数据重建及数据验证

版本日志

Version	Comments	Owner
pre-v	需求讨论	Yue, Xing, {Yotta}
pre-v0.1	+讨论结果，功能细化以及接口设计	Yue, Xing, {Yotta}
v0.1	数据重建模块实现	Yue, Xing
v0.2	数据抽样模块实现	Yue, Xing
v0.2b	1. 更新工程组织结构和config说明 2. 更新到新的ytfs p2p接口	Yue, Xing

需求说明

相关参数解释

DF:数据分片表，记录每个数据分片的存储信息 PND:一个数据块内最多容许掉线存储节点的数量。PNR:Parameter of Number of Rebuild nodes，需用多少个节点来重建故障节点 EC: Erasure Coding 纠删码

数据重建（来自黄皮书）

(5)数据重建 数据重建(MVP 最好实现) 当一个存储节点出现故障后，就将其数据转存到其他节点，具体如下:

1. 故障节点所属超级节点 BPM 另外分配 PNR(例如 100)个存储节点，每个节点重建 1/PNR的数据。
2. 包括该分片 hash 值、所属数据块的其它分片 信息(序号、hash 值、存储节点 ID 和访问地址);将这些信息组成重建列表，平均分配给所有重建节点。
3. 重建节点收到重建任务后，读取其它分片信息，还原出丢失的数据分片，存入本地。将重建结果反馈给超级节点 BPM，其中可能的错误包括溢出(Hash 值位于已经满的分组中)或还原数据出错等。
4. 对完成数据重建的数据分片，超级节点 BPM 更新该数据分片的存储节点信息，并相应增加该存储节点的单位收益;对于故障节点应该在调用本流程前就已经将其单位收益清 为零，并可能还会有进一步的处罚措施;对于因为溢出等可恢复错误而不能重建的数据 分片，超级节点 BPM 将其重建任务分配给另外的存储节点，直到所有数据分片全部重 建完毕;万一有实在无法重建的数据，向超级节点管理员报警。

数据验证（来自黄皮书）

(4) 存储共识: 采用改良心跳法，将每个节点主动写心跳信号，变为每个节点被抽查方式形成心跳信号，

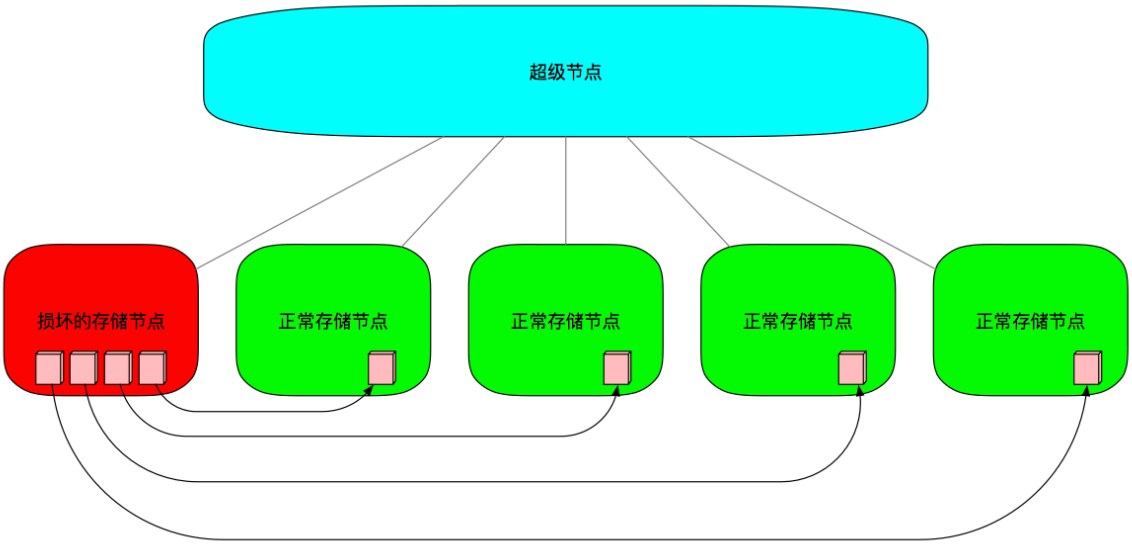
从而解决存储节点不可信的问题。该方法简单易行，可靠性足够好(即使每个小时才抽查一 次，如果一个存储节点保存了一半数据，那也只有不到 400 万分之一的概率能活过第一天 而不被发现)，而且工程实现简单，对存储设备资源损耗低，可以有效降低系统成本。具体 方法为:

