软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(11):5042−5057 [doi: 10.13328/j.cnki.jos.006741]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

BETASCO: 面向智能合约分片的联盟区块链系统*

吴恺东[1,2], 马 郓[3], 蔡华谦[1], 景 翔[2], 黄 罡[1,2,3]

[1](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

[2](北京大学 软件与微电子学院, 北京 100871)

[3](北京大学 人工智能研究院, 北京 100871)

通信作者: 马郓, E-mail: mayun@pku.edu.cn; 黄罡, E-mail: hg@pku.edu.cn

摘 要: 基于区块链的去中心化应用已在加密数字货币、云存储、物联网等多个领域提供健壮、可信且持久的服务, 然而区块链的吞吐能力难以满足去中心化应用日益增长的性能需求. 分片是当前主流的区块链性能优化技术, 但现有的区块链分片主要面向用户和用户之间的转账交易, 并不完全适用于以智能合约调用交易为主的去中心化应用. 针对此问题, 设计并实现面向智能合约分片的联盟区块链系统 BETASCO. BETASCO 为每个智能合约提供一个分片作为独立执行环境, 通过基于分布式散列表的合约定位服务将交易路由至目标智能合约所在的分片, 并通过智能合约间的异步调用机制满足跨智能合约的通信和协作需求. BETASCO 通过节点虚拟化允许一个节点加入多个分片, 支持同一组节点上多个智能合约的并行执行. 实验结果表明, BETASCO 整体吞吐能力可随智能合约数量的增加而线性增长, 且执行单个智能合约的吞吐能力与 HyperLedger Fabric 相当.

关键词: 区块链; 智能合约; 去中心化应用; 区块链分片; 异步调用

中图法分类号: TP311

中文引用格式: 吴恺东, 马郓, 蔡华谦, 景翔, 黄罡. BETASCO: 面向智能合约分片的联盟区块链系统. 软件学报, 2023, 34(11): 5042–5057. http://www.jos.org.cn/1000-9825/6741.htm

英文引用格式: Wu KD, Ma Y, Cai HQ, Jing X, Huang G. BETASCO: Consortium Blockchain System Based on Smart Contract-oriented Sharding. Ruan Jian Xue Bao/Journal of Software, 2023, 34(11): 5042–5057 (in Chinese). http://www.jos.org.cn/1000-9825/6741.htm

BETASCO: Consortium Blockchain System Based on Smart Contract-oriented Sharding

WU Kai-Dong[1,2], MA Yun[3], CAI Hua-Qian[1], JING Xiang[2], HUANG Gang[1,2,3]

[1](Key Lab of High Confidence Software Technologies, Ministry of Education (Peking University), Beijing 100871, China)

[2](School of Software & Microelectronics, Peking University, Beijing 100871, China)

[3](Insititute for Artificial Intelligence, Peking University, Beijing 100871, China)

Abstract: Blockchain-based decentralized applications have been providing robust, reliable, and durable services in multiple fields, such as encrypted digital currency, cloud storage, and Internet of Things. However, the throughput capacity of blockchain can no longer meet the increasing performance requirements of decentralized applications. Sharding is the current mainstream performance optimization technology for blockchain. Nevertheless, the existing blockchain sharding approaches, mainly focusing on transactions among users, are not always applicable to decentralized applications dominated by the transactions of smart-contract invocation. To solve the above problem, this study designs and implements the consortium blockchain system BETASCO based on smart contract-oriented sharding. BETASCO provides a shard that serves as an independent execution environment for each smart contract, routes transactions to the shards holding the target smart contracts by a contract location service based on the distributed hash table (DHT), and supports communication and collaboration needs across smart contracts by availing the asynchronous invocation mechanism among smart contracts. By virtualizing the nodes,

* 基金项目: 国家自然科学基金杰出青年基金 (61725201); 北京高校卓越青年科学家计划 (BJJWZYJH01201910001004)
收稿时间: 2021-10-21; 修改时间: 2022-02-16, 2022-04-23; 采用时间: 2022-07-05; jos 在线出版时间: 2023-06-16
CNKI 网络首发时间: 2023-06-19

BETASCO allows each node to join multiple shards to support parallel executions of multiple smart contracts on the same set of nodes. The results of experiments show that the overall throughput capacity of BETASCO linearly increases as the number of smart contracts grows, and the throughput capacity for the execution of a single smart contract is comparable to that of HyperLedger Fabric.

Key words: blochchain; smart contract; decentralized application; blockchain sharding; asynchronized invocation

基于区块链的去中心化应用 (简称去中心化应用) 利用区块链和智能合约技术所提供的可信存储与计算能力, 在无信任中心的条件下提供数字资产、汇款和在线支付等金融服务[1]. 随着联盟链技术的发展, 去中心化应用也进入了数据治理[2]、云存储[3]、物联网[4]、供应链[5]等领域, 为利益相关者建立可信环境, 以实现健壮、可信且持久的服务.

在蓬勃发展的同时, 区块链却常常遇到可扩展性问题, 其有限的交易吞吐量限制了所承载的去中心化应用的性能[6]. 分片是一种有效解决区块链可扩展性问题的技术, 通过将网络划分为不同运行单独共识协议的子集以处理不同的交易集合, 达到交易吞吐量随分片数量线性增长的效果[7].

然而, 现有的区块链分片技术并不完全适用于以智能合约调用交易为主的去中心化应用. 一方面, 大多数分片技术只关注用户到用户的转账交易, 它们发生在大量的分散账户之间; 而去中心化应用的智能合约调用交易集中于少数账户上, 更容易出现流量失衡导致的拥塞[8], 影响冷门智能合约的运行. 另一方面, 分片技术中在处理跨分片交易时需要通过交易的输入和输出判断参与跨分片交易的分片, 转账交易的输入和输出在发起时就明确指定, 但智能合约调用交易的输入和输出则只能在运行时计算得出, 使得跨分片交易的处理更为复杂. 这些问题使得在当前的分片技术中, 智能合约的性能依旧会因为流量失衡导致的拥塞而下降, 跨分片交易的处理更是会同时影响到多个分片上智能合约的执行, 使得去中心化应用无法从当前分片技术带来的性能提升中获益.

针对上述问题, 本文的思路是将一个智能合约的调用交易划分到同一个分片上, 从而使每个智能合约得到独立的运行环境, 以提高智能合约的运行效率, 满足去中心化应用的性能需求. 这一思路主要受到以下事实的启发: 首先, 区块链上的交易中智能合约调用交易逐渐成为主流[6], 这一点在以支持跨组织/机构可信协同的联盟区块链中尤甚. 因此面向智能合约分片的区块链更适应其作为去中心化应用平台的场景. 其次, 多数去中心化应用只有少量的智能合约. 例如, 截至 2022 年 2 月 24 日, 主流去中心化应用收集网站 State of the DApps 上登记有 1 192 个 Ethereum 去中心化应用, 其中 71.31% 的应用只有 1 个智能合约, 94.80% 的应用智能合约数量不多于 10 个[9]. 这表明将智能合约部署于单独的分片中已可以支持大多数去中心化应用. 最后, 多数智能合约调用交易也只涉及一个智能合约. 例如, 最大的去中心化应用平台 Ethereum 的交易数据统计显示, 在北京时间 2022 年 2 月 1 日全天提交的 9 705 个区块 (高度 14 118 953–14 128 657) 中, 涉及智能合约间调用的交易仅占所有智能合约调用交易的 10.66%, 其中 76% 的交易只有一次智能合约间调用, 82.26% 的交易调用了不超过 2 个智能合约. 这表明智能合约调用交易中涉及多个智能合约, 且进行多次智能合约间调用的情况是极少数的. 在实现面向智能合约分片之后, 也仅有少部分交易需要进行特殊的跨分片交易处理.

基于这一思路, 本文设计并实现了面向智能合约分片的联盟区块链系统 BETASCO. 首先, BETASCO 为每个智能合约都分配一个单独的分片, 仅处理所有该智能合约的交易, 以隔离其执行环境, 解决智能合约流量失衡导致的拥塞问题. 其次, 为解决分散在网络中的智能合约执行环境的定位问题, BETASCO 借鉴分布式散列表技术, 提供可靠的智能合约目录服务将交易路由给目标智能合约所属的分片. 最后, BETASCO 将交易定序和状态验证进行分离, 以支持模块化的共识协议, 并通过智能合约间的异步调用机制支持它们的通信和协同, 以在保证跨分片调用原子性的同时保持高性能.

