时间结绳：一种公平事务定序共识系统

李宽
kuan@mail.nwpu.edu.cn

2021 年 5 月 20 日

（正文只有 9 页，阅读大约需要 5 分钟）

摘要

本文提供一种公平事务定序共识系统“时间结绳”，其中方法包括：节点不间断地构造相连的绳结，将接收到的事务立即引入当前绳结中，形成事务到达本节点时序证据；节点之间同步所述证据，各自还原出对于事务进入网络的时序判断，并经共识处理形成事务进入网络的时序共识；各节点将已共识事务依次引入共识绳并执行事务；进一步地，利用共识绳定义了一种共识时间。

时间结绳系统可以实现事务公平定序共识，解决现有技术中因信息损失而造成的事务定序能力不足，因竞选主节点而造成不公平竞争和浪费，因主节点主观操纵而造成的事务定序失公平，以及由上述问题衍生的问题。本文将对时间结绳所需的最小完整模型内容做出阐述。

简介

事务定序决定了状态变化轨迹，而改变一个状态的机会是一次性资源。竞序场景对公平提出要求。现有，事务定序受到中心化机构或“去中心化”网络权威节点的操纵，进而大量能源被浪费。对于习惯使用强权和金元来获取优先权的少数人来说，上述平台是“公平的”，但这其实侵害了主流群体参与文明的时间权利，最终伤害平台的公信力和吸引力。

时间结绳能为先到先得提供透明依据。凡是需要“公平地抢”的场景都可以用时间结绳，例如证券交易、游戏抢装备、抢门票/车票、电商抢购、艺术品竞拍、元宇宙时间、NFT 确权，甚至还能确定 PoW 出块顺序和跨链交易顺序。当然，时间结绳并不是要求任何场景都只能先到先得，但是能为先到先得提供依据，至于权利让渡和规则叠加则交由应用层自主实现。

相关工作

事务定序共识是指网络中各节点不信任外部时间，而仅通过网络内部的计算和通信手段形成不被少数操控的时间机制，（也就是网络内生时间机制），对进入网络的一组事务的因果顺序达成不可逆的一致性共识，从而使各数据副本状态有一致的变化轨迹。

受狭义相对论影响，研究者们把不同的节点理解为不同参考系中的观察者，并且认为不同参考系的观察者对事务的因果顺序有不同的认识，只能通过在不同参考系中传递事务因果顺序来形成共

识，于是要先选定以哪一个参考系中的观察者所判断的事务因果顺序为准，也就是确定主参考系（对应选主）；并将主参考系的观察者所判定的事务因果顺序强制推行到其他参考系（对应出块），作为整个网络对事务因果顺序的共识（对应上链）。

于是，事务定序共识的“公平性”被妥协为，不能由单一参考系一直垄断事务定序权，而是要设计一种机制来随机变换主参考系，但这并没有改变单一参考系在成为主参考系后对事务因果顺序的主观操纵能力，没有根本上解决公平问题。然后，由某一参考系判定的事务因果顺序本身不具有排他性，只能由后续多轮次的参考系通过继承这种判定来逐渐巩固这种事务因果顺序的不可逆转性，于是事务因果顺序的最终确认必然存在明显的滞后性。最后，单一参考系对一组因果顺序的判定只能从该参考系出发向全网其他参考系传播，并且预留足够的传播时间使所述判定能尽量到达各参考系，必然导致一组事务的共识效率难以提升。

上述这些问题会导致主体对于所提交的事务被如何定序及何时执行缺乏稳定的预期，对系统的公平性提出质疑，并承担越来越高的成本，从而不愿意使用这种系统。因此，现有技术局限于对事务定序实时性、公平性和吞吐量要求不高的业务场景。

何为公平

事务进入网络的瞬间表示主体发起的事务已完成向网络空间的交付，事务进入网络的时序是客观的。因此，将事务进入网络的时序作为事务因果顺序是最为客观公正的。

此外，还要避免节点对事务定序的主观操控。至少有四层思路：第一，将节点操纵事务因果顺序的机会窗口压缩到小于所需的反应时间；第二，让节点操纵因果顺序所需的机会成本大于所能获得的预期收益；第三，让节点不操纵事务因果顺序所能获得的收益大于操纵事务因果的预期收益；第四，不设主节点，让节点操纵的事务因果顺序只作为中间结果而无法侵入到最终结果，根本上杜绝操纵的负面影响。

技术思路

时间机制是定序的基础。一般的，人类将一个或一组可持续、可预测、不可逆、公共可见、变化足够频繁、不受人类控制的状态变化轨迹作为参考系，（即时间定义），设计一种计数系统对参考系的状态进行编号，（即时间解释），当参考系产生新状态时就要将当前编号更换为新状态对应的编号，（即时间推进），将事务分别与该参考系的状态编号建立映射，（即计时或盖时间戳），并以一种不易失和可举证的方式记录这些映射，（即时序举证），再以状态的先后顺序确定事务因果顺序，（即定序，时序确定）。

网络捕获信息消除事务因果的不确定性并形成对事务因果顺序的公共知识，保持各节点不断恢复认知平衡。信息越充分，消除的不确定性就越多，就越能输出确定的知识。进一步地，已共识的事务序列可以作为时间参考系，由于每个节点都按照相同规则维护相同的已共识的事务序列，时间定义权、时间解释权、时间推进权就分散于每个节点，无法被某个节点主观操控。所定义的时间又可用于事务到达节点的计时，从而又增加了一个信息维度。

一组事务一般从多个节点进入共识网络，于是每个节点都无法直接测得所有事务进入共识网络的时序，但是每个节点都直接测得事务到达本节点的时序。这就需要找到一种方法，使各节点通过分析事务到达节点的时序信息判断出事务进入网络的时序信息，继而形成对事务进入网络的时序共识。第一，由于事务通过某一个共识节点进入网络，那么该节点就有能力判断该事务进入网络的时间必然晚于之前接收到的其他事务；第二，由于先进入网络的事务大概率先到达更多的共识节点，那么共识节点综合其他共识节点接收事务的顺序可以对比出哪个事务率先到达更多的节点，也就可以大概率正确判断事务进入网络的先后顺序；第三，节点间多轮次分享自己对事务进入网络的时序判断，并接受多数结果，最终可以收敛出对事务进入网络的时序的一致性判断，从而达成共识。

