filecoin节点

节点类型

节点接口

```
import repo "github.com/filecoin-project/specs/systems/filecoin_nodes/
repository"
import filestore "github.com/filecoin-project/specs/systems/filecoin_f
iles/file"
import clock "github.com/filecoin-project/specs/systems/filecoin_nodes
/clock"
import libp2p "github.com/filecoin-project/specs/libraries/libp2p"
import message_pool "github.com/filecoin-project/specs/systems/filecoi
n_blockchain/message_pool"
import ipld "github.com/filecoin-project/specs/libraries/ipld"
import key_store "github.com/filecoin-project/specs/systems/filecoin_n
odes/key_store"

type FilecoinNode struct {
    Node          libp2p.Node

    Repository    repo.Repository
    FileStore     filestore.FileStore
    Clock         clock.UTCClock
    LocalGraph    ipld.GraphStore
    KeyStore      key_store.KeyStore
    MessagePool   message_pool.MessagePoolSubsystem
}
```

链验证器节点

```
type ChainVerifierNode interface {

  FilecoinNode

  systems.Blockchain
}
```

客户端节点

```
type ClientNode struct {
  FilecoinNode

  systems.Blockchain
  markets.StorageMarketClient
  markets.RetrievalMarketClient
  markets.MarketOrderBook
  markets.DataTransfers
}
```

存储旷工节点

```
type StorageMinerNode interface {
  FilecoinNode

  systems.Blockchain
  systems.Mining
  markets.StorageMarketProvider
  markets.MarketOrderBook
  markets.DataTransfers
}
```

检索旷工节点

```
type RetrievalMinerNode interface {
  FilecoinNode

  blockchain.Blockchain
  markets.RetrievalMarketProvider
  markets.MarketOrderBook
  markets.DataTransfers
}
```

中继节点

```
type RelayerNode interface {
  FilecoinNode

  blockchain.MessagePool
  markets.MarketOrderBook
}
```

网络接口

Filecoin节点使用Libp2p协议进行对等发现，对等路由和消息多播等。Libp2p是对等网络堆栈通用的一组模块化协议。节点彼此之间打开连接，并在同一连接上安装不同的协议或流。在最初的握手中，节点交换他们各自支持的协议，所有与Filecoin相关的/fil/...协议都将安装在协议标识符下。

这是Filecoin使用的Libp2p协议的列表。
Graphsync: 用于传输区块链和用户数据。
关于Graphsync
Gossipsub：区块头和消息通过Gossip PubSub协议广播，其中节点可以订阅区块链数据的主题并接收这些主题中的消息。当接收到与主题相关的消息时,节点将处理该消息并将其转发给也订阅同一主题的同级。
关于Gossipsub
Kademlia DHT：是一个分布式哈希表，在特定节点的最大查找数上具有对数范围。Kad DHT主要用于Filecoin协议中的对等路由以及对等发现。
参考实施
Bootstrap List：是新节点加入网络后尝试连接的节点列表。引导节点列表及其地址有用户定义。
Peer Exchange：是一种发现协议，使对等方可以针对所需对等方针对其现有对等方创建并发出查询
关于Peer Exchange
DNSDiscovery：(截至文档完成，正在设计与完善中) HTTPDiscovery：(截至文档完成，正在设计与完善中)
Hello：处理与Filecoin节点的新连接。这是环境协议（例如KademliaDHT）发现过程中的重要组成部分。

时钟

```
type UnixTime int64  // unix timestamp

// UTCClock is a normal, system clock reporting UTC time.
// It should be kept in sync, with drift less than 1 second.
type UTCClock struct {
    NowUTCUnix() UnixTime
}

// ChainEpoch represents a round of a blockchain protocol.
type ChainEpoch UVarint

// ChainEpochClock is a clock that represents epochs of the protocol.
type ChainEpochClock struct {
    // GenesisTime is the time of the first block. EpochClock counts
    // up from there.
    GenesisTime              UnixTime

    EpochAtTime(t UnixTime)  ChainEpoch
}
```

```
package clock

import "time"

// UTCSyncPeriod notes how often to sync the UTC clock with an authori
tative
// source, such as NTP, or a very precise hardware clock.
var UTCSyncPeriod = time.Hour

// EpochDuration is a constant that represents the duration in seconds
// of a blockchain epoch.
var EpochDuration = UnixTime(15)

func (_ *UTCClock_I) NowUTCUnix() UnixTime {
    return UnixTime(time.Now().Unix())
}

// EpochAtTime returns the ChainEpoch corresponding to time `t`.
// It first subtracts GenesisTime, then divides by EpochDuration
// and returns the resulting number of epochs.
func (c *ChainEpochClock_I) EpochAtTime(t UnixTime) ChainEpoch {
    difference := t - c.GenesisTime()
    epochs := difference / EpochDuration
    return ChainEpoch(epochs)
}
```

