Hyperledger Fabric 中多版本并发控制冲突的早期检测

Helmi Trabelsi*、Kaiwen Zhang*
*蒙特利尔高等技术学院软件与信息技术工程系
{helmi.trabelsi.1@ens.,
kaiwen.zhang@}etsmtl.ca

arXiv:2301.06181v1 [cs.DC] 15 Jan 2023

摘要——**Hyperledger Fabric 是一种广受欢迎的许可型区块链系统，具有高度模块化且可扩展的特点，可用于部署许可型区块链，预计将对广泛领域产生重大影响。与比特币和以太坊等传统区块链系统不同，Hyperledger Fabric 使用 EOV 模型进行交易处理：提交的交易由背书节点执行，由排序服务排序并打包成批次，再由验证节点进行验证。由于采用这种 EOV 工作流，一个有据可查的问题是多版本并发控制冲突。当两笔交易同时尝试对账本中的同一键执行写操作和读取操作时，就会发生这种情况。针对这一问题，现有解决方案包括：不再使用区块而改用流式交易、在排序阶段修复冲突，以及使用 CRDT（无冲突复制数据）技术自动合并冲突交易。**

在本文中，我们提出了一种名为*MVCC* 冲突早期检测的新颖解决方案。该方案会在交易执行的早期阶段检测冲突交易，而不是一直处理它们，直到它们在验证阶段被中止。该方案的优势在于能够尽早检测冲突，从而最大限度地降低冲突交易给网络带来的开销，进而缩短端到端交易延迟并提高系统的有效吞吐量。我们已在 Hyperledger Fabric 中成功实现该方案，并提出了三种实现早期检测的不同方案。结果表明，我们的方案均优于基线 Fabric；其中表现最佳的 SyncMap 可将有效吞吐量提高至多 23%，并将延迟降低至多 80%。

索引词——**Hyperledger Fabric、MVCC、EOV。**

I. 引言

Hyperledger Fabric 是一种广受欢迎的免许可系统，可用于开发金融、医疗保健和供应链管理等多个领域的区块链应用。与以太坊 [1]和比特币 [2], 相比，Fabric 凭借所采用的共识算法以及网络中对等节点数量较少（相较于公有区块链），可提供更出色的性能。此外，Fabric 还提供了一个灵活的框架，通过成员服务提供者（MSP）管理各方的职责。MSP 是一个抽象组件，负责维护属于同一组织的所有节点的保存属于同一组织的所有节点的身份和角色。

大多数现有区块链平台（如以太坊和 Hyperledger Fabric）都允许通过为给定交易执行智能合约来进行图灵完备计算 [3]。为了保证网络中的一致性，对等节点会在获知下一个区块的内容（要执行哪些交易以及按什么顺序执行）后，执行交易以生成区块链的下一个状态：这称为排序—执行（OX）交易流程 [4]。OX 模式的缺点在于，每个区块内交易的顺序执行限制了系统的吞吐量。此外，每个对等节点都需要了解所有智能合约才能计算区块链的下一个状态，这可能会带来机密性和隐私问题。

相反，Fabric 采用执行-排序-验证（EOV）模式：交易在称为背书节点的沙箱中执行，以生成读写集，其中包含交易模拟所用键的版本；随后，排序服务将交易排序并打包成区块，最后对其进行验证并提交到账本。EOV 通过支持交易在不同背书节点上并行执行，克服了 OX 的局限性。

EOV 模式的缺点是使用读写锁来同步执行阶段和验证阶段。为解决这一问题，Hyperledger Fabric 实现了多版本并发控制机制，以保证区块链的一致性。验证节点验证交易时，会将生成的读写集中的版本与账本中各键的版本进行比较，以避免两个交易同时尝试修改或读取相同的键值对。如果某笔交易的读集中包含某个键的旧版本，该交易将被拒绝，客户端必须重新提交。

前述 MVCC（多版本并发控制）冲突问题是 Hyperledger Fabric 中有据可查的一项局限，因为它会降低系统的有效吞吐量（也称为有效吞吐量）。这是因为区块可能包含

已中止交易仍会占用大小限制，从而浪费宝贵的区块空间。相应的客户端还必须将这些已中止交易作为全新交易重新提交，这会给系统带来额外负载。实际上，MVCC 问题可能会严重影响区块链的性能。近期研究表明，在电子健康记录等实际场景中，40% 的交易因并发冲突而失败 [5]。以往的研究尝试通过采用无锁方法提供事务隔离 [6] ，或在排序阶段重新排列交易，以尽量减少验证并提交交易到账本时发生的冲突 [7] [8]。

在本文中，我们提出了一种解决 MVCC 问题的新方法，称为 *MVCC* 冲突早期检测 （EMVCC）。我们引入了一种 EMVCC 检测机制，旨在减少交易之间的冲突数量，从而提高系统的整体有效吞吐量。与现有研究相比，我们的解决方案的优势在于，MVCC 冲突会在交易首次接触区块链网络时，于背书策略阶段被检测出来，从而提升网络性能。

本文的贡献如下：

1）我们提出了在背书阶段早期检测冲突交易的问题表述，而非采用传统方法，在验证阶段处理 MVCC 冲突（第 IV-B 节）。2 ）我们提出了一种新颖的解决方案，称为使用背书节点本地缓存的 MVCC 冲突早期检测（第 IV-C 节）。我们对该解决方案进行了理论分析，以根据背书策略、组织数量等参数计算其预期效果（第 IV-D 节）。3）我们介绍了该解决方案的三种参考实现：SyncMap、无锁和互斥锁；它们的主要区别在于用于存储信息的数据结构（第 IV-E 节）。4）我们评估了该解决方案并将其与基线进行比较，还进行了敏感性分析，以研究不同网络和负载参数对性能的影响（第 V 节）。

本文接下来进入第二节，概述 Hyperledger Fabric。随后在第三节中，我们将介绍与 MVCC 问题相关的已有研究。

II. 背景

本节概述 Hyperledger Fabric 组件，详细介绍交易流程，并介绍背书策略和多版本并发控制读写冲突（MVCC）。

A. Hyperledger Fabric 组件

Hyperledger Fabric 是一种专为商业应用设计的许可型分布式账本系统。其模块化架构使区块链解决方案具备保密性、韧性和灵活性。Hyperledger Fabric 包含多个基本组件，例如客户端和对等节点，其中对等节点分为背书节点和提交节点。本文将重点讨论这些组件，因为只有这些组件会受到我们提出的解决方案影响。

