Vite: 一个异步高性能的通用去中心化应用平台

刘春明 王东 伍鸣
charles@vite.org daniel@loopring.org woo@vite.org

2018 年 4 月 5 日

Abstract

摘要。

1 简介

1.1 定义

一个通用去中心化应用平台，可以支持一组智能合约，每个智能合约都是一个拥有独立状态以及不同操作逻辑的交易状态机，它们之间可以通过消息传递的方式进行通信。

从整体上，这个系统是一个交易状态机。系统的状态 $\boldsymbol{s} \in \boldsymbol{S}$ 也称为世界状态，是由每一个独立账户的状态构成的。一个能够引起世界状态改变的事件称为交易。更形式化的定义如下：

Definition 1.1 (交易状态机) 一个交易状态机是一个四元组：$(\boldsymbol{T}, \boldsymbol{S}, \boldsymbol{g}, \boldsymbol{\delta})$，其中，$\boldsymbol{T}$ 是交易的集合，$\boldsymbol{S}$ 是状态的集合，$\boldsymbol{g} \in \boldsymbol{S}$ 为初始状态，也称为“创世块”，$\boldsymbol{\delta} : \boldsymbol{S} \times \boldsymbol{T} \to \boldsymbol{S}$ 是状态转移函数。

这种交易状态机的语义是一个离散迁移系统，定义如下：

Definition 1.2 (交易状态机的语义) 一个交易状态机 $(\boldsymbol{T}, \boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{\delta})$ 的语义是一个离散迁移系统：$(\boldsymbol{S}, \boldsymbol{s_0}, \to)$。$\to \in \boldsymbol{S} \times \boldsymbol{S}$ 是迁移关系，

同时，去中心化应用平台是一个分布式系统，具有最终一致性。通过某种共识算法，节点间可以就最终的状态达成一致。在现实场景中，智能合约的状态中保存的是一个去中心化应用的全部数据，体积比较大，无法在节点间传输。因此，节点间需要通过传递一组交易，来达成最终状态的一致。我们将这样的一组交易组织成某种特定的数据结构，通常称之为账本。

Definition 1.3 (账本) 账本是由一组交易构成的，具有递归构造的抽象数据类型，定义如下：

$$\begin{cases} l = \Gamma(S_t) \\ l = l_1 + l_2 \end{cases}$$

其中，$S_t \in 2^T$，表示一组交易，$\Gamma \in 2^T \to L$，表示通过一组交易构造一个账本的函数，L 是账本的集合，$+ : L \times L \to L$，表示将两个子账本合并成一个账本的操作。

需要注意的是，在此类系统中，账本通常用来表示一组交易，而不是一个状态。在比特币 [1] 和以太坊 [2] 中，账本是一个区块链结构，其中交易是全局有序的。修改账本中的一个交易，需要重新构造账本中的一个子账本，从而提高了篡改交易的成本。

根据同一组交易，可以构造出不同的账本，两个账本都是有效的，但它们所表示的交易顺序不同，因此可能会导致系统进入不同的状态。当这种情况发生时，我们称账本发生了“分叉”。

Definition 1.4 (分叉) $S_t \in 2^T$，$l_1 = \Gamma_1(S_t)$，$l_2 = \Gamma_2(S_t)$，若 $l_1 \neq l_2$，则称账本 l_1 和 l_2 是分叉的。

根据交易状态机的语义，我们可以很容易的证明，从一个初始状态开始，如果账本不分叉，则每个节点最终会进入相同的状态。那么，如果接收到分叉的账本，就一定进入不同的状态吗？这取决于账本中交易的内在逻辑，以及账本如何组织交易之间的偏序。现实中，经常会出现一

些满足交换律的交易，却因为账本设计的问题，频繁的引起分叉。当系统从一个初始状态出发，接收两个分叉的账本，最终进入同一状态，我们称这两个账本为伪分叉账本。

Definition 1.5 (伪分叉) *有初始状态 $s_0 \in S$，账本 $l_1, l_2 \in L$，$s_0 \xrightarrow{l_1} s_1, s_0 \xrightarrow{l_2} s_2$。若 $l_1 \neq l_2$，且 $s_1 = s_2$，则称这两个账本 l_1, l_2 为伪分叉 (false fork) 账本。*

