Hyperledger Fabric:一个授权的分布式区块链操作系统

Elli Androulaki Artem Barger Vita Bortnikov

IBM

Christian Cachin Konstantinos Christidis Angelo De Caro David Enyeart

IBM

Christian Cachin Konstantinos Christidis Angelo De Caro David Enyeart

IBM

Srinivasan Muralidharan State Street Corp.

Chet Murthy

Binh Nguyen State Street Corp.

Manish Sethi Gari Singh Keith Smith Alessandro Sorniotti

IBM

颜 亮

翻译

IBM GCG

Chrysoula Stathakopoulou Marko Vukolic´ Sharon Weed Cocco Jason Yellick

IBM

概述

Fabric 是一个模块化和可扩展的开源系统，用于部署和操作授权的区块链系统(国内称为联盟链)，是由 Linux 基金会（www.hyperledger.org）托管的 Hyperledger 项目之一。

Fabric 是第一个真正可扩展的、用于运行分布式应用程序的区块链系统 。它支持模块化可插拔的共识协议，允许系统根据特定用例和信任模型进行客户化。 Fabric 也是第一个用标准通用编程语言编写的分布式区块链系统，不会对原生加密货币产生系统依赖。这与现有的区块链平台形成鲜明对比，现有的块链平台需要用特定语言编写“智能合约”或依赖加密货币。 Fabric 使用灵活轻便的 Membership 概念实现了区块链中的授权模型，可以与行业标准身份管理集成。为了支持这种灵活性，Fabric 引入了一种全新的区块链设计理念，并改进了区块链应对非确定性、资源耗尽和性能攻击的各种异常应对方式。

本文描述了 Fabric 的体系结构，各种设计的基本原理，最突出的技术实现，以及它的分布式应用程序编程模型。其中，我们通过将 Fabric 和比特币数字货币进行对比，我们得出了结论：Fabric 在一些主流的部署配置中实现了每秒 3500 个 TPS 的端到端吞吐量，并具有秒级以内的延迟，另外，peer 节点可以很好地扩展到 100 多个。

1 前言

区块链是一种运行在参与者相互不信任的分布式网络中，用于记录相互交易的不可串改的分布式分类账本。其中， 每个参与方(以下称 Peer)都维护着一个账本的副本，Peer 通过执行共识协议来验证交易，将它们分组打包为块，并在块上按顺序构建哈希链。 此过程需要通过对交易进行排序来形成分类账，这是保障交易一致性的必要条件。 区块链技术随着比特币的兴起而引起广泛关注，并被广泛认为是在数字世界中运行高可信信息交换的最有前景的技术。

在无权限控制区块链(国内称公有链)中，任何人都可以在没有明确身份的情况下参与交易。 公有链通常涉及数字加密货币，并且通常使用基于“工作量证明”（PoW）和其他经济激励的共识。 而联盟链不同，它是在一组已知的、可识别参与者身份之间运行的区块链。联盟链提供了一种机制来保护具有共同目的、但彼此不完全信任的一组实体之间的交易，例如资金、商品或信息交换。 对于一些授权的 Peers,联盟链可以使用传统的拜占庭容错（BFT）共识。

区块链可以以智能合约的形式执行任意的可编程交易逻辑，如以太坊[5]。 比特币中的脚本(Script)是该概念

Work done at IBM.

的前身。 智能合约作为受信任的分布式应用程序，它的安全性来自于区块链中众多参与者一起进行共识。 这非常类似于使用状态复制机(SMR)[48]构建弹性应用程序的方法。然而，区块链在很大程度上与传统的具有拜占庭故障的SMR 又有区别:(1)区块链由 1 个或以上的分布式应用同时运行; (2)应用程序可以由任何人动态部署; (3)应用程序代码不受信任，甚至可能是恶意代码。 因此，区块链的这些具有差异的特性需要我们对系统进行重新设计。

许多现有的智能合约都遵循 SMR[48]的标准并实施所谓的主动复制[27]：首先，共识或原子广播协议会对交易进行排序，确定后将其广播给所有参与者; 其次，每个参与者按顺序执行交易，我们称之为订 Order-Execute 架构; 它要求所有参与者必须执行所有的交易，并且这些交易都是确定性的。 Order-Execute 架构几乎可以在所有现有的区块链系统中找到，包括如公有链中的以太坊（基于 PoW 的共识）到联盟链（具有 BFT 类型的共识），如 Tendermint [14]，Chain [4]和 Quorum [13]。 Order-Execute 设计并非存在于所有系统中，这是因为额外的交易验证步骤可能会使其变得更为复杂，其局限性是与生俱来的：每个参与者会执行每个交易，而且交易必须是确定性的。

联盟链也会受到许多限制，这些限制通常源于他们的授权模式和使用 Order-Execute 架构。 尤其是：

- 共识会在平台中进行硬编码，这会很难满足现实需要，没有无所不能的（如 BFT）单一共识协议[52];
- 交易验证的信任模型由共识协议来确定，无法适应智能合约的要求;
- 智能合约开发必须是固定的某种语言，而非标准或流行的语言，阻碍了技术推广，也可能导致更多编程错误;
- 所有节点都要顺序执行任所有的交易，这种大量重复计算既浪费资源，又限制了性能，并且需要采取额外复杂的措施来防止那些源于不可信合约的异常攻击（比如 Ethereum 通过对各种操作收取一定的“Gas”来限制大量/长时间的操作，预防可能引发的异常攻击）
- 用编程的方式难以确保交易的确定性;
- 每个智能合约都运行在所有节点上，这有悖于对交易的保密性要求，而且还会禁止将合约代码和状态传播给其子节点；

在本文中，我们将会向你介绍一个开源[8]的区块链平台：Hyperledger Fabric(简称:Fabric)，它很好地解决了上述限制。 Fabric 是 Hyperledger [7]的项目之一，由 Linux 基金会[11]进行托管。 Fabric 现已是被应用在 400 多种业务原型、概念验证和生产环境下的分布式账本系统，同时也跨越不同的行业和使用场景。 这些使用场景包括但不限于：各种争议解决、物流与贸易、外汇结算、食品安全、合同管理、钻石追索、积分管理、低流动性证券交易和结算、身份管理以及通过数字货币结算等领域。

Fabric 引入了一种新的区块链架构，旨在实现更好伸缩性、灵活性、可扩展性和保密性。 是一种模块化和可扩展的通用联盟区块链，也是第一个使用标准编程语言编写(Go)的分布式区块链系统，它可以在许多节点上一致地执行交易， Fabric 已经成为了联盟链领域的第一个分布式操作系统[54]。

Fabric 的体系结构遵循一种全新的 Execute-Order-Validate 架构，用于在不受信任的分布式环境中执行不受信任的代码。 它将交易的处理分为三个步骤，这三个步骤可以在系统中的不同参与方运行：（1）执行交易并检查其正确性，从而对其进行背书（对应于其他区块链中的“交易验证”）；（2）通过共识协议进行排序，而不会考虑交易的语义内容；（3）基于特定的信任假设来验证交易，可以避免由于并发而引起的资源竞争(或重复交易)。

这种设计与 Order-Execute 架构完全不同，因为 Fabric 通常在达成排序共识之前执行交易。 它结合了两种众所周知的复制方法，即被动和主动，如下所示。

首先，Fabric 使用分布式数据库中常见的被动或主备复制[21,27]，使用基于中间件的非对称更新过程[40,41]，并移植到具有拜占庭故障的不可信环境。在 Fabric 中，每个交易仅由所有参与者的子集执行（背书节点），使用“execute-verify”的 BFT 复制[37] 解决潜在的非确定性因素。一个清晰的、可被认可的规定定义了哪些 peer 或其中有多少 peer 需要保证正确执行给定的智能合约。

其次，Fabric 包含主动复制，交易对账本状态的影响仅在对所有交易集达成排序共识后，才会被写入。 这允许 Fabric 根据对交易进行背书来处理特定的信任假设。此外，排序服务是基于一致的模块化组件（也就是原子广播）来进行共识的，这是一种无状态的过程，并且在逻辑上将参与执行交易和账本维护的 peer 进行分离。由于共识是模块化的，因此其部署可以根据特定部署的信任假设来进行定制。 虽然很容易使用区块链中的 peer 来实现共识，但两个角色的分离增加了灵活性，并

Work done at IBM.

允许我们使用完善的工具包进行 CFT（崩溃容错）或 BFT(信任容错)来进行排序。

总之，这种混合复制设计结合了拜占庭模型中的被动和主动复制，execute-order-validate 架构也是 Fabric 的主要创新之处。 他们解决了之前提到的问题，并使 Fabric 成为可扩展的区块链系统，支持灵活的信任假设。

为了实现此体系结构，Fabric 包含以下模块：

- 排序服务以原子方式向 peer 广播状态更新，并就交易顺序建立共识。
- Membership 负责将 Peer 与加密身份进行关联，确保了 Fabric 的联盟链特性。
- 一个可选的 P2P gossip service 来将排序服务输出的区块分发到各个 peers；
- Fabric 中的智能合约会单独启动一个容器进行运行，将其与其它组件进行隔离 。 它们可以用标准编程语言编写（如 GO,Node.JS），但不能直接访问账本状态。
- 每个 Peer 都在其本地维护一个 append-only 的区块链账本，并用键值对的格式来保存最新状态的快照；

本文的其余部分描述了 Fabric 的架构以及我们使用它的经验。第 2 节总结了现有技术，并解释了各种设计决策背后的基本原理。第 3 节详细介绍了 Fabric 的体系结构和 execute-order-validate 的原理，介绍了交易的执行流程。在第 4 节中，描述了 Fabric 的关键组件，特别是排序服务，会员服务，点对点广播，账本数据库和智能合约 API。在第 5 节中给出了在商用公有云虚机上进行 Fabric WAN 集群运行测试结果，描述了在此过程中，Fabric 的各项性能指标和运行结论：Fabric 在主流的部署配置中实现了吞吐量超过 3500 个 tps 的能力，最终实现了将交易延迟控制在 1 秒以内，同时，还可以很好地将 Peer 扩展到 100 多个 。在第 6 节中，我们讨论了 Fabric 的一些实际生产案例。最后，第 7 节讨论了后续的相关工作。

