计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(10):34−44 [doi: 10.15888/j.cnki.csa.009253]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于键值状态分片的 HyperLedger Fabric 性能优化①

董建亮[1], 潘恒宇[1], 王　硕[1], 王　峰[2], 李　京[1]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 网络信息中心, 合肥 230026)
通信作者: 李　京, E-mail: lj@ustc.edu.cn

摘　要: HyperLedger Fabric 是受关注度较高的开源联盟区块链. 针对现有区块链分片方法不适用于 Fabric 三阶段交易模型的问题和分片粒度过粗导致存在热点访问的问题, 提出一种基于 Fabric 实现的细粒度键值状态分片方法. 首先, 详细设计了 Fabric 在键值状态分片下的跨片交易处理, 引入跨分片排序节点和两阶段提交处理流程, 高效保证跨片交易的一致性和原子性. 然后, 针对细粒度分片可能导致交易跨片概率上升进而影响性能的问题, 提出启发式的交易提案路由表, 旨在减少预执行阶段交易的跨片读数据请求, 降低计算资源和网络资源的消耗. 最后, 在 Fabric 仿真系统上实现改进的分片方案并进行性能测试. 实验结果表明, 该方法在提升 Fabric 性能的基础上, 有效解决了热点访问问题和高跨片交易占比下的性能下降问题.

关键词: 区块链; 联盟链; HyperLedger Fabric; 分片; 横向扩容

引用格式: 董建亮,潘恒宇,王硕,王峰,李京.基于键值状态分片的 HyperLedger Fabric 性能优化.计算机系统应用,2023,32(10):34–44. http://www.c-s-a.org.cn/1003-3254/9253.html

HyperLedger Fabric Performance Optimization Based on Key-value State Sharding

DONG Jian-Liang[1], PAN Heng-Yu[1], WANG Shuo[1], WANG Feng[2], LI Jing[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Network Information Center, University of Science and Technology of China, Hefei 230026, China)

Abstract: HyperLedger Fabric is an open-source consortium blockchain that has received a lot of attention. Since the existing blockchain sharding method is not suitable for the three-stage transaction model of Fabric, and there is the problem of hotspot access caused by coarse sharding granularity, a fine-grained key-value state sharding method based on Fabric is proposed. First of all, the cross-shard transaction processing of Fabric under key-value state sharding is designed in detail, and a cross-shard sequence node and a two-stage submission process are introduced to efficiently ensure the consistency and atomicity of cross-shard transactions. Then, in view of the problem that fine-grained sharding may lead to an increase in the probability of transaction cross-shards and thus affect performance, a heuristic transaction proposal routing table is proposed to reduce the cross-shard data read requests of transactions in the pre-execution stage and lower the consumption of computing and network resources. Finally, the improved sharding scheme and performance test are realized on the Fabric simulation system. The experimental results show that on the basis of improving the performance of Fabric, this method effectively solves the hotspot access problem and the performance degradation problem under the high proportion of cross-shard transactions.

Key words: blockchain; consortium blockchain; HyperLedger Fabric; sharding; horizontal scalability

① 基金项目: 安徽省高校省级质量工程重大教育教学改革研究项目 (2019zdjg30)
收稿时间: 2023-03-14; 修改时间: 2023-04-20; 采用时间: 2023-04-28; csa 在线出版时间: 2023-08-09
CNKI 网络首发时间: 2023-08-10

区块链是一种分布式账本技术, 具有去中心化、防篡改、匿名性等优点[1], 在金融[2]、教育[3]、能源[4]等诸多行业有着广泛的应用. HyperLedger Fabric (后文简称 Fabric)[5] 是 Linux 基金会主导、IBM 开发的一个模块化、可扩展的开源联盟链系统. 目前, 已经有许多政府和商业组织使用 Fabric 去支持联盟业务[6].

在文献 [7–9] 中的性能测试实验表明, Fabric 的性能比传统联盟区块链更优秀, 吞吐量能达到三千多 TPS (每秒交易数), 但是其与实际的中心化交易系统几万甚至几十万的 TPS 存在较大差距. Fabric 的性能无法支撑现代金融、物联网等领域的应用, 因此需要进一步的优化.

分片是一种能够有效提升区块链性能的技术[10]. 目前在公有链或者联盟链上的分片方法[11–13] 无法很好地应用于 Fabric, 有以下两点原因.

1) 传统联盟链采用两阶段交易模型“排序-执行”, Fabric 采用三阶段交易模型“执行-排序-验证”. 交易的处理逻辑存在较大差别, 导致分片后交易处理逻辑需要重新设计.

2) 很多区块链分片方法的分片粒度比较粗, 或基于智能合约或基于账户数据. 如图 1 所示, 因为区块链上的数据存在热点访问, 分片粒度较粗容易导致负载不均衡问题.

在 Fabric 中, 智能合约分为逻辑代码和状态数据两部分, 状态数据被存储在 key-value 类型数据库中. 如图 1 所示, 为了有效解决智能合约热点访问问题. 本文将同一智能合约的键值数据分配到不同分片, 实现更细粒度的分片.

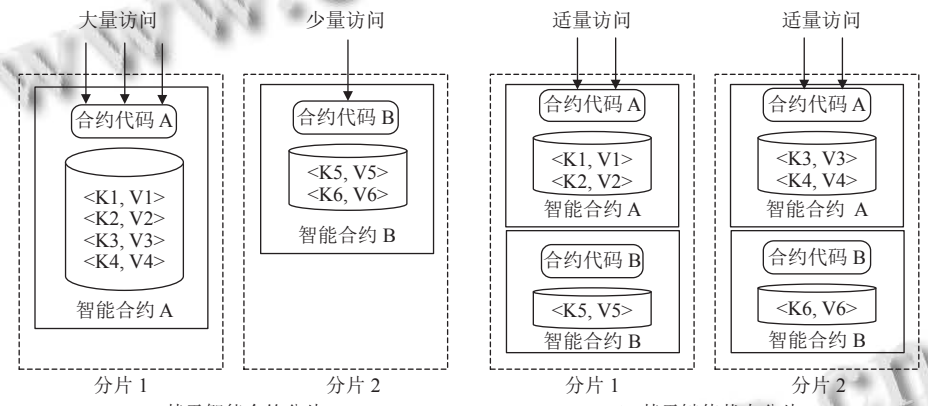

图 1 热点访问示意图

细粒度分片虽然有效解决了智能合约的热点访问问题, 但是容易引发更多的跨片交易, 影响分片改善性能的效果. 本文观察发现客户发送交易读写的数据存在时间局部性, 故提出启发式的交易提案路由表, 减小跨片交易的代价.

