区块链共识机制介绍

科普版本

刘鹏

2019.02.21

PoET

背景

- Intel在Sawtooth Lake项目中所采用的算法。
- 本质是利用intel的硬件安全可信环境

基本原理

- Leader选举：每个节点都从可信的SGX环境中获得一个随机等待时间，这个等待时间与节点贡献到网络中的资源反比，即贡献越大，随机等待时间越短。
- 随机等待时间最短的节点，会被选举成为Leader的节点，打包出块。
- 选举和验证均是基于intel的CPU中提供的指令，intel保证其CPU是完全可信的，所以整个出块和验证的过程均为可信的。（潜台词是依赖于第三方）
- SGX的发展也可能出现漏洞，因此该共识依赖于专用硬件的安全性。

Paxos

解决问题

- 使得参与分布式处理的每个参与者逐步达成一致意见。
- 无拜占庭节点的前提假设，大多用于联盟链，无法用于公链的共识。

主要思想

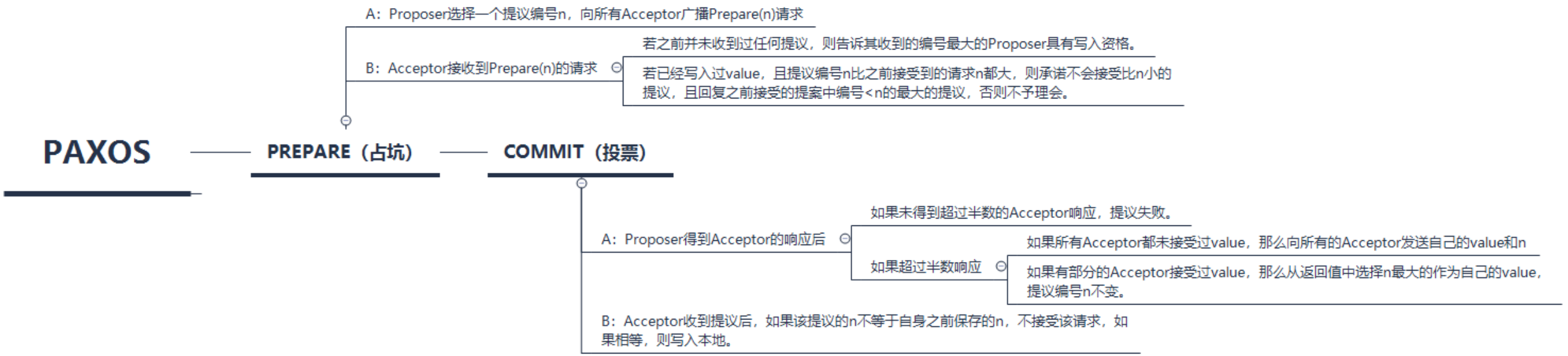
- 法定集合的性质，即两个法定集合一定存在交集。对，就这么简单。

通俗理解

- 第一阶段：因为存在多个“提议者”，如果都提意见，那么“接受者”接受谁的不接受谁的？太乱了。所以，要先明确哪个“提议者”是意见领袖有权提出提议，未来，“接受者”们就主要处理这个“提议者”的提议了。（目的是为了尽早形成多数派）
- 第二阶段：由上阶段选出的意见领袖提出提议，接受者反馈意见，如果多数接受者认可一个提议，那么提议就通过了。

Paxos

算法流程


PAXOS
PREPARE（占坑）
A：Proposer选择一个提议编号n，向所有Acceptor广播Prepare(n)请求
B：Acceptor接收到Prepare(n)的请求
若之前并未收到过任何提议，则告诉其收到的编号最大的Proposer具有写入资格。
若已经写入过value，且提议编号n比之前接受到的请求n都大，则承诺不会接受比n小的提议，且回复之前接受的提案中编号<n的最大的提议，否则不予理会。
COMMIT（投票）
A：Proposer得到Acceptor的响应后
如果未得到超过半数的Acceptor响应，提议失败。
如果超过半数响应
如果所有Acceptor都未接受过value，那么向所有的Acceptor发送自己的value和n
如果有部分的Acceptor接受过value，那么从返回值中选择n最大的作为自己的value，提议编号n不变。
B：Acceptor收到提议后，如果该提议的n不等于自身之前保存的n，不接受该请求，如果相等，则写入本地。

