# Wan 链管理

Jqg

## 最长链规则(maxvalid)

*  收到所有链,为本地的链,不超过父链 k 个块
* 计算最长链
* 如果有多个和  长度相同并且是最长链,则返回 C
* 如果  不是最长链,但有多个最长链,则返回  中的第一个(排序?)

## 链增长

* 初始化:
* Registration 接口 🡪 Ui{U1,U2,…,Un}
* 如果 Ui 是 sl1 的出块者,则发送 (genblock req, U i ) 给 ,返回 (genblock,S0 ,ρ,F)

Genblock: genisis block

S0: 初始 stake

ρ:随机数

F: Leader 选择函数

设置lcoal chain 为 C=B0=(S0, ρ), state 为 H(B0) B0: 位 block0,即 genblock?

* 如果非 sl1 出块者,从广播接口接收 chain C, 设置 local chain 位 C, state 为 H(HEAD(C))
* 链扩展规则

1. Index>2 的第 j 个新 epoch, 用户 Ui 在取 （j-1）R – 2k 个slot 对应的块状态数据,发送 (epochrnd req,Ui ,ej ) 到 , 获取到 (epochrnd,ρ j )

链增长率 1/2 之上，2k 保证会生成 k 个块

b) 用户收集所有有效的链到集合

* 验证每个链  (收到的链都有公共的 prefix?),
* 验证每个块 (在 C’ 中有 parent?,  ?)
* 每条链需要 （true）

 是当前 epoch j 的 stakeholder  的验证key

* 计算 ,设置  为localchain,设置链状态 

c) 如果用户 Ui为slotleader ，slotleader 通过  选出)

* 生成新块 

St:链状态

d: 数据

sl: 当前slot

: 

* Ui 计算 ,广播 .C 为本地的最长链
* 设置  为localchain,设置链状态 

## Epoch genesis

在每个 epoch 开始时创建 epoch genesis B0块，其内容包括：

{

protocolMagic byte[] //常量字符串, 不必要?

epochId uint //当前 epochId

preEpochLastestBlkHash Hash //上一个 epoch 最后一个块的 hash

slotLeaders byte[][] //当前 epoch slotleader 的所有 public key

genesisBlkHash Hash //该genesis 块 hash

extrea byte[ ] //空,备用

}

该块不需要上链,在线节点可以计算保存在本地数据库.

每个 epoch 生成的链为:

B0 {B1,B2,…….,Bn}

B0 为 epoch genesis 块,该块不广播,但如果有 peer 请求,可以为该 peer 提供.

## 一个 epoch 中链处理

一个 epoch 中

1. 在 K 块中缓存所有链的分叉信息
2. 每在缓冲的 K 个块中使用最长链规则确定新块所在链
3. 如果 statedb 中所采用的链与缓冲中选择的最长链不一致,则 reorg 链至新链



## 实现方案

### 方案

按照 cardano 方案, 每个epoch的第一个main block块的previous hash是指向自己的genesis block的,不是指向上个epoch最后一个main block hash. 每个epoch的genesis block的previous hash又是指向上个epoch的最后一个main block的hash

实现过程

1. 某个 epoch 第一个 slot 每个节点按照自己当前链的状态生成 epoch genesis 块,保存在内存中
2. 相应的 slot leader 生成对应的 block, block header 的 parent 指向 epoch genesis 块,广播该块
3. 其它节点接收到epoch 的第一个块后,检查 epoch genesis 块是否相同

* 如果相同,对前面 epoch 块不做处理,按照前面的链增长规则处理后面的块
* 如果不同,则向发出该 epoch 第一块的节点或其它 peer 请求该 epoch 的genesis 块,获取该 genesis 块后,根据 genesis 块中的 main block的hash信息和前面 epoch 的 k 个块的缓冲信息,对链进行 reorg. 当前 epoch 的增长仍旧按照 前面的链增长规则处理后面的块

流程:



### 实现步骤

* 第一阶段(POC?)

实现 maxvalid 和 链增长规则

* 第二阶段

实现向其它 peer 请求 epoch genesis, 创建 epoch genesis, 并根据 epoch genesis 确定前一个 epoch 最后 k 个块的状态

### 实现内容

* 第一阶段修改

1. 添加接口实现所有链的缓冲

type ForkMem interface{

maxvalid(blockNum) block

push(block)

delete(blockNumber)

getBlock(blockHash) block

}

1. 添加接口

func (bc \*BlockChain) SwithForkChain(chain types.Blocks) (int, error).

将当前工作链切换到另一条链

* 第二阶段

1. 在 downloader 和 fecher 中添加功能

RequestEpochGenesis(epochId)

SendEpochGenesis()

VerifyEpochGenesis()

1. 在 blockChain 添加

func (bc \*BlockChain) CheckEpochGenesis(chain types.Blocks) (int, error).

检查上一个 epoch

## Checkpoint

与 epoch genesis 配合实现 fast sync 功能