综上, 本文使用细粒度的键值状态分片方法改造 Fabric, 在优化 Fabric 性能的基础上, 解决智能合约热点访问问题和减小高跨片交易占比对分片性能的影响, 为使用 Fabric 搭建联盟链的组织或企业提供一种改善 Fabric 扩容能力的方案. 本文主要贡献如下.

1) 实现 Fabric 细粒度键值状态分片, 引入跨片排序节点和两阶段提交保证跨片交易的一致性和原子性, 优化 Fabric 性能并解决智能合约热点访问问题.

2) 提出启发式的交易提案路由表方案, 降低跨片交易跨片访问带来的时延, 减小高跨片交易占比对分片性能的影响.

3) 在 Fabric 仿真系统上实现本文的分片方案并进行性能测试. 实验结果表明, 细粒度键值分片 Fabric 的性能比智能合约分片 Fabric 更好, 并且有效解决了热点问题. 在跨片交易占比高时, 使用交易提案路由表可以有效阻止分片 Fabric 性能下降.

1 相关研究

目前有许多优化 Fabric 性能的研究工作. Yuan 等人[14] 提出使用广义随机 Petri 网 (GSPN) 来分析 Fabric 系统的性能, 通过分析不同配置对系统性能的影响提出一种配置选择方法来最大化系统吞吐量. Gorenflo 等人[15] 提出了一系列优化措施, 例如分离交易元数据并

只对交易 ID 排序、缓存区块解析、分离验证节点的背书功能和提交功能避免硬件资源竞争等, 从而减少 Fabric 排序阶段和验证阶段的 I/O 开销和计算开销. Sharma 等人[16] 提出在排序阶段构建交易依赖图, 通过分析依赖图丢弃部分交易, 重排序交易以最大化交易成功数. Ruan 等人[17] 在前者基础上全面分析交易冲突, 提出更细粒度的冲突处理. Hang 等人[18] 提出一种基于模糊逻辑的交易流量控制方法来提升 Fabric 交易处理性能.

以上对 Fabric 的研究通过优化参数配置、减少资源开销和交易冲突等方式优化 Fabric 性能, 但是无法有效地利用新增的服务器去提升性能, 不能改善 Fabric 的横向扩容能力. 横向扩容能力是指当服务器数量增加后, 系统的吞吐量按比例增长的能力. 分片将区块链网络分割成多个区域, 每个区域作为一个分片. 交易根据特定规则发送到不同分片, 实现交易的并行处理. 在实际应用中, 可以利用新增服务器资源

创建分片, 提升总体吞吐量. 因此引入分片技术能够改善 Fabric 的横向扩容能力. 目前有一些引入分片改造 Fabirc 的研究工作. Androulaki 等人[19] 提出了基于通道的分片设计, 其将每个通道 (在 Fabric 中, 一个通道管理一条区块链) 作为一个分片, 通道之间通过预言机来发现通道和验证数据有效性. 但是该工作没有通过定量实验证明方案的有效性. Thakkar 等人[9] 提出了稀疏节点的概念, 该节点会有选择性地提交区块中的部分交易, 从而避免组织内多个节点间冗余工作的问题. 让每个稀疏节点通过过滤交易来执行和验证区块中的部分交易, 实现交易分片. 其在验证阶段采用了依赖分析和数据同步的方式保证并发交易的隔离性, 分片间的同步消息复杂度是 $O(N^2)$ 的, 效率较低.

以上 Fabric 分片研究集中于设计和实现一个 Fabric 分片系统, 但都没有考虑智能合约热点访问和跨片交易对分片性能的影响, 且存在缺乏实验分析、效率不高等缺点.

在公链分片领域有许多工作研究如何减少公链分片系统中的热点访问和跨片交易. Hong 等人[20] 设计了一个分层次的分片系统, 通过让一些分片持有多个分片数据的方式将跨片交易转化为片内交易. Huang 等人[21] 通过构建账户交易状态图并使用图分割算法重分配账户状态来实现分片的负载均衡. Li 等人[22] 为重分配账户的迁移设计了高效安全的迁移协议. 以上公链分片研究都集中于公链领域, 无法支持 Fabric 这种三阶段交易模型的联盟链系统. 而且重分配的方法会要求额外的角色, 因为状态的迁移会打破状态与分片的映射关系, 需要额外的节点存储映射索引、管理元数据和计算重分片策略.

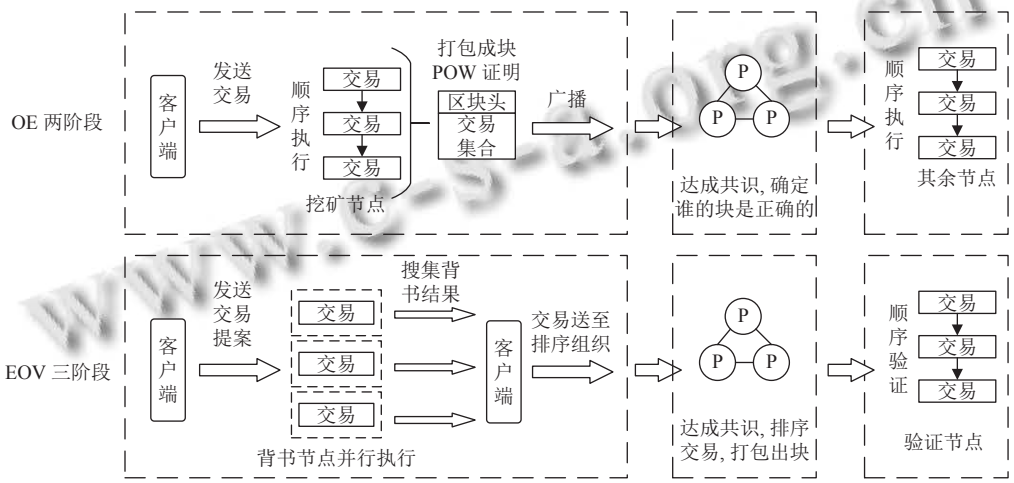

图 2 两阶段与三阶段交易流程示意图

2 相关知识

Fabirc 是一个联盟链系统, 由多个组织维护一个通道, 通道内各组织持有一致的区块链账本和世界状态. 每个组织的节点需要持有各自组织的证书颁发机构 (CA) 所颁发的身份证书来参与到联盟链的工作中. Fabric 引入三阶段交易流程“执行-排序-验证” (EOV), 其与传统的两阶段交易流程“排序-执行” (OE) 区别如图 2 所示. 以图 2 为例简单介绍 Fabric 的交易处理流

程, 关于 Fabric 的详细知识可参考文献 [5].

