集群链：

基于集群的超高吞吐区块链系统

V 1.1 版本

ClusterChain.org

摘要

集群链（Cluster Chain），是一个基于集群的，具有高可扩展性、高伸缩性、高吞吐的区块链系统。依靠强大的 NodeMarket，集群链通过使用大量的外部 CPU 来解除系统本身的 CPU 计算瓶颈，从而带来性能方面质的提升，这是其他区块链所不具备的。它不需要大量的并行链就可以实现十万甚至百万 TPS。

集群链具备以下特征：

1. 单链超高吞吐量

   由于 CPU 的性能瓶颈，单节点处理能力有限，通常在 1000-3000TPS。业界一度认为多链并行是实现可扩展的最主要手段。为获取 10 万 TPS，需要几十甚至上百条链并行工作。跨链交易代价高昂，单链越多则跨链交易占比越高，最终会反过来抑制系统可扩展性。集群链通过集群的方式扩展基本计算单元，使得单链就可实现 10 万乃至百万 TPS。

2. 高可扩展性

   集群链的可扩展体现在两个层面：一个是集群层面，可通过动态增加节点数量来增强处理能力；另一个是多链并行，由于每个单链都有超高吞吐量，因此少量的并行链就可以满足海量需求。

3. 高可伸缩

   在系统繁忙时，可通过 NodeMarket 增加计算资源来加大处理能力。

4. 抗 DoS 攻击

   DoS 攻击几乎是所有区块链的难题。依靠强大的 NodeMarket，集群链在受到 DoS 攻击后，可大量使用廉价的短期节点来快速扩充集群最外层的接待节点（RN）数量从而应对攻击。

5. 低廉的运营成本

   运营成本是影响区块链扩展的重要因素。尽管从技术上讲，可以尽可能多地增加并行链数量来扩大吞吐，但每条单链是有成本的，不可能扩展到系统构建者都亏损的程度。集群链创新性地提出可信接待层（TRL），将耗时的计算转移至 NodeMarket 中的海量 CPU，在实现高性能的同时，可降低至少 80%的运营成本，从而使集群链在经济上的可扩展空间可以达到其他区块链的 25 倍以上。

6. 简单的账户管理

   尽管可能存在多条并行链，但用户只需创建一个账户，无需感知多条单链的存在。

7. 反向生态建立

   集群链吸引每个人将自己计算机以时间段的方式放入 NodeMarket 中，这将造就海量的参与群体，而这些持有代币的群体正是未来生态的潜在用户，将吸引应用构建于集群链之上。相对于其他区块链使用先建应用后吸引用户的方式，集群链这种生态反向建立模式具有明显的优势。

1. NodeMarket

世界上每时每刻都有数以千万计的计算机处于可用状态。我们构建了 NodeMarket，使得这些计算机可以以时间段的方式出售自己的算力。每个计算机会标注自己的硬件信息，包括 CPU，内存，磁盘，带宽等，同时它还将附上自己可用的时间段。退出时间点可以修改，但必须保证足够的缓冲时间以便使用者能够做出调整。

集群，就是通过网络互连的一群计算机，通过相互间的协作来共同完成任务。集群是动态的，节点可以加入，也可以随时因为各种原因而退出。结合 NodeMarket，我们通过引入抽象节点（Abstract Node）的概念，为集群构建了一个高稳定性模型。

抽象节点是由多个实际节点接力而成，如图 1 所示，当一个实际节点快要到达使用终点时，可以从 NodeMarket 中选择一个符合要求的新节点，该节点加入后立即与将要退出的实际节点进行状态交接。应当确保有足够的时间进行交接，并且引入新的实际节点前需要进行测试以验证其声明的硬件条件。与抽象节点对应的是常驻节点，该类型节点应该较为固定地运行于集群内。