一个设计良好的账本，应该尽量降低伪分叉发生的概率。

当分叉发生时，每个节点都需要从多个分叉的账本中选择一个，为确保状态的一致性，这些节点需要采用同一个算法完成选择，这个算法称为共识算法。

Definition 1.6 (共识算法) *共识算法是一个函数，它接收一个账本的集合，返回其中唯一一个账本：*

$$\Phi : 2^L \to L$$

共识算法是系统设计的一个重要内容，一个好的共识算法应该具有较高的相交速度 (high convergence speed)，减少共识在不同分叉间摇摆，并对恶意攻击具有较高的防范能力。

1.2 当前进展

1.3 以太坊

以太坊 [3] 率先实现了这样一个系统。在以太坊的设计中，世界状态的定义是：$S = \Sigma^A$，是由每个账户 $a \in A$ 与该账户的状态 $\sigma_a \in \Sigma$ 构成的映射。因此，以太坊的状态机中任何一个状态都是全局的，这表示一个节点在每个时刻都可以获取任何一个账户的状态。

以太坊的状态转移函数 δ，是由一组程序代码来定义的，每组代码被称为一个智能合约。以太坊定义了一个图灵完备的虚拟机，称为 EVM，运行在其上的指令集称为 EVM 代码。用户可以通过一种语法类似于 javascript 的程序语言 Solidity 来开发智能合约，并编译成 EVM 代码，部署到以太坊上 [4]。一旦智能合约部署成功，就相当于定义了该合约账户 a 收到一个交易后的状态转移函数 δ_a。EVM 目前在此类平台中被广泛使用，但它也存在一些问题。例如，缺少库函数支持，安全性问题突出等。

以太坊的账本结构是区块链 [1]，区块链由区块构成，每一个区块中包含一组交易的列表，后一个区块引用前一个区块的 hash，构成一个链状结构。

$$\Gamma(\{t_1, t_2, ...|t_1, t_2, ... \in T\}) = (..., (t_1, t_2, ...)) \quad (1)$$

这个结构的最大好处是有效的防止交易被篡改，但由于它维护的是所有交易的全序，任何两个交易交换顺序，都会生成一个新账本，也造成这种结构具有较高的分叉概率。事实上，在这个定义下，交易状态机的状态空间被看作一棵树：初始状态是根节点，不同的交易顺序代表不同的路径，叶子节点是最终状态。现实的情况下，大量叶子节点的状态是相同的，这就造成了大量的伪分叉。

以太坊的共识算法 Φ 称为 PoW，该算法率先在比特币协议中提出 [1]。PoW 算法依赖于某个易于验证但难于求解的数学问题，例如，根据一个 hash 函数 $h : N \to N$，求解 x，使 $h(T + x) \geq d$，d 是一个给定的数，称为难度，T 是区块中包含的交易列表的二进制表示。在区块链的每个区块中，都会包含这类问题的一个解。将全部区块的难度加起来，就是一个区块链账本的总难度：

$$D(l) = D(\sum_i l_i) = \sum_i D(l_i) \quad (2)$$

因此，在从分叉中选择正确账本的时候，只要选择总难度最高的分叉即可：

$$\Phi(l_1, l_2) = \begin{cases} l_1 \text{ if } D(l_1) \geqslant D(l_2) \\ l_2 \text{ otherwise} \end{cases} \quad (3)$$

PoW 共识算法具有较好的安全性，迄今为止在比特币和以太坊中运行得很好。但这个算法有两个主要问题，第一是求解数学难题需要消耗大量计算资源，造成能源浪费；第二是该算法相交速度较慢，因而影响了系统整体的吞吐。目前，以太坊整体的 TPS 只有 15 左右，完全无法满足去中心化应用的需求。

1.4 改进方向

在以太坊诞生之后，以太社区和其他一些同类项目开始从不同方向对系统加以改进。从系统的抽象模型来看，可以改进的方向主要包括以下几个：

- 改进系统状态 $\boldsymbol{S}$
- 改进状态迁移函数 $\boldsymbol{\delta}$
- 改进账本结构 $\boldsymbol{\Gamma}$
- 改进共识算法 $\boldsymbol{\Phi}$

1.4.1 改进系统状态

对系统状态的主要改进思路是将全局的世界状态局部化，每个节点不再关心全部交易和状态转移，只维护整个状态机的一个子集。这样集合 S 和集合 T 的势都大为缩减，从而提高了系统扩展性。此类系统包括：Cosmos[5]，Aelf[6]，PChain 等。