本文的主要贡献如下.

(1) 以区块链分片技术为基础, 为每个智能合约提供单独的分片作为独立的运行环境, 以支持面向智能合约的分片, 并通过节点虚拟化解决分片有重叠的智能合约之间的矛盾.

(2) 基于分布式散列表技术, 提出能将智能合约执行环境分散在网络中的智能合约分片选择与定位算法, 并支持在网络发生变化时能将交易稳定地路由给目标智能合约的分片成员节点.

(3) 分离了智能合约执行过程中的定序和执行阶段, 以支持模块化的共识协议, 并通过智能合约间的异步调用

机制支持它们的通信和协同.

(4) 与 HyperLedger Fabric 进行了对比实验, 实验结果表明, 去中心化应用的性能因面向智能合约的分片得到提升, 验证了 BETASCO 系统设计和实现的有效性.

本文第 1 节介绍并总结了相关工作. 第 2 节从 BETASCO 智能合约执行的定位-定序-执行 3 个阶段出发介绍 BETASCO 的概念设计, 以及 BETASCO 的信任和威胁模型. 第 3 节介绍 BETASCO 的系统结构与相关构件的实现. 第 4 节通过实验评估 BETASCO 的效果. 第 5 节总结全文工作并进行展望.

1 相关工作

区块链分片技术已经成为一种有效的区块链可扩展性优化技术. 区块链分片技术受到数据库分片的启发, 将区块链的网络分割成多个互不重叠的子网, 称为分片, 然后将交易也分割为同等数目的子集分别由这些分片运行单独的共识协议执行和维护[7], 如图 1 所示. 这样, 就能达成区块链的交易吞吐量随分片数量增加的线性增长.

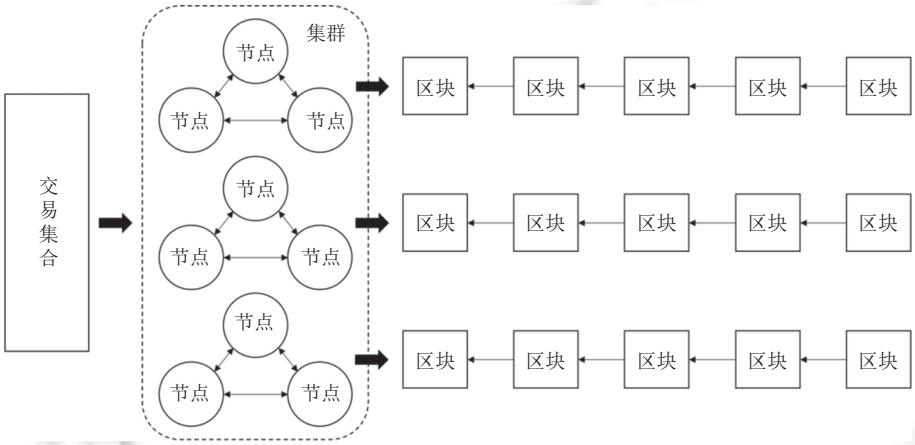

图 1 区块链分片技术

现行的分片式区块链主要面向交易进行分片, 可从并行化资源使用的角度分为两类. 其一是面向计算的分片, 代表为 ELASTICO[10]、OmniLedger[11]和 DP-Hybrid[12]等. 这些区块链中使用两层共识, 顶层周期性地更新分片配置, 底层的分片内共识则负责处理划分后的交易集, 并将其保存在共享的账本中. 其二是面向存储的分片, 如 RapidChain[13]和 Monoxide[14]等. 其中每个分片具有单独的共识和存储, 分片配置相对固定, 也因此需要通过复杂的机制处理跨分片交易. 这些技术大多根据涉及交易的账户来划分交易集合. 但若用于支持智能合约时, 因为智能合约调用交易的流量比转账交易更加集中在少数个智能合约账户上, 则更容易出现流量失衡导致的拥塞.

面向交易分片的区块链用于支持智能合约时的另一个问题在于跨分片交易的处理. 跨分片交易发生时, 首先需要确定的就是参与本次交易的分片. 转账交易通常具有明确的输入和输出, 其涉及的分片在交易发起时即可指定. 然而智能合约的行为受到代码的控制, 其输入和输出只有在运行时才能通过计算得出, 因此需要更复杂的跨分片交易机制来处理. 产业界的一些分片式区块链要么禁止跨分片智能合约相互调用 (如 Harmony[15]), 要么将交易涉及的智能合约移动到同一分片中来处理 (如 Elrond[16]和 Ethereum 2.0 方案[6,17]). 面向存储分片的智能合约平台 Chainspace[18]则改变了智能合约模型, 使用类似于转账交易的数据结构表示智能合约调用交易, 以通过“交易输入先确认, 交易输出后确认”的两阶段提交机制处理跨分片交易. 但这些工作处理跨分片交易的效率都不高, 而且会影响不跨分片的智能合约调用交易的执行过程.

总之, 区块链分片技术通过并行化资源使用, 能够达成区块链交易吞吐量的线性提高. 但多数现存区块链分片技术只关注了用户和用户之间的转账交易. 在支持智能合约时, 因为智能合约调用交易的流量集中, 且跨分片的智能合约调用交易涉及分片只能运行时得出等特性, 智能合约容易因拥塞、跨分片交易处理而性能下降, 并不能因

区块链分片得到提升, 无法满足去中心化应用的需要.

此外, 近年来还有一些工作提出了多链架构[19−22]. 多链架构与分片架构类似, 通过维护一组并行的区块链, 以提高整体的交易吞吐量. 在多链架构中, 每个节点都可以同时参与每条并行链, 更便于跨链交易的处理. 多链架构可视为对区块链系统的纵向扩展, 使整体的交易吞吐量能逐渐逼近节点性能 (主要指网络带宽) 所能支持的上限. 但在联盟链环境中, 交易吞吐量主要就受限于节点的网络和存储资源, 采用多链架构反而会因为加剧资源竞争而导致性能下降, 因此也并不满足联盟链中去中心化应用的需要.

2 基础知识

要实现面向智能合约的分片, 需要将智能合约一一对应地部署在分片中, 为智能合约提供独立的运行环境, 再将智能合约调用交易路由到相应环境中. 本节首先概述了 BETASCO 的基本设计思想, 然后介绍定位-定序-执行 3 阶段的智能合约执行流程.

2.1 系统概述

BETASCO 是一种分片式联盟区块链系统, 将每个智能合约都部署在不同的分片之中, 通过异步调用机制支持智能合约之间的通信与协作. 这样, 每个智能合约都将得到独立运行环境, 执行过程不会受到其他智能合约的影响, 性能能够达到单个分片的性能上限, 且仍然能支撑同一个去中心化应用. BETASCO 面向联盟链场景, 用于维护智能合约、支撑去中心化应用, 没有内置的加密数字货币, 因此并不需要在账本中维护用户账户的信息.

BETASCO 采用了面向存储的分片设计, 部署在不同分片中的智能合约将代码和状态维护在不同的账本中, 因此具有了独立的执行环境. 在此基础上, BETASCO 使用可靠的智能合约定位机制来找到一个特定智能合约的所在分片, 从而保持智能合约对用户和其他智能合约的可见性.

图 2 展示了 BETASCO 中智能合约的执行流程. 智能合约的执行流程从客户端开始. 用户在使用去中心化应用的过程中涉及了对智能合约的调用, 于是发起了一个交易. 该交易记录了本次智能合约调用的必须信息, 如被调用智能合约的 ID、调用智能合约的方法名和参数, 以及用户或客户端的签名等, 而无需事前指定在处理该交易时可能被调用的其他智能合约. 在对等网络中, 该交易会被节点转发, 直到到达部署了智能合约的节点为止, 这一步被称为定位. 交易到达智能合约对应的分片 (简称合约分片) 后即进入定序阶段. BETASCO 采用现存区块链广泛采用的定序-执行架构[23], 分片的成员先通过共识算法对收到的交易进行定序, 在相互连接的区块中持久化. 在最终的执行阶段, 分片成员按相同的顺序执行这些交易, 并返回结果给客户端. 如果执行阶段遇到了对其他智能合约的调用, 则再发起一次由该智能合约向被调用智能合约发起的交易. 智能合约间调用的执行也会经过定位-定序-执行 3 个阶段, 并且持久化存储在被调用智能合约分片的账本中.