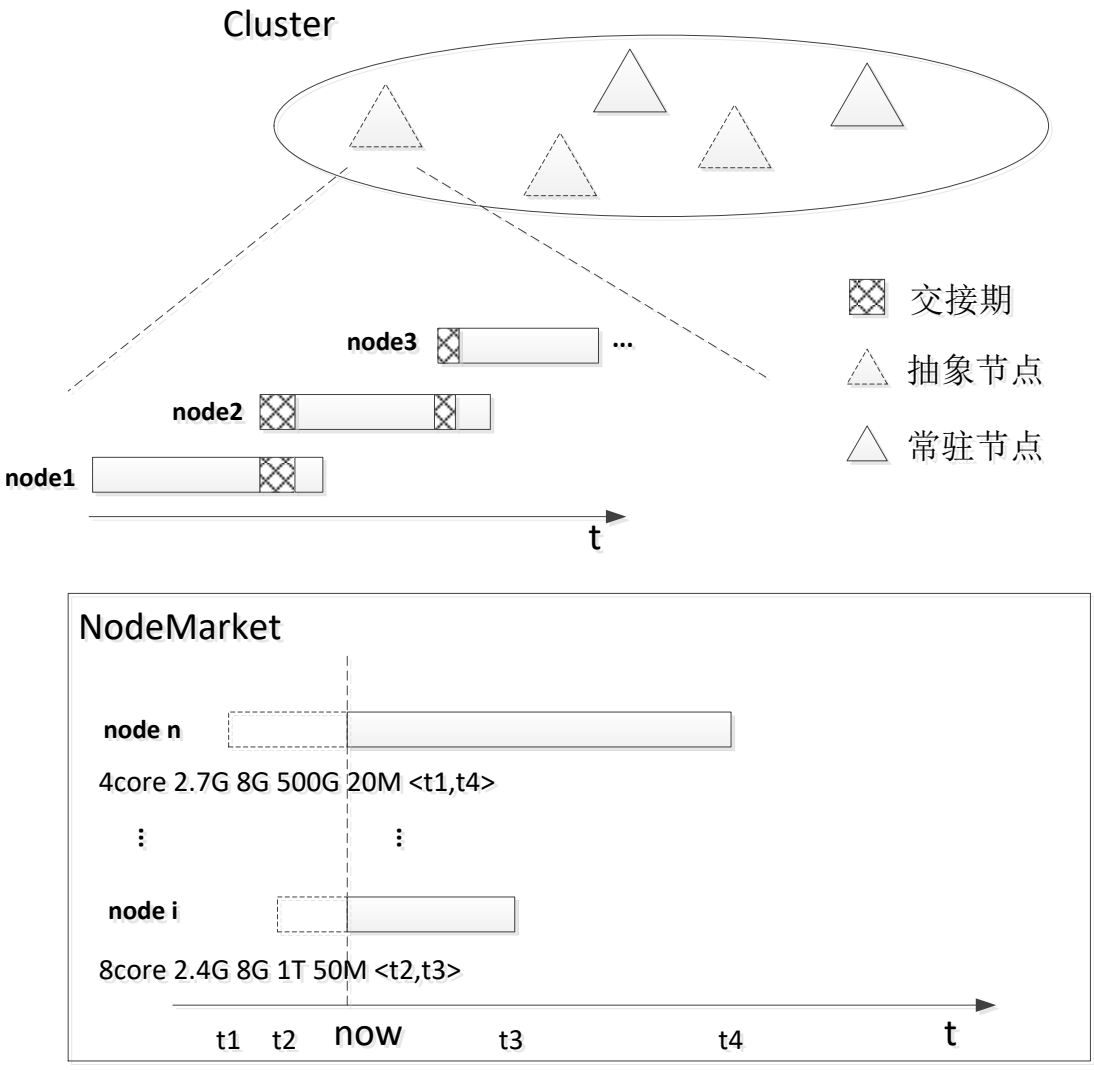

图 1　Node Market 与抽象节点

实际节点以时间段的方式将自己放入到 NodeMarket 中，这些节点的数量非常庞大，这为集群链高可扩展性和超高 TPS 奠定了资源基础，同时海量的参与者获得了代币，他们将成为未来集群链生态的潜在用户。

集群维护抽象节点与实际节点之间的映射，以便对实际节点进行报酬分配。实际节点可以和集群之间进行相互评价，该评价以交易的形式保存在区块链上。这些评价作为信誉值的重要组成部分，也构成实际节点与集群之间相互选择的一个重要因素。

即使 NodeMarket 有 500 万节点，在内存中也只占用几百兆字节，因此每个集群都可以保存有一份 NodeMarket。可以使用 Gossip 协议来同步不同集群的 NodeMarket。

2. 集群架构

如图 2 所示，集群由两部分组成，第一部分是可信接待层，它把接收到的交易流转化为受信任的消息流，并为逻辑处理层分担代价较高的计算；另一部分是逻辑处理层，它接收可信接待层传过来的消息流，快速地处理交易，并生成或验证区块。

逻辑处理层由常驻节点组成，这些常驻节点通过万兆以太网连接成高速局域网，之间的通讯代价可以忽略。这些常驻节点又被称为逻辑节点（Logic Node，LN）。LN 的 CPU 不再是瓶颈，在带宽充足的情况下，即使 1 台 LN 也可以实现很高吞吐。

可信接待层可由抽象节点构成，这些节点被称为接待节点（Reception Node，RN）。RN 的状态迁移较为容易，可由 NodeMarket 中的节点接力而成。集群可根据需要通过 NodeMarket 快速增加 RN 数量。RN 同时还负责将没有问题的交易转发给其他集群。

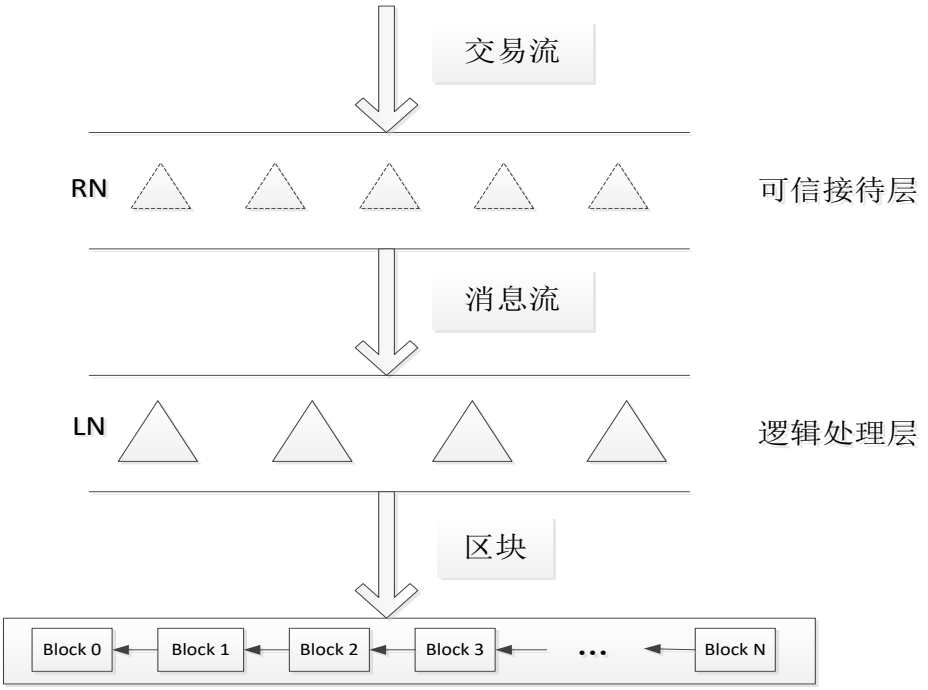

图 2 集群架构

2.1 可信接待层（Trusted Reception Layer,TRL）

CPU 是区块链系统最大的性能瓶颈所在。通过将签名验证等代价较高的计算转移给可信接待层的 RN 节点，将 LN 的 CPU 从耗时的计算中解放出来，从而使系统吞吐量获得质的提升。

NodeMarket 中的每个节点都会有一个信誉值，集群在从中选择 RN 时会将该值作为重要参考，因此被一个集群选中的 RN 是非诚实节点的概率不会很高，我们设其上限为 10%。如果一笔交易通过了 7 个随机选取的 RN 的验证，那么它为非法交易或被篡改的可能性仅为千万分之一，如果想要这笔非法交易最终写入区块链，那么需要超 2/3 的集群确认，这意味需要超过数十个甚至上百个 RN 串通，从概率上讲基本是不可能的。因此处理该交易的 LN 可以放心地省略掉对该交易签名验证等高代价计算。

可信接待层的 RN 数量记做#RN，对每个 RN 从 1 到#RN 进行编号。当一笔交易到达RN_i后，执行以下步骤：

1）RN_i选择该交易签名的前 20 位作为种子产生一个在[1,#RN]范围内的随机数 k，然后将这笔交易转发给RN_k；

2）RN_k将自己验证通过的交易放入缓冲池，每过 0.3 秒，将缓冲池中的交易打包，计算出该交易包的哈希值，将该哈希值作为种子，产生 6 个在[1,#RN]范围内的不同随机数（不包括k），然后将交易包发送给对应的 6 个 RN；