从本质上讲，此类基于侧链的方案牺牲了状态的全局性，以换取系统的扩展性。这使得每个运行在其上的 dApp 的去中心化程度都被削弱——一个智能合约的交易历史不再被全网每一个节点保存，而只被一部分节点保存。除此之外，跨合约交互也会成为此类系统的瓶颈。例如，Cosmos 中，不同的 Zone 交互，需要通过一个共同的链 Hub 来完成 [5]。

1.4.2 改进状态迁移函数

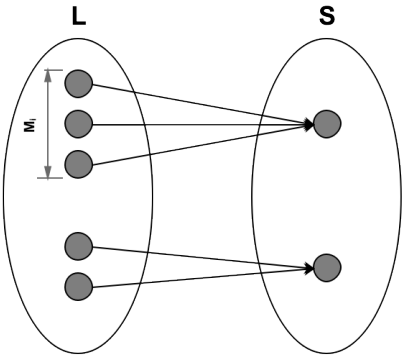
一些项目立足于改进 EVM，提供更为丰富的智能合约编程语言。例如，RChain 定义了一种基于 π 演算的智能合约语言 Rholang；NEO 中的智能合约称为 NeoContract，可以用 Java，C# 等主流编程语言开发；EOS 使用 C/C++ 来编程。

1.4.3 改进账本结构

账本结构的改进方向是构造等价类，将多个交易全局有序的线性账本规约为一个只记录部分偏序关系的非线性账本，这种非线性账本结构是一个 DAG(有向无环图)。目前，Byteball[7]，IOTA[8]，Nano[9] 等项目基于 DAG 的账本结构实现了加密货币功能。也有一些项目在尝试利用 DAG 实现智能合约，但迄今为止在这个方向上的改进还在探索中。

1.4.4 改进共识算法

共识算法的改进大部分是为了提高系统的吞吐，主要方向是抑制伪分叉的产生。下面我们讨论伪分叉与哪些因素有关。



如图所示，L 是针对某个交易集合所有可能的分叉账本的集合，S 是以不同顺序执行这一组交易，所能到达的状态的集合。根据定义 1.4，映射 $f : L \to S$ 是一个满射；而根据定义 1.5，这个映射不是单射。下面我们来计算伪分叉的概率：

假设共有 C 个用户有权生产账本，$M = |L|$，$N = |S|$，$M_i = |L_i|$，其中，$L_i = \{l|f(l) = s_i, s_i \in S\}$。则伪分叉概率为：

$$P_{ff} = \sum_{i=1}^{N} \left(\frac{M_i}{M}\right)^C - \frac{1}{M^{C-1}} \tag{4}$$

从这个公式可以看出，为了降低伪分叉概率，可以有两种途径：

- 在账本集合 L 上建立等价关系，对其划分等价类，构造分叉更少的账本
- 限制有权生产账本的用户，从而减少 C

第一种途径是 Vite 设计的重要方向，后文将详细论述；第二种途径现在已被多种算法所采用。在 PoW 算法中，任何用户都有权生产区块；而 PoS 算法将生产区块的权力限制在那些拥有系统权益的用户中；DPoS 算法将有权生产区块的用户进一步限制在一组代理节点范围内。

目前，通过改良共识算法，已经产生出一些比较有影响力的项目。例如，Cardano 采用了一种 PoS 算法，称为 Ouroboros，文献 [10] 对该算法相关性质给出了严格证明；EOS 采用了 DPoS 算法，通过快速生产区块，提高系统的吞吐；Qtum[11] 的共识算法也是一种 PoS 算法;RChain 采用的 Casper 算法也是一种 PoS 算法。

还有一些其他项目在共识算法的改进上提出了自己的方案。NEO 采用了一种 BFT 算法，称为 dBFT；Cosmos 采用了一种称为 Tendermint[12] 的算法。

