化操作，这个操作只有拥有标签的那一方才能完成，因此需要将预测结果汇总到标签方，由标签方计算最终的预测结果，并将结果以文件的形式存到 XuperDB 中，供需求方获取。

CHAPTER 13

我们的团队

团队的主要成员来自百度超级链研发团队和联通链研发团队，具备区块链、隐私计算、大数据等多领域的技术背景。

如果您对我们的研究方向或者技术团队感兴趣，欢迎加入我们。点击投递简历

CHAPTER 14

参与开发 & 测试

欢迎用户和开发者帮助我们开发新功能，或者进行代码优化、文档更新和 bug 修复。

点击查看如何贡献

CHAPTER 15

参考文献

[1] Konečný J, McMahan H B, Yu F X, et al. Federated learning: Strategies for improving communication efficiency[J]. arXiv preprint arXiv:1610.05492, 2016.

[2] 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.

[3] Goodfellow I, Bengio Y, Courville A. Deep learning[M]. MIT press, 2016.

[4] Goodfellow I, Bengio Y, Courville A. Machine learning basics[J]. Deep learning, 2016, 1(7): 98-164.

[5] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]//International conference on the theory and applications of cryptographic techniques. Springer, Berlin, Heidelberg, 1999: 223-238.

[6] Lo H K. Insecurity of quantum secure computations[J]. Physical Review A, 1997, 56(2): 1154.

[7] Chen H, Laine K, Rindal P. Fast private set intersection from homomorphic encryption[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 1243-1255.

[8] Shamir A. How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612-613.

[9] https://xuper.baidu.com/n/xuperdoc/general_introduction/brief.html