Raft

解决问题

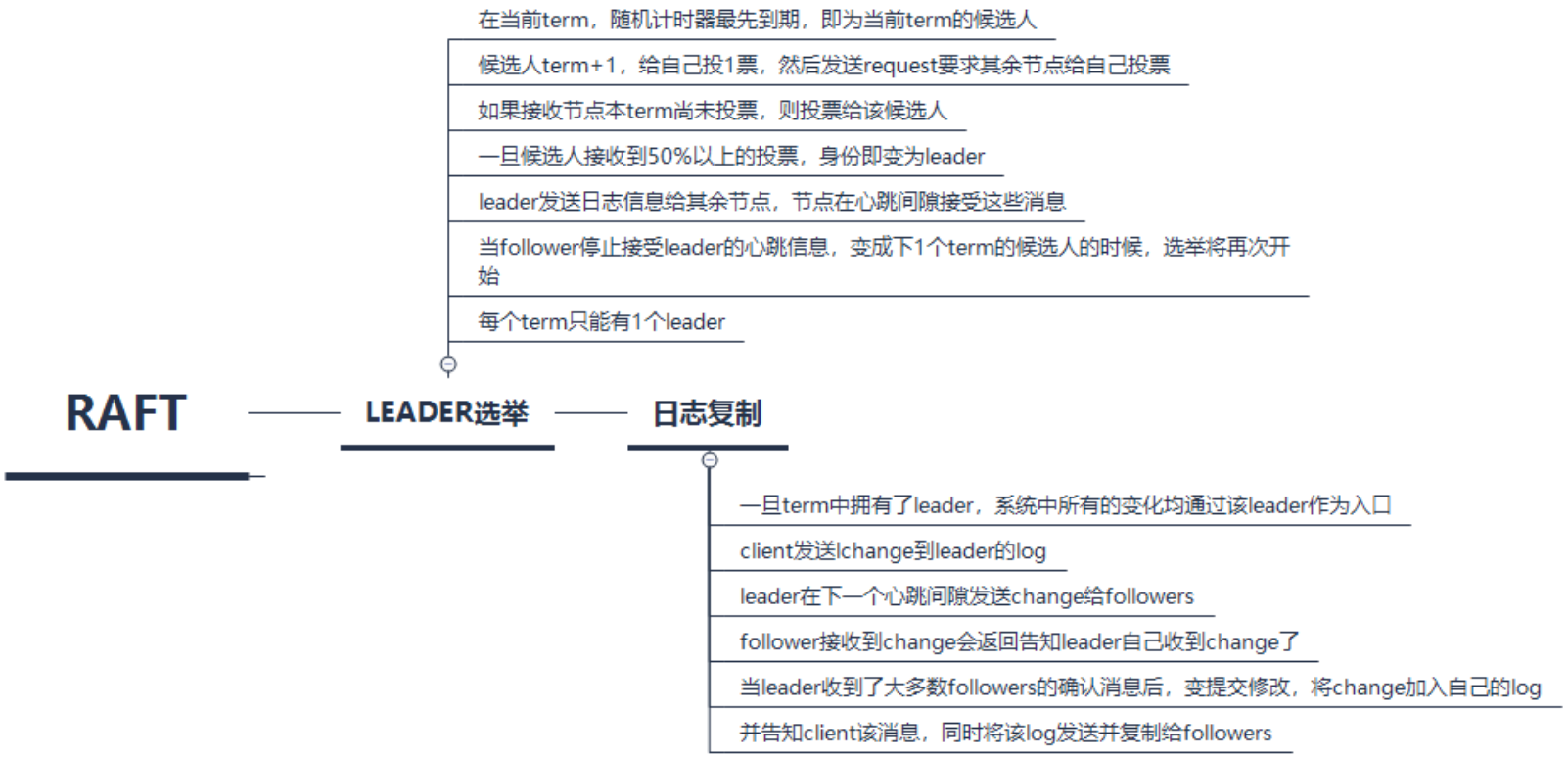
- 相比于Paxos，构建一个容易理解和工程实现的分布式协议。
- 无拜占庭问题的假设前提。

基本思想

- 在随机时间leader选举基础上叠加了各种补丁的方案。
- Log复制机制状态机
  - 首先，选出leader，leader主服务器负责接收客户端的命令请求，并将该指令以log日志的形式分发复制到其他服务器保存起来，在主服务器认为数据可靠有保障的时候（超过50%的服务器完成复制），再通知其他服务器（包括自己）将这条指令在各自的状态机中进行执行。
  - 为了区分日志的时序关系，Raft对日志的时序赋予了两个概念：Terms和序号，每当发送leader选举时，terms都会递增，而序号也是单向递增的。这两个概念是为了作为一个逻辑时钟存在。
  - http://thesecretlivesofdata.com/raft/?spm=a2c4e.11153940.blogcont610948.8.30a799bfUMsBTx