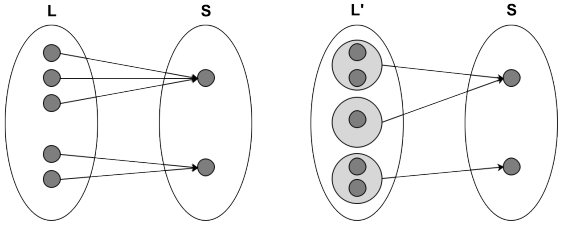
2 账本

2.1 概述

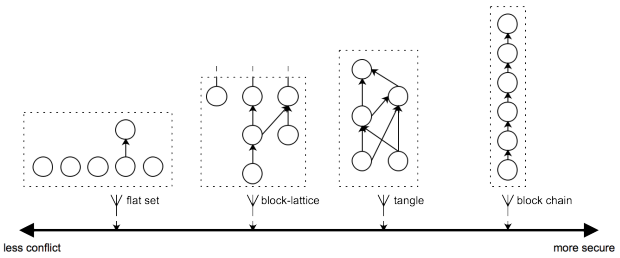
账本的作用主要是为了确定交易之间的顺序，交易的顺序会影响以下两个方面：

- **状态一致性：** 由于系统的状态不是一个CRDT(Conflict-free replicated data types)[13]，因此，交易不都是可交换的，不同的交易执行顺序可能会导致系统进入不同的状态。
- **hash 有效性：** 账本中，交易会被打包成区块，区块中包含互相引用的 hash。交易的先后顺序会影响账本中 hash 引用的连通性。这种影响的范围越大，篡改交易的成本就越大。这是因为，改变任何一个交易，都必须重建所有直接或间接引用该交易的区块的 hash。

而账本的设计也有两个主要目标：



- **降低伪分叉率：** 如前文讨论，降低伪分叉率可以通过建立等价类，尽量将导致系统进入同一状态的一组账本合并成一个账本来实现。如上图，根据伪分叉率公式可以算得，左图的账本伪分叉率 $P_{ff} = \left(\frac{3}{5}\right)^C + \left(\frac{2}{5}\right)^C - \frac{1}{5^{C-1}}$；而合并账本空间后，右图的伪分叉率为 $P_{ff}{}' = \left(\frac{2}{3}\right)^C + \left(\frac{1}{3}\right)^C - \frac{1}{3^{C-1}}$。可知当 $C > 1$，$P_{ff}{}' < P_{ff}$。也就是说，我们应尽量减小账本中交易之间的偏序关系，允许更多的交易之间的顺序可交换。
- **防篡改：** 当账本 l 中一个交易 t 被修改，账本的两个子账本 $l = l_1 + l_2$ 中，子账本 l_1 不受影响，而子账本 l_2 中的 hash 引用需要重建，以构成一个新的有效账本 $l' = l_1 + l_2{}'$。受影响的子账本 $l_2 = \Gamma(T_2), T_2 = \{x | x \in T, x > t\}$。由此可见，想提高篡改交易的成本，需要在账本中尽量多的维护交易之间的偏序关系，以扩大篡改的影响范围 $|T_2|$。



显然，以上两个目标是互相矛盾的，在设计账本结构时必须作出必要的权衡取舍。由于账本维护的是交易之间的偏序，因此它本质上是一个偏序集 (poset)[14]，如果用哈斯图 (Hasse diagram)[15] 来表示，在拓扑上就是一个DAG。

上图对比了几种常见的账本结构，靠近左侧的账本维护更少的偏序关系，哈斯图显得比较扁平，具有更低的伪分叉率；靠近右侧的账本维护更多的偏序关系，哈斯图比较细长，具有更高的防篡改特性。

图中，最左侧的是一种中心化系统中常见的基于集合的结构，没有任何防篡改特性；最右侧则是典型的区块链账本，具有最好的防篡改特性；而介于二者中间的是两种DAG 账本，左侧的是 Nano 采用的 block-lattice 账本 [9]，右侧是 IOTA 采用的 tangle 账本 [8]。从特性上看，block-lattice 维护了更少的偏序关系，更适合作为高性能去中心化应用平台的账本结构。但由于它的防篡改特性较差，会产生安全隐患，因此，迄今为止，除了 Nano 采用了该结构之外，还没有其他项目采用。

为了追求高性能，Vite 采用该账本结构。同时，通过引入一种额外的链式结构 Snapshot Chain，并且改进了共识算法，成功的弥补了 block-lattice 安全性方面的不足，后文将详细论述这两个改进。

