默克尔-CRDT
Merkle-DAG 遇到 CRDT

He´ctor Sanju´n[1]、Samuli Po ¨yhta ¨ri[2]、Pedro Teixeira[1]和

Ioannis Psaras[1,3]1Protocol Labs
[2]哈加网络
[3]伦敦大学学院

摘要——我们研究 Merkle-DAG 作为无冲突复制数据类型 (CRDT) 的传输和持久层，创造了术语 Merkle-CRDT，并概述了所涉及的不同概念、属性、优点和局限性。我们展示了 Merkle-DAG 如何充当逻辑时钟，使 Merkle-CRDT 有可能极大地简化具有弱消息传递层保证和大量副本的系统中收敛数据类型的设计和实现。Merkle-CRDT 可以利用高度可扩展的分布式技术，如 DHT 和在其下运行的 PubSub 算法，以利用内容寻址的安全性和重复数据删除特性。这种面向内容的系统的示例可以包括机会连接的移动设备、物联网设备或在 Web 浏览器中运行的用户应用程序之间的点对点内容交换和同步应用程序。

I. 介绍

区块链技术的出现已经将点对点网络与加密定向无环图（称为 Merkle-DAG）的使用推广到一起，以在加密货币等应用程序中实现全球分布且最终一致的数据结构。在这些系统中，Merkle-DAG 是一种内容寻址数据结构，用于提供因果关系信息和对象的自我验证，可以在去信任的对等环境中轻松有效地共享。需要维护和应用某些规则以在对抗性场景中向区块链添加新块通常需要使用共识算法。

在分布式系统中获得最终一致性的另一种方法是使用无冲突复制数据类型 (CRDT) [28], [31]。 CRDT 在非对抗场景中很有用，在这种场景中，参与的副本已知行为正确。CRDT 依赖于数据对象本身的一些属性，这些属性能够在不需要共识的情况下收敛到一个全局的、唯一的状态。CRDT 有两种主要形式：基于状态的 CRDT[1]－其中副本的状态形成一个连接半格并在它提供的保证下合并－以及基于操作的 CRDT[2]
－－其中交换操作由每个副本广播并应用于本地状态。此外，δ-CRDT 是基于状态的 CRDT 的优化，以减少副本发送的有效载荷的大小。

Merkle-DAG 和 CRDT 都提供了有趣的支持

erties：前者允许分布式系统利用内容寻址层，用于数据的解析/可发现性和自我验证，无论源位置如何；后者允许全局状态收敛，而无需－－通常是复杂且昂贵的－－共识机制。通过在 Merkle-DAG 节点中嵌入 CRDT 对象，我们获得了两个世界的最佳属性，即我们获得了一个可以利用 DAG 作为逻辑时钟的收敛系统。这个逻辑时钟由每个副本提供和构建，无需协调。副本可以在没有交付保证的松散网络环境中不间断地运行。正如我们将看到的，基于 MerkleCRDT 的系统完全不知道系统如何在副本之间宣布和发现数据，因此能够利用不同的方法，例如 DHT 和 PubSub 机制提供的方法，而无需绑定到它们的特定版本。这与传统的共识算法形成鲜明对比，例如 Raft，与特定的消息传播协议相关联。

我们认为这种方法对于完全分布式和非结构化对等应用程序非常有用，其中副本是公共数据集的写入者，通常以数据库的形式。例如，在分布式和完全复制的文件系统、聊天组或包存储库索引中就是这种情况。我们发现使用星际文件系统 (IPFS) [14] (见节II-F) 作为内容寻址的点对点分散文件系统和内容分发网络，基于 Merkle-CRDT 的系统可以很好地扩展到数千个副本的数量级，这些副本可以机会性地加入和离开－－这是使用移动设备时非常常见的情况，浏览器或物联网设备。

InterPlanetary File System 提供了一个内容寻址的对等文件系统 [14]，它支持 Merkle-DAG 与任意格式和有效载荷的无缝同步，使其成为 PeerPad 等不同类型分布式应用程序的强大构建块[3] 或 OrbitDB[4]，均由 CRDT 和 IPFS 提供支持。 IPFS 不仅为处理内容寻址数据（如我们将使用的 Merkle-DAG）提供了合适的环境，而且还允许我们的 CRDT 应用程序与分布式系统的较低级别完全分离：网络传输、发现和数据传输设施以模块形式出现，可以独立交换和调整。利用这样的系统极大地简化了 CRDT 层的设计和优化，因此

[1]也称为收敛 CRDT 或 CvRDT。
[2]也称为交换 CRDT 或 CmRDT。
[3]PeerPad 是一个实时 p2p 协作编辑工具 (https://peerpad.net).
[4]OrbitDB 是去中心化网络的点对点数据库 (https://github.com/orbitdb/orbit-db).

它可以很好地适应它要服务的用例。

在本文中，我们将我们所称的 MerkleCRDT 形式化。目标是概述它们的特性、优势和局限性，以便为未来的研究和空间优化奠定基础。例如，Merkle-CRDT 允许在网络中的现实世界系统中构建完全分布式的键值存储，没有消息传递保证和完全灵活的副本集，可以在不影响 CRDT 层的情况下随时增长和缩小。

本文的贡献如下：

- 我们将 Merkle-Clocks 定义为基于 Merke-DAG 的逻辑时钟，以在分布式中表示因果关系信息
  系统。使用 MerkleDAG 嵌入因果关系信息是加密货币和 Git 等源代码控制系统的核心，但它们很少被单独视为一种逻辑时钟。我们证明了 Merkle-Clocks 可以用来代替 CRDT 传统上使用的其他逻辑时钟，如版本向量和向量时钟。我们展示了 Merkle-Clocks 实际上可以被视为 CRDT 对象本身，它们可以同步和合并，并且我们可以正式证明不同副本之间的最终一致性。
- 我们将 Merkle-CRDTs 定义为 CRDT 有效载荷的通用传输和持久层，它利用了 Merkle-Clocks 的属性，使用 DAG-Syncer
  和广播器通过设计提供每个对象的因果一致性。这使得可以在具有弱消息传递层保证和大量副本的系统中使用简单的 CRDT 类型。