2 背景

2.1 Order-Execute 区块链架构

之前所有的区块链系统，都遵循订 Order-Execute 架构。 这意味着区块链网络首先使用共识协议对交易进行排序，然后依次在所有 peer 上以相同的顺序执行它们。

例如，基于 PoW 的公有链（如以太坊）采用如下步骤，将交易的共识和执行过程结合在一起:(1)每一个 Peer（即参与共识的节点）会将一组有效交易（为了建立有效性，这个 peer 已经预先执行了那些交易）进行打包;（ 2）Peer 试图解决一个 PoW 的难题[44]；(3)如果某个幸运的 peer 第一时间解决了这个问题，它会通过 gossip 协议将该块广播到网络;（ 4）所有的节点都会收到这个基于该难题解决方案的区块和区块内的所有交易。 事实上，每个 peer 会从第一步开始重复执行那个幸运 peer 的交易。 此外，所有 peer 会按顺序执行交易（包括在本区块和跨区块）。 order-execute 架构如图 1 所示。

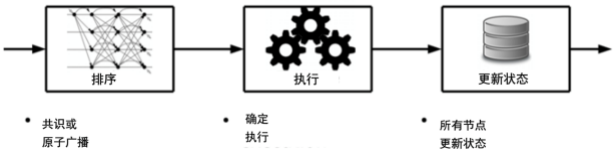


图 1：Order-execute 架构

现有的联盟链（如 Tendermint，Chain 或 Quorum）通常使用由 BPFT [26]或其他原子广播协议来实现 BFP[24]共识。 然而，它们都遵循相同的 order-execute 架构并实现经典的 Active SMR [27,48]。

2.2 Order-Execute 架构的缺陷

Order-Execute 在概念上非常简单，因此也被广泛使用。 但是，当它用于联盟链时，它有几个缺陷。我们接下来讨论其中三个最重要的缺陷：

顺序执行。 在所有 peer 上依次执行所有的交易限制了区块链系统的吞吐量。 特别是，我们往往追求高吞吐量与低延迟，这将成为智能合约的性能瓶颈(除非你是极其简单的合约)。 此外，与传统的 SMR 相比，区块链形成了一个多方参与的计算引擎，其智能合约也可能会被不受信的参与方部署。 一次 DoS 攻击就会严重降低此类区块链的性能，当然也可以简单地引入需要很长时间才能执行的智能合约。 例如，执行无限循环的智能合约具有致命的效果，而且还无法自动被检测到。

为了解决这个问题，带有加密货币的公有链可以解决执行成本问题。 例如，以太坊[58]引入了交易执行所消耗的 Gas 的概念，交易执行过程中，Gas 价格转换为加密货币的成本并计入交易的提交者（执行交易需要成本，时间越长，成本越高）。 以太坊在很大程度上支持这一概念，为每个执行步骤分配成本 。 虽然这似乎是

Work done at IBM.

公有链的可行解决方案，而对于联盟链是不够的，因为它不是一种通用的加密数字货币系统。

相对于按序执行场景，分布式系统提出了许多改进性能的方法，例如通过并行执行无关的操作[46]，然而，这些技术也可以成功应用于智能合约的区块链环境中。例如，如果一方想尝试去推断智能合约中所有的依赖关系，这在具有保密性约束的情况下尤其具有挑战性。 此外，这些技术对来自不受信任的开发人员的合同代码的DoS 攻击毫无帮助。

非确定性代码。 order-execute 架构的另一个重要问题是非确定性交易。 在 Active SMR 中达成共识后执行的操作必须是确定性的，所有参与方都保持相同的状态。 这通常需要编写特定语言（例如，以太坊 Solidity）来解决，这对于其应用程序而言可行， 但只限于确定性的执行。 然而，这些语言学习和使用都比较困难，需要程序员进行额外的学习。 用通用语言（例如 Go，Java，C / C ++）编写智能合约会更具吸引力，还能加速区块链解决方案的推广和应用。

然而不幸的是，通用语言在确保确定性执行方面存在许多问题。 即使应用程序开发人员没有引入明显的非确定性操作，隐藏的实现细节也可能具有相同的破坏性（例如，map 迭代器在 Go 中不是确定性的),更糟糕的是，在区块链上，创建确定性应用程序的风险取决于可能不受信任的程序员。 只有一个具有恶意意图的非确定性合约就足以使整个区块链停止运行。 有调查显示可以在区块链上过滤掉一些操作[23]，但在实践应用中代价高昂。

执行的保密性。 根据公有链的普遍愿景，授限系统会在所有节点上运行所有智能合约。 然而，这些区块链有许多层面还是需要具有保密性（即限制对合约逻辑、交易数据或账本状态的访问权限）。在数据加密的角度，尽管高级零知识证明[18]等加密技术可以帮助实现保密性，但这通常会带来相当大的开销，并且在实践中不可行。

幸运的是，它可以将相同的状态广播给所有参与方，而不是在其上运行相同的代码。 因此，智能合约的执行范围可以限制在该任务所信任参与者的子集，并保证执行的结果。 这种设计从主动复制转向被动复制[21]，更加适用于区块链的信任模型。

2.3 架构的更多不足

固化信任模型。 大多数联盟链依赖于异步 BFT 复制协议来建立共识[57]。 这样的协议通常依赖于安全性假设，即在 n> 3f 个节点中(n 为总参与节点，f 为异常节点)，可接受 f 个被容忍的异常行为而表现出所谓的拜占庭故障[20]。在相同的安全性假设下，相同的节点也经常执行应用程序（即使实际上可以将 BFT 执行范围缩小到更少的对等体[59]）。 然而，无论这些节点在系统中的角色如何，这种定量信任假设可能与智能合约执行所需的信任机制不匹配。 在一个灵活的系统中，不应该在应用程序级别信任协议级别的信任。 通用区块链应该将这两个假设分离，并为应用提供灵活的信任模型。

硬编码共识。 Fabric 是第一个引入可插拔共识的区块链系统。在 Fabric 之前，几乎所有区块链系统(无论是否联盟链)都带有硬编码的共识协议。然而，几十年来对共识协议的研究表明，这种“一刀切”的解决方案是不行的。例如，在差异很大的环境中部署 BFT 时，它们的性能差异也很大[52]。虽然具有“链码”通信模式的协议在具有对称和相似链路的 LAN 集群上运行时，可以表现出可验证的最佳吞吐量[34]，但在广域或异构网络上则很差。此外，在所部署的环境中，外部条件可能随时间而变化(如系统负载、网络参数、实际故障或攻击)。由于这些原因，我们应该对 BFT 共识进行重新设计，而尽可能理想地适应不断变化的环境[17]。另一个重要方面是将协议的信任假设与给定的区块链部署方案相匹配。实际上，我们可能希望将基于替代信任模型（例如 XFT[43]）或 CFT 协议（例如 Paxos / Raft [45]和 ZooKeeper [36]）或甚至用开放协议替换 BFT 共识。

2.4 使用 Order-Execute 区块链的经验

在实现 Fabric 的 execute-order-validate 架构之前，我们也有过在 order-execute 模型中构建联盟链平台的经验(共识采用 PBFT [26])。 也就是说，早期版本的 Fabric（2016 年 9 月发布的 v0.6）是按照传统的 order-execute 架构进行设计的。

从我们许多 POC 的经验来看，这种方法的局限性变得越来越明显。 例如，用户经常在节点中察觉到异常状态，并报告共识协议中的 Bug, 往往在这种情况下，如果你仔细检查，就会发现罪魁祸首是非确定性交易代码。 另外，我们已经了解到区块链系统的关键属性，即一致性、安全性和性能，这些属性不得依赖于其用户的

Work done at IBM.

知识和能力，特别是因为区块链本就应该在不受信任的环境中运行。

3 架构

在本节中，我们将介绍三阶段 execute-order-validate体系结构，然后解释其交易执行流程。

3.1 Fabric 概述

Fabric 是用于联盟链的分布式操作系统，其执行是以通用编程语言（例如，Go，Java，Node.js）编写的分布式应用程序。它专注于在 append-only 复制账本体系中安全地跟踪执行的历史过程，没有加密货币功能。

Fabric 引入了 execute-order-validate 区块链体系结构（如图 2 所示），并不遵循标准的 order-execute 设计，原因在第 2 节中会有说明。简而言之，Fabric 的分布式应用程序由两部分组成：

一种名为 chaincode 的智能合约，它是实现应用逻辑并可运行的程序代码。 chaincode 是 Fabric 中分布式应用程序的核心部分，也可能是由不受信任的开发人员编写。 用于管理区块链系统和参数的特殊 chaincode，统称为系统 chaincode（第 4.6 节）。

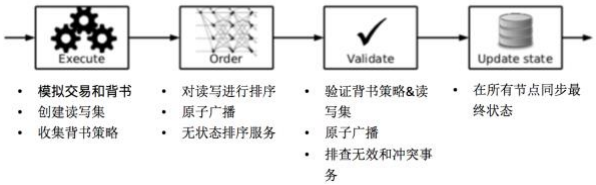


Figure 2: Fabric 的 Execute-order-validate 架构 (rw- set 意为读写集).

