Jidar: A Jigsaw-like Data Reduction Approach without Trust Assumptions for Bitcoin System

Xiaohai Dai, Jiang Xiao, Wenhui Yang, Chaofan Wang, and Hai Jin

摘要

针对痛点：比特币交易量的增加使得每个用户存储整个账本的完整副本变得不切实际。前人工作：已经有一些工作通过重新设计系统协议来减少存储开销。一个是运行轻节点，它信任全节点来存储其相关数据并及时回复其数据请求；但是，也带来了数据永久丢失的风险，因为轻节点需要的数据可能被全节点裁剪掉。另一种尝试是基于信任假设，将节点组织成多个协作单元，这在实践中很难得到满足。文中的创新性工作：Jidar 的每个节点只存储感兴趣的交易和来自完整区块的相关 Merkle 分支，就像从拼图中选择几块。实验结果：Jidar 可以在较低的通信成本下将一个普通节点的存储成本降低到原始比特币系统的 1.03% 左右。

关键词：比特币, 区块链, 布隆过滤器, 默克尔树

1 引言

截止 2019 年，比特币链上数据存储量已超过 210GB，没每年新生成的数据大约需要 50GB 的存储空间，一是不利于存储资源有限的普通用户腾出空间；二是由于数据存储的大小与区块链存储系统的性能（TPS）成正比，比特币的的单位交易量实在太低[1]。

由于一个比特币全节点必须在本地存储所有的交易，随着时间的推移，存储成本大大增加。由于现在比特币的系统性能太低，最多只有每秒 7 次交易（TPS）。表一反映了 TPS 和额外数据存储的相应成本之间的关系。TPS 越大，存储成本就越大。特别是，一旦比特币的 TPS 在不久的将来被提升到 2000，每年的额外存储成本大约为 15TB，这对普通用户来说太高了[2][3]。

文中提出的系统不依赖于任何信任假设，具体来说，每个普通节点只保留整个块的相关数据，就像从拼图游戏中选几块一样。系统基于如下三个理念设计：比特币上的许多用户只关心和自己相关的数据，对别人的交易不感兴趣因为没有报酬，用户没有义务存储其他人的数据，这可能导致数据保存者不负责任的数据裁剪。系统的优化不应该打破原有的性能，如去中心化、不可逆和 trust-free 的性质。

文章的贡献在于：对大存储成本问题进行了定性和定量的详细分析，并总结了相关工作。提出了设计数据缩减方法的三个目标在既不做任何信任假设，也不牺牲区块链的原始属性的条件下，阐述了一种旨在减少数据存储的方法实现了一个评估 Jidar 的原型，其实验结果证明了它的可能性和效率。

2 相关工作

目前已经有一些工作来缓解区块链系统的数据存储压力。根据这些工作开展得不同层次，可以分为两类：系统协议层和存储引擎层。

2.1 系统协议层

一个大的方向是基于各种信任假设，重新设计系统协议层。

第一个是 Nakamoto 在白皮书里提出的[4]，轻节点，只存储区块头，有数据请求则取找全节点. 这种方法有缺陷，一是轻节点必须信任全节点存储了相关数据并会回复其数据请求，这可能存在数据永

久丢失的风险。二是与一个用户相关的数据可能会被全节点清除。三是当多个轻节点从一个全节点请求块数据时，可能会出现网络瓶颈。

第二个是试图减少节点信任单元中的数据重复（例如 CUB）[5]，通过将节点组织成若干个信任单元，一个信任单元中的每个节点只需要存储整个区块链数据的一部分，其规模相对较小，然而，信任单元通常是基于信任单元中的所有节点必须相互信任的假设来组织的，这个假设太强，可能无法无法实现。

2.2 存储引擎层

另一类方案试图消除底层存储引擎层的数据重复，其代表作品包括 Forkbase[6]和 Ustore[7]。作为一个可分叉应用的通用存储引擎，Forkbase 旨在提高存储性能，同时减少存储规模。

在基于分块的重复数据删除的帮助下，Forkbase 成功地删除了具有大量共同内容的大文件的重复部分。然而，由于存储在节点中的交易和区块的重复率通常很低，存储引擎层面的重复数据删除在区块链的场景中做得不好。

