解密区块链：从数据处理的角度看区块链系统

Tien Tuan Anh Dinh，Rui Liu，Meihui Zhang,Member，IEEE,Gang Chen,
Member,IEEE，Beng Chin Ooi，Fellow，IEEE,and Ji Wang

摘要：区块链技术在最近几年中获得了巨大的发展动力，区块链是一个分布式的账本，这个账本能够保证互不信任的各方维护一组全局状态，各方在这个全局状态的存在，值，历史纪录方面达成一致意见。随着科技的快速发展，掌握区块链的核心技术，尤其是它在数据处理方面的能力是很重要也很有挑战性的。在本文中，我们首先调查了区块链的设计，重点关注私有区块链（在这些区块链中，各方都经过了认证）。我们从四个方面分析了生产和研究系统:分布式账本，加密，共识协议和智能合约。接着我们介绍了BLOCKBENCH，这是一个用于理解私有区块链在数据处理工作负载方面性能的基准框架。基于BLOCKBENCH，我们对以太坊，以太坊钱包和超级帐本这三个主要的区块链系统进行了综合评估。结果表明了在设计上的这种，以及区块链和数据库系统之间巨大的性能差异。从数据库系统的设计原则出发，我们讨论了一些使区块链性能更接近数据库领域的几个研究方向。

关键词_———区块链，分布式数据库，调查

1 介绍

区块链技术正在风靡世界,这很大程度上是因为比特币[1]的成功。区块链，也称分布式账本，本质上一个仅追加的数据结构，这个数据结构由一组互不信任的结点来维护。区块链系统中的结点对于一组区块的顺序达成一致意见，每一个区块都包含有大量的交易，因此区块链能够被看作是一系列有序交易的日志。从数据库的角度，区块链也能被看作是分布式事务管理的解决方案，结点保留数据的备份,并且对事务执行的顺序达成一致意见。然而传统数据库假设可信的环境，并且采用了著名的并发控制技术[2],[3],[4]来保证事务有序执行。区块链的关键属性是它假设结点以任意（或者拜占庭）方式运行。通过设计来容忍拜占庭错误，相比于现有的数据库系统，区块链提供了更强的安全性。

在最初的设计中,比特币的区块链存储货币作为系统的状态,在这种应用中，比特币结点实现了一个简单的复制状态机模型，在这种模型中，比特币从一个地址移动到另一个地址。从那时起，区块链的发展已经超越加密货币，以支持用户自定义的状态和图灵完备状态机模型。例如,以太坊[5]能够支持任何分布式的，重复的应用程序，这些程序称为智能合约。更重要的是，工业界对区块链的兴趣开始驱动一种新型的参与者均经过认证的区块链平台的发展。这种环境下的区块链系统称为私有的（或者许可的），这与早期在公有环境中任何人都能够参与和离开的系统（或者无须认证的）相对。诸如证券交易与结算[6]，资产和财政管理[7],[8],银行和保险[9]等应用正在建立和评估之中。这些应用目前由Oracle和MySQL等企业级的数据库系统支持，但是区块链很有可能会改变这种现状，因为它降低了系统架构和人力成本[9]。特别是，区块链的不可篡改和透明的特性帮助降低了人为错误和由于数据冲突而需要的手动干预。区块链能够通过在数据管理上消除重复的工作来简化业务流程。高盛估计在当前资本市场中将会节省

60 亿美金[9]，J.P.摩根预测在 2020 年区块链将开始区块当前多余的基础设施[8]。

在日益增长的商业和学术兴趣中，大量的区块链系统正在迅速涌现，每一个系统都拥有一些独特的功能。私有和公共领域都在大声呼吁采用区块链，但是他们都面临着巨大的选择。尽管具有挑战性，理解区块链技术能和不能做什么是非常重要的。要理解区块链，根本上必须回答以下问题：

1）区块链是什么？具体来说它的哪些特点能够对当前和未来的应用有益?

2）当前的区块链系统在设计和性能方面相互都有哪些不同？

3）当前的挑战是什么？未来区块链会是怎么样的？

为了回答这些问题，在本文中，我们首先区分两大区块链系统，即公有和私有区块链。然后我们解释了当前系统所涉及的四个关键技术概念：分布式账本，加密，共识和智能合约。接着我们描述了 BLOCKBENCH[10]，这是我们用于定量评估和比较私有区块链的基准框架。通过 BLOCKBENCH，我们对以太坊[5]，以太坊钱包[11]和超级帐本[12]这三个主要的区块链进行了综合评估。结果显示，当前区块链的性能有限，远低于最先进的数据库系统所能提供的性能。最后，我们从我们搭建大规模数据库系统的经验中提取了一些能够提高未来区块链性能的设计理念。

其他一些对区块链系统深入的调查[13],[14]主要关注加密货币，这是一个重要的区块链应用但它不能充分体现区块链作为数据处理平台的潜力。我们对这项技术尤其是它的性能提出了更深入、更广泛的观点。总的来说，我们的贡献是：

1）我们提供了对区块链系统的深入调查。我们讨论了最新的技术，并且从四个维度对当前系统进行分类：分布式账本，密码学，共识协议和智能合约。

2）我们讨论了我们的基准框架：BLOCKBENCH，它用于理解私有区块链在数据处理工作负载方面的性能。

3）我们对以太坊、以太坊钱包和超级帐本进行了广泛评估。结果显示了区块链作为数据处理平台的限制。这些限制表明了一些系统瓶颈，因此能够作为未来区块链研究和开发的基准。

在下一部分，，我们将概述区块链系统，将其分为公有和私有链。第 3 部分介绍了构建区块链的四个模块，这些模块将被用于第 4 部分来对现有区块链进行划分。第 5 部分描述了 BLOCKBENCH，接着是第 6 部分对三个区块链系统的评估。第 7 部分讨论了从性能研究中学到的一些经验教训，以及如何将数据库的设计理念用于改进区块链。第 8 部分是结论。

2 区块链：私有 VS 公有

典型的区块链系统由多个不完全互相信任的结点组成。一些节点会表现出拜占庭式的行为，但是大多数是诚实的。同时，这些节点维护了一组共享的，全局状态并且执行交易来修改这些状态。区块链是一个特殊的数据结构，这种数据结构保存了历史状态和交易。所有系统中的结点对交易和他们的顺序达成一致意见。图 1 展示了区块链的数据结构:每一个区块通过一个加密的指针指向它的前继，依次可以回溯到第一个（创世）区块。由于此，区块链通常被称为分布式账本。

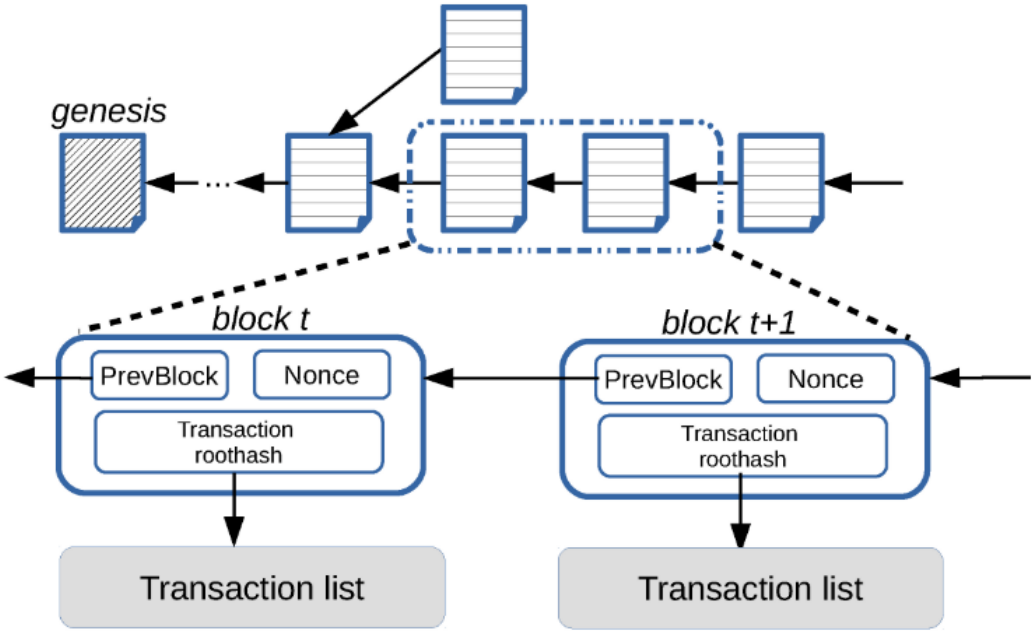


图 1：区块链数据结构。交易打包成区块并且链向前一个区块。

区块链中的交易和传统数据库中的事务相同：都是一系列应用于一些状态的操作。因此，区块链要求相同的 ACID 语义。两者主要的的区别是失效模型。当前事务型的分布式数据库[15],[16]采用了传统诸如两阶段提交的并发控制技术来确保 ACID。由于简单的碰撞故障模型，这些并发技术能够实现很高的性能。相反，最初区块链的设计考虑到了不利的环境，即一些结点可能表现出拜占庭行为，在这种模型下，并发控制的开销要高得多[17]。

在较高层面，区块链系统可以分为共有或私有。前者中，任何节点能够加入和离开这个系统，因此区块链是完全分布式的，这与 P2P 系统[18]类似。后者中，区块链实施了严格的身份认证。更具体来说，在系统中有一个接入控制机制来决定哪个结点可以加入这个系统。因此，每一个节点都是认证过的，并且其他结点知道它的身份。

2.1 公有区块链

比特币[1]是最为著名的公有链的例子。在比特币中，状态是数字硬币（加密货币），一笔交易将一些比特币从一组地址移动到另一组。每一个节点广播它想要执行的交易，称为矿工的特殊的结点收集交易并打包成区块，检查它们的合法性，接着通过共识协议来将区块追加到区块链上。比特币使用工作量（PoW）证明来达到共识：只有成功解决了计算难题(为区块头找到正确的随机数)的矿工才能将区块追加到区块链上。工作量证明能够容忍拜占庭错误，但是它的本质是概率的，也即可能会有两个区块同时追加到区块链上，这个时候区块链就产生了分叉。比特币通过一个区块后面追加了一定数量(通常是 6 个区块)的区块以后才确认的方法来解决上述问题。这种基于概率的方法带来了安全和性能问题：已经有仅拥有 25%算力的攻击者攻击的[19]例子，并且比特币的吞吐量很低(每秒 7 笔交易[20])。

大多数公有链系统使用 PoW 的变体来达成共识。PoW 在公有环境下运行得很好因为它能够防范女巫攻击[18]。然而，因为 PoW 是不确定的且计算开销很大，对于一些必须以确定性方式处理大量交易的诸如银行和金融等应用，它是不适用的。

2.2 私有区块链

Hyperledger[12]是最著名的私有链。由于在私有环境中结点身份是已知的，大多数区块链采用来自分布式共识的大量文献中的协议之一。Zab[21],Raft[22],Paxos[23],PBFT[17]是今天仍然在使用的著名的协议。Hyperledger 直接使用 PBFT1，其他的区块链像 Parity[11]，Ripple[6],和 ErisDB[24]开发了他们自己的变体。PBFT 是一个三阶段协议，在预备阶段，一个领导者广播一个将由其他结点提交的值，接着，在准备阶段，各个节点广播他们将要提交的值，最后，在提交阶段，如果在上一阶段，超过三分之二的结点对值达成一致意见，该值将被提交。PBFT 是基于通信的，但是它在部分同步网络中保证了安全性和活跃性。除了确定性共识，私有链还支持智能合约，通过合约可以表述高度复杂的交易逻辑。这些特点在商业和财政系统中是很令人满意的。事实上，私有链激起了银行和财政机构的兴趣，一些机构甚至声称私有链能够改变当前数据管理实践[8],[9]。

3 核心概念

将区块链分为公有和私有对于识别一些区块链的主要特征是有用的。然而，理解它们细微的不同需要更好的分类。这部分介绍了四个基础的概念，基于此可以获得更加系统更加详细的分类。

3.1 分布式账本

账本是一个包含有一系列有序交易的数据结构。例如，账本可能记录了多家银行之间金融交易，或者在已知方之间的货物交换。在区块链中，账本在所有节点上复制。此外，交易被打包成区块，接着被链接到一起。因此，分布式的账本实际上是一个重复的仅追加的数据结构。区块链从一些初始状态开始，账本记录了对状态进行更新操作的全部历史。

支持分布式账本的系统能够从三个维度来描述，如表 1 所示。首先账本之上的应用决定了存储在账本中的数据模型。数据模型提取出了关键数据抽象，使应用程序能够简单的表达它的逻辑。例如，加密货币应用可能采用用户账户模型，这与传统银行系统相似。另一方面，通用区块链可能采用低层次的模型，例如表或者键值对。第二，系统可能有一个或者多个互相相连的账本。例如一个大型企业可能拥有多个账本，每个账本对应一个部分：工程部，客户服务部，供应链，工资单等。第三，账本的拥有者也可能不同，从完全公开到完全被一方控制。例如，比特币是完全公开的，因此它需要代价昂贵的共识协议来确定谁能更新账本。另一方面，Parity 预先确定了[11]一组能够记账的结点，这些结点仅需要广播区块即可。

3.2 共识

账本的内容反映了区块链维护的历史的和当前的状态。为了使账本在节点间复制，对账本的更新必须在各方间达成一致意见。换句话说，多方必须达成共识。注意，这与许多现实世界的应用程序不同，例如法定货币，其中一个实体（银行或者政府）决定更新。

区块链系统的一个关键特征是结点之间互不信任，这意味着一些结点可能表现出拜占庭行为。因此共识协议必须能够容忍拜占庭错误。关于分布式共识的研究文献很多，而且有很多基于以前提出协议的变种被开发运用到区块链中[25]。这些协议能够从一个大范围上进行分类。一个极端是存粹基于计算的协议，它通过计算能力证明来随机选择一个结点单独决定下一步操作。比特币的 PoW 就是一个例子。另一个极端是存粹基于通信的协议，这种协议中结点拥有同等的投票权，通过多轮通信来达到共识。这些协议中，PBFT[17]是最好的例子，它被用于私有环境中，因为所有节点都已经过认证。

在这两个极端之间的是为了提高 PoW 和 PBFT 性能的混合协议。一些被用到公有链中来解决 PoW 的低效率问题。一个例子是消逝时间证明（PoET），它和另一种基于可信硬件的协议例如 SGX 代替了 PoW。另一个例子是 Elastico[26]和 Algorand[27]，这两个协议通过每一轮随机采样一组结点来提高 PoW 性能。其他的混合协议，例如权威证明(PoA)[28]，Stellar[29]和 Ripple[6]，被用于私有链中，它们通过在称为联邦的较小的网络中执行共识算法来提高 PBFT 性能。

Data Model	Number of ledgers	Owner	Example
Accounts	One	Administrator	Traditional ledgers used in financial institutions.
Assets	Many	Group of users	Private ledger used within a financial institution, or between small groups of financial organizations, e.g. global financial services.
Coins or accounts	One	Any user	Crypto-currencies like Bitcoin or Ethereum.