2.2 前置约束

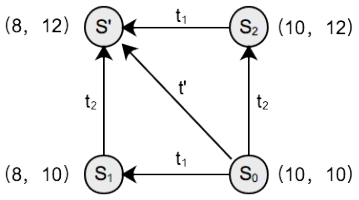
首先，我们来看一下采用这种账本结构对状态机模型的前置要求。这种结构本质上是将整个状态机看作是一组独立的状态机的组合，每个账户对应一个独立的状体机，每个交易只影响一个账户的状态。在账本，对所有交易按账户分组，并把同一账户的交易组织成一条链。因此，我们对 Vite 中的状态 S 和交易 T 做出如下限制：

Definition 2.1 (交易的单自由度约束) *系统状态 $s \in S$，是由每个账户的状态 s_i 构成的向量 $s = (s_1, s_2, ..., s_n)$。*

对于 $\forall t_i \in T$，执行交易 t_i 后，系统状态发生如下转移：$(s_1', ..., s_i', ..., s_n') = \sigma(t_i, (s_1, ..., s_i, ..., s_n))$，需满足：$s_j' = s_j, j \neq i$。该约束称为交易的单自由度约束。

直观上，一个单自由度的交易只会改变一个账户的状态，不会影响系统中其他账户的状态。在状态空间向量所在的多维空间中，执行一次交易，系统状态只会沿平行于某个坐标轴的方向移动。请注意，这个定义要比比特币、以太坊等模型中的交易定义更为严格，比特币中的一个交易，会改变发送者和接收者两个账户的状态；以太坊则可能通过消息调用，改变两个以上的账户的状态。

在这个约束条件下，交易之间的关系得以简化。任何两个交易，要么是正交的，要么是平行的。这为依照账户对交易进行分组提供了条件。下面举一个例子来说明：



如上图所示，假设 Alice 和 Bob 各有 10 元钱，系统的初始状态为 $s_0 = (10, 10)$。当 Alice 想给 Bob 转账 2 元时，在比特币和以太坊的模型中，可以通过一个交易 t'，使系统直接进入最终状态：$s_0 \xrightarrow{t'} s'$。

而在 Vite 的定义中，交易 t' 同时改变了 Alice 和 Bob 两个账户的状态，不符合单自由度的原则。因此，这个交易必须被拆分成两笔交易：

1) 表示 Alice 转出 2 元的交易 t_1

2) 表示 Bob 转入 2 元的交易 t_2。

这样，从初始状态到达最终状态 s' 可以有两条不同的路径 $s_0 \xrightarrow{t_1} s_1 \xrightarrow{t_2} s'$ 和 $s_0 \xrightarrow{t_2} s_2 \xrightarrow{t_1} s'$。这两条路径分别通过中间状态 s_1 和 s_2，这两个中间状态是最终状态 s' 在两个账户上维度的投影。也就是说，如果只关心其中某一个账户的状态，只需要执行该账户对应的所有交易，而不需要执行其他账户的交易。

下面我们来定义如何将以太坊中的交易，拆分成 Vite 所要求的单自由度交易：

Definition 2.2 (交易投影) 将一个自由度大于 *1* 的交易拆分成一组单自由度交易的过程，称为交易投影。一笔转账交易可以拆分成一个出账交易和一个入账交易；一个合约调用交易可以拆分成一个合约请求交易和一个合约响应交易；每个合约内部的消息调用，可以拆分成一个合约请求交易和一个合约响应交易。

这样，账本中就有两种不同类型的交易，它们被称为“交易对”：

Definition 2.3 (交易对) 一个入账交易或合约请求交易，统称为“请求交易”；一个出账交易或合约响应交易，统称为“响应交易”。一个请求交易和一个对应的响应交易，称为交易对。发起请求交易 t 的账户，记作 $A(t)$；对应的响应交易记作：$\widetilde{t}$，该交易对应的账户，记作 $A(\widetilde{t})$。