3 本文方法

3.1 Jidar 设计要点

Jidar 中的每个节点只存储感兴趣的交易和相关的 Merkle 分支，从而大大降低了存储成本[8]。Jidar 不做任何信任假设，保留了区块链的特性。每个用户都将自己的相关数据存储在本地，不依赖他人，增强了用户保护自己数据的能力。同时，由于不要求节点存储其他节点的数据，因此，新交易的验证依赖于交易提供者提供的证明。Jidar 增加了将不同节点存储的所有碎片凝聚成一个完整块的机制，来满足块内数据完整的需求，就像将所有碎片拼成一个完整的拼图一样。Jidar 的方法的原型是基于 Golang 中的比特币来实现的。

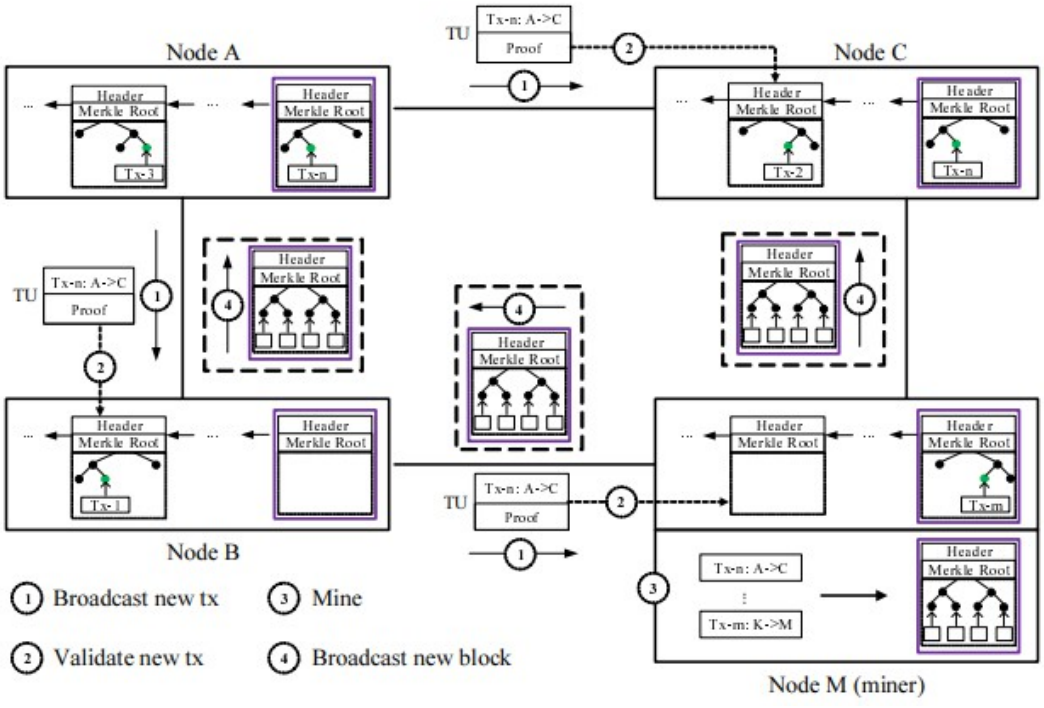

图 1: 设计概要概览

3.2 设计概要

为了简单起见，我们首先介绍一个草根设计，以减少交易层面的存储规模，其概况见图 1。在草人设计中，每个节点将存储所有的块头，而只存储每个块体的一小部分。以节点 A 为例，由于交易 Tx-3 与节点 A 有关（例如，节点 A 是 Tx-3 的接收者之一），该节点只存储 Tx-3 以及连接 Tx-3 和树根的 Merkle 分支。包括 Tx-0、Tx-1 和 Tx-2 在内的其他分支被它丢弃。

3.2.1 新交易的广播和验证

由于每个节点都没有存储完整的数据副本，新交易的验证依赖于交易提议者提供的证明。当一个用户提出一个新的交易时，它必须将证明与交易一起广播，如图 1 中的第一步所示。我们把一个交易加上相关的证明称为一个交易单元（TU）。图 2 描述了 TU 的细节，其中交易 Tx-n 将 Tx-3 的输出 2 作为输入。因此，TU 的证明应该包括 Tx-3，连接 Tx-3 和树根的 Merkle 分支，以及识别特定块的 BlockNumber。