表 1 ： 分布式账本例子

3.3 加密

区块链大量使用加密技术来保证账本的完整性。这里的完整性指能检测到区块链数据篡改的能力。这种特性在公有环境中是至关重要的，因为公有环境中没有预先建立的信任。例如，公众对比特币等加密货币的信心决定了加密货币的价值，这种信任取决于账本的完整性，也即账本必须能够检测出双花问题。即使在私有链中，完整性也是同等重要的，因为认证的结点仍然能够进行恶意的行为。

系统中必须有至少两个级别的完整性保护。第一，全局状态通过一个哈希（默克尔）树来保护，它的树根哈希值存储在区块里。任何状态的改变将会导致新的树根。树的叶子结点包含状态，内部的结点包含它们孩子的哈希值。例如，Hyperledger v0.6 使用一个桶哈希树，在这棵树里，状态被集中（通过哈希映射）到预先定义数目的桶中。另一方面，以太坊使用了帕特夏默克尔树，这种树和 trie 树类似，其叶子结点是键值状态。第二，区块的历史是受保护的，也即区块一旦被附加到区块链上就不能被改变。主要的技术是通过一系列的加密哈希指针来链接区块：第 n+1 个区块里的内容包含第 n 个区块的哈希。通过这种方式，区块 n 的任何改变会立即使其后的区块失效。通过将默克尔树和哈希指针结合起来，区块链提供了一个安全有效的模型，该模型能够追溯历史上所有对全局

状态更改的信息。

区块链的安全模型假设了公钥密码学的可用性，身份，包括用户和交易身份从公钥证书中派生出来。因此，安全密钥管理对于任何区块链都是至关重要的。与其他安全系统一样，丢失私钥意味着丢失访问权限。但是在诸如加密货币的区块链应用中，丢失密钥有直接并且不可撤销的财务影响。我么将在 4.2 节讨论不同密钥和身份管理机制。

有许多研究系统使用新颖和复杂的密码协议来扩展原来的区块链设计。这些系统旨在通过零知识证明，组签名和可信硬件等深奥技术来提高区块链的安全性和性能。我们将在 4.2 节详细讨论这些技术。

3.4 智能合约

智能合约指交易执行时执行的运算。它可以被看成是交易发起时调用的存储过程。每一个节点对智能合约执行时的输入、输出和影响的状态达成一致意见。

所有的区块链都有它们内置的智能合约来实现交易的逻辑。例如，在加密货币中，内置的智能合约首先通过检查签名来验证交易输入。接着，它验证输出地址的金额和输入地址的金额是否匹配。最后，它将改变应用到状态上。本文的后面章节中，我们没有将这种内置的逻辑当成只能合约，相反，我们仅仅考虑能够被用户定义的智能合约。

一种描述智能合约的方法是通过它的语言。一种极端是比特币提供了少于 200 个操作码，用户可以使用这些操作码编写基于栈的脚本。例如，下列的脚本验证是否三个中有两个签名是可用的。

```
OP_2 <Pub1> <Pub2> <Pub3> OP_3 OP_CHECKMULTSIG
```

另一个极端是以太坊的智能合约能够指定任意的计算，即它是图灵完备的代码。图 2 展示了以太坊上运行的一段真实的智能合约。它实现了一个金字塔骗局：用户转账给该合约，合约接着将利息付给早期的参与者。这个合约有它自己的状态即一个参与者列表，并且有一个称为 enter 的接口函数。一个用户通过交易进行转账来激活这个合约。合约执行时，通过 msg.sender 来获取输入地址（用户账号），通过 msg.amount 来获取交易金额。接着更新合约积累的余额，计算每一个参与者的利息。最后通过调用 etherAddress.send 来进行支付。

在这两个极端中间的是一些能够提供比比特币具有更强表达能力，但是却非图灵完备的智能合约系统。Kadena[30]和 BigchainDB[31]支持复杂逻辑，但是却对语义进行限制的合约，以便能够对其进行形式化的安全性检查。

另一种分类智能合约系统的方法是通过运行时环境。大多数系统执行智能合约的环境与区块链的栈相同。我们称这些智能合约使用本地的运行环境。例如，Kadena 解析以类似于 Haskell 的语言编写的智能合约并且直接像 Haskell 程序一样执行他们。另一方面，以太坊自带虚拟机来执行以太坊字节码。Hyperledger 为了可移植性，使用 Docker 容器来执行智能合约。

```
contract Doubler {
  struct Partitipant {
    address etherAddress;
    uint amount;
  }
  Partitipant[] public participants;
  uint public balance = 0;
  ...
  function enter() {
    ...
    balance+= msg.value;
    ...
    if (balance >
              2*participants[payoutIdx].amount) {
      transactionAmount = ...
      participants[payoutIdx].
              etherAddress.send(transactionAmount);
      ...
    }
  }
  ...
}
```

图 2 一个以太坊智能合约的例子，用 solidity 编写，实现了金字塔骗局

4 技术现状

在本节中，我们使用第三节讨论的四个概念来比较当前的区块链。我们详细地解释了它们的设计并且突出了它们之间细微的差别。我们也讨论了正在解决的研究问题。

表 2 展示了一些区块链和它们的属性。这张表里包含了一些主要的区块链，但是我们强调这个列表并非是详尽的，尤其是鉴于商业和学术领域对区块链日益增长的兴趣。以斜体显示的系统或者是不再被维护，或者是仍处于开发的初始阶段。例如，Hydrachain[38]代码库最新的更新是写这篇文章前的 8 个月，而且 IOTA 的代码库只包含了一些参考实现。这张表没有显示加密这一栏，因为所有的系统（除了 ZCach）都使用了第 3 节描述的标准的技术。4.2 节讨论了一些新的尚未被集成的加密协议。

TABLE 2: Comparison of blockchain systems. Ones in italics are deemed inactive or at early phases of development.

	Application	Smart contract execution	Smart contract language	Data model	Consensus
Hyperledger v0.6.0 [32]	General applications	Dockers	Golang, Java	Key-value	PBFT
Hyperledger v1.0.0 [33]	General applications	Dockers	Golang, Java	Key-value	Ordering service (Kafka)
Bitcoin	Crypto-currency	Native	Golang, C++	Transaction-based	PoW
Litecoin [34]	Crypto-currency	Native	Golang, C++	Transaction-based	PoW (memory)
ZCash [35]	Crypto-currency	Native	C++	Transaction-based	PoW (memory)
Ethereum [5]	General applications	EVM	Solidity, Serpent, LLL	Account-based	PoW
Multichain [36]	Digital assets	Native	C++	Transaction-based	Trusted validators (round robin)
Quorum [37]	General applications	EVM	Golang	Account-based	Raft
HydraChain [38]	*General applications*	*Python, EVM*	*Solidity, Serpent, LLL*	*Account-based*	*Trusted validators (majority)*
OpenChain [39]	*Digital assets*	-	-	*Transaction-based*	*Single validator*
IOTA [40]	*Digital assets*	-	-	*Account-based*	*IOTA's Tangle Consensus*
BigchainDB [31]	Digital assets	Native	Python, crypto-conditions	Transaction based	Trusted validators (majority)
Monax [24]	General applications	EVM	Solidity	Account-based	Tendermint [41]
Ripple [6]	Digital assets	-	-	Account-based	Ripple consensus
Kadena [30]	*Pact applications*	*Native*	*Pact*	*Table*	*ScalableBFT [42]*
Stellar [29]	Digital assets	-	-	Account-based	Stellar consensus
Dfinity [43]	*General applications*	*EVM*	*Solidity, Serpent, LLL*	*Account-based*	*Threshold relay*
Parity [11]	General applications	EVM	Solidity, Serpent, LLL	Account-based	Trusted validators (round robin)
Tezos [44]	Michaleson applications	Native	Michaleson	Account-based	Proof of Stake
Corda [45]	Digital assets	JVM	Kotlin, Java	Transaction-based	Raft
Sawtooth Lake [46]	General applications	Native	Python	Key-value	Proof of elapsed time

表 2：区块链系统的比较，斜体的系统或者不再被维护或者仍处于开发初期。

4.1 分布式账本

回忆支持分布式账本的系统可以按照它的目标应用、账本的数目或者账本的拥有者来分类。接下来我们按照目标应用来将表 2 中的系统进行分类。

加密货币

区块链最成功的应用是加密货币。由于比特币的成功，大量其他货币竞相出现。大部分的这些货币例如 dodgecoin 都采用了和比特币相似的数据模型。相反，以太坊舍弃了比特币的基于交易的模型，而实现了基于账户的模型。货币应用自身的特点要求账本必须是公开的，并且系统只能维护一个账本。

数字资产

数字资产是一系列与现实世界价值想联系的数据，加密货币是其中的一个实例。不像加密货币是在区块链上创造出来并且从区块链里获取它的价值。数字资产通常从现实实体中流出，区块链仅仅只是一个媒介来纪录它的存在性和交换。Multichain[36],BigchainDB 和 Corda 通常提供账本用来存储和追溯资产历史。类似于比特币，它们的数据模型是基于以资产为核心的交易的。这些系统是面向私有环境的，在私有环境下，大量的组织能够在全网各个节点之间进行资产交易。这些组织就是账本的拥有者，而且系统中拥有多于一个账本也是很常见的。Stellar，Ripple 和 IOTA 发行它们自己的资产（代币）作为交换的媒介或者小

额交易的平台。特别是 IOTA 允许通过它的代币进行零费用的小额交易，这使账本对 LoT 设备之间的交换式有用的。上述系统采用基于用户的数据模型。每一个系统有一个账本，且账本公开；因此任何人都能够购买代币并且加入到交易中

通用应用

除了加密货币和数字资产，一些账本支持运行通用的，用户定义的运算（或者称为智能合约）。以太坊和它的派生物，即 Hydrachain，Quorum，Monax，Parity 和 Dfinity 允许用户编写任意在账本上执行的商业逻辑。例如，以太坊合约范围从简单的众筹活动到复杂的如 DAO[47]一样的基金投资。Difinity 有一个特殊类型的合约——政府合约——这增强了构建在类似于以太坊区块链之上的现实世界的规章制度。Hyperledger 和它的近邻 Sawtooth Lake 同样支持运行图灵完备的代码。他们提供了键值对数据模型，基于此，应用程序能够创建和更新区块链之上的键值对元组。Kadena 和 Tezos 通过设计它们自己非图灵完备却能够形式化验证的语言来限制应用程序的功能。Tezos 的数据模型是基于账户的，然而 Kadena 是基于表的。特别的，Kadena 的应用程序能对具有模式、版本和列历史的行键结构进行操作。

4.2 加密

身份管理

区块链中的用户通过公钥证书唯一标识。在公有环境下，用户首先生成密钥对（默认选项是基于 Secp256k 椭圆曲线的 ECDSA），接着从公钥的哈希值中提取出身份证明。这个哈希值在加密货币系统中主要是充当交易地址和账户数字，为了证实某一用户发出交易的所有者，该用户用和该公钥相对应的私钥为交易签名。私有环境中有附加的接入控制层。Hyperledger 通过身份提供服务和认证授权服务将该层从区块链中分离出去。管理员能够通过这些服务实现任意的策略来控制能够进入到区块链的用户。签名的请求在被下一个(共识)组件处理前首先需要经过这些服务检查。。Multichain 提供了一个有着固定数目全局许可的简单模型，然而其余的系统几乎没有提供它们协议的细节。

