5.1 痛点分析

目前区块链技术发展飞快并日趋成熟，但仍有不少企业对应用区块链还有些顾虑，主要因为传统区块链技术要落地到商业应用特别是金融应用，仍有比较多的问题，其中最大的一个问题是交易性能。对于商业应用来看，交易吞吐量和时延是企业最关心的交易性能指标。诚然，类似比特币区块链交易频率约6.67次/秒,每次交易需要6个区块确认,10分钟才能产生一个区块,全网确认一次交易需要1个小时，明显，这样的交易性能无法满足金融机构所涉及的高频交易。

5.2 研究现状

为了提升性能，工业界提出了一些如闪电网络、分片处理等创新的设计，下面简述现阶段具有代表性的方案。

1. Ethereum

可扩展性是以太坊网络承接更多业务量的最大限制。以太坊项目未来希望通过分片（sharding）机制来提高整个网络的扩展性。分片是一组维护和执行同一批智能合约的节点组成的子网络，是整个网络的子集。

支持分片功能前，以太坊整个网络中的每个节点都需要处理所有的智能合约，这就造成 了网络的最大处理能力受制于单个节点的处理能力。

分片后，同一片内的智能合约处理是同步的，彼此达成共识，不同分片之间则可以是异步的，这样就可以提高网络的可扩展性。

1. Bitcoin

为了提升性能，Bitcoin社区提出了闪电网络的创新设计。闪电网络的主要思路十分简单——将大量交易放到比特币区块链之外进行，只把关键环节放到链上进行确认。

闪电网络主要通过引入智能合约的思想来完善链下的交易渠道。核心的概念主要两个：RSMC（recoverable sequence maturity contract），即“可撤销的顺序成熟度合同”，其原理类似于资金池机制。首先嘉定交易双方之间存在一个“微支付通道”（资金池）。交易双方先预存一部分资金到“微支付通道”里，处处是情况下双方的分配方案等于预存的金额。每次发生交易，需要对交易后产生资金的分配结果共同进行确认，同时签字把旧版本的分配方案作废。任何一方需要提现时，可以将他受众双方签署过的交易结果写到区块链网络中，从而被确认。另一个概念是HTLC（hashed time lock contract），这其实就是限时转账。通过只能合约，双方约定转账方先冻结一笔钱，并提供一个哈希值，如果在一定时间内有人能提出一个字符串，使得它哈希后的值与已知值匹配，则这笔钱转给接收方。

RSMC保障了两个人之间的直接交易可以在链下完成，HTLC保障了任意两个人之间的转账都可以通过一条“支付”通道来完成。闪电网络整合这两种截止，实现任意两个人之间的交易都是在链下完成。

1. Hyperledger

Hyperledger Fabric针对之前Peer节点承担了太多的功能，从而带来了扩展性差、交易性能低的问题，针对上述问题，做出很大的改进和重构：

* 解耦了原子排序环节与其他复杂处理环节，消除了网络处理瓶颈，提高可扩展性。
* 解耦交易处理节点的逻辑角色为背书节点、确认节点、可根据负载进行灵活部署。

上述涉及的两方面，一方面解决了全网交易一致性的问题，另一方面解决了数据处理的性能问题。

存在的问题：

为了提高可扩展性和交易性能，都会采用“众人拾材火焰高”的多任务模式，减少主链的压力，其中就不可避免的涉及到如下两个问题：

1. 通信手段

不同链之间的通信手段问题，现阶段有两种模式，一种是使用区块链自身作为消息传递的媒介，另一种不依赖区块链，通过区块链间的“连接者”传递通信消息。但这两种模式各有优势和不足：

用区块链传递消息，则可以不必基于信任来通信，但是由于共识机制的存在，这样的通信速度一般会比较慢。然而不依赖区块链，虽然速度上会有很大的提高，但是这是一种能而是基于信任的转移，在信任不高的环境下，这种机制的安全性会有很大的挑战。