Raft

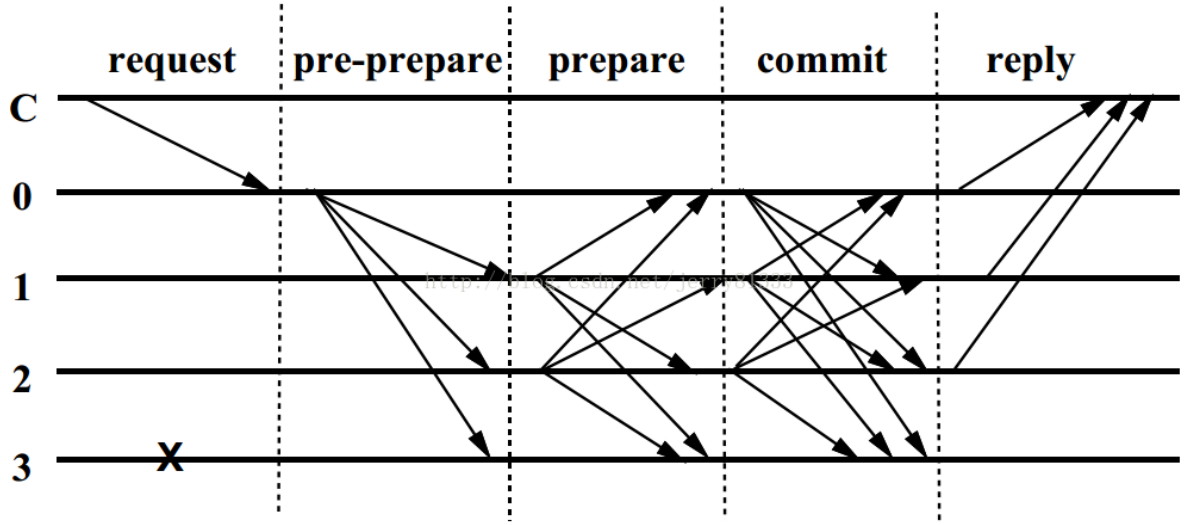
算法流程


RAFT
LEADER选举
在当前term，随机计时器最先到期，即为当前term的候选人
候选人term+1，给自己投1票，然后发送request要求其余节点给自己投票
如果接收节点本term尚未投票，则投票给该候选人
一旦候选人接收到50%以上的投票，身份即变为leader
leader发送日志信息给其余节点，节点在心跳间隙接受这些消息
当follower停止接受leader的心跳信息，变成下1个term的候选人的时候，选举将再次开始
每个term只能有1个leader
日志复制
一旦term中拥有了leader，系统中所有的变化均通过该leader作为入口
client发送change到leader的log
leader在下一个心跳间隙发送change给followers
follower接收到change会返回告知leader自己收到change了
当leader收到了大多数followers的确认消息后，变提交修改，将change加入自己的log
并告知client该消息，同时将该log发送并复制给followers