我们在本文中的目的是表明副本之间的最终一致性可以独立于底层传输机制并在所有对等点都相等的设置中实现。这意味着 Merkle-CRDT 可以在任何底层对等发现和路由系统（例如，DHT、PubSub）之上实现。我们认为这是一个非常强大的概念，有可能导致新的最终一致性系统设计。它也与传统的共识算法（例如 Raft）有着根本的不同，传统的共识算法在任何给定时间都需要有一个领导节点来从所有其他节点收集最新状态。因此，我们不提供针对这些传统方法的评估，因为它们的要求和设计原则根本不同。

论文的其余部分组织如下：在部分II，我们首先介绍相关的背景概念和现有技术。在节III，我们公开了我们系统模型的特征，并介绍了存储和同步 Merkle-CRDT 所需的设施。

在节IV，我们介绍了 Merkle-Clocks，并在前面部分的基础上，在部分V，我们定义了 MerkleCRDT。我们讨论了不同的 CRDT 有效载荷（无论是基于操作、基于状态还是基于 δ 的）如何从 MerkleCRDT 中受益。最后，我们描述了 Merkle-CRDT 的一些局限性和低效率，并在部分介绍了克服它们的技术VI.

II. 背景及相关工作

A. *最终一致性*

分布式系统中的最终一致性（EC）是指系统副本之间的状态可能不相同的情况，但是，如果有足够的时间，并且可能在网络分区、停机和其他意外事件得到解决之后，系统设计将确保状态在任何地方都变得相同。

最终一致性定义的主要弱点是它不能保证共享状态何时收敛或在此之前允许单个状态发散多少[5]. 强最终一致性（SEC）通过建立额外的安全保证来解决这些问题：如果两个副本收到相同的更新，它们的状态将相同。

共识算法，或者对本文更重要的是，无冲突复制数据类型（CRDT）是在分布式系统中实现（强）最终一致性的方法。

B. *默克尔 DAG*

有向无环图（DAG）是一种边有方向且不允许循环的图。Merkle-DAG 是一个 DAG，其中每个节点都有一个标识符，这是对节点内容进行哈希处理的结果。识别一个
数据对象（如 Merkle-DAG 节点）通过其哈希值被称为内容寻址（例如，A = Hash(B) → B = Hash(C) → C = Hash(∅)）。因此，我们将节点标识符命名为内容标识符或 CID。

Merkle-DAG 是自我验证且不可变的结构。节点的 CID 与其有效载荷的内容及其所有后代的内容明确相关联。因此，具有相同 CID 的两个节点明确表示完全相同的 DAG。这将是高效同步 Merkle-CRDT 的关键属性，而无需复制完整的 DAG，就像 IPFS 等系统所利用的那样。

Merkle-DAG 被广泛使用。源代码控制系统 [13] 像 Git [17] 使用它们以一种能够消除重复对象和检测分支之间冲突的方式有效地存储存储库历史记录。在 Dynamo 等分布式数据库中 [19]，Merkle-Trees 用于有效比较和协调副本之间的状态。在哈希历史中 [21]，内容寻址用于引用表示状态的 Merkle-Tree。

Merkle-DAG 也是区块链的基础块——它们可以被视为具有单个分支的 Merkle-DAG——以及它们最常见的应用：加密货币（例如，[3], [1], [18]）。像比特币这样的加密货币 [24] 受益于链中编码的嵌入因果关系信息：链中更深的区块中的交易总是发生在较早区块的交易之前。加密货币的主要问题之一是让所有参与的对等方就链的尖端/头部/根达成一致。

许多这些系统的一个共同点是 Merkle-DAG 隐式嵌入了因果关系信息。这

[5]EC 只提供活性保证：系统在收敛过程中不会卡住。

DAG 可以显示某个事务先于另一个事务，或者需要合并 Git 提交而不是快进。这将是我们在 Merkle-CRDT 中使用的属性之一，本文将其明确化并与其他称为逻辑时钟的因果关系编码机制形成对比。

C. 逻辑时钟

因果收敛系统的设计涉及不同副本之间不同状态版本的协调，例如，当事件同时发生时。

逻辑时钟是全球时间的替代方案，传统上证明很难在分布式系统中实现。25]。它们提供了对分布式系统中不同参与者已知的事件之间的因果信息进行编码的方法。

逻辑时钟是因果历史的表示[12] 并在事件之间提供部分排序。也就是说，给定两个事件 a 和 b，逻辑时钟应该能够告诉我们 a 是否发生在 b (a → b) 之前，反之亦然 (b → a)，或者两者都发生
a 和 b 同时发生 (a /I b)。

逻辑时钟的实际实现通常涉及与系统中每个事件相关的元数据。最常见的逻辑时钟形式之一是版本向量 [26]：每个副本维护并广播一个向量，该向量跟踪所有副本的状态所在的版本。当副本执行状态修改时，它会增加其版本。当一个副本从不同的副本合并一个状态时，它在本地版本和另一个副本提供的版本之间取最高的
事件。因此，给定两个事件 a，b，具有版本向量 V^a，
V^b：a → b 如果 $V^a \leq V^b$ 对于向量中的每个位置 i。如果

$$i\ i$$

a → b 和 b → a，根据该定义，a 和 b 是并发的。

在本文中，我们证明了 Merkle-DAG 可以充当逻辑时钟。正如我们将要展示的，Merkle-Clocks 提供一组不同的属性，但对事件的因果信息进行编码。

D. 无冲突复制数据类型 (CRDT)

CRDT 是一种数据类型，通过要求状态和/或修改它的操作的某些属性，在分布式系统中的不同副本之间提供强大的最终一致性。此外，CRDT 还具有单调性。应用于数据类型的单调性概念是这样一种概念，即每次更新都是一次膨胀，使状态增长，而不是大小，而是相对于先前的状态。这意味着状态之间总会有秩序。单调性意味着无论更新发生的顺序如何，都不需要回滚状态。

有两种突出的 CRDT 类型：基于状态和基于操作的 CRDT。在基于状态的 CRDT 中，系统中的所有状态——即不同副本和不同时刻的状态——形成一个单调的连接半格。这意味着，对于任何两个状态 X 和 Y ，两者都可以“加入”
（合并，或形成联合）（⊔）结果是一个新的状态
对应于两个 [的最小上限 (LUB)28]。换句话说，副本对状态所做的每一次修改都必须是膨胀，并且两个状态 X 和 Y 的并集是能够同时包含两个 X 的最小状态
和 Y 而不是更多 (LUB)。因此，连接半格是一个偏序集，其 LUB 是能够包含半格中所有状态的最小状态。这意味着
⊔ 操作必须是幂等的 (X ⊔ X = X)、可交换的 (X ⊔Y = Y ⊔X) 和结合的 ((X ⊔Y)⊔Z = X ⊔(Y ⊔Z))。