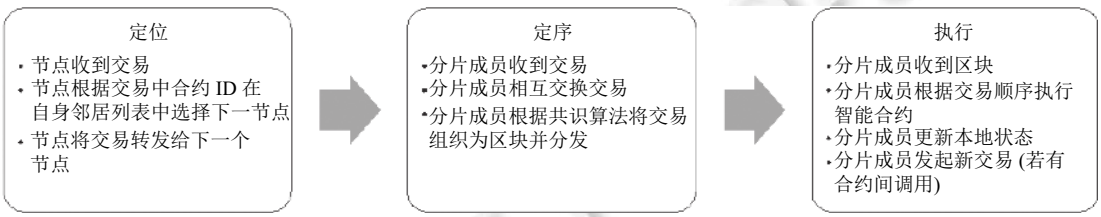

图 2 BETASCO 的智能合约执行流程

BETASCO 区块链构建于一组节点形成的对等网络中, 每个节点都属于某个组织或者机构, 通过证书来验证彼此的身份. 除了组织或机构用于发放证书的网络管理与成员管理服务 (不属于 BETASCO 网络) 外, 网络中没有任何中心化的管理结构, 所有节点平等地承担相同的职能. 但是在某个交易的执行流程中, 节点可能扮演以下角色 (如图 3 所示).

• 客户端: 客户端是智能合约执行流程的起点, 用户或者智能合约发起交易后首先经过客户端的转发到达下一个节点. 因此, 客户端在网络拓扑结构中的位置直接决定了本次交易的执行响应延迟. 若本次交易是智能合约间调

用, 则上一个交易执行流程中的分片成员节点即是本次交易中的客户端.

• 转发节点: 转发节点负责交易的转发, 根据交易中的智能合约 ID 与自身的邻居节点列表, 将交易转发给定位服务中更接近合约分片成员的节点. 转发节点的交易转发策略是定位阶段的核心, 也是智能合约异步执行的重要支撑. 当交易由用户发起时, 转发路径上的第 1 个节点, 即从客户端收到交易的转发节点, 会与所属组织/机构的成员管理服务进行交互, 以判断该交易是否来自一个合法的组织/机构成员用户. 如果不是, 则该节点会拒绝本次交易.

• 分片成员节点: 分片成员节点是共同构成合约分片的节点. 每个分片成员节点都拥有一份智能合约的代码和状态副本, 通过对等连接传递交易与区块, 并对智能合约进行执行. 用于分片的共识采用了模块化设计, 只以交易为输入、以区块为输出, 而状态的维护与验证与之独立, 简化了共识协议的替换.

BETASCO 中的账本由各个合约分片独立维持, 合约分片成员节点只需要本智能合约相关的账本和状态数据就可以参与智能合约执行流程, 因此每个节点都无需持有所有智能合约的账本和状态. 这降低了每个节点的存储压力, 并且因为交易顺序和状态的持久化的分离, 可以通过创建状态检查点以进一步压缩存储空间.

接下来的 3 节中, 本文将解释 BETASCO 中的交易执行流程 (如图 4 所示) 并说明定位、定序和执行阶段的步骤. 最后, 本文总结了 BETASCO 的信任和故障模型.

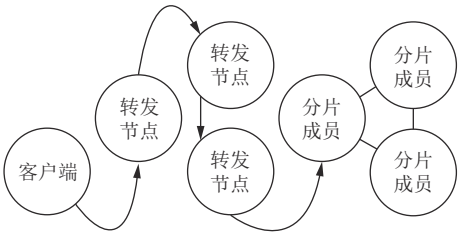

图 3 一次智能合约执行流程中节点的 3 种角色

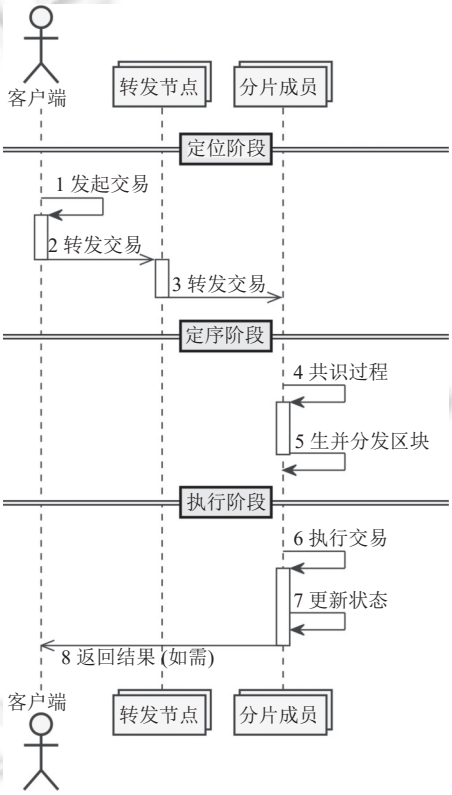

图 4 BETASCO 智能合约执行流程顺序图

2.2 定位阶段

交易在客户端上被发起后, 智能合约执行流程进入定位阶段. 定位阶段的目标是将交易逐级转发给部署了智能合约的分片成员节点. BETASCO 是一个联盟链, 不能使用中心化的智能合约注册表, 因此需要一种面向去中心无结构对等网络的分布式目录服务. BETASCO 借鉴了分布式散列表技术[24−26]来实现这一点.

在分布式散列表技术中, 节点是否为资源提供服务是由二者在同一地址空间中的距离决定的. 简单来说, 分布式散列表技术使用同一个散列函数处理节点 ID 和资源 ID, 将其映射到同一个地址空间中, 然后使节点为那些 ID

散列值与自身 ID 散列值相同的资源提供目录服务. 这样, 资源就可以通过分布式方法确定地分配给节点.

参考这一思想, BETASCO 将节点 ID 和智能合约 ID 使用相同的散列函数映射到同一个地址空间中, 然后选择节点 ID 散列值与智能合约 ID 散列值相近的节点作为智能合约的部署节点, 为其建立对等连接, 形成合约分片. 图 5 展示了智能合约 ID 与合约分片成员节点对应关系的一个例子. BETASCO 构建于 5 个节点构成的对等网络之上, 节点 ID 分别为 *A* 到 *E*. 网络中部署有一个智能合约, ID 为 *X*, 合约分片规模为 3. 将节点 ID 和智能合约 ID 分别由一个值域为 0–15 的散列函数处理. *X* 的散列值为 7, 因此在散列值集合中寻找散列值接近 7 的节点 ID, 发现节点 *B*、*C*、*D* 的 ID 散列值分别为 8、3、5, 是 ID 散列值最接近 7 的 3 个节点, 因此 *B*、*C*、*D* 就是合约分片的 3 个成员节点.

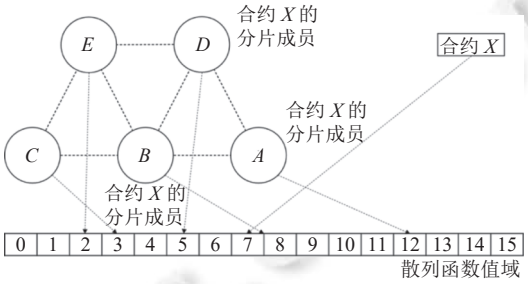

图 5 智能合约分片成员节点与智能合约 ID 的对应

按照上述对应方法, 合约分片成员的选择仅与智能合约 ID、节点 ID、分片的规模以及散列函数相关. 当网络中的节点发生变化时, 某个智能合约对应的分片成员也可能随之变化. 但因为一个智能合约对应多个分片成员, 所以变化前后的分片成员节点仍然有重合部分. 这能够在联盟链网络中保证智能合约服务的持久与稳定.

根据以上智能合约 ID 与分片成员的对应规则, 本文设计了 BETASCO 中节点对交易的转发算法, 如算法 1.

算法 1. 定位阶段交易转发算法.

输入: 智能合约 ID *X*, 邻居节点 ID 列表 *L*, 节点 ID *N*, 散列函数 $hash$;
输出: 目标节点 *D*.

1. $L \leftarrow L \cup \{N\}$
2. $L_1 \leftarrow \varnothing$
3. **for each** $n \in L$ **do**
4. $\quad L_1 \leftarrow L_1 \cup \{hash(n)\}$
5. **end**
6. $D \leftarrow findClose(hash(X), L_1)$ // 找到同一地址空间中最接近的节点 ID

每个转发节点都会将交易转发给散列值最接近的节点, 这样最终会到达一个所找到的节点 ID 与智能合约 ID 散列值最接近的节点就是自身的节点, 该节点即可能为分片成员节点. 如果该节点不是分片成员节点, 则按照上述的对应关系继续查找那些 ID 散列值次接近的节点. 如果还未找到, 则系统会先为终端用户响应一条还未找到的返回信息, 然后通过洪泛方式继续尝试转发交易.