1. 并行计算

并行计算的使用，是提高区块链系统计算速度和处理能力的一种有效手段。但同时也会造成一些问题，以Hyperledger为例，由于其对事务的处理都是基于状态的，并行处理与状态机在本质上是相违背的。并行处理会造成状态的混乱，无法达到一致。

5.3 解决方案

本文博众家之所长，同时结合政府主导下的联盟链场景的特点，提出了多链并行计算的创新架构。

5.3.1 方案架构

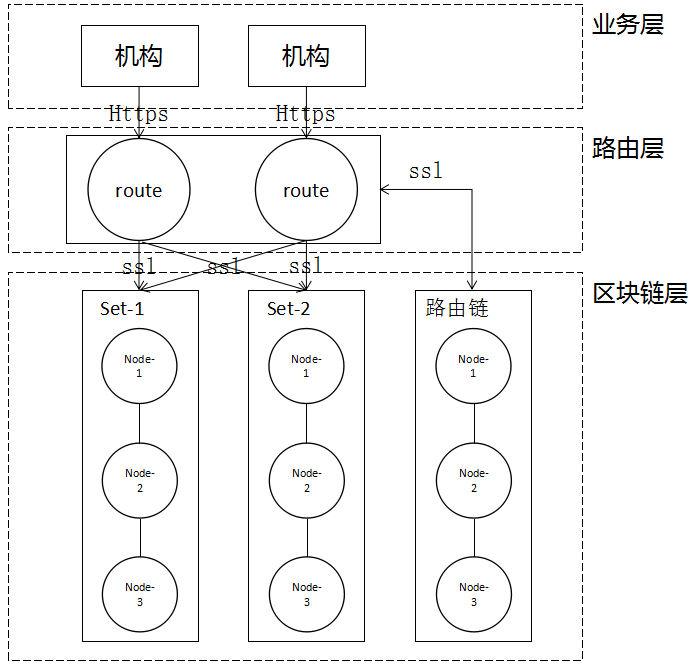
在该场景下，运行环境安全可信，因此在区块链层与业务层之间增加中继路由模块，负责接收业务层的请求向多条区块链进行消息转发，多条链协同处理，并行计算，此模块相对独立于区块链系统，因此该模块的性能不会受制于区块链本身的性能瓶颈，因此大大提升了整个系统的性能；同时中继路由模块可以根据所链接的每个区块链处理载荷的实时情况，对请求进行负载均衡，大大提升了系统的鲁棒性。

为了保障系统的安全，业务层与路由层的通信采用Https通信协议，路由层与区块链层的通信采用ssl通信协议。

中继路由模块又非完全独立于区块链系统，中继路由模块的路由规则来源于区块链，在多条区块链中，独立设置一条路由链，路由链上运行着与路由规则相关的智能合约，中继理由模块通过安全的ssl通信协议与路由链通信，获取路由规则，对业务层发送过来的请求进行相应路由。所有规则来源于区块链，又服务于区块链，保障了整个系统任然是以一种分布式的方式运行，保障了系统的安全和规则的透明。

同时，中继路由模块的设计，也提高了系统的扩展性，实现了可插拔、模块化。只需要路由链制定不同的规则智能合约，中继路由模块即可实现不同的功能。

具体的方案架构如下图所示：



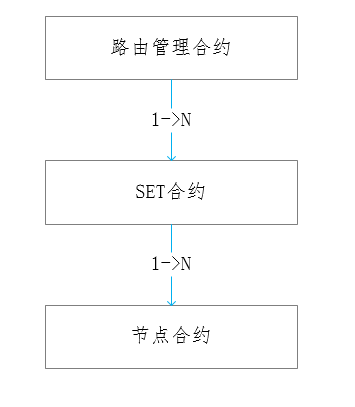
5.3.2 关键模块

1. 路由链
   1. 模型

路由链为一条完整独立的区块链。用于制定多链并行计算架构中中继路由模块的路由规则，在该条区块链上部署了路由规则相关的路由管理合约，和全局区块链系统相关的Set合约和Node合约。

