五个数据点，求最优解。

将原本集中式求解的方法拆分成分布式迭代的方法，

5个数据点各自在链下本地求解各自局部最优解，将5个解送到链上求解出一个变量，如果这个变量满足某种条件则直接输出作为最终的答案，如果不满足则将该变量返回链下，与5个点继续求解局部最优。。。

如果在某次迭代的过程，b节点上链的数据并不是由上一个迭代的lambda计算的出的，或者a和b串通，它们上链的数据并不是各自的最优解，那么在链上计算前，我们需要设计一个验证的过程

结果 ADMM有恢复功能，即使在迭代过程发生恶意修改，增加迭代次数最后还是能达到最优解。

如果有节点恶意退出交易，故意不报数据或者宕机了

在联盟链里：等待时间超过（），自动撇去这个恶意节点，重新匹配计算，并且将该恶意节点列入黑名单，下次出清的时候不让它参加

1. **联盟链和私有链的灵活性**：联盟链和私有链通常更容易实施此类功能，因为它们通常具有更多的控制权，可以自定义共识规则和验证过程。
2. **公有链的挑战**：在公有区块链（如比特币和以太坊）上，实现此类功能可能更具挑战性，因为公有链的性质要求更加去中心化和开放。在这些网络中，通常更依赖于激励机制来防止节点的恶意行为，而不是直接的处罚措施。

测试网络的组件

在部署测试网络后，你可以花一些时间来检查其组件。运行以下命令列出在你的机器上运行的所有Docker容器。你应该会看到由network.sh脚本创建的三个节点：

与Fabric网络交互的每个node and user都需要隶属于一个organization，以便参与网络。测试网络包括two peer organizations，Org1和Org2。它还包括一个单一的orderer，负责维护网络的订购服务。

Peers:

Peers是任何Fabric网络的基本组件。Peers存储区块链账本并在将交易提交到账本之前验证它们。Peers运行智能合约，这些智能合约包含用于管理区块链账本上的资产的业务逻辑。

网络中的每个对等节点都需要隶属于一个organization。在测试网络中，每个组织都运营一个对等节点，分别是peer0.org1.example.com和peer0.org2.example.com。

每个Fabric网络还包括一个ordering service。虽然对等节点验证交易并将交易块添加到区块链账本中，但它们不负责决定交易的顺序或将其包含到新的区块中。在分布式网络中，对等节点可能远离彼此，并且没有共同的视图来确定交易的创建时间。就交易顺序达成共识是一个昂贵的过程，会为对等节点带来额外的负担。

订购服务允许对等节点专注于验证交易并将其提交到账本中。在订购节点接收到来自客户端的已背书的交易后，它们就会就交易的顺序达成共识，然后将它们添加到区块中。然后，区块被分发给对等节点，对等节点将这些区块添加到区块链账本中。

示例网络使用一个由订购组织操作的单节点Raft订购服务。你可以在你的机器上看到以orderer.example.com命名的orderer organization正在运行。尽管测试网络只使用了单个节点的订购服务，但生产网络会有多个订购节点，由一个或多个订购组织操作。不同的订购节点将使用Raft共识算法来就整个网络上的交易顺序达成一致。

创建通道

现在我们的机器上运行着peer and orderer nodes，我们可以使用脚本来创建一个用于Org1和Org2之间交易的Fabric通道。Channels是特定网络成员之间的私有通信层。Channels只能被受邀加入通道的组织使用，并对网络的其他成员是不可见的。每个通道都有一个独立的区块链账本。已受邀的组织通过“加入”通道来将其对等节点连接到通道，以存储通道账本并验证通道上的交易。

你可以使用`network.sh`脚本来创建Org1和Org2之间的通道，并将它们的对等节点加入到通道中。运行以下命令来创建一个名为`mychannel`的通道：

./network.sh createChannel

成功则：Channel 'mychannel' joined

你还可以使用`channel`标志来创建一个具有自定义名称的通道。例如，以下命令将创建一个名为`channel1`的通道：

./network.sh createChannel -c channel1

`channel`标志还允许你通过指定不同的通道名称来创建多个通道。在创建`mychannel`或`channel1`之后，你可以使用以下命令来创建第二个名为`channel2`的通道：

./network.sh createChannel -c channel2

在通道上启动链码

在创建通道之后，你可以开始使用智能合约与通道账本进行交互。智能合约包含管理区块链账本上资产的业务逻辑。网络成员运行的应用程序可以调用智能合约来在账本上创建资产，以及更改和转移这些资产。应用程序还可以查询智能合约以读取账本上的数据。

为了确保交易的有效性，使用智能合约创建的交易通常需要由multiple organizations签名才能提交到通道账本。Multiple signatures是Fabric的信任模型的一部分。要求多个组织对交易进行endorsements可以防止通道上的一个组织篡改其对等节点上的账本或使用未经同意的业务逻辑。为了签署交易，each organization需要在其peer上调用和执行智能合约，然后签署交易的输出。如果输出一致并已经获得足够多的组织签名，交易可以提交到账本。指定通道上需要执行智能合约的一组组织的策略被称为the endorsement policy，这是作为链码定义的一部分为每个链码设置的。