基于状态的 CRDT 中的副本修改它们的状态——或膨胀它——并将结果状态广播给其余的副本[6]. 收到状态后，其他副本将其与本地状态合并。状态的属性确保，如果副本正确接收了其他副本发送的状态（反之亦然），它们最终会收敛。

基于操作的 CRDT [28]，另一方面，不对状态本身强制执行任何属性，而是对用于修改它的操作强制执行任何属性，这些属性必须是可交换的（至少对于同时（并发）发出的不同操作而言。副本广播操作而不是状态。如果两个操作在两个副本中同时发生，则其他副本应用它们的顺序无关紧要：结果状态将相同。

因此，如果操作广播没有到达副本——例如由于网络故障——，该副本将永远无法应用它，状态也不会收敛。因此，与基于状态的 CRDT 不同，基于操作的 CRDT 中的最终一致性需要一个可靠的消息传递层，最终交付所有操作。11]。额外的约束可能是必要的，例如，如果操作不是幂等的：在这种情况下，消息传递层应该确保每个操作只传递一次。

逻辑时钟，如上一节中所见，通常用于实现 CRDT 类型：它们有助于
确定两个更新何时同时发生并且需要
合并。 CRDTs 已在以下领域成功使用和优化
不同的应用程序和分布式数据库，Basho 的 Riak [15], [16] 是最突出的例子之一。

E. 以信息为中心的网络中的同步协议

有多种类型的逻辑时钟与前面讨论的版本向量类似，但满足不同的需求或解决它们的一些缺点：向量时钟 [20]，有界版本向量 [8]，点阵向量 [27]，树钟 [23] 或间隔树时钟 [9] 是其中一些。

在以信息为中心的网络 (ICN) 领域，最近在分布式数据集同步方面进行了大量工作。30], [29], [33], [7]。以信息为中心的架构提倡在网络层直接命名内容，并根据内容名称和（在某些情况下）最长前缀匹配进行后续路由和转发（通过核心网络路由器）。

时间同步 [33] 正在利用 NamedData 网络架构的特性 [32] 同步不同数据集之间的状态。 ChronoSync 是一种数据层机制，然而，它利用了 NDN 正在构建的分层和灵活的命名方案。启发

[6]这里的一个重要说明是 CRDT 只是数据类型。副本之间CRDT对象的传输策略是独立的。某些 CRDT 在设计上比其他广播机制更适合某些广播机制，并且可以促进优化，例如仅广播到随机子集而不是每个副本。

通过 Merkle Trees，ChronoSync 正在使用加密摘要和过滤器来同步对等点之间的数据集。 ChronoSync 利用底层网络 (NDN) 基于名称的特性，并为每个对等点分配唯一的发布前缀。这个独特的前缀，连同序列号和网络层持久/长寿命的“兴趣包”正在取代其他方法试图用时钟和 CRDT 做的大部分事情。虽然在网络层集成同步功能允许更多原生设计，但某些功能不可避免地会丢失。例如，ChronoSync 无法处理同时（并发）数据发布。回合同步［30］通过将同步过程分成几轮来部分解决这个问题。

矢量同步［29］是 ChronoSync 的增强版
[33］它使用版本向量来提高对等点之间的同步效率。如前所述，版本向量在检测不一致方面比简单的消息摘要更有效。然而，与该领域的其他提案类似，VectorSync 需要实现“基于领导者的成员资格管理”，以便处理更新数据集状态的活跃成员。

在网络的网络层本地集成分布式数据集同步功能显然是一项先进的努力，它带来了自己的挑战。我们相信 Merkle-CRDT 带来的“传输不可知”状态同步的优势可以应用并提高 ChronoSync、RoundSync 或 VectorSync 等协议的性能。另一方面，直接通过命名网络对象处理基于 Merkle-CRDT 的状态同步为 Merkle-CRDT 带来了标准 ICN 优势。因此，我们认为这两种不同的状态同步方法是互补的。

F. IPFS：行星际文件系统

IPFS [14］是一个内容寻址的分布式文件系统。它使用分布式哈希表 (DHT) 来宣布和发现哪些副本（或对等方）提供某些 Merkle-DAG 节点。它实现了一个名为 bitswap 的节点交换协议，以从任何提供商处检索 DAG 节点。 IPFS 建立在 libp2p 之上［5］，一种用于 P2P 网络的模块化网络协议栈，另外还提供了主要基于发布订阅模型的高效广播机制［2].

*IPFS 还使用 IPLD，即星际链接数据格式［*4］，一个描述具有任意节点格式并支持多种类型 CID 的 Merkle-DAG 的框架［6］，使创建和同步自定义 DAG 节点变得非常容易。

这些特性使 IPFS 成为实施 Merkle-CRDT 的合适层，因为它提供了在潜在的非常大的网络中发现、路由和宣布内容的必要机制。这并不是说其他传输机制不适合在顶部构建 Merkle-CRDT。

III. 系统模型和假设

我们的 Merkle-CRDT 方法既简单又便于在具有大量副本且没有消息传递保证（即不可靠传输）的对等分布式系统中使用 CRDT。

我们假设存在一个异步消息传递层，它在不同的副本之间提供通信通道。该通道由每个副本利用的两个设施管理：DAG-Syncer 和 Broadcaster 组件（定义如下）。

我们假设消息可以被丢弃、重新排序、损坏或复制。不需要事先知道参与系统的副本数量。副本可以随意加入和离开，无需通知任何其他副本。可能存在网络分区，但一旦重新建立连接并且副本广播新事件，它们就会被解决。

副本可能具有持久存储，这取决于它们自己的要求和数据类型。使用 Merkle-CRDT，只要至少有一个其他副本处于最新的系统状态，没有持久存储的新副本和崩溃副本将最终能够重建系统的完整状态。

A. DAG-Syncer 组件

DAG-Syncer 是一个组件，它使副本能够从给定内容标识符 (CID) 的其他副本获取远程 Merkle-DAG 节点，并使其自己的节点可用于其他副本。由于节点包含指向其直接后代的链接，给定根节点的 CID，DAG-Syncer 组件可用于通过跟踪每个节点中的子节点的链接来获取完整的 DAG。因此，我们可以如下定义 DAGSyncer：