在部署流程中, 交易最终到达的节点会成为首个分片成员, 然后该成员会将交易再次转发, 目标为节点 ID 与智能合约 ID 散列值次接近的节点, 并附上自身的连接信息. 以此类推, 最终部署智能合约的交易会到达一系列 ID 散列值与智能合约 ID 相近的节点, 且这些节点有足够的信息建立彼此之间的对等连接. 在执行流程中, 若交易最终到达的节点并非分片成员节点 (当网络发生变化时), 该节点也可以通过这种再次转发机制找到一个原成员节点, 然后获得足够的信息继续进行执行流程.

2.3 定序阶段

定序阶段是智能合约执行流程中正式执行的前置, 主要通过分片成员间的共识过程处理. 区块链的智能合约依赖于共识机制形成的共识存储, 正是共识机制将收到的交易进行排序保存, 区块链的各节点才能通过按相同顺序执行智能合约来维持智能合约全网一致的状态. BETASCO 部分遵循了这一原理. 因为 BETASCO 的智能合约部署在规模较小的分片中, BETASCO 可以使用轻量级小规模的共识来对交易进行排序与持久化, 以达成交易的快速处理与响应.

借鉴交易并行执行方法, BETASCO 采用模块化的共识协议, 并且分离交易定序与状态验证. 合约分片的共识协议由开发者在部署智能合约时指定, 可以选择 PBFT、Raft、Kafka 等, 并且可以随时切换. 智能合约分片的共识协议以每个分片成员收到的交易为输入, 一个有序的交易序列为输出, 并形成相互链接的区块形式的共识存储, 如图 6 所示. 区块结构内不包含智能合约的状态信息, 因此共识协议对区块的处理只需要验证顺序而不需要执行交易.

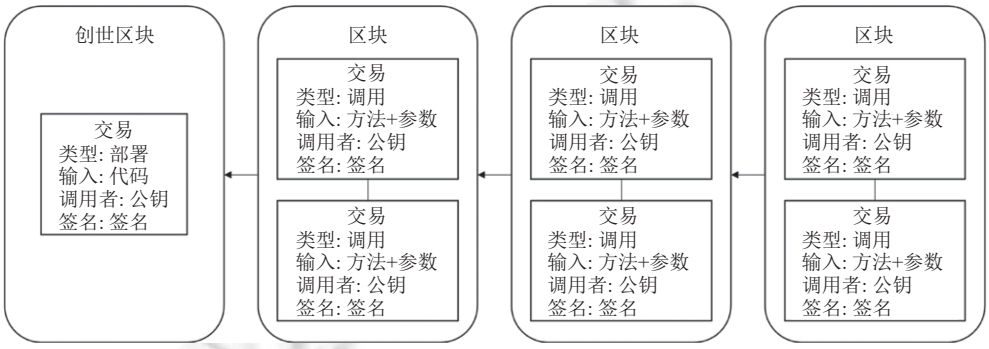

图 6 共识协议的输出: 相互链接的区块

任何发给某个智能合约的交易都会被定序存储, BETASCO 不会对这些交易进行过滤. 但是, BETASCO 提供了一些 API 供开发者获取交易发起者的相关信息, 以在执行阶段拒绝不符合要求的交易的执行.

2.4 执行阶段

定序结束后, 每个分片成员都会收到一些区块, 其中交易以相同的顺序排列. 每个分片成员都会按顺序执行这些交易, 然后更新所持有的智能合约状态的副本.

在交易的执行阶段, 分片成员首先从本地获取智能合约的代码和状态, 然后加载它们. 在部署智能合约时, 代码就以交易内容的形式保存在创世区块中. 智能合约的状态则维护在本地的数据库中. 分片成员将代码和状态加载到内存中, 然后按照交易内容中的方法名和参数执行智能合约, 更新状态, 得出结果. 执行结束后, 节点会将内存中的状态写回. 为和交易序列的执行对应, 对智能合约状态的读写不允许多线程. 在状态写回后, 分片成员再将本次调用的结果返回.

如果在智能合约调用交易的执行过程中调用了别的智能合约, 此时分片成员会创建新的交易以进行智能合约间的异步调用. 智能合约分片成员创建的交易与客户端发起的交易类似, 但是需要使用智能合约自身的密钥进行签名. 同样的, 智能合约发起的交易也无需标注本次交易可能被调用的其他智能合约.

BETASCO 将在两个阶段保证智能合约间异步调用的原子性: 只有在智能合约调用交易成功执行 (没有触发任意程序异常) 时, 智能合约间异步调用才会被发起并且执行. 一次智能合约调用交易的执行过程中可能会触发多次智能合约间调用, 分片成员会在本次执行过程成功结束后统一发起它们, 以保证异步调用第 1 阶段的原子性. 被发起的交易会经由智能合约定位服务被转发到目标智能合约的分片成员, 进入后续的定序与执行过程. 智能合约分片在对收到的交易定序时会优先对智能合约发起的交易排序并打包. 如果执行失败, BETASCO 会创建一笔新交易发往触发异步调用的智能合约, 以调用其异常处理方法. 每个智能合约都具有一个空的默认异常处理方法, 开

发者可以通过重载该方法来根据实际业务逻辑的需要调整合约状态. 这样就保证了异步调用第 2 阶段的原子性.

通过控制智能合约间异步调用交易的发起和执行, BETASCO 保证了跨分片智能合约调用交易的原子性. 任何智能合约间异步调用交易都在交易执行过程中被创建并发起, 也就无需在发起智能合约调用交易时标注所有可能涉及的智能合约. BETASCO 对跨分片交易提供的原子性保证能够满足联盟链的需要, 主要有以下两点原因: 首先, 联盟链中的智能合约被用于表达业务逻辑, 其功能一般是高度内聚的, 合约间松耦合, 合约间调用较少; 其次, 从软件开发的角度考虑, 组织/机构间出于可信协同目的开发的合约具有相对更高的完备性, 一般情况下合约调用是可以正确执行的. 对于少数的异常情况, 也可以通过 BETASCO 提供的异常处理方法进行处理, 这样可以避免系统层面的合约状态回滚, 最大化地提升性能. 另外, 在一些场景中跨分片调用的原子性也并不是必须的, 例如医疗机构和保险公司合作的场景中, 电子病历在记录之后通知保险业务的情况, 此时异步调用仅用于消息的通知, 触发异步调用的合约已经提供了主要功能, 无须、也不应该因异步调用交易的失败而回滚.

即使 BETASCO 对节点的可信假设较为宽松, 智能合约的状态也有可能因为节点的故障而发生不一致. 因此, BETASCO 引入了检查点交易, 分片成员会定期使用智能合约状态的概要信息创建状态检查点, 通过共识协议与其他节点分享, 并在执行时检查状态是否一致. 如果状态不一致, 则不一致的分片成员检查自身保存的区块, 然后从交易序列中恢复正确的状态.

2.5 信任和故障模型

BETASCO 构建于多个组织或机构所属的节点之上, 通过组织或机构颁发的证书来验证节点的成员身份, 因此安全假设较为宽松. 但 BETASCO 将智能合约按一定的规律分散在网络之中, 并通过共识算法来维护智能合约的可信与持久性, 本质上仍然将所有节点都视为潜在的恶意节点或拜占庭节点, 因此可以适应灵活的信任和故障假设. 智能合约分片对拜占庭节点的容忍度取决于共识算法, 例如, PBFT 维护的智能合约分片可以容忍不超过 1/3 的拜占庭节点. 开发者可以通过选择不同的共识算法和分片规模, 以满足对智能合约可信性和持久性的不同需求. 此外, 提供节点的组织或机构也可以要求交易必须从其指定的入口服务器进入 BETASCO 系统, 通过智能合约定位服务将交易转发到目标智能合约分片成员节点, 而将转发过程和网络拓扑结构对外部完全隐藏, 以提高安全性.