1) 客户端： 客户端是代表用户在 Fabric 网络上通过 Hyperledger Fabric SDK 提交交易的应用或软件。

2) 对等节点： 对等节点是网络的基本组件，因为它们保存网络状态的记录和账本副本。网络中的每个对等节点都属于某个组织，而一个组织可以拥有一个或多个对等节点。管理员可以创建、启动、停止或重新配置对等节点。要成为网络的一部分，节点必须加入所有参与者共享的通道。对等节点分为两类：背书节点和验证/提交节点。在典型场景中，每个 Fabric 节点通常同时承担背书和验证这两种角色。

背书节点：此类节点托管智能合约，用于执行交易模拟并使用节点签名为交易背书。安装链码（智能合约）时，管理员通过定义背书策略规则，指定哪些对等节点将为交易背书。

验证/提交节点：它们负责运行 VSCC（验证系统链码），用于验证背书策略，并进行多版本并发控制（MVCC）验证，以确保背书阶段读取的键版本在账本中仍然相同，从而保证交易的确定性执行。随后，提交节点将区块添加到共享账本，并更新区块链世界状态。

B. 交易流程

图 1 展示了 Hyperledger Fabric 中的基本交易流程。该流程包含五个步骤：

- **步骤 1**：希望发起交易的客户端会根据链码背书策略（见下一节第 II-C 节），向背书节点发送交易提案，其中包含客户端希望调用的链码函数及其参数。

- **步骤 2**：背书节点接收交易提案，执行链码并生成

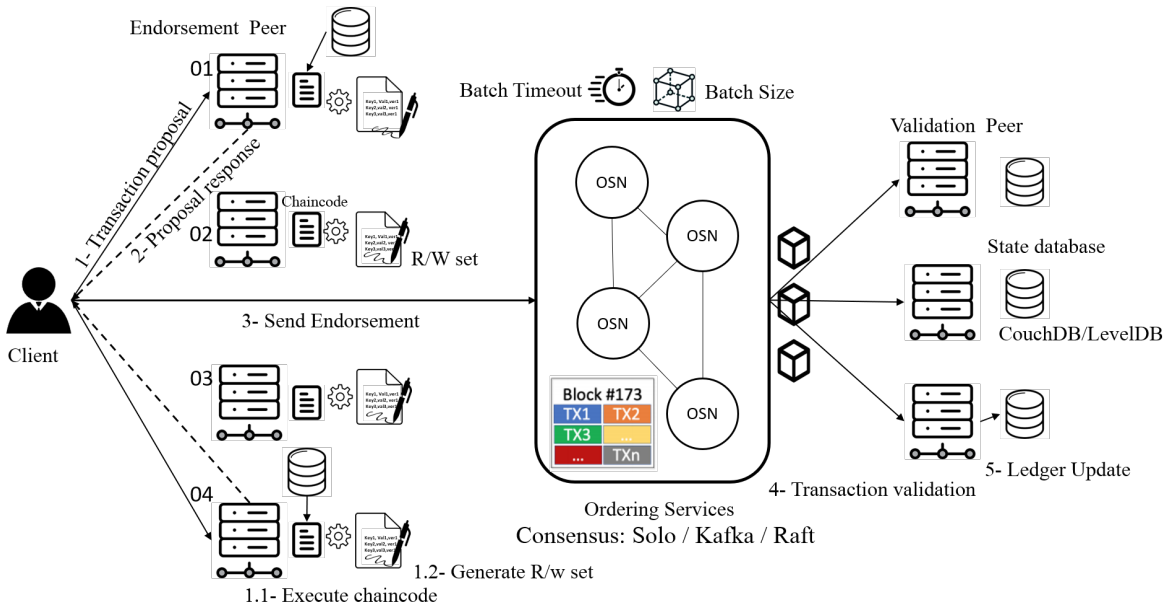


图 1：交易流程

读写集，其中包含执行交易时读取或写入的键的值和版本。随后，对等节点创建提案响应，对其签名并将响应发送给客户端。

- **步骤 3**：客户端按照背书策略的定义收集到所需数量的背书后，将交易发送给排序服务。此操作包含有关初始提案的详细信息，以及所有对等节点的背书和读写集。排序服务根据区块批次大小和区块批次超时，将客户端收到的交易排序并打包成区块，然后创建该区块并将其交付给验证对等节点。

- **步骤 4**：提交对等节点收到区块后，会遍历区块中的所有交易，对每笔交易执行语法验证、VSCC 验证和 MVCC 验证（见下一节 II-D）。

- **步骤 5**：交易通过这三项检查并被标记为有效后，会被添加到新区块中，并通过应用该交易的写集更新账本；否则，无效交易将被拒绝。

C. 背书策略

Hyperledger Fabric 允许开发者在链码级别设置策略。背书策略是一些规则，用于指定在交易执行后、将其添加到账本之前，哪些对等节点可以达成一致。通常，背书策略在安装链码时配置，只有在链码升级期间才能修改。客户端创建交易后，会向所有满足背书策略的背书节点发送交易提案 [9]，并等待提案响应。当客户端收到足够的响应和签名，以满足链码的背书策略时，就可以将带有背书签名的交易提交给排序服务。背书策略可以定义如下：

- 与('Org1.member','Org2.member')：客户端需要从每个组织获得一个签名，才能提交交易
- OR('Org1.member','Org2.member')：客户端需要这两个组织中的任意一个提供一份签名，才能提交交易
- OR('Org1.member',AND('Org2.member' ,' Org3.member'))：客户端需要以下任一条件满足：组织1成员提供一份签名，或组织2成员和组织3成员各提供一份签名
- Out-Of(2,'Org1.member', 'Org2.member',' Org3.member')：三个组织中至少有两个组织必须为交易背书，交易才有效。这等价于 OR(AND('Org1.member','Org2.member'),与('Org1.member','Org3.member'), 与(' Org2.member','Org3.member'))

本文研究前两个示例（与、或），并将其用于我们的解决方案。不过，类型 3 和 4 已有相关研究，但由于篇幅所限，本文不予讨论。此外，我们假设在策略允许选择的情况下，每个客户端都会随机选择背书节点。

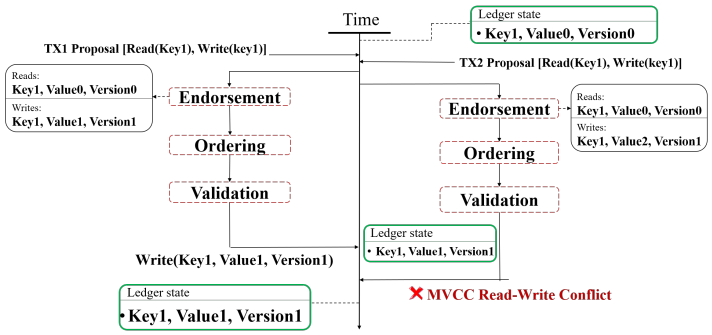


图 2：MVCC 读写冲突示例

D. 多版本并发控制读写冲突（*MVCC*）

Hyperledger Fabric 使用多版本并发控制系统来确保账本的一致性。该机制会验证，在交易背书时读取的键版本在验证阶段 [10]是否仍保持不变。此过程可确保不会读取已被另一笔并发交易更改的旧值。在背书阶段与验证阶段之间，如果另一笔交易更新了读集中列出的键的版本，该交易将在 MVCC 验证时失败。