定义 1.（DAG-同步器）。 DAG-Syncer 是一个具有两种方法的组件：

- 获取（CID）：节点
- 放置（节点）

我们没有指定更多细节，例如宣布和检索节点的协议是什么样的。理想情况下，DAG-Syncer 层不应对系统模型施加任何额外的约束。我们的方法依赖于 DAG-Syncer 和 Merkle-DAG 的特性来容忍上述所有网络突发事件。

B. 广播组件

广播器是将任意数据从一个副本分发到所有其他副本（直接或通过中继）的组件。理想情况下，有效载荷将到达系统中的每个副本，但这不是每个广播消息的要求：

定义 2.（广播公司）。广播器是具有一种方法的组件：

- 广播（数据）

C. IPFS 作为 DAG-Syncer 和 Broadcaster 组件

上面的组件可以通过使用 IPFS 来实现（如第II）。 IPFS 可以充当 DAG-Syncer，而其 libp2p 层提供的一种 PubSub 机制可以执行 Broadcaster 组件的任务。

这样的实现一般应该允许副本集的极端可扩展性。网络中的对等点不需要与其他人完全连接，并且系统非常模块化和可配置，以适应小型

设备和大型存储服务器。设置和实现的选择会影响系统在不同环境和网络拓扑结构下的性能，但与 Merkle-CRDT 对象和数据类型无关。

IV. 默克钟

A. 概述

Merkle-Clock M 是一个 Merkle-DAG，其中每个节点代表一个事件。换句话说，给定系统中的一个事件，我们可以在这个 DAG 中找到一个代表它的节点，并允许我们将它与其他事件进行比较。

DAG 是通过根据一些简单的规则合并其他 DAG（其他副本中的那些）来构建的。新事件被添加为新的根节点（现有节点的父节点）。请注意，默克尔时钟在给定时间可能有多个根。

例如，给定 M_α和 M_β（α 和 β 是这些 DAG 中的单个根 CID[7]）：

1) 如果 $\alpha = \beta$ 不需要任何动作，因为它们是相同的 DAG。
2) 否则如果 $\alpha \in M_\beta$，我们保留 M_β作为我们的新时钟，因为 M_α中的历史已经是它的一部分。在这种情况下，我们说 $M_\alpha < M_\beta$。
3) 否则如果 $\beta \in M_\alpha$，我们出于同样的原因保留 M_α。在这种情况下，我们说 $M_\beta < M_\alpha$。
4) 否则，我们通过保持两个 DAG 原样来合并两个时钟，因此有两个根节点，这些节点由 α 和 β 引用。请注意，M_α和 M_β可能完全不相交，也可能不相交，这取决于它们是否共享一些更深的节点。如果我们想记录一个新事件，我们可以创建一个新的根 γ，它有两个孩子，α 和 β。

我们已经可以看到，通过确定一个 MerkleClock 是否包含在另一个 MerkleClock 中，我们在 Merkle-Clocks 之间引入了顺序的概念。以同样的方式，我们在每个时钟中的节点之间有一个顺序的概念，因为较早发生的事件将始终是较晚发生的事件的后代。此外，我们还引入了一种根据它们的比较方式合并 Merkle-Clocks 的方法。生成的 Merkle-Clock 始终包含来自两个 Merkle-Clock 的因果关系信息。这最终意味着每个副本中存储在 Merkle-Clock 中的因果关系信息在合并后将收敛到同一个 Merkle-Clock。

Merkle-Clocks 提供的因果顺序在构建具有类似规则的 Merkle-DAG 时被嵌入，并且通常被忽略为非常直观的东西。然而，重要的是要形式化我们如何定义 Merkle-Clocks 之间的顺序，并证明在它们同步和合并时维护因果关系信息。这是下一节的主题，将成为 Merkle-CRDT 的重要属性。

B. Merkle-Clocks 作为收敛的复制数据类型

本节将 Merkle-Clock 的定义及其作为 Merkle-Clock DAG 的表示形式化。我们将展示 Merkle-Clock DAG 可以被视为一个增长集（G-Set）CRDT，因此可以在多个副本中收敛。

设 S 是所有系统事件的集合：

[7]在示例中，不失一般性，我们假设我们从包含单个根而不是多个根的 DAG 开始。

定义 3.（Merkle-Clock 节点）。Merkle-Clock 节点 n_α 是一个三元组：

$$(\alpha, e_\alpha, C_\alpha)$$

表示事件 $e_\alpha \in S$，α 是节点 CID，C_α是 n_α的直接后代的 CID 集。

定义 4.（Merkle-Clock DAG）。Merkle-Clock DAG 是一对：

$$N, \leq$$

其中 N 是一组不可变的 DAG 节点和偏序 $\leqslant$ on N，定义如下：

$$n_\alpha, n_\beta \in N : n_\alpha < n_\beta \Leftrightarrow n_\alpha \text{是 } n_\beta \text{的后代}$$

换句话说，如果存在从 n_β到 n_α的链接节点路径，则 $n_\alpha < n_\beta$。

为了维持这种关系，Merkle-Clock DAG 必须使用以下实现规则构建：

红外线。系统中的每个新事件都必须表示为现有 Merkle-Clock DAG 的新根节点。特别是，C 集必须包含先前根的 CID。

定义 5.（默克尔时钟）。Merkle-Clock（M）是一个函数，它给定一个事件 $e_\alpha \in S$ 从 Merkle-Clock DAG N 返回一个节点：

$$M : S \to N$$

评论。Merkle-Clock 满足强时钟条件 [22]。我们看到每个节点都代表比其子节点晚的事件：

$$\forall (\beta, e_\beta, C_\beta) \in N : \forall \alpha \in C_\beta : e_\alpha \to e_\beta$$

由于每个事件都是使用实现规则构建的（子）DAG 的根，我们可以立即看到较早的 Merkle-Clock 值是较晚的值的后代：

$$M(e_\alpha) < M(e_\beta) \Leftrightarrow e_\alpha \to e_\beta$$

我们现在可以将 Merkle-Clocks DAG 的连接半格定义为一对：

$$J, \subseteq_J$$

其中 J 是一组 Merkle-Clocks DAG，$\subseteq_J$是该集合的偏序，定义如下。给定 M, N $\in$ J：

$$M \subset_J N \Leftrightarrow \forall m \in M, \exists n \in N \mid m < n \Leftrightarrow M \subset N$$