模型定义

各种区块链系统是事务定序共识系统在不同时间分辨率情况下的特例。所有的事务定序共识系统都在维护一种表达时间戳的有向无环依赖关系和事务与时间戳绑定关系的数据结构，并使数据实现一定程度上不可逆且不可篡改的增量存储。一个时间戳至少要前向依赖一个时间戳，但可以绑定或不绑定事务。通过时间戳的有向无环依赖关系可以排列出时间戳的次序，再根据时间戳与事务的绑定关系可以排列出事务的次序，最终完成事务的定序——这就是事务定序机制。事务定序共识系统本质在维护一个时钟。每个时间戳的产生都意味着时间推进，时间戳的产生机制决定了共识机制。

时间模型

状态：在一个有记忆的系统中，一个或一组主体设定若干个参考项来描述一个事物，当这些参数项的值一定时，则称这个事物“处于一个状态”，当这些参考项中任意一个值发生了变化，或者参考项的个数发生了变化时，则称这个事物发生了“状态变化”。其中，采用C表示可选的参考项的键名的集合，s 表示一个状态，由一组参考项的键值对表达，右上角（n）为状态编号。

$$C = \{c_0, c_1, \cdots, c_k, \cdots\}$$
$$s^{(n)} = \left(x_0^{(n)}, x_1^{(n)}, \cdots, x_k^{(n)}\right), \quad \forall x_i^{(n)} \in s^{(n)} \Rightarrow x_i^{(n)} = (c_j, v^{(n)}), c_j \in C$$

一个事物的状态从这个事物上一个状态变化而来，也就是每次状态变化都是从一个“始态”变

为一个“终态”。采用S表示这种状态变化关系，而所述状态变化关系也是一种事物，因而S也可以表示所述状态变化关系作的状态，右上角的（n）为状态的编号。

$$S^{(n+1)}=\left(S^{(n)},s^{(n+1)}\right)$$

状态变化轨迹：当一个事物的状态不断变化，这个事物也随之产生新的状态变化关系，历次产生的状态变化关系按照产生的顺序形成一个序列，称为这个事物的“状态变化轨迹”，记作Ψ。一个事物的状态变化轨迹必然是单向延长的。

$$\Psi = S^{(0)},S^{(1)},\cdots,S^{(n)},\cdots$$

通过一组连续的自然数与状态变化轨迹中的元素建立一一映射的关系，自然数的大小顺序与状态变化轨迹中的元素产生的顺序一一对应。

进一步地，约定，状态展开集合是状态中所有参考项的集合，记作，

$$\widetilde{S}^{(n)}=\left\{x_0^{(n)},x_1^{(n)},\cdots,x_k^{(n)},\cdots\right\}$$

状态轨迹的展开集合是轨迹中所有元素的展开集合的元素的集合，记作，

$$\widetilde{\Psi}=\left\{x_0^{(0)},x_1^{(0)},\cdots,x_k^{(0)},\cdots x_0^{(n)},x_1^{(n)},\cdots,x_k^{(n)},\cdots\right\}$$

事件：将一个事物的一次状态变化称为一个事件，也把一组事件称为一个事件。

时间：时间是一个或一组主体对所在系统的状态变化的感受。当任意一个事物的状态发生变化，或者一个主体认为一个事物的状态发生了变化时，则被主体视为时间变化。

时间参考系：由于现实中任何一个主体都无法穷尽认识所有事物的所有状态变化，为了简化了主体感知时间变化和描述时间感受的难度，一个或一组主体协定将且仅将一个或多个事物的状态变化轨迹作为时间参考系。时间参考系记作：

$$\Theta=\left\{\Psi_i\middle|i\in N,\Psi_i=S_i^{(0)},S_i^{(1)},\cdots,S_i^{(n)},\cdots\right\}$$

时刻：当时间参考系被当作一个整体时，时间参考系也有状态变化轨迹，则时间参考系的状态变化轨迹中的每一项是一个时刻。

时延：两个时刻定义的时间区间的时间长度。

时序：利用所述时间参考系的状态变化轨迹来标记事件发生时间或预期发生时间的顺序。已知事件 0 在时刻 i 和时刻 m 之间发生，且事件 1 在时刻 m 和时刻 n 之间发生，且已知时刻 j 早于时刻 m，则可以确定事件 0 早于事件 1 发生。

$$S^{(i)}\prec e_0\prec S^{(j)}\wedge S^{(m)}\prec e_1\prec S^{(n)}\wedge S^{(j)}\prec S^{(m)}\Rightarrow e_0\prec e_1$$

时间戳：为了书写交流方便，采用符号与时间参考系的时刻一一对应，所述符号为时间戳。由

于所述时间参考系状态变化轨迹不断延长，所述时间戳也需要有相应生成方式。根据时间参考系状态变化轨迹的时刻对应生成时间戳的方法，称为“历法”。

$$t_n \leftrightarrow n \leftrightarrow S^{(n)},\ \ n \in N$$

$$t_n = (n, \varphi(S^{(n)})),\quad s.t. \forall i \neq j \Leftrightarrow t_i \neq t_j$$

$$t_n \prec t_{n+1} \Leftrightarrow S^{(n)} \prec S^{(n+1)}$$

$$\Gamma = t_0, t_1, \cdots, t_n, \cdots$$

计时：将事件与一个时间戳建立关联的行为，则称为事件计时。

同时：如果一组事件与相同的时间戳建立关联，则认为这组事件同时发生。

时钟：一套模拟时间参考系状态变化且能直接或间接输出当前时间戳的设备。主体对于时间参考系的认识，也是一种对时间参考系的模拟，也可以视作一个时钟。

时间分辨率：一个时钟所能表达的最小时延，称为所述时钟的时间分辨率。时间分辨率体现了时钟帮助主体分辨时间变化和事件时序的能力。

时间定义：当时间参考系和历法确定以后，主体对于时间的理解就确定了，因此把确定时间参考系和历法的行为称为“时间定义”。

时间解释：当时间被定义之后，把确定当前时间的行为称为“时间解释”。

时间推进：在确定当前时间后，把产生当前时间戳的行为，称作“时间推进”。

网络时间：将网络内部的事物状态变化轨迹作为时间参考系所定义的时间。采用网络内生时间是为了摆脱对网络外部时间源的依赖。而去中心化的网络内生时间指的是时间不由某个中心控制和授时，而是由网络内的节点通过某种分散的机制共同推进的时间。

