Ultrain 网络的多链架构

By Ultrain Team, March 2019

可扩展性 (Scalability) 是任何一个区块链网络在试图大规模商用的时候所不可避免的问题. 扩容方案的研究一直是业界的热点方向，不同的方案有链上/链下扩容，水平/垂直扩容，多链/分片等等不同的架构设计. 这其中的背景知识和各种方案的相关优劣在此不做详述，本文着重描述伴随 Ultrain 网络的主网上线时的多链架构的设计方案. 复杂架构的设计往往都是需要考虑各种的取舍 (tradeoff)，比方说了理论的复杂性和现实的可行性，长期的愿景与短期的落地，效率和公平等等，我们在此不但记录了我们的在 Ultrain 主网上线时候的多链的实际落地架构，也同时记录了这些设计背后的关于各种取舍的思考，以及会做一些将来架构演进方向的探讨.

主侧链架构概述

Ultrain 的多链可以认为是一种链上的水平扩容方案. 基本思想是将具体的 DApp 放到侧链上去运行，主链只负责管理一些全局性的管理事务. 每条侧链都有他的资源容量上限，当现有的 DApp 到达现有侧链的资源容量上限的时候，新的侧链会被创建用来给新的 DApp 提供资源，以此来实现水平扩容. 每条侧链都运行独立的 DApp 集合，也就是说各个链之间的状态空间是完全隔离的，所以这个多链架构是天然的实现了计算和存储的链间隔离. 主链上不运行具体的业务相关的 DApp，而是只负责整个多链网络的管理工作，比方说全网的账号创建与维护，矿工的注册和代币抵押，矿工节点在不同链间的调度等等.

思考一：是否需要多级主侧链架构，比方说类似以太坊的 Quadratic Sharding 或者 Polkadot 的无限级联架构？我们非常理解分形 (Fractal) 和递归的概念在设计上令人非常的着迷，但是我们认为在现有的一级主侧链架构下，大约可以支撑上万个节点量级的网络的运行（单链 80~320 节点 * up to ~50 条侧链), 这样的容量已经能够允许 Ultrain 网络在相当长的时间内支撑大量的严肃的大规模商业应用. 因此，在这个版本的设计上我们选择了简单的一级主侧链架构.

思考二：是否需要设计异构（heterogeneous）多链架构？因为 Ultrain 的定位是基础公链项目，而不是链的互操作性 (interoperability) 项目，所以我们选择的是同构（homogeneous）的多链架构, 也就是主侧链都是基于我们的 R-POS 共识机制, 不同的链遵循一样的最终性(Finality) 假设（虽然不同的链可以有不同的 confirmation time，但是都是 deterministic finality）. 这样的设定也简化了我们的链间通信 IBC (Inter-Blockchain Communication) 机制的设计.

对照以下的 “主侧链架构图”，我们描述下主侧链的工作流程:

1. 矿工通过主链的系统合约注册成为 Ultrain 网络的矿工，这个过程需要在主网上抵押 Ultrain 网络的原生代币 UGAS；成功后会进入到等待队列，等待时间过后会被主链系统合约随机调度到特定的侧链进行挖矿.
2. 矿工在侧链挖矿（出块）的结果会被上报到主链的系统合约，被确认有效后由主链来分发挖矿奖励.

3. 主链系统合约会定时的对矿工做随机调度，所以矿工不会一直停留在某条侧链挖矿，这是出于全网安全的考虑，后续章节会详述这个安全设计；调度对矿工来说是透明的，他的长期经济收益预期不会因为具体的侧链分配而改变.