多版本并发控制读写冲突是指两个客户端同时尝试更新和读取同一个键时发生的问题。图 2 展示了这种冲突的一个示例。假设*user1* 提交*TX1*，同时*user2*提交*TX2*，这两笔交易将读取并更新 *Value1* 这一*Key1*的值。两笔交易同时经过背书和排序后，*TX1* 将通过验证，其写集将应用到账本中，从而将*Key1* 关联的值和版本分别修改为*value1* 和 *version1*。然而，*TX2* 经过 MVCC 验证时会失败，因为该交易读集中的*Key1* 版本与账本中的版本不一致。因此，MVCC 验证会检测到账本与背书结果之间的不一致，并返回 MVCC 错误。

III. 相关工作

Hyperledger Fabric 是一个相对较新的系统，目前已在进行一些重大的架构改进。大多数相关研究旨在提高网络的吞吐量并降低延迟，但针对 MVCC 问题的有效解决方案仍然不足，而且多数情况下，评估结果并未考虑交易冲突因素。本节回顾了近期有关提升 Hyperledger Fabric 性能的技术研究。我们将这些研究分为三类：优化背书阶段的研究、改进排序阶段的研究，以及增强验证阶段的研究。

A. 背书阶段优化

[11] 所做的工作旨在通过在背书过程中区分读交易和写交易，改进读交易处理。与传统 Fabric网络相比，这种方法将交易背书延迟降低了 60%。该方法与我们的解决方案互为补充，因为仍可应用早期检测，进一步降低失败交易的延迟。

B. 排序阶段优化

针对排序阶段，[12] 提出以流式处理交易取代区块概念，从而减少批处理开销。此外，作者为排序服务实现了基于现场可编程门阵列（FPGA）的共识机制，通过将延迟减半，使提交延迟降至一毫秒以下，相较于基于Raft的排序服务实现了这一目标。

Fast Fabric 的作者 [13] 重新设计了 Fabric 排序服务，使其仅处理交易ID：将交易头与负载分离，只依据交易ID确定交易顺序，从而加快排序服务中的交易处理并提高吞吐量。

[7] 提出了一种解决方案，旨在通过在排序阶段利用冲突图重新排列交易，并中止无法序列化的交易，从而降低 MVCC 失败率。++

[8] 设计了一种优化后的 Fabric 扩展++，能够处理区块间和区块内 MVCC 读冲突。

我们提出的解决方案会在背书阶段检测到 MVCC 冲突，而 Fabric++和 FabricSharp 则旨在排序阶段对交易重新排序，以避免尽可能多的冲突。

C. 验证阶段优化

多篇文章提出并行执行验证过程（语法验证、背书策略验证、MVCC 验证），以加快区块验证 [14], [13], [12], [15], [16]。

FabricCRDT 专注于使用无冲突复制数据类型技术自动合并冲突交易，而不拒绝它们 [17]。[12] 通过使用本地批处理器引入磁盘写入批处理机制，该批处理器会累积账本更新操作，直到达到批次大小或批处理超时时间，再将其写入数据库。

最后，[6]提出了一种用于实现事务隔离的无锁解决方案。该方法在确保事务隔离的同时，允许移除共享锁。

验证阶段的改进方案基于验证过程的并行化，或对数据库读写操作的优化。所有这些工作都为改进 Hyperledger Fabric 的性能提供了有益见解，并可作为对我们所提方案的补充。

D. 数据库系统中的 *MVCC*

MVCC 是数据库领域研究最多的问题之一，并被区块链系统沿用。多项研究都探讨了这一问题，例如 [10] 提出了两种针对内存数据库的优化 MVCC 机制。与单版本锁定相比，所提方案开销更高，但对热点和长时间运行的交易不那么敏感。此外，[18] 提出了 BOHM，这是一种用于主存多版本数据库系统的并发控制机制，可确保读操作不会阻塞写操作。这些方案有助于改进基于数据库的区块链系统。然而，我们所提方案具有预防性，因为它在验证阶段、MVCC 问题发生之前就对其进行处理。

IV. EMVCC 检测与解决方案

本节将对背书阶段的 MVCC 检测问题进行形式化描述。随后，我们将介绍所提出的解决方案——提前 MVCC 检测，并展示不同的缓存数据结构。此外，我们还将对不同的背书策略进行理论分析。

A. 系统模型

在本节中，我们将使用以下模型：给定一个包含 N 个组织的 Hyperledger Fabric[1] 网络，其中每个组织有 M 个对等节点并且 M_i 表示组织 i 的对等节点数量。系统使用 $Nb_{Tx/Block}$ 作为每个区块的交易数。系统的排序服务可以使用 Hyperledger Fabric 中可用的任何共识协议（例如 Raft、Kafka、Solo）。应用执行的链码会生成 %$cfct$ 作为冲突率，表示因 MVCC 冲突错误而中止的交易所占的百分比。直观地说，交易对少数共享键进行读写的链码，其冲突率会高于交易分别独立操作不相交键的链码。为了存储链状态，系统可以使用任何与 Hyperledger Fabric 兼容的可用数据库（CouchDB、LevelDB 等）来实现。

[1]我们的解决方案已在 Fabric 2.0 及更高版本中进行测试。

B. 问题形式化

我们要解决的问题是：根据每个背书节点的已背书交易历史，在背书阶段筛选交易，从而检测 MVCC 冲突。这与 Fabric 当前解决的问题不同：Fabric 在交易流程中较晚的验证阶段检测 MVCC 冲突。

我们首先定义一些后文将用到的基本术语：

- **未检测概率：** 这是用于评估我们解决方案性能的一项指标。理想的解决方案应具有 $\mathcal{P}(ND) = 0$。它表示这样一种概率：尽管某个交易与另一笔交易存在冲突，我们提出的解决方案却未能在背书阶段检测到该冲突交易，导致 MVCC 冲突直到验证阶段才被检测到。