3）收到交易包的 RN 对交易包验证，对处理结果与该包的哈希值进行签名，然后连同处理结果发送给RN_k；

4）RN_k收到所有反馈后，将交易包及各个节点的处理结果（包含签名），以及自己的处理结果作为整体，进行签名后发送给一个 LN，记为LN_j；

5）LN_j收到RN_k传送过来的消息包之后，做如下检查：

a)验证RN_k消息包的签名，不通过则丢弃此包；

b)根据交易包的哈希值产生 6 个在[1,#RN]范围内的不同随机数，看是否与消息包中的一致，不一致则视RN_k为不诚实节点；

c)验证另外 6 个 RN 的签名是否正确，不正确则舍弃该包；

d)验证 7 个 RN 对该交易包的处理结果是否一致，如发生不一致则自己验证该交易包，识别出可能不诚实的 RN，通告其他 LN，并协商将不诚实的 RN 剔除，并用交易记录下来；

6）检查通过的交易包被视为可信消息包进行后续处理，无需再做签名验证等已经被 RN 做过的计算。

随机产生 7 个 RN 是为了防止有不诚实的 RN 相互串通。通过RN_k搜集反馈结果，不会增加 LN 的带宽消耗。繁重的 CPU 计算从 LN 上转移到可信任的 RN 集合上。通过对交易流进行批处理，大大降低了 LN 上签名验证的计算次数。如果交易稀少，交易可直接由 LN 进行验证。

一些全内存操作的交易平台如 LMAX，其单线程能够每秒处理 600 万订单。在理想情况下，只有一个 LN 的集群也可以达到数百万 TPS，这需要有足够的 RN，足够的带宽，足够的内存。在实际情况下，包含一个 LN 的集群可实现数万到数十万的 TPS，这比单节点区块链的 1000-3000TPS 提高 10-100 倍以上。这是由于从 NodeMarket 中使用了大量廉价的 CPU 资源。可以通过优化，比如若干集群共享某个集群的 TRL 层来降低所需的 RN 数量。

RN 的处理过程可以看成是一种挖矿，挖矿的行为将交易流转化成可信任消息流。7 个 RN 组团挖矿，当它们正确处理了一笔交易包后，可以认为它挖到了一定数量的代币。可以根据处理的交易数量进行代币分配。

2.2 低运营成本

尽管从技术上讲，很多区块链可以通过多链并行来不断扩展 TPS，但这不能无限制地进行下去，因为每条链都有成本，当链扩展到一定数目之后，网络建设者将面临亏损，也就无法再继续增大 TPS。

集群链通过 TRL 大大降低了运营成本。例如，51 个集群，每个集群 2 台 LN，实现 10 万 TPS，每 4 个集群共享同样的 RN 集合，每个 RN 每秒 3000 次签名验证，那么大概需要 3000 个 RN 及 102 个 LN；而一个有 51 节点的单链，假设每条链 3000TPS，需 40 个单链才能达到 10 万 TPS，对应约 2040 台机器。后者的机器要一直运营维护，而前者的 RN 可以是廉价的短期节点，假设它们的价格比为 1:3，那么两者成本比为 1102:2040.集群中的 RN 数量是可伸缩的，假设平均下来集群单链的利用率为 30%，则两者的成本比为 330:2040，即 0.16:1。因此采用 TRL 这种模式可以将网络运营成本降低 80%以上。或者说，同样的成本，其他区块链实现 10 万 TPS，集群链可实现 50 万 TPS，如果链的价值与 TPS 成正比，那么集群链在经济上的可扩展空间是其他区块链的 25 倍以上。

2.3 抗 DoS 攻击

DoS 攻击是几乎所有区块链难以解决的问题。由于有巨大的 NodeMarket 支撑，当一个集群受到 DoS 攻击时，它可以立即增大#RN，进而从 NodeMarket 中获得更多的 CPU 和带宽来应对攻击。#RN 甚至可以扩大到远远超出 LN 的处理能力上限。RN 构成了集群的第一层防护。

3. 集群单链

集群单链由主力集群和替补集群构成。主力集群轮流出块，替补集群作为主力集群的后备，也做块的验证工作。当一个主力集群出现问题后，可以选择替补集群中的一个优秀者加入到主力集群中。实现中选取 34 个主力集群和 17 个替补集群，每 3 秒出一个块。一个集群的逻辑处理层可以由一台高性能高带宽的计算机组成，也可由若干台通过万兆以太网连接的普通计算机构成。

3.1 共识机制

集群单链使用 PoP（Proof of Performance，PoP）和 DPoS 混合的共识机制。

起初主力集群和替补集群都是由权益持有者通过投票产生。接下来的过程中，对一个集群的评价不再完全依靠投票，还要看该集群在过去一段时间的表现。PoP 的目的是选取最优秀的集群保留在主力集群中。投票可增加灵活性，通过权益持有者的投票可以间接地将更多因素融入到对一个集群的考核中。因此，使用 PoP 混合 DPoS 的机制能够公正高效地选出负责区块生成的集群。

集群单链并不是区块产生之后广播该区块到验证集群，而是使用反向验证的方式使得验证集群与出块集群近乎同步工作，因此，当区块生产完成之时，其它集群的验证工作也接近尾声，因此出块集群和接下来要出块的若干集群可以较为迅速地得到从其他集群传过来的验证结果，当超过 2/3 的集群验证通过后，该块将被加入到区块链中。

连续数次验证失败的集群将被淘汰进入替补集群，替补集群表现良好（包含最近的验证表现）的一个集群将被选入。如果出块集群出块失败或所出的块未通过其他集群的验证，那么该区块将被跳过，由下一个集群接着出块，出块失败集群将被放入替补集群。因此可以说，区块链出现分支会被控制在很小范围内，并且是短暂的。

问题集群会被及时放入替补集群中，这可导致替补集群中问题集群占比升高。可以采取如下措施：主力集群对区块达成共识的同时，替补集群也独立地形成共识；如果两边的结果不一致，则将此事件广播，由社区通过投票将问题集群剔除。

4. 集群链结构

4.1 集群链的三个维度

在比特币、以太坊为代表的区块链中，每个节点进行同样的任务，就像是一个连接着多个点的直线，可以认为这些区块链工作在一维空间里。

随着越来越多的项目（例如，[4][5][7][9]）把分片或者多链并行作为提高吞吐量和可扩展性的手段，区块链被拓展到二维空间。所有的节点不再构成一条线，而是由很多平行的线（单链）构成的平面，如图 3a 所示。不同于并行单链对节点进行纵向切割，集群单链使用的是横向切割，如图 3b 所示。链数目越多，跨链交易的比例就越高。而跨链交易代价高昂，过多的并行链反而会抑制可扩展性。