PBFT

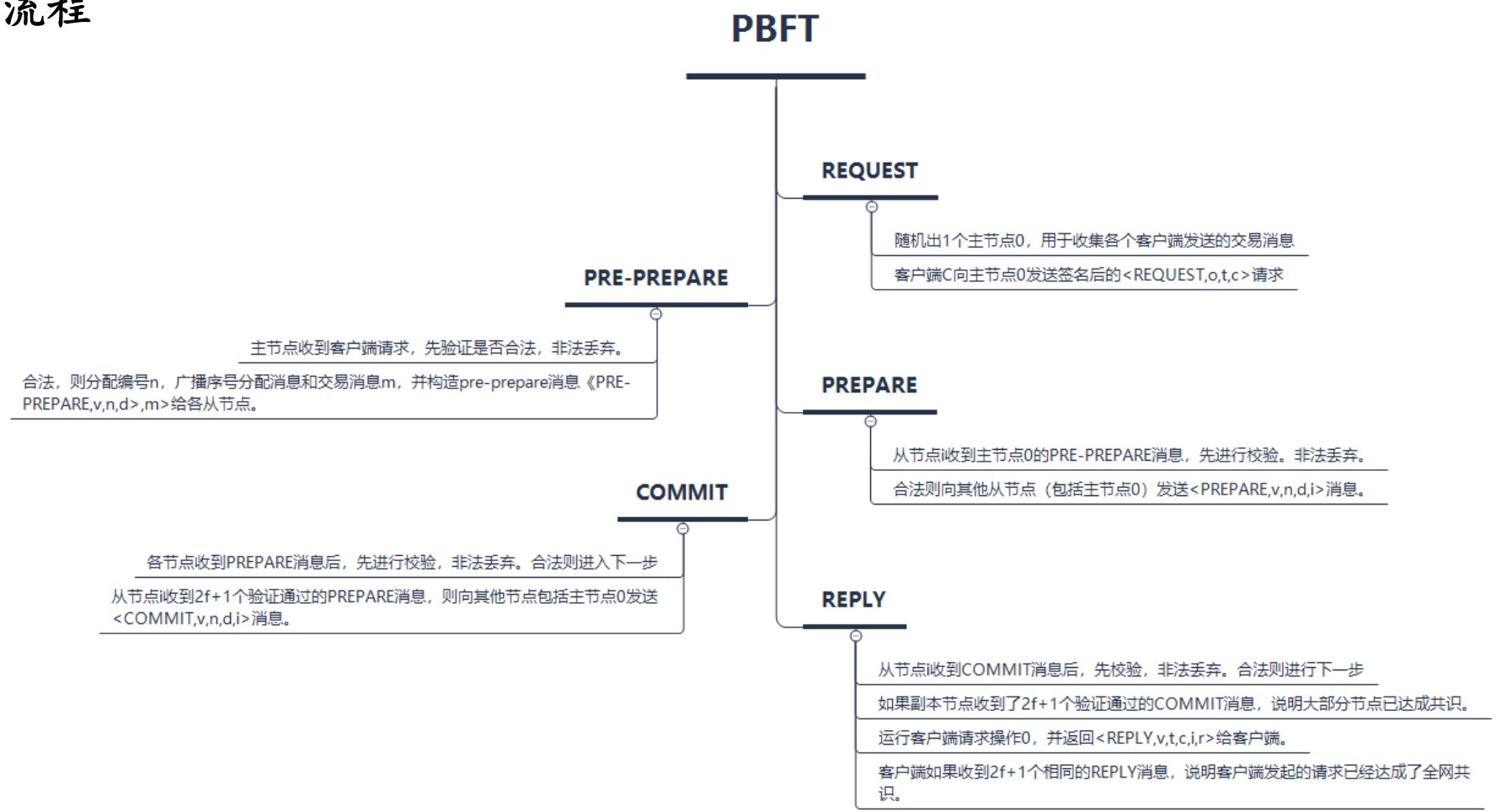
关键点

- PBFT（Practical Byzantine Fault Tolerance），实用拜占庭容错算法。
- 解决了原始拜占庭容错算法效率不高的问题，时间复杂度是O(n^2)。
- 每个节点都需要与其他节点进行P2P共识同步，节点增加，性能下降很快。主要用于联盟链。



PBFT

算法流程


PBFT
REQUEST
随机出1个主节点0，用于收集各个客户端发送的交易消息
客户端C向主节点0发送签名后的<REQUEST,o,t,c>请求
PRE-PREPARE
主节点收到客户端请求，先验证是否合法，非法丢弃。
合法，则分配编号n，广播序号分配消息和交易消息m，并构造pre-prepare消息《PRE-PREPARE,v,n,d>,m>给各从节点。
PREPARE
从节点收到主节点0的PRE-PREPARE消息，先进行校验。非法丢弃。
合法则向其他从节点（包括主节点0）发送<PREPARE,v,n,d,i>消息。
COMMIT
各节点收到PREPARE消息后，先进行校验，非法丢弃。合法则进入下一步
从节点收到2f+1个验证通过的PREPARE消息，则向其他节点包括主节点0发送<COMMIT,v,n,d,i>消息。
REPLY
从节点收到COMMIT消息后，先校验，非法丢弃。合法则进行下一步
如果副本节点收到了2f+1个验证通过的COMMIT消息，说明大部分节点已达成共识。
运行客户端请求操作0，并返回<REPLY,v,t,c,i,r>给客户端。
客户端如果收到2f+1个相同的REPLY消息，说明客户端发起的请求已经达成了全网共识。