在 validation 阶段进行评估的背书策略(endorsement policy)。 不受信任的应用程序开发人员无法选择或修改这个背书策略。 一个背书策略是对 Fabric 中交易进行验证的静态库，它只能通过 chaincode 进行参数化。 只有指定的管理员才有权通过系统管理功能对背书策略进行修改。 一个典型的背书策略必须由 chaincode 所指定交易的背书节点参与; 可使用逻辑表达式来表示，例如“三个或多个”或“(A∧B)∨C”。自定义背书策略可以实现任意业务逻辑（例如，我们在第 5.1 节中类似于比特币的加密货币 Fabcoin）。

客户端将交易发送给由背书策略指定的节点，每个交易会由这些特定的 peer 执行，并记录其输出; 这一步也称为背书(endorsement)。 执行完成后，交易进入排序阶段(ordering phase)，该阶段使用可插拔的共识协议将排序后的所有有效交易打包分组，并按序生成区块，最后通过 gossip(可选)广播给所有节点。 这和标准的全量交易输入排序体系中的主动复制[48]不同，Fabric 在执行阶段，排序的交易输出会与状态依赖性相结合，每个 peer 节点会验证来自背书交易的状态变化以及在 validation 阶段中执行的一致性。 所有 peer 以相同的顺序验证交易，并且这个过程是确定性的。 从这个意义上讲，Fabric 在拜占庭模型中引入了一种新的混合复制模型，即，结合了被动复制（状态更新的预先共识模拟）和主动复制（执行结果和状态变化后的共识验证）。

Fabric 区块链由同一组网络中的节点组成（参见图 3）。 在 Fabric 联盟链中，网络中的所有节点都具有特定的身份，所有的身份信息由 membership service provider(MSP)提供和管理（第 4.1 节）。 Fabric 网络中的所有节点都是以下三种角色之一：

客户端(Clients)负责提交需要执行的交易，在 execution 阶段进行协调，并将交易广播给排序服务。

参与者(peers) 执行交易提案并进行验证。 所有参与者都共同维护一个分布式账本，这是一种用哈希链记录所有交易和账本状态的数据结构，描述了一个分布式账本的最新状态 。 并非所有peers都会执行所有交易，只有它们中的一部分称为endorsing peers 的背书节点才执行，背书节点在chaincode中所指定 。

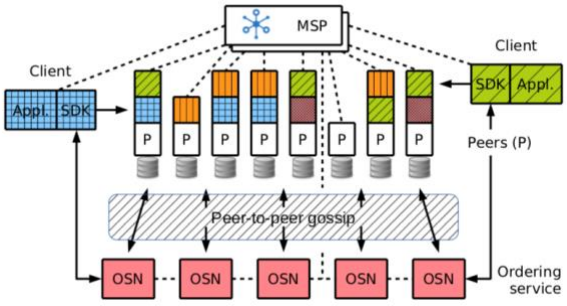


图3.基于MSP下运行多节点多chaincodes的执行架构

排序节点(OSN)，也可简称 orderers，所有 orderers 共同组成一项排序服务。 简而言之，排序服务对 Fabric 中所有交易安排统一的执行顺序，每个交易包括在执行阶段的状态更新和各种依赖性操作，也包括背书节点的加密签名(私钥) 。排序节点完全不了解应用程序的状态，也不会参与执行或验证交易。 这种设计使 Fabric 中的共识尽可能模块化，并简化了 Fabric 中的共识协议。

在一个 Fabric 网络中，支持多个区块链同时连接到同一个排序服务中，这样的区块链被称为 channel，

可能由不同的 peer 构成。 channel 可用于划分区块链网络的状态，但不协调跨 channel 的共识，并且每个 channel 中的交易总顺序与其他 channel 进行隔离。同一个 channel 的 peer 被视为受信任的成员，当然也可以进行访问控制。 在下文中，我们只专注于单个 channel。

在接下来的三个部分中，我们将解释 Fabric 中的交易（如图 4 所示）执行流程，并说明执行、排序和验证阶段的步骤。 然后，我们会总结 Fabric 的 CFT 和 BFT 模型（Sec.3.5）。

3.2 Execution 阶段

在 Execution 阶段，客户端(Client)签署并将交易提议（或简称为 proposal）发送给一个或多个背书节点以便执行。 回想一下，每个 chaincode 都会通过背书策略隐含地指定一组背书节点。 一个提案包含：提交客户端的身份（MSP 提供的）、执行交易时的参数和 chaincode 标识，每个客户端仅使用一次的随机数（例如作为计数器或随机数）加上客户端标识符来对交易进行标识。

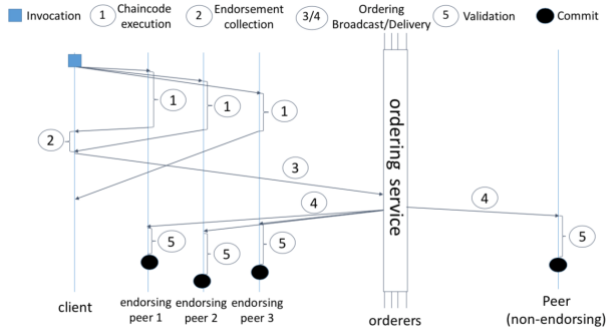


图4: Fabric单个交易执行流程

背书节点会模拟执行提案，具体内容已由安装在区块链上的特定chaincode所设计，此时系统会启动一个专门的Docker容器运行chaincode，与主要背书进程相隔离。

背书节点在其本地账本上会对提案进行模拟执行，此过程不会与其他 peer 同步，是一个独立的过程。 此外，背书节点不会将模拟执行的结果保留到账本中。区块链的状态由 peer transaction manager(PTM)以版本化的键值对存储的方式进行维护，其中对密钥的连续更新只通过增加版本号（第 4.4 节）来实现。由 chaincode 创建的状态仅限于该 chaincode，不能由另一个 chaincode 直接访问。 另外还要注意，chaincode 不应该维护程序代码中的状态，只能维护它在使用 GetState，PutState 和 DelState 对区块进行操作的状态。 在给定适当的许可的情况下，chaincode 可以调用另一个 chaincode 来访问同一 channel 内的状态。

作为模拟执行提案后的运行结果，每个背书节点会产生一个写入集(writeset)，包括由模拟产生的状态更新（即，修改 key 及其新的 value)，以及模拟执行提议的版本依赖 (即,在模拟期间所读取的所有 key 及其版本号). 在模拟执行提案之后，背书节点会以加密方式签署一个名为“endorsement”的消息，其中包含 readset 和 writeset（如交易 ID,背书节点 ID 和背书节点签名之类的数据），并在提案响应中将其发送回客户端。 客户端收集所有的背书结果，直到他们满足 chaincode 策略中所认可结果（参见第 3.4 节）。 特别是，这要求由 chaincode 策略所指定的所有背书节点产生相同的执行结果（即,相同的 readset 和 writeset)然后，客户端会将交易提交给排序服务进行排序。

为什么要这么设计。 由于背书节点在不与其他节点同步的情况下模拟提案，双方可以在各自账本的不同状态下执行它并产生不同的输出。 对于要求多个背书节点产生相同结果的要求，这意味着在使用具有高度竞争性的同一 key 的情况下，一个客户端可能无法满足背书策略的要求 。与通过中间件同步复制数据库中的主备复制相比，假设没有一个 peer 在可信区块链中正确地执行，这是一个新的考虑因素[40]。

我们有意识地采用了这种设计，因为它大大简化了架构，也适用于典型的区块链应用。 像比特币那样通过分布式应用程序，在正常情况下减少或完全消除访问相同状态的资源冲突（例如，在比特币中，修改相同“对象”的两个操作是不允许的，它代表双重支付[44]）。在未来，我们计划在 Fabric 中逐步增强对处理资源竞争情况下的能力，特别是支持 CRDT [51]以补充对当前版本依赖性检查，以及作为每个 chaincode 的交易序列 。

在排序之前容忍执行非确定性的 chaincode 交易是很重要的（另请参见第 2 节)。Fabric 中运行非确定性的 chaincode 只会危及其自身运行，例如，客户端可能难以收集足够数量的背书结果。 与 order-execute 架构相比，这是一个基本优势，其中非确定性操作会导致各 peer 状态的不一致。

最后，允许执行非确定性方案还解决了对来自不受信任 chaincodes 的 DoS 攻击，因为如果它识别到

DoS 攻击，背书节点就可以根据本地策略非常简单地中止其执行。 这还不会危及系统的一致性，但是在 order-execute 架构中，这种单方面的中止执行是不可能的。

3.3 Ordering 阶段

当一个客户端收集足够多的有效背书结果时，会将交易打包提交给排序服务，打包内容包括交易的各种配置信息(如运行参数，chaincode 操作，交易元数据以及各背书节点的情况)，排序阶段主要针对每个 channel 范围内所有交易。换句话说，通过向背书节点进行原子广播的方式，就算存在失效的排序节点，也能对交易建立有效的共识 。此外，排序服务会将多个交易打包成块，并输出包含交易块的哈希序列。 这种将交易分组或分批处理的方式，可以提高区块的广播的吞吐量，这是 CFT 中的普遍做法 。

在技术细节上，排序服务的接口仅支持两种操作：由 peer 进行调用或由合法的 channel 进行隐式参数化调用:

- 广播(tx)：客户端调用这个服务对任意交易(tx)进行广播，tx 包含了交易的各种配置、客户端的签名，以便于进行有效广播；
- B ← deliver(s) 客户端通过一串序号(s)对一个区块(b)进行检索，则该区块(b)包括交易列表[tx1, . . . , txk]和一串哈希值 h(前一区块 s-1 的哈希结果)。客户端可以多次进行检索调用，如果总能返回相同的块时，我们可以说该 peer 产生了一个以 s 作为序号的区块 B。

排序服务确保在一个 channel 上所交付的所有区块完全有序。 更具体地说，排序服务确保了每个 channel 的以下安全属性：

协议：对于任意两个区块 B 和 B’ ，B 用序号 s 进行标识，而 B’是在 peer 上用 s’进行标识，我们可以一致认为 s=s’,B=B’；

Hash 链的完整性：如果某些有效的 peer 创建了一个用 s 为序号的区块 B，并且其他 peer 也创建了一个用 s+1 为序号的区块 B’ =([tx1,...,txk],h’)，那么 h’ = H(B),其中 H(.)表示加密的 Hash 函数。