网络模型

时间结绳系统由若干个事务输入模块、状态获取模块和一个共识网络组成。

事务：主体所产生的变更某个目标事物状态的意志的载体。主体通过发起事务来表达自己变更某个事务状态的意志，当事务被接受且满足执行条件后就会被执行，事务的执行会改变目标事物的状态。

$$T : s^{(n)} \rightarrow s^{(n+1)}$$

$$T : S^{(n)} \rightarrow S^{(n+1)}$$

用 G 表示共识网络拓扑，V 表示共识网络的节点集合，E 表示节点连接关系集合。例如，一个拥有 4 个共识节点的共识网络表示为，

$$G = (V, E)$$

$$V = \{n_0, n_1, n_2, n_3\}$$

$$E = \{\langle n_0, n_1\rangle, \langle n_0, n_2\rangle, \langle n_1, n_2\rangle, \langle n_1, n_3\rangle, \langle n_2, n_3\rangle\}$$

入口节点：主动发起事务或接受其他主体的委托向网络内输入事务的共识节点，称为所述事务的入口节点。一个事务输入模块向共识网络的某个入口节点输入事务，被表达为，

$i:T^{(n)} \mapsto n, i \in I \in M, n \in V$

事务进入网络：一个事务被入口节点完成计时并签名之后从入口节点向其他节点发出，视作该事务进入了共识网络。本文提供的实施例假设事务被入口节点盖时间戳后立即签名并立即发出，于是用事务的共识时间戳表示事务进入网络的时间。

事务同步：一个过程，即事务在节点之间转发，以尽量使得网络内所有节点都获取到所述事务。事务在节点之间的转发关系表达为，

$n_j:T^{(n_i)} \mapsto n_k,\ n_i,n_j,n_k \in V, j \neq k, i \neq k$

事务到达节点：一个事务以本节点为入口节点或者被其他共识节点转发向本节点，被节点获取，视作事务到达了本节点。本文假设事务被节点获取后被立即处理并填写本地时间戳和共识时间戳并立即引入本地到达绳。

顺序模型

本真序：事务输入模块各自向共识网络输入事务，客观上产生了事务进入共识网络的时序，称为“本真序”。共识网络是分布式的，事务输入模块输入的事务会以共识网络中不同的共识节点为入口进入共识网络，在现有技术条件下不存在某个时钟能立即记录下所有事务进入网络的时序，而共识节点也不会以任何外部的时间戳作为事务排序的依据。因此，所述的本真序不会在事务进入网络后立即被共识网络的所有共识节点获取。记作，

$\mathrm{O}_T = T_0 \prec T_1 \prec T_2 \prec T_3 \prec \cdots$

到达序：共识网络中的各共识节点获取事务并向其他共识节点同步事务，客观上产生了事务到达各共识节点的时序，称为“到达序”。各共识节点通过各自构建的时钟记录下到达序。所述事务“到达”一个共识节点的方式包括：事务以所述共识节点为入口共识节点从而到达了所述共识节点，或者事务进入共识网络后经过同步机制被传播到了所述共识节点。显然，各共识节点记录的到达序可能不同，到达序也不等同于本真序。打比方说，池塘上的浮标有两种方式知道一个雨滴落在了池塘上，一种是雨滴刚好落在了浮标上，另一种是雨滴落在水面上后散出来的水波推进到了浮标。所述到达一般指的是首次到达。记作，

$\mathrm{O}_A^{n_0} = T_0^{(n_0)} \prec T_1^{(n_1)} \prec T_2^{(n_4)} \prec T_3^{(n_0)} \prec \cdots, \quad n_i \in V$

还原序：共识节点之间同步各自的到达序记录，通过预设规则分析出每个到达序记录中蕴含的部分本真序信息，并将各部分本真序信息整合，从而各自还原出对本真序的判断。所述共识节点还

原出的对本真序的判断，称为“还原序”。记作，

$$\mathrm{O}_R^{(n_0)} = T_0^{(n_0)} \prec T_1^{(n_1)} \prec T_2^{(n_4)} \prec T_3^{(n_0)} \prec \cdots, \quad n_i \in V$$

共识序：共识网络对一组事务的还原序的一致认识。共识节点之间共享各自的还原序，通过预设的共识处理方法形成对本真序的一致性判断。一般的，共识序一旦形成后不允许逆转，只能在末端加入新的共识事务而单向延长。于是，共识节点可以将共识序的一次单向延长作为共识网络的一次时间推进，即把共识序作为共识网络公用的时钟，共识节点可以利用共识序记录时间。记作，

$$\mathrm{O}_C = T_0 \prec T_1 \prec T_2 \prec T_3 \prec \cdots$$

时序还原度：一种度量指标，即通过事务定序共识流程所输出的共识序与所述事务的本真序之间的相似度，反映的是事务定序共识系统多大程度上还原了本真序。

结绳模型

结绳：一种数据结构，记作：Φ 。包括：一组连续生成的绳结，其中，每个绳结在上一个绳结生成后立即开始打结创建，且依赖所述上一个绳结的数据。绳结的结构包括：绳结编号、事务引用、校验项和校验规则；绳结的符号记作：K