在Fabric中，智能合约以chaincode包的形式部署在网络上。chaincode被安装在the peers of an organization上，然后部署到通道中，然后可以用于endorse transactions和与区块链账本交互。在链码可以部署到通道之前，通道的成员需要就链码定义达成一致，以建立链码治理。当足够多的组织同意时，链码定义可以提交到通道，链码就可以开始使用了。

在使用`network.sh`创建了通道之后，你可以使用以下命令在通道上启动链码：

./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go

`deployCC`子命令将在`peer0.org1.example.com`和`peer0.org2.example.com`上安装`asset-transfer (basic)`链码，然后部署到指定的通道，使用`channel`标志（如果未指定通道，则默认为`mychannel`）。如果是首次部署链码，脚本将安装链码的依赖项。你可以使用`-ccl`标志来安装Go、typescript或javascript版本的链码。你可以在`fabric-samples`目录的`asset-transfer-basic`文件夹中找到`asset-transfer (basic)`链码。这个文件夹包含作为示例提供的链码，用于教程中突出Fabric特性。

export FABRIC\_CFG\_PATH=/home/yulin/blockchain/fabric/Hyperledger/peer

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

与网络交互

在启动测试网络之后，您可以使用`peer`命令行界面（CLI）与您的网络进行交互。`peer` CLI 允许您从命令行中调用已部署的智能合约、更新通道，或者安装和部署新的智能合约。

请确保您正在`test-network`目录中操作。如果您按照安装样本、二进制文件和Docker映像的说明进行操作，您将在`fabric-samples`存储库的`bin`文件夹中找到`peer`二进制文件。使用以下命令将这些二进制文件添加到您的CLI路径中：

export PATH=${PWD}/../bin:$PATH

您还需要设置`FABRIC\_CFG\_PATH`以指向`fabric-samples`存储库中的`core.yaml`文件：

export FABRIC\_CFG\_PATH=$PWD/../config/

您现在可以设置环境变量，以便将`peer` CLI 操作为 Org1：

# Environment variables for Org1

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

`CORE\_PEER\_TLS\_ROOTCERT\_FILE` 和 `CORE\_PEER\_MSPCONFIGPATH` 环境变量指向组织文件夹中的 Org1 加密材料。

如果您使用了 `./network.sh deployCC -ccl go` 来安装和启动资产转移（basic）链码，您可以调用（Go）链码的 `InitLedger` 函数来将初始资产列表放入分类帐上（如果使用 TypeScript 或 JavaScript，例如 `./network.sh deployCC -ccl javascript`，您将调用相应链码的 `InitLedger` 函数）。

运行以下命令来使用资产初始化分类帐。请注意，CLI 不访问 Fabric Gateway 对等节点，因此必须指定每个背书对等节点：

```bash

peer chaincode invoke -o <orderer\_endpoint> --tls --cafile $ORDERER\_CA -C mychannel -n basic --peerAddresses <peer0\_org1\_endpoint> --tlsRootCertFiles $CORE\_PEER\_TLS\_ROOTCERT\_FILE --peerAddresses <peer0\_org2\_endpoint> --tlsRootCertFiles $CORE\_PEER\_TLS\_ROOTCERT\_FILE --peerAddresses <peer0\_org3\_endpoint> --tlsRootCertFiles $CORE\_PEER\_TLS\_ROOTCERT\_FILE -c '{"Args":["InitLedger"]}' --waitForEvent

```

在上述命令中，请将以下参数替换为您的网络配置：

- `<orderer\_endpoint>`: 您的订购服务的端点地址。

- `<peer0\_org1\_endpoint>`, `<peer0\_org2\_endpoint>`, `<peer0\_org3\_endpoint>`: 您的组织1、组织2 和组织3 中的 Peer 节点端点地址。

- `$ORDERER\_CA`: 您的订购服务的TLS CA 证书文件路径。

这将使用资产初始化分类帐，确保替换上述命令中的占位符和路径，以适应您的具体网络配置。

运行以下命令来使用资产初始化账本。请注意，CLI不访问Fabric Gateway对等节点，因此必须指定每个背书对等节点：

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'

现在你可以使用CLI查询账本。运行以下命令来获取已添加到通道账本的资产列表：

peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'

当网络成员想要在账本上转移或更改资产时，会调用链码。使用以下命令通过调用`asset-transfer (basic)`链码来更改账本上资产的所有者：

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "$*{PWD}*/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "$*{PWD}*/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "$*{PWD}*/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["asset6","Christopher"]}'

由于`asset-transfer (basic)`链码的背书策略要求交易由Org1和Org2签名，因此链码调用命令需要使用`--peerAddresses`标志同时指定`peer0.org1.example.com`和`peer0.org2.example.com`。由于网络启用了TLS，命令还需要使用`--tlsRootCertFiles`标志引用每个对等节点的TLS证书。

在我们调用链码之后，我们可以使用另一个查询来查看调用如何改变了区块链账本上的资产。由于我们已经查询了Org1对等节点，现在可以利用这个机会来查询在Org2对等节点上运行的链码。设置以下环境变量以操作为Org2：

