基于缓存机制的 Hyperledger Fabric 并发冲突检测方法*

王盛姣，董建亮，熊　航，李　京
（中国科学技术大学 计算机科学与技术学院，安徽 合肥 230026）

摘　要： Hyperledger Fabric（Fabric）是一个受关注度较高的许可链平台，具有高度模块化、可定制化和可插拔的特点。针对 Fabric 在高并发的场景下会出现并发冲突导致交易无效的问题，提出一种冲突检测与处理的方法，即利用缓存交易写集的方式在执行阶段检测交易是否冲突，最大程度减小冲突交易在 Fabric 系统的资源消耗。实验结果表明，在具有冲突交易的场景下，提出的方法能降低平均交易时延，提高系统有效交易吞吐量；并且在没有冲突交易的情况下，不会明显降低性能。

关键词： Hyperledger Fabric；区块链；并发冲突；缓存机制
中图分类号： TP311.13　　**文献标识码：** A　　**DOI：** 10.19358/j.issn.2096-5133.2022.06.015
引用格式： 王盛姣，董建亮，熊航，等. 基于缓存机制的 Hyperledger Fabric 并发冲突检测方法[J].信息技术与网络安全，2022，41(6)：94-101，108.

Hyperledger fabric concurrency conflict detection method based on caching mechanism

Wang Shengjiao，Dong Jianliang，Xiong Hang，Li Jing
(School of Computer Science and Technology，University of Science and Technology of China，Hefei 230026，China)

Abstract： Hyperledger Fabric (Fabric) is a popular permissioned chain platform that is highly modular，customizable and pluggable. To solve the problem that transactions are marked invalid due to concurrent conflicts in Fabric with high concurrency，we propose a method of conflict detection and processing: caching transactions write set to detect whether the transaction is conflicting during the execute stage，and minimize the resource consumption of conflicting transactions in Fabric. The experimental results show that the proposed method can reduce the average transaction delay and improve the effective transaction throughput of the system in the case of conflicting transactions. And there is no significant performance degradation without conflicting transactions.

Key words： Hyperledger Fabric；blockchain；concurrency conflict；caching mechanism

0 引言

随着比特币[1]热潮的出现，其背后的区块链技术广受关注。区块链是一种分布式账本技术，具有去中心化、数据可信、不可篡改和可溯源等优点。区块链构建了点对点对等网络，由网络中的对等节点集体维护账本，运用数据加密和区块+链式数据结构来存储验证数据，通过共识机制产生新区块，利用以太坊虚拟机[2]或 docker 容器等技术提供对智能合约的支持，具有可编程功能。

随着研究和发展的深入，区块链已经有了较多实际应用，如医疗数据安全共享[3]、供应链管理系统[4]、物联网访问控制[5]、数字版权[6]等。

区块链根据节点是否可以自由加入分为非许可链和许可链。Hyperledger Fabric（Fabric）[7]是一个受关注度较高的许可链平台，具有开源、高度模块化、可定制、可插拔的特点。当前大多数的区块链采用排序-执行（Order-Execute，OE）交易处理模型，系统串行处理交易使得性能受到限制。因此，Fabric 提出了执行-排序-验证（Execute-Order-Validate，EOV）的交易处理模型。在执行阶段，客户端发送交

* 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）

易请求到相应节点，节点响应请求将带有处理结果的交易返回给客户端。在排序阶段，Orderer 节点将客户端发来的交易按序打包成区块，并广播给节点。在验证阶段，节点接收到区块后串行化验证交易并更新账本。Fabric 通过背书策略去配置不同交易请求所需要的节点数目，实现执行阶段交易的并发处理。除此之外，Fabric 还引入组织的概念，组织节点之间并发地处理发送给该组织的交易，提高了系统的并发能力。

已有研究结果[8]表明，Fabric 性能显著优于比特币和以太坊。Fabric 内部集成了身份管理、业务数据分离、隐私数据保护等功能，使其受到广泛的使用，其中大部分的企业级区块链项目在 Fabric 网络上进行[9]。因此，Fabric 在学术和工业上都具有研究价值。

并发性为 Fabric 带来优越性能的同时也带来了一些问题。当并发的交易需要修改相同的数据时，会出现交易并发冲突，而 Fabric 上交易之间互相隔离，执行和排序阶段都不会检测和处理冲突，所以验证阶段中部分冲突交易会被标记为无效。无效交易的验证占用了系统计算和存储的资源，导致性能下降。已有工作表明[10]，部分并发情况下交易的成功率只有不到 1/4。因此想要提升 Fabric 的性能，交易并发冲突成为亟待解决的问题。

本文针对交易并发冲突进行 Fabric 性能优化研究，提出一种基于 Fabric 实现的冲突检测与处理的方法，即 FabricDT。将冲突的检测提前到背书阶段，以减少后续排序和验证环节的系统开销，从而提高系统的吞吐量，通过实验验证了其有效性。