1. 超级节点定期向其管辖的具备抽查能力的节点(不需要中继节点即可直接访问所有存储 节点的节点)发送被抽查名单(一次可以发送多个被抽查节点 ID 及其访问地址，以及 被抽查的数据分片的 hash 值)，被抽查名单是随机生成的。每个抽查周期 PLL(例如 1 小 时)每个存储节点都要被抽查一次(每次生成抽查名单都采用遍历所有存储节点，随机产 生对应抽查节点的方式)，然后再随机指定存储在该存储节点上的被抽查数据分片。
2. 抽查节点接收到超级节点下发的抽查指令后，将抽查名单加入到抽查队列中;
3. 抽查节点在每个抽查周期从抽查队列中取一项，向被抽查节点发送数据读取指令，读取指定的数据分片，然后验证该数据分片的 hash 值是否与指定的 hash 值相同。
4. 在下一次超级节点发送抽查名单时，抽查节点向超级节点反馈之前的抽查结果。对于未 通过抽查的存储节点，超级节点将进行核实。如果该存储节点通过了超级节点的核实(例 如连续抽查 100 次都正确)，则不处罚，但是留下记录供今后类似判断时使用。如果该 存储节点经核实属于恶意下线(例如 48 小时都不在线)，则没收押金，踢出存储节点名 单;如果经核实属于临时故障，但数据重建已经开始了，则仅扣除数据重建费用，该存储节点从零开始重新接单，无其它惩罚(这样允许不是特别稳定的存储节点加入进来， 从而降低整个系统的平均存储成本);如果在数据重建前恢复正常，则不做任何惩罚; 对于矿池，只要同时故障率低于 PFR(例如 5%)，则不做任何惩罚。