一旦一个节点从网络上收到 TU，它将在进一步广播之前验证 TU 中新交易的可靠性。验证过程类似于简化支付验证（SPV）机制中的验证。具体来说，它验证了从叶子到根的 Merkle 分支，然后将 TU 中的根与块头中的树根进行比较。图 1 中的第二步描述了验证过程。由于证明可以由交易提议者自己提供，而不需要信任其他人。

3.2.2 挖掘和广播新区块

作为图 1 中的一个矿工，节点 M 在验证了 TU 后做的事情与原始比特币系统中的相同。它将根据从 TU 收集的交易创建一个候选区块，然后进行工作证明（PoW）计算，如图 1 中的第三步所示。

一旦新区块被开采出来，它将被节点 M 广播到整个网络，如图 1 中的第四步所示。每个节点在收到新区块后将存储相关的交易和 Merkle 分支，这只是总区块数据的一小部分。由于新区块包含节点 A 提出的交易，它存储 Tx-n 和包括 Tx-n 的 Merkle 分支。相比之下，节点 B 除了区块头之外什么都不存储，因为新开采的区块不包含与之相关的交易。节点从区块中挑出相关数据，与从拼图中分出几块的过程非常相似。为了得到通知，每笔交易将在整个网络中至少存储两次，因为一笔交易通常至少涉及一对发送者-接收者节点。通过只存储相关的数据，存储规模可以大大减少，此外，一个区块中的不同部分由不同的节点分别存储。

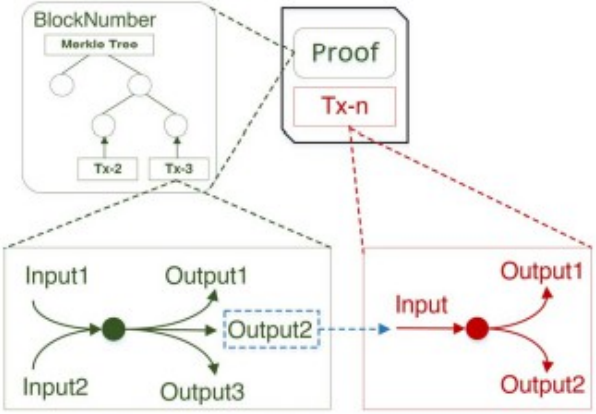

图 2: TU 的示意图

3.2.3 交易验证问题

虽然概要设计在减少存储成本方面似乎效果不错，但在交易验证过程中存在一个重要问题。比特币协议要求新交易的输入必须是未消耗的交易输出之一（UTXO）。Merkle 分支和相关交易只能保证输入是交易输出之一，但在证明输出之前没有被花费方面无能为力。如何证明交易产出之前没有花费？

3.3 布隆过滤器的设计

为了解决上述挑战，我们从 Bloom Filter[9]中吸取教训。我们首先详细介绍了在我们的设计中采用 bloom filter 的情况，然后介绍了与假阳性匹配有关的挑战。接下来阐述应对这一挑战的解决方案。

3.3.1 布隆过滤器抉择

布隆过滤器是一种牺牲空间以提高效率的方法，它被广泛用于测试一个元素是否包含在一个集合中。我们在区块头中增加了一个布隆过滤器的字段，表示新交易的输入是否被当作之前的输入使用过。如图 3 所示，我们采用了一个 16 位布隆过滤器和三个哈希函数。为简单起见，我们假设每个交易由一个输入和一个输出组成。通过计算 Tx-0 输入的三个哈希函数，布隆过滤器的第 9、14 和 12 位被设置。对 Tx-1、Tx-2 和 Tx3 也是这样做的。当检查一个新的交易的输入是否已经在这个区块中花费，三个哈希函数再次被计算。

如果得到的一个位在 bloom filter 中没有被设置，我们就可以确定新的输入没有在这个区块中花费。

作为一个例子，图 3 中的正常情况展示了这种情况。

因此，新交易的输入验证过程包括两部分：1）验证输入是否作为过去交易的输出而存在（即存在性验证）；2）验证输入在创建后是否被花费（即未花费性验证）。

以图 4 中的 Tx-n 为例，根据 TU 证明中的'BlockNumber' 字段，根据第 k 个块头进行存在验证。需要注意的是，在输出（即新交易的输入）产生后，可能会有很长一段时间，这可能会导致大量的验证工作无法进行。然而，这不会花费太多的时间，因为哈希函数的计算只需要一次完成，其结果可以储存起来供以后使用。此外，验证工作可以并行进行，以加快速度

