用于探讨稀疏连接、部分复制的对等系统中一致性和原子性保证的框架

Sreeja S. Nair
DittoLive Inc.
法国

Nicholas E. Marino*
DittoLive Inc. 美国

Nick Pascucci
DittoLive Inc.
英国

Russell Brown*
DittoLive Inc. 英国

Arthur P. R. Silva
DittoLive Inc. 瑞典

Tim Cummings
DittoLive Inc. 美国

Connor M. Power
DittoLive Inc. 美国

arXiv:2603.03899v1 [cs.DC] 4 Mar 2026

摘要

要使离线优先协作应用以真正的对等方式运行，其协作功能必须在互联网连接受限或不可用的环境中仍能正常工作。每个对等节点可能只关注与其工作负载相关的应用数据子集，并且该子集可以以不同方式与其他对等节点的子集重叠。访问控制和网状网络技术所施加的限制常常导致对等节点处于稀疏连接状态。在这些系统中对一致性进行推理十分困难，尤其是在考虑可能于同一事务中更改不同数据集合的事务性更新时。我们提出交集原子性和交集因果一致性作为模型，用以推理那些稀疏连接、并依赖于对更广泛数据集合中不同子集进行部分复制的离线优先协作应用。随后，我们利用这些模型提出一组指导原则，帮助开发者设计具备原子性和一致性保证的应用程序。

1 引言

历史上，合并与同步协作数据变更一直是一个非平凡问题。Google Docs[5] 和 Apache Wave[17] 等流行应用程序所使用的操作转换算法[6] 需要中心化服务器来协调变更。因此，这些系统仅提供有限的离线能力。用户可以在离线时继续编辑文档，但必须重新连接到服务器，才能重新同步并将本地数据与其他用户的变更进行合并。

近年来，一种称为 无冲突复制数据类型 （CRDT）的专用数据类型 [12] 越来越受欢迎，它能够让对等节点之间合并数据更改，而不一定需要借助中心化服务。不过，尽管理论前景广阔，许多现实世界中的 CRDT 实现仍然需要中心化服务来合并和同步更改。常见的例子包括 Riak[7]、Redis[9], 和 Cosmos DB[13] 等流行数据库。这种尽管存在潜力却仍需某种集中协调的倾向，部分原因可能是较早期 CRDT 类型的局限性。Shapiro 等人 [12], 所述的实现中，基于状态的无冲突复制数据类型需要通过网络发送大量数据；而基于操作的无冲突复制数据类型则需要网络提供难以保证的条件，这在连接稀疏或间歇中断的系统中尤其如此。两者都不适用于采用对等网状技术（例如蓝牙）的应用程序，因为这类技术的带宽相对较低。尽管诸如 Δ-CRDTs[15] 这样的新技术能够克服这些局限，但在构建完全对等的同步功能时，我们仍面临挑战：如何理解能够提供哪些类型的一致性保证。对于要求支持作用于多个应用数据子集的事务性更新的应用程序而言，这一点尤为突出。

尽管近几十年来移动网络连接不断改善，许多现实应用仍可受益于这样一种能力：无需访问远程服务器，即可在用户之间直接同步数据。蜂窝互联网并非总是可靠或普及——即便是通常被认为更可靠、覆盖更广的 Wi-Fi 网络，在仓库或制造工厂等大型设施中也存在信号盲区。此外，现代移动设备通常早已具备通过蓝牙或 Wi-Fi Direct[2]，彼此直接通信的能力，因此，部署这类功能并不受可用硬件的限制。真正的挑战在于软件技术的发展现状，以及缺乏用于推理点对点复制的模型。通过解决部分软件方面的难题，我们希望推动开发更多离线优先的点对点应用，从而帮助消除集中式客户端-服务器架构带来的局限。

*这项工作是在 DittoLive Inc. 任职期间完成的。

本文首先阐述此类离线优先应用面临的一些典型挑战（§ 2）。随后，我们定义了数据同步系统的系统模型（§ 3）：系统中的设备具有明确的兴趣集，旨在支持设备稀疏连接的网状网络上的点对点协作应用。针对这一系统模型，我们研究实现传统原子性和一致性保证所涉及的权衡（§ 4）。我们提出交集原子性（IntersectionAtomicity）和交集 CC（IntersectionCC）两种模型，用于推理这样一种点对点系统：对等节点订阅受限的数据集；我们还讨论了保证这些模型成立的条件（第 5 节）。

2 问题空间

设想一个供飞机维修机组使用的应用程序。机组人员在机库内不同地点工作，而这些地点可能无法连接网络。因此，机组主管与机组成员必须相互协调、协同工作。

在这个示例场景中，Alice 和 Bob 是飞机维修人员，Alice 负责维护飞机外部涂装，Bob 则负责起落架。Sarah 是机组主管。尽管这三名维修人员都拥有移动设备，但每位人员执行特定工作所需的数据可能因其角色而异。