请注意，$m < n$，意味着 m 是 n 的后代，因此必须属于同一个 DAG，那么 $\subset_J$仅仅意味着 M 是 N 的子集。

这允许我们将两个 Merkle-Clocks DAG（$\sqcup_J$）的最小上界定义为集合的常规并集：

$$M \sqcup_J N = M \cup N$$

不出所料，Merkle-Clock 表示实际上对应于基于状态的 CRDT 形式的 Grow-Only-Set（G-Set）[31]。该集合的元素是不可变的、加密链接的并代表系统中的事件。

在下一节中，我们将看到 Merkle-DAG 的属性如何允许以比常规基于状态的 G-Set 更有效的方式同步 Merkle-Clock。

C. 时钟中的默克尔：默克尔时钟的属性

到目前为止，我们已经定义了一种对每个副本的因果关系信息进行编码的方法，并确保两个副本可以合并它们的 Merkle-Clocks。现在我们将看到 Merkle-DAG 的属性如何允许使用拉而不是推方法，与内容寻址一起，实现副本之间的有效时钟同步并克服网络分区或突发事件的影响。下面给出了在副本之间进行 MerkleClock 同步的步骤。

1) 广播 Merkle-Clock 只需要广播当前的根 CID。整个时钟由其根的 CID 明确标识，其完整的 DAG 可以根据需要从它走下来。
2) Merkle-DAG 的不可变特性允许每个其他副本执行快速比较并仅拉/取那些它尚未拥有的节点。
3) Merkle-DAG 节点通过其 CID 进行自我验证，因此不受损坏和篡改的影响。因此，可以从任何愿意提供它们的来源获取（拉取）它们，无论是否信任。
4) 相同的节点通过设计去重复：每个事件只能有一个唯一的表示。

在实践中，每个副本只是从其他副本中获取增量因果历史，而无需在系统的任何地方明确构建这些增量。没有以前历史的全新副本将自动获取完整历史[8].

Merkle-Clocks 可以替换版本时钟和其他通常作为 CRDT 一部分的逻辑时钟。这有一些考虑：

- 通过使用 Merkle-Clocks，我们可以将因果关系信息与副本数量分离，这是版本时钟的一个常见限制。这使得有可能
  在实现 CRDT 时减少消息的大小，最重要的是，解决了在副本随机加入和离开系统时保持时钟工作的问题。
- 不利的一面是，因果信息随着每个事件而增长，即使事件信息被合并到更小的对象中，副本也可能存储大量的历史记录。
- 保留整个因果历史使新副本能够开箱即用地从头开始同步事件，而无需向新来者明确发送系统快照。如何
  –
  曾经，如果历史记录非常大，那么同步可能会很慢。我们将与 Merkle-CRDT 一起探索这方面的潜在优化。

Merkle-Clocks 与传统版本时钟相比的一个显着优势在于，它们还可以处理多种类型的网络异常：

- 丢弃的消息可能会阻止其他副本了解新的根。但是由于每个 Merkle-Clock DAG 都被未来的 DAG 和每次下载所取代
  获取 DAG 的所有缺失部分，网络分区

  和副本停机时间对整体没有影响
  系统，并在问题解决后开始自动修复。
- 出于同样的原因，乱序交付没有问题。将获取并处理丢失的 DAG
  为了。
- 复制的消息只会被副本忽略，因为它们已经合并到他们的默克尔时钟中。
- *损坏的消息有两种形式：* a) 如果广播新根的消息损坏，那么它将是对应于不存在的 DAG 的散列，它不能
  由 DAG-Syncer 获取，最终将被忽略； b) 如果 DAG 节点在下载时损坏，并且 DAG-Syncer 组件（或应用程序）的 CID 与下载的内容不匹配，则可以丢弃它。

正如我们在上一节中展示的，Merkle-Clocks 表示事件的严格偏序。并非系统中的所有事件都可以进行比较和排序。例如，当有多个根时，Merkle-Clock 无法说出哪个事件先发生。

总订单可能很有用 [22] 并且可以例如通过将并发事件视为相等来获得。类似地，可以通过根据节点的 CID 或基于附加到时钟节点的附加信息的任何其他任意用户定义的策略对并发事件进行排序来构建严格的总顺序。任何此类方法都可以作为数据层冲突解决方案。

V. MERKLE-CRDTS：带有效载荷的默克尔时钟

定义 6. (Merkle-CRDT)。 Merkle-CRDT 是一个 MerkleClock，其节点携带任意 CRDT 有效载荷。

Merkle-CRDT 保留了之前在 Merkle-Clocks 中看到的所有属性。但是，要使有效载荷收敛，它们本身需要是收敛数据类型 (CRDT)。优点是 Merkle-Clocks 已经嵌入了排序和因果关系信息，否则这些信息需要嵌入到 CRDT 对象中（通常以其他逻辑时钟的形式）。或由可靠的消息传递层提供。

因此，一个 Merkle-CRDT 节点的实现是：

$$(\alpha, P, C)$$

α 是内容标识符，P 是具有 CRDT 属性的不透明数据对象，C 是子标识符的集合。

A. 每个对象的因果一致性和差距检测

Merkle-CRDT 的定向链接特性允许按事件顺序遍历系统的完整因果历史，提供了所有必要的属性，以确保每个对象的因果一致性和设计上的差距检测，而无需修改我们的系统模型。

这意味着 Merkle-CRDT 非常适合承载基于操作的 CRDT，因为它们可以确保没有操作丢失或无序应用[9].