需求讨论

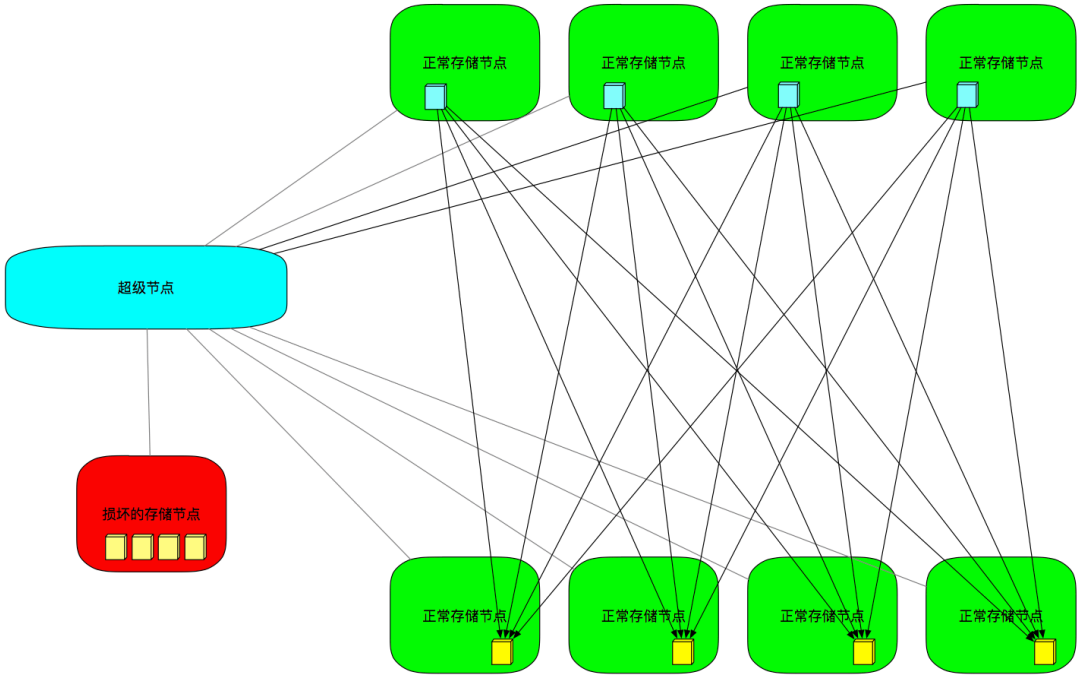
数据重建

1. 故障检测（另有模块处理？）
2. 重建模式

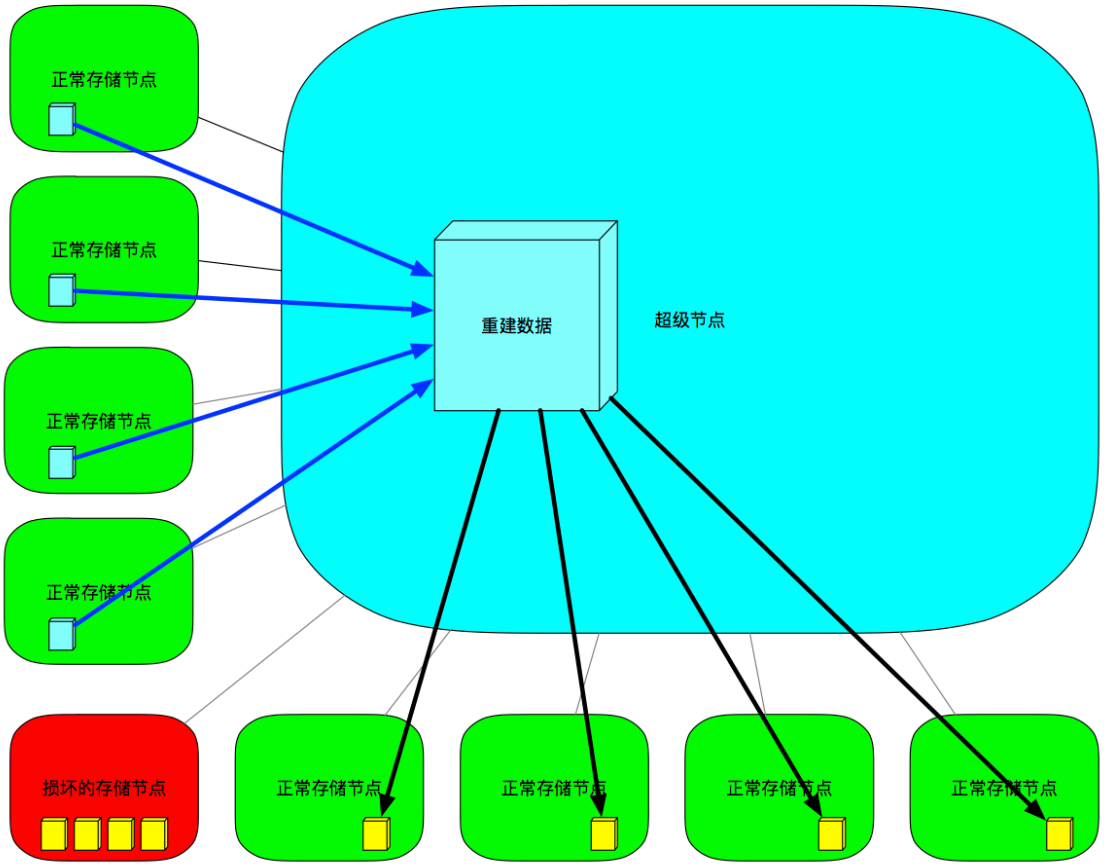


任务比较明确，如上图所示，但从PRD的角度看有两种不同的实现方式：

1. 由存储节点自己发起恢复请求，基本上超级节点只需要将任务描述下发，收到任务的节点自己从网络上下载需要的数据，然后完成恢复工作。副作用是数据传输冗余比较大，基本上要将EC需要的所有数据在每一个收到任务的节点上都传一次。



2. 由超级节点恢复数据并将恢复后的数据存储到正常节点，好处是避免数据传输，坏处是增加了超级节点的开销。



从黄皮书的描述看采用的第1种方式，要不要考虑方案2?

`[Yotta]用超级节点恢复数据的方案2并没有带来价值，因为超级节点自己也没有数据，也需要从其它数据分片读取数据还原丢失的数据，和重建节点来做是一样的，但重建节点是分散的，可以分散网络和计算负载`

3. 对于数据重建的关键EC编码算法选择，有很多的编码标准，YottaChain推荐哪一种?

`[Yotta]EC编码用的是RS。RS编码的算法库已经有了，可以复用`

数据验证

主要问题来源于黄皮书对抽查的描述：

`抽查节点在每个抽查周期从抽查队列中取一项，向被抽查节点发送数据读取指令，读取指定的数据分片，然后验证该数据分片的 hash 值是否与指定的 hash 值相同。`

问题是抽查节点有没有作恶的可能？？ **`[Yotta]抽查是随机的，包括谁抽查谁也是随机的，每次都不一样，如果某个抽查节点作恶，掩盖了被抽查节点的问题，那下一次是另一个节点抽查，就会被发现作弊`**

流程

数据重建

接受重建任务（超级节点下发）--->抓取重建数据（涉及到P2P模块，接口？）--->数据恢复（调用RS lib函数）--->重新存储（本地的YTFS）

数据验证

本地验证

自己调用Scan或者Sample函数对自己存储的数据进行抽查

其他节点抽查

抽查节点从超级节点拿到hash，从数据节点拿到数据进行比对即可