例如，当一架飞机抵达进行维护时，必须用该飞机所需执行的全部任务来更新 Sarah 的设备。当 Alice 和 Bob 开始轮班时，他们的设备会与 Sarah 的设备或固定工作站同步；得益于持久有线连接，固定工作站始终拥有 Alice 和 Bob 完成本班次所分配任务所需的全部数据。Alice 将订阅涂装任务，Bob 将订阅与起落架相关的任务。请注意，如果某项任务需要对起落架进行涂装工作，该任务将同时出现在 双方 Alice 和 Bob 的设备上。Alice 和 Bob 可以各自独立处理各自的任务，更新将在他们、Sarah 与固定工作站之间进行机会性复制。

离线优先协作应用需要一种复制协议，类似于示例场景中所描述的应用，这种协议旨在确保每台设备所订阅数据的完整性。开发者需要一个保证模型，以帮助他们推理不同设备之间的数据完整性。现有文献表明，在保持高可用性的同时所能达到的最高一致性级别是事务性因果+一致性（TCC+）[1]。TCC+ 包含原子性、因果一致性和强收敛保证。原子性确保应用程序具有全有或全无 语义，因果一致性帮助开发者维护发生先于关系，而收敛则确保同一数据的不同副本收敛到相同的状态。

为了演示这三项属性保证，让我们重新审视上一示例中的场景。设想维修人员之一 Bob 被分配了一项维护检查清单任务：更换起落架中的一颗螺栓。要完成此任务，Bob 必须先从库存中领取一颗新螺栓，然后再进行更换。在该应用程序的上下文中，此事务将表示如下：

```
事务 replace_bolt { inventory.bolts.new -= 1;
landing_gear.bolt.replaced_on = '2024-02-16';
inventory.bolts.old += 1; checklist.landing_
gear.bolt.health = true; }
```

与任何订阅该事务的其他已连接对等节点一样，机组主管 Sarah 所属的设备必须将 Bob 的任务视为一个完整、不可分割的整体。如果没有这种原子性保证，对等节点数据就可能出现差异。

现在考虑另一位维修人员 Alice：她的任务是在 Bob 更换螺栓后为其涂漆。要完成这项任务，Alice 首先需要确认新螺栓已安装到位。确认后，Alice 就可以从库存中取出白色油漆，然后为螺栓涂漆。在应用程序中，这个事务将表示如下：

```
事务 paint_bolt { inventory.paint.white -= 1;
landing_gear.bolt.painted_on = '2024-02-16';
checklist.landing_gear.bolt.paint = true; }
```

完成后，Sarah 收到更新，得知 Alice 已经完成了螺栓的喷漆。然而，如果没有因果性保证，Sarah 可能会误以为 Alice 喷漆的是旧螺栓，因为 Bob 的任务发生在因果过去中更早的时间，而 Sarah 不会收到该更新。

我们再来看另一种情形：作为起落架检查流程的一部分，Bob 注意到起落架管的漆面上有一道小裂纹，于是给它打上补丁。将这一操作表示为一个事务，如下所示：

```
事务 paint_管 { inventory.paint.white -= 2; 起落架.
tube.painted_on = '2024-02-16';
checklist.landing_gear.tube.paint = true; }
```

借助收敛性保证，无论设备观察到的更新顺序如何，最终都会收敛到相同的状态。例如，如果一天开始时库存显示有 20 升白色油漆，订阅的对等节点将按如下方式查看白色油漆的扣减和库存状态：

```
inventory.paint.white = 17;
```

现在让我们介绍 David，他是团队的维修检查清单主管。由于 David 只关注（并订阅）与维修检查清单相关的更新，Alice 和 Bob 仅向 David 复制相关信息。接收信息后，David 的设备观察到的事务如下：

事务 replace_螺栓 { checklist.landing_gear.bolt.health = true; } 事务 paint_螺栓 { checklist.landing_gear.bolt.paint = true; } 事务 paint_管道 { checklist.landing_gear.tube.paint = true; }

尽管只能获得有关这些事务的部分信息，David 仍掌握了与其他设备进行一致且安全交互所需的全部信息。传统的原子性和一致性保证不足以用于推理此类示例中的应用程序的部分复制；我们必须确保即使某个事务的更新只有子集被复制到给定设备上，也能维护更新的正确因果顺序，并且 这些子集仍必须以原子方式应用。

3 系统模型

节点 是具有某种连接性、本地存储和处理能力的物理设备。节点可以使用任何可用的技术/协议连接到一个或多个相邻节点。这些连接可以是直接连接或路由连接，唯一的约束是节点应能够以点对点方式进行通信。当两个节点连接时，我们称它们为对等节点，因为它们之间不存在层级关系（不同于具有客户端/服务器、主/从等关系的节点）。