序号不跳跃：如果一个有效的 peer 节点 P 创建了一个用 s>0 为序号的区块，如果 i=0,……s-1,那么 P 肯定已经创建了 i 序号的区块。

无创建：当某个有效的 peer 创建了一个 s 为序号的区块 B，那么在此之前，客户端已经对区块 B 中的事物 tx 进行了原子广播。

排序服务必须支持以下“最终”属性：

有效性：如果一个客户端对 tx 进行了广播，则每个 peer 最终会创建将 tx 打包的区块，并用一系列序号对这些区块进行标识 。

然而，每个独立的排序可以按照客户端的要求进行活跃性和公平性保证。

在区块链网络中可能会存在大量的 peer 节点，但只有相对较少的节点用来实现排序服务(排序节点)，Fabric 可以使用 gossip 服务从排序节点向所有 Peer (第 4.3 节)广播区块。 gossip 是可以动态扩展的，并且与排序服务的特定实现无关，可以同时适用基于 CFT 和 BFT 排序服务，确保 Fabric 的模块化。

排序服务还可以执行权限校验，以查看 peer 节点是否有权广播信息和在指定的 channel 内接收区块。排序服务的这一功能和其他功能将在第 4.2 节中进一步说明。

为什么要这么设计。有一点非常重要：排序服务不会维护区块链的任何状态，既不验证交易也不执行交易。这种架构是 Fabric 的一个关键特性，它使 Fabric 成为第一个可以完全将交易执行和验证进行分离的共识区块链系统 ，让共识尽可能模块化，从而丰富排序服务的共识技术生态(排序共识选择多了)。 完整的 Hash 链使每个 peer 节点对有序区块进行完整验证时更加有效。 最后，请注意我们不要求排序服务来防止重复交易。 这一点虽然简化了它，但这不是问题，因为在 validation 期间，peer 在读写检查时会过滤掉重复的交易。

3.4 Validation 阶段

区块可以通过排序服务或 gossip 直接传递给其他 Peer。 之后，这个新的区块将进入 validation 阶段，该阶段包含三个连续步骤：

1) 根据背书策略对区块内的所有交易进行验证 ,是 validation system chaincode (VSCC) 的任务，VSCC 是一个静态库，是区块链配置的一部分，负责对 chaincode 的所有配置和策略进行验证(参见 Sec4.6)。如果不符合要求，则该交易会被标记为无效交易，其直接被忽略。
2) 对区块中的所有交易按顺序进行读写检查。对于每个交易，它将 readset 字段中的 key 的版

Work done at IBM.

本与 peer 所在账本中的进行比较。 如果版本不匹配，则该交易被标记为无效，其直接被忽略。

3) 最后就是更新账本状态，新区块会在每个 peer 进行创建，并更新区块链状态，而且前两个步骤中的有效性检查结果也会保留，用来表示区块内的有效交易。 这有利于后续进行状态重建。此外，所有状态的更新是通过将 writeset 中的所有键值对信息写入本地 。

Fabric 中默认的 VSCC 可以在一些配置有 chaincode 的 Peer 上执行简单的逻辑表达式校验，并且通过对交易所认可的有效签名表达式进行校验以验证其是否有效，不同的 VSCC 策略可以静态配置。

为什么要什么设计? Fabric 的账本包括了所有的交易(当然也包括那些无效的交易) ，这是一个整体的考量，因为排序服务是在无法获知 chaincode 状态的情况下产生区块的，而 validation 阶段只是在所有 peer 达成共识后进行的。这一特性可以满足后续对无效交易场景的跟踪，这与其他区块链(如比特币和以太坊)形成对比，这些账本仅包含有效的交易。Fabric 的天然属性是联盟链，它很容易防护客户端使用大量无效交易来进行 DOS 攻击(比如将其列入黑名单),另外，一个特定的 chaincode 部署通过对调用进行收费，从而实现计费的场景，这样 DOS 攻击成本也会随之增加。

3.5 CFT&BFT

Fabric 可以很好而灵地处理信任和错误假设问题，通常来说，每一个 Client 都可以被视为一个有潜在恶意或拜占庭节点，Peer 被分配到具体的组织中，成为一个单独的信任域，使得 peer 在组织内是可信的，但在组织外不是，而排序服务将所有的 peers(client)视为潜在的拜占庭节点。

Fabric 网络的完整性完全依赖于排序服务的一致性，排序服务的信任模型直接取决于其部署的情况(参见第 3.3 节)，从 1.0.6 版本开始，Fabric 支持集中和单点部署、在开发和测试环境部署、在集群环境中使用 CFT 排序服务，以及基于 SMaRt [19]原理的，可容忍多达三分之一排序节点异常的拜占庭 [53]场景。

需要注意的是，Fabric 将应用程序和共识模型进行了分离，使得分布式应用程序可以定义自己的信任模型，并通过背书策略进行传递，而且独立于排序服务(参见 3.4 节)。

4 Fabric 的组件

Fabric采用go语言编写，通过gPRC框架在客户端、peers和排序节点之间进行通信，主要组件如下：

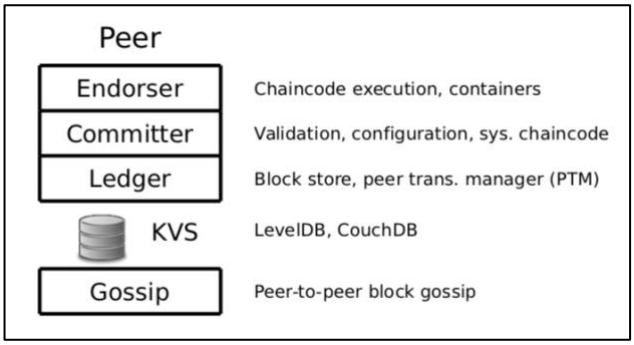


图5：Fabric peer的组件

4.1 Membership 服务

MSP 主要负责维护系统中所有节点(client,peers 和 OSNs)的身份,负责颁发用于身份验证和授权的凭证。Fabric 是联盟链，因此，所有通信的节点必须使用数字签名的方式进行身份的验证。Membership 服务可以在每个节点进行交易及完整性验证、签名和验证背书节点，以及验证其他区块操作。秘钥管理和节点注册工具也是 MSP 的一部分。

MSP 是一种针对不同实例化的抽象实现，默认基于标准 PKI 来实现数字签名，并且可以和 CA 系统进行集成。Fabric 提供一个标准和独立的 CA 服务-Fabric-CA。特别是，在执行交易过程中，它依赖于匿名的凭证来对用户授权，而不是将其授予明确的身份。

Fabric 存在两种模型来配置区块链网络：一种是 offline 模式，CA 统一分发数字证书并且分发到所有的节点，peers 和 orderers 只能在 offline 模式下注册(固定签名秘钥)。对于客户端，Fabric-CA 提供在线模式分发加密证书(随机的交易秘钥)。MSP 必须在所有节点中对身份的有效性与重复性进行识别。

MSP 还具有身份联邦的能力。比如：当多个组织参与区块链网络时，每个组织可以向组织内的成员发放证书，每个 peer 也认可所有组织下的成员(CA 发放根证书给组织，组织自己发放证书)，这一过程可以提供多个 MSP 实例进行实现，比如在每个组织和 MSP 直接建立映射。

4.2 排序服务

排序服务管理多个 channel，它在每个 channel 中提供如下服务：

1) 原子广播并对交易进行排序；
2) 查询配置 channel。成员广播一个针对 channel 的配置交易时，对 channel 进行配置；
3) 访问控制(可选)。作为一个可信的主体，对交易广播和节点接受特定区块进行控制。

排序服务在 system channel 中用初始区块进行引导，该区块包含了用于定义排序服务属性的配置。

目前针对 OSNs 的各种操作配置是在系统 channel 中进行实现。精准的原子广播功能是由一个 Kafka 实例实现的，该实例以 ZK 为框架，提供一个可伸缩的、基于消息发布订阅的并且可以有效实现 CFT 保障的框架。Kafka 可以和 OSNs 分开部署，后者充当 peers 和 Kafka 的代理。

OSN 直接将新接收的交易进行原子广播(发给 Kafka)，OSN 依据原子广播对交易进行打包，满足以下三个条件即可打包一个区块(译者注:OSN 既是 Kafka Topic 的生产者，也是消费者)：
（1）交易数已达到指定数值；（2）该区块链大小已达到最大值；(3)距上一个区块生成，达到指定的时间。如下所述：

打包是一个有很强确定性的过程，会在所有节点创建相同的区块，这一过程在实现前两个条件很容易(交易数或区块大小达标)，那针对第三个条件(时间)，为了确保在距离上一个区块产生时间后，按时间间隔产生区块，系统采用的策略是：一个节点会在广播来的区块中的读取第一个交易后启动一个计时器，如果没有打包并且超时，则 OSN 会广播一个 time-to-cut 的事物到该 channel 中，该交易会指定一个连续的区块编号用来标识该区快。另一方面，每个 OSN 也会在接收到 time-to-cut 事务后，用指定的序号立刻打包一个新区块。此交易会自动提交给所有连接的 OSN，他们都有包含相同交易列表的区块，OSN 会将最近接收到的一系列区块直接写入到它们的文件系统。所以，OSN 还可以为 peers 重建区块(译者注：peers 恢复或新成员加入)。

基于Kafka的排序服务是三种实现的方式之一。Solo是运行在开发和单节点环境下的方式，PBFT SMaRt [19]也是一种可用的方式。这就是Fabric中共识模块可插拔的特性。

4.3 Peer Gossip

将 execution, ordering 和 validation 进行分离的好处是可以将他们进行独立扩展，但是大多数基于 CFT 和 BFT 的算法会受限于网络带宽，因此，排序服务的吞吐量也会受到其节点网络的限制。通过增加更多的节点无法对共识进行扩展[28,57]，相反会降低其吞吐量。但是，由于排序和验证是分离的，可以在排序之后将执行结果有效地广播到所有节点以进行验证。将状态同步到新加入节点和长期离线节点，这正是 Gossip 的的目标。Fabric gossip 以此为目的进行多方广播，这些区块由排序服务进行标识的，这意味着所有 peer 都可以接收所有的区块(channel 内)，并且对其进行独立打包和完整性验证。

Gossip 基于 gRPC 进行通信， 采用 TLS 将所有节点进行身份识别以实现多方认证。gossip 系统中维护了一个关于在线 peers 的 membership 视图，所有 peers 都可以根据 membership 定期地将此视图在本地进行重建。此外，peer 可以在宕机和网络中断后，可以重新连接到该视图。