Filecoin假定系统参与者之间的时钟同步较弱。也就是说，系统依赖于参与者可以访问全局同

步时钟（可承受一定限度的漂移）。
Filecoin依靠此系统时钟来确保共识。具体来说，时钟是支持验证规则所必需的，该验证规则可防止块生产者使用未来的时间戳来挖掘块，并且阻止领导者选举的发生超出协议允许的频率。

时钟用途

使用FIlecoin系统时钟：
通过同步节点来验证传入块是否在给定时间戳的适当纪元内被挖出（请参见块验证）。这是可能的，因为系统时钟始终映射到唯一的纪元号，该纪元完全由创世块中的开始时间确定。
通过同步节点以放置来自未来纪元的数据块
通过挖掘节点以允许参与者在下一轮尝试领导者选举（如果当前轮中没有人产生障碍）的情况下保持协议的活跃性（请参阅储能共识）。

为了允许旷工执行上述操作，系统时钟必须：
1.相对于其他节点具有足够低的时钟漂移（sub 1s），以使不会在其他节点的迫切情况下的被认为是未来纪元的纪元中挖掘区块（这些块直到根据验证规则的适当纪元/时间才被验证）。
2.设置节点初始化的时期数等
于 `epoch = Floor[(current_time - genesis_time) / epoch_time]`
预计其他子系统将从时钟子系统注册到NewRound () 事件。

时钟要求

用作Filecoin协议一部分的时钟应保持同步，且漂移应小于1秒，以便进行适当验证。
可以预期，计算机时钟晶体的漂移速率约为1ppm（即每秒1微秒或每周0.6秒），因此，为了满足上述要求：
客户端应 `pool.ntp.org` 每小时查询NTP服务器（建议）以调整时钟偏斜。
我们建议以下之一：
`pool.ntp.org` （可以迎合特定区域）
`time.cloudflare.com:1234` （有关Cloudflare时间服务的更多信息）
`time.google.com` (有关Google Public NTP的更多信息)
`ntp-b.nist.gov` (NIST服务器需要注册)

采矿业务有强烈的动机来防止其时钟向前偏移一个以上的时间，以防止其提交的区块被拒绝。同样地，他们有动机来防止其时钟向后漂移超过一个纪元，从而避免将自己与网络中的同步节点分开。

未来的工作

如果以上任一指标显示随时间推移出现明显的网络倾斜，则Filecoin的未来版本可能会定期包含潜在的时间戳/历元校正周期。

当从异常链暂停中断恢复时（例如，所有实现都对给定的区块感到恐慌），网络可能会选择按

中断的“死区“规则来禁止在中断时期内编写区块，以防止与未挖掘时期相关的攻击媒介链重启。

Filecoin协议的未来版本可能会使用可验证延迟功能（VDF）来强制执行阻止时间并满足此领导者选举要求；我们选择明确假设时钟同步，知道硬件VDF安全性得到更广泛的证明为止。

文件和数据

Filecoin的主要目的是根据存储客户的文件和数据。本节详细介绍与处理文件，分块，编码，图形表示 `Pieces` ,存储抽象等相关的数据结构和工具。

文件

```
// Path is an opaque locator for a file (e.g. in a unix-style filesyst
em).
type Path string

// File is a variable length data container.
// The File interface is modeled after a unix-style file, but abstract
s the
// underlying storage system.
type File interface {
    Path()   Path
    Size()   int
    Close()  error

    // Read reads from File into buf, starting at offset, and for size by
    Read(offset int, size int, buf Bytes) struct {size int, e error}

    // Write writes from buf into File, starting at offset, and for size
    Write(offset int, size int, buf Bytes) struct {size int, e error}
}
```