1 相关工作

已经有不少研究学者提出了解决 Fabric 并发冲突的方法，从解决思路上主要可以分为两类。

第一类方式是在客户端消解交易冲突。Zhang 等人[11]提出在客户端中添加一个缓存队列用来缓存从背书节点返回的交易，判断交易间的冲突性，将冲突交易锁住，并监听区块更新事件。当有区块更新之后，会检测缓存中锁住的交易，若锁因为区块更新被释放，则由缓存重新将此交易作为请求发出。但是，这只能解决单个客户端的冲突交易，无法处理多个客户端间的冲突。Xu 等人[12]提出客户端对交易预处理，更改冲突交易对应的数据库索引，并在交易提交之后将临时索引归并到原索引，几乎可以成功处理所有的冲突交易。但是，实现节点统一生成和归并临时索引需要一个可信的分布式锁服务来同步访问，会耗费大量的网络资源。

第二类方式是在排序阶段，分析交易之间冲突依赖关系，并通过丢弃部分交易和重排序的方式去得到一个无冲突的交易顺序并打包成区块。Sharma 等人[10]最先以此为思路提出了 Fabric++，它在排序阶段首先分析交易的冲突关系建立依赖图，计算图中的强连通分量以此得到不可解的冲突，然后丢弃强连通分量中度最多的交易直到依赖图中不存在闭环，最后利用拓扑排序生成一个不冲突的交易顺序并打包成区块。之后 Ruan 等人[13]提出 Fabric-Sharp，相较于 Fabric++对冲突的处理粒度更细，考虑了冲突类型和跨块冲突。此类方法存在一定问题，即在排序阶段集中分析冲突关系，使用图类算法求解，当交易数目较多时容易造成性能瓶颈。

此外，有一些方法针对 Fabric 中并发冲突进行优化。Gorenflo 等人[14]提出 Execute－Orderer－Execute 的交易流程，他认为一笔交易如果只是验证阶段因为冲突问题被标记为无效，这笔交易整个执行流程上并没有任何信任问题，因此可以在验证阶段发现冲突时，由节点本地执行交易得到最新的结果。然而，该方法忽视了 Fabric 构建的许可链中仍然存在信任问题，不同节点可能会恶意写入错误数据导致账本数据出错。

Nasirifard 等人[15]提出使用特定的无冲突复制数据类型（Conflict－free Replicated Data Type，CRDT）[16]去解决冲突问题。将交易执行的读写集合转换为特定的数据对象和相关操作，而这些数据对象和操作构成代数半格，满足可交换的特点，在将转换之后的数据对象逐步更新到账本后，就不存在冲突的问题。但这个方法也有一个缺陷，即对应不同的交易需要单独设计与之对应的 CRDT 数据结构，缺乏通用性。

综上所述，现有的并发冲突解决方案具有明显的局限性，存在检测能力弱、通信代价大、时间复杂度较高和缺乏通用性等问题。

2 Fabric 介绍

2.1 Fabric 架构

（1）Fabric 账本。Fabric 的账本包含两部分：链式结构和键值数据库。链式结构即传统意义上的区块链账本，记录了所有历史交易。客户端发送的交易在执行过程中经常需要读取链上数据，为了更快

地完成交易，Fabric 内置了一个键值数据库，用来记录当前链上所有数据和最新状态。键值数据库的每一条数据由键和版本号组成，其中键表示了具体的数据对象，版本号记录了数据当前值和对应交易所在块号和交易序号。以表 1 第一行为例，账户 A 中的余额为 100，将其余额修改为 100 的交易是第 23 个区块中的第 32 条交易。

表 1 键值数据库

账户号	版本号 <余额，块号，交易号>
A	100，23，32
B	43，23，14
C	91，19，4

（2）链码。链码是 Fabric 中智能合约的具体实现，包含了应用的处理逻辑，支持 Golang、Node.js、Java 语言。Fabric 提供接口使得客户端能够以交易的形式调用链码与账本交互。

（3）Peer。Peer 节点是维护账本和链码，响应客户端交易请求，验证区块内容的基本单位。当接收到客户端的交易请求时，执行对应的链码，访问账本数据，并返回执行结果给客户端（也称为背书）；当接收到新区块时，会结合当前账本验证区块内容，以确保新区块对账本的更改合理合法。另外，Peer 节点按照组织进行划分，一个 Peer 节点只能属于一个组织，内部包含多个 Peer 节点，不同组织之间的节点互不信任，同一个组织的节点相互信任。

（4）Orderer 节点。Orderer 节点主要功能是产生区块，它接收客户端发送的交易，并将其打包成区块发送给 Peer 节点。

（5）背书策略。Fabric 对每个链码都会设置背书策略，它声明了一笔合法的交易需要哪些组织进行背书。以“3outof(Org1,Org2,Org3,Org4,Org5)”为例，调用该链码的交易必须得到 Org1、Org2、Org3、Org4 和 Org5 五个组织中三个以上的背书结果。因此客户端会将交易请求至少发送给三个组织中进行背书；当得到足够背书时，会对比执行结果是否一致，如不一致则说明网络中可能存在恶意的节点企图非法修改链上数据。如果恶意组织的数量小于三个，客户端通过对比交易结果就能检测到异常，从而实现了 3/5 容错。一个合适的背书策略既能达到预想的容错，又减少了所需要的背书节点数量从而提升了整体性能。