上面忽略的一个最重要的问题是 Bloom 滤波器内在的假阳性匹配。以图 3 中" 挑战案例" 中的 Tx-n 为例，有三个比特的计算结果已经被 Tx-1、Tx-2 和 Tx-3 设置。

因此，Tx-n 的输入将被误认为已经花掉了，尽管它在这个区块中正好没有花掉。

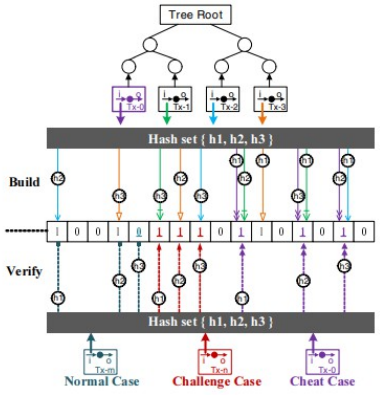

图 3: 块状头中采用布洛姆滤波器的示意图

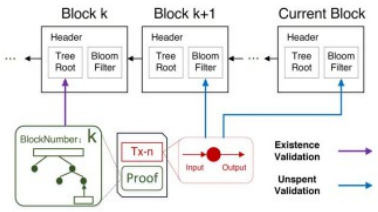

图 4: 交易验证过程的两个部分

3.3.2 处理错误结果的解决方案

有两个目标与处理 Bloom filter 的假阳性匹配有关：1）未花费的输出（即输入）不应该被误认为是花费的（即抗错性），这在上面已经阐述过了；2）花费的输出（即输入）不应该被伪造成未花费的（即抗骗性）。为了更清楚地描述防作弊的目标，我们以图 3 中的" 作弊案例" 为例。Tx-0 的哈希计算结果的三个比特分别由 Tx-1、Tx-2 和 Tx-3 设置。尽管 Tx-0 的输入在这个区块中被准确地花费了，但它可以谎称相应的位被其他交易设置，这很难直接检查。

该解决方案仍然依赖于新交易提出者提供的证明。众所周知，每个节点都保持着多个 UTXO（即 UTXO 集），用来组成新交易的输入。如果在 III-B1 节描述的未使用的验证过程中没有假阳性匹配，那么 TU 中的证明只需要包含 Merkle 分支和相关交易。但是，如果在一个区块中发生假阳性匹配，应该提供该区块的整个 Merkle 树和交易作为证明。通过提供整个数据，一个交易的输入可以被其他节点证明是未花费的。具体来说，该输入与区块中的所有输入逐一比较，这不仅提供了抗错性，也提供了抗骗性。

无论是默克尔分支还是整个默克尔树，作为证明提供的数据都应该由相关节点事先存储。因此，每次当一个节点收到一个新的区块时，它必须检查该区块中的已用输入和其 UTXO 集的输出之间是否存在假阳性匹配。