时间结绳：一种用于定义时间和事件计时的结绳，所述结绳的状态变化轨迹作为时间参考系。时间结绳蕴含的信息本质是时间戳集合、事务集合、时间戳的顺序关系集合、事务与时间戳的绑定关系集合组成的四元组，其中，一个时间戳可以绑定 0 个或 1 个或多个事务。

$$\Phi \to \left(\Lambda, T, \left\{\left\langle t_i, T_j \right\rangle \middle| t_i \in \Lambda, T_j \in T\right\}, \left\{t_i \prec t_j \middle| t_i \in \Lambda, t_j \in \Lambda\right\}\right)$$
$$\Lambda = \left\{t_0, \cdots, t_n\right\}$$
$$T = \left\{T_0, \cdots, T_m\right\}$$

到达绳：一种由共识节点构造，定义本地时间，并记录事务到达序的时间结绳，是一种共识节点以结绳方式记录的事务到达时序数据。由于所述结绳记录了事务到达共识节点的时序证据，故称作所述结绳为“到达绳”。其中，一般的，一个共识节点只构造一条到达绳。例如，一条由节点 n0 构造的本地到达绳可以被记作：

$$\Phi_A^{(n_0)} = K_{(T_0)}^{(0)} \prec K_{(T_1)}^{(12)} \prec K_{(T_2)}^{(60)} \prec K_{(T_3)}^{(130)} \prec K_{(T_4)}^{(400)} \prec K_{(T_5)}^{(425)}$$

其中，空结（引入 0 个事务的绳结）被省略不计，只列出事务结。

所述到达绳相比于比特币方案的交易缓冲区，至少增加了事务到达共识节点、事务到达共识节点的时间、事务输入模块三个信息维度，可以极大地扩展了信息表达能力。

到达绳不仅能反映事务到达共识节点先后时序，还能一定程度上反映事务到达共识节点的频率变化，进一步还能反映当前网络拥堵情况，还能近似反映不同共识节点的相对打结频率，还能反映

出本地事务晚于此前到达的外地事务进入网络。

从安全性角度讲，由于结绳的每个绳结都是前向依赖的，如果想逆转两个事务的顺序就必须重新花费时间构造所涉及的绳段，而在构造的过程中又需要把不断新接收的事务引入结绳，否则会积压多个待引入的事务而不得不连续地引入事务，这种伪造行为会在结绳上留下痕迹并且很容易被其他节点发现。然而，单一节点对事务到达时序的伪造难以影响到整个网络最终对事务进入网络时序的共识。可见，节点伪造到达绳是徒劳的。

还原绳：一种由共识节点构造，记录事务还原序的数据结构。例如，由节点 n0 构造的地到达绳可以被记作：$\Phi_R^{(n_0)}$

共识绳：一种由共识节点构造，定义共识时间，记录事务共识序的结绳。

解决方案

这里只列出了最基本的步骤，更多的实现细节、优化细节，包括共识节点的分工，以及硬件设备的基本组成，详见《时间结绳技术白皮书》。

时间结绳的事务定序共识方法，其特征在于，包括：

获取待共识事务和相应的事务信息，所述事务信息包括事务编号、事务输入模块信息、事务入口节点信息，事务待执行信息和获取待共识事务的路径信息；

根据所述获取待共识事务的路径信息，将所述待共识事务划分为本地事务和外地事务；

验证、解析、同步所述待共识事务；

获取共识节点上所述本地事务的到达序信息，将所述本地事务引入不间断打结的本地到达绳；

同步不同共识节点之间的本地到达绳，得到同步后各节点到达序信息；

根据预设还原规则分析所述各节点到达序信息，得到本节点中两两事务的还原序以及一组事务连续的还原序信息，构建本地还原绳；

交换和同步不同共识节点之间的本地还原绳，得到同步后各节点还原序信息；

根据预设共识规则对本节点还原序信息进行共识处理，得到共识序信息，构建共识绳，所述共识绳包含已共识事务；

根据对所述共识绳解析，得到共识绳中已共识事务；

根据已共识事务更新状态数据。

附录中，我们通过理论模拟和实验模拟两种方式进行原型演示。

经济模型

我们认为，一个事务被打结到时间结绳上，本身就是一件很有意义的事。

时间结绳平台本身不发行同质化的数字货币，是真正的“无币”项目。但是共识绳可以向应用层无限供应 NFT，应用层可以自行决定是否发行应用内数字货币。所有基于时间结绳技术发行数字货币的项目需自行解决合规性问题，且自愿使用时间结绳技术。时间结绳平台与应用解耦，只对交易的数字摘要进行定序，对交易内容不知情，予以免责声明。

我们已建立了时间结绳社区，接受时间结绳技术支持者的捐赠（优先接受数字人民币），捐赠记录会被打结到时间结绳上生成 NFT，并将捐赠内容向全网公开。

时间结绳社区欢迎志愿者加入，积极为公社发展做出贡献，贡献记录会打结到时间结绳上生成 NFT，并按照志愿者意愿决定是否公开。

结语

当人类涌入硅基文明之后，稀缺性开始凸显，事务因果不可忽略，公平竞序就成了必然要求。发端于狭义相对论的传统分布式共识理论和区块链技术无法满足这一要求。为此，本文提出了时间结绳理论、技术、系统、项目的全部概要内容，未来将不断更新。

参考文献

[1] Lamport, Leslie. Time, Clocks, and the Ordering of Events in a Distributed System[J]. Communications of the Acm, 1978, 21:558-565.