（6）客户端。客户端是用户与链交互的工具。用户通过客户端发起交易请求，指定需要调用的链码和参数，并根据对应的背书策略将请求发送给对应的组织节点。

2.2 执行–排序–验证的交易处理流程

以图 1 为例，来具体说明 Fabric 系统的交易处理流程。

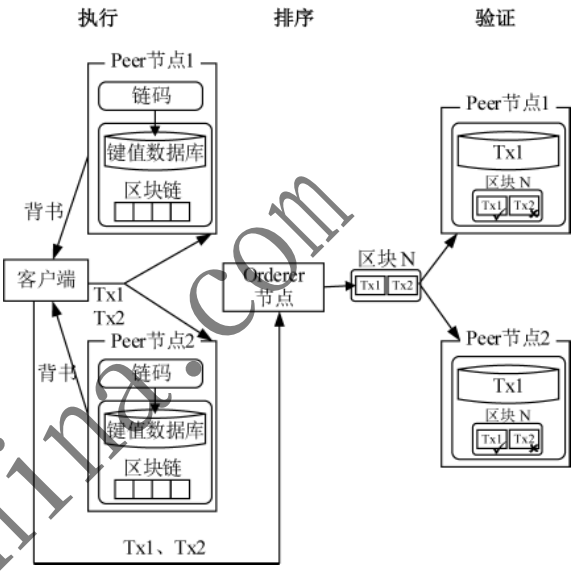


图 1 执行–排序–验证流程

（1）执行阶段。客户端按照背书策略将交易请求发送到不同的背书节点。节点在接收到请求之后，会以本地账本为数据库执行相关链码。在链码的执行过程中，将其需要读取和修改的数据以读写集的形式保存，即执行结果，并附上节点签名作为背书返回给客户端。

假设客户端发起交易请求“A 账户向 B 账户转账 15”的交易请求，某一背书节点账本的键值数据库如表 1 所示，其背书结果如表 2 所示。节点背书的执行结果中，读集包含了读取数据 A 和 B 的版本号，写集保存了需要修改的值，并在末尾对结果进行签名。

需要注意的是执行阶段链码的执行结果并没有更改 Fabric 账本。客户端在接收到结果后，会进行签名验证和背书策略的验证：背书数量是否满足背书策略，不同节点的执行结果中的读写集是否一致。

（2）排序阶段。当客户端验证交易合法之后，会将其发送给 Orderer 节点。当达到出块条件时，Orderer 节点会将交易打包成区块，并发送给不同组

表 2 背书结果

条目	值
读集	<数据 A:版本号< 23, 32> >, <数据 B:版本号<23, 14> >
写集	<数据 A:值<85> >, <数据 B:值<58> >
节点签名	ASFGSSDF

织的 Peer 节点。

(3)验证阶段。Peer 节点在接收到区块后会根据区块更新账本。

(4)修改过程。验证签名、背书策略、读集版本号是否与当前键值数据库中的一致,若都一致则按照写集更改键值数据库,并补全对应的块号和交易号;否则将交易标记为无效。

2.3 Fabric 交易冲突

Fabric 系统中交易可以并发执行,当多个交易读写相同的数据时,会导致交易并发冲突。这些冲突交易在验证阶段标记无效,降低了 Fabric 性能。冲突交易定义如下:

定义 1 冲突交易:客户端发送两笔交易请求 Tx1 和 Tx2,背书节点执行后分别得到交易读写集:ReadSet (Tx1)、WriteSet (Tx1)、ReadSet (Tx2)、WriteSet (Tx2)。假设 Tx1 和 Tx2 并发执行,且以下条件任意为真时,Tx1 和 Tx2 互相冲突。当 Tx1 先于 Tx2 验证,且满足条件(1)或者(3)时,Tx2 为冲突交易。

(1) $WriteSet(Tx1) \cap ReadSet(Tx2) \neq \emptyset$

(2) $ReadSet(Tx1) \cap WriteSet(Tx2) \neq \emptyset$

(3) $WriteSet(Tx1) \cap WriteSet(Tx2) \neq \emptyset$

以图 2 为例说明交易冲突的影响。

假设目前网络上所有 Peer 节点的账本都一致,区块数量为 25,键值数据库中的内容如表 1 所示,此时两个客户端发起了两笔交易请求,内容为 Tx1:"从 A 转帐 15 到 B",Tx2:"从 A 转帐 5 到 C",并且两笔交易都得到了合法的结果,那么 Tx1 的执行结果为读集<A: 版本号<100,23,32>, B:版本号<43,23,13> >,写集为<A:值<85>, B:值<58> >,Tx2 执行结果的读集为<A: 版本号<100,23,32>, C:版本号<91,19,4> >,写集为<A:值<95>, B:值<96> >。