Fabric 中有两种类型的节点, 一是 Peer 节点, 其参与背书和验证两个工作. 二是 Orderer 节点, 其参与排序工作.

智能合约执行: 客户端会根据想要调用智能合约的背书策略, 向多个组织的 Peer 发送交易提案, 提案里包含身份证书和调用智能合约的参数. Peer 收到交易提案, 会进行合约执行. Fabric 使用了 MVCC (多版本并发控制) 技术, 合约执行过程中会查询状态数据库生成读集合与写集合, 读集合里包含了合约调用过程中查询的对象的当前版本号, 写集合则包含了要修改的对象的新值. 此阶段不会将写集合更新到账本, 所以也称为“模拟执行”. 最后 Peer 会对这份结果进行数字签名, 代表组织为这份提案背书. 背书结果会被返回给客户端.

背书搜集: 客户端在搜集到满足背书策略数量的背书后, 会比较读写集结果, 然后将读写集合和有效的多个背书签名打包生成交易, 发送给排序组织的 Orderer 节点. Fabric 通过背书策略来限制节点作恶行为, 所以排序服务可以使用非拜占庭的共识算法.

排序服务: 排序组织的 Orderer 节点收到交易后, 运行共识算法 (目前 v2 版本 Fabric 默认共识为 Raft[23]) 对一定数量的交易顺序达成一致并生成区块. 将区块发送给各个组织的主节点 Peer (由 Gossip 算法选举). 主节点通过 Gossip 算法在组织内部做进一步的分发.

验证提交: Peer 收到交易后, 首先校验交易数据格式, 然后判断交易为配置更新交易还是智能合约调用交易. 若为后者, 则进入验证阶段: 先并发地验证各个交易地背书策略是否满足; 然后顺序地进行交易的 MVCC 检查, 具体是检查读集合中键的版本号与当前状态库中是否一致, 上述检查都通过的交易视为有效. 最后 Peer 将区块附加有效交易位图后保存到本地账本, 按顺序更新有效交易到状态数据库. 客户端可以通过 Peer 得知交易上链结果.

3 方案设计

在细粒度的键值状态分片下, 每个分片 Peer 的状态数据库只保存部分的键值状态数据, 区块链文件中的区块也只包含修改对应状态的交易. 设计 Fabric 的键值状态分片, 存在以下几个问题.

1) 如何计算一个键值状态数据应该分配到哪个分片, 即分片 Peer 如何判断一个交易所请求的数据存在于本地还是其他分片上.

2) 跨片交易在模拟执行阶段需要读取的数据不存在本地该如何处理.

3) 交易是一种分布式事务. Fabric 利用排序服务保证分布式事务的最终一致性. 在排序阶段对于跨片交易该如何处理从而保证一致性.

4) 跨片交易对状态的操作应保证原子性, 即跨片交易对各个分片数据的操作应都成功或都失败. 各分片间如何对跨片交易的验证结果达成一致.

5) 细粒度分片导致的一个显然结果是交易跨片的概率增大. 对跨片交易达成一致的代价会导致性能下降, 该如何缓解这个缺陷.

本节后续部分对以上问题按顺序做出分析并提供解决方案.

3.1 状态数据的分片

在前文介绍过, Fabric 中的状态数据以键值对的形式进行存储. 本节讨论如何计算键值对数据的目标分片. 数据分片的方案应当满足以下两个目标.

定义 1. 平衡性: 不同分片之间的负载大小应该尽可能保持相近, 即分片负载维持平衡.

定义 2. 单调性: 如果增加新的分片参与状态分片, 应该保证已分配数据能够被映射到新的分片或者原来的分片, 而不是被映射到其他分片.

本架构采用跳跃一致性哈希算法进行键值数据的划分, 其满足平衡性和单调性. 文献 [24] 中的实验证明, 跳跃一致性哈希算法相比经典的一致性哈希算法性能更好, 缺点是新增或者删除分片只能从分片序列的尾部进行, 而不能从中间删除. 考虑到 Fabric 的 Peer 拥有备份节点, 一般不会出现某一分片的 Peer 集体崩溃的情况, 所以不会出现删除中间分片的情况, 分片架构可用性不受影响.

算法 1. 键值数据划分算法

输入: 数据键 *key*; 分片数量 *N*
输出: 目标分片序号 *X*

```
1  X←−1, j←0 ; // 初始化
2  设置随机种子 random.seed(key);
3  while (j<N) do
4     X←j;
5     r←random.next();
6     j←⌊(X+1)/r⌋;
7  return X;
```

键值数据划分算法的伪代码如算法 1 所示. 跳跃一致性哈希算通过线性同余算法 (LCG) 生成的伪随机数来决定 *key* 的映射关系. *key* 会被作为种子生成伪随机序列, 从而保证了给定一个哈希桶数 *N* 时, 用同一个 *key* 作为输入, 算法的输出是确定的. 通过概率分析使得 *j* 的增加跳步进行, 算法时间复杂度从 O(*N*) 优化至 O(ln(*N*)).

3.2 跨片交易的模拟执行

对于跨片交易, 可能出现智能合约执行过程中需要读的键不在本地状态库中的情况, 这时候需要进行跨片查询.

以图 3 为例, 客户端发送给 PeerB 合约执行请求, 调用智能合约中的函数 T. 函数 T 在模拟执行过程中会读键 A 和键 B 的数据, 通过算法 1 计算键值数据所属的分片. 键 B 的数据存在 PeerB 的本地状态数据库中, 所以可以直接读取; 键 A 的数据存在另一个分片中, 所以 PeerB 通过 RPC 请求获取 PeeA 状态数据库中键 A 的值和版本号. 最终 PeerB 汇总生成读集合. 对于写集合, 因为是模拟执行不会直接写入数据库, 则直接根据智能合约执行过程生成.