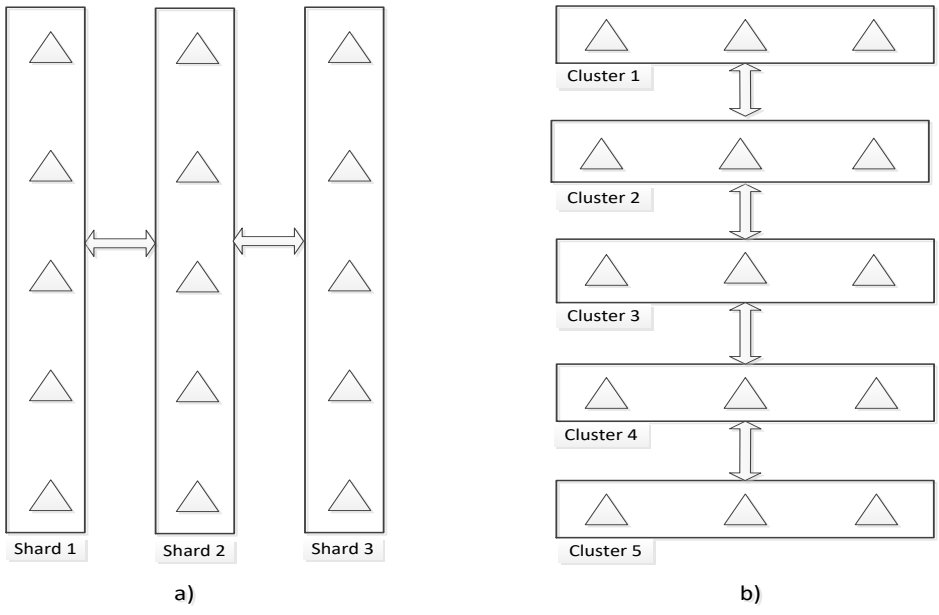

图 3　将区块链拓展至二维空间 a）分片或并行链（纵向）； b）集群单链（横向）

启用多条单链将使集群链运行在三维空间，如图 4 所示。第一维是集群的规模，代表着可扩展性，集群中的节点数目越多，带来的吞吐量也就越大；第二维是单链中的集群数量，数目越多代表去中心化程度和安全性越高；第三维是单链的数量，代表着去中心化和可扩展性。

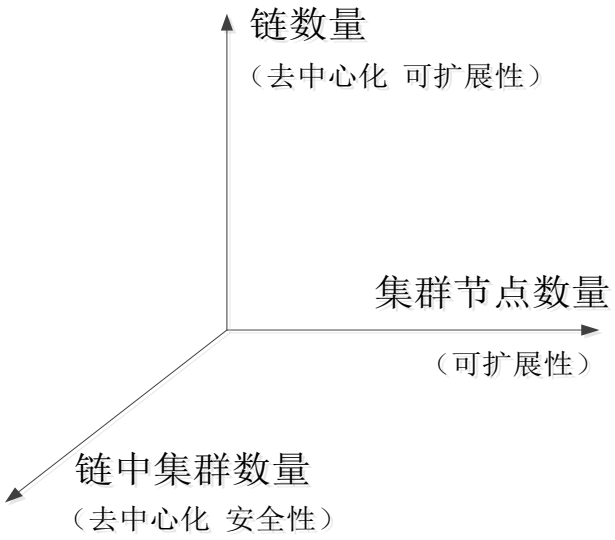

图 4　集群链的三个维度

4.2 命名空间

每个集群单链都构成一个命名空间，我们用$\mathrm{NS_i}$表示第 i 个被创建的单链的命名空间。一个账户 A 在一条单链上可以表示为 $\mathrm{NS_i}(A)$，它在该链上的余额可表示为$\beta(\mathrm{NS_i(A)})$，账户 A 的总余额为 $\sum_{i=1}^{n}\beta(\mathrm{NS_i(A)})$，n 为单链数量。一笔跨链转账可以表示为 $\mathrm{TX_h}: NS_i(A) \overset{a}{\rightarrow} NS_j(B)$。当钱包需要将一笔金额 a从账户 A 转移到账户 B 时，它会首先找到 $\beta\big(\mathrm{NS_i(A)}\big) > a$的链，然后执行一笔链内转账。如果没有满足条件的链，那么钱包将寻找一个满足余额之和大于a的链集合，然后将相应的金额从这些链转移到目标链的账户。这与 OmniLedger[4]中的跨片交易很类似，OmniLedger 针对 UTXO 模型进行分片扩展，跨片交易使用 Atomix 协议来保证交易原子性。集群链也可以采用类似的协议实现跨链交易：对于 client 在执行过程中可能出故障的情况，可以随机选择一定数量的节点作为 client 的备份节点，只要其中一个能够完成 Unlock 操作即可。跨链交易代价较大，应该尽可能地减少使用。如果一个账户需要频繁向某个命名空间中的账户转账，那么它可以提前转移足够的资金到自己在该空间的账户，这样绝大多数的转账都变成链内交易。由于集群单链的性能可达 10 万甚至百万 TPS，因此大多数应用都会运行在一个链中，跨链交易的比例并不会很高。

5. 经济模型

集群链上发行的代币被称为 Cluster Chain Token，简称为 CCTK，用于在整个生态中流通。CCTK 初始量为 100 亿枚，每年增发 3%，用与奖励网络的建设者。

具体分配采取以下方式：

1）每笔交易的费用由接待并成功验证该笔交易的 7 个 RN 获得；

2）增发的代币，按照每条单链的处理容量以及表现进行分配；对每条单链的配额又可以分为两个部分：

a) 一部分用于奖励集群（不包括 RN），主力集群分配的要多些；

b) 另一部分用作对 RN 的补贴；

分配的基本原则是保证网络的每个建设者都得到应有的回报，并且尽可能地降低交易费用。

为了更好的激励计算资源参与到 NodeMarket 中，增加它们的积极性和黏性，对每个进入 NodeMarket 的节点都有一个信誉值，它是节点表现的一个量化值，与节点收益挂钩。如果一个节点在工作时间段内突然退出或者网络不稳定，那么它的信誉值就会被调低；如果节点被检测到作弊，则信誉值会被清空；如果节点在 NodeMarket 中长时间表现优异，那么它的信誉值会增加，以鼓励节点长期参与；如果节点很长时间不回来，则逐渐调低其信誉值。为了鼓励新节点加入，新加入的节点会给予一个平均信誉值，在接下来的一段试用期内，如果出现舞弊或不稳定情况，则大幅调低其信誉值甚至清空。