CONFLUX

要解决的问题

- 比特币的性能和效率问题;
- 现有对比特币的改进方式简单分为 增加区块容量 和 提高出块率，但会带来更多分叉，引发安全性和资源浪费的问题。

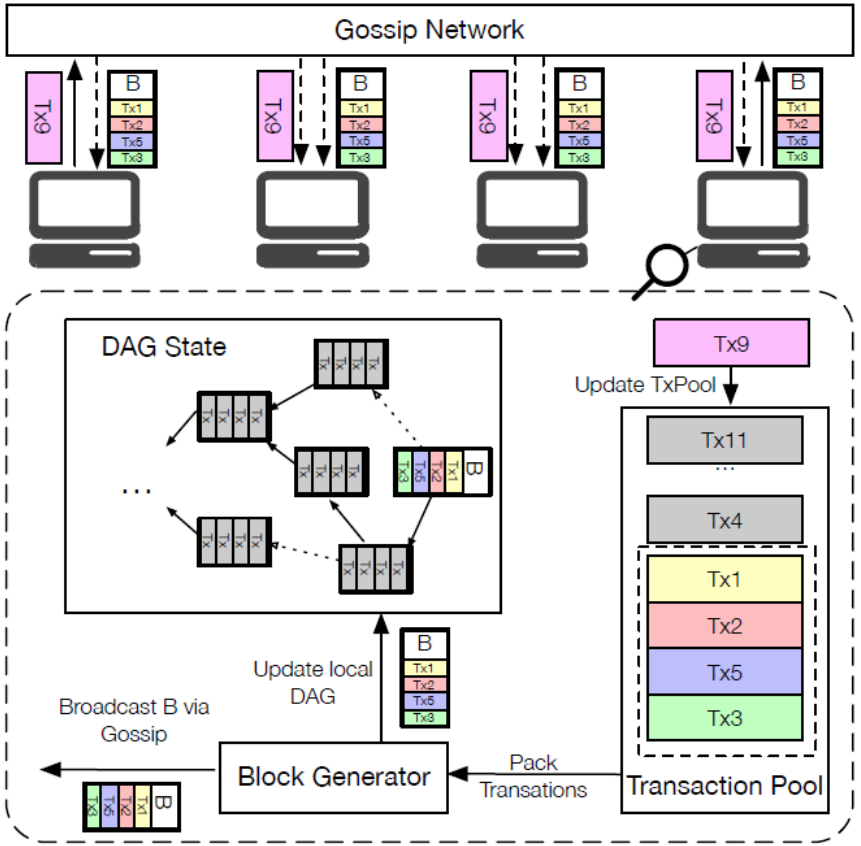
核心思想

- 采用DAG组织区块，并发的处理交易和区块，延迟确定事务的全局顺序。
- 主链不一定是最长链，防止双花。
- 通过DAG的组织和交易的全局排序，将分叉的无效区块变成有效区块，提升了有效区块的比例，从而提升系统的吞吐率，减少浪费。

CONFLUX

总体架构

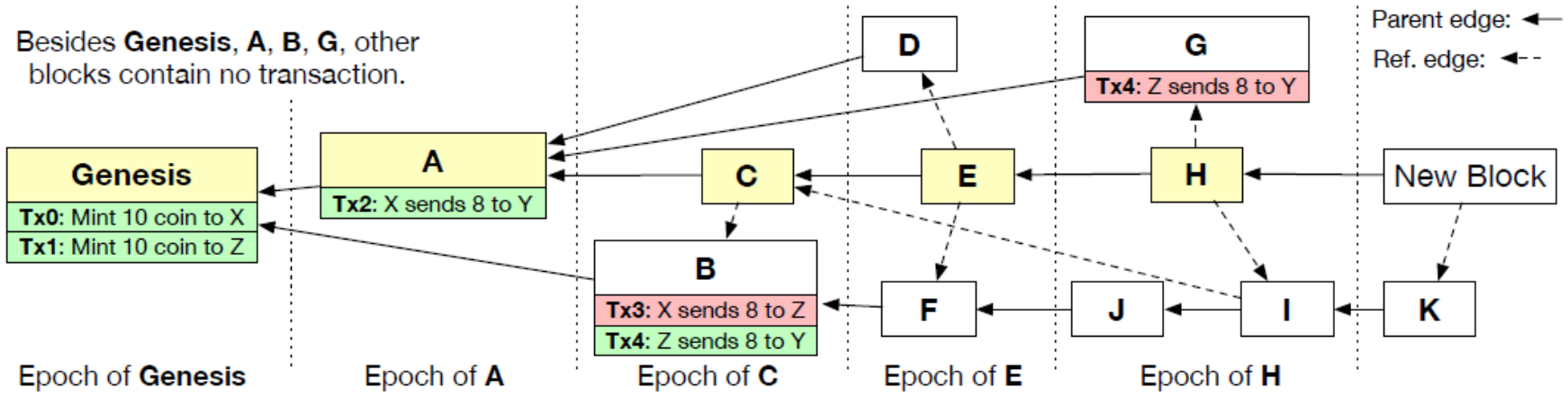
- 和比特币一样，采用Gossip实现P2P网络交互，实现交易或者区块的广播。
- 虚线部分表达了1个全节点的工作流程：node接收到1个Tx后，将Tx放入TxPool，Block generator模块将交易池中的交易进行打包。
- 打包后，将新的Block更新至本地DAG状态，将新的Block广播至全网，并根据DAG状态规定的区块顺序对该Block中的交易进行删除。
- 对于其他的全节点，如果接收到其他节点广播的block，会直接更新至本地DAG状态，并将该Block中包含的Tx从本地TxPool中删除，以避免重复交易打包。