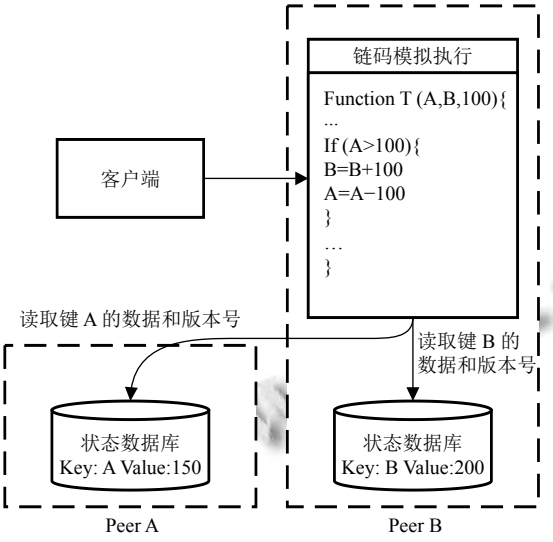

图 3 跨片交易模拟执行示例图

3.3 跨片交易的排序

区块链作为分布式系统实现最终一致性的原理是复制状态机. 区块内部交易顺序在区块打包时固定, 区块链共识算法保证区块按序接收, 从而实现全局交易的按序接收. 分散的节点们初始状态相同, 并经过同样顺序的接受交易并更新状态, 最终达成一致的状态. 在分片 Fabric 中, 也需要通过以上的机制, 来实现分布式交易的一致性.

根据交易是否会跨片, 可以将交易分为跨片交易和片内交易. 为了解决并发交易读写冲突的问题, 需要一个在所有交易上的偏序关系, 其满足以下两个特性.

定义 3. 片内可比较: 所有本分片涉及的交易之间有全序关系.

定义 4. 冲突跨片可比较: 可能发生冲突的跨片交易之间有全序关系. 也就是说, 多笔交易的读写数据如果有重复, 应该能互相比较确认先后顺序.

为了满足上述两个特性, 在排序组织中根据功能不同设置以下两种类型的 Orderer. 整个排序阶段的交易处理流程如图 4 所示.

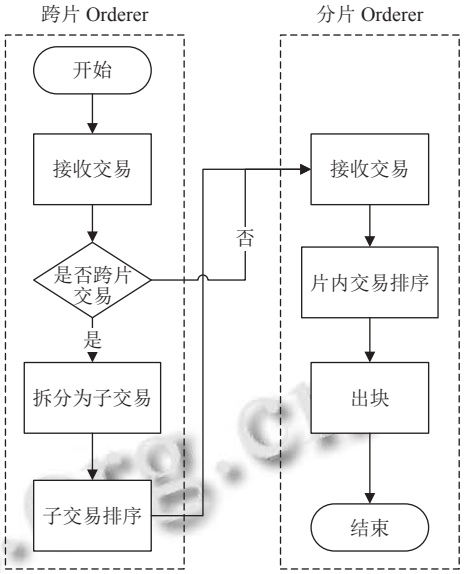

图 4 排序服务流程图

算法 2. 跨片交易拆分算法

输入: 跨片交易 *Tx*
输出: 子交易关联表 SubTxMap[分片号→子交易]

```
1. 初始化 SubTxSet;
2. for readKV in Tx.ReadSet do //遍历读集合
3.    shardNum←计算 readKV.Key 的分片号;
4.    if SubTxMap 未存在 shardNum 键 then
5.       SubTxMap[shardNum] ←SubTx;
6.    SubTxMap[shardNum].ReadSet 增加 readKV;
7. for writeKV in Tx.WriteSet do //遍历写集合
8.    重复相似步骤;
9. return SubTxMap;
```

跨片 Orderer: 客户端的交易首先发送给跨片

Orderer. 跨片 Orderer 通过扫描读写集判断交易是否跨片. 若为跨片交易, 则进行交易拆分. 如果一个交易的读写集数据涉及多个分片的状态数据, 那么这笔交易会被拆分成多笔子交易, 每笔子交易的读写集中只包含某一分片的状态数据. 跨片交易拆分算法见算法 2. 跨片 Orderer 拆分的子交易会在跨片 Orderer 排序后发送给各个分片. 以上保证了冲突跨片可比较.

分片 Orderer: 分片 Orderer 群运行 Raft 共识算法, 负责对涉及该分片的交易做排序并且打包出块. 总共会处理两种类型的交易, 一是片内交易, 二是跨片交易的子交易. 对于收到的片内交易, 按照时间先后进行排序; 对于收到的跨片子交易, 则会维持跨片 Orderer 的排序基础上, 将其插入排序缓存队列. 这样打包到区块里的交易就同时满足了片内可比较和冲突跨片可比较.

3.4 跨片交易的验证

在 Peer 收到交易后, 需要对交易进行验证, 验证后有效的交易才会提交到账本. 跨分片交易的验证提交, 其本质是分布式事务. 本文参考两阶段提交的工作模式, 设计了跨片交易的验证, 其流程参考图 5.

首先, 每个分片会将子交易的读集合中键的版本号与数据库中对应键的版本号进行比对, 如果所有键的版本号相同, 则意味该分片中原交易涉及的数据状态与模拟执行阶段时保持一致, 验证标记为 true, 否则标记为 false.

然后, 各个分片的 Peer 会将子交易的验证结果提交给排序组织中通过 Raft 算法选举出的 Leader 节点, 由其搜集所有子交易的验证结果.

最后, Leader 节点确认交易结果, 将最终结果返回给各个分片, 分片将交易进行提交.

分片 Peer 的跨片子交易验证也设置有超时机制, 这是为了防止网络分区 (通信断连导致原有网络分割成多个子网络) 使得部分子交易结果未知, 导致交易验证阻塞. 有两种情况, 一是 Leader 搜集子交易结果超时, 二是分片 Peer 节点等待总结果超时. Leader 对于超时的子交易验证结果, 会标记为无效. 分片 Peer 节点等待交易验证超时, 则将子交易标记为超时状态加入队列并跳过, 后续涉及相同状态数据的交易的验证都会进入待队列等待. 当网络分区恢复后, 会查询实际结果, 重新标记交易状态并对相关数据进行重放. 实际中网络发生断连较短, 重放代价是可以接受的. 若断连时间较长, 意味着后续的区块都无法从排序组织接收到, 节点工作状态等同于崩溃. Fabric 有崩溃重启的追赶机制.