BETASCO 的智能合约定位服务对智能合约执行流程具有重要影响, 虽然不会影响智能合约的执行性能. 但是会影响对终端用户的响应延迟, 也是面向动态网络容错的主要来源. BETASCO 面向联盟链环境, 其节点由多个组织或机构提供, 同属于一个组织或机构的节点位于相同的局域网中, 自有其网络管理与准入机制, 然后部分节点通过点对点的连接与其他组织或机构的节点相连. 因此, 整体上 BETASCO 的网络是去中心无结构的对等网络. 在此环境中, BETASCO 为智能合约构建了分布式的定位服务, 与对等网络的拓扑结构相结合, 以在缓慢变化的环境中提供持久的服务.

在交易进入 BETASCO 后, 就会被智能合约定位服务在节点间进行转发. 一般而言, 交易在组织或机构的子网中转发跳数较少、速度较快, 因此其转发代价和出错的可能主要发生在子网之间. 最坏情况下, 交易需要经过所有子网才能到达目标, 即在子网间转发的最大跳数为组织或机构的数量. 通过增加子网间节点的连接可以减少出现最坏情况的可能, 有业务上的关联的组织或机构建立跨子网的连接则更能有效减少子网间的交易转发跳数, 因此本文认为交易路由的代价是可以接受的. 考虑到为加速智能合约分片成员间交换交易和区块, 可以使其在构建分片之初就相互连接. 此时, 随着部署的智能合约的数量增加, ID 散列值相近的节点在转发交易前就会先一步建立连接. 这会进一步减少整体转发跳数, 增加稳定性.

定位服务对智能合约分片提供的面向动态网络的容错性则主要取决于智能合约分片的规模. 对节点离线的情况, 只要不是智能合约的分片成员全部同时离线, 智能合约就能够正常运行. 对新节点加入的情况, 如果新加入的节点数大于等于智能合约分片规模, 则定位服务中计算得到的分片对应的成员可能全部发生错位 (即最坏情况下新加入的节点 ID 散列值都与智能合约 ID 散列值更接近), 此时定位服务将不能将交易稳定转发给目标智能合约的分片成员. 换言之, 如果新加入的节点数小于智能合约分片规模, 则能够保证至少一个智能合约的分片成员保持在线, 然后在后续的过程中新成员将从老成员处获得区块和状态检查点等, 维持完整的状态和执行环境, 从而保证已部署智能合约服务的持久性. 在部署有多个智能合约分片的影响下, 同一时间离线的节点数量小于规模最小的

智能合约分片的节点数, 就能保证所有智能合约服务的持久性.

3 系统实现

BETASCO 的系统体系结构分为 5 层, 自底向上分别为存储、网络、共识、协议和应用 5 层. BETASCO 的创新主要集中在网络层和协议层. 本节首先概述 BETASCO 的整体架构, 然后重点介绍 BETASCO 的智能合约执行环境和编程模型.

3.1 系统体系结构

BETASCO 的系统体系结构如图 7 所示.

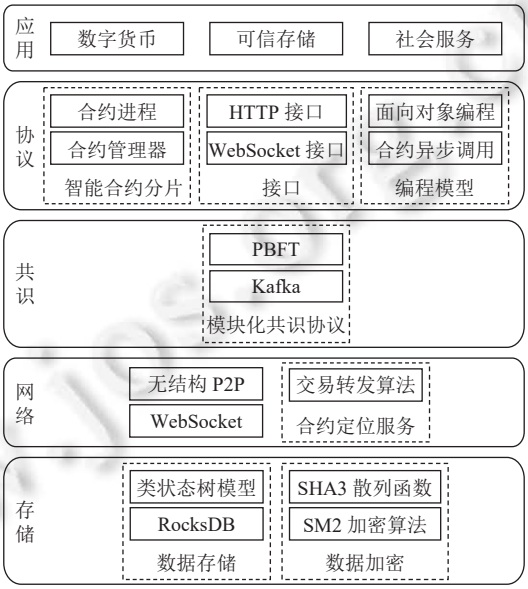

图 7 BETASCO 系统体系结构

• 存储层: BETASCO 借鉴状态树模型, 根据需要定义了新的交易和区块数据结构, 在底层使用 RocksDB, 并使用 SHA 散列函数和 SM2 加密算法, 在此基础上进行数据的交换、验证、存储、读取和检索.

• 网络层: BETASCO 的各组成节点之间通过 WebSocket 链接, 形成去中心无结构对等网络. 在此基础上, 所有节点的交易转发功能形成一个整体的智能合约定位服务, 负责将交易路由给目标智能合约的分片成员节点.

• 共识层: 以模块化的共识协议为主, 交易集合为输入, 定序后的交易序列为输出, 不涉及状态的验证. 所采用的共识协议以轻量级的共识协议为主, 例如 Kafka 排序和 PBFT 共识, 以适用于规模相对较小的合约分片.

• 协议层: 为智能合约提供独立执行环境, 维护有智能合约的代码和状态. 也包括用于接收交易和返回结果的 HTTP 接口和 WebSocket 接口. BETASCO 提供类 JavaScript 语言和相应执行器用于智能合约编程, 也提供有内置的功能性 API, 帮助开发者实现更为复杂的应用逻辑.

• 应用层: 由 BETASCO 中的智能合约支撑的去中心化应用, 可应用于数字货币、可信存储和社会服务等多个领域.

接下来的两节将介绍智能合约执行环境的实现, 以及帮助开发者实现智能合约间通信和协作的智能合约接口.

3.2 智能合约分片

BETASCO 通过将智能合约部署在独立的分片中实现智能合约执行环境的相互独立. 但按照传统的分片方式, 网络中节点的数目并无法满足智能合约的需要; 而且, 按照基于分布式散列表的智能合约定位服务, 不同智能合约对应的分片成员集很可能发生重叠. 因此 BETASCO 采用节点虚拟化来解决智能合约部署所需的节点数量和网络

中节点总数之间的冲突.

BETASCO 的智能合约分片框架如图 8 所示. 通过虚拟化, 物理上的一个节点可以对应逻辑上多个分片中的节点. 分片在逻辑上独立, 在物理上则可以有重叠, 这样就可以在集群中部署超出节点数量的智能合约. 因为分片在逻辑上的独立性, 每个智能合约都可以单独执行目标为自身的交易, 而不会造成相互干扰.

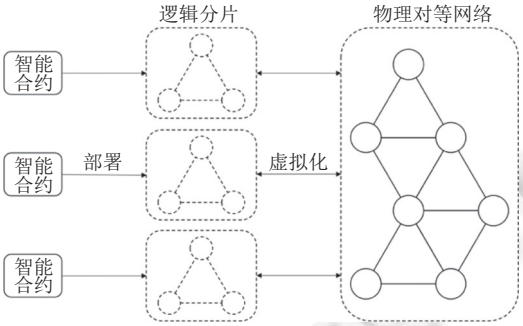

图 8 BETASCO 智能合约分片框架

BETASCO 的智能合约分片实现如图 9 所示. 首先, 智能合约只需要代码、交易序列和执行器就能够进行独立的状态校验与交易执行, 因此这三者可以构成一个智能合约的最小执行单元. BETASCO 将该执行单元称为合约进程, 物理上对应操作系统中的一个活动进程, 逻辑上则对应合约分片中的一个成员. 一个智能合约在一个节点上只有一个合约进程. 每个节点上唯一的合约管理器负责管理本地运行的合约进程, 包括消息分发和共识管理. 通过合约管理器上的共识模块, 新区块被产生并在数据存储中持久化, 然后交给相应的合约进程进行交易执行和状态更新. 因为执行环境的独立性, 状态副本不需要反复从数据存储中读取, 而是由合约进程管理在内存中, 相当于状态数据存储的缓存. 状态副本可以随着交易执行而更新, 当异常被触发时, 合约进程依然能从数据存储中恢复出执行前的状态.

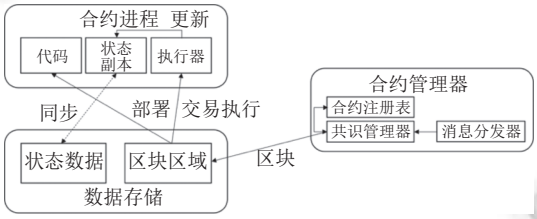

图 9 BETASCO 智能合约分片实现

3.3 智能合约接口

为帮助开发者开发复杂应用, 除 JavaScript 内置的 API 之外, BETASCO 为开发者提供了一些额外的 API.

• random() // 利用散列函数返回一个随机 16 进制数, 对同一个 ID 的交易执行返回的随机数相同.

• executeContract(cID, func, arg) // 以参数 arg 异步执行智能合约 cID 的方法 func, 不要求回调.