除了路由链之外的区块链称之为业务链，路由链与业务链的区别在于链上智能合约的不同。业务链部署与业务先关的智能合约合约，而Route链只部署路由管理合约、Set合约和Node合约。

* 1. 智能合约模型



三种合约之间是有相互依赖关系的，如上图所示，自上而下属于1对N的关系，层层嵌套，路由管理合约只有一个，Set合约根据有几条业务链对应几个Set合约，Node合约根据每条链上有几个节点对应几个Node合约。

* 1. 路由管理合约（RouteManager）

|  |  |
| --- | --- |
| **存储数据** | **说明** |
| Array<SET合约地址> | 数组结构，存储所有Set合约的地址 |
| Map<用户ID,SET序号> | Map结构，Set合约和用户ID的映射，即该ID的用户的数据请求应该由哪个Set处理 |
| Uint 业务ID |  |
| Uint 当前空闲SET序号 | 递增 |

|  |  |  |  |
| --- | --- | --- | --- |
| **接口** | **输入** | **输出** | **说明** |
| registerSet | SET合约地址 |  | 注册新SET，即将新的SET合约的地址加入到数组中 |
| registerRoute | 用户ID | Boole | 注册用户路由，即分配新用户的数据请求交由哪个Set处理 |
| getRoute | 用户ID | SET序号 | 获取当前用户是由那个Set处理 |

* 1. Set合约

|  |  |
| --- | --- |
| **存储数据** | **说明** |
| Array<用户ID> | 数组结构，存储当前Set需要处理的user列表 |
| Array<节点合约地址> | 数组结构，存储属于该Set的所有Node合约的地址 |
| Uint SET序号 |  |
| Uint maxNum | 最大用户数 |
| Uint warnNum | 告警用户数 |

|  |  |  |  |
| --- | --- | --- | --- |
| **接口** | **输入** | **输出** | **说明** |
| registerRoute | 用户ID |  | 注册用户路由 |
| getNodeList |  | 节点信息列表 | 获取SET的节点列表 |
| addNode | 节点信息 |  | 增加节点 |
| removeNode | 节点信息 |  | 删除节点 |
| IsFull |  | 布尔 | 返回当前SET是否已满  不允许继续注册用户 |

* 1. Node合约

|  |  |
| --- | --- |
| **存储数据** | **说明** |
| String NodeId |  |
| String ListenIp |  |
| Uint P2PPort |  |
| Uint RpcPort |  |
| Uint NodeType |  |
| String Desc |  |
| String CAHash |  |
| String agencyinfo |  |