# Environment variables for Org2

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org2MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:9051

现在，你可以查询在`peer0.org2.example.com`上运行的`asset-transfer (basic)`链码：

peer chaincode query -C mychannel -n basic -c '{"Args":["ReadAsset","asset6"]}'

部署智能合约到channel

最终用户通过调用智能合约与区块链账本进行交互。在Hyperledger Fabric中，智能合约以称为chaincode的包的形式部署。希望验证交易或查询账本的组织需要在其对等节点上安装链码。在链码安装到加入通道的对等节点之后，通道成员可以将链码部署到通道，并使用链码中的智能合约来创建或更新通道账本上的资产。

使用称为the Fabric chaincode lifecycle的过程部署链码到通道。Fabric链码生命周期允许多个组织在可以用于创建交易之前达成共识，关于如何操作链码。例如，尽管endorsement policy指定哪些组织需要执行链码来验证交易，通道成员需要使用Fabric链码生命周期来达成共识关于链码的背书策略。有关如何在通道上部署和管理链码的更详细概述，请参阅Fabric链码生命周期。

注意：这些说明使用了v2.0版本引入的Fabric链码生命周期。如果你想使用以前的生命周期来安装和实例化链码，请访问Fabric文档的v1.4版本。

开始网络

我们将首先部署一个Fabric测试网络的实例。在开始之前，请确保您已按照"getting\_started"上的说明安装了必要的软件。使用以下命令导航到您的fabric-samples存储库本地克隆中的测试网络目录：

cd fabric-samples/test-network

出于本教程的目的，我们希望从一个已知的初始状态操作。以下命令将终止任何活动的或陈旧的Docker容器并删除先前生成的文件

./network.sh down

然后，您可以使用以下命令启动测试网络：

./network.sh up createChannel

createChannel命令创建了一个名为mychannel的通道，其中包括两个通道成员Org1和Org2。该命令还将属于每个组织的对等节点加入了通道。如果网络和通道创建成功，您可以在日志中看到以下消息被打印出来：

========= Channel successfully joined ===========

现在，我们可以使用Peer CLI来部署asset-transfer（basic）链码到通道，以下是使用的步骤：

1. 打包智能合约

在安装到对等节点之前，我们需要对链码进行打包。如果您要安装使用Go、JavaScript或TypeScript编写的智能合约，步骤会有所不同。

在打包链码之前，我们需要安装链码的依赖项。导航到包含JavaScript版本的资产转让（基本）链码的文件夹：

依赖项在asset-transfer-basic/chaincode-javascript目录中的package.json文件中列出。您应该花一点时间查看这个文件。以下是显示的依赖项部分：

"dependencies": {

"fabric-contract-api": "^2.0.0",

"fabric-shim": "^2.0.0"

package.json文件将Fabric contract类导入智能合约包中。您可以在文本编辑器中打开lib/assetTransfer.js文件，查看合同类被导入智能合约并用于创建asset-transfer（basic）类。

const { Contract } = require('fabric-contract-api');

class AssetTransfer extends Contract {

...

}

AssetTransfer类为智能合约中定义的函数提供了读取和写入数据到区块链账本的事务上下文：

async CreateAsset(ctx, id, color, size, owner, appraisedValue) {

const asset = {

ID: id,

Color: color,

Size: size,

Owner: owner,

AppraisedValue: appraisedValue,

};

await ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)));

}

要安装智能合约的依赖项，请从asset-transfer-basic/chaincode-javascript目录运行以下命令。

npm install

如果命令成功，JavaScript包将安装在node\_modules文件夹内。

既然我们已经安装了依赖项，我们可以创建链码包。返回到test-network文件夹中的工作目录，这样我们就可以将链码与其他网络构件一起打包。

您可以使用peer CLI来创建所需格式的链码包。peer二进制文件位于fabric-samples存储库的bin文件夹中。使用以下命令将这些二进制文件添加到您的CLI路径中：

export PATH=${PWD}/../bin:$PATH

您还需要将FABRIC\_CFG\_PATH设置为指向fabric-samples存储库中的core.yaml文件：

export FABRIC\_CFG\_PATH=$PWD/../config/

要确认您能够使用peer CLI，请检查二进制文件的版本。二进制文件的版本需要是2.0.0或更高版本才能运行本教程。

docker exec -it cli bash

后即可进入cli容器

peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go/ --lang golang --label basic\_1.0

org1:

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

安装：

peer lifecycle chaincode install basic.tar.gz

换成org2:

export CORE\_PEER\_LOCALMSPID="Org2MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:9051

安装：

peer lifecycle chaincode install basic.tar.gz

调用链码

查询链码ID：

peer lifecycle chaincode queryinstalled

在链码定义已经提交到通道之后，链码将在加入了安装该链码的通道的对等节点上启动。资产转移（basic）链码现在已经可以被客户端应用程序调用。使用以下命令在账本上创建一组初始资产。请注意，调用命令需要针对足够多的对等节点以满足链码背书策略。 （请注意，CLI无法访问Fabric网关对等节点，因此必须指定每个背书对等节点。）

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'