接口定义

数据重建

1. `func RecoverData(td TaskDescription) TaskResponse` 接受重建任务，内部调用p2p模块拿到数据，再利用rs lib完成重建，然后存到本地的YTFS

   TaskDescription至少需要包括以下几个成员：

   ```
   PNR:Parameter of Number of Rebuild nodes，需用多少个节点数据块来重建故障数据块（似乎有歧义，对于M+N型的EC编码来说，最多允许N个数据丢失，恢复任意数据需要M个数据参与计算，但是M包含的数据可能存在同一个节点上，那么M>PNR）
   PND:一个数据块内最多容许掉线存储节点的数量（即上文提到的N，同样有N>PND的问题）
   [M+N]Hash: EC encoded data hash array，长度为M+N，然后nil或者全0hash用来表示需要重建的数据。
   [M+N]P2PAddress: EC encoded data location和hash一一对应
   ```

   TaskResponse主要报告状态，大概包含pending/handling/success/error。

   Error处理：

   RecoverData一旦失败，需要判断是哪个模块的问题，然后进行下一步处理：

~~可恢复的情况大约两种~~

1. ~~某个节点P2P失败，上报超级节点P2P失败，要求超级节点用备份数据节点信息重发RecoverData~~
2. ~~EC失败，这里有可能是其他节点给予的恢复数据有误（hash和data对不上?），同样上报超级节点，要求备份节点给数据~~

~~（如果task Desc里面包含备份节点，那就不需要上报超级节点自己可以处理这些状态了。）~~

v0.1补充：如果没有其他的备份存在，那么task Desc里面已经包含了所有的data+parity数据信息，如果恢复失败，就真的失败了，失败的原因要么是从M+N个p2p节点里面拿M个data失败，要么是EC恢复出来和hash对不上。