在私有链中管理用户密钥的问题和在典型企业系统中相同，因此现存的解决方案能够被轻易地集成。然而在公有链中，丢失私钥所带来的巨大规模和财政影响要求更加安全和更有适用的协议。特别是在比特币系统中，因为新的交易会导致密钥的更新，用户需要管理大量的密钥，所以这种问题对该系统也是一个很大的挑战。Eskandari 等人.[48]评估了比特币密钥(或钱包)管理的六种方法:本地存储，密码保护存储，线下存储，空隙存储，关键字生成密码，以及托管存储。作者发现由于在传统货币中这些隐喻的滥用和令人混淆的抽象概念，没有一种方法是有用的。

可信硬件

最近的分布式系统通过使用诸如 Intel SGX 和 ARM TrustZone 这样的可信硬件来降低安全开销以提高性能[49],[50]。Sawtooth Lake 提出将消逝时间证明作为工作量证明的有效替代。TownCrier[51]使用 SGX 来实现了一个可信方用来审查外部内容并且将它们导入区块链。这些系统基于可信模型比纯基于密码的系统要弱。特别是，它们的安全性依赖于可信的运行于可信硬件内部的计算基础

（TCB）。更小的 TCB 意味着更好的安全性。

所有基于可信硬件的系统依赖于远程认证协议。一个称为认可密钥(EK)的密钥对在每一个设备制造出来时被设置到该设备中。这样的密钥对是信任的基础，其他一些短期的密钥从这个密钥对中可以派生出来。在一块代码加载进设备之前，硬件通过对代码进行哈希并且用其中一个密钥来签名以验证该代码。签名方法和密钥证书向远程的第三方证明了在本地设备中运行的内容。这个协议要求一个认证机构来维护和签署一系列已知的证书和撤销的证书。例如直接匿名证明[52]在没有认证机构的情况下使用高度复杂的证明机制提供了硬件匿名性。

交易隐私

大部分区块链被设计用来保护交易完整性，但是他们没有考虑到交易隐私性。区块链在具有以下条件时被认为具有交易隐私性（1）交易不能被链接到其他交易上，（2）交易内容只能被参与者知道。在私有环境下，交易历史的完全透明可能不是问题。或者交易的透明性对于像财政和审计一类的应用是必要的，或者添加一个访问控制曾来保护区块链数据是直接的。另一方面，在公有环境下，对于交易隐私的需要受两方面因素驱动。首先，公开的攻击成功的覆盖了比特币网络的底层结构[53]，甚至直接将比特币地址和真实世界身份联系到一起[54]。第二，交易的可连接性会降低货币的价值，由于历史原因，一些代币比其他的更有价值。

Zerocoin[55]是第一个提供交易不可链接的区块链。它对比特币进行了扩展，允许比特币和特殊的称为零币的代币之间交易。Zerocoin 本质上实现了一种加密方法隐藏了零币和对应比特币之间的联系。每一个零币是两个随机值（s,r）的加密承诺。当时用零币时，零币的拥有者展示 s 以证明该代币未被使用，以及对 r 的零知识证明。交易的不可连接性通过使用的代币可能是许多未花费代币中的一个来保证。

Zerocash[35]通过提高 Zerocoin 的加密操作对 Zerocoin 进行了扩展。与 Zerocoin 是比特币的一个扩展不同，它是一个独立的区块链。Zerocah 中的交易包括分发和整合交易都是完全秘密的。复杂的零知识证明只能揭示存在一定数目的尚未使用的代币。Zerocoin 和 Zerocash 都使用 zkSNARK[56]来实现，底层的零知识证明带来了大量的开销。例如，Zerocash 要求可信方来安全的创建和分发一些公共的参数，这些参数的大小有成百上千兆字节。

高级签名

比特币支持多签名，即只有至少 n 中的 t 个人的签名合法时，一笔交易才能被引用。多签名通过将解密和广播能力赋予一组用户来抵御恶意节点。闪电网络[57]是比特币的扩展，支持近乎于即时的交易确认，它依赖多签名来首先在区块链上存储一些共同的基金。一旦确认，从上述资金中发起的付款能够在区块链之外发生并且立即确认。最后，这些基金由对应的具有所有要求签名的交易关闭。比特币多签名机制的扩展能够直接建立在 ECDSA[58]之上。更高级的机制，例如[56],[60]，可以被使用（尽管在当前的设计中没有重大改变）

Byzcoin[61]使用称为 Cosi[62]的组签名的机制来降低 PBFT 中的通信开销。Cosi 包括了四轮通信，最终产生了各个节点的签名并且该签名被该组所有成员验证。签名的被组织为 Schnorr 签名树。它显著的降低了 PBFT 协议准备和提交阶段网络中广播的消息的大小，因为每一个节点不再需要所有其他节点的签名进行验证。

4.3 共识机制

回忆区块链系统中存在很大范围的共识协议，从像 PoW 一样存粹基于计算的协议到像 PBFT 一样存粹基于通信的协议。表 3 总结了主要协议的关键属性，我们接下来将详细的介绍它们。

Consensus Protocol	Network Settings	Description
PBFT-based	Private	Hyperledger uses the original PBFT [17]. Tendermint [41] enhances it by assigning unequal weights to votes. Other variants include Scalable BFT [63], Parallel BFT [64], Optimistic BFT [65], etc.
Stellar	Federated	Stellar network [66] proposes its own consensus protocol where the nodes form intersecting groups (federates). Consensus is agreed in each group, then propagated to the rest of the network.
Ripple	Federated	Ripple payment system [6] proposes a variant of PBFT where the nodes belong to intersecting groups, and in each group there is a large majority of non-Byzantine nodes.
Proof-of-Work (PoW)	Public	Bitcoin uses pure proof-of-work, which leads to scalability issues. Bitcoin-NG [67], Byzcoin [61] separate leader election from transaction validation in PoW, thus increase the overall performance.
Proof-of-Stake (PoS)	Public	Tendermint [41] uses PoS, in which a node's ability to create new block is determined by its stake in the blockchain, e.g. the amount of currencies it owns [68]. A set of high-stake owners uses another consensus mechanism, which is usually faster than PoW, to reach agreement on a new block.
Threshold Relay	Public	Dfinity [43] proposes *threshold relay* in which nodes form random group based on a public verifiable random function (Byzcoin [61] and Elastico [26] adopt similar approaches). The nodes in the group create a new block by signing it using threshold signature.
Proof-of-Authority (PoA)	Private	Parity [11] uses PoA, in which some pre-defined nodes are considered trusted authorities and they can propose the next blocks. It then uses round-robin scheduling to assign every authority node a time window during which it can propose blocks.
Proof-of-Burn (PoB)	Public	Slimcoin [69] uses PoB, in which a node destroys some base currencies it owns in another blockchain in order to get a chance of proposing a new block. Slimcoin supports PoB based on Peercoin [70].
Proof-of-Elapsed Time (PoET)	Private	Sawtooth [46] uses PoET, in which each node runs a trusted hardware, for example Intel SGX [71], that generates random timers. The first node whose timer has expired can propose the next block.
Others	Public	Other protocols based on PoW are of the form *proof-of-X*, for examples: Proof-of-Activity [72], Proof-of-Space [73], Proof-of-Luck [74], etc.

表 3： 共识协议的比较

工作量证明变种：

所有工作量证明协议要求矿工基于加密哈希找到加密问题的解。具体来说，对于给定的哈希函数 H，一个阈值 t 和当前区块的内容 b，该解释一个随机数 n 使得：

$$H(n||H(b)) < t$$

最初在比特币中使用 SHA-256 作为这个哈希函数。用户通过硬件来加速哈希计算导致其他加密货币采用需要使用内存的哈希函数。以太坊使用 DaggerHashimoto 函数，Litecoin 和 Dogecoin 使用 scrypt，以及 ZCash 使用 Equihash 函数。这些函数能够抵抗 ASIC，因为它们要求大量内存方面的投资，但是他们很容易验证。

区块产生的速度与难题的难度有关。比特币将 t 设置为保证每十分钟一个区块的值。Litecoin,Dodgecoin 和 ZCash 降低 t 来将平均出块时间限制到几分钟。t 不能任意小因为这会导致区块链上没有必要的分叉。分叉不仅带来资源的浪费而且还带来安全问题，因为这会使双花成为可能。以太坊采用 GHOST[75]协议来将区块生成时间降低到几十秒，而不用牺牲很多安全性。GHOST 中，区块链可以

拥有尽可能多的分支，只要这些分支不包含冲突的交易。

权益证明

通过工作量证明挖矿是很昂贵的。这个过程特别耗能，估计其消耗的电力与像丹麦[76]这样的小国家消耗的电力相同。PoS 被提出来用以大幅降低挖矿的消耗。不像以太坊的 GHOST 协议，POS 维护了单个分支，但是对问题的难度进行调整，使其与矿工在网络中的权益成比例。一个股份本质上是一个有着一定余额的被封锁的账户，用以代表矿工的贡献以保持网络的健康。假设 s 是返回股份的函数，一个矿工 M 可以通过解决下面形式的难题来产生新的区块：

$$H(n\|H(b)) < s(M).t$$

可以看到股份 s(M)越大，越容易找到 n。

Peercoin 是比特币的一个分支，它是首先使用 PoS 的系统之一，它通过运行 PoW 生成代币。Peercoin 中的函数 s(.)以代币 C 作为输入，返回 C.age(C)，其中的 age(C)是代币 C 的年龄。Nxt[77]是另一个使用 PoS 的系统，它通过销售它的代币来逐渐部署它的系统。Nxt 中的函数 s(.)既考虑到矿工的零钱也考虑到从上一个区块开始消逝的时间。上一个区块距离现在越久，难题越容易解决。特别地：

$$s(M, b_h) = bal(M).age(b_{h-1})$$

其中 bn 是当前的区块，它所处的高度是 h，bal(M)返回 M 账户中代币的数量，age 返回从一个特定高度的区块开始已经经过了多少时间。

以太坊即将推出的 PoS 协议实现为了智能合约。简称为 Casper，它允许矿工通过向 Casper 账户存储以太币的方式成为验证者。合约接着根据存储的数目选取一个验证者来提议下一个区块。然而它的特点是强迫验证者行为正确，不然该验证者就会丢失它存储的全部以太币。特别地，每一个验证者都来对某个特定的区块是否会在未来确认打赌，如果区块被确认了，这个验证者将会得到一小部分奖励，否则，验证者将会丢失它全部的存储金额。这种机制避免了利益无关问题，这种问题是指验证者们能同时提议不同的分支。Tezos 实现了一个简化的 Casper 版本，即结点能够通过购买成为权威，这样可以能够对区块链底层的改变达成一致意见。区块链的软分支和硬分支问题是本身具有的，Tezos 旨在提供一个可修正的区块链。

PBFT 变种

PoW 问题在于非最终确定，即追加到区块链上的区块直到它被许多其他区块链接后才被确认。即使如此，它在区块链上的存在性也只是概率性的。例如，日食利用这种概率保证对比特币[78]进行攻击实现了双花。相反，最初的 PBFT 协议[17]是确定性的。这个协议在早一点的 Hyperledger 版本（v0.6）中实现，它确保了一旦一个区块被追加，它是最终的而且不能被代替或者修改。记 N 为网络

中节点的数目，协议每一轮的协定会产生$O(N^2)$的网络消息。然而在实际中，最初的协议扩展性很糟糕，甚至在达到网络限制[79]时就已经崩溃了，在使用BLOCKBENCH 框架对 Hyperledger 进行评测中,我们也遇到了相同的扩展性问题。

Tendermint 在 PBFT 之上提出了小的修改。在 Tendermint 中，每一个节点不再有相同的投票权，相反，每一个节点可能拥有与他们在网络中股份成比例的投票权力。为了在 Tendermint 中达到共识，仅聚集网络中全部投票权的三分之二是有必要的。在网络中存在一小部分节点拥有较高投票权的情况下，这种方法比等待网络中三分之二结点响应的代价要低。

目前与改进 PBFT 相关的工作主要关注它的性能。Zyzzyva[80]通过推测执行的方法来对通常情形(网络中没有失败)进行优化。XFT[81]假设网络中存在相比于纯拜占庭网络较少的敌对节点，并且通过降低网络中消息个数的方法展示了更好的性能。另一方面，HoneyBadger[82]主要关注在异步网络中提升安全性。它使用了一个随机化的共识协议，并且即便在网络异步的情况下达到了很大概率的安全性。通过优化网络层，它在网络同步情况下性能超过 PBFT。Zyzzyva，XFT 和 HoneyBadger 实现了很大的优化，但是它们都未被集成到任何区块链中。

可信硬件

PoW 和 PBFT 的大部分开销可以归因于网络中结点以拜占庭方式行事的假设。然而，Intel SGX[83]和 ARM TrustZone[84]的可用性使得放松拜占庭环境中的信任模型成为可能。特别是，一个配有可信硬件的节点能够可靠地检查某些属性，例如，它正在运行特定的软件。