1. 中继路由模块
   1. 模型

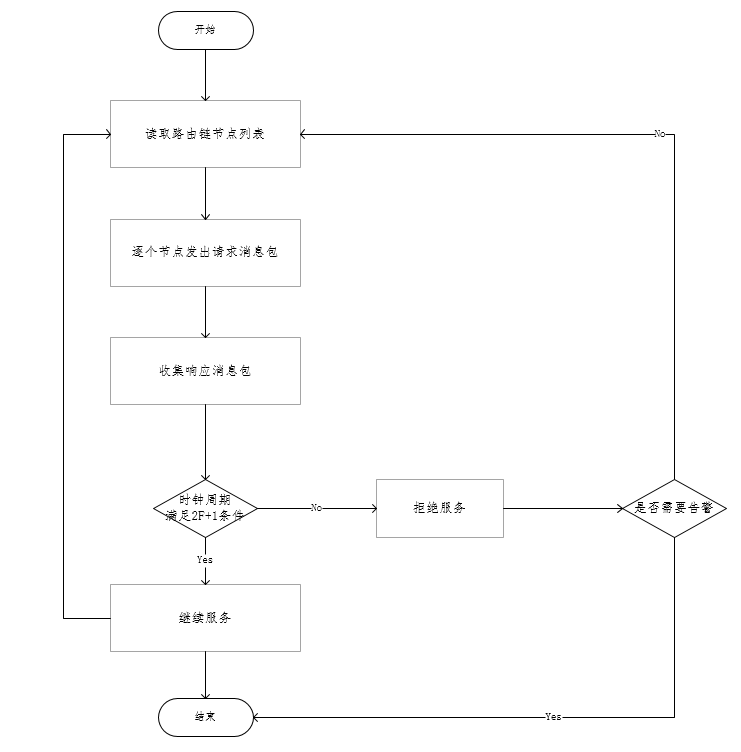
使用NodeJs实现高并发接入，实现内存缓存路由信息提高路由速度。

* 1. 功能列表

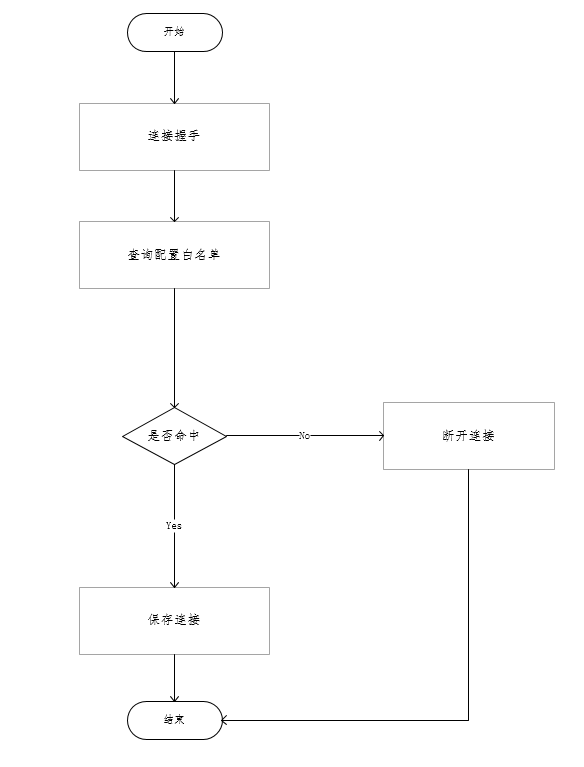
|  |  |
| --- | --- |
| **功能项** | **说明** |
| 接入准入 | 控制请求的接入授权管理。 |
| 路由链请求转发 | 将路由链的请求转发至路由链 |
| 业务链请求转发 | 判定业务路由规则并对请求进行业务链内的分发 |
| 回调模块 | 添加支持对交易成功后进行回调 |