Fabric gossip 采用两种机制进行信息同步，第一步：Push，每一个 Peer 会从 membership 视图中随机选择一组活动的节点，将信息发给对方(给别人推送数据)。第二步：Pull，每一个 peer 会周期性地随机选择一组 peers 对丢失信息进行同步（给自己拉取数据），有研究表明[29,38]：使用这两种机制可以很好地在有限带宽的网络中，确保所有 Peers 高效的接收数据。为了减少从排序节点在发送数据造时占用太多网络资源，该协议支持以一个 leader 节点来从排序服务中提取区块并启动 goosip 分发，另外在 leader 节点失效时具有很好地容错性 。

4.4 账本

每一个 peer 都有自己的一个账本，并通过持久化的存储对状态进行记录。让后续的模拟交易、验证和账本更新成为可能。广义上讲，它由一个区块存储和 peer 交易管理器(PTM)组成。

账本以文件的形式对交易进行持久化存储，由于每个区块被按序编号并且不可改变，这种 append-only 的结构可以实现最大的性能。另外，区块会存储索引，用于对区块或区块内交易进行随机访问。

PTM 使用键值对的存储保存了最新的状态，以(key,val,ver)的格式保存了任意 chaincode 中所定义的 key 的内容，val 为要 update 的值，ver 为 query 到的值。版本由区块序号和区块内交易序号构成。这使得版本是唯一并且递增的。PTM 使用本地键值对存储来对版本变化进行存储，这个存储系统可以是 LevelDB 和 CouchDB.

在模拟交易期间，PTM 为交易的最后状态创建一个稳定的快照，如 3.2 节所述，PTM 会创建一个 readset 元组(key ver)用来响应所有 getstate 的请求，还会创建一个 writeset 元组（key,var)来记录交易进行 PUTSTATE 操作的值。另外，PTM 支持范围查询，并为此查询结果进行 hash 加密,并将查询结果和 hash 添加到 readset(译者注：如果有前置交易或其他原因修改了当前状态，则在验证阶段 Hash 对比将不通过，交易无效).

对于交易验证(3.4 节)，PTM 会按顺序验证区块内所有的交易，检查交易是否与前置(本区块内或更早)交易有冲突，对于 readset 中的任何 key，如果 readset 中记录的版本与最新状态的版本不同(假设提交了有效的前置交易)，则 PTM 将该交易标记为无效(避免重复交易)。对于指定范围的查询，PTM 会重新执行查询，并将 hash 值与 readset 中的 hash 进行比较，以确保不会产生脏数据。

在对账本进行更新期间，可以容忍 peer 突然崩溃的场景。在重新接收一个新区块后，由于 PTM 会如 3.4 节提到的那样，已经对区块进行了验证，并将其标识为有效或无效，此时就可以将区块写入到账本磁盘存储中，然后重新对区块存储索引进行更新。此时，PTM 依据有效交易中所有的 writeset，对账本进行了改变，最后，会计算和序列化一个 savepoint 值，用来表示最大成功创建的区块标号(译者注：类似于比特币区块高度)，savepoint 值用来在节点崩溃时，从持久区块中恢复索引和最新状态。

4.5 Chaincode 执行

Chaincode 在节点以松耦合的方式执行，目前支持 Go，Java 和 Node.JS(译者注：Fabric 1.2.0 后不再支持 Java)。

每一个用户或应用层面的 chaincode 会以一个单独的 docker 容器进行运行，将 chaincode 和其他组件进行隔离，这样做可以简化对 chaincode 生命周期的管理(启动，停止，卸载)。chaincode 和 peer 采用 gRPC 进行信息交互，通过这种耦合的方式，peer 无需关心 chaincode 由何种语言编写。

与应用级 chaincode 不同，系统 chaincode 是在 peer 中直接运行的，可以实现 Fabric 所需的特定功能，也可以在用户 chaincode 过度隔离的情况下使用。有关系统 chaincode 的更多详细信息，请参见下一节：

4.6 配置与系统 chaincodes

Fabric 通过一种特殊的 chaincode 即 system chaincode 对 channel 进行配置。

每个 channel 都会组成一系列逻辑区块，channel 的配置信息元数据是保存在一个特殊的配置区块中，每个配置信息区块包含 channel 的所有配置信息，但是不包含任何交易信息。channel 从存放有配置信息的原生区块进行系统引导，这些配置信息指定了 channel 的如下配置信息：（1）定义 MSPs 节点，（2）OSNs 的 IP（3）共识部署的相关信息以及排序服务的信息，如打包的区块大小和超时参数，（4）针对排序服务操作(如广播)的规则管理（5）定义 channel 可配置范围的规则。

针对 Channel 的更新必须提交一种特殊的交易：channel 配置更新交易，这种交易包含具体更新的内容以及一系列签名，排序节点会通过使用当前的配置来对更新内容和签名进行验证，然后排序服务会将其打包成一个新的区块，将新的配置信息嵌入到其中，peers 在接收这个区块时，会跟进当前配置信息进行授权验证，通过则更新本地当前配置。

应用层面的 chaincode 部署会涉及到背书系统链代码(ESCC)和验证系统链代码(VSCC)，ESCC 的输出可以是 VSCC 输入的一部分，ESCC 将提案和提案的模拟结果作为输入，如果结果令人满意，则 ESCC 会产生一个包含本次背书结果的回复。对于默认的 ESCC，这个背书过程只是每个 peer 对自己签名. VSCC 将一个交易作为输入，然后输出交易是否有效。对于默认的 VSS，会根据 chaincode 所定义策略来对的背书过程进行收集和验证。system chaincodes 还有其他的功能，如 chaincode 的生命周期管理。

5 系统评估

目前来说，Fabric 还是一个未经严格性能调整和优化的系统，但我们将会在本节中阐述一些初步的性能数据。 Fabric 是一个复杂的分布式系统，它的性能取决于许多参数的设置，包括分布式应用程序和交易量的选择，排序服务和共识实现及其参数的设置，网络中节点的网络参数和拓扑情况，节点所运行的硬件配置，节点和 Channel 的规模，以及一些更高级的配置参数。 因此，针对 Fabric 全面的性能评估工作，将会是我们未来工作的一部分。

在缺乏区块链标准的情况下，我们基于目前一些典型的区块链平台来对 Fabric 进行评估参照。这是一种基

Work done at IBM.

于比特币数据模型的加密数字货币，我们称之为FabCoin(以下简称为 Fabcoin), 这样我们就可以将Fabric 的性能和其他联盟链平台进行对比（通常来自于比特币和以太坊）,当然也可以被其他联盟链用来进行测试参照。

在以下章节中，我们首先会介绍 Fabcoin，以及演示了如何在 validation 阶段和背书策略进行客户化。在 5.2 节中，我们将提交我们的测试基准并对测试结果进行讨论。

5.1 Fabric Coin

UTXO 加密数字货币。这是一种在比特币上实现的经典模型，称为“未消费的交易输出”或 UTXO，同时也被其它加密数字货币和分布式应用广泛采用。基于 UTXO 框架，数据对象的每次演变过程都会将其记录在彼此隔离的账本状态中。这种状态由某个交易所产生，必须在后续一个唯一的交易进行消费。每一笔交易会基于一窜序列进行输入，同时产生一个或多个输出。比特币中，一个币最开始是通过对矿工进行挖矿奖励所得，在账本中会标识该币的拥有者。任何一个币都可以通过交易重新分配新的拥有者，这时，该交易会消费当前拥有者的状态，并且重新创建新拥有者的币状态。

在 Fabric 中，我们基于键值对的存储方式，沿用了 UTXO 模型：每一个 UTXO 状态会对应一个唯一的 KV 值，创建时币的状态为“未消费”，销毁时的状态为“消费”。同样，每一个状态都可以视为标识有逻辑版本的个体(从 V0 开始)，如果被消费则标识为 V1，对状态的并发修改被认为是非法的（例如：试图用不同的方式对币状态进行修改会造成”重复支付”）

UTXO 模型中的值是通过交易中的各种输入状态进行传输的，谁是状态的拥有者，则其公钥必须被写入其中，每一笔交易员创建一个或多个基于 KV 的状态输出，并且在 KV 中消费状态输入，并确保输出状态中值得总和等于输出状态的总和。同时也有一些策略来决定值是如何创建（比特币种的挖矿或其他系统中的特殊途径）和消费的。

在 Fabcoin 中，每一个状态会用如下格式描述：(key,val)=(txid.j,(amount,owner,label)). 针对某个交易的第 j 个输出，会用新所有者的公钥对其进行标识，并附上 Lable，Lable 用来标识币的类型(如”USD”,”EUR“)。交易标识是一串用来标识 Fabric 交易的的短值。Fabcoin 系统包括三个部分：（1）客户钱包；（2）Fabcoin 的链代码；（3）针对 Fabcoin 背书策略进行定制的 VSCC。

客户钱包。默认情况下，每个 Fabric 客户端会在本地维护一个由一组秘钥构成的客户钱包，以便客户消费这些币。每次创建消费交易都会涉及到一个或多个币，客户端钱包会创建一个 Fabcoin 的请求：Request=(inputs,outputs,sigs)包括：（1）用户想消费的币状态的输入列表，inputs=[in,……]（2）币状态的输出列表 outputs = [(amount, owner, label), . . .]. 客户钱包会将输入的币状态用私钥进行签名。Fabcoin 会将其进行串联并附上一系列的签名(这是每个 Fabric 交易的一部分)。当输入币状态的金额总和等于输出币总和时，消费的交易才有效。当用 mint 交易来创建新币时，输入只有中央银行（CB）中的一个身份（公钥），而输出则包含任意数量的币状态。为了保证其有效性，MINT 交易的签名必须是属于 CB 下的公钥。Fabcoin 可以配置到多个 CBs，也可以配置从 CBs 所获取的秘钥数量阀值。最后，客户端钱包会发起一个 Fabcoin 交易请求，并且发送到其指定的一个 Peer.