Sawtooth Lake 使用 SGX 和一个更有效的称为 PoET 的协议代替 PoW。具体来说，PoET 运行在被 SGX 保护的环境中。它首先将区块号作为输入，接着产生一个随机的间隔 t 的定时器，这之后，它能能够提供一个证书表明从这个计时器启动经过了多少时间。PoET 生成最小 t 的结点能够在时间过期以后将区块附加到区块链上。特别地，该结点将 PoET 证书也附加到区块上，只要 t 比其他结点产生的值小，该节点就被接收。

A2M[85]和 Hybster[86]都通过使用可信硬件将为了容忍 f 个失败结点而需要的备份从 3f+1 降低到了 2f+1.这意味着有 N 个节点的网络能够最多容忍 N/2 个敌对的结点，这与最初的 PBFT 只能容忍 N/3 个结点不同。A2M 和 Hybster 的安全性依赖于可信代码库（TCBs），它实现了简单的功能:前者是数据结构日志，后者是单调的计数器。

联邦

尽管对最初的协议有大量的改进，基于 PBFT 的协议仍是受限于通信问题，因此在节点规模超过一定数目时，它最终会失败。为了在不牺牲安全性的前提下克服这个局限，Stellar 和 Rapple 采用了将网络分成小组的方法，这些小组称为联邦。每一个联邦在它的成员之中运行本地的共识协议，因为联邦内网络规模小，所以就不再有规模问题。本地的共识接着会可通过联邦交集的结点广播到全网。全局的共识在一定的条件下能够实现。对于 Stellar 而言，这个条件是每两个联邦的交集是非拜占庭节点。Ripple 的安全条件是在每一个联邦中都有大量的诚实节点，因此任意两个联邦的交集将至少包含一个诚实结点。

Stellar 和 Ripple 假设联邦是预先定义的，而且他们的安全性能够通过网络管理员加强。在分布式环境中，结点的身份是未知的，这种假设不再成立。Byzcoin[61]和 Elastico[26]提出了新颖的融合了 PoW 和 PBFT 的两阶段协议。在第一阶段，PoW 被用来形成一个共识组。Byzcoin 通过在区块链上维护一个滑动窗口，并且选择区块在这个窗口的矿工来实现这一点。Elastico[26]通过身份来分组，这些身份在每一个时期都会改变。特别地，每个节点的身份是一个加密问题的解。在第二阶段，被选择的结点运行 PBFT 来决定下一个区块。改进的结果是在规模超过传统 PBFT(超过 1000 节点)的情况下有着更快的区块确认速度。

与 Byzcoin 和 Elastico 类似，Dfinity[43]和 Algorand[27]在每一轮随机选择一组结点，这些结点可以提议新区快。与前者不同，它们不适用 PoW，相反后者使用可验证的随机函数（VRFs）来选择共识组。在 Dfinity 中，VRF 基于前一个区块的阈值签名。在 Algorand 中，函数基于前一个区块发布的随机种子和节点的私钥。

非拜占庭

这一节目前所讨论的系统都能够容忍拜占庭错误，这种特性使它们在公有环境和私有环境中都具有吸引力，这种环境中使用信任三方（例如，由第三方托管资产）的代价很高。然而，在一些区块链中，通过假设可信的第三方来简化它们的设计。这些区块链在任意一方有恶意行为时都没有安全性保证。

Openchain[39]依赖单个可信的第三方（称为验证者），由它来决定下一个区块。因此，它也最容易被攻击，因为只需要验证者结点失败即可。Multichain 和 Parity 拥有多于一个信任方，这些信任方在各自的系统中被称为权威机构，每一个权威机构都有一个时间片，通过轮询法进行调度，每一个权威机构在它的时间片内能够将新的区块添加到链上。这种简单的权威证明协议避免了单点故障，同时确保在权威机构之间工作负载平衡。HydraChain 和 BigChainDb 也有多个权威机构，但是一个权威不能单方面的决定下一个区块，相反，区块通过大多数投票来确定。Quorum[37]使用 Raft[22]作为权威机构之间的共识协议。Raft 实现了容错状态机复制，这是现代分布式系统的重要组成部分。通过使用 Raft，即使某些全为节点崩溃，Quorum 系统仍能安全运行。

Corda 的共识协议由一组称为公证人的可信方执行，这些公证人检查一个给定的交易在以前是否已经执行过。通过将检查委托给区块链外的实体，Corda 能够通过 Raft 来达到共识。Corda 系统中的交易先发送给这些公证人接着才在区块链中被确认。这些公证人使用 Raft 协议来确保交易在公证人之中重复并且即使有崩溃仍然能保持高可用性。

最新发布的 Hyperledger（v1.0）将共识组件外包给 Kafka，Kafka 是通常在分布式数据库系统中发现的另一个组件。更确切地说，交易被发送到一个中心化的 Kafka 服务，该服务将这些交易排列成一个事件流。每一个节点都订阅相同的 Kafka 流，因此能够按照交易发布的顺序收到新交易。因为系统中只有一个 Kafka 服务，因此每一个节点看到的交易顺序都是相同的。

其它

IOTA[40]使用它自己的称为 Tangle 的共识协议，这种协议中，区块形成一个有向无环图而非一个链。不仅如此，Tangle 里的区块只包括一个交易。当添加一个区块时，区块必须支持有向无环图中的两个其他区快，并且创建指向它们的

链接。当一个区块被很多其他去看快支持时，该区块才被确认。在 LoT 环境中，Tangle 的主要目标是效率及低成本支付。虽然它的安全性还未被严格的分析，Tangle 中的低价值的交易（小额支付）实际上可以阻止拜占庭行为。

Kadena[30]提出了能够处理拜占庭错误的 Raft 协议扩展。它在 Raft 之上引进了多种技术，诸如消息签名，客户认证和增量散列。然而，如 Tangle 一样，是否这个协议能够保证安全性和活跃性仍然是不清楚的。

4.4 智能合约

回想智能合约可以按照它的语言表达能力和执行环境来分类。除了 Openchain，IOTA，Ripple 和 Stellar 之外，表 2 中的所有系统都允许用户自定义交易逻辑来适应其应用程序中。接着，我们将区块链按照智能合约语言的表达能力来分类。

脚本：

比特币提供了大约 200 个操作码，但是在最新的实现中，操作码中的大部分是不可用的。用户可以用操作码编写基于栈的程序。比特币中最受欢迎的合约涉及多重签名。一个例子是托管合同，在使用代币时，其要求三分之二的签名。该语言也能实现悬赏风格的合约，例如只有找到一个哈希值的原像才能得到代币奖励。

BigchainDB[31]采用了称为加密条件的表达能力更强的合约，作为账本内部协议项目[87]的一部分来开发，加密条件允许在许多类型签名中指定复杂的布尔表达式。一个加密条件脚本包括条件和实现，这些是脚本的输入和输出。可用的条件包括超时，该条件使定时发布的合约称为可能。加密条件的编码比比特币的操作码具有更高层次，这也使表达复杂的逻辑更加容易。

图灵完备

以太坊是首先提供图灵完备智能合约的区块链中的一个。用户可以使用 Solidity，Serpent 或者 LLC 语言来编写他们的合约，这些合约接着在被编译成 EVM 字节码。EVM 能够执行标准的加密货币交易，而且它将智能合约字节码看作特殊的交易。特别是，每一个智能合约有它自己的内存来存储局部状态。内存展示为键值存储，尽管 Solidity 提供了高级数据类型，例如 map，array 和 composite 结构。智能合约在执行时的 CPU 和内存资源消耗由 EVM 纪录，而且向交易的发起者进行收费。EVM 也纪录中间状态改变，当交易发起者没有足够的资金来为执行付费时，EVM 会撤销该交易改变的状态。

Hyperledger 没有它自己的字节码。相反它在 Docker 容器中运行语言不可知的智能合约。具体来说，智能合约可以用任何语言来编写，接着它被编译成本地代码并被打包进 Docker 镜像中。当智能合约上传时，每个节点启动一个包含该映像的新容器。通过 Docker API 来调用智能合约。合约可以接口层的 getState 和 putState 方法来访问区块链的状态。Hyperledger 的一个好处是它支持多种诸如 Go 和 Java 的高级编程语言。然而，它区块链的键值对接口需要额外的应用程序逻辑来将高级数据结构映射到键值元组。

Sawtooth Lake 支持交易族形式的智能合约。每一个族的个体是一个由用户定义的 python 类，它在启动时被加载进账本。智能合约像正常的 python 程序一样在本地运行环境中执行。

支持图灵完备的智能合约的一个后果是软件的漏洞是不可避免的。以太坊的智能合约尽管增加了自主性，但是该模型因为使用以太币来解决程序漏洞而收到了强烈的批评。DAO 攻击[47]确实体现了安全问题，通过该攻击，攻击者偷了价值 20M 美元的资产。攻击利用了 DAO 智能合约中的并发的漏洞，该漏洞允许用户重复的取出多于交易中指定的金钱。这种漏洞是像 EVM 这类语言所固有的，因为它们对语义有很弱或者几乎没有形式化的规范。OYENTE[88]展示了安全漏洞的三种主要原因：交易顺序依赖性，时间戳依赖性和处理不当的异常。它形式化了以太坊语义并且提出了一个工具基于 EVM 字节码来检查异常。这个工具发现了超过 8000 以太坊合约（价值超过 6000 万美元）存在安全漏洞。

像区块链上的任何其它交易一样，智能合约的执行时透明的。这意味着输入和输出以及合约的状态在网络中是可见的。Hawk[89]扩展了 Zerocash 来为智能合约提供交易隐私性。和 Zerocash 相比，最大的挑战在于任意的交易逻辑，而在 Zerocash 中，逻辑受到一小部分操作的限制。另一个挑战时保护本地状态，这在 Zerocash 是不适用的。给定一个合约，Hawk 使用 zkSNARK 编译它来使它的隐私性得以保存。交易的输入和输出通过 Hawk 进行预处理和后处理，以隐藏复杂的加密细节。尽管协议带来了大量的时间和空间开销，Hawk 展现了一个实用的加密系统，其实现了交易的隐私性和公平性。

可验证

在 DAO 攻击之前，一些区块链拒绝一些允许无限制计算的模型。Kadena，Tezos 和 Corda 系统中的语言比比特币的脚本更加强大，但是他们牺牲了图灵完备来保证安全性。Kadena 的语言是一个称为 Pact[30]的类似于 Lisp 的函数语言。一个 Pact 合约以人类可读的形式存储在账本中，该合约接着在 Ocaml 中解析和执行。它是强类型的，且能够被形式化的验证。同样地，Tezos 系统中称为 Michelson 的基于栈的语言具有强大的类型系统和完备的语义。因此 Tezos 合约能够通过静态的检查保证其安全性。在 Corda 中，合约是一连串不对状态进行修改的函数。因为函数仅是约束条件，因此合约的安全性能够被形式化和验证。

5 BLOCKBENCH

先前的章节对于现存的区块链展示了一个全面定性的分析。这一章节，我们描述了称为 BLOCKBENCH[10]的基准测试框架。该框架对将区块链作为数据处理平台进行定量分析，其针对区块链为私有链且具有图灵完备智能合约。BLOCKBENCH 是开源的[90]，包含数据库基准测试中常用的数据处理工作负载。

5.1 层次

BLOCKBENCH 面向的目标是作为数据处理平台的区块链。这样的区块链必须在应用逻辑上没有限制，因此，它必须支持图灵完备的智能合约。图 3 展示了区块链软件层的逻辑组件，我们将第四节描述的分类法细化为图 4 所示的四个具体层次。对于每一层，都有多个 BLOCKBENCH 工作负载来进行单独评估。

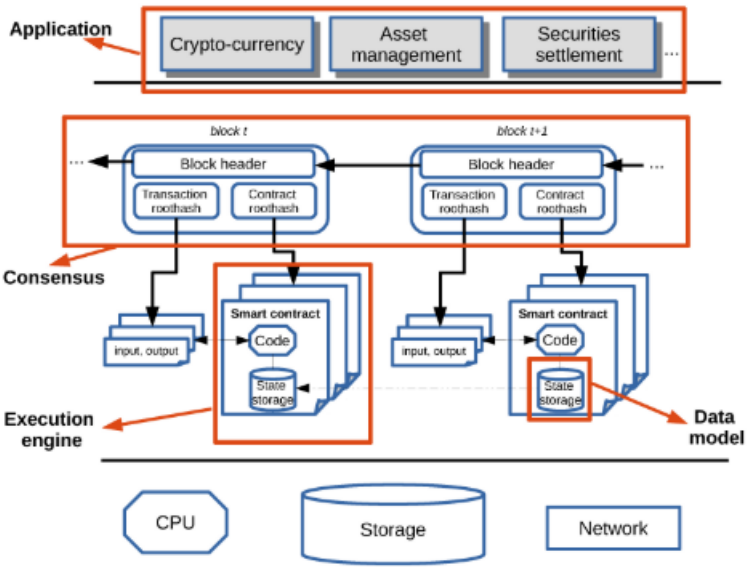


图 3：区块链合法节点上的软件层次

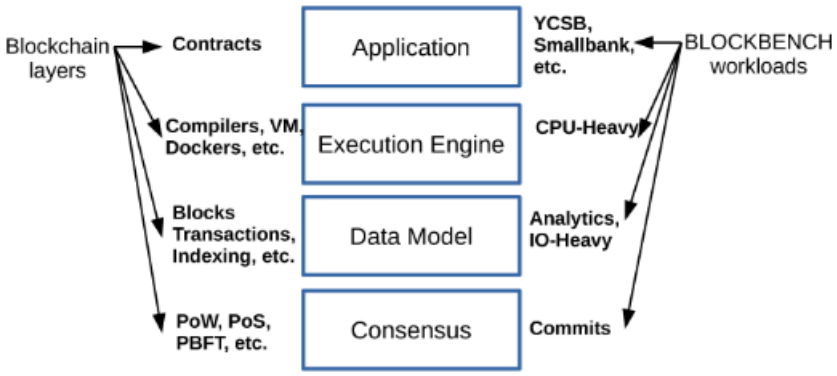


图 4：区块链软件层和具体的工作负载

共识层实现了共识协议。数据层包括区块链上数据的结构，内容和操作。执行层包括执行智能合约执行时运行环境。最后，应用层包括区块链应用类。Croman 等人[20]提出将区块链分成几部分:网络，共识，存储，视图和侧面。尽管和 BLOCKBENCH 的四个层次类似，这种平面抽象是面向加密货币应用的，没有考虑到智能合约的执行。