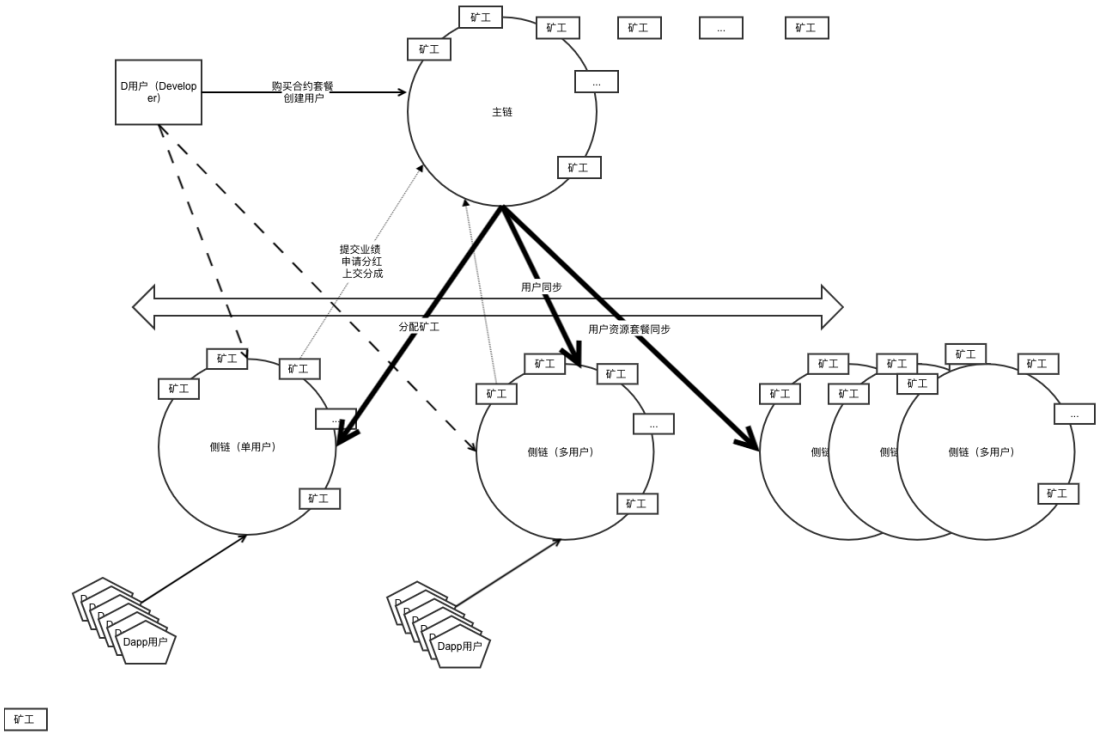

图 1 主侧链架构图

4. 对普通用户来说，用户账号的第一次创建必需通过主链的系统合约进行；主链有着全网所有账户的全貌；当用户需要和某条侧链上特定的 DApp 发生交互的时候，可以申请授权（empower）到特定的侧链，主侧链通信机制会为用户在被授权的侧链上创建同名账户.

5. 对 DApp 开发者来说，当他希望在某个侧链上部署他的 DApp 的时候，他需要通过主链的系统合约来购买针对特定侧链的系统资源套餐 (cpu/storage/net combo)，然后这个资源套餐购买信息也会被同步到特定的侧链，这样这个开发者就可以在特定的侧链上部署和运行他的 DApp 了.

   思考：为什么是资源套餐模型 ？ 当我们在谈水平扩容设计的时候，往往忽视了资源售卖模型在其中扮演的角色. 我们认为现有的一些公链的资源模型，比方说以太坊和 EOS 都是属于超卖模型，而我们的设计是预售模型；超卖模型里面，随着新进资源购买者的增加，所有开发者/用户拥有的系统资源都会被逐渐稀释（直接的，或者间接的通过资源购买成本的提高），这就像一趟火车，当卖出远远大于座位数的票的时候，所有人的空间都变得越来越少；而在预售模型里，我们就最多只卖等于座位数目的票数，这样能保

证所有人都有座位；也就是我们把一条侧链的系统总资源分成若干份，开发者只能按份购买（我们称之为资源套餐），当所有资源售罄的时候，新开发者只能去新的空闲侧链上购买资源；本质上，超卖模型强调的是更多的获取资源售卖的经济利益；而预售模型强调的是资源购买者成本和体验的平衡.

6. 系统原生代币 UGAS 在设计之初是作为 utility token，也就是用来作为系统资源购买方(DApp 部署者) 和系统资源提供方 (矿工) 的结算工具；所以他的主要应用范围是限定在主链上的：作为矿工 Stake 的抵押，挖矿收益的发放，DApp 开发者对系统资源的购买以及一些系统操作的收费. 当然，作为结算工具，可以预见侧链上的 DApp 也会有需要用到 UGAS 作为 DApp 结算手段的需求，因此我们也设计了原生的主侧链之间的 UGAS 跨链转账流通功能，后面会有详述.

矿工的链间随机调度

多链或者分片等水平扩容方案面临的一个核心安全问题是如何在算力/Stake 稀释的情况下仍然保持单链/单分片的安全性？也就是大家常说的 1% attack 问题：当从单链架构变为 100 条侧链/分片的情况下，每条侧链/分片就只有原来的 1% 的算力/Stake，那么攻击这条侧链/分片就只需要攻击原来的 1%的算力/Stake. 现在业界并没有完全公认成熟的解法，我们比较认可且选择实现的思路是矿工在链间的随机调度的方法：也就是矿工同一时间只能在主链指派的一条特定侧链上挖矿，也只有在这条指定侧链上的挖矿产出才能得到奖励；主链会定期的把矿工在不同的侧链间进行调度，这样在 M 条侧链的情况下，虽然每条侧链的总算力/Stake 只有原来的 1/M，但是攻击者“很难”让这 1/M 都是受他控制的算力/Stake. 那么这个“很难”是如何量化的呢？如果我们设定 N 为单侧链的委员会数目，p 为攻击者在全网算力/stake 的总占比，那么，攻击者能实现控制特定侧链的概率（在 R-PoS 共识下需要>=2/3 委员会成员）是一个二项式展开公式：$\sum_{k=\frac{2N}{3}}^{N} p^k\,(1-p)^{N-k}\binom{N}{k}$