- **假阳性：** 我们将使用本解决方案错误地判定为无效、但采用标准方法本可成功确认的交易称为假阳性。如果我们的解决方案在检测冲突时过于激进，就可能发生这种情况：它会过早中止原本有机会成功的交易，迫使客户端使用新的交易重新执行整个背书过程。假阳性率的计算公式如下：

$$FP = \frac{Tx/Block \times \%cfct^2 \times \mathcal{P}(ND) \times (1-\mathcal{P}(ND))}{\%cfct \times \mathcal{P}(ND) - \%cfct + 1} \quad (1)$$

其中，$Tx/Block$ 表示每个区块的交易数，$\mathcal{P}(ND)$表示未检测概率，%$cfct$ 表示冲突率

- **假阴性：** 这是相反的错误：交易在 EMVCC 验证阶段被判定为有效，但实际上它与另一笔交易存在冲突，导致其随后失败。理想情况下，假阴性的数量应为零。

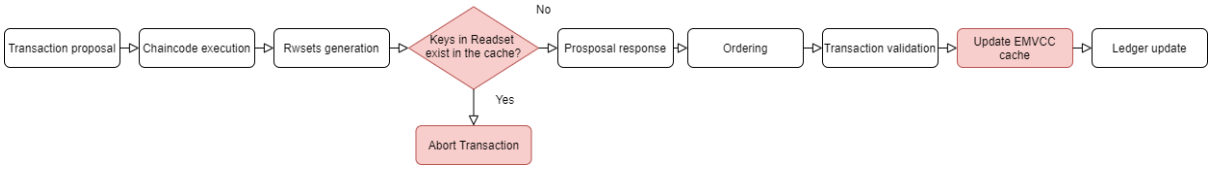


图 3：EMVCC 检测机制

$$FN = TX_rate \times \%cfct \times \mathcal{P}(ND) \quad (2)$$

- **有效吞吐量**[19] 是网络能够写入区块链的成功交易速率。相比之下，网络的吞吐量是所有经过排序阶段的交易总速率，其中包括因 MVCC 冲突而将被中止的交易。理想情况下，有效吞吐量应等于吞吐量。在标准 Fabric 实现中，有效吞吐量会直接受到 MVCC 冲突率的影响，因为任何被中止的交易仍然属于提议区块的一部分。我们的解决方案可以在排序阶段之前检测此类冲突，防止这些冲突反映在区块链中，从而提高有效吞吐量。

我们可以按如下方式计算：

$$goodput = \frac{Nb_valid_TX}{Nb_TX} \times Throughput \quad (3)$$

C. 所提方案

在 Fabric 的当前实现中，冲突交易会经过网络，最终失败，从而消耗不必要的资源。在我们提出的方案中，我们在背书阶段增加了一层，用于筛选交易并中止任何检测到的冲突交易。我们将这种方法称为早期多版本并发控制（EMVCC）。

如图 3 所示，我们的 EMVCC 方案在 Fabric 执行链码并生成读写集之后、背书响应发送回客户端之前运行。我们的 EMVCC 方案会将读集与存储在对等节点本地缓存中的待处理交易列表进行比较。此缓存由该对等节点此前已背书、当前仍在处理中且尚未确认（或中止）的交易填充。

如果交易读取的键存储在缓存中，则该交易将被中止，因为检测到了早期 MVCC 冲突。如果没有，则将该交易存储在本地缓存中，并照常向客户端发送背书。一旦交易在验证阶段得到验证或被拒绝，就会更新缓存，移除该交易写集中的键。

请注意，由于我们的 EMVCC 无法完美检测冲突，通常的 MVCC 阶段仍会在验证阶段执行

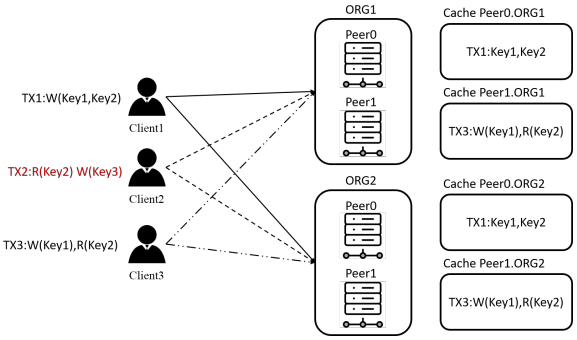


图 4：EMVCC 检测示例

（参见第 IV-D 节）。这样，常规验证阶段便能捕获通过我们的 EMVCC 方案未检测到的任何 MVCC 冲突，从而确保 EOV（执行-排序-验证）执行流程的正确性。

图 4 展示了 EMVCC 检测的工作示例。假设有一个由两个组织组成的 Fabric 网络，每个组织有两个对等节点。链码背书策略为与 (Org1, Org2)。假设客户端*1* 提交*TX1* ，该交易对*Key1* 和 *Key2*执行写操作。该交易由*Peer0.ORG1* 和 *Peer0.ORG2*背书。现在假设有一笔交易*TX2* ，它读取*Key2*并对*Key3*执行写操作。如果该交易由 *Peer0.ORG1* 或*Peer0.ORG2*背书，那么它将在背书阶段被中止，因为*TX1* 位于*Peer0.ORG1* 或 *Peer0.ORG2*的 EMVCC 缓存中。假设另一笔交易 *TX3* 到达，它对*Key1*执行写操作并读取*Key2*。如果该交易由*Peer1.ORG1* 和*Peer1.ORG2*背书，EMVCC 检测机制将无法检测到它，因为这些对等节点的 EMVCC 缓存中没有与这些键相关的冲突。不过，它仍会在之后的 MVCC 验证阶段失败。

D. 未检测到的理论分析

由于我们的方法性质轻量，且仅使用本地缓存而不依赖任何同步，因此冲突交易可能在未被检测到的情况下通过 EMVCC 阶段。如果为当前交易背书的对等节点子集此前未曾为与其冲突的待处理交易背书，就可能发生这种情况。因此，背书策略、对等节点数量以及组织数量都会直接影响

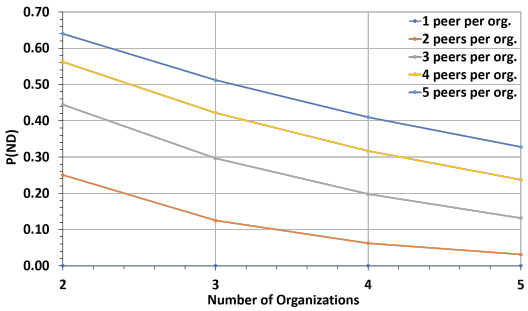


图 5：使用 AND 策略时的未检测概率

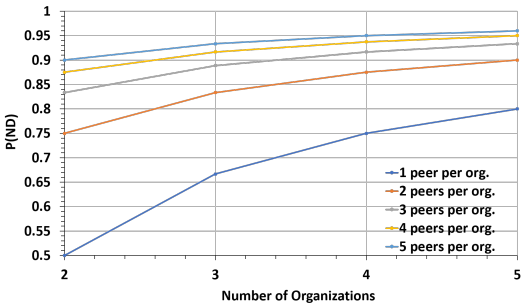


图 6：使用 OR 策略时的未检测概率

我们提出的 EMVCC 方案的未检测概率。

为了评估我们方案的性能，我们提出了一个解析模型，用于计算采用 AND 和 OR 背书策略时该方案的预期未检测概率。我们将在第V节中将这些理论结果与实际结果进行比较，以验证实现的正确性。在此分析中，我们假设当客户端需要从多个对等节点（或组织）中选择节点（或组织）以满足背书策略时，会以均匀随机的方式进行选择，这与 Hyperledger Fabric [9] 当前的实现方式一致。由于篇幅有限，我们将此处推导的数学公式的完整细节留待扩展报告中介绍。