5.2 实现

BLOCKBENCH 栈包括一个前端接口和驱动，前者用来集成新的区块链，后者用来驱动工作负载。一个新的区块链能够通过实现 IBlockchainConnector 接口来集成到框架的后端，这些接口包括部署智能合约的操作，可以通过发起交易来调用该接口，且该接口可用来查询区块链状态。Ethereum，Parity 和 Hyperledger 是当前框架的后端，ErisDB（或者 Monax），Quorum 和 Sawtooth Lake 的集成正

在部署中。一个新的基准测试负载可以通过实现 IWorkloadConnector 接口添到框架中。驱动将工作负载作为输入，根据用户定义的配置文件（操作的数据，客户，线程的数目等）将交易发送到区块链中。它收集运行时数据，并将其用于计算五个重要度量。

吞吐量：每秒成功交易的数目，一个用户可以通过配置大量的客户和线程来使区块链吞吐量饱和。
延迟：每一笔交易的响应时间。驱动实现区块交易，例如它等到一笔交易完成后才开始下一笔。

可扩展性：增加结点数量和并发工作负载时吞吐量和延迟的改变。

容错：结点失败时吞吐量和延迟的改变。我们模仿了崩溃，网络延迟和随机消息损坏。

安全指标：主链中区块数目和确认区块的数量的比值。这个比值越小，这个系统越容易受自私矿工的双花攻击。

5.3 工作负载

BLOCKBENCH 使用宏基准工作负载评估应用层，使用微基准工作负载分析较低的层次。图 4 展示的工作负载的智能合约实现是可用的而且能够很容易部署到 Ethereum，Parity 和 Hyperledger 上。

宏基准工作负载：我们将两个著名的数据库基准工作负载移植到 BLOCKBENCH 中，即 YCSB 和 Smallbank。YCSB 在评估 NoSQL 数据库时被广泛使用，因此我们实现了一个简单的智能合约，该合约充当键值存储。WorkloadClient 基于 YCSB 驱动[91]，该驱动能够将一定数目的纪录预加载到每一个存储中，并且支持不同比率读写操作的请求。对于 Smallbank[92](一种 OLTP 工作负载的流行基准)，我们实现了一个将金钱从一个账户转移到另一个账户的智能合约。
除了数据库工作负载，BLOCKBENCH 也提供了三种基于现实以太坊合约的工作负载。第一个是 EtherId，该合约实现了域名注册。第二个是 Doubler，这个金字塔计划合约，该合约在图二中展示。第三个是 WavesPresale，它通过数字代币销售实现了众筹运动。

微基准工作负载：对于共识层，BLOCKBENCH 提供了 DoNothing 工作负载，该合约仅仅接收一笔交易作为输出接着返回。因为合约执行包括包括执行层和数据模型层最小数目的操作，综合的性能将通过共识层决定。

对于数据模型层，BLOCKBENCH 提供了和 OLAP 工作负载相似的分析工作负载。特别地，它执行扫描式和聚合查询，其性能由系统的数据模型决定。具体来说，有两个查询：

Q1：计算区块 i 和区块 j 之间提交的总交易值。
Q2：计算给定一个状态（账户），计算区块 i 和区块 j 之间最大交易值。

对 Ethereum 和 Parity 而言，两个查询都通过 JSON-RPC APIs 实现，该 API 返回一个特定区块的交易细节和账户余额。然而，对于 Hyperledger 而言，第二

个查询必须通过一个智能合约（VersionKVStore）实现，因为 Hyperledger 没有查询历史状态的 API。图 5 展示了 Hyperledger 中智能合约的执行。为了支持历史数据查询，合约为每一个账户的键值附加了一个计数器，获取一个账户特定的版本使用了键值 account:version。最新的版本存储在键值 account:latest 中 。合约同时为每一个版本在数据域维护了一个 CommitBlock 值来指向当前版本提交的区块。为了从一个给定区块范围内获取到某一账户的余额，合约扫描该账户的每一个 CommitBlock 值在给定范围内的版本并且返回对应的余额。

另一个数据模型层的工作负载强调持续的存储。特别地，IOHeavy 工作负载通过调用一个对本地状态大量随机读写的合约评估区块链的 IO 性能。

```
type account_t struct {
    Balance     int
    CommitBlock int
}
type transaction_t {
    From string
    To   string
    Val  int
}
func Invoke_SendValue(from_account string,
        to_account string, value int) {
    var pending_list []transaction_t
    pending_list = decode(GetState("pending_list"))
    var new_txn transaction_t
    new_txn = transaction_t {
        from_account, to_account, value
    }
    pending_list = append(pending_list, new_txn)
    PutState('pending_list', encode(pending_list))
}
func Query_BlockTransactionList(block_number int)
     []transaction_t {
    return decode(GetState("block:"+block_number))
}
func Query_AccountBlockRange(account string,
        start_block int, end_block int)
     []account_t {
    version := decode(GetState(account+":latest"))
    var ret []account_t
    while true {
        var acc account_t
        acc = decode(GetState(account+":"+version))
        if acc.CommitBlock >= start_block &&
           acc.CommitBlock < end_block {
            ret = append(ret, acc)
        } else if acc.CommitBlock < start_block {
            break;
        }
        version -= 1
    }
    return ret
}
```

图 5：用于分析工作负载的 VersionKVStore 智能合约的代码片。

最后，对执行层而言，BLOCKBENCH 提供 CPUHeavy 工作负载。它通过调用一个对大规模数组进行快排的合约来评估执行层对大量计算任务的效率。

6 评估

我们使用 BLOCKBENCH 对 Ethereum，Parity 和 Hyperledger 进行了比较研究。它们在设计领域占据了不同的位置，而且被认为在代码库和用户群方面最为成熟。我们使用 Ethereum 最流行的 Go 语言实现，geth v1.4.18，Parity 发布的 v1.6.0 版本。除非明确指出，Hyperledger 的版本是 v0.6.0-预览版。我们通过定义一个创世快且直接在矿工网络添加节点为 Ethereum 和 Parity 建立了一个私人测试网络。对于以太坊，我们手动调整了创世区块的难度变量以确保矿工在大型网络上不会发生分歧。对于 Parity，我们将 stepDuration 变量设置为 1。在 Ethereum 和 Parity 中，confirmationLength 的值是 5 秒。默认 Hyperledger 的批量大小是 500.

实验运行在 48 个节点集群上。每一个节点都有 E5-1650 3.5GHz CPU，32GB RAM，2TB 硬盘，运行在 Ubuntu 14.04 Trusty 系统上，通过 1GB 的交换机与其他节点连接。对于 Ethereum，我们从 12 个核中保留了 8 个，使客户端驱动进程的顶起轮询不会干扰挖矿进程。我们主要的研究结果如下:

Hyperledger 在基准测试中表现始终比 Ethereum 和 Parity 好。但是它无法扩展超过 16 个节点。

Ethereum 和 Parity 对节点故障更具有适应能力，但是它们容易受到区块链分叉的安全攻击。

Hyperledger 和 Ethereum 主要的瓶颈是共识协议，但是 Parity 的瓶颈则是来自交易签名。

Ethereum 在内存和磁盘使用方面产生大量的开销。它们的执行引擎同样比 Hyperledger 的更低效。

Hyperledger 的数据模型是低层次的，但是它的灵活性可以为分析查询定制优化。

6.1 宏基准测试

这一节我们使用 YCSB 和 Smallbank 基准框架从应用层讨论了区块链的性能。

吞吐量和延迟

图 6 显示了 8 个服务器和 8 个并发客户端在 5 分钟内的最高性能。我们发现就吞吐量而言，Hyperledger 在两个基准测试中都优于另外两者。Hyperledger 和 Ethereum 之间的差距是因为共识协议的不同:一个基于 PBFT，另一个基于 PoW。8 个 server 的系统中，广播消息的通信消耗小于难度设置为大约 2.5s 一个区块的挖矿时间消耗。Parity 和 Hyperledger 之间的差距不是由于共识协议，因为 Parity 的 PoA 协议相比于 PoW 和 PBFT 是更简单且更有效的。相反，我们观察到 Parity 以一个固定的速率处理交易，这将最大客户请求速率限制在 80tx/s。

为了将它们的性能表现放在具体情境下，我们使用 YCSB 和 Smallbank 工作负载将这三个区块链和著名的 H-Store 内存数据库做了比较。区块链和数据库在设计目标方面并不相同：前者不是为了通用的数据处理而设计的，后者也不是为了在拜占庭错误下保护数据完整性而设计的。而且，我们认为这个比较给了我们这两个系统在设计权衡方面和相对的性能方面有用的见解。我们运行 H-Store 自己的基准测试驱动并且把交易速率设置在 100000tx/s。图 7 展示了两个系统在吞吐量上一个数量级和延迟方面两个数量级的差距。特别地，H-Store 实现了 140Ktx/s 的吞吐量，同时维保持了亚毫秒级的延迟。性能方面的差异主要是由

于共识协议。例如，对于 YCSB，H-Store 几乎不需要对等方之间的协调，而 Ethereum 和 Hyperledger 则需要承受 PoW 和 PBFT 的开销。一个有趣的观察是 Smallbank 的开销。回忆一下，相比于 YCSB，Smallbank 由更复杂的交易组成，其中多个关键字在单个交易中更新。Smallbank 很简单，但是却代表了大量交易的工作负载，例如 TPCC。我们观察到在 H-Store 中，Smallbank 的吞吐量比 YCSB 低 6.6 倍，延迟比 YCSB 高 4 倍，这也反映了分布式事务管理的成本。相反，区块链性能下降幅度适中：吞吐量下降 10%，，延迟下降 20%。这是因为区块链中的每一个节点维护了完整的状态，既然没有分区，它也没有协调分布式事务的开销。

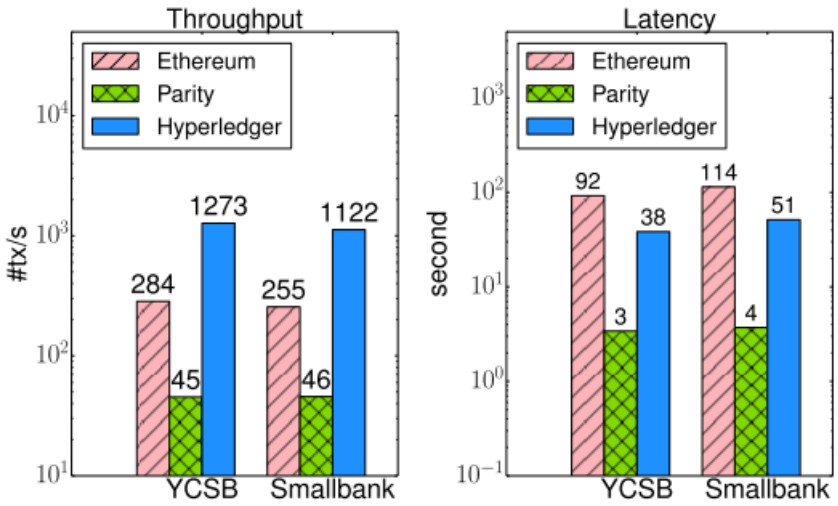


图 6：区块链在 8 个客户端和 8 个服务器时的最高性能。基准工作负载时 YCSB。这和 Smallbank 工作负载的性能是类似的。

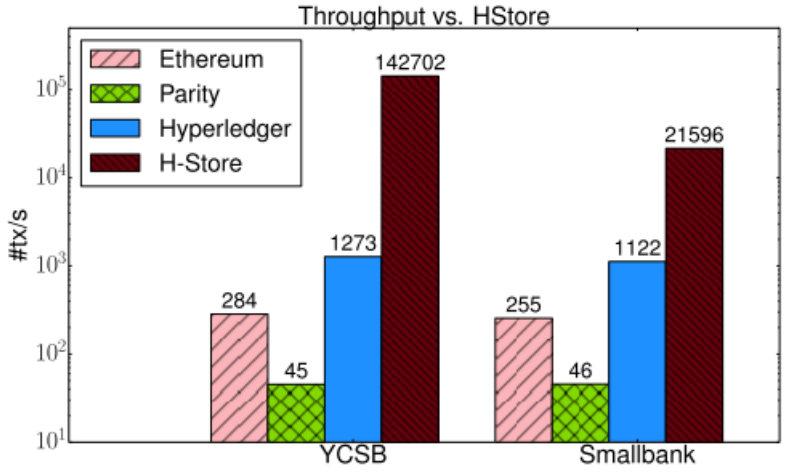


图 7：区块链和 H-Store 的性能比较

可扩展性

我们固定客户请求速率（Hyperledger 中每秒 320 个请求，Ethereum 和 Parity 中每秒 160 个请求），增加客户和服务器的数量。图 8 展示了三个系统如