与分配模型相关的各个参数可由社区投票决定。

6. 技术实现

集群采用异步消息模式，可充分地利用 CPU 资源。不同的集群可根据自身情况采用不同的容错等级。通过将整个全局状态切分，不仅提高集群处理的并发度，还可实现集群内部负载均衡，最大化处理能力。

6.1 集群 leader

每个集群都会分配一个唯一的 ClusterID（CID），并且每个集群都会有一个 leader 代表本集群与其他集群交互。当 leader 出现问题后，可从其他的常驻 LN 中选出一个新 leader，并向其他集群广播。

6.2 超级分布式文件系统 SDFS（Super Distributed File System）

IPFS[6]根据文件内容进行寻址，这一方式适用于去中心化环境下存储不可变文件。我们构建基于集群的超级分布式文件系统 SDFS，来存储区块链以及系统状态快照等数据。

图 5 展示了 SDFS 的基本原理。每个文件在存储时都可以指定副本数量。每个文件根据文件内容生成一个哈希值，这个哈希值与集群的 CID 属于同一空间。利用集群单链的分布式哈希表（DHT，Distributed Hash Table）可以根据文件哈希值（即文件名）找到对应的存储集群（包括副本所在的集群），这样就可以将存储请求发送给这些集群的 leader。集群 leader 收到请求后，根据本集群的 DHT 查找到负责存储该文件的一个常驻节点，由该节点与客户端完成存储工作。

对于高吞吐量的区块链来说，存储的增长是相当快速的。相对于每个集群都要存储一份区块链数据，使用 SDFS 可以使整个集群链的存储效率大幅提高。例如，51 个集群，对每个文件采用 8 副本，那么每个区块的存储消耗缩减了 85%。同一时间储存这个区块的 8 个节点全部出故障的几率是极低的，只要有一小部分副本还在，系统将迅速地恢复到原来的数量。

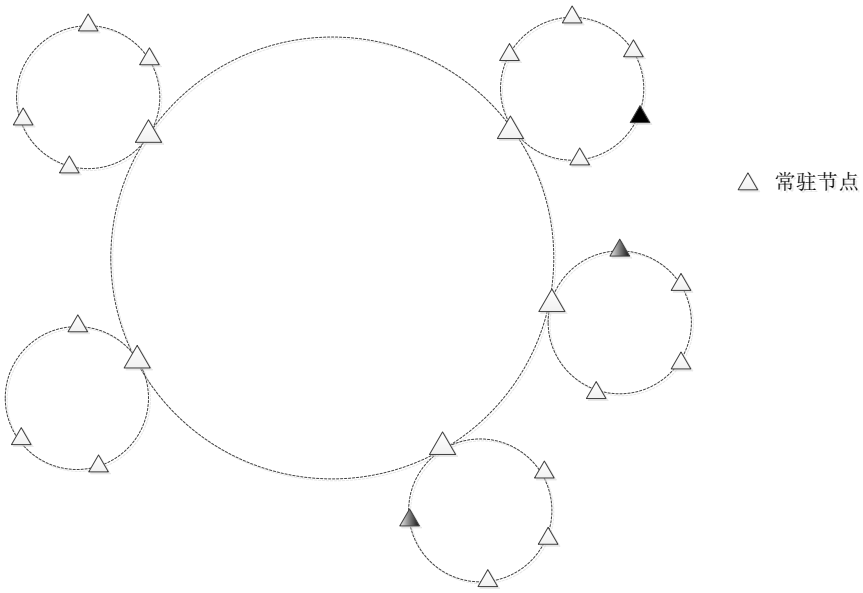

图 5 一个三备份文件在 SDFS 上存储示意图

6.3 账户（Account）及全局状态（Global State）

同以太坊类似，集群链使用账户模型。账户分为两种类型：普通账户和合约账户。每个普通账户都拥有一个私钥和一个公钥，其地址由公钥衍生而来，取公钥的最后 20 个字节。合约账户由外部账户或另一个合约账户创建，其地址由创建者的地址和创建者总共创建的交易数量（即 nonce）决定。

每个账户（无论普通账户还是合约账户）都有一个账户状态，包含以下元素：

1．nonce（64 位）：初始值为 0，记录发出的交易的数量。

2．balance（128 位）：非负，记录账户的余额。

3．code_hash（256 位）：对于合约账户，记录合约代码的哈希值；对于普通账户，记录空字符串的哈希值。

4．storage_root（256 位）：账户存储内容的状态。例如，可以将内容表达为 key-value 对，这些 KV 对储存在一颗 Merkle Patricia Tree 中，那么该树的根节点的哈希值可以代表整体状态。

整个系统拥有一个全局状态（Global State），这个全局状态由所有账户和它们各自的状态组成。根据每个账户的地址，全局状态被组织成一颗 Merkle Patricia Tree（MPT）。根据账户地址的前 10 位，将这些账户分成 1024 份，即 1024 个子树，每个子树被称为基本子树（Basic SubTree，BST），它是 LN 处理全局状态的基本单位，集群中的每个 LN 可以同时处理多个 BST。每个 BST 都可使用一个数据库来存储状态信息，也可以完全放于内存之中。