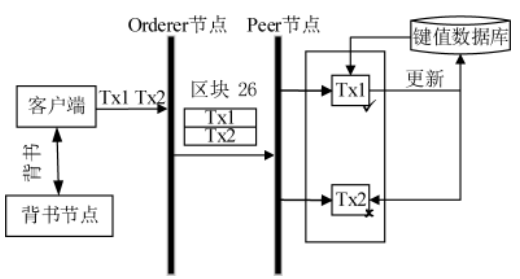


图 2 冲突交易实例

客户端得到结果后,验证通过并发送给 Orderer 节点,Orderer 节点将这两笔交易打包为成第 26 个区块,块内序号分别为 1 和 2,然后 Orderer 节点将区块发送给 Peer 节点,进入验证阶段。

验证阶段,Peer 节点首先验证到第一笔交易,在验证读写集时,发现读集与当前账本状态一致,验证通过,并更改键值数据库到表 3 形式,然后验证第二笔交易,在验证读写集时发现读集中的<A:版本号<100,23,32> >与当前键值数据库中内容不相同,则将第二笔交易标记为无效,然后验证后续的交易。

表 3 键值数据库

账户号	版本号 <余额, 块号, 交易号>
A	85, 26, 1
B	58, 26, 1
C	91, 19, 4

3 基于缓存机制的 Hyperledger Fabric 并发冲突检测

Fabric 中交易并发执行可能出现交易冲突,因执行阶段和排序阶段没有任何冲突检测机制,冲突交易只能在验证阶段被判断为无效,从而浪费了系统的计算资源等,影响了系统的性能。

本文针对交易冲突提出了一个冲突检测方案,在交易流程的执行阶段,Peer 节点本地维护一个缓存去记录还未提交的交易所访问的链上数据,并以此对接收到的背书请求进行冲突检测和处理。

3.1 冲突检测模块

为了尽早地检测到交易之间的冲突,本文在 Peer 节点上新增一个未验证交易缓存用来记录交易情况,并基于此进行冲突检测。以图 3 为例,来说明如何进行缓存的更新,和交易冲突的检测与处理。

3.1.1 未验证交易缓存的构建与更新

当一个交易请求通过执行阶段时,客户端除了会将交易发送给 Orderer 节点,同样也会发送给之前的背书节点。背书节点在接收到交易时读取其写集,以其键值(key)、交易 ID(txID)和时间戳(time)组

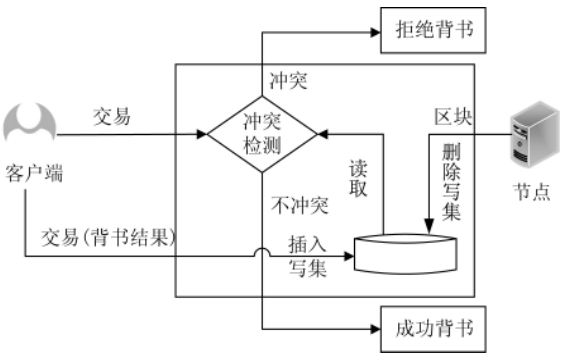


图 3 总体设计

成键值对(key,<txID,time>)存入缓存。

当 Peer 节点接收到新区块进行验证时,若区块中存在与缓存中相同的交易 ID,那么 Peer 节点会删除缓存中对应 ID 的所有记录。

另外,可能存在网络原因导致客户端发送给 Orderer 节点的交易出现丢失、延迟的情况,这将使得未验证交易缓存无法及时删除对应交易 ID 的记录,这部分键永远留在缓存中,会影响后续交易的冲突检测。为了解决这一问题,Peer 节点内部设置一个时间阈值,对缓存中超过阈值的数据清除。

3.1.2 交易冲突的检测与处理

Fabric 冲突是因为不同交易之间要读取和修改数据键相同,而由上小节可知,未验证交易缓存中记录了网络中通过执行阶段但还未写入账本的交易要修改的数据键,如果后续有客户端发送交易请求,需要读取缓存中的键,在最终的验证阶段,这一笔交易极大概率是要与之前的交易相冲突。

因此,当 Peer 节点接收到交易请求时,会获取请求所读取的数据键,如果该键存在于缓存中,则返回背书失败的结果,否则对其进行背书。

3.2 基于冲突检测的三阶段交易流程

在新增冲突检测处理模块之后,其三阶段交易流程可由图 4 表示,具体如下。

(1)执行阶段。客户端根据背书策略将交易请求发送到部分背书节点。背书节点根据请求调用相关链码,得到读写集,然后本地进行冲突检测,若不冲突,则将背书结果返回给客户端,否则返回背书失败。