FileStore-文件的本地存储

`FileStore` 是用来指任何底层系统或设备，其将Filecoin数据存储到一个抽象，它基于Unix文件系统语义，并包含的概念 `Paths` 。在这里使用这种抽象是为了确保Filecoin的实现使最终用户可以轻松地使用适合他们需求的基础替换底层存储系统。最简单的版本 `FileStore` 只是主机操作系统的文件系统。 ``` // FileStore is an object that can store and retrieve files by path. type FileStore struct { Open(p Path) union {f File, e error} Create(p Path) union {f File, e error} Store(p Path, f File) error Delete(p Path) error

```
// maybe add:
// Copy(SrcPath, DstPath)
```

}```

变化的用户需求

Filecoin用户的需求差异很大，许多用户（尤其是旷工）将在Filecoin的下方和周围实施复杂的存储架构。 `FileStore` 这里的抽象是为了使这些变化的需求易于满足。FIlecoin协议中的所有文件和扇区本地数据缓存都是通过此 `FileStore` 接口定义的，这使得实现易于交换，并且使最终用户可以轻松选择所选择的系统。

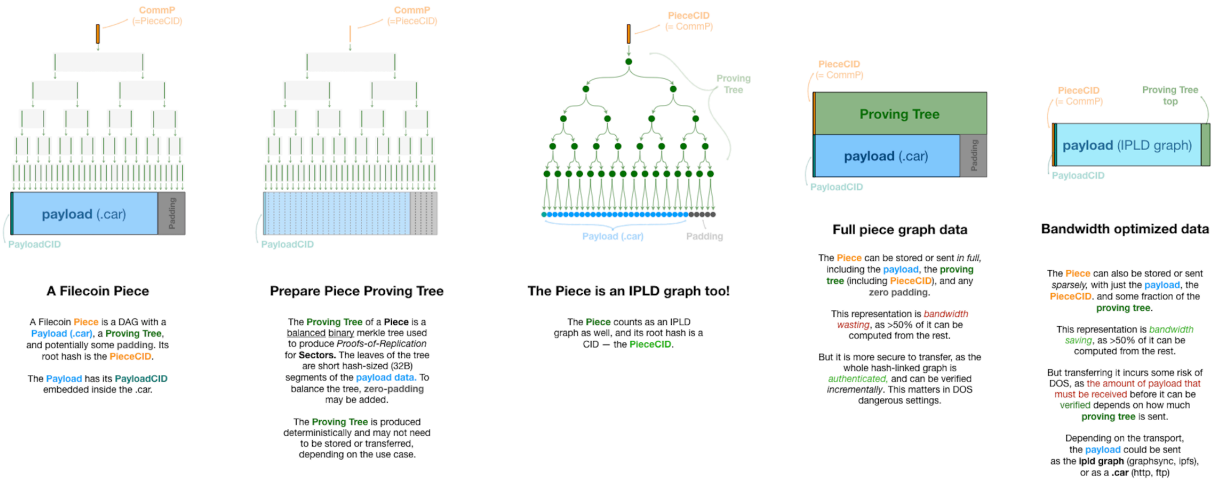
实施实例

该 `FileStore` 接口可以由多种后备数据存储系统来实现。例如：
1.主机操作系统文件系统
2.任何Unix/Posix文件系统
3.RAID支持的文件系统
4.联网的分布式文件系统（NFS，HDFS等）
5.IPFS
6.资料库
7.NAS系统
8.原始串行或块设备
9.原始硬盘驱动器（hdd扇区等）