	N=40	N=80	N=160
p = 0.4	0.0004	6.1e-7	6.1e-12
P = 0.33	8.5e-8	3.0e-10	2.2e-18
P = 0.25	1.9e-8	1.6e-15	9.7e-29

我们可以看到，当单侧链的委员会数目达到 80 以上的时候，即使恶意攻击者在全网的算力能占到 40%，他能在特定的侧链占到大于 2/3 委员会成员的概率也是非常非常低的. 所以在我们的多链架构下，我们设定的单侧链冷启动最少委员会成员为 40，安全运行的建议值是至少大于 80 个委员会成员. 虽然我们看到单侧链委员会数目增加后，安全性能也随之提升，但是单侧链节点数的增加也会导致网络通信成本增加，单侧链的性能下降，以及全网总体容量的下降（因为侧链数目的减少），所以基于现有的实现，我们推荐单侧链的节点数目为 80 ~ 160，最多不超过 320 个节点.

之前我们有提到侧链与侧链之间的数据是完全隔离的，他的好处是矿工节点每一时刻只需要保存当前指派的侧链的数据，减轻了存储的压力. 但是同时当矿工节点被调度而需要切换链的

时候，就需要同步目标链的数据. 链的同步最常见的就是同步所有的历史块，然后通过回放所有的历史块的来重新构建当前最新的世界状态，然后才能加入网络参与共识. 随着区块的增长，同步所有的历史数据并回放会变成非常耗时的操作，甚至可能一次同步会花费数以天计的时间，这对矿工是非常不友好的，因为矿工不能参与共识出块的时间都是有经济的成本的. 为了加速对链的状态同步过程，我们设计了世界状态快照 (world state snapshot) 机制:

1. 每隔一定的周期，某条侧链的所有矿工节点会把当前的世界状态做成一份快照存储在本地，并把这份快照的 hash 值记录到主链上.

2. 主链的系统合约负责记录侧链的世界快照 hash，只有当某条侧链在特定时刻的所有委员会成员的 2/3 以上的矿工节点上传了一样的世界状态 hash，主链才会认为这个世界状态的 hash 值是有效的；这个可以认为是一次近似的对特定世界状态的共识过程.

3. 当矿工节点被调度到某个侧链的时候，它可以从主链获取目标链最新的若干个世界快照的 hash 值；当矿工节点从目标侧链的其他节点同步世界状态文件的时候，他就知道他接收并恢复的世界状态是否是安全有效的世界状态.

4. 同步世界状态文件的效率要远远高于同步所有历史块并回放的效率，从而减少了矿工节点因为链间切换而造成的无法出块的经济损失.

思考：调度的频率问题？直觉上，我们会觉得调度速度越快越好，比方说这样会让攻击者没有足够的时间来组织发起攻击；但是有两方面的现实考虑 1）根据上面的表格，在一定的参数设置下，e.g. 40 个委员会成员，恶意节点 40% 的全网占比，攻击者有 0.00004 的概率占到单链 2/3 以上的委员会成员，这个概率其实并没有小到完全可以忽略，也就是说更快的调度是让这个攻击条件在更短的时间内得到满足； 2）如上所述，调度对矿工来说是有切换成本的，过快的调度频率如果让挖矿时间相对切换同步时间不占优的话，那从经济角度来说是不合理的. 所以调度的频率并不是完全的理论推算，而且也要和具体的网络配置情况和调度同步的实现效率相结合起来考虑. 以太坊有一个有趣的 stateless client 想法，可以实现节点在链间”瞬时”切换而不需要同步数据，从而可以让链间的调度频率非常的高. 我们认为这个想法在实际的可行性上还存有很大的不确定性，但后续会持续关注他的实现情况.

主侧链通信和轻客户端

有了多链之后，必然要考虑链间的通信 (Inter-Blockchain Communication) 机制. 从系统设计的角度来看，我们需要考虑什么是需要被跨链通信的？为了达到这些通信，哪些原生的基础设施是需要系统提供的？哪些是可以在 Layer 2 层面由库或者 DApp 来提供的. 在我们第一版上线的多链实现里面，我们的设计是：

1. 提供原生的主链与侧链互相为轻客户端的支持，也就是主链和侧链互相保存了对方的块头信息，能够互相验证对方链的有效性以及某个交易确实在对方的链上发生且确认.