附录

理论模拟

模拟一个事务定序共识系统，包括一个共识网络 G，若干个事务输入模块 I 和若干个状态获取模块 F。其中，

事务输入模块有 3 个，记作：

$$I = \{i_0, i_1, i_2\}$$

状态获取模块有 3 个，记作：

$$F = \{f_0, f_1, f_2\}$$

节点 0、1、2、3 组成一个共识网络，记作：

$$G=(V,E)$$
$$V=\{n_0,n_1,n_2,n_3\}$$
$$E=\{\langle n_0,n_1\rangle,\langle n_0,n_2\rangle,\langle n_1,n_2\rangle,\langle n_1,n_3\rangle,\langle n_2,n_3\rangle\}$$

为了便于计算和说明，假设各个节点之间的延迟 D 分别为一个固定值，具体为：

$$D(n_0,n_1)=3,D(n_0,n_2)=2,D(n_1,n_2)=2,D(n_1,n_3)=4,D(n_2,n_3)=2$$

为了模拟一组事务在共识网络中被共识的过程，假设客观存在一个本真序，

$$\mathrm{O}_T=T_0\prec T_1\prec T_2\prec T_3\prec T_4\prec T_5$$

并且，假设存在一种绝对时间，按照绝对时间和本真序设计一个事务进入网络的计划表。其中，本真序中的事务由事务输入模块在指定时刻输入给指定节点。例如，在时刻 10，事务输入模块 1 以节点 0 为入口节点输入事务 T0，以此类推如下：

$$t(i_1:T_0\to n_0)=10.0$$
$$t(i_0:T_1\to n_1)=12.8$$
$$t(i_1:T_2\to n_2)=14.6$$
$$t(i_0:T_3\to n_3)=16.4$$
$$t(i_1:T_4\to n_0)=18.2$$
$$t(i_0:T_5\to n_1)=20.0$$

开始计时后，按照计划表由指定事务输入模块将指定事务输入共识网络的指定入口节点。假设 t=0 时，各节点的本地事务到达序开始打结，分别如下：

$$\Phi_A^{(n_0)}=K_{a(n_0)}^{(0)}$$
$$\Phi_A^{(n_1)}=K_{a(n_1)}^{(0)}$$
$$\Phi_A^{(n_2)}=K_{a(n_2)}^{(0)}$$
$$\Phi_A^{(n_3)}=K_{a(n_3)}^{(0)}$$

由于各节点打结频率不同，当 t=10 时，各节点的本地事务到达序如下，

$$\Phi_A^{(n_0)}=K_{a(n_0)}^{(0)}\prec K_{\varnothing}^{(1)}\prec\cdots\prec K_{\varnothing}^{(1000)}$$
$$\Phi_A^{(n_1)}=K_{a(n_1)}^{(0)}\prec K_{\varnothing}^{(1)}\prec\cdots\prec K_{\varnothing}^{(2000)}$$
$$\Phi_A^{(n_2)}=K_{a(n_2)}^{(0)}\prec K_{\varnothing}^{(1)}\prec\cdots\prec K_{\varnothing}^{(3000)}$$
$$\Phi_A^{(n_3)}=K_{a(n_3)}^{(0)}\prec K_{\varnothing}^{(1)}\prec\cdots\prec K_{\varnothing}^{(4000)}$$

此时，T0 被入口节点 0 获取，节点 0 判断 T0 为本地事务，验证、解析、存储和同步本地事务 T0。节点 0 将事务 T0 打结到本地事务到达序的 1001 号 Knoten，记作：

$$\Phi_A^{(n_0)}=K_{a(n_0)}^{(0)}\prec K_{\varnothing}^{(1)}\prec\cdots\prec K_{\varnothing}^{(1000)}\prec K_{(T_0)}^{(1001)}$$

此时，节点 0 将本地事务到达序的未同步 segment 进行同步。

当 t=11 时，由节点 0 同步的 T0 到达了节点 1，节点 1 判断 T0 为外地事务，验证、解析、存

储和同步外地事务 T0。此时，节点 1 的本地事务到达序末尾编号为 2200，于是节点 1 将事务 T0 打结到本地事务到达序的 2201 号 Knoten，记作：

$$\Phi_A^{(n_1)} = K_{a(n_1)}^{(0)} \prec K_{\varnothing}^{(1)} \prec \cdots \prec K_{\varnothing}^{(2200)} \prec K_{(T_0)}^{(2201)}$$

此时，节点 1 将本地事务到达序的未同步 segment 进行同步。

经过一段时间后，各事务都已经输入到了共识网络中且同步到各节点，且各节点的本地事务到达序也都同步到了共识网络的其他节点。

为简化计算，将节点为了同步一个事务而在本地产生的处理时间忽略不计，则根据节点之间的延迟设定可以知道事务到达各个节点的“绝对时间”为：

$$t(T_0 \to n_0)=10,\quad t(T_0 \to n_1)=13,\quad t(T_0 \to n_2)=12,\quad t(T_0 \to n_3)=14$$
$$t(T_1 \to n_0)=15.8,\quad t(T_1 \to n_1)=12.8,\quad t(T_1 \to n_2)=14.8,\quad t(T_1 \to n_3)=16.8$$
$$t(T_2 \to n_0)=16.6,\quad t(T_2 \to n_1)=16.6,\quad t(T_2 \to n_2)=14.6,\quad t(T_2 \to n_3)=16.6$$
$$t(T_3 \to n_0)=20.4,\quad t(T_3 \to n_1)=20.4,\quad t(T_3 \to n_2)=18.4,\quad t(T_3 \to n_3)=16.4$$
$$t(T_4 \to n_0)=18.2,\quad t(T_4 \to n_1)=21.2,\quad t(T_4 \to n_2)=20.2,\quad t(T_4 \to n_3)=22.2$$
$$t(T_5 \to n_0)=23,\quad t(T_5 \to n_1)=20,\quad t(T_5 \to n_2)=22,\quad t(T_5 \to n_3)=24$$