不可恢复的情况YTFS节点无法处理，只能上报超级节点

1. p2p数据拿不到（节点断网？？）
2. ec恢复不出来（ec错误）

2. `func RecoverStatus(td TaskDescription) TaskResponse` 查询某个重建任务的完成情况。考虑到重建可能是个比较耗时的任务，异步处理+查询结果模式可以减轻超级节点的负担。

数据验证

自检

1. ~~`func SampleData(Hash) bool`~~ ~~查询某个data在不在，判断hash是否相等~~
2. ~~`func SampleStorage(index) bool`~~ ~~抽查storage上某个位置的data，判断hash和index的一致性~~
3. ~~`func ScanStorage(Storage) bool`~~ ~~扫描整个storage，确认数据完整性，比较耗时肯定~~

[v0.1]暂时不考虑自检

其他节点抽查

1. ~~`func get(Hash) []byte`~~，~~直接调用YTFS的get拿到数据进行比对，直接返回data或者包装成heartbeat形式返回都可以~~

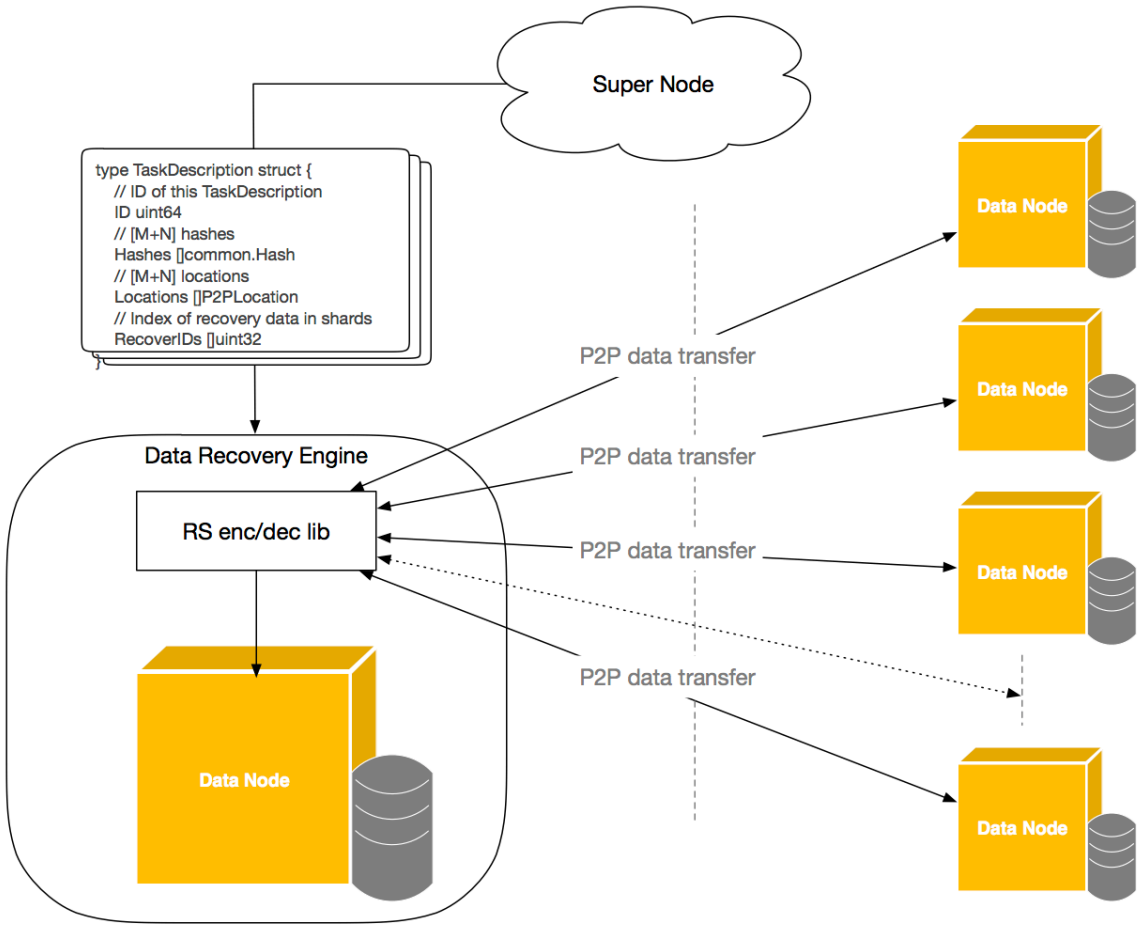
[v0.1]实现抽查模块，本质上是一个任务队列，然后按照一定的时间间隔完成任务（计划任务队列），接口和重建类似，一个提交任务，一个查询任务完成情况。