根据以上定义，我们可以得出 Vite 中，任何两个交易之间可能存在的关系：

Definition 2.4 (交易的关系) 对于两个交易 t_1 和 t_2，可能存在如下关系：

正交：若 $A(t_1) \neq A(t_2)$，则称两个交易正交，记作 $t_1 \perp t_2$；

平行：若 $A(t_1) = A(t_2)$，则称两个交易平行，记作 $t_1 \parallel t_2$；

因果：若 $t_2 = \widetilde{t_1}$，则称两个交易具有因果关系，记作 $t_1 \rhd t_2$，或者 $t_2 \lhd t_1$。

2.3 账本定义

定义一个账本，就是定义一个偏序集。首先，我们来定义 Vite 中，交易之间的偏序关系：

Definition 2.5 (交易的偏序) 我们用二元关系 $<$ 来表示两个交易的偏序关系，有：

一个响应交易，必须排在一个对应的请求交易之后：$t_1 < t_2 \Leftrightarrow t_1 \rhd t_2$；

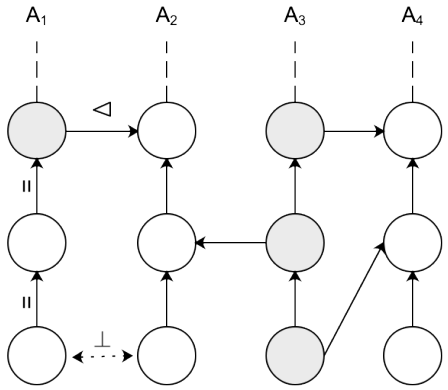
一个账户的所有交易，必须严格全局有序：$\forall t_1 \parallel t_2$，必有：$t_1 < t_2$，或 $t_2 < t_1$。

由于建立在交易集合 T 上的偏序关系 $<$ 满足：

- 非自反性 (irreflexive)：$\forall t \in T$，不存在 $t < t$；
- 传递性 (transitive)：$\forall t_1, t_2, t_3 \in T$，若 $t_1 < t_2, t_2 < t_3$，则有 $t_1 < t_3$；
- 非对称性 (asymmetric)：$\forall t_1, t_2 \in T$，若 $t_1 < t_2$，则不存在 $t_2 < t_1$

这样，我们就可以用严格偏序集的方式来定义 Vite 的账本：

Definition 2.6 (Vite 账本) *Vite 账本是给定的交易集合 T，以及偏序关系 $<$，所构成的严格偏序集。*



一个严格偏序集，可以对应到一个 DAG 结构。上面所定义的 Vite 账本，结构类似于 block-lattice。交易分为请求交易和响应交易两种，每个交易对应一个单独的区块，每个账户 A_i 对应一条链，一个交易对中，响应交易引用其对应的请求交易的 hash。

3 共识

3.1 交易确认

当账本发生分叉时，共识结果可能会在两个分叉账本间摇摆。例如，基于区块链结构的系统，如果节点接收到一个更长的分叉链，就会选择这个新分叉作为共识结果，而原分叉将被废弃，原分叉上的交易也会被回滚。在此类系统中，交易被回滚是一个非常严重的事件，将会导致双花 (double spend)。试想，一个商家接收到一笔付款，提供了商品或服务，之后这笔付款又被撤回，商家可能会因此面临损失。因此，用户在收到一笔付款交易时，需要等待系统对这笔交易进行“确认”，以确保这笔交易被回滚的概率足够低。

Definition 3.1 (交易确认) *当一个交易被回滚的概率小于一个给定的阈值 ϵ 时，称该交易为已确认 (confirmed)。$P_r(t) < \epsilon \Leftrightarrow t$ is $\boldsymbol{confirmed}$。*

交易的确认是一个非常容易被混淆的概念，因为一个交易是否被确认实际上取决于隐含的置信度 $1-\epsilon$。一个售卖钻石的商家和一个售卖咖啡的商家，在被双花攻击的时候，所蒙受的损失是不同的。因此，前者需要对交易设置更小的 ϵ。这也是比特币中确认数的本质。在比特币中，确认数表示一个交易在区块链中的深度，确认数越大，交易被回滚的概率越低 [1]。因此，商家可以通过设置等待多少确认数，间接设定确认的置信度。

交易被回滚的概率随时间而降低，是由于账本结构中存在 hash 引用关系。如前文所述，当账本的设计具有较好的防篡改特性时，回滚一个交易需要重构该交易所在区块的所有后继区块。随着新的交易被不断加入账本，某个交易的后继节点也越来越多，因此，被篡改的概率也随之下降。

而 block-lattice 结构中，由于交易是按账户分组的，一个交易只会附加到其所属账户的账户链末端，大部分其他账户产生的交易不会自动成为该交易的后继节点。因此，采用这个结构必须合理的设计共识算法，以避免双花的隐患。