Fabcoin 链代码。一个 peer 会模拟执行 Fabcoin 的 Chaincode 并且创建读写集。简而言之，对于一个消费交易，对于每一个币输入状态 in ∈ inputs chaincode 会首先执行 GetState(in)，也包括当前版本下的 readset 子集 in。当 chaincode 为每一个输入 in 执行 delstate(in)，则会将 in 添加到 writeset，并且将其 coin state 标识为”spent”。最后，对于 j=1,……,[outputs],chaincode 会执行 Putstate(txid.j,out)，并且将第 j 个输出用 out=(amount,owner,label)进行输出。另外，一个 peer 可以像 Fabcoin 中 VSCC 所描述的那样进行交易验证；但这不是必须的，因为 VSCC 实际上会验证交易，但是它会允许正确的 peer 节点过滤掉那些可能的异常交易。在我们的运行过程中，chaincode 运行 Fabcoin VSCC 而无需加密验证签名(仅在实际 VSCC 中验证)。

自定义 VSCC。最后，每一个 peer 通过自定义 VSCC 验证 Fabcoin 交易，会验证各个公钥中的 sigs 加密签名并执行如下验证过程：对于一个创建新币的交易，它会检查输出的状态是否和交易标识（txid)相匹配,并且所有的输出是否正确。对于消费币的交易，VSCC 会附加如下验证：(1)对于所有输入的币状态，在 readset 中已创建的内容是否同时也添加到了 writeset 中，并表示为 deleted.(2)所有输入的币状态总和是否等于输出的总

Work done at IBM.

和。（3）输入\出币标签是否匹配。VSCC 可以通过查询本地账本的值获得输入币的总数。

VSCC 不会检查重复支付，这是 Fabric 的标准验证过程，会在自定义 VSCC 运行之后开始。另外，如果两个交易都试图将相同的 unspent 币状态分配给新的所有者，则两者都会通过 VSCC 的逻辑，但随后会在 PTM 执行读写冲突检查时被捕获。像 3.4 和 4.4 节所描述的那样，PTM 会验证存储在本地账本的版本号是否与读写集中的一致，因此，在第一次交易改变状态版本之后，随后的第二次交易会被视为无效。

5.2 测试过程与结论

运行环境：除非有明确的说明，我们的测试环境是在如下情况下进行的：（1）所有节点运行在 Fabric V1.1.0 环境下通过本地日志进行性能测试。（2）单个 IBM softlayer 中专用的 VM，网络带宽 1Gbps.(3)所有节点配置 2.0 GHz 16-vCPU,8G 内存 Ubuntu 和本地 SSD。（4）单 channel 的排序服务运行在 3 个 ZK 的节点、4 个 KAFKA 实例，3 个 Fabric 排序节点，都运行在不同的 VM 中（5）5 个 peer 被分配给不同的组织，且都是背书节点（6）使用 256 位 ECDSA 进行数字签名.为了实现在跨多节点的交易流中测量延迟问题，整个实验中节点都使用 NTP 服务同步时钟。所有 Fabric 使用配置 TLS 进行通信。

测试方法：第一步我们会运行 MINT 操作产生新币，第二步我们会基于这些新币发起 SPEND 交易(单输出\入 SPEND 交易)，在测试吞吐量这一指标时，我们会在单个 VM 上通过增加 Fabric CLI 请求数，直到端到端吞吐量达到饱和。吞吐量指标在报告中是指稳定状态下的平均值。忽略掉一些已经停止提交共享交易的客户端。每个实验中，客户端线程会执行至少 50 万个 MINT 和 SPEND 交易。

实验 1：区块大小的选择

一个影响 Fabric 吞吐量和延迟的重要指标是区块大小的设置。在后续实验中，我们会修改区块大小值(0.5MB-4MB)，并评估其对性能的影响。实验结果如图 6 所示。显示了 peer 节点吞吐量峰值以及平均端到端的延迟情况。

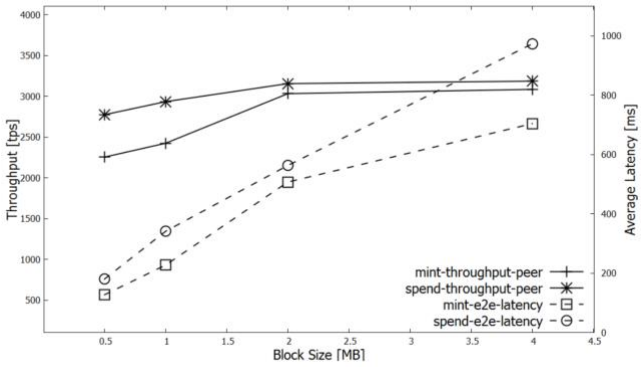


图 6 区块大小对吞吐量和延迟的影响

我们发现吞吐量在增加区块大小为 2M 后不再明显提升，但是延迟会越来越差。因此，我们会采用 2M 区块大小进行下面的测试，当然是在可以接受 500ms 延迟的前提下。

交易量。我们也会对 MINT 和 SPEND 的交易数进行观察。其中，2MB 的区块里可以包含 473 个 MINT 或 670 个 SPEND 交易。平均每个交易的容量为 3.06KB（SPEND）和 4.33KB（MINT）。总体上，Fabric 的交易容量会大一些，因为它携带有一些认证信息。另外，MINT 交易量会比 SPEND 要大，因为其会携带 CB 的验证信息。这一点也是 Fabric 和 Fabcoin 以后需要优化的地方。

实验 2：CPU 对性能的影响

Fabric 会使用 CPU 进行比较密集的加密操作，为了测试 CPU 对吞吐量的影响。我们将 4 个 peer 运行在 4,8,16 和 32 核的 VM 中，同时对区块进行了粗粒度的延迟，以识别其瓶颈。

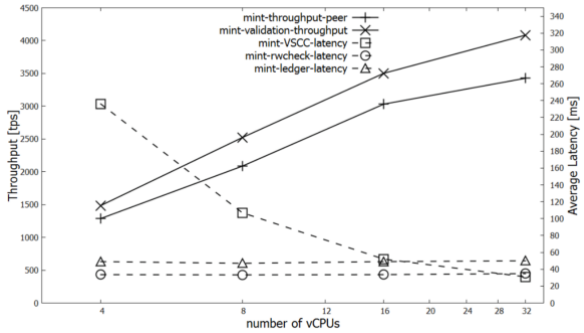
我们的实验侧重于验证阶段，因为基于 KAFKA 的排序服务在我们的集群中(同一个数据中心)不会是一个瓶颈。在验证阶段，特别是 Fabcoin 的 VSCC 验证，需要进行数字签名验证，这一过程是计算密集型的。我们会测量 peer 节点在验证阶段的吞吐量。

结论是：基于 2MB 的区块大小，如图 7 所示，图 7a 是 MINT 交易的结果，图 7b 是 SPEND 交易的结果。两种交易在 CPU 增幅范围内对吞吐量和延迟进行分析。我们发现验证阶段吞吐量受 CPU 影响非常明显。另外，由于背书策略验证在 Fabric VSCC 中进行并行执行，使得验证性能受 CPU 的影响是线性的。但是，对读写集检查和账本访问是串行的，因此 CPU 数量对其几乎没有影响。这对 SPEND 交易更加明显，在一个 2MB 的区块链中，相比 MINT 交易，包含 SPEND 交易数更多，

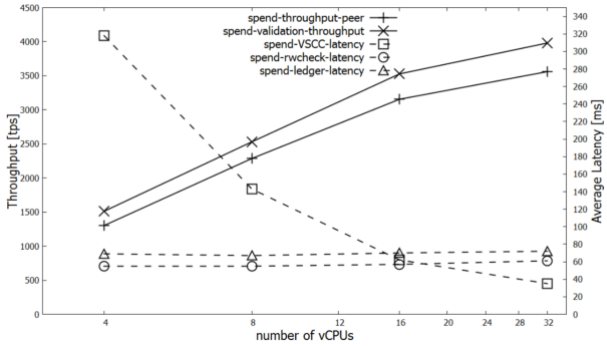
Work done at IBM.

这就延长了顺序验证阶段的时间(即读写检查和账本访问的时间)。

本实验也表明 Fabric 未来的版本在验证阶段可以进行并行(现在是串行)的方式，来消除 peer 中顺序执行所带来的开销，优化存储访问和并行化读写检查。



7a 仅 MINT 交易区块性能指标



7b 仅 SPEND 交易区块性能指标

图 7 Peer CPU 配置对端到端吞吐量、验证吞吐量和区块验证延迟的影响。

最后，在本实验中，我们测试了 32 核 vCPU 配置下 3560tps 的平均端到端的 SPEND 吞吐量，总体上，MINT 略低于 SPEND，但差异基本维持在 10%以内，32vCPU 情况下 MINT 吞吐量保持在 3420tps.

账本延迟分析。我们在上面实验中进一步对报告中峰值吞吐量进行了粗粒度的延迟测试，结果如表 1 所示：

排序阶段延迟由广播延迟和 peer 验证开始前的内部延迟。该表报告了 MINT 和 SPEND 交易的平均延迟、标准偏差和尾部延迟。

我们发现排序阶段占据了大部分延迟，平均延迟在 550 毫秒一内，另外，最高的延迟是发生在生成第一个区块并对其进行 build up 的时候。使用区块时间参数可以调节和降低低频交易下的延迟（参加 3.3 节），我们在实验中基本不使用该参数，会将其设置为一个比较大的值。

	avg	st.dev	99%	99.9%
(1) endorsement	5.6 / 7.5	2.4 / 4.2	15 / 21	19 / 26
(2) ordering	248 / 365	60.0 / 92.0	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 88.5	59.0 / 93.3
(5) ledger	50.6 / 72.2	6.2 / 8.8	70.1 / 97.5	72.5 / 105
(6) validation (3+4+5)	116 / 169	12.8 / 17.8	156 / 216	199 / 230
(7) end-to-end (1+2+6)	371 / 542	63 / 94	612 / 805	646 / 813

表 1：MINT 和 SPEND 交易在 5 个阶段的延迟分析，基于 32 核 vCPU 和 2MB 区块大小。验证阶段延迟包括：3，4 和 5.端到端的延迟包括 1-5.