何扩展已处理更大的 YCSB 工作负载。由于服务器恒定的处理速度，随着网络的规模和工作负载的增加 Parity 的性能始终是固定的。有趣的是，尽管 Ethereum 的吞吐量和延迟性能在超过 8 个服务器之后线性降低，Hyperledger 在超过 16 台服务器以后停迟工作。

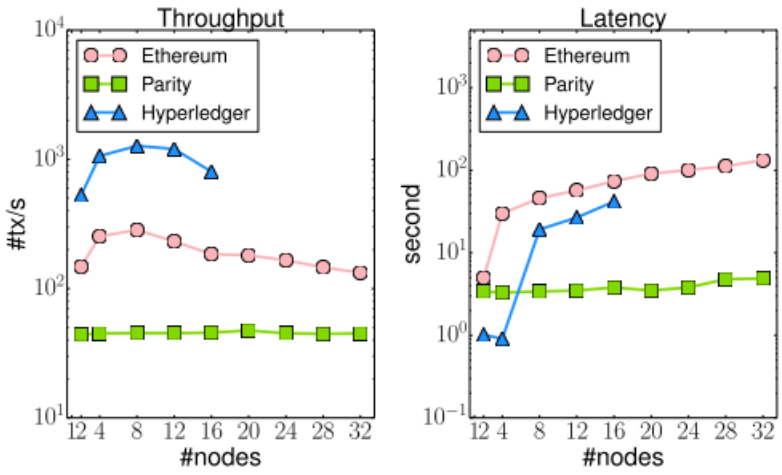


图 8：性能的可扩展性（相同数目的客户和服务器）。基准工作负载是 YCSB。Smallbank 基准工作负载可扩展性与其类似，只是 Hyperledger 在 8 个而非 16 个节点时就已经故障。

为了理解 Hyperledger 在 16 个服务器和 16 个客户端时不能扩展的原因，我们检查了系统的日志找到节点在任意一批交易中都未能达到共识。即使少于 16 个服务器和客户机，也有大量的消息被丢弃。此外，服务器重复触发试图更改但是却从未成功。在客户端，随着时间流逝，请求花费的时间更长，这表明服务器或者网络已经饱和。由于原始的 PBFT 协议能保证活跃性和安全性，我们可以将失败归因于 Hyperledger 的实现。进一步的调查揭示事实确实如此。

Hyperledger 使用 gRPC 来在服务器间通信。每一个服务器都为网络中每个其他服务器维护一个单独的消息队列。队列的大小在初始化步骤中定义（默认大小是 1000 个消息）而且当队列满时，默认行为是丢弃消息。在当前的设计中，客户端请求（交易）和共识消息（预处理阶段，准备阶段，提交阶段，视图更改）都发送到相同的通道上，即它们共享相同的队列。对于大量并发的客户和服务器，通道被客户请求占领，增加了共识消息被丢弃的可能性。没有充足的共识消息，批量计数器或者视图更改计数器会过期。在第一种情况下，PBFT 的领导者重新发送当前轮次的共识消息。在第二种情况下，服务器启动视图更改阶段，这会广播多轮共识消息。当客户请求始终占据网络通道时，共识消息和视图更改信息都有很大概率被丢弃。因此，网络陷入永久尝试建立稳定视图循环之中。PBFT 对网络条件敏感在过去[79]就已发现。

我们注意到最新版本（v1.0）的 Hyperledger 将 PBFT 代替为了全局排序服务。使用 Kafka 实现，这种新的共识引擎可能提供比 PBFT 更高的吞吐量。但是它对拜占庭错误没有任何保护。

到目前的结果表明，扩展客户端和服务器会降低性能，甚至导致 Hyperledger 失败。我们接着检查了将客户端数目固定为 8，只增加服务器数量时性能的损耗。图 9 显示了随着服务器的增多，性能变差，这意味着系统产生了一些网络开销。对于 Hyperledger 而言，有更多服务器意味着更多消息交换和更高的开销。特别地，为了对较大的网络中一批交易确认，Hyperledger 中的领导者需要等待更多消息，因此会降低整体吞吐量。我们注意到固定客户端数目，Hyperledger 能够扩展到 32 个节点，这与图 8 中 16 个结点失败相反。这是因为客户端越少，每一个节点的消息队列不会因为客户请求饱和因此共识消息也不太可能被丢弃。

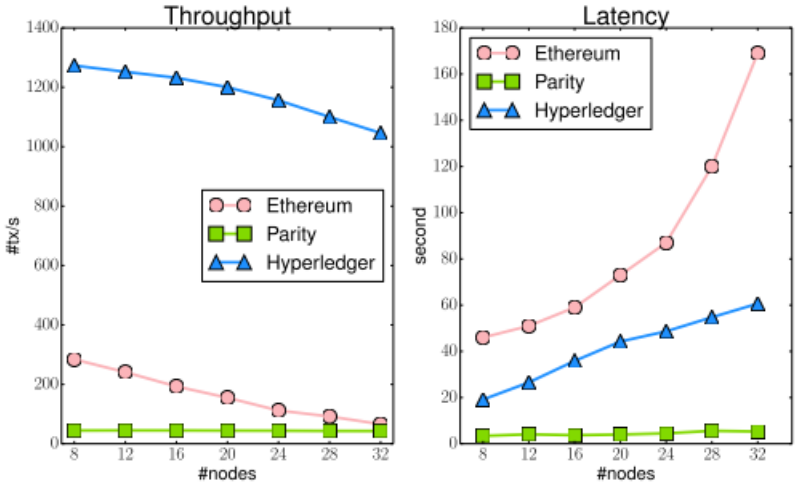


图 9：扩展性性能（8 个客户端）

对于以太坊而言，即使有计算界限，广播交易和区块仍会消耗适量的网络资源。此外，在网络较大的情况下，为了解决较长的网络延迟，难度也会增加。我们观察到为了阻止网络分叉，难度级别比节点数量增加速率更高。因此以太坊吞吐量降低的一个原因是网络的大小。另一个原因是我们的环境，8 个客户端仅仅将请求发送给 8 个服务器，但是这些服务器不将交易广播给对方（他们继续在自己的交易池中挖矿）。结果是网络挖矿的能力没有被充分利用。

6.1.1 容错和安全性

为了评估系统如何应对崩溃，我们运行有着 12 和 16 太服务器的系统，其中 8 个客户端运行超过 5 分钟，我们在第 250 秒时关闭 4 太服务器。由于空间限制，我们只突出最关键的部分，读者可以从[10]中找到更多细节。首先，Ethereum 不受这种改变的影响，这表明失败的服务器对于挖矿没有显著贡献。第二，Parity 的吞吐量也不受影响。这是因为每一个节点都有相同的时间片，在该时间片内服务器能够产生区块，因此在 Parity 中 4 个节点失败意味着剩下 8 个结点被赋予更大的时间片。第三，Hyperledger 在 12 个服务器的网络中发生故障后停止产生区块，这是因为 PBFT 在 12 个服务器的网络中只能容忍 4 个节点的失败。在 16 个服务器的网络中，Hyperledger 仍能产生区块，但是速度减慢，这是因为剩下的服务器在失败后只有通过同步视图达到稳定后才能继续工作。

我们接下来模拟了能够使区块链遭受双花的攻击。该攻击本质上是从第 100 秒开始持续 150 秒创建网络分支。我们将分区的大小设置为原来的一半。图 10 比较了 8 个客户端和 8 个服务器的三个区块链的易受攻击性。回忆一下，易受攻击性是根据区块总数和主分支区块数的差异来衡量的。我们称其为Δ。Ethereum 和 Parity 在第 10 秒产生分叉，随着时间的推移Δ增加。在攻击的时间内，高达 30%的区块在分支中生成，这意味着系统更容易受双花或者自私矿工的攻击。当分区合并时，结点开始在主链达成共识并且丢弃分叉的区块。结果是，Δ在 250 秒后不久就会停止增加。与之形成鲜明对比的是，Hyperledger 没有预期的分叉，因为它的共识协议被证明能够保证安全。然而我们注意到，Hyperledger 相比于另外两个系统需要更长时间才能从攻击中恢复（大约 50 秒）。这是因为在分区节点重新连接后执行同步协议。特别地，当节点重新连接后，它们进入到视图更改阶段，并且相互之间交换检查点来建立新的稳定的视图。

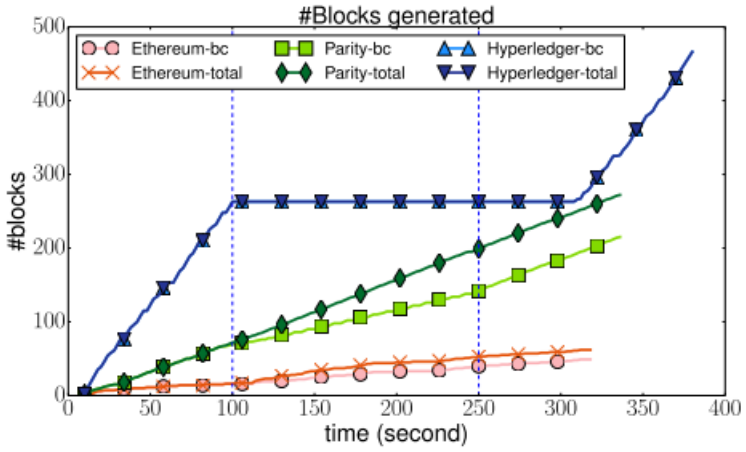


图 10：区块链由于第 100 秒开始持续 150 秒的网络分区（各 1/2 节点）攻击产生的分叉。X-total 表示区块链 X 中产生的区块总数。X-bc 表示区块链 X 上达到共识的区块数目。

6.2 微基准测试

这一节讨论了区块链执行层，数据模型层和共识层性能表现。前两层，工作负载运行在一个客户端一个服务器的系统中。至于共识层，工作负载运行在 8 个客户端和 8 个服务器的系统中。

执行层

我们部署了 CPUHeavy 智能合约，它使用一个给定大小的整数数组来初始化。这个数组按照降序进行初始化。我们调用该合约使用快排算法对其进行排序，并测量执行时间和服务器使用的最大内存。图 11 显示了不同输入大小的结果。尽管 Ethereum 和 Parity 使用了相同的执行引擎，即 EVM，Parity 的实现更优，因此它具有更好的计算能力和更好的内存效率。一个有趣的发现是 Ethereum 产生了大量的内存开销。在给 100M 元素排序时，它使用了 22GB 内存，而 Hyperledger 只使用了 473MB。Ethereum 在给多余 10M 元素排序时耗尽了内存。Hyperledger

的智能合约在本机的 Docker 环境中编译和运行，因此它没有与执行高层次 EVM 字节码相关的开销。结果是，Hyperledger 在速度和内存仿效效率更高。最后我们注意到这三个系统未能利用多核架构，即他们只是用一个核来执行智能合约。

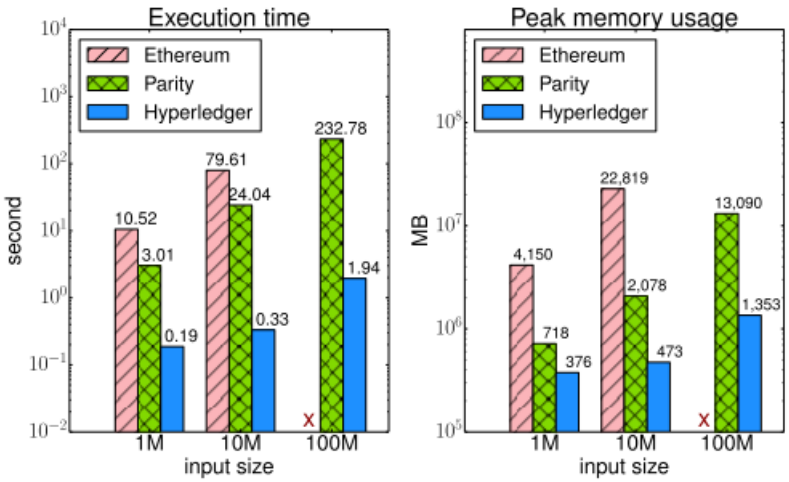


图 11： CPUHeavy 工作负载，‘X’表示内存溢出错误

数据模型-IOHeavy

我们部署了 IOHeavy 的智能合约，它执行大量键值元组的读写操作。我们使用了 20 字节的键和 100 字节的值。图 12 展示了这些操作的吞吐量和磁盘使用情况。Ethereum 和 Parity 使用相同的数据模型和内容的索引结构，因此他们产生了相似的空间开销。两者的存储空间都比使用简单键值对模型的 Hyperledger 多了一个数量级。Parity 将所有的状态信息存在内存中，所以它具有更好的 I/O 性能，但是无法处理大量数据（在我们的硬件环境设置下，超过 30M 状态）。相反，Ethereum 仅仅将部分状态信息缓存到内存中（使用 LRU 驱逐策略），因此以吞吐量为代价，相比于 Parity，它能够处理更多数据）。Hyperledger 利用 RocksDB 来管理其状态，这使其在大规模数据方面更加高效。