1. `func SampleDataRequest(td TaskDescription) error`
2. `func TaskStatus(td TaskDescription) TaskResponse`

实现细节

数据重建

原理图如下



RS lib 采用https://github.com/klauspost/reedsolomon

P2P模块现在没有，只是一个mock模块用来产生测试数据。

实现上采用异步带超时检测的任务队列模型，问题主要是在输入和输出的规模上，目前任务队列建在内存里面，如果任务量太大，可能对内存有压力，后期应该考虑轻量级数据库的辅助（v0.1为内存版本）。

性能上RS lib本身的性能非常好，参考github的data和我们自己的测试，所以瓶颈全部集中在P2P模块这里。

1. RS lib benchmark data from github

Here are the throughput numbers with some different selections of data and parity shards. For reference each shard is 1MB random data, and 2 CPU cores are used for encoding.

Data	Parity	Parity	MB/s	SSSE3 MB/s	SSSE3 Speed	Rel. Speed
5	2	40%	576,11	2599,2	451%	100,00%
10	2	20%	587,73	3100,28	528%	102,02%
10	4	40%	298,38	2470,97	828%	51,79%
50	20	40%	59,81	713,28	1193%	10,38%

2. DataRecoveryEngine Performance with mock P2P module（3data+4parity）

Data reconstruct benchmark (without YTFS writing)

Configuration	DataBlockSize	Mock P2P Bandwidth	Mock P2P delay	Single Data Rebuild time
fast	32KB	10Mb	250ms	250ms
slow		100Mb	25ms	25ms
uneven		10Mb ~ 100Mb	[250,211,173,136,99,62,25]ms	124ms

可见重建很快，时间都花在p2p数据传输上（不考虑数据写入YTFS）。

这里可以估算一下重建所需要的时间，假设1T的数据，10Mb的带宽，3data+4parity的条件下，从p2p网络拉取数据需要$\frac{1T*8b*3}{10Mb/s} = 0.8Ks = 800,000s$，也就是说，如果1T数据损失要在10Mb带宽的p2p网络上恢复，需要至少222个小时（10天）。如果考虑到写入YTFS的速度大约也是10Mb/s（1.5MB/s）这个水平，采用更大带宽的p2p网络恢复数据反而会卡在YTFS写上。

另外data+parity的选择也很重要，选择更大的data shard，比如5+2或者10+2这种，会增加p2p网络数据传输量。

实现

config设置：

Option	Vaule	Comments
DataShards	1～INTMAX / default 3	原始数据分片数
ParityShards	1 ~ INTMAX / default 4	校验数据分片数
MaxTaskInParallel	1 ~ INTMAX / default 12	并发任务数
TimeoutInMS	1 ~ INTMAX / default 5000	单个重建任务Timeout设置