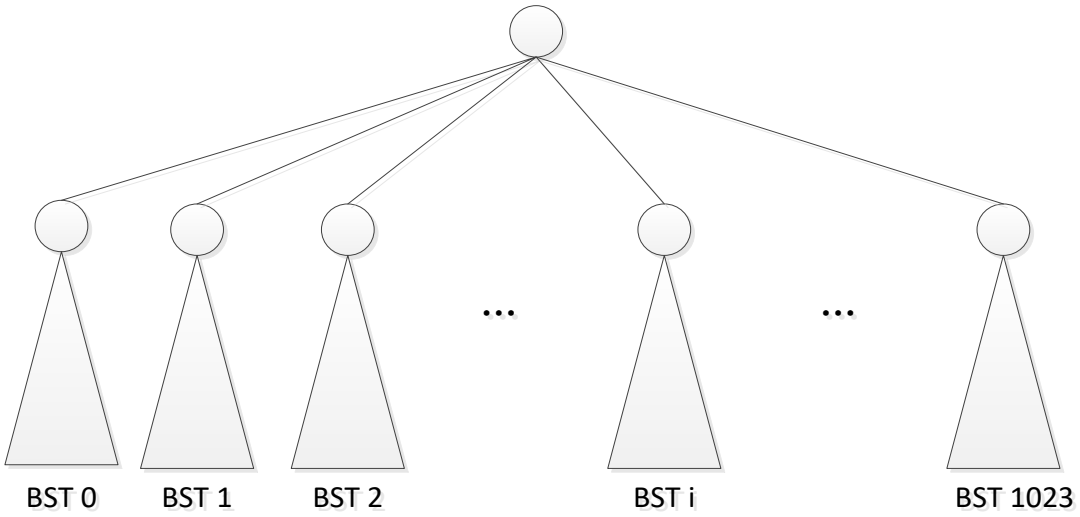

图 6 全局状态及 BST

6.4 交易（Transaction）

每一笔交易都是由账户发起，其执行会改变全局状态。交易包含的主要元素如下：

1．nonce（64 位）：等同于发起这个交易的账户的 nonce。

2．to（160 位）：目标账户地址。创建一个合约账户时无需指定。

3．amount（128 位）：需要转移的代币数量

4．gas_limit（128 位）：转账交易无需设置，为恒定值。调用智能合约的交易需要设置，以防止合约无限执行下去。

5．code：创建合约账户时合约的字节码。

6．data：调用智能合约时所应包含的信息。

7．pubkey（264 位）：用来验证签名的公钥。该公钥也决定了交易发送者的地址。

8．signature（512 位）：交易的签名，用于验证交易是否来自发送者。

每个交易由交易 id 标识，该 id 是交易（去除 signature）的哈希值。

6.5 区块（Block）

区块链由区块连接而成，每个区块都包含前一区块的哈希值，这样修改之前任一区块都将导致后面所有区块的变化。区块由区块头和区块体组成，区块头包含的主要元素如下：

1．parent_hash（256 位）：父块的哈希值。
2．height（256 位）：块的高度。创世块高度为 0。
3．timestamp（64 位）：块创建的时间。
4．coinbase：一个数组，记录集群成员的账户及对应的报酬
5．state_root：区块中交易全部处理完后的全局状态，即全局状态 MPT 的根哈希值。
6．transaction_root：该区块所有交易的 Merkle 树的根哈希
7．tx_summary_root：该区块所有交易的处理概要的 Merkle 树的根哈希
8．body：区块体在 SDFS 上的文件名

区块体包含了处理过的交易以及交易的处理概要，如交易费用，是否成功等。

6.6 区块验证

单节点区块链验证区块时依次执行块中的交易。对于集群链来说，全局状态在集群中是分布式存在的，这给交易处理带来了很大的变化。

用$\beta(A)$表示账户A的余额，用 $TX_h: A \xrightarrow{n} B$ 表示一笔转账交易，其中h是交易TX（不包括签名）的哈希值，该交易将数量为n的代币从账户 A 转移到账户 B。如果 A 和 B 位于同一 BST 下，那么该笔交易处理起来就比较简单，以下只考虑交易的双方位于不同 BST 的情况。我们可以将TX_h分解成Withdraw和Deposit两个操作，分别表示为W_h和D_h，TX_h在分布式环境下的执行被转化成W_h和D_h操作对，TX_h的原子性要求这个操作对要么全都执行，要么全不执行，否则就会破坏数据的一致性。

假设当前$\beta(A) = 50, \beta(B) = 30, \beta(C) = 10$，有两笔交易：

$TX_{h1}: A \xrightarrow{20} B, \quad TX_{h2}: B \xrightarrow{40} C$

如果在单节点环境顺序执行的话，两笔交易都能够成功，最后三个账户的状态分别为$\beta(A) = 30, \beta(B) = 10, \beta(C) = 50$。但在分布式环境下，即使$TX_{h1}$先于$TX_{h2}$到达集群，也面临两种执行结果：一种如图 7a 所示，账户A执行$W_{h1}(20)$操作后，账户B首先收到了$D_{h1}(20)$，执行后$\beta(B) = 50$，这样$W_{h2}(40)$就可以在账户B上执行，最后三个账户的余额为$\beta(A) = 30, \beta(B) = 10, \beta(C) = 50$；另一种情况如图 7b 所示，$W_{h2}(40)$先到达账户 B，而此时$\beta(B) = 30$，扣款操作无法执行，因此$TX_{h2}$被拒绝，最后三个账户的余额为$\beta(A) = 30, \beta(B) = 50, \beta(C) = 10$。

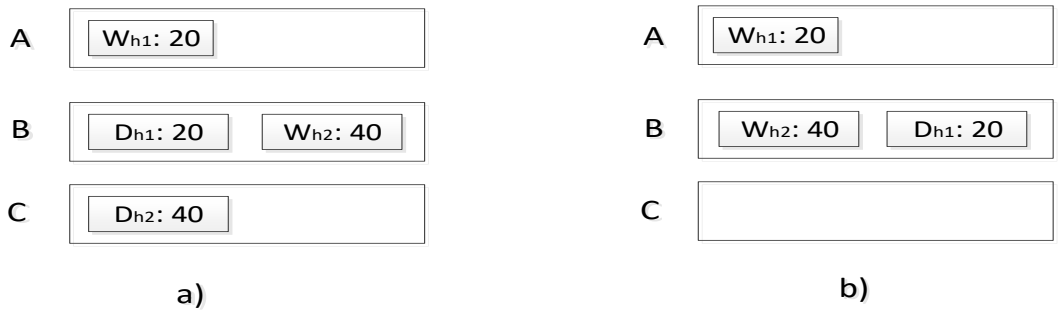

图 7　分布式环境下交易执行的不同结果

执行出现的两种结果都是合理的，它反映了在分布式环境下时序的不确定性。单节点区块链中的验证节点只要按照块中的交易依次进行验证即可，这种方式在集群链中就不适合了，将同样的交易序列传给不同的集群，最终达到的账户全局状态可能会各不相同。

我们使用反向验证（BV，Backward Verification）的方式来解决这一问题。BV 确保了全局状态的最终一致性。

类似于 Actor 模型中的消息传递机制，一个账户向另一个账户发送的消息，会按发送的先后顺序依次到达。每个账户在处理完一个 W 或 D 操作后，除了在本地用日志记录下来，还将其广播至其他集群。每个操作都会带有一个序列号，这保证了不同集群的相同账户以相同的次序处理这些操作。每个集群收到这个操作流后便开始反推交易。如果账户 A 接收到W_h，那么会先执行它：如果W_h被拒绝，那么对应的TX_h执行结果就为 Reject；如果W_h执行成功，那么 A 就会发送一条消息给接收账户 B，告知 B 扣款已经成功，如果 B 收到对应的D_h就意味交易TX_h交易成功（对应的 A 发来的消息将被清除）。如果 A 接收到D'_h，那么它会等待，直到收到一条来自扣款账户的扣款成功消息。