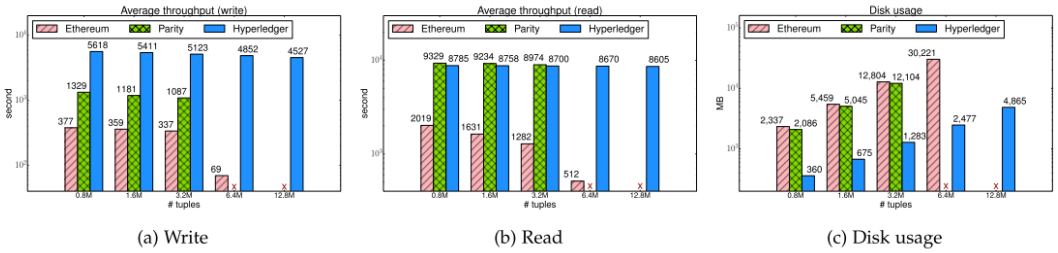
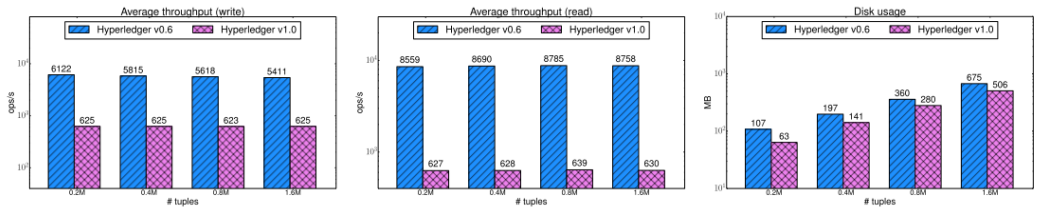


图 12：IOHeavy 工作负载，‘X’表示内存溢出错误

Hyperledger v1.0. 我们使用相同的 IOHeavy 智能合约来比较 Hyperledger v1.0 版本的 I/O 性能。如图 14 所示，v1.0 的吞吐量比 v0.6 低一个数量级。此外，v1.0 在多于 0.8M 操作时崩溃，并且报告消息过大的异常。主要的差异归因于 v0.6 到 v1.0 系统架构的改变，前者结点参与 PBFT 来确认一个区块，在这种

情况下，IOHeavy 工作 load 的交易不产生任何共识开销，因为只有一个节点。后者中，一个新的服务，串行者被引入网络中来为交易排序并且提供共识。因为这种新服务，IOHeavy 工作负载的交易需要和串行者沟通来使其确认。更加具体来说，与 v0.6 相比，v1.0 多执行三个步骤来完成交易。因为通信消耗增加，吞吐量降低。结果表明将 PBFT 代替为中心化服务不仅无法保护区块链免遭拜占庭式失败，同时可能损害综合性能。

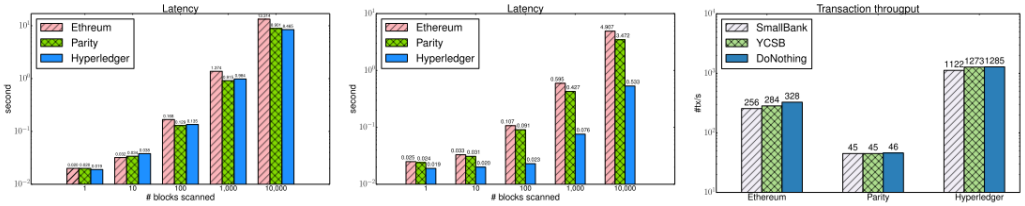


(a) Write (b) Read (c) Read

图 14：Hyperledger v0.6.0 和 v1.0.0 在工作 IOHeavy 工作负载上的比较。

数据模型-分析

我们通过使用有着固定余额的超过 12 个账户对三个系统进行初始化，来实施分析工作负载。我们接着预加载了 10 万个区块，平均每一个区块包含 3 笔交易。交易从一个随机的账户向另一个随机的账户转一定价值的钱。由于在很多账户时 Parity 在交易签名上的开销，我们仅使用了 1024 个账户进行交易。我们接着执行 5.3 节描述的两个查询并且测量它们的延迟。图 13 显示 Q1 在三个系统中表现相似，然而 Hyperledger 和其他两个系统在 Q2 上表现有着很大的差异。。我们注意到 Q1 和 Q2 主要的瓶颈是由客户端发送的网络请求(RPC)的数量。对于 Q1 而言，所有系统中客户端发送相同数目的请求，因此它们的性能表现相似。另一方面，对于 Q2 而言，Ethereum 和 Parity 每一个区块发送一个 RPC 请求，因为我们对智能合约实现的优化，Hyperledger 值发送一个 RPC 请求。这节省了网络往返时间，将 Q2 的延迟提高到 10 倍以上。



(a) Analytics workload (Q1) (b) Analytics workload (Q2) (c) DoNothing workload

图 13：分析和 DoNothing 工作负载

6.2.1 共识

我们部署了 DoNothing 智能合约，其接受一笔交易并且立即返回。我们测量了这个工作负载的吞吐量，并将其与 YCSB 和 Smallbank 进行了比较。与其它工作负载相比，图 13 显示的差异直接表明共识协议和其他软件堆栈的成本。特别地，我们观察到 EthereumDoNothing 相比于 YCSB 在吞吐量上提升了 10%，这意味着执行 YCSB 交易占 10%的开销。我们观察到这些工作负载在 Parity 中没有任何不同，因为它的瓶颈是消息签名（即使空交易仍需要签名），而非交易执行。

7 讨论

在这一节，我们首先从 Ethereum、Parity 和 Hyperledger 的比较实验中提取出一些经验教训。接着我们讨论了如何将数据库系统的设计原则用来帮助提升区块链的性能。

7.1 性能实验中得到的经验教训

理解区块链系统：BLOCKBENCH 旨在促进对不同私有链设计和性能方面更好的解。随着越来越多的区块链系统的提出，每一个系统都有一些不同的特点，BLOCKBENCH 的主要价值在于它将设计空间缩减到了四个不同的抽象层。四个层次从我们在第四节的分类中得到，这些层次充分的捕捉到了区块链系统关键且细微的特征。通过对这些系统进行基准测试，我们能够对设计权衡和性能瓶颈获得深入的理解。例如，通过使用 IOHeavy 工作负载，我们确定 Parity 通过将状态存在内存中，降低了性能获得了更好的可扩展性。此外，工作负载揭示了最新版本 Hyperledger 潜在的性能问题。另一个例子是 Analytics 工作负载表明系统在数据模型上的权衡。特别地，Hyperledger 的简单键值对模型意味着一些分析的查询语句不能被直接支持。然而，它允许优化以帮助提高分析查询语句的效率。最后，我们确定 Parity 系统中的瓶颈不是由于共识协议，而是因为服务器的交易签名。我们认为，没有系统的分析框架，获得这些见解是不容易的。

区块链的可用性：我们的对三个区块链系统的工作经验证实在目前的状态下，区块链还不能支持大量使用。它们的设计和代码库仍在不断地改进，而且除了加密货币也没有其他已经建立的应用。三个系统中，Ethereum 在代码库、用户群和开发者社区方面较为成熟。我们遇到的另一个可用性问题是关于如何将智能合约从一个系统移植到另一个系统，因为他们的编程模型不同。随着更多区块链平台的提出，这个问题可能进一步恶化[6],[30],[44],[93].

与数据库系统比较：区块链与上一中介绍的 H-Store 的比较表明区块链在当前数据库系统的数据处理任务中表现不佳，虽然数据库在设计时未考虑安全性和拜占庭容错问题，我们注意到这个间隔对于区块链太大而不能对现任的数据库系统造成破坏性影响。在高性能数据处理方面，区块链能从数据库中学到很多，我们将在下面讨论这些。同样，数据库也能从区块链的流行和成功方面学到很多有用的经验教训。可能最有趣的经验是，对于能够容纳大量用户且能容忍拜占庭错误的数据库处理系统是有需要的。分布式数据库通过假设非拜占庭错误而与 P2P 系统设计不同[18]，但是随着更快且更可信硬件可用性的提高，社区可能需要将兴趣转移到更高性能、分布式、P2P 数据库系统中。

7.2 将数据库设计思想运用到区块链中

通过优化共识协议来扩展区块链所面临的挑战正在许多最近的文献[26],[61]中得到解决。而且，我们的比较学习表明区块链存在其他的性能瓶颈。图 15 描述了一笔交易在被认为在区块链中提交之前所经历的不同阶段。每一个

阶段都可能是一个潜在的瓶颈，并且有待未来的优化。首先交易批量打包为一个区块，接着区块称为共识协议的输入，一旦区块被协议选择，它会被发送到执行引擎。最后，该引擎执行整批交易，创建新的状态并且将该区块追加到链上。我们注意到这个过程与分布式数据库中事务流具有惊人的相似性。事实上，唯一的不同是共识协议：数据库使用两阶段提交协议或者称为 Paxos，然而区块链使用容忍拜占庭的协议或者其变种。鉴于这种相似性，我们提出了四种数据库中的设计原则来提高区块链的性能。

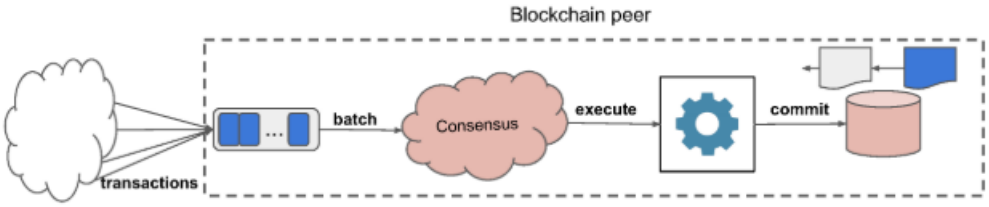


图 15：区块链交易的生命流程

解耦各层并且分别优化:一个可能的方向是将存储，执行引擎，和共识层彼此分离并且单独优化和扩展。例如，Tezos 和 Corda 通过将共识层外包给独立的一方来将其分离。数据模型层同样能够分离。例如，当前的系统使用通用的键值对存储，这对区块链上某种数据结构和操作可能并非特别适合。UStore[94]表明依据区块链数据结构设计的存储相比于现存的实现能够获得更好的性能。更重要的是，我们观察到 Ethereum 和 Hyperledger 目前的数据模型对于回答分析查询不是特别理想。特别地，他们都不支持交易级别的细粒度版本查询：除非查询区块级别的状态和处理每一个区块的状态，查询一个状态上一个版本不能够立即得到响应。当存储层和区块链其他部分分离时，实现一个新的数据模型不是很复杂。

使用新的硬件原语：许多数据处理系统通过使用新的硬件来提升性能[95],[96],[97]。对于区块链而言，通过使用可信硬件，底层的拜占庭容错协议能够被修改产生更少的网络消息[85]。通过可信硬件，区块链能够容忍更多错误，并且由于较少的网络消息，它也能够具有更好的可扩展性。像 Parity 和 Ethereum 这样的系统能够利用多核 CPU 和大容量存储器来提高合约执行和 I/O 性能。

分区：区块链本质上是一个重复状态机系统，在这个系统中每一个节点都维护相同的数据。因此区块链与像 H-Store 一样的数据库系统跟本上是不同的，在这些数据库系统中数据在每个节点分区存储。分区能够降低计算成本，并且可以使交易处理得更快。分区主要的挑战是如何在多个分区中确保一致性。然而，现存的数据库系统一致性协议并不能容忍拜占庭错误。尽管如此，它们的设计可以为实现区块链更具有可扩展性的分区协议提供见解。Corda 将账本分为属于较小组的子帐本，从而避免了全网广播交易的需要。然而，它始终依赖于一个外部的，集中的组件来使子帐本间达到共识。最近的工作[26]表明分区共识协议的可行性，使整个区块链分区提供了重要步骤。

支持声明性语言：拥有一套可以以声明方式组成的高级操作，可以使定义复杂的

智能合约成为可能。它同时支持低层次的优化来提高执行效率。Hawk[89]是最近的一个将编译技术用来实现智能合约隐私保护的例子。然而，它旨在加强智能合约的安全性而非提高它们的性能。

8 结论

在本文中，我们对区块链技术进行了全面的调查。我们展示了区块链背后的四个基础概念，并且使用这些概念分析了最新的技术。我们展示了我们的基准框架，BLOCKBENCH，它被设计来评估区块链作为数据处理平台的性能。最后，我们讨论了四个受到数据库设计原则启发的研究方向，用来提高区块链的性能。我们希望调查和基准测试框架能够作为未来设计不仅安全，而且可在现实世界扩展 和使用的区块链系统设计和实现的指导，。

9 致谢

这项工作由国家研究基金会，新加坡总理办公室根据其竞争研究计划（CRP Award No. NRF-CRP8-2011-08）提供资金。我们要感谢同事和匿名审稿人，它们提供了宝贵的反馈意见以帮助改进论文。

参考文献

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
[2] Q. Lin, P. Chang, G. Chen, B. C. Ooi, K. Tan, and Z. Wang, “Towards a non-2pc transaction management in distributed database systems,” in Proceedings ofACM International Conference on Management of Data (SIGMOD), San Francisco, CA, USA, 2016, pp. 1659 - 1674.
[3] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. J. Abadi, “Calvin: fast distributed transactions for partitioned database systems,” in Proceedings of ACM International Conference on Management of Data (SIGMOD), Scottsdale, AZ, USA, 2012, pp. 1 - 12.
[4] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica, “Coordination avoidance in database systems,” PVLDB, vol. 8, no. 3, pp. 185 - 196, 2014.
[5] “Ethereum blockchain app platform,” https://www.ethereum. org/.
[6] Ripple, “Ripple,” https://ripple.com. [7] Melonport, “Blockchain software for asset management,” http:// melonport.com.
[8] J. Morgan and O. Wyman, “Unlocking economic advantage with blockchain. a guide for asset managers.” 2016.
[9] G. S. Group, “Blockchain: putting theory into practice,” 2016.
[10] T. T. A. Dinh, J. Wang, G. Chen, L. Rui, K.-L. Tan, and B. C. Ooi, “ Blockbench: a benchmarking framework for analyzing private blockchains,” in SIGMOD, 2017.
[11] Ethcore, “Parity: next generation ethereum browser,” https:// ethcore.io/parity.html.