各节点的事务到达序数据如下：

$$\Phi_A^{(n_0)} = K_{a(n_0)}^{(0)} \prec K_{(T_0)}^{(1001)} \prec K_{(T_1)}^{(1581)} \prec K_{(T_2)}^{(1661)} \prec K_{(T_4)}^{(1821)} \prec K_{(T_3)}^{(2041)} \prec K_{(T_5)}^{(2301)}$$
$$\Phi_A^{(n_1)} = K_{a(n_1)}^{(0)} \prec K_{(T_1)}^{(2561)} \prec K_{(T_0)}^{(2601)} \prec K_{(T_2)}^{(3321)} \prec K_{(T_5)}^{(4001)} \prec K_{(T_3)}^{(4081)} \prec K_{(T_4)}^{(4241)}$$
$$\Phi_A^{(n_2)} = K_{a(n_2)}^{(0)} \prec K_{(T_0)}^{(3601)} \prec K_{(T_2)}^{(4381)} \prec K_{(T_1)}^{(4441)} \prec K_{(T_3)}^{(5521)} \prec K_{(T_4)}^{(6061)} \prec K_{(T_5)}^{(6601)}$$
$$\Phi_A^{(n_3)} = K_{a(n_3)}^{(0)} \prec K_{(T_0)}^{(5601)} \prec K_{(T_3)}^{(6561)} \prec K_{(T_2)}^{(6641)} \prec K_{(T_1)}^{(6721)} \prec K_{(T_4)}^{(8881)} \prec K_{(T_5)}^{(9601)}$$

每个节点在接收到外地事务到达序后，就开始进行事务还原序处理。以节点 0 为例，根据本地和外地事务到达序可以获知事务到达各节点的时序，如下：

$$\mathrm{O}_A^{(n_0)} = T_0 \prec T_1 \prec T_2 \prec T_4 \prec T_3 \prec T_5$$
$$\mathrm{O}_A^{(n_1)} = T_1 \prec T_0 \prec T_2 \prec T_5 \prec T_3 \prec T_4$$
$$\mathrm{O}_A^{(n_2)} = T_0 \prec T_2 \prec T_1 \prec T_3 \prec T_4 \prec T_5$$
$$\mathrm{O}_A^{(n_3)} = T_0 \prec T_3 \prec T_2 \prec T_1 \prec T_4 \prec T_5$$

根据还原规则 1，目前尚未有已共识事务，无法利用所述规则 1。

根据还原规则 2，已知，节点 0 的本地事务有 T0 和 T4，节点 1 的本地事务有 T1 和 T5，节点 2 的本地事务有 T2，节点 3 的本地事务有 T3，则，

$$\left(T_0^{(n_0)} \prec T_4^{(n_0)}\right) \in \mathrm{O}_A^{(n_0)} \Rightarrow \left(T_0 \prec T_4\right) \in \mathrm{O}_R$$

$$\left(T_1^{(n_1)} \prec T_5^{(n_1)}\right) \in \mathrm{O}_A^{(n_1)} \Rightarrow \left(T_1 \prec T_5\right) \in \mathrm{O}_R$$

根据还原规则 3，

$$\left(T_1^{(n_1)} \prec T_4^{(n_0)}\right) \in \mathrm{O}_A^{(n_0)} \wedge \left(T_1^{(n_1)} \prec T_4^{(n_0)}\right) \in \mathrm{O}_A^{(n_1)} \Rightarrow \left(T_1 \prec T_4\right) \in \mathrm{O}_R$$

$$\left(T_2^{(n_2)} \prec T_4^{(n_0)}\right) \in \mathrm{O}_A^{(n_0)} \wedge \left(T_2^{(n_2)} \prec T_4^{(n_0)}\right) \in \mathrm{O}_A^{(n_2)} \Rightarrow \left(T_2 \prec T_4\right) \in \mathrm{O}_R$$

$$\left(T_0^{(n_0)} \prec T_5^{(n_1)}\right) \in \mathrm{O}_A^{(n_1)} \wedge \left(T_0^{(n_0)} \prec T_5^{(n_1)}\right) \in \mathrm{O}_A^{(n_0)} \Rightarrow \left(T_0 \prec T_5\right) \in \mathrm{O}_R$$

$$\left(T_2^{(n_2)} \prec T_5^{(n_1)}\right) \in \mathrm{O}_A^{(n_1)} \wedge \left(T_2^{(n_2)} \prec T_5^{(n_1)}\right) \in \mathrm{O}_A^{(n_2)} \Rightarrow \left(T_2 \prec T_5\right) \in \mathrm{O}_R$$

$$\left(T_0^{(n_0)} \prec T_2^{(n_2)}\right) \in \mathrm{O}_A^{(n_2)} \wedge \left(T_0^{(n_0)} \prec T_2^{(n_2)}\right) \in \mathrm{O}_A^{(n_0)} \Rightarrow \left(T_0 \prec T_2\right) \in \mathrm{O}_R$$

$$\left(T_0^{(n_0)} \prec T_3^{(n_3)}\right) \in \mathrm{O}_A^{(n_3)} \wedge \left(T_0^{(n_0)} \prec T_3^{(n_3)}\right) \in \mathrm{O}_A^{(n_3)} \Rightarrow \left(T_0 \prec T_3\right) \in \mathrm{O}_R$$

根据还原规则 5，结合规则 2 和 3 的结果，

$$\left(T_0 \prec T_2\right) \in \mathrm{O}_R \wedge \left(T_2 \prec T_5\right) \in \mathrm{O}_R \Rightarrow \left(T_0 \prec T_2 \prec T_5\right) \in \mathrm{O}_R$$

$$\left(T_0 \prec T_2\right) \in \mathrm{O}_R \wedge \left(T_2 \prec T_4\right) \in \mathrm{O}_R \Rightarrow \left(T_0 \prec T_2 \prec T_4\right) \in \mathrm{O}_R$$