1）AND 背书策略中的未检测：使用 AND 背书策略时，未检测概率可按如下方式计算：

$$\mathcal{P}(ND) = \prod_{n=1}^{N} \frac{(M_i - 1)}{M_i} \tag{4}$$

其中，N 为组织数量，M_i 为每个组织的节点数量

图 5 绘制了采用 AND 背书策略时，EMVCC 方案在不同组织数量和对等节点数量下未检测概率的理论变化趋势。我们观察到，组织数量增加时，未检测概率会降低；然而，每个组织的对等节点数量增加时，未检测概率会升高。

2) OR 背书策略中的未检测：我们还分析了使用 OR 背书策略时未检测概率的变化，其计算公式如下：

$$\mathcal{P}(ND) = \begin{cases} \frac{NM-1}{NM} & \text{if } M_i{=}M_{i+1} \\ \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M_i} & \text{else} \end{cases} \tag{5}$$

其中，N 为组织数量，M_i 为每个组织的对等节点数量

图 6 展示了在采用 OR 背书策略时，不同组织数量和对等节点数量下未检测概率的理论变化。我们发现，由于节点缓存之间缺乏同步，组织数量或对等节点数量增加时，未检测概率也会增加。

E. 缓存数据结构的选择

对于每个对等节点维护的本地缓存，我们在参考实现中提出了三种不同的技术：

MutexLock：在此实现中，我们使用基于锁的数据结构，以确保多线程系统中的缓存一致性 [20]。例如，当背书节点读取或写入缓存时，该缓存条目会被锁定，验证节点无法更新此缓存条目。这种方法的主要缺点是，锁定机制会使读写操作产生延迟，因为某些线程必须等待锁释放。

LockFree：采用这种方法时，多个线程可以并发访问缓存。读写操作会先存入缓冲区；当缓冲区达到阈值后，再将批量操作应用到缓存中，从而确保大多数线程在每一步都能取得进展 [20]。这项技术有助于减少获取或释放锁的次数。

SyncMap：同步映射是一种经过优化且安全的数据结构，自 Golang 1.9 起便属于 sync 包。它使用脏映射写入新值或更新值，使读取操作无需加锁即可完成。然而，当大量并发线程（协程）同时向脏映射写入，且只读映射通过一批来自脏映射的操作进行更新时，互斥锁仍不可或缺。

V. 基准测试结果

在本节中，我们使用三种不同的缓存实现——互斥锁、LockFree 和

SyncMap，评估所提出的 EMVCC 解决方案的有效性。我们首先将这三种不同的实现与作为基线的标准 Fabric 实现进行比较。随后，我们开展敏感性分析，以研究某些系统参数和工作负载特征对系统性能的影响。表 I 展示了实验所使用的默认参数。我们将每个区块的交易数设为 500，并将区块时间设为 2 秒；根据我们的实验测试，这些排序参数可带来最高吞吐量（见第V节-D）。此外，我们将冲突率设为 40%，因为在实际场景中，该比例代表因并发冲突而失败的交易所占的百分比 [5]。我们选择 Raft 共识，而不是另外两种排序服务，因为独立模式仅建议用于开发和测试 [21]，并且 Kafka 已在 Fabric 2.x 版本中弃用。我们还选择了 3 个排序服务节点，以在网络成本与其性能之间实现权衡。

A. 测试环境特征

我们的集群由托管在 Google Cloud Platform 上的六台 E2 虚拟机组成。其中两台充当对等节点，另外三台运行排序服务，并采用 Raft 共识算法。剩余的一台虚拟机运行 Hyperledger Caliper [22] 基准测试工具，充当客户端。每台虚拟机配备 8 个虚拟 CPU （虚拟中央处理器）和 32 GB 随机存取存储器。所有虚拟机均运行 Ubuntu 16.04 操作系统。Fabric 对等节点配置为使用我们修改版的 Hyperledger 2.3 镜像，并使用 CouchDB 作为状态数据库。链码方面，我们使用 Fabcar 链码，它允许我们在区块链上创建汽车并修改其所有者。

B. 性能指标

我们的基准测试主要采用以下指标：

1) **有效吞吐量**：写入区块链的已提交交易的有效吞吐量，不包括已中止的交易（第 IV-B 节）。2) **延迟**：从客户端发起交易请求到交易最终提交到账本所经历的时间。3) **EMVVC 与 MVCC 比率**：因 EMVCC 错误被拒绝的交易所占比例，与因 MVCC 错误被拒绝的交易所占比例。4) **TD EMVCC 与 TD MVCC**：检测时间是指从客户端提交交易到该交易因 MVCC 或 EMVCC 错误而被拒绝的总时长。

C. 与基线方案的比较

在本节中，我们将所提方案与 Hyperledger Fabric 2.3 版进行比较。吞吐量和交易延迟均取十次运行的平均值。

表 I：默认评估参数

参数	数值
区块大小	500 笔交易
区块时间	2 秒
组织数量	2
每个组织的节点数量	2
排序共识	Raft（3 个排序节点）
MVCC 冲突率	40%
背书策略	与(Org1,Org2)