[8]这就是参与加密货币挖掘的同行同步他们的分类账的方式。

[9]回想一下，Merkle-Clock 提供了严格的事件偏序。在在这种情况下，应用于对象的两个非并发操作将按时钟排序。

为了促进在 Merkle-CRDT 中处理 CRDT 有效载荷的任务，在下一节中，我们将介绍一种通用且简单（未优化）的反熵算法，该算法可用于为任何 CRDT 嵌入对象获得每个对象的因果一致性。

B. *Merkle-CRDT 的通用反熵算法*

定义 7. (MerkleCRDT 的通用反熵算法）。

令 R^A和 R^B是两个使用 Merkle-CRDT 的副本 M_α和 $M_\emptyset$分别作为它们当前的 Merkle-CRDT DAG。

1) R^B通过创建新的 DAG 节点（β, P, {θ}）并将其作为新的根添加到其 MerkleCRDT 中，从而发出新的有效载荷，即 $M_\emptyset$。
2) R^B将 β 广播到系统中的其余副本。
3) R^A接收 β 的广播并检索完整的 $M_\emptyset$。它通过从根 β 开始并使用 DAG-Syncer 组件沿着 DAG 向下走以获取所有不在 M_α中的节点，同时在 CID-Set D 中收集它们的 CID。 DAG，对于 M_α中已经存在的任何 CID，整个子 DAG 可以是
跳过。
4) 如果 D 为空，则无需进一步操作。 R^A必须已经处理了 $M_\emptyset$中的所有有效载荷。这意味着 $M_\emptyset \subseteq M_\alpha$。
5) 如果 D 不为空，我们使用 Merkle-Clock 提供的顺序对 D 中的 CID 进行排序。如果我们的系统不需要因果交付，我们可以跳过订购。 D 中的项目数量将取决于系统中的并发量以及两个 MerkleCRDT 允许发散的时间，但在正常情况下应该很小。
6) R^A处理与 D 中 CID 对应的节点关联的有效载荷，从最低到最高。
7) 如果 $\alpha \in D$，则 $M_\alpha \subseteq M_\emptyset$和 $M_\emptyset$成为 R^A中新的局部 Merkle-CRDT。
8) 否则，$M_\alpha \subset M_\emptyset$和 $M_\emptyset \subset M_\alpha$。 R^A将两个节点都作为根。

C. *基于操作的 Merkle-CDRT*

定义 8. 基于操作的 Merkle-CRDT 是节点嵌入基于操作的 CRDT 有效载荷的那些。

基于操作的 Merkle-CRDTs 是 Merkle-CRDTs 最自然的应用。操作很容易定义，因为它们只需要是可交换的，因此无论它们接收操作的顺序如何，结果状态在每个副本中都是相同的。然而，这也意味着要使状态收敛，必须接收每个操作。可靠的消息传递层［11］然后是收敛的先决条件，但在具有大量副本的现实世界网络中，通常无法确保没有消息丢失。这给我们留下了复杂的解决方法，比如额外的因果负载、缓冲和重试机制，这些机制必须伴随着 CRDT 实现，将原本应该是简单的 CRDT 实现变成了相当复杂的东西。

Merkle-DAG 提供消息传递层的所有属性，其中消息始终按顺序传递、经过验证且永不重复或丢弃。因此，Merkle-CRDTs 在以前不容易使用的上下文中启用基于操作的 CRDTs。

正如我们所看到的，由于嵌入了它们的 Merkle-DAG，每个副本只需要 DAG 中缺失的部分，一旦知道根就可以获取这些部分。这包括加入系统的新副本，这些副本将能够获取和应用所有操作。我们不需要保留完整副本集的知识，也不需要将高效广播的责任放在 Broadcaster 组件中。

D. *基于状态的 Merkle-CRDT*

定义 9. 基于状态的 Merkle-CRDT 是节点嵌入基于状态的 CRDT 有效载荷的那些。

在每个 Merkle-CRDT 节点中嵌入完整状态是违反直觉的，因为基于状态的 CRDT 已经提供了每个对象的因果一致性，并且可以通过设计处理不可靠的消息层。

此外，虽然最终状态是由 Merkle-CRDT 节点中所有状态的合并产生的，但 DAGSyncer 组件仍然需要存储这些状态，这在处理大型状态对象时是禁止的。也就是说，Merkle-CRDT 不需要附加因果关系元数据并将其与副本数量分离，这对于与副本数量相比具有非常小的状态的基于状态的 CRDT 可能很有趣。

一个更有趣的方法是 δ-CRDTs［10］，而不是广播完整的状态，能够发送更小的部分（增量）。 δ-mutations，正如这些对象所称的那样，可以像任何完整状态一样在下游合并，而无需更改联合操作的语义。因此，可以合并多个增量以形成所谓的 δ 组并提高广播有效载荷的效率。正如在［10］，*“一个完整的状态可以被看作是一个特殊的（极端的）delta-group”*。

然而，在 δ-CRDT 的普通形式中，当消息丢失并且基于状态的 CRDT 的每个对象因果一致性属性丢失时，一致性会无限延迟。这些问题可以通过额外的反熵算法来解决，该算法对丢失的增量进行分组、排序、跟踪和重新发送，如［10］，但在 δ-state-Merkle-CRDTs 的情况下，反熵算法和任何附加到原始对象的因果信息都不是必需的。本质上，这种方法使 δ 状态 MerkleCRDT 更接近于基于操作的对应物。

VI. MERKLE-CRDTS 的限制和优化

A. *Merkle-CRDT 的局限性*

到目前为止，我们一直专注于解释与传统 CRDT 方法相比 Merkle-CRDT 提供的不同品质，但我们还必须强调它们带来的内在和实际限制。

a) 不断增长的 DAG-Size：Merkle-CRDTs 最明显的后果是，虽然 CRDTs 通常会合并、应用、合并和丢弃广播对象，但 Merkle-CRDTs 构建了一个必须存储且不断增长的永久 Merkle-DAG。正如我们所见，这提供了许多有利的特性，但也带来了一些影响：

- DAG 的大小可能会增长到超出可接受范围。增长率将取决于
  事件和有效载荷的大小。这与区块链如何及时增长到大尺寸非常相似[10].在某些情况下，可以将状态表示为所有 Merkle-CRDT 操作的结果的压缩，但这将我们带到下一点。
- 如果副本仅存储 Merkle-DAG，并且知道可以从中重建完整状态（从而节省空间），则从具有非常大的 Merkle-DAG 的副本开始
  可能特别慢，因为他们需要重新处理完整的 DAG，即使在本地可用时也是如此。否则，结果状态和 Merkle-DAG 中都会存储冗余信息。
- 当新副本加入时，Merkle-CRDT 从头开始同步对系统来说是可能的并且是自然的。然而，MerkleDAGs 不仅不断增长，而且往往很深
  和瘦[11].新副本将从广播操作中学习根 CID，并且需要从中解析完整的 DAG。由于薄，不可能并行获取多个分支。冷同步可能比发送快照所需的时间长得多，从而使 Merkle-DAG 的这种嵌入属性几乎没有价值。