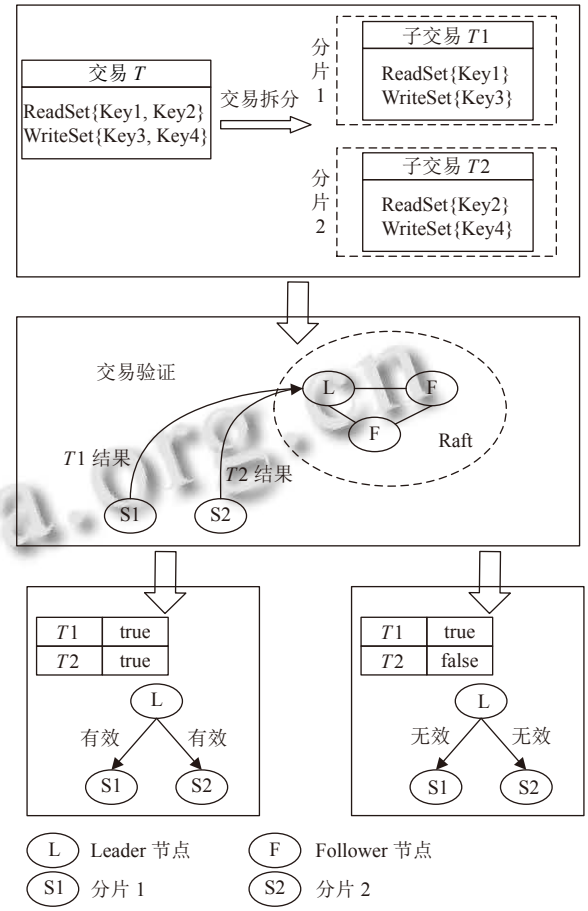

图 5 交易验证流程图

3.5 交易提案路由

交易提案路由问题即请求路由问题, 具体指当客户端想要发出请求时, 如何知道连接到哪个节点. 在传统数据库的分区 (分片) 技术中, 请求路由的客户端或者转发服务器都可以根据分区规则直接计算出要访问的数据所在. 但是在 Fabric 中, 因为一笔交易的模拟执行可能存在判断分支逻辑, 在执行过程中才能判断需要读哪些键, 所以此时交易请求无法得知应该发往哪个分片.

如果一笔交易提案被发送到一个错误的分片 Peer 上进行模拟执行, 那么可能进行多次跨片读请求, 执行 Peer 就需要阻塞计算, 造成性能损失.

在 Fabric 中, 客户端发送交易提案给 Peer 时, 会带上客户身份证书用来证明用户身份. 观察发现, 在 Fabric 中同一个用户在短时间内多次调用的交易涉及的键值数据往往是相同的. 基于该启发, 设计了如表 1 所示的路由表. 每个分片 Peer 都会维护这样一张路由

表, 路由表是一张哈希映射表, 键为交易提案的身份证书哈希值, 值内容为<是否转发, 转发目标>.

表 1 交易提案路由表

Hash (身份证书)	<是否转发, 转发目标>
0xAAAD	<是, 分片2>
0xBBBC	<是, 分片3>
0xCCCE	<否, NULL>

当一个交易提案被客户端随机发送到 Peer 之后, Peer 根据提案中的身份证书计算出键, 在交易提案路由表中检索对应的条目, 若存在且标记转发, 则会进行转发并告知客户端, 转发后根据转发目标反馈, 会再次更新路由表; 若不存在或者标记不转发, 则在本地进行模拟执行. 在这次模拟执行的过程中若发现对某一分片 X 的跨片读次数大于本地读次数, 那么在执行结束后, 对交易提案路由表进行更新, 标记为<是, 目标分片 X>.

表的容量根据启动配置设定上限. 若表满, 通过 LRU 算法进行替换.

3.6 总体流程

基于键值数据状态分片后的 Fabric, 交易处理流程与原 Fabric 对比如图 6 所示. 分片后 Fabric 的 Orderer 和 Peer 节点根据启动配置会分配到不同的分片中, 每个分片并行地处理不同交易. 分片前后每个阶段的对比如下.

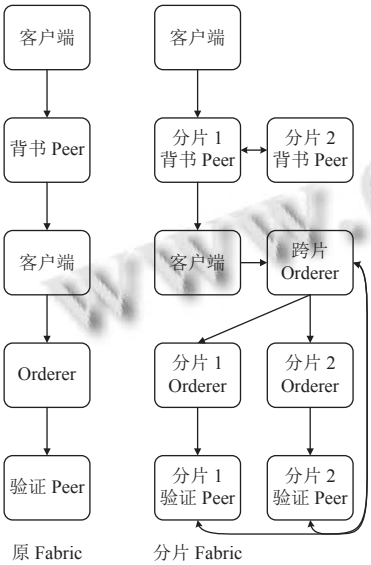

图 6 分片交易流程对比图

背书阶段: 与原 Fabric 相比, 基于键值状态分片的 Fabric 在交易执行背书阶段通过跨片读请求和交易路由转发两种方式, 分别解决跨片交易读数据不在本地的问题和多次跨片读代价过大的问题, 因此不同分片的背书 Peer 之间存在交互.

排序阶段: 通过增加跨片 Orderer 进行交易分割与冲突跨片交易的排序, 保证跨片交易的最终一致性. 分片后的排序阶段, 交易首先被发送给跨片 Orderer. 在跨片 Orderer, 跨片交易被分割排序后发送给各个分片 Orderer; 非跨片交易直接被转发给各个分片 Orderer. 分片 Orderer 为各个分片排序交易并打包出块, 各分片区块被发送给分片验证 Peer.

验证阶段: 分片后的验证 Peer 在验证跨片交易时, 采用类似两阶段提交的工作机制, 让跨片 Orderer 作为协调者收集并分发跨片交易的验证结果, 保证跨片交易的原子性.