(2)排序阶段。客户端在接收到背书结果之后,会将交易发送给 Orderer 节点和之前的背书节点,背书节点会对未验证交易缓存进行更新,Orderer 节点则会打包交易成区块,并发送给 Peer 节点。

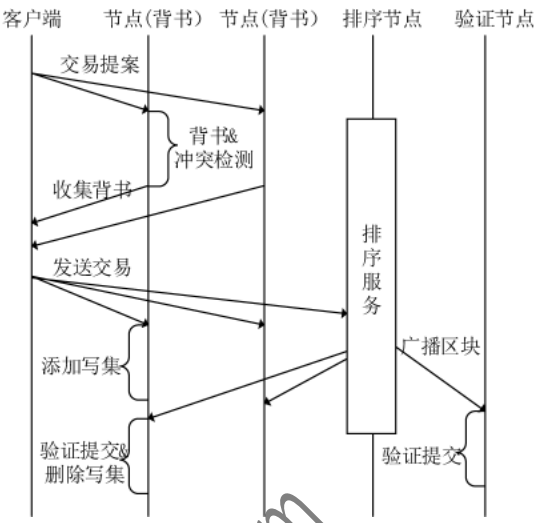


图 4 交易处理流程

(3)验证阶段。Peer 节点在接收到区块之后,会验证区块更新账本,同时遍历区块内所有交易写集并更新未验证交易缓存。

4 实验

为了证明该方法的有效性,本节实现了 FabricDT 并设计了两个实验。第一个实验是验证在无冲突交易的情况,FabricDT 相对于 Fabric 并没有明显的性能下降;第二个实验是对比现有先进的算法 Fabric++,以证明存在并发冲突交易的情况下 FabricDT 性能的优越性。

实验的硬件配置为 Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz,内存为 256 GB,通过 1 Gb/s 以太网交换机连接,操作系统版本为 Ubuntu 18.04,Golang 版本为 go1.16.10。Fabric 网络设置为一个 Orderer 节点,两个组织,分别为 Org1、Org2,每个组织都包含两个 Peer 节点,其中一个 Peer 节点安装链码设置为背书节点,另外有一个客户端用来发送交易。Orderer 节点、Peer 节点和客户端分别部署在不同的服务器上,出块设置如表 4 所示。

本节设计了一个模拟银行系统业务的链码,包含 despoit、transfer、query 三种交易类型。despoit 用来向账户增加一定金额,transfer 用来将一定金额从一个账户转移到另一个账户(这两种写交易会修改区

表 4 出块设置

参数	值
区块最大交易数	100
最长时间间隔/s	2
区块大小/MB	2

块链账本),query 用于查询账户余额。链码的背书策略设置为“And(Org1,Org2)”。实验按比例生成读交易和写交易,读交易的比例为 P_q,写交易中 despoit 和 transfer 随机生成。所有实验都会预先创建 10 000 个账户,生成随机数初始化账户余额。此外,实验设置通过 Zipfian[17]分布生成交易访问的账户,以构造出冲突交易。冲突程度随分布偏度的增加而增大[18],当分布偏度为 0 时均匀地随机访问所有账户(如图 5 所示)。

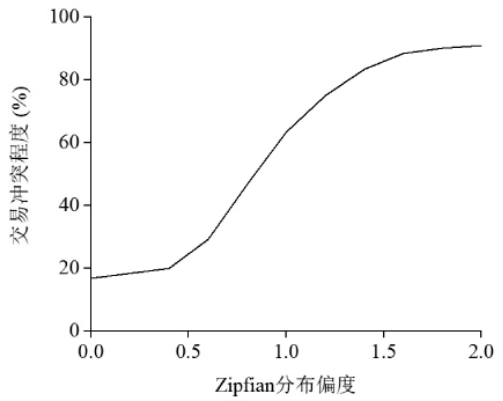


图 5 不同 Zipfian 分布偏度下交易冲突程度

为了衡量 Fabric 系统性能,本节实验基于测试结果统计了如下指标:

(1)吞吐量:单位时间内客户端成功验证提交的交易数量。

(2)平均交易时延:所有交易从发起到最后提交所需的时间的平均值。

(3)成功率:成功验证提交的交易数量占所有交易的比例。

4.1 无冲突时性能分析

为了验证 FabricDT 并不会带来巨大的额外开销,以造成性能下降,首先在无交易冲突的情况下去对比 FabricDT 和 Fabric。在发送速率为 50 tps、100 tps、150 tps、200 tps、250 tps、300 tps 时,持续发送 50s 交易,吞吐量和平均交易时延结果(如图 6 所示)。