在某些情况下，非常大的 DAG 和缓慢的同步不是问题，可以被视为可以接受的权衡，但确实强调了探索“垃圾收集”和 DAG 压缩机制的必要性。

b) Merkle-Clock 排序：合并两个 Merkle-Clock 需要比较它们以查看它们是否包含在彼此中并找出差异。如果 DAG 出现显着差异（或很久以前），这可能是一项代价高昂的操作。

c) DAG-Syncer 延迟：副本依赖 DAGSyncer 组件从消息传递层获取和提供节点。如前所述，Merkle-CRDT 与用于同步消息的机制（例如 DHT 或 PubSub）无关，但除非仔细选择，否则这种机制可能会引入同步延迟。根据应用程序的要求，这种延迟可能会也可能不会被接受。

这些限制的实际影响取决于应用程序的要求。特别是，在考虑采用 Merkle-CRDT 时，用户应该考虑 Merkle-CRDT 在以下方面是否是最佳方法：i）节点数量与状态大小，ii）冷同步时间，iii）更新传播延迟， iv）预期的副本总数，v）预期副本集修改（加入和离开），vi）并发事件的预期数量。

B. *优化 Merkle-CRDT*

*a) 延迟 DAG 节点：*在副本发布频繁更新的场景中，我们可以在发布包含所有负载的单个节点之前将多个负载分组。很明显，这种方法会带来一些好处，但也有一些好处：更新不会立即发送出去，因此需要更长的时间来传播。

b) 快速 Merkle-DAG 包含检查：将本地副本 DAG 与远程 DAG 合并需要检查一个 DAG 是否包含另一个。通过沿着第一个 DAG 寻找与第二个 DAG 的根匹配的节点 CID，这样做是可能的，但效率低下。将本地 DAG 的 CID 存储在可以快速检查 CID 是否是本地 DAG 的一部分的键值存储中使事情变得更加容易[12].当遍历远程 DAG 以检查是否包含本地 DAG 时，可以检查其任何节点的子节点的 CID，以查看它们是否是本地 DAG 的一部分，在这种情况下，可以方便地修剪它们的分支。然而，这意味着实现必须知道并且可以访问节点的本地存储系统。当前定义的 DAG-Syncer 无法区分本地或远程可用节点。布隆过滤器、缓存和一些数据结构也可以提高效率，但它们通常是所选存储后端的一部分。

只要应用程序可以容忍它们强加的约束，就可以通过在有效载荷中嵌入版本向量来实现类似的效果。比较有效负载之间的版本向量是一种包含检查，无需执行 DAG 行走。

*c) 广播负载调整：*我们的标准方法通过仅包含新根的 CID 来减少广播的大小。发布机制足够复杂，并且总是受益于较小的有效载荷。

d) 减少 Merkle-DAG 节点大小：我们可以尝试通过压缩和删除 CRDT 本身不需要的冗余信息来尽可能地减少有效载荷的大小。例如，我们可以对广播消息进行签名，而不是对 CRDT 有效负载进行签名以确保它们来自受信任的副本，从而将签名排除在 Merkle-DAG 之外。

*e) 节点中的附加指针：*解决thin-DAG 问题的一种方法是在发布新节点时定期引入对DAG 更深部分的引用。这种方法基本上是向节点添加额外的子节点。通过能够跳转到 DAG 的其他部分，它在获取 DAG 的缺失部分时允许更多的并行性，从而导致更快的遍历。额外链接的实际数量及其目的地将取决于应用程序的需要。

在任何 Merkle-CRDT 实施中都应考虑上述建议，因为它们比之前描述的未优化版本具有显着优势。哪些优化适合哪些实现主要是

[10]在撰写本文时，比特币链使用超过 220GB，以太坊（Parity）使用超过 165GB。

[11]在没有许多并发事件的情况下，Merkle-DAG 会很薄，否则具有高分支因子。在这两种情况下，分支都是每次从副本发出新事件时都会进行合并，从而创建 *DAG 中的细腰。*

[12]快速键值存储（例如内存中的键值存储）通常会付出很高的内存占用量损失，而磁盘支持的键值存储会更慢。

特定于应用程序。我们将 DAG 压缩和垃圾收集的主题留给未来的工作，尽管我们直觉地注意到，在确保每个副本都知道它们之前，不应尝试丢弃 Merkle-DAG 的部分。

VII. 结论

在本文中，我们将 Merkle-DAG 视为具有自我验证和高效同步特性的因果编码结构。这导致我们引入了 MerkleClock 的概念，证明它们可以被描述为基于状态的 CRDT，它通过广播组件宣布并通过 DAG-Syncer 工具获取，在所有副本中收敛。

然后，我们将 Merkle-CRDT 呈现为带有 CRDT 有效载荷的 Merkle-Clocks。我们展示了 Merkle-CRDT 如何在几乎没有消息层保证和对副本集没有约束的情况下工作，副本集可以是动态的和未知的，同时提供每个对象的因果一致性。 Merkle-CRDTs 在 IPFS 生态系统中被广泛用于数据库日志操作[13]，在 OrbitDB 中[14]，一个分布式 P2P 数据库，它的无服务器应用程序 Orbit[15]，以及分布式协作编辑[16] 和移动照片共享应用程序。[17]

Merkle-CRDT 是传统区块链之间的联姻，需要共识才能收敛，而 CRDT 则通过设计收敛，从而继承了两个世界的正面和负面方面。通过这项工作，我们希望为进一步研究该主题奠定良好的基础。

参考文献

[1] DAG 币。https://dagcoin.org.

[2] 八卦子: A 安全的 发布订阅 协议 对于非结构化， 去中心化 P2P 叠加。 https://research.protocol.ai/posts/201912-resnetlab-launch/PL-TechRep-gossipsub-v0.1-Dec-30.pdf。

[3] 物联网。https://iota.org.

[4] IPLD: 行星际关联数据格式。https://ipld.io/.

[5] libp2p: P2P 网络的模块化网络协议栈。https://libp2p.io/.

[6] 多格式: 面向未来的自我描述值。https://multiformats.io/.