目前还不确定 T0 与 T1，T1 与 T2，T1 与 T3，T2 与 T3，T2 与 T4，T3 与 T4，T4 与 T5 的还原序关系。

根据还原规则 4：对于 T0 与 T1 来说，在节点 0、2、3 中均有 T0 早于 T1 到达，只有节点 1 是 T1 早于 T0，则判断 T0 早于 T1 进入网络，同理可以判断出 T1 早于 T3，T2 早于 T3，T2 早于 T4，T3 早于 T4，T4 早于 T5，此时再根据规则 5，有，

$$\left(T_0 \prec T_1 \prec T_3 \prec T_4 \prec T_5\right) \in \mathrm{O}_R$$

$$\left(T_0 \prec T_2 \prec T_3 \prec T_4 \prec T_5\right) \in \mathrm{O}_R$$

但 T1 和 T2 各有两个节点的到达序早于对方，目前还不确定 T1 与 T2 的还原序关系。

根据还原规则 6，获取 T0 和 T1 在本地和外地事务到达序的 Knoten 编号并计算。

$$\Phi_A^{(n_0)} : \frac{\left\| K_{(T_1)} - K_{(T_2)} \right\|}{\left\| K_{(T_0)} - K_{(T_5)} \right\|} = \frac{-80}{1300}, \quad \Phi_A^{(n_1)} : \frac{\left\| K_{(T_1)} - K_{(T_2)} \right\|}{\left\| K_{(T_1)} - K_{(T_4)} \right\|} = \frac{-760}{1680},$$

$$\Phi_A^{(n_2)} : \frac{\left\| K_{(T_1)} - K_{(T_2)} \right\|}{\left\| K_{(T_0)} - K_{(T_5)} \right\|} = \frac{60}{3000}, \quad \Phi_A^{(n_3)} : \frac{\left\| K_{(T_1)} - K_{(T_2)} \right\|}{\left\| K_{(T_0)} - K_{(T_5)} \right\|} = \frac{80}{4000},$$

$$\frac{-80}{1300} + \frac{-760}{1680} + \frac{60}{3000} + \frac{80}{4000} \approx -4.526 < 0 \Rightarrow (T_1 \prec T_2) \in \mathrm{O}_R$$

至此，节点 0 可以得出还原序，

$$\left(T_0 \prec T_1 \prec T_2 \prec T_3 \prec T_4 \prec T_5\right) \in \mathrm{O}_R$$

同理，节点 1、2、3 根据相同的方法可以得出还原序，

$$\left(T_0 \prec T_1 \prec T_2 \prec T_3 \prec T_4 \prec T_5\right) \in \mathrm{O}_R$$

节点 0 将本地还原序发送给其他节点，其他节点回复本节点的本地还原序，节点 0 对比所收到的外地还原序，发现完全一致，则认定还原序达成共识序。

节点 0 将共识序按顺序写入共识 Knoten，则得到共识 Knoten 为，

$$\Phi_C^{(n_0)} = K_{init}^{(0)} \prec K_{(T_0)}^{(1)} \prec K_{(T_1)}^{(2)} \prec K_{(T_2)}^{(3)} \prec K_{(T_3)}^{(4)} \prec K_{(T_4)}^{(5)} \prec K_{(T_5)}^{(6)}$$

节点 0 按共识序解析共识 Knoten 中的事务，写入事务执行队列，并逐一调取事务对应的事务执行器执行事务执行队列中的事务，完成对状态数据的变更。其他节点也是做相同的操作。例如，所述事务的待执行内容分别为：

T0:要求对变量 A 加 1；

T1:要求对变量 A 加 2；

T2:要求对变量 B 从 2 加 3；

T3:要求对变量 A 减 3；

T4:要求对变量 A 减 2；

T5:要求对变量 B 从 2 加 1；

存在约束条件：A 和 B 不能小于 0；

在所述事务输入网络之前，节点的状态数据包括：

A：值为 1；

B：值为 2。

当共识序开始执行后：

执行 T0，A 的值变为 1+1=2；

执行 T1，A 的值变为 2+2=4；

执行 T2，B 的值为 2，从 2 加 3 变为 5；

执行 T3，A 的值变为 4-3=1；

执行 T4，A 的值 1-2<0，不符合所述约束条件，T4 舍弃；

执行 T5，B 的值为 5，无法从 2 加 1，T5 舍弃；

同时，状态获取模块从任意节点中获取状态数据。例如：

状态获取模块 0 向节点 0 请求 A 的值，A 向节点 0 返回此时 A 的值；

状态获取模块 1 已向节点 2 订阅了 B 的值，每当 B 的值更新时，节点 2 就向所述状态获取模块 1 推送 B 的值；

状态获取模块 2 同时从节点 0、1、2、3 请求 A 的值；

状态获取模块 3 同时从节点 1、2 订阅 A 的值；

实验模拟

为了更好地获取数据，本实验对事务的数据结构进行了简化，每个事务省略去了具体事务信息，用编号代替。通过 100 笔事务，4 个全节点来模拟整个原型系统。

原型适用 Go 语言开发，在 Centos+Docker 环境中运行，实验输入数据如图所示。由客户端（事务输入模块）按顺序向网络发送事务，每个事务随机选择入口节点。

```
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:134  ----- Analyze Sorted Knot List -----
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[6]Payload="0"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[8]Payload="1"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[10]Payload="2"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[11]Payload="3"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[11]Payload="4"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[15]Payload="5"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[18]Payload="6"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[17]Payload="7"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  f8df87497b145[22]Payload="8"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[23]Payload="9"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[26]Payload="10"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[28]Payload="11"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[25]Payload="12"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  f8df87497b145[31]Payload="13"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  f8df87497b145[33]Payload="14"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[32]Payload="15"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  f8df87497b145[37]Payload="16"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[39]Payload="17"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  f8df87497b145[39]Payload="18"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[44]Payload="19"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[47]Payload="20"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[49]Payload="21"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[48]Payload="22"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[52]Payload="23"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[55]Payload="24"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[53]Payload="25"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  6d8aced649a52[56]Payload="26"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  e11dfe7f9c2f4[60]Payload="27"
2020-11-15T00:57:58.833+0800  INFO  calc/analyze_knot.go:136  b72b0f66b294d[61]Payload="28"
```