Nano 采用了一个基于投票的共识算法 [9]，由一组用户选择的代表节点对交易进行签名，每个代表节点有一个权重，当某个交易获得的签名累计起来有足够多的权重时，就认为该交易被确认。这个算法有以下几个问题：

首先，如果需要更高的确认置信度，则需要提高投票权重的阈值，如果没有足够多的代表节点在线，就无法保证相交速度，有可能一个用户永远也搜集不到确认一个交易所必需的票数；

其次，交易被回滚的概率不随时间递减。这是因为任何时候，推翻一个历史投票的结果，所付出的成本都是一样的。

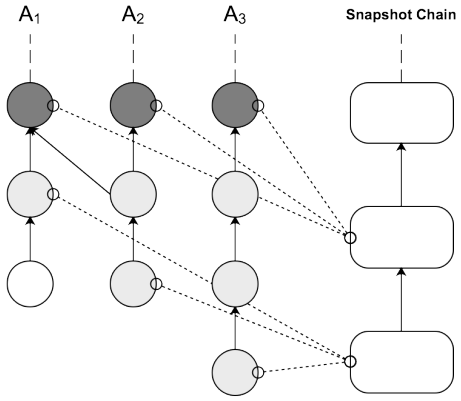
最后，历史的投票结果并没有被持久化到账本中，只保存在节点的本地存储中。当一个节点从其他节点获取账本时，没有办法可靠的量化一个历史交易被回滚的概率。

从本质上，投票的机制是一个偏中心化的解决方案。我们可以把投票结果看作是对账本状态的一个快照，这个快照会分布式的保存在网络中各个节点的本地存储中。为了拥有和区块链同样的防篡改能力，我们可以将这些快照也组织成链式结构，这就是 Vite 设计的核心之一——快照链 [16]。

3.2 快照链

快照链是 Vite 中最重要的存储结构，它的主要作用是维护 Vite 账本的共识。首先我们给出快照链的定义：

Definition 3.2 (快照块和快照链) *一个快照块存储一个 Vite 账本的状态快照，状态包括：账户的余额、合约状态的 Merkle root，每个账户链中最后一个块的 hash。快照链是由快照块组成的链式结构，后一个快照块引用前一个快照块的 hash。*



一个用户账户的状态包含余额和账户链最后一个块的 hash；一个合约账户的状态，除了包含以上两个字段之外，还包含该账户合约状态的 Merkle root hash。账户的状态的结构如下：

```
struct AccountState {
  // 账户余额
  uint256 balance;
  // 合约状态的Merkle root
  optional uint256 storageRoot;
  // 账户链最后一个交易的hash
  uint256 lastTransaction;
}
```

快照块的结构定义如下：

```
struct SnapshotBlock {
  // 前一个块的hash
  uint256 prevHash;
  // 快照信息
  map<address, AccountState> snapshot;
  // 签名
  uint256 signature;
}
```

快照链中第一个快照块称为“创世快照”，保存的是账本中创世块的快照。

由于快照链中每个快照块都对应于 Vite 账本的唯一分叉，因此，在快照块不发生分叉的情况下，通过快照块，可以确定 Vite 账本的共识结果。

3.3 快照链与交易确认

引入了快照链之后，block-lattice 结构天然的安全性缺陷就得到了弥补。攻击者想生成一个双花交易，除了要重建 Vite 账本中的 hash 引用之外，还需要重建快照链中，首次快照该交易的快照块之后的所有区块，并且需要产生一条更长的快照链。这样，攻击成本将大大提高。

在 Vite 中，交易的确认机制类似于比特币，定义如下：

Definition 3.3 (Vite 交易确认) *在 Vite 中，一个交易被快照链所快照，则成为该交易被确认。第一次快照该交易的快照块的深度，称为交易的确认数。*

在这个定义下，快照链每增长一个区块，此前所有已确认交易的确认数都增加 1，双花攻击成功的概率随着快照链的不断增加而逐渐下降。这样，用户就可以根据具体的场景，通过等待不同的确认数，来定制所需的确认置信度。

快照链本身依赖一个共识算法，如果快照链发生分叉，则选取**最长的**分叉作为合法分叉。当快照链切换到新的分叉时，原有快照信息会被回滚，相当于原来对账本达成的共识被推翻，由新的共识结果取代。因此，快照链是整个系统安全性的基石，需要非常认真的对待。