工程按照Yotta的目录结构重新组织，放在https://github.com/yottachain/YTFS-DN/tree/master/dataRecovery

```
// DataRecoverEngine the rs recoverEngine to recovery data
type DataRecoverEngine struct {
    recoveryEnc reedsolomon.Encoder
    config      *DataRecoverOptions
    ytfs        *ytfs.YTFS

    p2phelper P2PNetworkHelper

    taskList   []*TaskDescription
    taskCh     chan *TaskDescription
    taskStatus map[uint64]TaskResponse

    lock sync.Mutex
}
// 构造函数：依赖本地的YTFS和当前P2P节点信息
```

```
func NewEngine(ytfs *ytfs.YTFS, selfNode ythost.Host, opt
*DataRecoverOptions) (*DataRecoverEngine, error) {...}

// 接口定义
// RecoverData recieves a recovery task and start working later on
func (recoverEngine *DataRecoverEngine) RecoverData(td *TaskDescription)
TaskResponse {...}
// RecoverStatus queries the status of a task
func (recoverEngine *DataRecoverEngine) RecoverStatus(td *TaskDescription)
TaskResponse {...}
```

测试

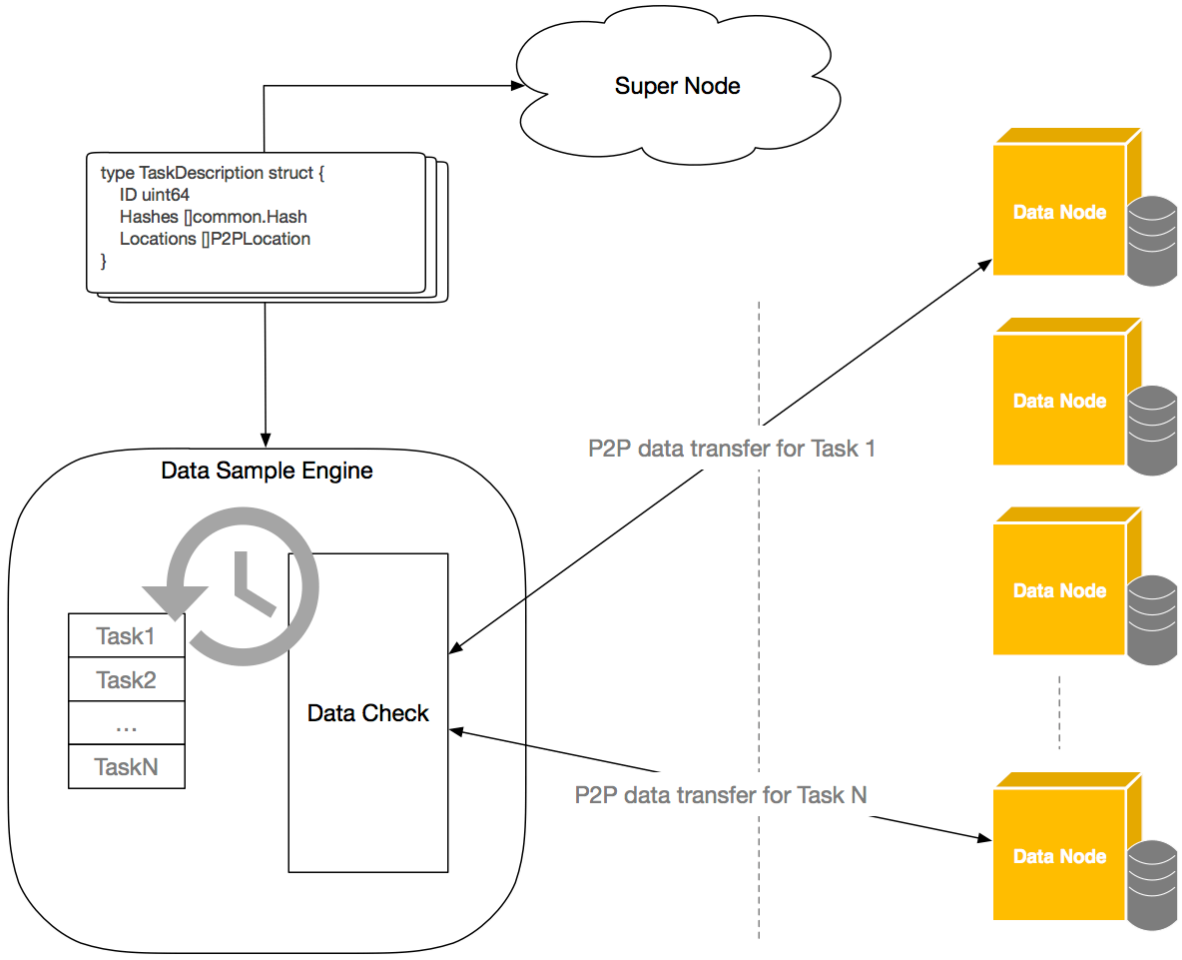
Recovery模块内部带有go test，使用的是mock p2p，具体逻辑参考https://github.com/yottachain/YTFS-DN/blob/master/dataRecovery/recovery/recovery_test.go