5.3.2 关键流程

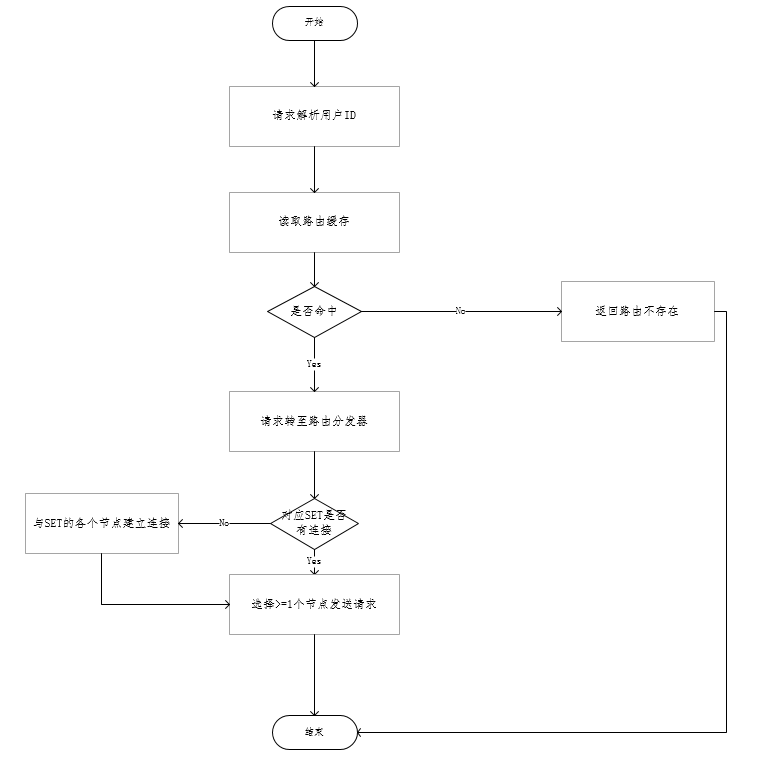
1. 接入代理数据校验



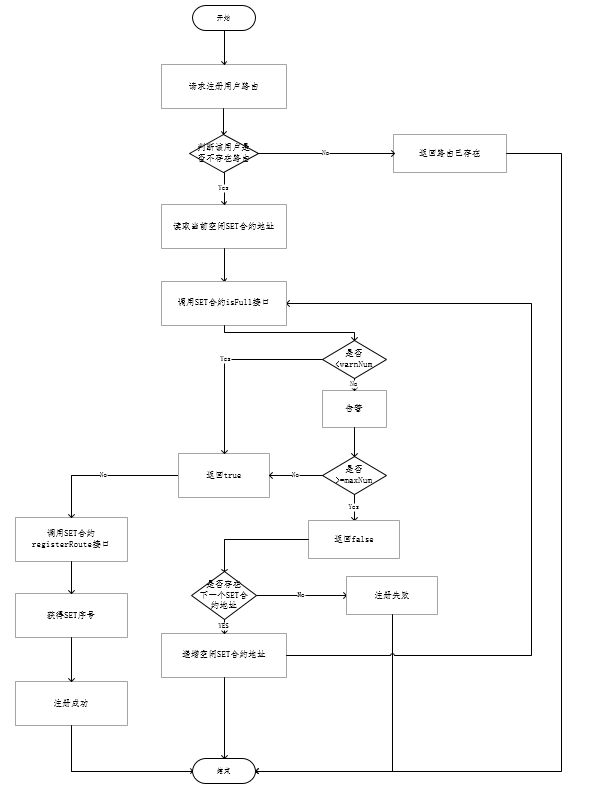
1. 接入准入



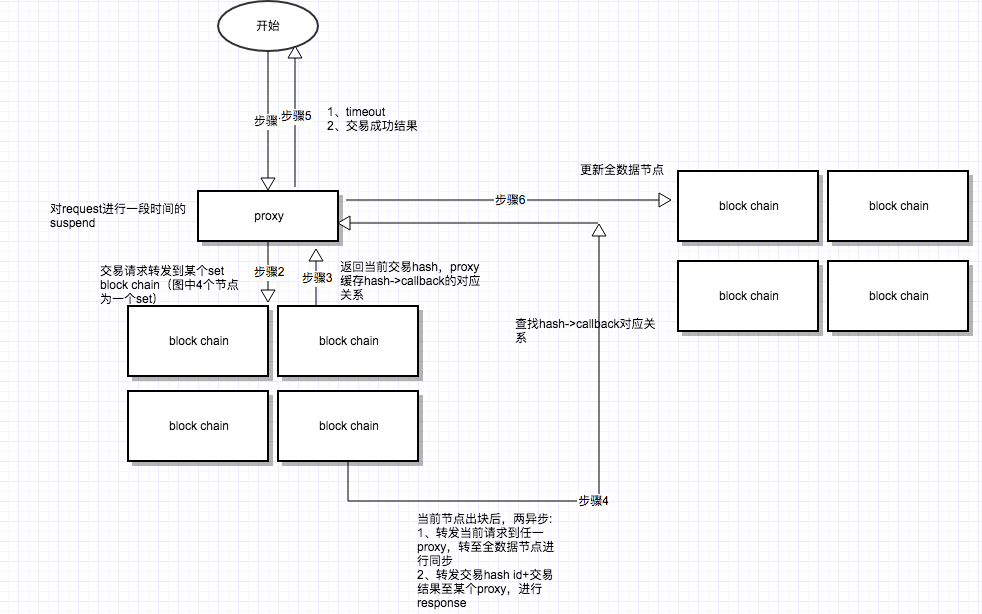
1. 请求转发



1. 路由分配策略



1. 回调策略

****

5.3.3 性能测试