如果是真的，整个数据被存储。否则，数据的一部分被存储。应该注意的是，存储整个数据的情况很少发生，因为假阳性率被调整到了一个非常小的数值。

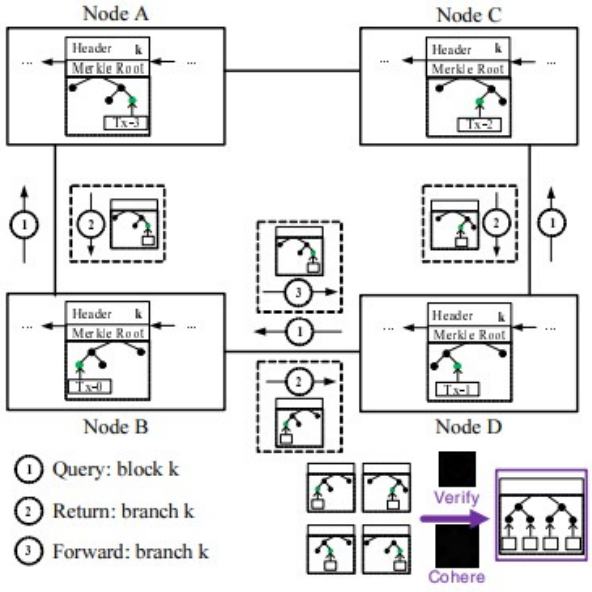

图 5: 查询全部数据的广播机制

4 Jidar 设计

到目前为止，上述设计的系统在大多数方面都能像比特币系统一样工作。然而，系统的稳健性和一些补充功能被忽略了。在这一节中，我们首先提出了对系统稳健性和功能完整性的要求。为了满足这些要求，我们提出了一些强化的解决方案。在本节的最后，我们对我们的方法（即 Jidar）做了一个总结。鉴于对系统稳健性和完整性的要求，如第 III-B2 节所述，每个节点一旦收到新的数据块，就必须分别找到可能的假阳性匹配。如果没有发现假阳性匹配，每个节点将在本地丢弃不相关的数据，以减少存储的大小。这就要求节点在新区块还在广播过程中的时候及时接收新区块。换句话说，一个节点很难在广播过程后从其邻居那里找到块的全部数据。然而，意外的网络断开或节点崩溃可能导致节点错过区块，这在我们的日常生活中很常见。如何处理意外的网络断开或节点崩溃的情况？作为我们对区块链系统的理解之一，绝大多数的用户只关心他们个人的相关数据。

而对别人的没有兴趣。因此，普通用户最常见的要求是查询个人数据。由于个人数据已被存储在本地，因此这种查询很容易和快速。然而，如果一个用户想在没有本地存储的情况下访问数据呢？例如，一个晚来的人想获得区块链上的所有数据，以做进一步的科学分析。

虽然这不是普通用户的一个非常普遍的要求，但查询全部数据应该被考虑在内，从而有助于系统的功能完整性。那如何在没有本地存储的情况下查询数据？

4.1 块状窗口

由于网络断开和节点崩溃都可以用相同的解决方案来解决，为了简单起见，我们将在下文中以节点崩溃为例。根据崩溃的持续时间，节点崩溃的问题可以分为两个子问题：持续时间小于时间 T 的短期崩溃和持续时间大于 T 的长期崩溃。我们做的一个合理的假设是，大部分的节点崩溃可以在时间 T 内修复，这意味着长期崩溃比短期崩溃要少得多。为了解决短期崩溃，每个节点被要求暂时存储最近的时间 T 内收到的块的完整数据，这构成了一个块窗口。

随着时间的推移，新收到的区块将被添加到区块窗口中，而最旧的区块将被丢弃。

在区块窗口机制的帮助下，从短期崩溃中恢复的节点可以很容易地从其邻居那里获得错过的区块，从而促进系统的稳健性[10]。

T 值的设置应该有所取舍。T 越大，系统就越健壮，同时需要更多的额外存储。

4.2 多重实例

虽然很少发生，但长期崩溃的问题应该被认真对待。解决这个问题的一个简单而可行的办法是由一个用户运行多个节点实例。如果一个实例崩溃了，其他实例仍然可以很好地工作以接收区块。显然，一个用户运行的实例越多，他/她的数据存储就越安全，同时需要更多的货币成本。正如 II-C 节所述，它依赖于用户来维护他们的相关数据并确保数据存储的安全。因此，由用户来决定如何部署多少个实例。对于拥有大量硬币的用户来说，通过云服务 [11] 的便利帮助或直接使用物理机器，在地理上分布多个实例是值得的和合理的。对于硬币少的用户来说，他们只需要利用他们的多种便携式电子设备，如手机、平板电脑和笔记本电脑。更重要的是，由于智能家居设备正变得越来越流行，它们可以作为节点的实例被连接到网络。

4.3 完整数据查询

到目前为止，用户存储的相关数据只包含以用户为发送方或接收方的交易（和相应的 Merkle 分支）。实际上，用户可以维护他/她感兴趣的任何交易，其唯一成本是额外的存储。如果用户想或计划对全部区块链数据进行分析，他/她可以从一开始就一直存储所有的数据。至于第 IV-A 节中的迟到者，他错过了过去的区块，在系统中加入了一个广播机制来查询全部数据。如图 5 所示，节点 D 想获得区块 k 的完整数据，首先，它向网络广播查询。每个节点在收到查询后，将其存储的片段返回给节点 D。在收到足够的片段后，节点 D 将进行验证，并将数据凝聚成一个完整的区块，就像将所有的碎片拼接成一个完整的拼图图片。