3.4 压缩存储

由于快照链中每一个快照块都需要保存所有账户的状态，耗费的存储空间非常大，因此需要对快照链进行压缩。

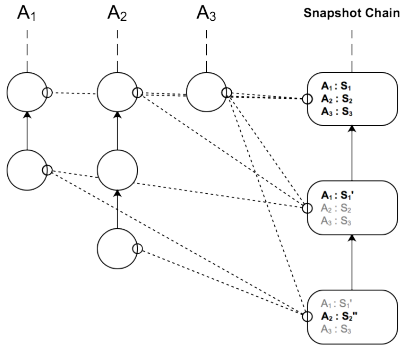


图 1: 压缩前的快照

压缩快照链存储空间的基本思路是利用增量存储：一个快照块只保存相比于前一个快照块发生变化的数据。如果一个账户在两个快照之间没有发生任何交易，则后一个快照块不保存该账户的数据。

要恢复快照信息，可以从前向后依次遍历快照块，将每一个快照块中的数据覆盖到当前的数据上即可。

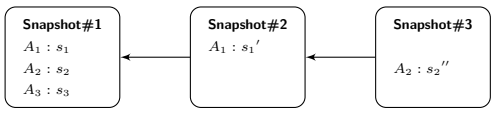


图 2: 压缩后的快照

由于每个快照只保存在快照时刻每个账户最后的状态，不关心中间状态，无论在两个快照块之间，一个账户产生了多少交易，在快照中也只保存一份数据。因此，一个快照块最多占用 $S * A$ 个字节。其中，$S = \text{sizeof}(s_i)$，为每个账户状态占用的字节数，A 是系统总账户数。

3.5 出块权

4 限流

$$TPS_n = \frac{10}{S \cdot (H_n \ - \ H_{n-9} + 1)}$$

5 虚拟机

6 总结

Vite 的特点总结如下:

- 特点一。

7 致谢

致谢部分。

参考文献

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[2] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.

[3] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform (2013). *URL {http://ethereum. org/ethereum.html}*, 2017.

[4] Chris Dannen. *Introducing Ethereum and Solidity*. Springer, 2017.

[5] Ethan Buchman Jae Kwon. Cosmos a network of distributed ledgers. *URL https://cosmos.network/whitepaper*.

[6] Anonymous. aelf - a multi-chain parallel computing blockchain framework. *URL https://grid.hoopox.com/aelf_whitepaper_en.pdf*, 2018.

[7] Anton Churyumov. Byteball: A decentralized system for storage and transfer of value. *URL https://byteball.org/Byteball.pdf*.

[8] Serguei Popov. The tangle. *URL https://iota.org/IOTA_Whitepaper.pdf.*

[9] Colin LeMahieu. Raiblocks: A feeless distributed cryptocurrency network. *URL https://raiblocks.net/media/RaiBlocks_Whitepaper___English.pdf.*

[10] Aggelos Kiayias Alexander Russell Bernardo David, Peter Gazi. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. *URL https://eprint.iacr.org/2017/573.pdf*, 2017.

[11] Patrick Dai, Neil Mahi, Jordan Earls, and Alex Norta. Smart-contract value-transfer protocols on a distributed mobile application platform. *URL https://qtum.org/uploads/files/cf6d69348ca50dd985b60425ccf282f3.pdf*, 2017.

[12] Anonymous. Byzantine consensus algorithm. *URL https://github.com/tendermint/tendermint/wiki/Byzantine-Consensus-Algorithm.*

[13] Carlos Baquero Marek Zawirski Marc Shapiro, Nuno Preguiça. Conflict-free replicated data types. *URL https://hal.inria.fr/inria-00609399v1*, 2011.

[14] Jayant V Deshpande. On continuity of a partial order. *Proc. Amer. Math. Soc. 19 (1968), 383-386*, 1968.

[15] Eric W Weisstein. Hasse diagram. *URL http://mathworld.wolfram.com/HasseDiagram.html.*

[16] Chunming Liu. Snapshot chain: An improvement on block-lattice. *URL https://medium.com/@chunming.vite/snapshot-chain-an-improvement-on-block-lattice-561aaabd1a2b.*