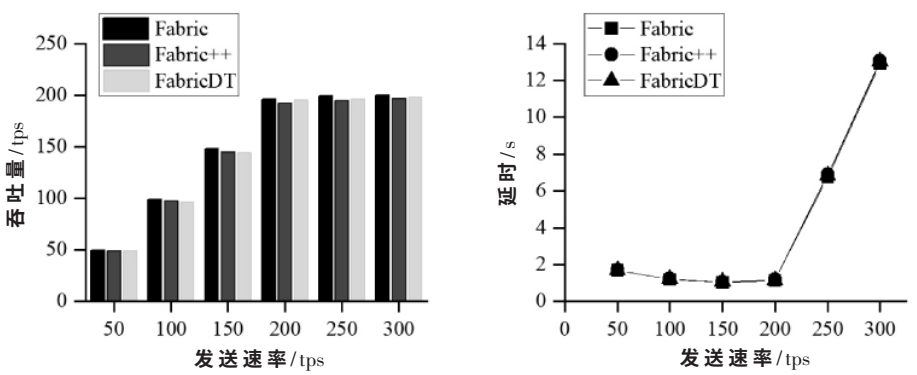
从图中可以看出,当发送速率从 50 tps 增加到 200 tps 时,系统的吞吐量增加,而平均交易时延有所下降,在这个阶段,负载并未饱和,背书节点能够响应客户端请求,Orderer 节点能够处理此量级的交易,每秒的交易数量不足以触发区块最大交易数量的出块条件。而当发送速率超过 200 tps 时,吞吐量达到瓶颈并稳定在 200 tps 左右,此时 Fabric 系统无法处理更多的交易,此时增加交易发送速率,必然导致一部分交易在排序阶段或验证阶段等待。因此,发送速率为 250 tps 和 300 tps 时,平均交易时延突然增加。

可以看到,FabricDT 与 Fabric 在吞吐量和交易时延上基本没有差距,FabricDT 在无交易冲突时,不会造成明显的性能下降。对比 FabricDT 和 Fabric 增加的平均交易时延、交易冲突检测的时间开销和维护未验证交易缓存的空间开销,可以看出各项损耗都在可以接受范围之内(如表 5 所示)。

实验结果显示系统的吞吐量上限在 200 tps 左右,在后续的实验中为了使系统处于满负荷状态,将发送速率设置在 250 tps。

4.2 不同冲突程度下性能分析

对比 FabricDT、Fabric++和 Fabric 在存在冲突交



(a) 吞吐量　　(b) 平均交易时延

图 6 无冲突时交易吞吐量和平均交易时延

表 5 资源损耗

发送速率/tps	平均交易时延增值/ms	冲突检测时间开销/μs	冲突检测机制空间开销/kb
50	0.18	21.59	85.38
100	0.21	21.30	111.92
150	0.28	48.20	137.20
200	1.9	51.79	192.20
250	5.23	75.82	1 746.90
300	5.73	447.87	3 217.57

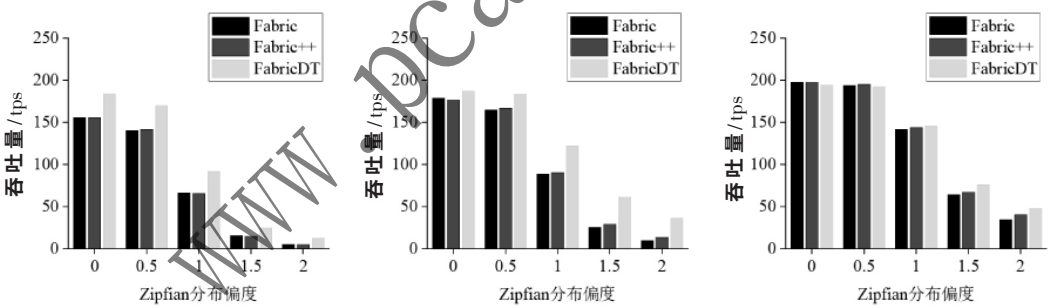
易时的性能差距。通过将读交易的比例 P_q 分别设置为 5%、50%、95%，将 Zipfian 分布的偏度从 0 调整到 2，步长为 0.5，以实现不同程度的冲突交易，交易发送速率固定为 250 tps，测试不同设置下的性能，结果可如图 7、图 8 和表 6 所示。

可以看到，FabricDT 在绝大多数情况下能够在吞吐量和平均交易时延上优于 Fabric 和 Fabric++。

图 7(a)和图 7(b)显示，在不同的 Zipfian 分布偏度下，FabricDT 吞吐量均大于 Fabric 和 Fabric++系统的吞吐量。但是，在图 7(c)中三个方法的差距不大，并且在偏度为 0 和 0.5 时，FabricDT 和 Fabric++的吞吐量略低于 Fabric，是因为写交易的比例只占 5%，冲突程度也比较低，因此冲突交易所占的总体交易比例很小，利用额外的机制去解决这些冲突的代价高于它所带来的收益，造成了系统性能下降，但是下降幅度非常小。