4.4 Jidar 设计总结

与原始的比特币系统相比，Jidar 所做的修改是节点存储和传输的数据结构。包括采矿、通信和共识，Jidar 与原始比特币保持一致。换句话说，Jidar 只从数据层重新设计了区块链架构，这就避免了很多安全问题，特别是共识协议的安全问题。

5 复现细节

5.1 与已有开源代码对比

没有开放源码

5.2 实验环境搭建

12th Gen Intel(R) Core(TM) i7-12700 2.10 GHz，32GB DRAM，Windows 10，用 Java 实现 bitcoin 原型（不开源，在此基础上做改进）

5.3 数据集与实验方法

数据集来自 2009 年 1 月 9 日—2018 年 12 月 31 日比特币 mainnet 上的所有交易；建立包含六个节点的链：分两类，1）ABCDE 为观察节点，存储和自己相关的交易 2）F 为辅助驱动节点，存储所有数据；

6 实验结果分析

可将存储成本降低到原始比特币系统的 1.03% 左右

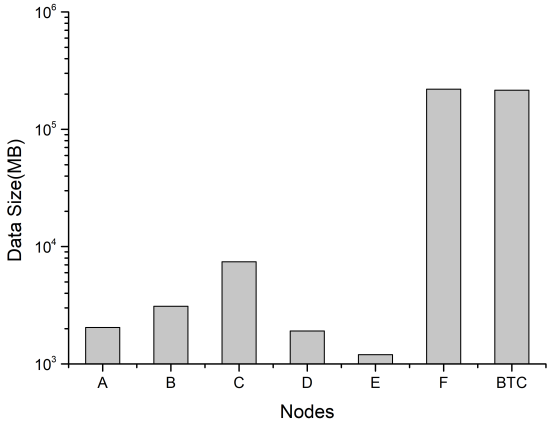

图 6: 与 BTC 的存储花费比较

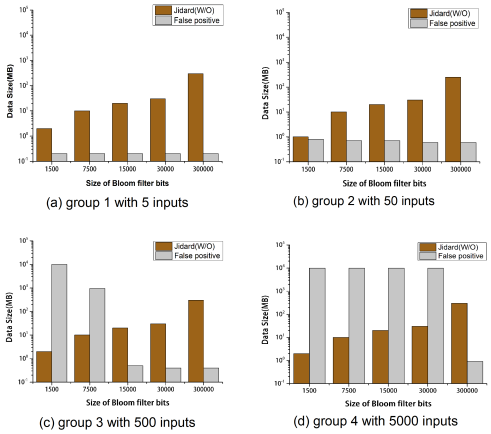

图 7: 不同输入规模下的假阳性率

布隆过滤器的位数设置：由于大多数区块的 UTXO 输入 <1000，因此根据经验，可将位数设置为 15000-30000

参考文献

[1] GRAY J, HELLAND P, O'NEIL P, et al. The dangers of replication and a solution[C]//Proceedings of the 1996 ACM SIGMOD international Conference on Management of Data. 1996: 173-182.

[2] EYAL I, GENCER A E, SIRER E G, et al. {Bitcoin-NG}: A scalable blockchain protocol[C]//13th USENIX symposium on networked systems design and implementation (NSDI 16). 2016: 45-59.

[3] GILAD Y, HEMO R, MICALI S, et al. Algorand: Scaling byzantine agreements for cryptocurrencies [C]//Proceedings of the 26th symposium on operating systems principles. 2017: 51-68.

[4] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.

[5] XU Z, HAN S, CHEN L. CUB, a consensus unit-based storage scheme for blockchain system[C]// 2018 IEEE 34th International Conference on Data Engineering (ICDE). 2018: 173-184.

[6] WANG S, DINH T T A, LIN Q, et al. Forkbase: An efficient storage engine for blockchain and forkable applications[J]. arXiv preprint arXiv:1802.04949, 2018.

[7] DINH A, WANG J, WANG S, et al. UStore: a distributed storage with rich semantics[J]. arXiv preprint arXiv:1702.02799, 2017.

[8] SAMANIEGO M, JAMSRANDORJ U, DETERS R. Blockchain as a Service for IoT[C]//2016 IEEE international conference on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData). 2016: 433-436.

[9] BLOOM B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.

[10] BUYYA R, YEO C S, VENUGOPAL S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation computer systems, 2009, 25(6): 599-616.