实现对主机OS文件系统的支持，实现对其他存储系统的支持。

Piece文件的一部分

片段是代表文件整体或一部分的对象，供交易中的客户和矿工使用。 客户雇用矿工来存储碎片。片段数据结构设计用于证明存储任意IPLD图和客户端数据。该图显示了一个Piece的详细组成及其证明树，包括完整的和带宽优化的块数据结构。




```
import ipld "github.com/filecoin-project/specs/libraries/ipld"
```

```
// PieceCID is the main reference to pieces in Filecoin. It is the CID
// of the Piece.
type PieceCID ipld.CID

type NumBytes UVarint  // TODO: move into util

// PieceSize is the size of a piece, in bytes
type PieceSize struct {
    PayloadSize   NumBytes
    OverheadSize  NumBytes

    Total()       NumBytes
}

// PieceInfo is an object that describes details about a piece, and al
lows
// decoupling storage of this information from the piece itself.
type PieceInfo struct {
    ID    PieceID
    Size  PieceSize
    // TODO: store which algorithms were used to construct this piece.
}

// Piece represents the basic unit of tradeable data in Filecoin. Clie
nts
// break files and data up into Pieces, maybe apply some transformatio
ns,
// and then hire Miners to store the Pieces.
//
// The kinds of transformations that may ocurr include erasure coding,
// encryption, and more.
//
// Note: pieces are well formed.
type Piece struct {
    Info       PieceInfo

    // tree is the internal representation of Piece. It is a tree
    // formed according to a sequence of algorithms, which make the
    // piece able to be verified.
    tree       PieceTree

    // Payload is the user's data.
    Payload()  Bytes

    // Data returns the serialized representation of the Piece.
    // It includes the payload data, and intermediate tree objects,
    // formed according to relevant storage algorithms.
```

```
    Data()     Bytes
}

// // LocalPieceRef is an object used to refer to pieces in local stor
age.
// // This is used by subsystems to store and locate pieces.
// type LocalPieceRef struct {
//   ID   PieceID
//   Path file.Path
// }

// PieceTree is a data structure used to form pieces. The algorithms i
nvolved
// in the storage proofs determine the shape of PieceTree and how it m
ust be
// constructed.
//
// Usually, a node in PieceTree will include either Children or Data,
but not
// both.
//
// TODO: move this into filproofs -- use a tree from there, as that's
where
// the algorightms are defined. Or keep this as an interface, met by o
thers.
type PieceTree struct {
    Children  [PieceTree]
    Data      Bytes
}
```

PieceStore-存储和索引件

A `PieceStore` 是可以用来存储和从本地存储中检索片段的对象。在 `PieceStore` 另外保持件的索引。

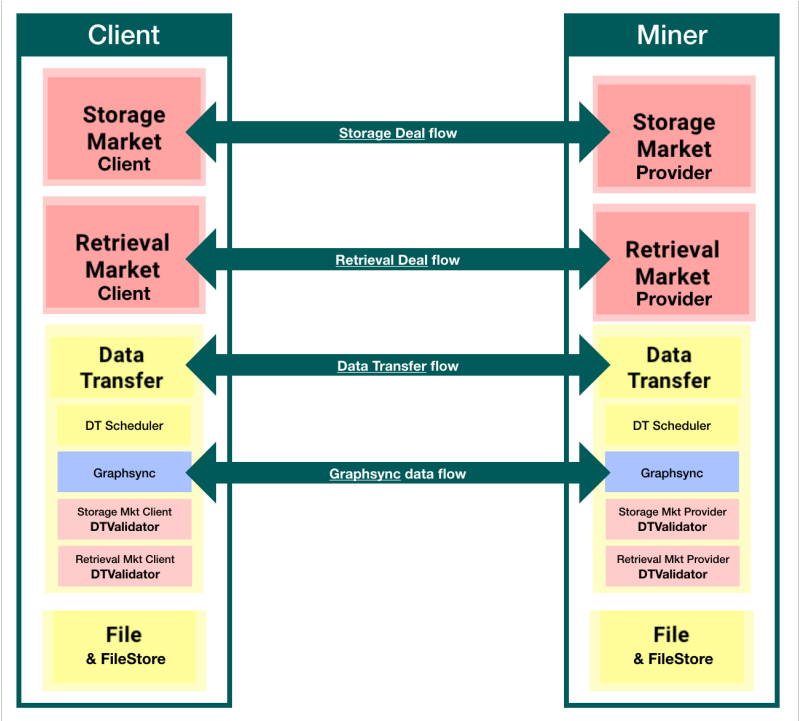
```
import ipld "github.com/filecoin-project/specs/libraries/ipld"

type PieceID UVarint

// PieceStore is an object that stores pieces into some local storage.
// it is internally backed by an IpldStore.
type PieceStore struct {
    Store               ipld.GraphStore
    Index               {PieceID: Piece}

    Get(i PieceID)      struct {p Piece, e error}
    Put(p Piece)        error
    Delete(i PieceID)   error
}
```