表 2 实验环境配置

参数	值
CPU	Intel(R) Xeon(R) CPU E5-2660 v3
内存	256 GB
系统	Ubuntu 18.04
网络	1 Gb/s
工作负载	SmallBank
Docker版本	20.10.06

4 实验与分析

4.1 实验设计与配置

本节实现了 Fabric 的键值状态分片, 称为 KVFabric. 首先为了证明该方法能够有效提升 Fabric 的性能和解决热点访问问题, 设计实验对比 Fabirc、SmartFabric[9] (基于智能合约分片)、KVFabric 在均衡负载和热点负载下的性能. 然后针对细粒度分片会导致跨片交易占比上升的场景, 设计实验测试 KVFabric 性能在是否增加提案路由表时受跨分片交易占比的影响, 从而证明交易提案路由方案的有效性. 最后, 将公链的分片重分配和状态迁移方法[22] 移植到 SmartFabric 上, 称为 LB-Fabric, 设计实验对比 LB-Fabric 和 KVFabric 在改善智能合约热点访问问题上的表现, 证明 KVFabric 在解决这个问题上的优越性.

实验的配置如表 2 所示, 工作负载采用迷你银行智能合约 SmallBank, 可以进行创建账户和转账等操作. 使用 Docker 创建多个容器模拟多节点工作并防止节点间发生资源竞争.

4.2 评价指标

本文采用系统吞吐量和交易确认时延作为性能评价指标. 两个指标的定义如下.

定义 5. 吞吐量: 在某段时间内 T 内, 所有交易完成的数量 S_{Tx} 与该时间 T 的比值, 计算公式为:

$$N_{TPS}=\frac{S_{Tx}}{T} \tag{1}$$

定义 6. 时延: 从客户端发送交易提案到最终确认交易上链所需要的时间, 计算公式为:

$$T_{\text{latency}}=T_{\text{commit}}-T_{\text{send}} \tag{2}$$

吞吐量数据代表系统的处理负载能力上限, 时延数据代表用户感知的交易确认延迟. 实验过程中运行多轮, 对吞吐量数据计算平均值, 对时延数据采用中位数 (第五十百分位数).

4.3 性能对比实验

本实验测试 Fabric、SmartFabric、KVFabric 这 3 个系统的吞吐量和时延随着提供节点资源数量的变化. 原 Fabric 系统没有进行分片, 所以增加节点意味着组织内 Peer 节点数增加. SmartFabric、KVFabric 中则利用增加的节点建立新的分片. 实验中对 SmartFabric、KVFabric 设置了 10% 的跨片交易.

假设分片数量为 N, 在链上部署 N 个功能相同的智能合约, SmartFabric 中每个分片分配一个智能合约. 等概率调用 N 个智能合约的测试结果如图 7、图 8 所示. 模仿热点问题, 设某智能合约调用概率为 X, 其余智能合约调用概率合计为 Y, 以 X:Y=8:2 概率进行调用, 测试结果如图 9、图 10 所示.

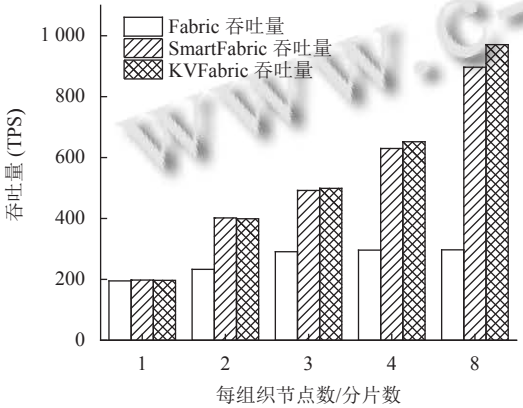

图 7 均衡负载测试吞吐量图

Fabric 因为在模拟执行节点实现了交易并行执行, 所以增加节点数可以增加交易吞吐量. 但因为在排序阶段和验证阶段交易都是顺序处理的, 吞吐量很快达到瓶颈. 因为节点各流程执行效率不受节点数影响, 故时延变化较小. 因为 Fabric 没有实现分片, 所以其在均衡负载测试和热点负载测试中表现一致. 无论在均衡负载或热点负载测试中, KVFabric 的吞吐量随着分片数量的增加大幅领先 Fabric, 时延因为跨片交易存在额外的处理流程而轻微增加. 在分片数为 8 时, 吞吐量提升 226.9%, 时延增加 18.4%.

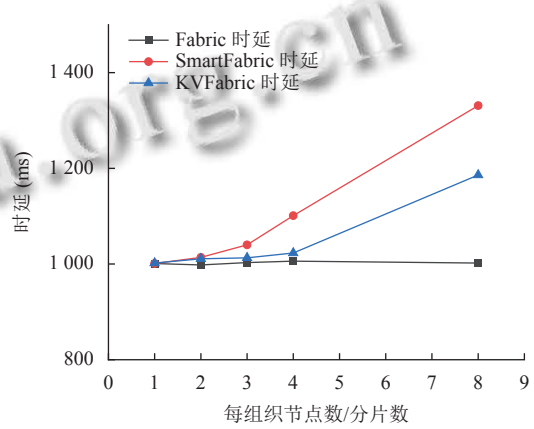

图 8 均衡负载测试时延图

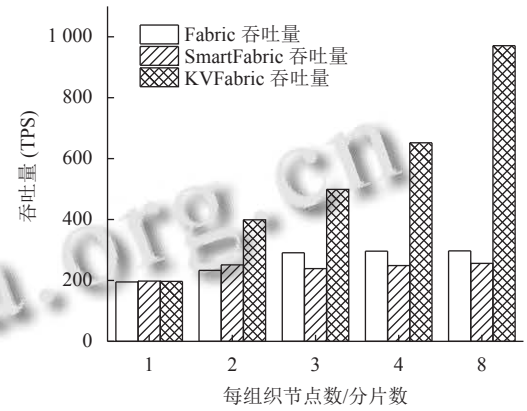

图 9 热点负载测试吞吐量图

在均衡负载测试中, SmartFabric 的吞吐量和时延效果都略差于 KVFabric. 因为在验证阶段随着分片数量的上升, SmartFabric 每个节点广播子验证信息的代价增长速度大于 KVFabric 的两阶段提交代价.