[12] Hyperledger, “Blockchain technologies for business,” https://www.hyperledger.org.
[13] F. Tschorsch and B. Scheuermann, “Bitcoin and beyond: A technical survey on decentralized digital currencies,” IEEE Communications Survey & Tutorials, vol. 18, no. 3, pp. 2084 – 2123, 2016.
[14] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “Sok: Research perspectives and challenges for bitcoin and crypto-currencies,” in 2015 IEEE Symposium on Security and Privacy. IEEE, 2015, pp. 104 – 121.
[15] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, “The end of an architectural era (it’s time for a complete rewrite),” in Proceedings ofthe 33rd International Conference on Very Large Data Bases (VLDB), Vienna, Austria, 2007, pp. 1150 – 1160.
[16] J. C. Corbett, J. Dean, and et al., “Spanner: Google’s globallydistributed database,” in Proceedings of 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Hollywood, CA, USA, 2012, pp. 261 – 264.
[17] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, 1999, pp. 173 – 186.
[18] Q. H. Vu, M. Lupu, and B. C. Ooi, Peer-to-Peer Computing Principles and Applications. Springer-Verlag, 2009.
[19] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in Proceedings of18th International Conference on Financial Cryptography and Data Security (FC), Christ Church, Barbados, 2014, pp. 436 – 454.
[20] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. G¨un, “On scaling decentralized blockchains,” in Proceedings of 3rd Workshop on Bitcoin and Blockchain Research, 2016.
[21] F. P. Junqueira, B. C. Reed, and M. Serafini, “Zab: Highperformance broadcast for primary-backup systems,” in Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Hong Kong, China, 2011, pp. 245 – 256.
[22] D. Ongaro and J. K. Ousterhout, “In search of an understandable consensus algorithm,” in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), Philadelphia, PA, USA, 2014, pp. 305 – 319.
[23] L. Lamport, “Paxos made simple, fast, and byzantine,” in Procedings of the 6th International Conference on Principles of Distributed Systems (OPODIS). Reims, France, 2002, pp. 7 – 9.
[24] “Monax: The ecosystem application platform,” https://monax.io.
[25] M. Vukolic, “The quest for scalable blockchain fabric: proof-

ofwork vs. bft replication," in Open Problems in Network Security iNetSec, 2015.
[26] L. Luu, V. Narayanan, C. Zhang, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in CCS, 2016.
[27] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in SOSP, 2017.
[28] "Proof of authority chains," https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains.
[29] "Stellar," https://www.stellar.org/. [30] "kadena," http://kadena.io/. [31] "Bigchaindb: a scalable blockchain database," https://github. com/bigchaindb/bigchaindb.
[32] "Hyperledger fabric v0.6.0," https://github.com/hyperledger/fabric/releases/tag/v0.6.0-preview.
[33] "Hyperledger fabric v1.0.0-rc1," https://github.com/hyperledger/fabric/releases/tag/v1.0.0-rc1.
[34] "Global decentralized currency," https://litecoin.org/. [35] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: decentralized anonymous payments from bitcoin," in IEEE Security & Privacy, 2014.
[36] "Multichain:open platform for blockchain applications," https:// www.multichain.com/.
[37] JPMorgan, "Enterprise-ready distributed ledger and smart contract platforms," https://github.com/jpmorganchase/quorum.
[38] "Hydrachain: Permissioned distributed ledger based on ethereum," https://github.com/HydraChain/hydrachain.
[39] "Openchain: Enterprise-ready blockchain," https://github.com/openchain.
[40] "Openchain: Next generation blockchain," https://iota.org/. [41] "Tendermint: Blockchain app development simplified," http://tendermint.com/.
[42] C. Copeland and H. Zhong, "Tangaroa: a byzantine fault tolerant raft," https://github.com/chrisnc/tangaroa.
[43] "Dfinity," https://dfinity.network/. [44] L. Goodman, "Tezos: A self-amending crypto-ledger position paper," 2014.
[45] R. Brown, "Introducing r3 cordatm: A distributed ledger designed for financial services," R3CEV blog, 2016. lake,"
[46] "Sawtooth sawtooth-core.
[47] "Decentralized autonomous organization," https://www.ethereum.org/dao.
[48] S. Eskandari, D. Barrera, E. Stobert, and J. Clark, "A first look at the usability of bitcoin key management," in Workshop on Usable Security, 2015.

[49] T. T. A. Dinh, P. Saxena, E. chien Chang, B. C. Ooi, and C. Zhang, “M2r: enabling stronger privacy in mapreduce computation,” in USENIX Security, 2015.
https://github.com/hyperledger/
[50] W. Zheng, A. Dave, J. Beekman, R. A. Popa, J. Gonzalez, and I. Stoica, “Oqaque: an oblivious and encrypted distributed analytics platform,” in NSDI, 2017.
[51] F. Zhang, E. Cecchetti, K. Croman, and E. Shi, “Town crier: an authenticated data feed for smart contracts,” in IEEE Security & Privacy, 2016.
[52] B. Smyth, M. Ryan, and L. Chen, “Direct anonymous attestation: ensuring privacy with corrupt administrator,” in European Workshop on Security and Privacy in Ad hoc and Sensor networks, 2007.
[53] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, “A fistful of bitcoins: characterizing payments among men with no names,” in IMC, 2013.
[54] “Chainalysis - blockchain analysis,” https://www.chainalysis.com.
[55] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoins: anonymous distributed e-cash from bitcoin,” in IEEE Security & Privacy, 2013.
[56] SCIPR Lab, “libsnark: a c++ library for zkSNARK proofs,” https://github.com/scipr-lab/libsnark.
[57] “Lightning network: scalable, instance bitcoin/blockchain transactions,” https://lightning.network/.
[58] S. Goldfeder, J. Bonneau, E. W. Felten, J. A. Kroll, and A. Narayanan, “Securing bitcoin wallets via threshold signatures,” in ACNS, 2016.
[59] V. Shoup, “Practical threshold signatures,” in EUROCRYPT, 2000.
[60] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” Journal of cryptology, 2004.
[61] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, “Enhancing bitcoin security and performance with strong consistency via collective signing,” in Proceedings of 25th USENIX Security Symposium (USENIX Security), Austin, TX, USA, 2016, pp. 279-296.
[62] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, “Keeping authorities honest or bust with decentralized witess cosigning,” in IEEE Security & Privacy, 2016.
[63] J. Behl, T. Distler, and R. Kapitza, “Scalable BFT for multi-cores: Actor-based decomposition and consensus-oriented parallelization,” in 10th Workshop on Hot Topics in System Dependability, HotDep ’14, Broomfield, CO, USA., 2014.

[64] M. Zbierski, “Parallel byzantine fault tolerance,” in Soft Computing in Computer and Information Science. Springer, 2015, pp. 321–333.
[65] W. Zhao, “Optimistic byzantine fault tolerance,” International Journal of Parallel, Emergent and Distributed Systems, vol. 31, no. 3, pp. 254–267, May 2016.
[66] D. Mazieres, “The stellar consensus protocol: A federated model for internet-level consensus,” Stellar Development Foundation, 2015.
[67] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, “Bitcoin-ng: A scalable blockchain protocol,” in NSDI, 2016.
[68] “Proof of stake,” https://github.com/ethereum/wiki/wiki/ Proof-of-Stake-FAQ.
[69] “Slimcoin a peer-to-peer crypto-currency with proof-of-burn,” www.slimcoin.club/whitepaper.pdf.
[70] “Peercoin - secure & sustainable cryptocoin,” https://peercoin.net.
[71] “Proof of elapsed time (poet),” https://intelledger.github.io/introduction.html#proof-of-elapsed-time-poet.
[72] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, “Proof of activity: Extending bitcoin’s proof of work via proof of stake,” IACR Cryptology ePrint Archive, vol. 2014, p. 452, 2014.
[73] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gazi, “Spacemint: A cryptocurrency based on proofs of space,” IACR Cryptology ePrint Archive, vol. 2015, p. 528, 2015.
[74] M. Milutinovic, W. He, H. Wu, and M. Kanwal, “Proof of luck: an efficient blockchain consensus protocol,” IACR Cryptology ePrint Archive, vol. 2017, p. 249, 2017.
[75] Y. Sompolinsky and A. Zohar, “Accelerating bitcoin’s transaction processing: fast money grows on trees, not chains,” Cryptology ePrint Archive, Report 2013/881, 2013, https://eprint.iacr.org/ 2013/881.pdf.
[76] S. Deetman, “Bitcoin could consume as much electricity as denmark by 2020,” https://tinyurl.com/yd8hq6n2, 2016.
[77] “Nxt: own your data, control your vote,” https://nxt.org/. [78] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse attacks on bitcoin’s peer-to-peer network,” in Proceedings of 24th USENIX Security Symposium (USENIX Security), Washington, D.C., USA, 2015, pp. 129–144.
[79] A. Clement, E. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, “Making byzantine fault tolerant systems tolerate byzantine failure,” in NSDI, 2009.
[80] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, “Zyzzyva: speculative byzantine fault tolerance,” in SOSP, 2007.
[81] S. Liu, P. Viotti, C. Cachin, V. Quema, and M. Vukolic, “Xft: Practical fault tolerance beyond crashes,” in OSDI, 2016.

[82] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, “The honeybadger of bft protocols,” in CCS, 2016.
[83] Intel, “Intel software guard extensions,” https://software.intel. com/en-us/sgx.
[84] T. Alves and D. Felton, “Trustzone: Integrated hardware and software security, enabling trusted computing in embedded systems,” White paper.
[85] B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, “Attested append-only memory: making adversaries stick to their word,” in Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), Stevenson, Washington, USA, 2007, pp. 189 – 204.
[86] J. Behl, T. Distler, and R. Kapitza, “Hybrids on steroids: Sgx-based high performance bft,” in Eurosys, 2017.
[87] “Interledger: the protocol for connecting blockchains.” https:// interledger.org/.
[88] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in CCS, 2016.
[89] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: the blockchain model of cryptography and privacy preserving smart contracts,” in CCS, 2016.
[90] “BlockBench: private blockchains benchmarking,” https://github. com/ooibc88/blockbench.
[91] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears,
“Benchmarking cloud serving systems with YCSB,” in Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC), Indianapolis, Indiana, USA, 2010, pp. 143 – 154.
[92] M. J. Cahill, U. R¨ohm, and A. D. Fekete, “Serializable isolation for snapshot databases,” in Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), Vancouver, BC, Canada, 2008, pp. 729 – 738.
[93] Crypti, “A decentralized application platform,” https://crypti.me. [94] A. Dinh, J. Wang, S. Wang, W.-N. Chin, Q. Lin, B. C. Ooi, P. Ruan, K.-L. Tan, Z. Xie, H. Zhang, and M. Zhang, “UStore: a distributed storage with rich semantics,” https://arxiv.org/pdf/1702.02799. pdf.
[95] K. Tan, Q. Cai, B. C. Ooi, W. Wong, C. Yao, and H. Zhang, “Inmemory databases: Challenges and opportunities from software and hardware perspectives,” SIGMOD Record, vol. 44, no. 2, pp. 35 – 40, 2015.
[96] H. Zhang, G. Chen, B. C. Ooi, K. Tan, and M. Zhang, “In-memory big data management and processing: A survey,” IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 7, pp. 1920 – 1948, 2015.
[97] A. Dragojevic, D. Narayanan, E. B. Nightingale, M. Renzelmann, A.

Shamis, A. Badam, and M. Castro, “No compromises: distributed transactions with consistency, availability, and performance,” in Proceedings of the 25th Symposium on Operating Systems Principles SOSP, Monterey, CA, USA, 2015, pp. 54–70.