CONFLUX

区块DAG

- Conflux的区块之间靠 父链接 和 引用链接 连接在一起，上图中实线表示 父链接，虚线表示 引用链接。链接在一起的区块关系，会形成DAG。
- Conflux中组成DAG的区块会先确定一条主链（Pivot Chian），在这个基础上再确定所有区块的先后顺序。



CONFLUX

确定主链

- G = <B, g, P, E>，其中，B是DAG中的所有区块，g是Genesis block，P是映射函数（每个区块可以通过该函数获取父区块），E是所有区块的“父链接”和“引用链接”的集合。
- 主链上下一个区块的选取规则为 子区块个数多的、或者在子区块个数相等时区块Hsah小的区块。（子区块的计算只包括父链接，不包括引用链接）

划分Epoch

- 主链Pivot Chain中的每一个block对应一个epoch，其他非Pivot Chain上的block也会被分配到相应的chain中。
- 分配的原则为，被主链上的区块“链接“，且没有被之前的区块”链接“的区块，即属于该区块Epoch。

Input : The local state $G = \langle B, g, P, E \rangle$ and a starting block $b \in B$
Output: The last block in the pivot chain for the subtree of b in G

```
if Child(G,b) = ∅ then
    return b
else
    s ⟵ ⊥
    w ⟵ −1
    for b′ ∈ Child(G,b) do
        w′ ⟵ |Subtree(G,b′)|
        if w′ > w or w′ = w and Hash(b′) < Hash(s)
         then
            w ⟵ w′
            s ⟵ b′
    return Pivot(G,s)
```

CONFLUX

确定区块顺序

- 按照Epoch进行定序，在同一个epoch中，不同的blocks按照parental tree的拓扑关系来排序。

确定交易顺序

- 在区块顺序确定的前提下，前一个区块中的交易在后一个区块中的交易前面。
- 同一区块的交易，按照区块中交易顺序进行排序。
- 当交易产生冲突时，第一个交易有效，后续冲突交易都无效。

Input : The local state $G = \langle B, g, P, E \rangle$ and a block a
Output: A list of blocks $L = b_1 \circ b_2 \circ \ldots \circ b_n$, where $b_1 = g$ and $\forall 1 \leq i \leq n, b_i \in B$

$a' \longleftarrow P(a)$
if $a' = \perp$ **then**
  return a
$L \longleftarrow \text{ConfluxOrder}(G, a')$
$B_\Delta \longleftarrow \text{Past}(G, a) - \text{Past}(G, a')$
while $B_\Delta \neq \emptyset$ **do**
  $G' \longleftarrow \langle B_\Delta, g, P, E \rangle$
  $B'_\Delta \longleftarrow \{x \mid |\text{Before}(G', x)| = 0\}$
  Sort all blocks in B'_Δ in order as $b'_1, b'_2, \ldots, b'_k$ such that $\forall 1 \leq i < j \leq k, \text{Hash}(b'_i) < \text{Hash}(b'_j)$
  $L \longleftarrow L \circ b'_1 \circ b'_2 \circ \ldots \circ b'_k$
  $B_\Delta \longleftarrow B_\Delta - B'_\Delta$
return L