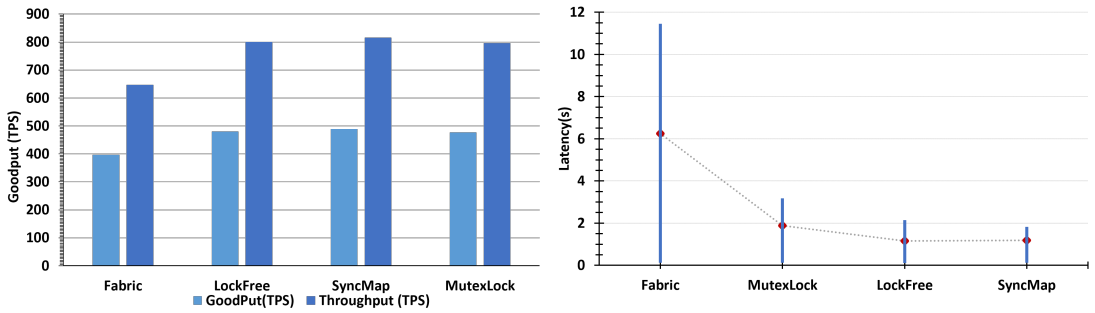
图 7a 展示了三种实现方案和 Hyperledger Fabric 的平均吞吐量。三种所提方案均优于 Fabric 基准方案，其中 SyncMap 的表现最佳，与 Fabric 相比有效吞吐量提升了 23.2%。互斥锁和 LockFree 的提升幅度分别为 20.4% 和 21.2%。此外，图 7b 展示了三种实现方案和基线的平均延迟。类似地，与基线相比，三种实现方案均降低了延迟：SyncMap 和 LockFree 的延迟降低了 80%，互斥锁则降低了 69%。这是因为在背书阶段即可早期检测到冲突交易，而无需等到验证阶段再将其中止。因此，我们得出结论：在对等节点拥有多个虚拟 CPU 时，SyncMap 是早期 MVCC 缓存实现中最适合的数据结构，因为它在读操作方面表现最佳。

D. 敏感性分析

本节分析各种网络和工作负载参数对性能指标的影响，例如冲突率、背书策略以及每个对等节点的计算资源等。

1) 冲突率的影响：冲突率是最重要的参数，能够体现所提方案的效用。图 8a 绘制了 Fabric 及我们的不同方案在不同冲突率下的平均有效吞吐量。正如预期，随着冲突率上升，有效吞吐量会下降，因为失败交易的数量不断增加；当冲突率达到 100% 时，有效吞吐量便降为零。此外，我们可以看到，针对不同的冲突率，三种所提方案的表现都优于 Fabric。其中 SyncMap 表现最佳：冲突率约为 20% 时，它能将有效吞吐量提高 10%；当冲突率达到 40% 时，其影响更为显著，提升幅度超过 20%，达到 23%。随着某个应用的冲突率上升，该方案相对于基线的性能优势也会扩大。

2）每个对等节点的计算资源的影响：在本部分中，我们将分析为每个对等节点增加或移除虚拟 CPU 的影响。我们将 CPU 数量从 2 个虚拟 CPU 调整到 8 个虚拟 CPU。图 8b 绘制了平均吞吐量——



(a) 吞吐量与基准方案的比较 (b) 延迟与基准方案的比较

图 7：与基线方案的比较

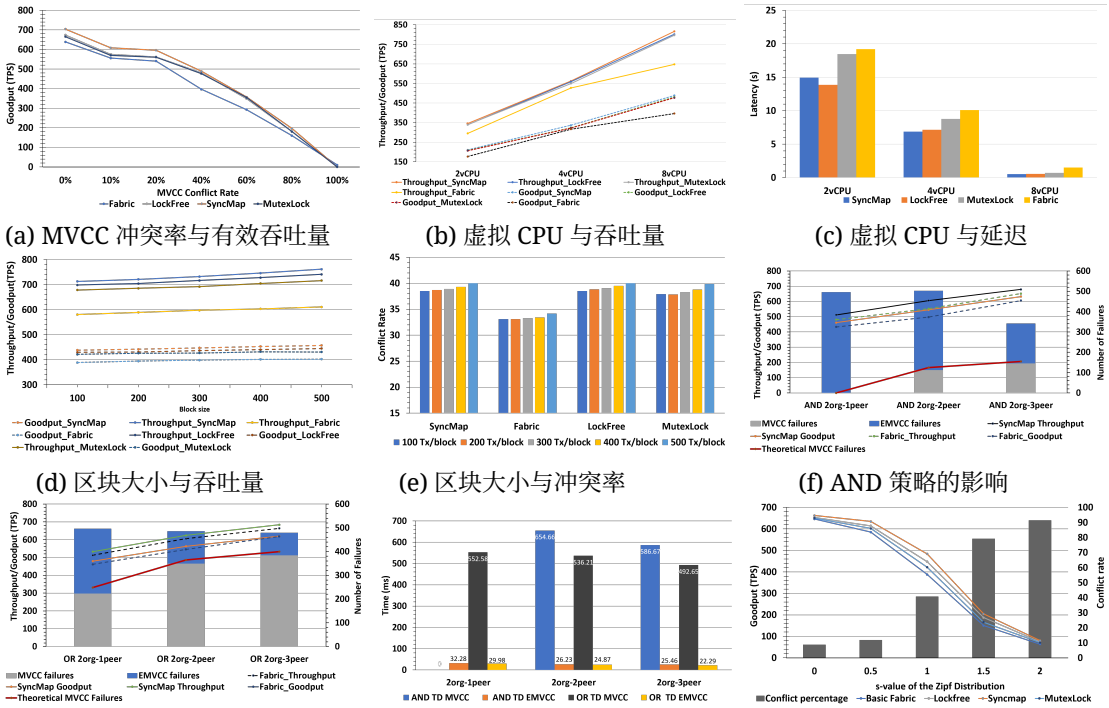
和有效吞吐量，图 8c 则绘制了三种实现方案以及 Hyperledger Fabric 基准方案在每个对等节点配置不同虚拟 CPU 数量时的平均延迟。随着每个对等节点的虚拟 CPU 数量增加，有效吞吐量上升，延迟下降。在不同的虚拟 CPU 数量下，我们的三种方案均优于基线。不过，我们观察到，在配置 8 个虚拟 CPU 时，SyncMap 与基线方案相比，有效吞吐量显著提升了 23%，延迟则降低了 65%。因此，我们得出结论：与 Fabric 基准方案相比，我们的方案能够更有效地利用额外的计算资源。

3) 区块大小的影响：我们通过将区块大小从每个区块 100 笔交易调整到每个区块 500 笔交易，分析区块大小的影响。图 8d 绘制了 Fabric 及我们的三种实现方案在不同区块大小下的平均吞吐量和有效吞吐量。对于所有方案，增大区块大小都会提高吞吐量和有效吞吐量，因为使用较大的区块会减少开销和网络通信。与 Fabric 相比，SyncMap 方法在不同区块大小下平均提升了 12%，这证实了我们提出的方案能够适应不同区块大小，具备可扩展性。图 8e 绘制了 Fabric 及我们的三种方法在不同区块大小下的冲突率。可以观察到，随着区块大小增加，冲突率也会升高：这是因为区块包含的交易数量越多，区块内冲突就越多。另一个观察结果是，Fabric 的冲突率低于我们提出的实现方案，这是因为早期 MVCC 检测器会产生一些假阳性和假阴性。