如果满足以下条件就代表区块验证通过：

1．所有的 W 和 D 操作都被执行

2．所有账户的消息队列都被清空

3．反推出来的交易集合与生成块中的交易集合相同，并且每个交易的处理概要也一致

4．全局账户状态达到一致

对于 3 和 4，当验证集群产生出区块头后，只需依次和传过来的区块头比较 state_root、transaction_root、tx_summary_root 的值即可。

对于一个交易 $TX_h: A \xrightarrow{n} B$，在反向验证的过程中可能会出现以下情况：

a）账户 A 的W_h被执行，但账户 B 的D_h未被执行，即 B 的消息队列始终有一条来自 A 的消息未被清空。这表明 A 的扣款未能转到 B 的账户上。原因可能是处理 B 账户的节点出现故障而出块集群未能识别（区块交易集中也应该不出现TX_h）。

b） 账户 A 的W_h未被执行，但账户 B 的D_h被执行，即 B 一直阻塞在D_h上等待来自 A 的消息。这表明 A 没扣款就将款转移到 B 上。因为 A 会在广播扣款操作之后再发送存款操作给 B，所以这种情况不应该发生。如果确实发生，意味着处理 A 的节点可能受到了攻击。

c） W_h或D_h中的金额与TX_h中的金额不一致。如果发生，则表明金额与TX_h不符一方的节点受到了攻击。

对智能合约的处理也类似，涉及到的合约调用，以及期间产生的转账交易，除了本地记录进日志以外，还将它们广播到其他集群以便进行同步验证。

为了确保传输安全，节点可以缓冲操作流，每隔 0.1 秒打包然后附上签名进行传输。

6.7 区块广播与存储

区块中的所有交易处理完成后，集群 leader 整合各个 BST 的状态，生成区块头广播给其他集群的 leader，同时在 SDFS 上保存区块头和区块体。

区块头生成过程如下：

1）每个 BST 将处理过的交易及交易概要序列化成一个文件，获得该文件哈希值$\mathrm{H}_{\mathrm{BST}}(i), 0 \leq i < 1024$，然后保存文件$\mathrm{H}_{\mathrm{BST}}(i)$到 SDFS 中，$\mathrm{H}_{\mathrm{BST}}(i)$会同时传给 leader；leader 收到所有的$\mathrm{H}_{\mathrm{BST}}(i)$之后会将它们整合到一个文件中，计算该文件哈希值$\mathrm{H}_{\mathrm{tx}}$，将文件$\mathrm{H}_{\mathrm{tx}}$写入 SDFS 中，这样就生成了区块头中的 body 这一元素，同时保存了区块体。

2）每个 BST 将自己全局状态的根哈希以及代表交易及交易概要状态的根哈希传送给 leader，leader 依次生成区块头中的 state_root、transaction_root 和 tx_summary_root；

3） 在区块头填入父块的哈希值（即父块头文件名），加上高度，时间戳，成员报酬等，最终构建出了区块头，计算出哈希值$\mathrm{H}_{\mathrm{header}}$，将文件$\mathrm{H}_{\mathrm{header}}$保存到 SDFS，同时广播区块头。

通过以上三个步骤，整个区块就存储到了 SDFS，同时也广播给了所有集群。

高吞吐量会给存储带来压力，太古老的区块体会被定时删除以释放空间。系统会定期地在 SDFS 中保存下全局状态，在这之前的区块体就可以择机删除，新加入的集群可以根据全局状态快照重放后面的交易来更新到最新状态。一般会在 SDFS 保存多个时间点的状态快照，构成一个滑动窗口。到了保存快照的时间点，生成区块的集群在区块获得认可后，由每个 BST 将自己的状态序列化成文件并保存到 SDFS 中，每个文件的哈希值被传给 leader，leader 将这 1024 个值组织成一个文件，将该文件保存到 SDFS 中，并广播该文件名。这样所有的集群都知道了这个时刻的全局状态被保存下来。

图 8 展示了逻辑处理层对一笔交易的处理过程。Dispatcher 根据交易中发送方的地址可以知道该往哪个 TxMaster 发送该交易。每个 TxWorker 节点都包含一些 BST，每个 BST 都可以从一个节点迁移到另外一个节点以实现负载均衡。每个 TxMaster 负责一个 BST 的集合，它知道每个 BST 当前在哪个 TxWorker 中。TxMaster 负责驱动一条交易的执行，它记录着已经完成的交易和等待完成的交易，同时它也是集群 leader 向 BST 传送消息的通道。

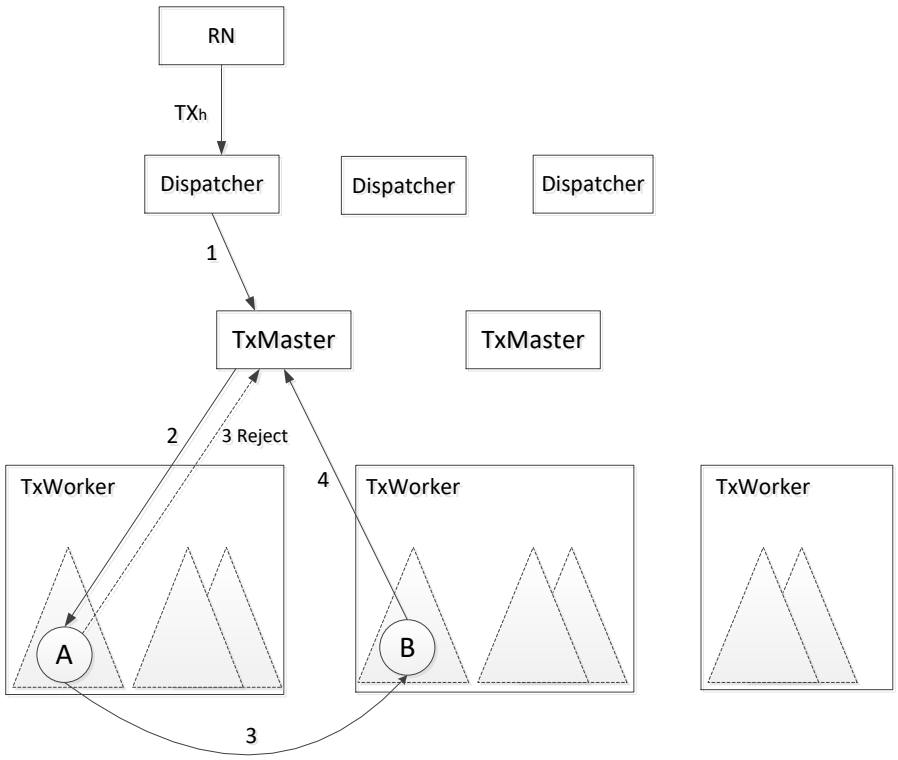

图 8　1）Dispatcher 收到交易后将其转发给相应的 TxMaster； 2）TxMaster 转发交易到转账者 A 的账户；3）A 执行 W 操作，拒绝这笔转账或向 B 发送收款信息；4）B 执行 D 操作后向 TxMaster 告知交易执行完成。