实验 3：SSD vs RAM Disk.

为了评估稳定的存储给系统带来的好处，我们重复了之前的实验，将所有 peerVM 以 RAM DISK(tmpfs)的格式进行挂载。带来的好处非常有限。tmpfs 只是在 peer 进行验证的时候有一些帮助。我们测量了基于 32 核 CPU 配置，3870 个 tps 情况下，tmpfs 的吞吐量比 SSD 大概只有 9%的提升。

实验 4：LAN 环境的扩展性。

在本实验以及后续实验中，我们会增加 peer(配置 16 核 vCPU)的数量来评估 Fabric 的扩展性。

本实验我们还是将每个组织分配一个 peer，运行在香港的单个 IBM Cloud 数据中心环境。所有 peers 不经过 gossip 直接从排序服务接受区块，20 个 peer 开始（其中 10 个是 Fabcoin 背书节点）逐渐增加至 100 个，峰值吞吐量与 peers 数理的关系如图 8 所示。

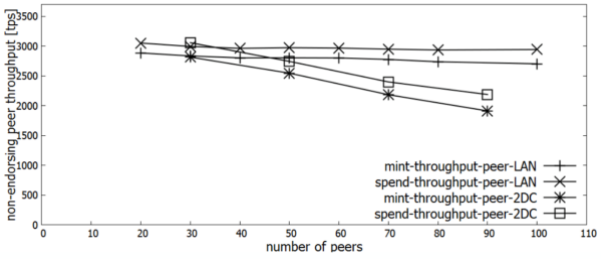


图 8 peer 数量对非背书节点吞吐量影响

我们发现，Kafka 在面对 peer 不断增长的同时，有很好的扩展能力。当所有 peer 随机链接到 OSNs 时，3 个 OSN 的带宽可能会成为 1Gbps 网络的吞吐量瓶颈，但结果却不是这样。我们也找出了具体原因：IBM Cloud 中分配的带宽要高于标准带宽。netperf 测试的结果始终在 5-6.5Gbps 之间。

实验 5：基于 2 个 DC 的扩展性和对 gossip 的影响。

在下列实验中，我们将排序服务、10 个背书节点和客户端迁移至东京 DC，只剩下非背书节点在香港 DC。这个(以及下一个)实验的目的是评估网络带宽成为瓶颈的场景。我们在香港 DC 将非背书节点从 20 增加到 80，除了连接东京的 10 个背书节点外，还直接与排序服务

Work done at IBM.

（每个组织一个 Peer）进行连接。东京和香港两地之间 VM 的 netperf 吞吐量为 240Mbps。

峰值吞吐量与 peer 数量的关系如图 8（2DC 曲线）。我们清楚地看到在 30 个 peer 以内，其影响和上次（同一 DC）基本一致。但之后会随着 peer 的增加，东京 3 个 OSNs 网络连接均已饱和，吞吐量有所下降，我们只测了香港 DC 中 90 个 peers 之上的 1910tps 的 MINT 交易和 2190tps 的 SPEND 交易。

为了提升在 WAN 环境的扩展性，Fabric 可以采用 gossip(4.3 节)。我们重复了最后一次实验，在香港 DC 中的 80 个 peer 分配给 8 个组织，平均每个组织 10 个 peer，在这个配置下，每个组织只有一个 leader peer 直联到排序服务，并且 gossips 区块到组织内的其他区块。本次实验中（gossip fanout 参数为 7），我们分析了 2544tps 的 MINT 和 2753tps 的 SPEND 交易峰值吞吐量，这意味着 gossip 的性能符合预期。其吞吐量略低于 LAN 环境，这是因为组织中的 leader peer（直接连接到 OSNs 的 peer）同时还要管理 gossip。

实验 6：多数据中心（WAN）下的性能。

最后，我们扩展到 5 个数据中心进行本次实验，分别是：东京、香港、墨尔本、悉尼和奥斯陆，每个 DC 配置 20 个 Peer，共计 100 个 peer。和前面实验一样，东京 DC 有排序服务、10 个背书节点和所有客户端。我们在没有 gossip(1 个组织 1 个 peer)和有 gossip（1 个组织 10 个 peer，每个 DC2 个组织，fanout 参数为 7）两种情况下进行了测试。结果如表 2 所示。为了更好地进行参照，第一行为该 DC 中 VM 与东京 DC 之间运行 netperf 的网络吞吐量（Mbps）。

	HK	ML	SD	OS
netperf to TK [Mbps]	240	98	108	54
peak MINT / SPEND throughput [tps] (without gossip)	1914 / 2048	1914 / 2048	1914 / 2048	1389 / 1838
peak MINT / SPEND throughput [tps] (with gossip)	2553 / 2762	2558 / 2763	2271 / 2409	1484 / 2013

表 2：5 个数据中心下 100 个 peer 的实验

这个结果再一次展现了在 WAN 环境下使用 gossip 所带来的优势。将 OS\SD 与 HK\ML 比较,我们发现一个有趣的结果：SD 中 gossip 的吞吐量低是因为受限于 VM 的 CPU 配置，同等条件下，其吞吐量要低于 HK 和 ML，而在 OS，总吞吐量则更低。然而瓶颈不是来自于排序服务的带宽，正如我们用 netperf 测试的结果一样，是源于 OS 和 TK 的单点 TCP 带宽。此时，gossip 的优势无法观察到。我们减少了从 OS 到 TK 的 gossip TCP 链接，从而略微改善了 OS 的系统吞吐量。

6 应用场景和用例

一些主要的云服务商已经基于 Fabric 提供了 BAAS 服务，包括 Oracle,IBM 和微软。另外，Fabric 目前有超过 400 个原型和基于分布式账本的 POC 在多个行业和案例落地，例如：食品安全追溯、基于云的银行业区块链平台、全球贸易等。本章节我们会介绍一些已经部署成功的 Fabric 案例。

外汇(FX)结算。在 Fabric 上运行外汇双边支付网络,使用 Fabric channel 为每个客户端提供隐私保障，具体负责净额结算的特殊机构是(“settler”)所有 channel 和排序服务。区块链有助于解决未结算的交易和将一些必要信息进行维护的账本。这些数据可以由客户实时访问，有助于解决纠纷，减少风险，并最大限度地降低信用风险[49]。

企业资产管理。该方案通过对资产在制造和部署环节的转移过程进行跟踪，同时捕获与硬件相关的软件许可情况。区块链记录了资产生命周期的信息和相关证据。账本作为资产所涉及的所有参与者之间透明的系统，可以解决传统方案难以解决的数据质量问题。多方参与共识的区块链在制造商、运输企业、仓储、客户及安装人员之间运行。它使用 3-tiered 架构，用户通过 Fabric 客户端连接到 peer.更多相关描述请参考[56]

全球跨境支付。通过与 Stellar.org 和 KlickEx Group 进行合作，IBM 自 2017 年 10 月开始运营多币支付方案，该方案用来处理太平洋地区 APFII 组织成员之间的交易。Fabric 区块以所有参与者认可的交易形式进行金融交易记录。所有相关方都可以访问并深入了解金融交易的清算和结算交易。

该方案适用于所有支付方式和内容，并且允许金融机构选择结算网络。特别是，结算可能会使用不同的方法，Fabric 会根据参与者的配置决定如何进行结算付款，一种可能的解决方案是通过名为“Stellar Lumens”的加密货币进行转换。其他方式是基于具体的贸易金融工具。

7 相关工作

Fabric 的架构类似于由 Kemme 和 Alonso [40]所开创的中间件数据库复制技术。然而，该架构仅解决了 CFT 的问题，对于 BFT 问题没有很好地解决。例如，依赖一个节点来执行每个交易，用非对称更新处理的方式复制数据库，在区块链中是不可行的。Fabric 的

Work done at IBM.

execute-order-validate 架构是对拜占庭模型的很好诠释，对分布式账本的最佳实践。

从 BFT 数据库复制角度看，Byzantium [32] and HRDB [55] 是 Fabric 的前身。Byzantium 运行事物并行并使用主动复制，但是在使用 BFT 中间件时需要完全遵循 BEGIN\COMMIT\ROLLBACK(译者注：这很类似于分布式事务:TCC)。在其优化模型中，每个操作都由一个单独的 master 来协调，如果怀疑 master 作假，则所有副本会执行 master 的事物操作，并触发昂贵的协议来改正 master.HRDB 则以更可靠的方式确保 master 的准确，但与 Fabric 相比，两者都使用主动复制，无法处理灵活的信任模型，无法依赖于确定性操作。

Eve[37]探索了 BFT SMR 的相关架构，peer 同时执行交易，然后使用共识协议验证它是否都达到相同的输出状态。如果状态不同，它们会回滚并按顺序执行操作，Eve 包含独立执行的元素，也存在于 Fabric 中，但不包含此外的其他功能。

最近，一些基于分布式账本技术的联盟链平台相继问世，也无法与所有平台进行对比，但是有一些比较突出的是 Tendermint [14]，Quorum [13]，Chain Core [4]，Multichain [12]， Hyperledger Saw-tooth [9]，the Volt proposal [50]等等，具体请参见最近[24,30]中的文献。

所有平台都遵循 order-execute 架构如第 2 节所述。以 Quorum 为例，这是一个以企业为中心的以太坊架构，共识是基于 raft，使用 gossip 对所有节点进行广播，而 raft leader（也叫 minter）将有效交易汇总到一个区块，使用 raft 进行分发。所有的 peer 会按照 lerader 所决定的顺序执行交易。这种方式所带来的限制，请参考 1-2 节。

8 结论

Fabric 是一种模块化、可扩展的分布式联盟链系统，它引入了一种创新的体系架构：将交易的执行与共识分开；基于背书策略的方式；基于中间件的数据库复制技术。

基于其模块化的架构，Fabric 非常适合对其进行进一步的改进和优化。下一步工作重点在：（1）通过探索更多参照和优化措施来提高性能;（2）在更大部署范围进行扩展;（3）一致性保证和更通用的数据模型;(4)通过不同的协议实现更多的伸缩性;(5)通过更多的技术探索，实现更好的加密技术，提升对交易和分布式账本数据的更好的隐私和机密保护。