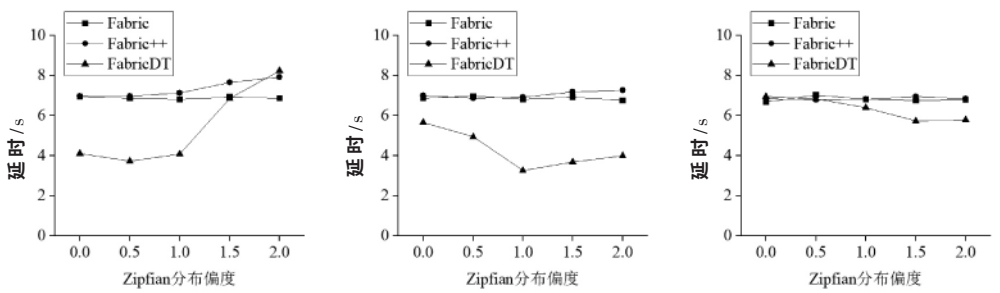
可以看到，FabricDT 在整体上要比 Fabric 和 Fabric++拥有更低的时延（如图 8 所示）。原因是 FabricDT 将冲突检测放在了第一个阶段，即执行阶段，对不冲突的交易不会造成明显的性能损失。而对冲突的交易，FabricDT 直接将其标记为无效，节省了系统资源，从而极大地降低了时延。因此，FabricDT 整体上具有最低的时延。

随着偏度的增加(从 1 增加到 2)，FabricDT 会出现时延增加的情况，原因是随着冲突增加，执行阶段未验证交易缓存的大小会增加，进行冲突判断和



(a) P_q = 5 %　(b) P_q = 50 %　(c) P_q = 95 %

图 7 不同冲突程度下交易吞吐量



(a) P_q = 5 %　(b) P_q = 50 %　(c) P_q = 95 %

图 8 不同冲突程度下平均交易时延

表 6 不同冲突程度下交易成功率

Zipfian 分布偏度	P_q = 5%			P_q = 50%			P_q = 95%		
	Fabric	Fabric++	FabricDT	Fabric	Fabric++	FabricDT	Fabric	Fabric++	FabricDT
0	0.79	0.79	0.86	0.89	0.89	0.91	0.99	0.99	0.99
0.5	0.72	0.72	0.82	0.83	0.84	0.86	0.97	0.98	0.98
1	0.34	0.34	0.46	0.46	0.47	0.61	0.72	0.73	0.73
1.5	0.08	0.08	0.12	0.13	0.15	0.31	0.33	0.35	0.38
2	0.03	0.03	0.06	0.05	0.07	0.18	0.18	0.21	0.24

后续对缓存的更新所需要的时间也会增加；同时，平均交易时延指标只统计了所有提交到区块的交易时延，在执行阶段标记为无效的交易不参与统计，也导致了平均交易时延的上升。但是从实验结果看，即使出现时延超过 Fabric 和 Fabric++，它们之间的差距也很小且可接受，而在其他情况下，FabricDT 都要优于两者。

FabricDT 在任何情况下都具有最高的交易成功率（如表 6 所示）。因此，可以认为 FabricDT 在总体性能上胜过 Fabric 和 Fabric++。

5 结论

本文分析了 Hyperledger Fabric 上冲突交易产生的原因，并且提出了一种利用缓存交易写集的方式，通过在 Peer 节点上设置冲突检测模块，在执行阶段检测交易之间的并发冲突性并进行处理。实验结果表明，本文的方法能够有效增加系统吞吐量，降低交易时延，提高成功交易的比例。

参考文献

[1] NAKAMOTO S. Bitcoin: a peer-to-peer electronic cash system [EB/OL]. (2008-XX-XX) [2022-03-15]. https://bitcoin.org/bitcoin.pdf.

[2] Ethereum Foundation. Ethereum project [EB/OL]. (2009-XX-XX) [2022-03-15]. https://wwwethereum.org/.

[3] FAN K, WANG S, REN Y, et al. Medblock: efficient and secure medical data sharing via blockchain [J]. Journal of medical systems, 2018, 42 (8): 1-11.

[4] TSE D, ZHANG B, YANG Y, et al. Blockchain application in food supply information security [C]// 2017 IEEE International Conference on Industrial Engineering and Engineering Management, 2017: 1357-1361.

[5] 李峰，梁任纲，李雪聪，等. 结合区块链和属性基的可信数据分发 [J]. 小型微型计算机系统，2021，42(7)：1524-1531.

[6] 白杰，杨鹏飞，孙鲜艳，等. 基于 CNWW3 区块链体系标准建立的数字版权应用[J]. 信息技术与网络安全，2020，39(7)：18-30.

[7] ANDROULAKI E, BARGER A, BORTNIKOV V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains [C]//Proceedings of the EuroSys Conference, 2018: 1-15.

[8] DINH T, WANG J, CHEN G, et al. Blockbench: a framework for analyzing private blockchains [C]// Proceedings of the ACM International Conference on Management of Data, 2017: 1085-1100.

[9] RAUCHS M, BLANDIN A, BEAR K, et al. 2nd global enterprise blockchain benchmarking study [J]. Available at SSRN 3461765, 2019.