4）背书策略和网络拓扑的影响：在此实验中，我们通过将每个组织的对等节点数量从一个增加到三个，采用不同的网络拓扑。图 8f 和图 8g 绘制了在 Fabric 基线方案中，使用 AND（Org1、Org2）和 OR（Org1、Org2）背书策略时，不同网络拓扑下的平均吞吐量和有效吞吐量，以及由 EMVCC 和 MVCC 验证导致的交易失败百分比，并展示了 Fabric 基线方案与 SyncMap 方案的结果。我们观察到，背书策略的类型会影响网络性能。如这两幅图所示，OR 背书策略的吞吐量高于 AND 策略，因为使用 OR 策略时，交易所需的对等节点背书少于使用 AND 策略时。对于未检测率，这些实验结果证实了第四节中给出的理论结果。例如，使用 AND 策略，且网络中有两个组织、每个组织各有两个对等节点时，有 22% 的冲突交易未被 EMVCC 检测到，并在 MVCC 验证时被中止（假阴性）。我们还可以看到，随着对等节点数量增加，两种背书策略下的吞吐量和有效吞吐量都会提高，因为组织拥有更多资源来处理交易。我们还观察到，每个组织的对等节点数量增加会提高假阳性率。与 Fabric 相比，使用 AND 策略时的改进更为显著。然而，使用 OR 策略时，假阳性数量会增加，从而影响方案的性能。我们可以得出结论，随着背书节点数量增加，我们的方案会变得更加高效。

图 8h 绘制了采用 OR 和 AND 背书策略时，不同网络拓扑下检测 EMVCC/MVCC 的平均时间。与在 EMVCC 阶段中止冲突交易相比，将其处理至 MVCC 验证阶段后再予以拒绝，可将交易执行时长缩短 95%；因此，我们的方案显著降低了交易延迟。我们注意到，随着每个组织的节点数量增加，检测 MVCC 的时间会缩短，这得益于计算资源的可用性。

5）链码实现的影响：为了模拟另一种链码行为，我们采用 Zipf 分布，通过将该分布的参数s从 0 调整到 2，来选择用于模拟交易的键。增大s会提高对某些相同键的使用偏好，从而增加冲突率。图 8i 绘制了



(a) MVCC 冲突率与有效吞吐量 (b) 虚拟 CPU 与吞吐量 (c) 虚拟 CPU 与延迟

(d) 区块大小与吞吐量 (e) 区块大小与冲突率 (f) AND 策略的影响

(g) OR 策略的影响 (h) EMVCC/MVCC 检测时间 (i) 冲突偏斜度

图 8：敏感性分析结果

在 Zipf 分布参数s 取不同值时，Fabric、SyncMap、LockFree 和 MutexLock 方案的平均有效吞吐量。可以看出，随着s 增大，冲突率上升，导致有效吞吐量下降。正如预期，随着参数 s 增大，我们的方案表现优于 Fabric。例如，当s 等于 1 时，SyncMap 可将有效吞吐量提高 24%；当s 等于 1.5 时，则可提高 35%。

E. 研究结果总结与讨论

我们提出的三种方案均优于 Fabric 基准方案。SyncMap 是用于存储键的缓存的最佳数据结构。与 Fabric 相比，改进幅度主要取决于所采用的背书策略、网络设计以及链码实现（冲突率）。在实际场景中，当 40% 的交易因 MVCC 冲突而失败时，SyncMap 方案可将吞吐量提高 23%，并将延迟降低 80%。

使用我们的解决方案时，如果业务逻辑允许，我们建议开发者采用 AND 背书策略，并使用最多的组织数量和每个组织最少的对等节点数量。不过，如果必须采用 OR 背书策略，我们建议使用最少的组织数量和每个组织最少的对等节点数量。应谨慎调整区块大小，以尽可能减少区块间和区块内交易冲突，从而最大化有效吞吐量。对于链码实现，重要的是编写能够确保冲突率低于 40% 的链码。这些解决方案也有助于利用 EOV 方法优化其他区块链的性能。

VI. 结论

在本研究中，我们提出了一种改进 Hyperledger Fabric 性能的机制，旨在利用不同的缓存技术，提前中止那些注定无法通过验证并提交到账本的交易。在实验评估中，我们将 SyncMap、LockFree 和 MutexLock 与 Fabric 的基础版本进行了比较。我们还通过改变区块大小、背书策略和资源分配等可配置参数进行了敏感性分析。结果表明，所提方案优于 Fabric 和 SyncMap；其中，SyncMap 是表现最佳的方案，与 Fabric 相比，吞吐量最高提升了 20%，延迟最高降低了 80%。

参考文献

[1] Vitalik Buterin. 下一代智能合约与去中心化应用平台。*Etherum*，第 1–36 页，2014年。[2] Satoshi Nakamoto. 比特币：点对点电子现金系统。《科学通论哲学杂志》，39(1):53–67，2008年。[3] Zibin Zheng、Shaoan Xie、Hong Ning Dai、Weili Chen、Xiang- ping Chen、Jian Weng 和 Muhammad Imran. 智能合约综述：挑战、进展与平台。《未来一代计算机系统》，105:475–491，2020年。

[4] Elli Androulaki、Artem Barger、Vita Bortnikov、Srinivasan Mu - ralidharan, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Chet Murthy, Christopher Ferris, Gennady Laventman, Yacov Manevich, Binh Nguyen, Man- ish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukoli´c, Sharon Weed Cocco, 和 Jason Yellick。Hyperledger Fabric：面向许可型区块链的分布式操作系统。第*13*届 *EuroSys* 会议论文集，*EuroSys 2018*，2018-Janua，2018。[5] Jeeta Ann Chacko、Ruben Mayer 和 HansArno Jacobsen。为什么我的区块链交易会失败？对 Hyperledger Fabric 的一项研究（扩展版）*。(4)，2021。[6] Hagar Meir、Artem Barger、Yacov Manevich 和 Yoav Tock。Hyperledger Fabric 中的无锁事务隔离。论文集——*2019*年第二届 *IEEE* 国际区块链会议，*Blockchain 2019*，第59–66页，2019年。[7] Ankur Sharma、Felix Martin Schuhknecht、Divya Agrawal 和 Jens Dittrich。模糊区块链与数据库系统之间的界限：Hyperledger Fabric 案例 CCS 概念 •信息系统 → 分布式数据库交易；ACM 引用格式。*ACM SIGMOD* 数据管理国际会议论文集