致谢

我们感谢 Pramod Bhatotia 和一些匿名审稿人非常有见地和建设性的评论。

这项工作的一部分的到了欧盟 Hori-zon 2020 框架计划的支持，该计划的授权协议为 643964（SUPERCLOUD），部分由瑞士国家教育，研究和创新秘书处（SERI）根据合同号 15.0091 提供支持。

该文档由 IBM 资深区块链架构师 颜亮 (yanlfz@cn.ibm.com) 翻译,并提交到社区。

参考资料

[1] Apache CouchDB. http://couchdb.apache.org.
[2] Apache Kafka. http://kafka.apache.org.
[3] Bitcoin. http://bitcoin.org.
[4] Chain. http://chain.com.
[5] Ethereum. http://ethereum.org.
[6] gRPC. http://grpc.io.
[7] Hyperledger. http://www.hyperledger.org.
[8] Hyperledger Fabric. http://github.com/hyperledger/fabric.
[9] Hyperledger Sawtooth. http://sawtooth.hyperledger.org.
[10] LevelDB in Go. https://github.com/syndtr/goleveldb/.
[11] The Linux Foundation. http://www.linuxfoundation.org.
[12] MultiChain. http://www.multichain.com/.
[13] Quorum. http://www.jpmorgan.com/global/Quorum.
[14] Tendermint. http://tendermint.com.
[15] IBM announces major blockchain solution to speed global payments. https://www-03.ibm.com/press/us/en/pressrelease/53290.wss, 2017.
[16] R. Aitken. IBM & Walmart launching blockchain food safety alliance in China with Fortune 500’ s JD.com. https://www.forbes.com/sites/tomgroenfeldt/2017/03/05/ibm-and-maersk-apply-blockchain-to-container-shipping/, 2017.
[17] P.-L. Aublin, R. Guerraoui, N. Knez˘evic´, V. Que´ma, and M. Vukolic´. The next 700
BFT protocols. ACM Trans. Comput. Syst., 32(4):12:1–12:45, Jan. 2015.
[18] E.Ben-Sasson,A.Chiesa,C.Garman,M.Green,I.Miers,E.Tromer,andM.Virza. Zerocash: Decentralized anonymous payments from bitcoin. In IEEE Symposium on Security & Privacy, pages 459–474, 2014.
[19] A. N. Bessani, J. Sousa, and E. A. P. Alchieri. State machine replication for the masses with BFT-SMART. In International Conference on Dependable Systems and Networks (DSN), pages 355–362, 2014.
[20] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. J. ACM, 32(4):824–840, 1985.
[21] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, Distributed Systems (2nd Ed.), pages 199–216. ACM Press/Addison-Wesley, 1993.
[22] C.Cachin,R.Guerraoui,andL.E.T.Rodrigues.IntroductiontoReliableandSecure Distributed Programming (2. ed.). Springer, 2011.
[23] C. Cachin, S. Schubert, and M. Vukolic´. Non-determinism in byzantine fault-tolerant replication. In 20th International Conference on Principles of Distributed Systems (OPODIS), 2016.
[24] C. Cachin and M. Vukolic´. Blockchain consensus protocols in the wild. In A. W. Richa, editor, 31st Intl. Symposium on Distributed Computing (DISC 2017), pages 1:1–1:16, 2017.
[25] J. Camenisch and E. V. Herreweghen. Design and implementation of the idemix anonymous credential system. In ACM Conference on Computer and Communica-tions Security (CCS), pages 21–30, 2002.
[26] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4):398–461, Nov. 2002.
[27] B. Charron-Bost, F. Pedone, and A. Schiper, editors. Replication: Theory and Practice, volume 5959 of Lecture Notes in Computer Science. Springer, 2010.
[28] K.Croman,C.Decker,I.Eyal,A.E.Gencer,A.Juels,A.Kosba,A.Miller,P.Saxena, E. Shi, E. G. Sirer, et al. On scaling decentralized blockchains. In International Conference on Financial Cryptography and Data Security (FC), pages 106–125.

Work done at IBM.

Springer, 2016.
[29] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In ACM Symposium on Principles of Distributed Computing (PODC), pages 1–12. ACM, 1987.
[30] T.T.A.Dinh,R.Liu,M.Zhang,G.Chen,B.C.Ooi,andJ.Wang.Untanglingblock- chain: A data processing view of blockchain systems. e-print, arXiv:1708.05665 [cs.DB], 2017.
[31] Fujitsu Limited. Fujitsu cloud service adopted by Japanese Bankers Association for blockchain-based nancial service testbed. http://www.fujitsu.com/global/about/resources/news/press-releases/2017/0914-01.html, 2017.
[32] R. Garcia, R. Rodrigues, and N. M. Preguiça. E cient middleware for Byzantine fault tolerant database replication. In European Conference on Computer Systems (EuroSys), pages 107–122, 2011.
[33] T. Groenfeldt. IBM and Maersk apply blockchain to container shipping. https://www.forbes.com/sites/tomgroenfeldt/2017/03/05/ ibm-and-maersk-apply-blockchain-to-container-shipping/, 2018.
[34] R. Guerraoui, R. R. Levy, B. Pochon, and V. Que´ma. Throughput optimal total order broadcast for cluster environments. ACM Trans. Comput. Syst., 28(2):5:1– 5:32, 2010.
[35] J. P. Morgan. Quorum whitepaper. https://github.com/jpmorganchase/ quorum-docs, 2016.
[36] F. P. Junqueira, B. C. Reed, and M. Sera ni. Zab: High-performance broadcast for primary-backup systems. In International Conference on Dependable Systems & Networks (DSN), pages 245–256, 2011.
[37] M. Kapritsos, Y. Wang, V. Que´ma, A. Clement, L. Alvisi, and M. Dahlin. All about Eve: Execute-verify replication for multi-core servers. In Symposium on Operating Systems Design and Implementation (OSDI), pages 237–250, 2012.
[38] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In Symposium on Foundations of Computer Science (FOCS), pages 565–574. IEEE, 2000.
[39] B. Kemme. One-copy-serializability. In Encyclopedia of Database Systems, pages 1947–1948. Springer, 2009.
[40] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. ACM Transactions on Database Systems, 25(3):333–379, 2000.
[41] B. Kemme, R. Jime´nez-Peris, and M. Patin¯o-Marti´nez. Database Replication. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.
[42] A.E.Kosba,A.Miller,E.Shi,Z.Wen,andC.Papamanthou.Hawk:Theblockchain model of cryptography and privacy-preserving smart contracts. In 37th IEEE Symposium on Security & Privacy, 2016.
[43] S. Liu, P. Viotti, C. Cachin, V. Que´ma, and M. Vukolic´. XFT: practical fault tolerance beyond crashes. In Symposium on Operating Systems Design and Implementation (OSDI), pages 485–500, 2016.
[44] S.Nakamoto.Bitcoin:Apeer-to-peerelectroniccashsystem.http://www.bitcoin.org/bitcoin.pdf, 2009.
[45] D.OngaroandJ.Ousterhout.Insearchofanunderstandableconsensusalgorithm. In USENIX Annual Technical Conference (ATC), pages 305–320, 2014.
[46] F. Pedone and A. Schiper. Handling message semantics with Generic Broadcast protocols. Distributed Computing, 15(2):97–107, 2002.
[47] B.Peterson.IBMtoldinvestorsthatithasover400blockchainclients–including Walmart, Visa, and Nestle. Business Insider UK, http://uk.businessinsider.com/ibm-blockchain-enterprise-customers-walmart-visa-nestl-2018-3/, 2018.
[48] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comput. Surv., 22(4):299–319, 1990.
[49] J. Sengupta, R. Komarraju, and K. Bear. Bridging the divide: How CLS and IBM moved to blockchain. IBM Institute for Business Value, https://www-935.ibm.com/services/us/gbs/thoughtleadership/bridgingdivide/, 2017.
[50] S. Setty, S. Basu, L. Zhou, M. L. Roberts, and R. Venkatesan. Enabling secure and resource-e cient blockchain networks with VOLT. Technical Report MSR-TR-2017-38, Microsoft Research, 2017.
[51] M.Shapiro,N.Preguiça,C.Baquero,andM.Zawirski.Con ict-freereplicateddata types. In Proceedings of the 13th international conference on Stabilization, safety, and security of distributed systems, SSS' 11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.
[52] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. BFT protocols under re. In Symposium on Networked Systems Design & Implementation (NSDI), pages 189–204, 2008.
[53] J. Sousa, A. Bessani, and M. Vukolic´. A Byzantine fault-tolerant ordering service for the Hyperledger Fabric blockchain platform. In International Conference on Dependable Systems and Networks (DSN), 2018.
[54] A.S.Tanenbaum.Distributedoperatingsystemsanno1992.whathavewelearned so far? Distributed Systems Engineering, 1(1):3–10, 1993.
[55] B. Vandiver, H. Balakrishnan, B. Liskov, and S. Madden. Tolerating Byzantine faults in transaction processing systems using commit barrier scheduling. In ACM Symposium on Operating Systems Principles (SOSP), pages 59–72, 2007.
[56] M. Venkataraman, M. Vridhachalam, A. Rosen, and B. Arthur. Adopt- ing blockchain for enterprise asset management (EAM). IBM de- veloperWorks, https://www.ibm.com/developerworks/cloud/library/ cl- adopting- blockchain- for-enterprise- asset- management- eam/index.html, 2017.
[57] M. Vukolic´. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In International Workshop on Open Problems in Network Security (iNetSec), pages 112–125, 2015.
[58] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Byzantium Version, https://ethereum.github.io/yellowpaper/paper.pdf.
[59] J.Yin,J.Martin,A.Venkataramani,L.Alvisi,andM.Dahlin.Separatingagreement from execution for Byzantine fault tolerant services. In ACM Symposium on Operating Systems Principles (SOSP), pages 253–267, 2003.

Work done at IBM.