[10] SHARMA A, SCHUHKNECHT F, AGRAWAL D, et al. Blurring the lines between blockchains and database systems: the case of hyperledger fabric [C]// Proceedings of the 2019 International Conference on Management of Data, 2019: 105-122.

[11] ZHANG S, ZHOU E, PI B, et al. A solution for the risk of non-deterministic transactions in hyper-ledger fabric [C]// 2019 IEEE International Conference on Blockchain and Cryptocurrency, 2019: 253-261.

[12] XU L, CHEN W, LI Z, et al. Locking mechanism for concurrency conflicts on hyperledger fabric [C]// International Conference on Web Information Systems Engineering, 2020: 32-47.

[13] RUAN P, LOGHIN D, TA Q T, et al. A transactional perspective on execute-order-validate blockchains[C]//

（下转第 108 页）

[2] 周沛,熊文华.交叉口信号控制评价指标体系[J].公路交通技术,2020,36(1):133-139.
[3] 姚树申,齐攀.城市交叉口信号控制评价指标浅析[J].科技信息,2009(13):544-545.
[4] 陈哲. 信号交叉口计算机仿真系统评价指标研究[J].交通标准化,2005(10):52-55.
[5] 孙吉瑞,车国鹏,温汉辉,等.综合待行区对交叉口通行能力及延误影响的研究 [J]. 公路交通技术,2015(1):121-125.
[6] 那娜,谢春丽.基于延误的哈尔滨平面信号交叉口服务水平评估[J].森林工程,2018,34(1):75-79.
[7] 韩直,王振科.道路交通参数敏感性分析[J].公路交通技术,2017,33(6):114-116.
[8] 朱湧,徐建川,陈晓利,等.基于多智能体的城市道路短时交通流预测与仿真研究[J].公路交通技术,2016,32(6):135-139,147.
[9] SAATY T L. Decision making:the analytic hierarchy and network processes (AHP/ANP)[J]. Journal of Systems Science and Systems Engineering,2004,13(1):1-35.
[10] 邸卉芳.基于强化学习的配电网重构策略研究[D].南京:南京师范大学,2021.
[11] 闫茂德,张倩楠,刘小敏.智能网联汽车变车距队列控制与仿真[J].计算机仿真,2020,37(1):126-130.
[12] 刘飞强,蔡文豪.多专家评价的AHP方法及其工程应用[J].广东土木与建筑,2019,26(3):69-71.
[13] 陈海清,吴悦,沈俊,等.基于AHP的审稿专家信息数据库[J].济南大学学报(自然科学版),2008(2):153-157.
[14] 马琳,陈复扬,姜斌.交通物联网中基于改进Webster方法的单点信号配时研究 [J]. 物联网学报,2018,2(4):49-55.

(收稿日期:2022-03-06)

作者简介:

倪茹(1998-),女,硕士研究生,主要研究方向:智能交通。

(上接第101页)

Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data,2020:543-557.
[14] GORENFLO C, GOLAB L, KESHAV S. XOX Fabric: a hybrid approach to blockchain transaction execution [C]//2020 IEEE International Conference on Blockchain and Cryptocurrency ,2020:1-9.
[15] NASIRIFARD P, MAYER R, JACOBSEN H A. FabricCRDT:a conflict-free replicated datatypes approach to permissioned blockchains [C]//Proceedings of the 20th International Middleware Conference,2019:110-122.
[16] SHAPIRO M, PREGUICA N, BAQUERO C, et al. Conflict-free replicated data types[C]//Symposium on Self-Stabilizing Systems,2011:386-400.
[17] ZIPF G K . Human behavior and the principle of least effort [J]. American Sociological Review, 1949, 14(6):822.
[18] XU X, WANG X, LI Z, et al. Mitigating conflicting transactions in hyperledger fabric-permissioned blockchain for delay-sensitive IoT applications[J]. IEEE Internet of Things Journal,2021,8(13):10596-10607.

(收稿日期:2022-03-15)

作者简介

王盛姣(1997-),女,硕士研究生,主要研究方向:区块链。

董建亮(1997-),男,硕士研究生,主要研究方向:区块链。

李京(1966-),通信作者,男,博士,教授,主要研究方向:分布式系统、云计算、区块链。E-mail:lj@ustc.edu.cn。

版权声明

经作者授权，本论文版权和信息网络传播权归属于《信息技术与网络安全》杂志，凡未经本刊书面同意任何机构、组织和个人不得擅自复印、汇编、翻译和进行信息网络传播。未经本刊书面同意，禁止一切互联网论文资源平台非法上传、收录本论文。

截至目前，本论文已经授权被中国期刊全文数据库（CNKI）、万方数据知识服务平台、中文科技期刊数据库（维普网）、JST 日本科技技术振兴机构数据库等数据库全文收录。

对于违反上述禁止行为并违法使用本论文的机构、组织和个人，本刊将采取一切必要法律行动来维护正当权益。

特此声明！

《信息技术与网络安全》编辑部

中国电子信息产业集团有限公司第六研究所