6.8 容错机制

容错机制主要针对由 LN 构成的逻辑处理层，可以分以下几个级别：

Rank 0 ：无容错

全局状态的任何部分都没有备份，任意一台 LN 出现问题，都导致集群无法运行下去，需要通过其他集群进行恢复

Rank 1：状态副本

一个 LN 的每个基本状态单元（如 BST），都有一个备份 LN 保存区块刚开始时的快照以及状态改变日志，只有当处理该基本状态单元的 LN 失效后，备份 LN 根据日志进行恢复并接力原先的 LN；

Rank 2: 双并行处理

一个 LN 的每个基本状态单元都有另一个 LN 在同时进行处理，当一个 LN 失效后，集群不受影响继续工作

Rank 3：多并行处理

一个 LN 的每个基本状态单元有多个 LN 在同时进行处理。

如果一个集群的 LN 仅有一台计算机构成，那么它可以采取 Rank 0，无需容错，或者 Rank3，从 NodeMarket 寻找一些时间段较长的机器，如果自己失效，集群依旧可继续运行，自己恢复后可重新加入；对于由若干台机器组成高速局域网的逻辑处理层，可以采用 Rank 1 或 Rank 2。通过合理的容错机制，可以将出块失败的概率限制在万分之一甚至十万分之一以下。

6.9 智能合约

集群链的智能合约运行在 Java 虚拟机的沙盒中，因此智能合约的开发不局限于 Java，还包括了所有能编译运行在 JVM 上的其他语言，如 Scala。系统将提供 SDK 以方便合约开发。由于以太坊使用 Solidity 作为合约开发语言，并且大量的合约是用该语言实现，因此为了使这些代码可以不用修改就可以运行在集群链上，EVM 可以以插件的形式被支持。

为了确保合约能够终止，需要计算每次合约调用的代价。我们将合约的执行不再看成一条条指令，而是一个个逻辑块，在划分逻辑块时要确保其内部不会出现循环。每个逻辑块都对应相同的 gas，这个值由社区决定。每次合约调用都会设置一个最大消耗值，当运行的逻辑块的总代价超过了这个最大消耗值，将终止合约的执行。

7. 团队核心成员

李波：大数据及人工智能方向技术专家，多年互联网及大数据研发经验，精通分布式文件系统、分布式计算、流处理、人工智能算法等领域。Hadoop Contributor，为 Hadoop 贡献了大量核心代码。曾就职于 Intel 大数据团队，参与主导了集群管理软件 Intel-Manager 的开发，具备集群管理方面的丰富研发经验。HDFS-EC 项目的主要设计者和实现者，该项目是 Hadoop 3.0 的主要改进之一。南京大学计算机学士学位，复旦大学计算机硕士学位。

程浩：Intel 大数据技术团队研发经理，大数据技术专家，带领近二十人的研发团队持续参与 Spark 开源社区，贡献了大量的技术优化和新特性代码，确保大数据软件框架在 Intel 服务器上的最佳运行效率，对 Intel 平台软硬件及相关加速库之上的计算优化有深刻认识和丰富调优经验，畅销书《Spark 大数据处理技术》作者之一。加入 Intel 之前，在移动 GIS 领域有丰富的研发和带领团队经验。华中师范大学计算机学士学位。

吕镭：拥有丰富的软件开发和管理经验。曾在上海群硕软件开发有限公司工作，任中方员工最高技术级别职务。带领过近百人开发团队，为微软、摩托罗拉等企业开发内部系统。在微软工程院（上海）带队参与了 Visual C++ 2008 的研发工作。南京乐科软件设计有限公司创始人。南京大学电子工程系本科及硕士学位。

8. 开发路线图


2018 Q2
ClusterChain技术路线发布
2018 Q3
MVP及性能报告
完成集群内部基本功能
2018 Q4
完成集群间交互
完成NodeMarket功能
完成智能合约SDK
完成跨链通讯
2019 Q1
完成钱包、区块浏览器
完成移动端App
测试网上线
2019 Q2
主网上线
启动第一条10万TPS单链
生态建设
根据需求扩充单链
或启动更多单链

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,http://bitcoin.org/bitcoin.pdf,” 2008.

[2] E. Foundation, “Ethereum’s white paper,” https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[3] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in Proceedings of the Third Symposium on Operating Systems Design and Implementation, ser. OSDI ’99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186

[4] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford,“Omniledger: A secure, scale-out, decentralized ledger,” *IACR Cryptology ePrint Archive*, vol. 2017, p. 406, 2017.

[5] “The Zilliqa Technical Whitepaper” , https://github.com/Zilliqa/Zilliqa,2017

[6] “IPFS-Content Addressed,Versioned,P2P FileSystem” , https://github.com/ipfs,2016

[7] “Internet of Services:The Next-generation,Secure,Highly Scalable Ecosystem for online Services” , https://iost.io,2017

[8] 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, 2015.

[9] EOS , https://github.com/EOSIO,2017

[10] Joseph Poon,Thaddeus Dryja,“The Bitcoin Lightning Network:Scalable Off-Chain Instant Payments”, 2015.

[11] 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 witness cosigning,” in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp.526–545

[12] 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 *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016, pp.279–296.

[13] EYAL, I., GENCER, A. E., SIRER, E. G., AND VAN RENESSE, R. Bitcoin-NG: A Scalable Blockchain Protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, Mar. 2016), USENIX Association.

[14] “HPB:High Performance Blockchain” , www.gxn.io,2017

[15] “Ethash”, https://github.com/ethereum/wiki/wiki/Ethash, Accessed on June 27, 2017., version 23

[16] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, “A secure sharding protocol for open blockchains,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, 2016, pp. 17–30.

[17] HDFS, http://hadoop.apache.org/

[19] Apache Spark, http://spark.apache.org/

[20] Apche Gearpump, http://gearpump.apache.org/overview.html