f

数据，18:105–122，2019年。[8] Pingcheng Ruan、Dumitrel Loghin、Quang Trung Ta、Meihui Zhang、Gang Chen 和 Beng Chin Ooi。关于执行-排序-验证区块链的事务视角。*ACM SIGMOD* 国际数据管理会议论文集，第 543–557 页，2020 年。[9] Yacov Manevich、Artem Barger 和 Yoav Tock。Hyperledger Fabric 的服务发现。第 *12* 届 *ACM* 分布式与事件驱动系统国际会议论文集，第 226–229 页，2018年 5 月。[10] Per-åke Larson、Spyros Blanas、Cristian Diaconu、Craig Freedman、Jignesh M Patel 和 Mike Zwilling。面向主存数据库的高性能并发控制机制。第 298–309 页，2012年。[11] Minsu Kwon 和 Heonchang Yu。改进 Hyperledger Fabric 排序与背书阶段的性能。*2019*年第六届物联网系统、管理与安全国际会议（*IOTSMS*），第 428–432 页，2019年 –。[12] Zsolt István、Alessandro Sorniotti 和 Marko Vukoli´c。StreamChain：区块链需要区块吗？*SERIAL 2018 - 2018* 年分布式账本可扩展与弹性基础设施研讨会论文集，第 1–6 页，2018年。[13] Christian Gorenflo、Stephen Lee、Lukasz Golab 和 Srinivasan Keshav。FastFabric：将 Hyperledger Fabric 扩展至每秒 20,000 笔交易。*ICBC 2019 - IEEE* 区块链与加密货币国际会议，第 455–463 页，2019年。[14] Christian Gorenflo、Lukasz Golab 和 Srinivasan Keshav。XOX Fabric：一种交易执行的混合方法。博士研讨会，系统与网络，2019年。[15] Haris Javaid、Chengchen Hu 和 Gordon Brebner。优化 Hyperledger Fabric 的验证阶段。*2019*年 *IEEE* 第 *27* 届建模、分析与仿真国际研讨会

f

计算机与电信系统（*MASCOTS*），第 269–275 页，2019年。[16] Parth Thakkar、Senthil Nathan 和 Balaji Viswanathan。Hyperledger Fabric 区块链平台的性能基准测试与优化。第 *26* 届 *IEEE* 计算机与电信系统建模、分析与仿真国际研讨会论文集，*MASCOTS 2018*，第 264–276 页，2018年。[17] Pezhman Nasirifard 和 Ruben Mayer。FabricCRDT：一种面向许可型区块链的无冲突复制数据类型方法。第 *20* 届国际中间件会议，第 110–122 页，2019年。[18] Jose M Faleiro 和 Daniel J Abadi。重新思考可串行化多版本并发控制。*arXiv* 预印本 *arXiv:1412.2324*，2014年。[19] Ruozhou Yu、Guoliang Xue、Vishnu Teja Kilari、Dejun Yang 和 Jian Tang。CoinExpress：一种用于基于区块链的支付通道网络的快速支付路由机制。计算机通信国际会议论文集

and Networks, ICCCN ，2018年- J uly, 2018年。

[20] Andrei Alexandrescu。《无锁数据结构》。第 7 页，2007 年。[21] Canhui Wang 和 Xiaowen Chu。Hyperledger Fabric 的性能特征分析与瓶颈分析。论文集——国际分布式计算系统会议，2020 年 11 月：1281–1286，2020 年。[22] Clemens Wickboldt。基于区块链的认证存储系统基准测试。2019年。

附录

在本附录中，我们将详细介绍如何计算以下方程：

对于方程 1，假阳性可按如下方式计算：

$$Tx_invalid_D = Tx/Bloc \times \%cfct \times (1 - \mathcal{P}(ND))$$

$$Tx_passing_endor = Tx_valid + Tx_invalid_ND$$

$$FP = Tx_passing_endor \times \frac{Tx_invalid_ND}{Tx_invalid_D}$$

$$FP = \frac{Tx/Bloc \times \%cfct^2 \times \mathcal{P}(ND) \times (1-\mathcal{P}(ND))}{\%cfct \times \mathcal{P}(ND) - \%cfct + 1}$$

其中，$Tx\ invalid\ D$ 是检测到的无效_ _交易数量，$Tx_invalid_ND$ 是未检测到的有效交易数量，$Tx_passing_endor$ 是通过背书阶段的交易数量

对于公式 2，我们按如下方式计算假阳性：

$$FN = Nb_cfct_TX \times \mathcal{P}(ND)$$
$$FN = Tx/Bloc \times \%cfct \times \mathcal{P}(ND)$$

其中，Nb_cfct_TX 是发生冲突的交易数量。

对于公式 4，使用 AND 背书策略时的未检测概率定义如下：假设有两笔交易（*TX1* 和*TX2*），客户端提交*TX1* ，并由网络中每个组织的一个对等节点背书。随后提交了第二笔与*TX1* 存在 MVCC 冲突的交易*TX2* 。未检测概率是指交易 *TX2*未被曾为交易*TX1*背书的对等节点背书的概率。未检测概率等于各组织未检测条件概率的乘积。已知某对等节点属于某组织时，未检测概率为该组织中未为*TX1* 背书的对等节点数（$M_i - 1$）除以该组织的对等节点总数（M_i）。

$$\mathcal{P}(ND) = \prod_{i=1}^{N} P(ND \mid Org_i)$$

$$\mathcal{P}(ND) = \prod_{n=1}^{N} \frac{(M_i - 1)}{M_i}$$

最后，对于公式 5，使用 OR 背书策略时的未检测概率计算如下：如果所有组织的对等节点数相同，则未检测概率通过用 1 减去检测概率来计算；对于每个组织有 M 个对等节点的情况，检测概率为$frac1NM$。

$$\mathcal{P}(ND) = 1 - \mathcal{P}(D)$$

$$\mathcal{P}(ND) = 1 - \frac{1}{NM}$$

$$\mathcal{P}(ND) = \frac{NM - 1}{NM}$$

否则，如果各组织拥有的对等节点数量不同，则未检测概率的计算方法如下：将交易获得某个组织背书的概率与该组织未检测到交易的条件概率相乘，再将这些乘积求和。特定对等节点提供背书的概率为 $\frac{1}{N}$，而组织未检测到交易的条件概率，即从未为第一笔交易背书的对等节点处获得背书的概率，等价于 $\frac{1}{M_i}$。

—

$$\mathcal{P}(ND) = \sum_{i=1}^{N} \mathcal{P}(Org_i) \times P(ND \mid Org_i) \quad (6)$$

$$\mathcal{P}(ND) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{C_{M_i}^{M_i - 1}} \quad (7)$$

$$\mathcal{P}(ND) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{M_i} \quad (8)$$

综上，我们得到使用方程 5 中定义的 OR 背书策略计算未检测概率的公式。