• executeContract(cID, func, arg, handler) // 以参数 arg 异步执行智能合约 cID 的方法 func, 将结果以调用 handler 的方式返回, cID 执行之后会再进行一次对该智能合约 handler 方法的调用.

• getContractInfo() // 返回智能合约信息, 包括合约名等.

• getExecuteInfo() // 返回执行信息, 包括区块 ID、参数 ID 等.

4 系统效果评估

本文测试了 BETASCO 的性能, 研究了分片规模、共用分片的智能合约数量以及智能合约间异步调用对 BETASCO 的性能影响, 并与联盟区块链系统 HyperLedger Fabric 进行对比.

4.1 实验设置

本文使用了 60 个云节点用于测试, 测试环境见表 1.

表 1 测试环境信息

测试环境	测试集群	客户端 (压测机)
硬件环境	2块vCPU (2.5 GHz主频) 4 GiB内存 40 GB硬盘 5 Mb/s网络	
软件环境	操作系统CentOS 8.2 64 bit OpenJDK Runtime Environment (build 1.8.0_302-b08)	
环境部署	BETASCO v1.4.5	压力测试工具http_load 09Mar2016
数量与分布	10台阿里云服务器 10台华为云服务器 10台天翼云服务器	10台阿里云服务器 10台华为云服务器 10台天翼云服务器

Ethereum 作为最大的去中心化应用平台, 其交易流量中涉及较多的智能合约, 在本文为数据分析所收集的 9 705 个区块中就涉及了 30 267 个智能合约. 在大规模高性能云节点的基础上, BETASCO 可以支持使用以太坊交易数据进行实验, 但是代价较高. 如 HyperLedger Fabric[23]和 Meepos[27], 则选择构造用于维护加密数字货币的智能合约, 并使用仿真数据集进行测试和评估. 虽然用 BETASCO 也可以实现类似的去中心化应用, 但是通过智能合约发行的加密数字货币很少有在设计上使用多个智能合约协作的情况.

在缺乏数据来源, 无法对现有联盟链应用的内部架构和业务逻辑进行统计和分析的情况下, 本文考虑含有独立日志存储与分析系统的应用场景: 应用的主要逻辑实现在一个智能合约中 (合约 MainService), 在运行过程中主应用构件通过异步调用方式在日志智能合约 (合约 Logger) 中记录日志. 智能合约的实现如图 10 所示. 这样既可以对日志进行即时记录和分析, 也不会影响主构件提供的应用服务.

```
contract MainService { // 主应用构件
  function onCreate() { // 合约的构造函数
    // Global是合约的状态对象，所有需要合约维护的状态必须创建为Global的成员
    Global.count = 0; // 代表应用构件内的主要状态
    Global.logger = ""; // 记录对应的日志构件的合约ID
  }
  // export是合约关键词，相当于Java中的public
  export function call(arg) {
    Global.count++;
    if (Global.logger) { // 只有在记录了logger的情况下才会启用日志记录
      executeContract(Global.logger, "log", "call"); // 记录本次执行信息
    }
  }
  export function get(arg) {
    if (Global.logger) { // 只有在记录了logger的情况下才会启用日志记录
      executeContract(Global.logger, "log", "get"); // 记录本次执行信息
    }
    return Global.count;
  }
  // 参数loggerID，日志构件的合约ID
  export function registerLogger(loggerID) {
    Global.logger = loggerID; // 注册日志构件的合约ID
  }
}
```

```
contract Logger { // 日志构件合约
  onCreate() { // 构造函数
    Global.logs = []; // 日志项
  }
  // 日志记录方法，参数content，代表日志信息
  export function log(content) {
    Global.logs.append(content)
  }
  // 返回记录的日志项的个数，代表简单的分析功能
  export function analyze(arg) {
    return Global.logs.length;
  }
}
```

图 10 智能合约 MainService 和 Logger 的实现

对 BETASCO 的性能测试包含 3 个部分: (1) 整体性能测试, 即在不同的分片中部署 MainService 和 Logger 合约, 并在注册 Logger 合约前后对 MainService 进行性能测试; (2) 智能合约分片的测试, 在不同数量的节点中部署分片规模等于节点数量的多个 MainService 合约, 然后测试其整体吞吐量, 检查分片规模和同分片智能合约数量对性能的影响, 以验证通过节点虚拟化实现的合约分片的效果; (3) 合约间异步调用的测试, 在不同规模的分片中同时部署 MainService 合约和 Logger 合约, 并将 Logger 合约注册为 MainService 合约的日志构件, 然后单独对

MainService 进行性能测试, 以考察异步调用对智能合约执行性能的影响.

在实验中, 分片的共识算法采用 Kafka 排序.

根据智能合约部署的分片规模, 每次实验中使用同等规模的压力机向 BETASCO 部署节点发起对 MainService.call 的调用, 然后计算正常响应的返回结果数量. 实验中对每组参数进行多次测试, 每次测试持续 30 s, 然后取返回结果数量的平均值并以此来计算吞吐量. 为避免冷启动带来的性能影响, 对每组参数做测试前会进行两次测试以进行预热.

因为在分片部署方案和跨分片智能合约调用交易的处理上与其他现有区块链分片技术有根本性的不同, 本文选择最热门的联盟链技术 HyperLedger Fabric 对单个智能合约分片的性能进行比较评估. HyperLedger Fabric 集群部署于同一批测试节点上, 基于 3 个节点的 Zookeeper 3.4.14 服务和 4 个节点的 Kafka 1.02 服务, 都使用 Docker 20.10.8 部署, Fabric 版本为 1.4.8, 使用与 MainService 等价的智能合约测试提交交易的性能. 该集群中的节点分别属于两个组织, 要求每个交易至少得到任意一个组织的任意一个节点的背书, 每个区块最多容纳 1 024 条交易. 对照分片规模数量划分, Fabric 集群的性能测试分为相同的组数, 每组测试中使用相应数量的排序节点.

4.2 结果与分析

4.2.1 整体性能

为测试 BETASCO 的整体性能, 本文将测试集群的 30 个节点分为 6 个规模为 5 的分片, 彼此互不重叠, 分别部署 5 个 MainService 合约和 1 个 Logger 合约, 通过性能测试来检查理想状态下整体的 TPS, 以此验证 BETASCO 分离智能合约执行环境后对去中心化应用的性能提升.

图 11 表明, 随着智能合约数量的增多, 这些智能合约的整体吞吐量呈线性上升, 这符合区块链分片技术中整体吞吐量随分片数量增加而线性上升的规律. 并且可以发现, 是否有智能合约间的异步调用对这些智能合约的执行效率并无太大影响. 这是因为 MainService 合约对 Logger 的合约间调用会转化为交易, 交给 Logger 所在的分片异步执行了, 并不干扰后续的交易处理, 这就在完成智能合约间协作的同时保持了每个智能合约的高性能. 另外值得一提的是, 在两个以上注册了 Logger 合约的 MainService 合约进行性能测试时, Logger 合约因为收到了超出其处理性能的交易, 所以在每次测试之后都有一段时间处于忙碌状态, 直到处理完所有 MainService 合约发起的交易后才能对新交易进行处理.

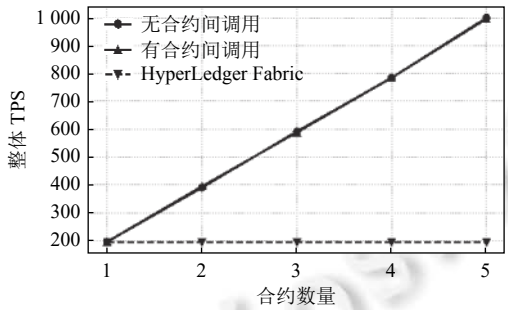

图 11 分片不重叠时智能合约的整体 TPS

4.2.2 智能合约分片

为测试分片规模和同分片智能合约数量对性能的影响, 本文针对不同分片规模的智能合约进行性能测试, 同时与同等规模的排序集群下 HyperLedger Fabric 的性能进行对比. HyperLedger Fabric 的智能合约执行流程为“执行-排序-验证”这 3 个阶段, 其中“执行阶段”是收集背书、用于帮助排序节点排序的交易执行的读写集合, 本质上是“预执行”, 并不会改变智能合约的状态. 其正式的执行流程和 BETASCO 一样也是从第 2 阶段“排序” (BETASCO 第 2 阶段称“定序”) 开始, 然后在第 3 阶段“验证” (BETASCO 第 3 阶段称“执行”) 改变账本状态之后结束. 因此在实验中我们主要通过测量交易吞吐量来对比 BETASCO 的“定序”“执行”阶段和 HyperLedger Fabric 的“排序”“验证”阶段的性能, 以此评估 BETASCO 的性能.