在热点负载测试中, KVFabric 的细粒度分片使得各个分片负载均衡, SmartFabric 大量的请求都发送到一个分片上, 造成单分片过载, 其余分片过闲, 故前者性能表现优于后者. SmartFabric 时延维持在 3 500 ms 左右波动并轻微上升的原因是测试时延时的交易发送速度不变, 且热点分片每秒处理的交易数量能力不变,

所以在不同分片数下, 热点分片的阻塞任务数差不多, 交易时延接近. KVFabric 因为有跨片交易的排序和两阶段提交, 时延略大于 Fabric.

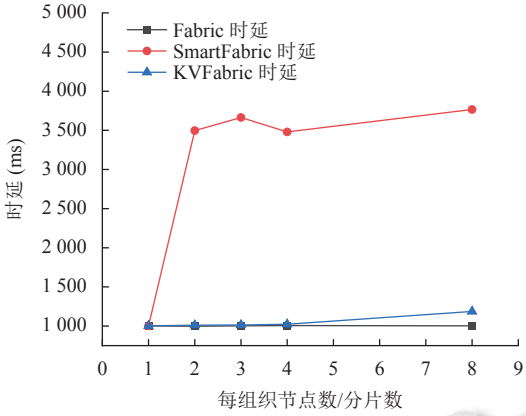

图 10 热点负载测试时延图

综上所述, 随着节点资源增多, KVFabric 的吞吐量增长远大于 Fabric, 时延略高于 Fabric. KVFabric 在均衡负载下性能略优于 SmartFabric, 在热点负载下性能大幅优于 SmartFabric.

4.4 交易提案路由表实验

本实验测试交易提案路由表方案是否能有效改善跨片交易占比上升对 KVFabric 带来的影响.

实验设置了 100 份客户身份证书, 发送交易会按顺序取用不同的身份证书. 交易将被等概率地发送到不同分片中. 为了模拟跨片交易在执行阶段多次跨片读的行为, 部署的转账智能合约修改为读取多个账户的金额为某一个账户转账. 跨片交易占比表示客户端发送的交易是跨片交易的概率. 实验测试了在是否增加交易提案路由表的情况下, 吞吐量和时延随跨片交易占比的变化. 性能测试结果如图 11、图 12 所示.

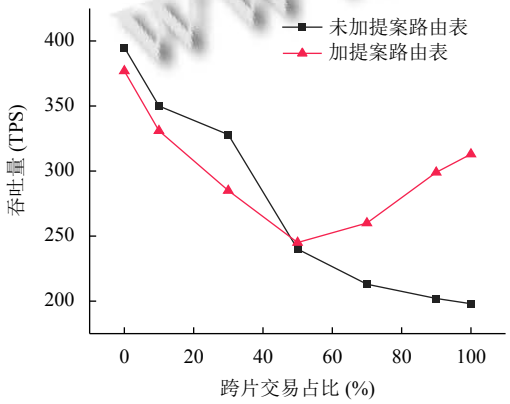

图 11 提案路由效果测试吞吐量图

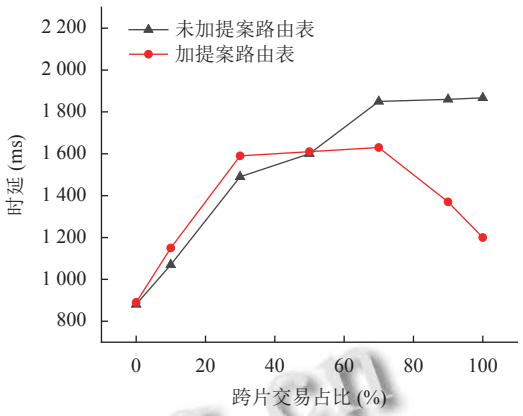

图 12 提案路由效果测试时延图

从图 12 可以观察到, 在跨片交易占比低的时候, 因为路由表的更新开销和命中率较低, 损失大于收益, 所以加路由表的时延会略微高于未加路由表. 当跨片交易占比超过 50% 之后, 因为跨片交易的占比增多提高了路由表的命中率, 所以时延会逐渐下降, 此时加路由表方案优于未加路由表.

从图 11 的吞吐量变化中也可以得到相同的结论. 100% 跨片占比下吞吐量无法回到原有水平的原因是因为转发交易存在损耗.

综上所述, 当跨片交易占比较高, 并且能保证命中率 (启发有效) 的情况下, 交易提案路由表能降低分片 Fabric 的时延, 提升其吞吐量.

4.5 热点访问改善效果对比实验

为了模仿现实中热点智能合约发生变化的情况. 本实验在热点负载测试的基础上, 每隔 30 s 切换热点智能合约, 即周期性调整对不同智能合约的访问频率. 测试 KVFabric 和 LB-Fabric[22] 中交易吞吐量随时间的变化, 测试结果如图 13 所示.

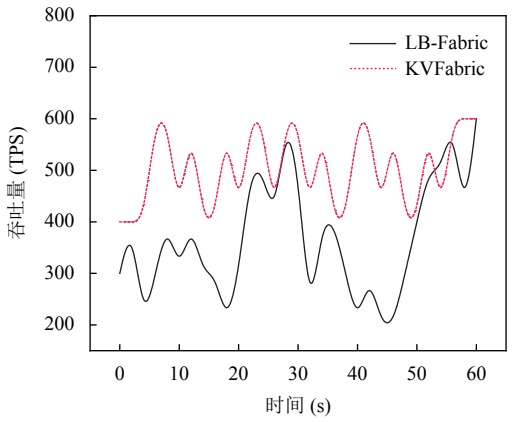

图 13 热点访问改善效果对比图

LB-Fabric 需要根据历史预测各个分片下一周期的交易量, 然后计算如何迁移智能合约能够尽可能保证分片间的负载均衡. 迁移智能合约会造成交易的阻塞, 并且打破了智能合约与分片的绑定关系, 导致客户端发送交易和处理跨片交易都需要向特定的角色查询目标分片. 以上计算和缺陷都可能影响系统吞吐量, 并且热点智能合约的切换会导致历史预测的失效. 实验结果证实了上述分析, LB-Fabric 的吞吐量都会周期性地下降, 然后再快速提升, 但最好的情况也无法超过 KVFabric.