[7] Alexander Afanasyev、Zhenkai Zhu、Yingdi Yu、Lijing Wang 和 Lixia Zhang。 chronoshare 的故事，或者 ndn 如何将分布式安全文件共享带回来。在 IEEE MASS，第 525-530 页，2015 年第 10 页。

[8] 乔斯·巴塞拉·阿尔梅达、保罗·塞尔吉奥·阿尔梅达和卡洛斯·巴克罗·莫雷诺。有界版本向量。在国际分布式计算会议 - ICDCS，第 3274 卷，第 102-116 页，日本东京，2004 年 3 月。Springer，Springer。

[9] Paulo S´rgio Almeida、Carlos Baquero 和 Victor Fonte。间隔树时钟。在第 12 届分布式系统原理国际会议论文集，OPODIS '08，第 259-274 页，柏林，海德堡，2008 年。Springer-Verlag。

[10] 保罗·塞尔吉奥·阿尔梅达、阿里·肖克和卡洛斯·巴克罗。通过 delta-mutation 实现高效的基于状态的 CRDT。 CoRR，abs/1410.2803，2014。

[11] 卡洛斯·巴克罗、保罗·塞尔吉奥·阿尔梅达和阿里·肖克。使基于操作的 CRDT 基于操作。在关于最终一致性原则和实践的第一次研讨会的会议记录中，PaPEC '14，第 7:1-7:2 页，纽约，纽约，美国，2014 年。ACM。

[12] 卡洛斯·巴克罗 (Carlos Baquero) 和努诺●普雷吉 (Nuno Pregui)。为什么逻辑时钟很容易。 ACM 队列，2016 年 4 月 14 日。

[13] 彼得·鲍迪斯。版本控制系统中的当前概念。 CoRR，abs/1405.3496，2014。

[14] 胡安·贝内特。 IPFS - 内容寻址、版本化、P2P 文件系统（草案 3），2014 年。

[15] 罗素·布朗、肖恩·克里布斯、克里斯托弗·米克尔约翰和山姆·艾略特。 Riak dt map：一个可组合的、收敛的复制字典。在关于最终一致性原则和实践的第一次研讨会的会议记录中，PaPEC '14，第 1:1-1:1 页，纽约，纽约，美国，2014 年。ACM。

[16] 罗素·布朗、泽尚·拉卡尼和保罗·普莱斯。 Big(ger) 集：在 riak 中分解的 delta CRDT 集。 CoRR，abs/1605.06424，2016。

[17] 斯科特·查康和本·斯特劳布。专业的 Git。美国加利福尼亚州伯克利 Apress，第 4 版，2018 年。

[18] 安东·丘留莫夫。 Byteball：用于存储和转移价值的去中心化系统，2016 年。

[19] Giuseppe DeCandia、Deniz Hastorun、Madan Jampani、Gunavardhan Kakulapati 、 Avinash Lakshman 、 Alex Pilchin 、 Swaminathan Sivasubramanian、Peter Vosshall 和 Werner Vogels。 Dynamo：亚马逊的高可用键值存储，2007 年。

[20] C.J.菲奇。保留偏序的消息传递系统中的时间戳。第 11 届澳大利亚计算机科学会议论文集，10(1):56–66，1988 年。

[21] Brent ByungHoon Kang、罗伯特●威伦斯基和约翰●库比亚托维奇。用于调和相互不一致的哈希历史方法。 In Proceedings of the 23rd International Conference on Distributed Computing Systems，ICDCS '03，pages 670–，Washington，DC，USA，2003. IEEE 计算机协会。

[22] 莱斯利·兰波特。分布式系统中的时间、时钟和事件顺序。社区。 ACM，21(7):558–565，1978 年 7 月。

[23] 托比亚斯·兰德斯。树钟：一种高效且完全动态的逻辑时间系统。 In Proceedings of the 25th IASTED International Multi-Conference：Parallel and Distributed Computing and Networks， PDCN'07，pages 375–380，Anaheim，CA，USA，2007. ACTA Press。

[24] 中本聪。比特币：点对点电子现金系统，2009 年。

[25] 乔治·内维尔-尼尔。时间是一种幻觉。 ACM 队列，13(9)，2016 年。

[26] D. S. 帕克、G. J. Popek、G. Rudisin、A. Stoughton、B. J. Walker、E. Walker 已 on、J. M . Chow、D. Edwards、S. Kiser 和 C. Kline。检测分布式系统中的相互不一致。 IEEE 翻译软件Eng.，9(3):240–247，1983 年 5 月。

[27] Nuno M. Pregui?a 、 Carlos Baquero 、 Paulo Se'rgio Almeida 、 Victor Fonte 和 Ricardo Gonc?alves。虚线版本向量：用于乐观复制的逻辑时钟。 CoRR，abs/1011.5808，2010。

[28] Nuno M. Pregui?a、Carlos Baquero 和 Marc Shapiro。无冲突复制数据类型 (CRDT)。 CoRR，abs/1805.06358，2018。

[29] 尚文涛、亚历山大·阿法纳西耶夫和张丽霞。 Vectorsync：基于命名数据网络的分布式数据集同步。 In Proceedings of the 4th ACM Conference on Information-Centric Networking， ICN '17，page 192–193，New York，NY，USA，2017. 计算机协会。

[30] 尚文涛、于英迪、王丽晶、亚历山大·阿法纳西耶夫和张丽霞。命名数据网络中分布式数据集同步的调查。技术报告 NDN-0053，NDN，2017 年 5 月。

[31] Marc Shapiro、Nuno Preguic?a、Carlos Baquero 和 Marek Zawirski。收敛和交换复制数据类型的综合研究。研究报告 RR-7506，Inria – Centre Paris-Rocquencourt ；INRIA，2011 年 1 月。

[32] Lixia Zhang、Alexander Afanasyev、Jeffrey Burke、Van Jacobson、kc claffy、Patric Crowley、Christos Papadopoulos、Lan Wang 和 Bechuan Zhang。命名数据网络。 ACM 计算机通信评论，2014 年 6 月。

[33] 朱振凯和亚历山大·阿法纳西耶夫。 Let's chronosync：命名数据网络中的分散式数据集状态同步。 In Proceedings - International Conference on Network Protocols，ICNP，pages 1–10，10 2013.

[13] https://github.com/orbitdb/ipfs-log

[14] https://github.com/orbitdb/orbit-db

[15] https://github.com/orbitdb/orbit

[16] https://github.com/peer-base/peer-pad

[17] 纺织品.照片