目前唯一的假设是数据节点的msgType，mock p2p简单注册了一个类型为“data”的data handler，如下代码所示：

```
const dataRetrieveMsgType string = "data"
...
//注册
{
        node.RegisterHandler(dataRetrieveMsgType, func(msg ythost.Msg)
[]byte {
            time.Sleep(delay * time.Millisecond)
            datahash := common.BytesToHash(msg.Content)
            return node.Data[datahash]
        })
}
...
//使用
{
    res, err := p2pHelper.self.SendMsg(p2pNodeInfo.ID, dataRetrieveMsgType,
dataHash[:])
    if err != nil {
        return res, err
    }
    return res, nil
}
```

在实际网络测试中需要使用真正的type string，把dataRetrieveMsgType改成对应的type即可。

数据验证



Data Check模块需要注意选择Hash函数，否则对不上。

另外Task的随机性是由发送方指定还是抽查方指定？目前假定是Task就是随机进来的，因此任务队列没有另外加入随机选择，也就是说内部TaskList目前是FIFO模式，也可以考虑做成可配置。

同样可能需要数据库支持（v0.2为内存版本）

和数据重建一样性能依赖P2P。（实现也依赖p2p，目前是mock的）

实现

工程目录：https://github.com/yottachain/YTChain-DN/tree/master/sampling

config设置：

OptionName	Value	Comments
FrequenceInSecond	1～INTMAX / default 300	抽样频率，多少秒进行一次抽样
MaxSamplingThread	1～INTMAX / default 1	抽样线程数，即同时进行的抽样任务
P2PTimeoutInMS	1～INTMAX / default 500	P2P timeout value，单位毫秒

代码说明：

```
// DataSampleEngine is the sampling task list manager
```

```
type DataSampleEngine struct {
    config     *Config
    p2p        P2PNetworkHelper
    taskList   []*TaskDescription
    taskCh     chan *TaskDescription
    taskStatus map[uint64]TaskRespond
    lock       sync.Mutex
}

// 构造函数，依赖当前P2P节点信息
// NewEngine creates a new sampling engine
func NewEngine(p2pHost p2pHost.Host, config *Config) (*DataSampleEngine,
error) {...}

// 接口定义
// RequestSampling is the interface of others to give in a sampling task
func (engine *DataSampleEngine) RequestSampling(task *TaskDescription) error
{...}

// ReportTaskStatus reports task status
func (engine *DataSampleEngine) ReportTaskStatus(task *TaskDescription)
TaskRespond {...}
```

注意，目前Sampling模块默认采用数据Hash算法为sha256，如果和实际测试网络不一致会导致sample报错。

测试

和Recovery模块一样，mock test直接运行go test即可，实网测试则需要使用实际在网络上注册的返回数据的的msgType用来拿到数据，比如实网注册的p2p handler entry是"/p2p/dataFunc/get"，那么对应的修改如下：

```
const dataRetrieveMsgType string = "/p2p/dataFunc/get"
```