5 总结与展望

本文针对 HyperLedger Fabric 现有分片方法存在热点访问的问题, 提出了细粒度的键值状态分片方案. 针对细粒度分片导致跨片交易增多的问题, 提出了交易提案路由表的解决方案. 通过实验证明, 以上方案能够有效地解决问题. 但是本文方案仍旧存在缺陷, 例如如何解决随着分片数上升性能提升效果下降的问题, 以及交易路由表失效情况下该如何优化高跨片交易占比下的交易处理等问题. 相关研究有待进一步展开.

参考文献

1 Ali O, Jaradat A, Kulakli A, *et al*. A comparative study: Blockchain technology utilization benefits, challenges and functionalities. IEEE Access, 2021, 9: 12730–12749. [doi: 10.1109/ACCESS.2021.3050241]

2 Zhang L, Xie YP, Zheng Y, *et al*. The challenges and countermeasures of blockchain in finance and economics. Systems Research and Behavioral Science, 2020, 37(4): 691–698. [doi: 10.1002/sres.2710]

3 Bhaskar P, Tiwari CK, Joshi A. Blockchain in education management: Present and future applications. Interactive Technology and Smart Education, 2021, 18(1): 1–17. [doi: 10.1108/ITSE-07-2020-0102]

4 Wang Q, Su M. Integrating blockchain technology into the energy sector—From theory of blockchain to research and application of energy blockchain. Computer Science Review, 2020, 37: 100275. [doi: 10.1016/j.cosrev.2020.100275]

5 Androulaki E, Barger A, Bortnikov V, *et al*. Hyperledger fabric: A distributed operating system for permissioned blockchains. Proceedings of the 13th EuroSys Conference. Porto: ACM, 2018. 30. [doi: 10.1145/3190508.3190538]

6 Li DC, Wong WE, Guo JC. A survey on blockchain for enterprise using HyperLedger Fabric and composer. Proceedings of the 6th International Conference on Dependable Systems and Their Applications. Harbin: IEEE, 2020. 71–80. [doi: 10.1109/DSA.2019.00017]

7 Kuzlu M, Pipattanasomporn M, Gurses L, *et al*. Performance analysis of a HyperLedger Fabric blockchain framework: Throughput, latency and scalability. Proceedings of the 2019 IEEE International Conference on Blockchain. Atlanta: IEEE, 2019. 536–540. [doi: 10.1109/Blockchain.2019.00003]

8 Wang CH, Chu XW. Performance characterization and bottleneck analysis of HyperLedger Fabric. Proceedings of the 40th International Conference on Distributed Computing Systems. Singapore: IEEE, 2020. 1281–1286. [doi: 10.1109/ICDCS47774.2020.00165]

9 Thakkar P, Natarajan S. Scaling blockchains using pipelined execution and sparse peers. Proceedings of the 2021 ACM Symposium on Cloud Computing. Seattle: ACM, 2021. 489–502. [doi: 10.1145/3472883.3486975]

10 Zhou QH, Huang HW, Zheng ZB, *et al*. Solutions to scalability of blockchain: A survey. IEEE Access, 2020, 8: 16440–16455. [doi: 10.1109/ACCESS.2020.2967218]

11 Wang JP, Wang H. Monoxide: Scale out blockchain with asynchronous consensus zones. Proceedings of the 16th Symposium on Networked Systems Design and Implementation. Boston: USENIX Association, 2019. 95–112.

12 Amiri MJ, Agrawal D, El Abbadi A. SharPer: Sharding permissioned blockchains over network clusters. Proceedings of the 2021 International Conference on Management of Data. Xi'an: ACM, 2021. 76–88. [doi: 10.1145/3448016.3452807]

13 Zheng PL, Xu QQ, Zheng ZB, *et al*. Meepo: Sharded consortium blockchain. Proceedings of the 37th IEEE International Conference on Data Engineering. Chania: IEEE, 2021. 1847–1852. [doi: 10.1109/ICDE51399.2021.00165]

14 Yuan P, Zheng K, Xiong X, *et al*. Performance modeling and analysis of a HyperLedger-based system using GSPN. Computer Communications, 2020, 153: 117–124. [doi: 10.1016/j.comcom.2020.01.073]

15 Gorenflo C, Lee S, Golab L, *et al*. FastFabric: Scaling Hyperledger Fabric to 20000 transactions per second. International Journal of Network Management, 2020, 30(5): e2099. [doi: 10.1002/nem.2099]

16 Sharma A, Schuhknecht FM, Agrawal D, *et al*. Blurring the lines between blockchains and database systems: The case of

HyperLedger Fabric. Proceedings of the 2019 International Conference on Management of Data. Amsterdam: ACM, 2019. 105–122. [doi: 10.1145/3299869.3319883]

17 Ruan PC, Loghin D, Ta QT, *et al*. A transactional perspective on execute-order-validate blockchains. Proceedings of the 2020 International Conference on Management of Data. Portland: ACM, 2020. 543–557. [doi: 10.1145/3318464.3389693]

18 Hang L, Kim B, Kim D. A transaction traffic control approach based on fuzzy logic to improve HyperLedger Fabric performance. Wireless Communications and Mobile Computing, 2022, 2022: 2032165. [doi: 10.1155/2022/2032165]

19 Androulaki E, Cachin C, De Caro A, *et al*. Channels: Horizontal scaling and confidentiality on permissioned blockchains. Proceedings of the 23rd European Symposium on Research in Computer Security. Barcelona: Springer, 2018. 111–131. [doi: 10.1007/978-3-319-99073-6_6]

20 Hong ZC, Guo S, Li P, *et al*. Pyramid: A layered sharding blockchain system. Proceedings of the 2021 Conference on Computer Communications. Vancouver: IEEE, 2021. 1–10. [doi: 10.1109/INFOCOM42981.2021.9488747]

21 Huang HW, Peng XW, Zhan JZ, *et al*. BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding. Proceedings of the 2022 Conference on Computer Communications. London: IEEE, 2022. 1968–1977. [doi: 10.1109/INFOCOM48880.2022.9796859]

22 Li MZ, Wang W, Zhang J. LB-chain: Load-balanced and low-latency blockchain sharding via account migration. IEEE Transactions on Parallel and Distributed Systems, 2023. [doi: 10.1109/TPDS.2023.3238343]

23 Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference. Philadelphia: USENIX Association, 2014. 305–319.

24 Lamping J, Veach E. A fast, minimal memory, consistent hash algorithm. arXiv:1406.2294, 2014.

(校对责编: 牛欣悦)