ALGORAND

背景

- Silvio Micali在2016年提出，75页的长文发表在Arxiv。
- 为了解决POW共识的算力浪费、扩展性差、TPS低、易分叉等问题。

优势

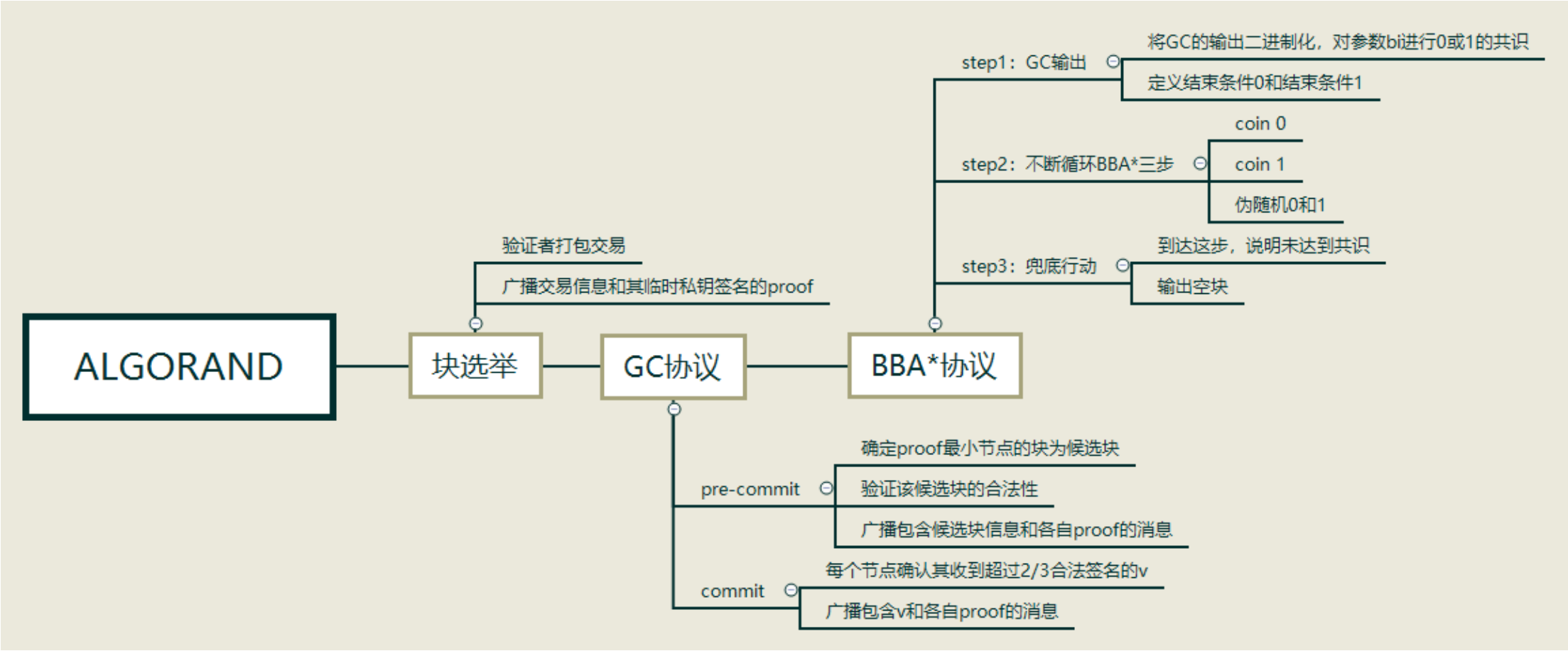
- 能耗低：1/1500的节点执行几秒钟的运算
- 民主化：不会出现矿池
- 分叉概率极低：10^-18
- 可扩展性好

缺点

- 强同步网络的假设
- 步骤太多，且每一步都要全网广播，导致时间过长

ALGORAND

算法流程



ALGORAND

关键技术

1、BBA*协议

2、种子参数

选取完全无法预测的种子参数，从而保证不被敌手影响，上一轮的种子会参与下一轮投票用户的生成。

3、秘密抽签和秘密资格

所有参与共识投票的用户都是本地得知自己的身份，投票后身份才被暴露，虽然敌手可以马上腐蚀他们，但是消息已经无法撤回。另外，消息生成后，用于签名的临时秘钥会被丢弃，敌手即使在该轮控制了他们，也无法生成合法消息。

4、用户可替换

BA*的每一步，SV都会轮换，且每一步的SV是相互独立的，因此可扩展性很强。

ALGORAND

关键技术

5、密码学抽签（VRF函数）

每一个节点持有私钥SK，并公开其验证公钥VK。如果一个用户想知道他是否被选中在参与出块r的委员会中，他讲行动如下：

- 调用Evaluate(SK,Qr-1)→ (H(σ), σ),Qr-1是系统中对所有人公开的随机种子， σ =SIGi(r,s,Qr-1)；
- 检查H(σ)是否在值域[0,P]范围内，该值域取决于用户拥有的权益比重。

如果检查通过，那么用户就被选中为SV成员，且可被他人通过一组值(Qr-1, VK, H(σ), σ)来进行验证。也就是说，用户被选择为SV成员的过程，是随机不可预测的，但又是可被他人验证的。