考虑到传统分布式系统中数据的默认备份数都为 3, 取分片规模的最小值为 3, 分片规模的最大值为总节点数量的一半, 则分片规模的取值范围为 [3, 15]. 另外, BETASCO 中不同智能合约对应的分片可能存在重叠, 实验中考虑极端情况——所有智能合约的分片对应物理上的同一组节点. BETASCO 为每个智能合约进程分配 200 MB 的内存, 实验中考虑同一个节点上同时最多有 5 个智能合约进程的情况, 即占用节点将近 1/4 的内存.

于是, 本文得到了如表 2 所示的实验结果.

表 2　分片/排序集群规模与同分片智能合约数量对 TPS 的影响

分片/排序集群规模节点数 (个)	合约数量					HyperLedger Fabric
	1	2	3	4	5	
3	188.42	191.02	194.36	196.82	197.04	192.74
5	193.34	193.42	196.24	196.81	196.24	193.59
7	188.92	193.42	190.11	190.88	192.61	194.12
9	192.48	192.52	193.80	194.06	194.13	192.11
11	190.87	190.91	189.52	189.26	190.30	192.90
13	190.06	189.26	188.72	189.61	189.77	193.00
15	187.56	188.07	188.72	188.78	188.95	194.66

结果显示, 部署相同数量的智能合约时, 分片规模越大, 能承载的最大 TPS 越低. 这是因为分片规模越大, 需要处理的通信数量就越多, 导致整体上交易的处理效率的下降. 另一方面, 在同一个确定规模的分片中部署的智能合约数量越多, 分片整体的 TPS 却略有提升. 这说明智能合约的执行环境分开后, 虽然仍然需要共享分片的计算资源, 但不再需要抢占唯一的执行器, 同一个节点上不同的合约进程并发地处理交易, 则所有智能合约的交易就会在同分片不同的节点上并行地执行, 缩短了每个交易的响应时间, 因此 TPS 会有略微的提升. 与采用相同数量的排序节点的 Fabric 集群相比, 在少于 5 个节点的分片规模/排序节点数量下, BETASCO 的整体 TPS 略高于 Fabric.

4.2.3　智能合约异步调用

本文在同一个分片内部署多个 MainService 合约和一个 Logger 合约, 在启用和未启用智能合约间异步调用的情况下分别进行性能测试, 以检查智能合约间异步调用对整体 TPS 的影响. 为了更显著地展示 TPS 的变化, 根据第 1 阶段的测试结果, 第 2 阶段的性能测试采用 5 个节点的分片, 部署 5 个 MainService 合约和 1 个 Logger 合约. 逐个启用 MainService 合约的 Logger 功能 (即调用 registerLogger 方法) 后, 对已启用 Logger 的 MainService 合约进行性能测试.

如图 12 所示, 启用异步调用的智能合约数量对整体 TPS 的影响不大. 与阶段 1 中的实验结果类似的, 启用异步调用的智能合约数增加时, 整体 TPS 也略微增加. 考虑到对 MainService 的每次调用都会触发一次对 Logger 的异步调用, 也就相当于发布了两个交易, 因此本文将实验结果与无智能合约间调用时同分片内同等数量智能合约的整体 TPS 做对比. 可以看出, 对同等数量的智能合约做性能测试时, 有智能合约间调用的整体 TPS 比无智能合约间调用的整体 TPS 的一半略高, 这也符合从阶段 1 实验结果中得出的推论.

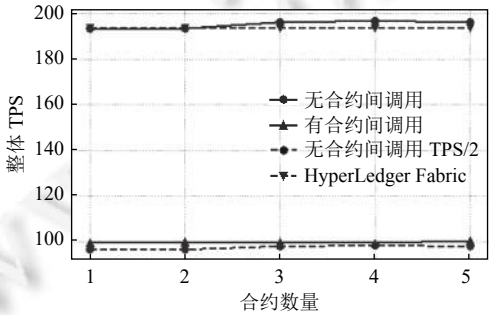

图 12　智能合约异步调用对 TPS 的影响

4.2.4 讨 论

上述 3 个部分的实验验证了 BETASCO 在维护高性能智能合约与支持智能合约间异步调用的有效性, 但是 BETASCO 也有一定的局限性: 实现为独立进程的合约进程虽然能更好地利用系统级并发, 但也导致智能合约并不能无限制地部署. 根据上面的实验结果, 本文对 BETASCO 承载智能合约的能力及其性能进行了一定的分析.

假设 BETASCO 部署于由 N 个节点构成的网络中, 每个节点上除 BETASCO 服务外还可以启动 p 个合约进程. 简便起见, n 个智能合约被部署在规模相同为 k 的分片中, 每个分片部署一个智能合约时交易吞吐量为 t, 且 k 能整除 N. 根据实验结果, 在 n 小于等于 $\frac{N}{k}$ 时, 按照理想情况, 所有智能合约分片在物理上互不重叠, 则整体交易吞吐量 TPS 随 n 的增加而线性增长, 即:

$$TPS = nt,\ n \leqslant \frac{N}{k} \tag{1}$$

当 $n = \frac{N}{k}$ 时, 不重叠的智能合约分片已经在物理上占据了网络中的所有节点, 则新的智能合约分片必与旧的智能合约分片重叠. 此时最优部署策略是和某个智能合约部署在相同的分片上, 并且优先选择智能合约数量最小的分片以最大化每个智能合约的交易吞吐量. 根据实验结果, 在同分片智能合约数量增加时, 整体交易吞吐量略有提升. 假设一个分片中, 整体交易吞吐量随合约数量增加而增加的系数为 q, 且 $n = \frac{N}{k} \times s + v$, $s \leqslant p-1$, $v < \frac{N}{k}$ 此时 v 个分片部署了 $s+1$ 个智能合约, $n-v$ 个分片部署了 s 个智能合约, 则有:

$$TPS = v(q(s+1-1)+t)+(n-v)(q(s-1)+t) = n(qs+t)-q(n-v) \tag{2}$$

$$n \leqslant \frac{pN}{k}, s = \left\lfloor n \middle/ \frac{N}{k} \right\rfloor = \left\lfloor \frac{nk}{N} \right\rfloor, v = n - \frac{sN}{k} \tag{3}$$

直到每个分片都部署了 p 个智能合约, 即每个节点上都有 p 个合约进程时, 新的智能合约就无法部署了, 因为每个节点的 CPU 和内存等资源都无法再支持启动一个新合约进程, 或者必须挂起一个旧合约进程, 此后 BETASCO 的整体交易吞吐量将恒定为:

$$TPS = \frac{N}{k}(q(p-1)+t), n > \frac{pN}{k} \tag{4}$$

综上所述, BETASCO 在智能合约满足最优部署策略的情况下, 智能合约数量不多于网络中能部署的最大不重叠分片数时, 交易吞吐量能随着智能合约数量增加而线性增长, 之后虽然仍然会随着智能合约数量增加而略微增长但增长趋势变缓. 直到每个节点都不能启动新的合约进程时, 网络中智能合约部署数量达到饱和, BETASCO 提供的整体交易吞吐量会自此保持稳定.

4.2.5 小 结

通过性能测试, 本文验证了 BETASCO 系统设计和实现的有效性. 通过将智能合约部署在不同的分片中, 隔离了它们的执行环境, 使其执行过程不会受到其他智能合约的干扰. 不同智能合约通过节点虚拟化部署在同一组节点中, 因此分片在物理上可能存在重叠. 在理想情况下, 每个智能合约都部署在不重叠的分片中时, 整体吞吐量随智能合约数量, 也就是分片数量的增加而线性增长. 即使在极端情况下, 不同智能合约对应的分片完全重叠, 整体吞吐量仍能随智能合约数量增多而略有提高.

另一方面, 因为智能合约间异步调用本质上是创建了新的交易, 所以当调用者和被调用者处于同一个分片内时, 整体吞吐量会受到影响, 同时也验证了智能合约间异步调用的有效性. 若调用者和被调用者不处于同一个分片时, 被调用者的交易执行不会干扰调用者的后续交易的执行, 但是被调用者单位时间收到的交易可能会超出其处理性能, 导致被调用者一定时间内的阻塞.