2. 提供基于委员会投票的通用跨链通信机制，e.g. 当侧链 A 需要同步主链的某个信息的时候，可以由侧链 A 的委员会成员在侧链 A 的系统多签合约发起对主链某个信息的投票，当大于 2/3 的委员会投票达成的时候，即可认为主链的某个信息在 A 链的到了确认，并触发后续的操作. 可以看到，这个是一种将通用的预言机（Oracle，将链下数据上链的机制）运用在跨链通信的使用实例.

3. 在具体的主侧链通信内容上，系统原生实现了 1）主链的块头同步到所有侧链上； 2）每个侧链的块头同步到主链上； 3）矿工在侧链间的调度信息从主链同步到侧链；4）主链的账号同步到侧链；5）DApp 开发者购买的资源套餐从主链同步到侧链；6）UGAS 可以在主侧链同一账户之间流转.

思考一：为什么没有实现跨链合约调用？实现了数据隔离的多链/分片机制天生就和跨链的同步合约调用是互斥的，因为本链既没有被调用合约的数据，也没有被调用合约的代码（执行环境）；所有如果一定需要另外一条链上的合约提供某种计算服务，只能通过 async 异步的方式把请求的服务委托到目标合约/链上，然后通过某种异步等待的方式拿到返回结果；如果恰好底层的共识协议不具有确定最终性（deterministic finality），这种搭配简直就是灾难. 所以我们认为原生的跨链合约调用在现阶段实现太过复杂且并没有倡导一种直观的编程范式. 我们认为同样的功能都能够在 Layer 2 由库提供者，跨链中继（relayer）服务提供者和 DApp 开发者来协同来实现.

接下来我们讨论 PoS 共识下的轻客户端设计的一些挑战. 首先要明确，我们的目的是 A 链和 B 链的跨链通信，而其中的基础是 A 链要能确认 B 链告诉自己的信息是真实可靠的（vice versa），那在交易(transaction)驱动的区块里世界里，就意味着 A 链要能确认某个交易确实在 B 链上执行且进入到了 B 链的不可逆块里面. 那比较直观的方法就是让 A 链也成为 B 链的节点，毕竟如果 A 链有图灵完备的虚拟机，那在 A 链上用智能合约构建一个 B 链的全节点在理论上也是完全没问题的. 但实际上 A 链并不需要参与 B 链的共识过程，而只是需要验证 B 链上的交易，那传统的做法是 A 链上只维护 B 链的精简的块信息（通常是块头信息），而不需要同步 B 链的完整块信息或者世界状态，这样子我们称 A 链为 B 链的轻客户端 (light client). 那么问题是，A 链到底需要维护 B 链的什么信息呢？在传统的基于 PoW 共识（比特币，以太坊）的情况下，轻客户端只需要维护目标链的块头信息就可以 1）验证区块链的有效性 2）验证某个交易是否进入到了某个区块（通过交易的 merkle path 加块头信息里的 merkle root）.但是在 PoS 的共识下，轻客户端需要维护怎样的块头数据才能够验证目标区块链的有效性呢？

（以下讨论假设读者已经熟悉了我们的 R-PoS 共识设计）在我们的设计中，块头信息除了常规的 proposer, transaction merkle, previous hash 等等之外，还有两个 optional 的数据结构会周期性的出现在块头里

1） Confirm Point 包含了当前委员会成员对上一块的有效性的聚合签名；我们知道，PoW 的块头是“自证清白”的，也就是只要块头里的 nonce 值是对的，那区块的链接就必然是有效的；但是 PoS 是基于委员会成员的投票的，有效的委员会签名代表了有效的投票，那一种思路是把每轮共识过程中委员会的投票信息都放到块头里面去，但是这样就需要消耗大量的存储空间（另外一个难点是要对哪些投票信息记录到块头里进行共识，所以只能是下一块记录上一块的投票信息）；我们采用的方法是周期性的选取一个块把对这个块的签名投票信息放到块头里；这样对于只维护块头的轻客户端来说，只有当遇到包含 Confirm Point 的块头的时候，才能够把这之前的块头都一起确认了；可见的坏处是跨链的块头/交易确认的延迟受限于 Confirm Point 的出现频率，而提高这个频率会有额外的存储和计算的开销，这个也算是效率与存储/计算资源之间的一种 tradeoff.

2） Epoch Point 包含了当前所有委员会成员的全列表；Epoch Point 只有在委员会成员发生变化的时候才会出现在块头里；上面提到了 Confirm Point 是当前委员会成员的签名投票的聚合，但是轻客户端即然不维护目标链的世界状态，它如何知道当前有效的委员会成员是哪些呢？那他就需要从最新的 Epoch Point 里面来拿到全列表了.

综上所属，可以看到 Confirm point & Epoch point 的设计理念就是把 PoS 的核心固化在块头信息里：谁 stake 了成为了委员会成员以及委员会的变化历史，哪些委员会成员签名投票承认了块的有效性；只有当轻客户端对接收到的块头信息的有效性有足够的信心的时候，跨链通信的有效性确认才成为了可能.