Filecoin中的数据传输

数据传输是一种用于 `Piece` 在进行交易时跨网络传输全部或部分网络的系统。

模组

此图显示了数据传输及其模块如何与存储检索市场相匹配。特别要注意，如何将来自市场的数据传输请求验证器插入“数据传输“模块，但其代码属于市场系统。



术语

1.**推送请求**：向对方发送数据的请求
2.**拉取请求**：请求对方发送数据的请求

3.**请求者**：发起数据传输请求的一方（无论推还是拉）
4.**响应者**：接收数据传输请求的一方
5.**数据传输凭证**：围绕存储或检索数据的包装，可以识别和验证向另一方的传输请求
6.**请求验证器**：仅当响应者可以验证请求是否直接与现有存储协议或检索协议绑定时，数据传输模块才启动传输。验证不由数据传输模块本身执行。而是由请求验证器检查数据传输凭证以确定是否响应请求。
7.**调度程序**：协商和确认请求后，实际的传输将由双方的调度程序管理。调度程序是数据传输模块的一部分，但与协商过程隔离。它可以访问基础可验证的传输协议，并使用它来发送数据和跟踪进度。
8.**Subscriber**：一个外部组件，通过订阅数据传输事件（例如进度或完成）来监视数据传输的进度。
9.**GraphSync**：调度程序使用的默认基础传输协议。完整的graphsync规范可以在 https://github.com/ipld/specs/blob/master/block-layer/graphsync/graphsync.md中找到

请求阶段

任何数据传输都有两个基本阶段：

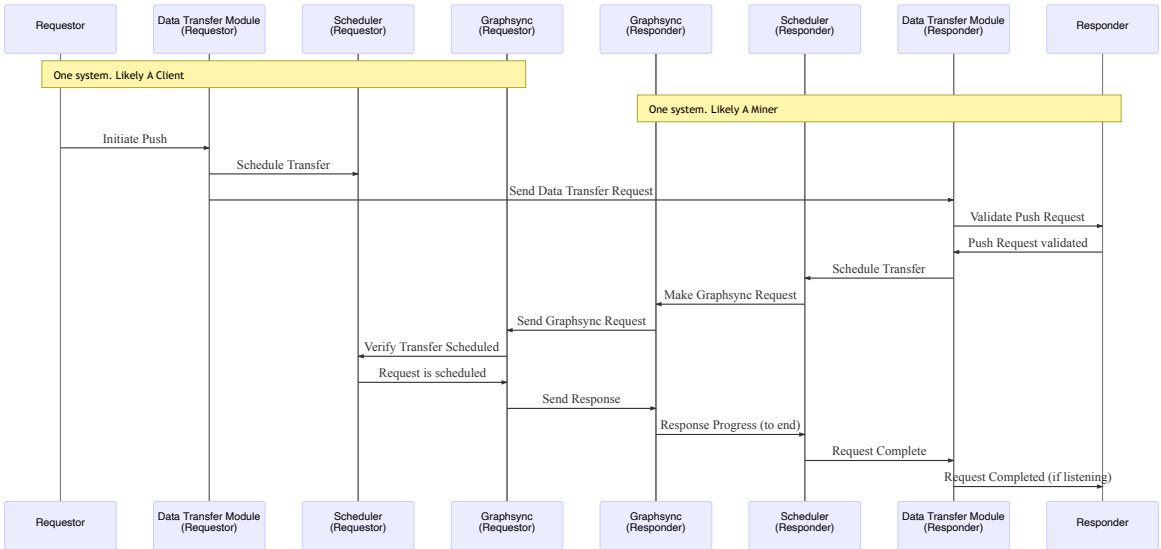
1.协商-请求者和响应者通过使用数据传输凭证进行验证来同意传输
2.传输-双方协商并达成协议后，数据实际上已传输。用于进行传输的默认协议是Graphsync

请注意，“协商”和“转移”阶段可以发生在单独的往返行程中，也可能在相同的往返行程中，其中请求方通过发送请求隐式同意，而响应方可以同意并立即发送或接收数据。

流程示例

推流

数据传输-推送流程图：



1.当请求者想要将数据发送给另一方时，它会发起“推”传输。

2.请求者的数据传输模块将把推送请求与数据传输凭证一起发送给响应者。还将数据传输放入调度程序队列中，这意味着它期望响应者在请求被验证后就开始传输
3.响应方的数据传输模块通过验证方验证数据传输请求，该验证方作为响应方提供的依赖项
4.响应者的数据传输模块安排传输
5.响应者对数据进行GraphSync请求
6.请求者收到graphsync请求，验证它在调度程序中，然后开始发送数据
7.响应者接收数据并可以显示进度指示
8.响应者完成接收数据，并通知所有侦听器

推送流程是存储交易的理想选择，其中客户一旦确认交易已签署并在链上就启动推送