5 总结与展望

本文提出并实现了 BETASCO, 一种面向智能合约分片的联盟区块链系统. BETASCO 将每个智能合约部署在不同的分片中以提供独立的运行环境, 使用基于分布式散列表的智能合约定位技术将交易路由给目标智能合约的

部署分片, 以保证智能合约执行过程不会受到其他智能合约的干扰, 同时使用异步调用机制支持智能合约跨分片的通信和协作. 因为共识参与节点数量的减少, BETASCO 得以使用轻量而快速的共识算法维护智能合约, 从而在联盟区块链环境中支持高性能的去中心化应用.

在信任和故障模型的讨论中, 本文分析了在联盟链环境中智能合约定位服务对交易转发性能和智能合约崩溃容错的影响, 没有对新节点加入导致两次智能合约定位得到的智能合约分片完全不重叠的情况进行处理, 而这会导致交易只能通过洪泛方式路由, 大大影响终端用户的体验. 因此本文的后续工作方向主要是提升智能合约定位算法的稳定性, 并加快交易路由过程. 此外, 随着智能合约部署数量的增加, 多个智能合约分片重合部分的节点可能会与其他节点建立大量网络连接, 也需要对节点上的连接建立有效的管理, 以减少网络连接数量, 降低通信负载.

References:

[1] Zheng ZB, Xie SA, Dai HN, Chen XP, Wang HM. An overview of blockchain technology: Architecture, consensus, and future trends. In: Proc. of the 2017 IEEE Int'l Congress on Big Data. Honolulu: IEEE Computer Society, 2017. 557–564. [doi: 10.1109/BigDataCongress.2017.85]

[2] Huang G, Luo CR, Wu KD, Ma Y, Zhang Y, Liu XZ. Software-defined infrastructure for decentralized data lifecycle governance: Principled design and open challenges. In: Proc. of the 39th IEEE Int'l Conf. on Distributed Computing Systems. Dallas: IEEE, 2019. 1674–1683. [doi: 10.1109/ICDCS.2019.00166]

[3] Ren YJ, Leng Y, Qi J, Sharma PK, Wang J, Almakhadmeh Z, Tolba A. Multiple cloud storage mechanism based on blockchain in smart homes. Future Generation Computer Systems, 2021, 115: 304–313. [doi: 10.1016/j.future.2020.09.019]

[4] Cai T, Lin H, Chen WH, Zheng ZB, Yu Y. Efficient blockchain-empowered data sharing incentive scheme for Internet of Things. Ruan Jian Xue Bao/Journal of Software, 2021, 32(4): 953–972 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6229.htm [doi: 10.13328/j.cnki.jos.006229]

[5] Liu ZY, Li ZP. A blockchain-based framework of cross-border e-commerce supply chain. Int'l Journal of Information Management, 2020, 52: 102059. [doi: 10.1016/j.ijinfomgt.2019.102059]

[6] Pîrlea G, Kumar A, Sergey I. Practical smart contract sharding with ownership and commutativity analysis. In: Proc. of the 42nd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation. ACM, 2021. 1327–1341. [doi: 10.1145/3453483.3454112]

[7] Dang H, Dinh T T A, Loghin D, Chang EC, Lin Q, Qoi BC. Towards scaling blockchain systems via sharding. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 123–140. [doi: 10.1145/3299869.3319889]

[8] ConsenSys. The inside story of the CryptoKitties congestion crisis. 2018. https://consensys.net/blog/news/the-inside-story-of-the-cryptokitties-congestion-crisis/

[9] Wu KD, Ma Y, Huang G, Liu XZ. A first look at blockchain-based decentralized applications. Software: Practice and Experience, 2021, 51(10): 2033–2050. [doi: 10.1002/spe.2751]

[10] Liu XL, Muhammad K, Lloret J, Chen YW, Yuan SM. Elastic and cost-effective data carrier architecture for smart contract in blockchain. Future Generation Computer Systems, 2019, 100: 590–599. [doi: 10.1016/j.future.2019.05.042]

[11] Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, Ford B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In: Proc. of the 2018 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE Computer Society, 2018. 583–598. [doi: 10.1109/SP.2018.000-5]

[12] Wen FL, Yang L, Cai W, Zhou P. DP-Hybrid: A two-layer consensus protocol for high scalability in permissioned blockchain. In: Proc. of the 2nd Int'l Conf. on Blockchain and Trustworthy Systems. Dali: Springer, 2020. 57–71. [doi: 10.1007/978-981-15-9213-3_5]

[13] Zamani M, Movahedi M, Raykova M. Rapidchain: Scaling blockchain via full sharding. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 931–948. [doi: 10.1145/3243734.3243853]

[14] Wang JP, Wang H. Monoxide: Scale out blockchains with asynchronous consensus zones. In: Proc. of the 16th USENIX Symp. on Networked Systems Design and Implementation. Boston: USENIX Association, 2019. 95–112. [doi: 10.13140/RG.2.2.32017.48489]

[15] Harmony team. Harmony: Technical whitepaper. 2018. https://harmony.one/whitepaper.pdf

[16] The Elrond Team. Elrond EGLD: A highly scalable public blockchain via adaptive state sharding and secure proof of stake. 2020. https://whitepaper.io/coin/elrond

[17] Ethereum Wiki. Shard chains. 2022. https://ethereum.org/en/eth2/shard-chains/

[18] Al-Bassam M, Sonnino A, Bano S, Hrycyszyn D, Danezis G. Chainspace: A sharded smart contract platform. In: Proc. of the 25th Annual

Network and Distributed System Security Symp. San Diego: The Internet Society, 2018.

[19] Martino W, Quaintance M, Popejoy S. Chainweb: A proof-of-work parallel-chain architecture for massive throughput. 2018. https://neironix.io/documents/whitepaper/6793/chainweb-v15.pdf

[20] Fitzi M, Gaži P, Kiayias A, Russell A. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, 2018.

[21] Yu HF, Nikolić I, Hou RM, Saxena P. OHIE: Blockchain scaling made simple. In: Proc. of the 2020 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2020. 90–105. [doi: 10.1109/SP40000.2020.00008]

[22] Wang Q, Li RJ. A weak consensus algorithm and its application to high-performance blockchain. In: Proc. of the 2021 IEEE Conf. on Computer Communications (IEEE INFOCOM 2021). Vancouver: IEEE, 2021. 1–10. [doi: 10.1109/INFOCOM42981.2021.9488725]

[23] Androulaki E, Barger A, Bortnikov V, *et al.* HyperLedger Fabric: A distributed operating system for permissioned blockchains. In: Proc. of the 13th EuroSys Conf. Porto: ACM, 2018. 30. [doi: 10.1145/3190508.3190538]

[24] Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A scalable peer-to-peer lookup service for Internet applications. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149–160. [doi: 10.1145/964723.383071]

[25] Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proc. of the 2001 IFIP/ACM Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing. Heidelberg: Springer, 2001. 329–350. [doi: 10.1007/3-540-45518-3_18]

[26] Maymounkov P, Mazières D. Kademlia: A peer-to-peer information system based on the XOR metric. In: Proc. of the 1st Int'l Workshop on Peer-to-peer Systems. Cambridge: Springer, 2002. 53–65. [doi: 10.1007/3-540-45748-8_5]

[27] Zheng PL, Xu QQ, Zheng ZB, Zhou ZY, Yan Y, Zhang H. Meepo: Sharded consortium blockchain. In: Proc. of the 37th IEEE Int'l Conf. on Data Engineering. Chania: IEEE, 2021. 1847–1852. [doi: 10.1109/ICDE51399.2021.00165]

附中文参考文献:

[4] 蔡婷, 林晖, 陈武辉, 郑子彬, 余阳. 区块链赋能的高效物联网数据激励共享方案. 软件学报, 2021, 32(4): 953–972. http://www.jos.org.cn/1000-9825/6229.htm [doi: 10.13328/j.cnki.jos.006229]

吴恺东(1994－), 男, 博士生, CCF 学生会员, 主要研究领域为软件工程, 区块链.

景翔(1979－), 男, 博士, CCF 专业会员, 主要研究领域为新型软件定义理论, 分布式系统架构, 工业互联网.

马郓(1989－), 男, 博士, 研究员, 博士生导师, CCF 专业会员, 主要研究领域为智能系统软件, Web 系统和移动系统的设计和优化.

黄罡(1975－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为系统软件, 软件自适应.

蔡华谦(1990－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为编程语言, 软件工程.