图 3 实验原型输入数据

```
----- Analyze Sorted Knot List -----  ----- Analyze Sorted Knot List -----  ----- Analyze Sorted Knot List -----  ----- Analyze Sorted Knot List -----
b72b0f66b294d[6]Payload="0"            b72b0f66b294d[6]Payload="0"            b72b0f66b294d[6]Payload="0"            b72b0f66b294d[6]Payload="0"
b72b0f66b294d[8]Payload="1"            b72b0f66b294d[8]Payload="1"            b72b0f66b294d[8]Payload="1"            b72b0f66b294d[8]Payload="1"
b72b0f66b294d[10]Payload="2"           b72b0f66b294d[10]Payload="2"           b72b0f66b294d[10]Payload="2"           b72b0f66b294d[10]Payload="2"
e11dfe7f9c2f4[11]Payload="3"           e11dfe7f9c2f4[11]Payload="3"           e11dfe7f9c2f4[11]Payload="3"           e11dfe7f9c2f4[11]Payload="3"
6d8aced649a52[11]Payload="4"           6d8aced649a52[11]Payload="4"           6d8aced649a52[11]Payload="4"           6d8aced649a52[11]Payload="4"
e11dfe7f9c2f4[15]Payload="5"           e11dfe7f9c2f4[15]Payload="5"           e11dfe7f9c2f4[15]Payload="5"           e11dfe7f9c2f4[15]Payload="5"
e11dfe7f9c2f4[18]Payload="6"           e11dfe7f9c2f4[18]Payload="6"           e11dfe7f9c2f4[18]Payload="6"           e11dfe7f9c2f4[18]Payload="6"
6d8aced649a52[17]Payload="7"           6d8aced649a52[17]Payload="7"           6d8aced649a52[17]Payload="7"           6d8aced649a52[17]Payload="7"
f8df87497b145[22]Payload="8"           f8df87497b145[22]Payload="8"           f8df87497b145[22]Payload="8"           f8df87497b145[22]Payload="8"
b72b0f66b294d[23]Payload="9"           b72b0f66b294d[23]Payload="9"           b72b0f66b294d[23]Payload="9"           b72b0f66b294d[23]Payload="9"
b72b0f66b294d[26]Payload="10"          b72b0f66b294d[26]Payload="10"          b72b0f66b294d[26]Payload="10"          b72b0f66b294d[26]Payload="10"
b72b0f66b294d[28]Payload="11"          b72b0f66b294d[28]Payload="11"          b72b0f66b294d[28]Payload="11"          b72b0f66b294d[28]Payload="11"
6d8aced649a52[25]Payload="12"          6d8aced649a52[25]Payload="12"          6d8aced649a52[25]Payload="12"          6d8aced649a52[25]Payload="12"
f8df87497b145[31]Payload="13"          f8df87497b145[31]Payload="13"          f8df87497b145[31]Payload="13"          f8df87497b145[31]Payload="13"
f8df87497b145[33]Payload="14"          f8df87497b145[33]Payload="14"          f8df87497b145[33]Payload="14"          f8df87497b145[33]Payload="14"
6d8aced649a52[32]Payload="15"          6d8aced649a52[32]Payload="15"          6d8aced649a52[32]Payload="15"          6d8aced649a52[32]Payload="15"
f8df87497b145[37]Payload="16"          f8df87497b145[37]Payload="16"          f8df87497b145[37]Payload="16"          f8df87497b145[37]Payload="16"
f8df87497b145[39]Payload="17"          f8df87497b145[39]Payload="17"          f8df87497b145[39]Payload="17"          f8df87497b145[39]Payload="17"
e11dfe7f9c2f4[39]Payload="18"          e11dfe7f9c2f4[39]Payload="18"          e11dfe7f9c2f4[39]Payload="18"          e11dfe7f9c2f4[39]Payload="18"
f8df87497b145[42]Payload="19"          f8df87497b145[42]Payload="19"          f8df87497b145[42]Payload="19"          f8df87497b145[42]Payload="19"
e11dfe7f9c2f4[44]Payload="20"          e11dfe7f9c2f4[44]Payload="20"          e11dfe7f9c2f4[44]Payload="20"          e11dfe7f9c2f4[44]Payload="20"
e11dfe7f9c2f4[47]Payload="21"          e11dfe7f9c2f4[47]Payload="21"          e11dfe7f9c2f4[47]Payload="21"          e11dfe7f9c2f4[47]Payload="21"
e11dfe7f9c2f4[49]Payload="22"          e11dfe7f9c2f4[49]Payload="22"          e11dfe7f9c2f4[49]Payload="22"          e11dfe7f9c2f4[49]Payload="22"
6d8aced649a52[48]Payload="23"          6d8aced649a52[48]Payload="23"          6d8aced649a52[48]Payload="23"          6d8aced649a52[48]Payload="23"
b72b0f66b294d[52]Payload="24"          b72b0f66b294d[52]Payload="24"          b72b0f66b294d[52]Payload="24"          b72b0f66b294d[52]Payload="24"
b72b0f66b294d[55]Payload="25"          b72b0f66b294d[55]Payload="25"          b72b0f66b294d[55]Payload="25"          b72b0f66b294d[55]Payload="25"
6d8aced649a52[53]Payload="26"          6d8aced649a52[53]Payload="26"          6d8aced649a52[53]Payload="26"          6d8aced649a52[53]Payload="26"
6d8aced649a52[56]Payload="27"          6d8aced649a52[56]Payload="27"          6d8aced649a52[56]Payload="27"          6d8aced649a52[56]Payload="27"
e11dfe7f9c2f4[60]Payload="28"          b72b0f66b294d[61]Payload="29"          e11dfe7f9c2f4[60]Payload="28"          e11dfe7f9c2f4[60]Payload="28"
b72b0f66b294d[61]Payload="29"          e11dfe7f9c2f4[60]Payload="28"          b72b0f66b294d[61]Payload="29"          b72b0f66b294d[61]Payload="29"
```

图 4 实验原型输出数据

实验结果证明，本原型事务共识顺序与输入顺序基本一致，事务时序得到有效还原。