DFINITY

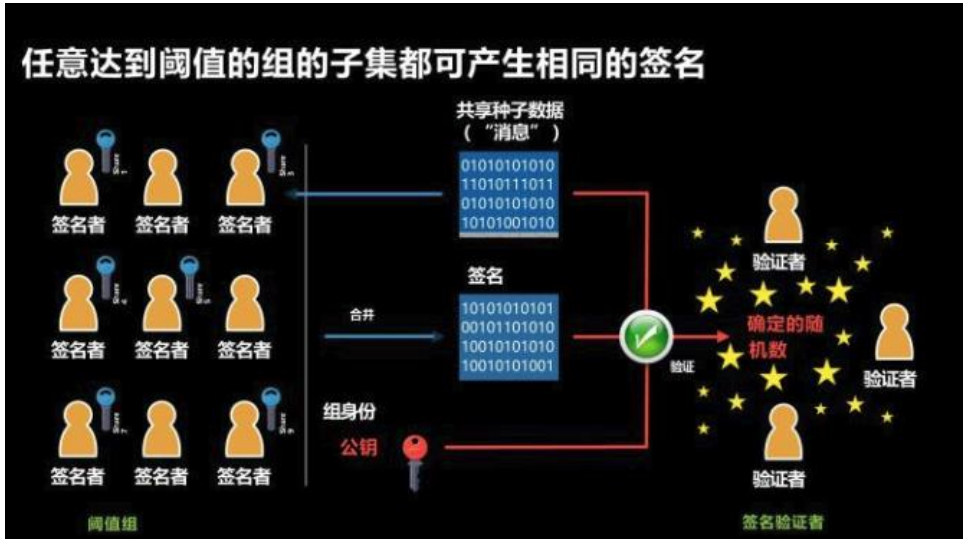
算法流程

DFINITY

关键技术

1、阈值签名（基于BLS算法）

- 一个由公钥识别的阈值组仅可在给定的种子数据上产生单个唯一有效的签名输出。
- 这个组具有容错能力，任何>=阈值的子集将组签名分片合成组签名，即可完成群签。
- 所得到的阈值签名可以由任何具有该组公钥和种子数据的人来验证。
- 这个签名是确定性产生的随机数。
- 给定阈值组的公钥和种子数据，验证者就可以在无需执行共识协议的情况下立即就随机数达成一致。

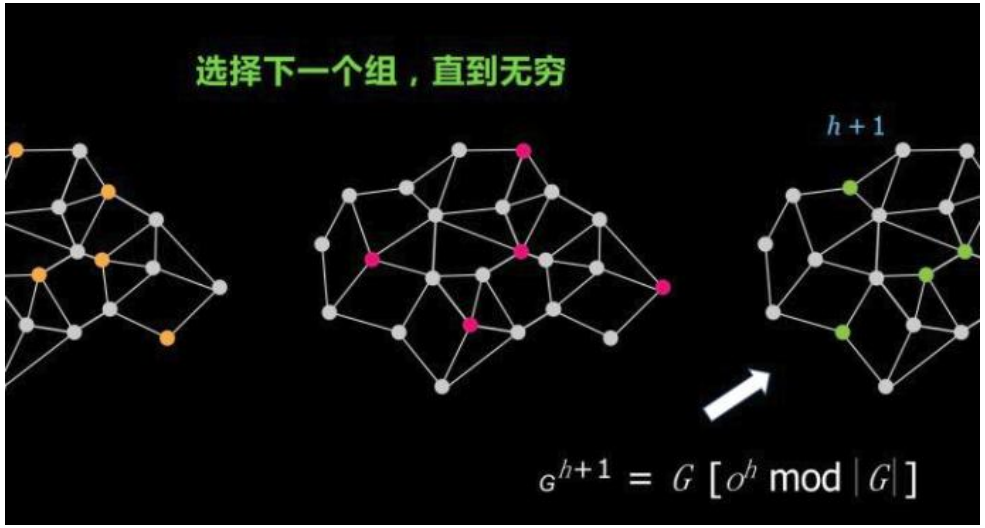


DFINITY

关键技术

2、阈值接力

- 高度为h的阈值组对高度h-1的阈值签名进行签名，生成随机数。
- 该随机数选择h+1高度的阈值组，这个过程称为阈值接力。
- 按此可产生一个去中心化的可验证随机函数（VRF）
- 该随机数序列是 确定的、不可操纵的、又是可验证的。
- 当前组达到阈值后，发布下一个值，因此又是不可预测的。



ACP

算法流程


ACP
PC SELECTION
Random Beacon
用全网已知的随机数（Helix方法）和r计算每个节点每轮的权重
根据该权重选取PC
每轮的PC成员全网透明
FC SELECTION
从PC中选出FC成员，用VRF
只有本地知道自己是否被选中
广播消息后可以被验证
BLOCK PROPOSAL
等待一段时间后，每个FC成员均收集交易信息，打包提交块
Reduction S1: 每个节点从其收到的块中选取体积最大的，体积相同选Hash(proof)最小的，作为准有效块再次提交
Reduction S2: 节点如果收到了2/3以上的Br，则输出Br；若没收到足够数量，则输出空块。
REACH CONSENSUS
Leadless PBFT*
随机选取多个leader并行跑PBFT
若共识了Br，则有效块为Br，向全网进行广播，其余节点直接接受这个共识结果。
若不能共识Br，则输出空块，向全网广播。