节点可能分布在某个地理区域内，也可能具有移动性；也就是说，它们可能随时改变位置。这种移动性可能会影响节点的连接性，使此前已连接的节点变得无法访问，或与此前未连接的节点建立新的连接。节点在移动过程中或移动之后也可能仍未连接。因此，与目标节点建立连接并非总能得到保证。

节点可以选择订阅与其工作流相关的数据。一个节点各项订阅的并集称为其订阅集。它可能包含整个数据集，但更常见的是仅包含其中的一个子集。节点会在本地存储其所订阅的数据，从而即使没有连接性也能不间断地工作（例如对数据进行变更）。这使得该应用程序成为离线优先：它可以修改本地可用的任何数据，可以插入新数据并将其存储在节点本地，并且可以将这些数据复制到其他对其感兴趣的节点。

每个节点都有一个 状态，由一组对象组成。该状态可以通过 操作 进行更新。多个操作可以组合在一起，通过事务以原子方式应用。源自某一给定节点的事务，在该节点的上下文中可称为本地事务。节点通过复制接收到的任何事务可称为远程事务。本地事务与远程事务都会更新给定节点的状态。

节点连接到对等节点时，双方会先交换各自的订阅声明（例如“所有起落架数据”），然后才交换数据。随后，对等节点会交换对方感兴趣的数据，确保在对等节点之间传输的所有相关数据都符合各自的订阅集。在向已连接对等节点发送数据时，我们假设数据的发送和接收是原子的。这一特性——称为边缘同步[8]——使应用程序即使在没有互联网连接的情况下，也能实现协作功能。

为了提高这一同步系统的效率，对等节点还可以交换数据摘要以及各自的订阅信息。该摘要可能会利用版本向量[10]等因果追踪机制，来检测对方是否已经拥有某个对等节点的部分或全部数据。接收此信息的对等节点会计算出需要发送给另一对等节点的差异，以使其数据保持最新。这种摘要交换纯粹是一种优化。请注意，任意两个对等节点只有在各自订阅集的交集范围内才能实现数据同步，因为每个对等节点只拥有符合自身订阅集的数据。

节点上的数据可能是该节点所订阅数据的超集，因为其中可能包含该节点未向其他对等节点请求的数据。在这种情况下，我们假设应用开发者并不期望对其订阅集之外的数据提供任何保证，因此不认为这部分数据需要满足任何保证。远程对等节点可能对本地节点持有、但本地节点自身并未订阅的数据感兴趣。远程对等节点会在连接时通过传达其 订阅集 来表达这一点。在这种情况下，我们仍会在订阅该数据的对等节点的语境下，讨论该数据所满足的保证。总而言之，只有当节点上的数据属于其自身的订阅集时，我们才考虑与该数据相关的保证。

在许多应用程序中，不同节点具有不同的角色，因此其访问权限可能仅限于应用数据的某些子集。例如，考虑这样一种场景：维修人员的健康信息也由同一个应用程序管理。他们的主管 Sarah 在紧急情况下可能有权访问这些信息，但 Alice 和 Bob 作为同事，不应能够访问彼此的健康信息。出于安全和保密原因，必须遵守这些限制。我们将节点获准访问的数据集合称为其权限。

就我们的目的而言，权限可以与任何给定节点的订阅相结合。我们将兴趣集 定义为节点订阅与权限的交集。这样可以确保仅当数据同时符合其订阅和 权限时，才会将其发送给对等节点。

节点可能会更改其兴趣集（权限和/或订阅）。它也可以选择清除本地存储的所有数据并重新开始。在这两种情况下，我们都认为当前节点离开系统，并由一个采用新配置的新节点加入。

4 使用边缘同步系统的应用程序的权衡

离线优先协作应用的复制协议设计涉及三个关键维度：

- 每个节点各自的兴趣集，即其订阅与权限的交集。
- 节点互联所依赖的网络拓扑与连接性。
- 应用程序必须维持的一组保证。

让我们详细考察这些维度。

兴趣集： 如第 3 节所述，离线优先的点对点应用并不会在分布式对等节点之间同步所有事务数据，而是只同步符合各自兴趣集的数据。兴趣集大小的下界由应用程序为实现流畅的离线优先操作与协作所需的数据决定。相反，其上界则受节点权限集的限制。

网络拓扑与连接性： 根据应用程序的类型，开发者可能能够影响网络拓扑，也可能无法做到。在光谱的一端，开发者可以灵活选择节点之间的静态连接。此外，网络层可能配备路由算法，以确保特定对等节点能够连接。与此形成鲜明对比的是，在光谱的另一端，开发者无法控制对等节点之间的连接性，因此也无法控制网络拓扑。在这种情况下，当对等节点彼此距离较近时，它们会利用兼容的通信协议进行机会性连接。

与节点兴趣集相匹配的所有更新最终都应通过其对等节点传达到该节点。网络层必须确保每个节点最终都能连接到一个或多个对等节点，而这些对等节点包含该节点的整个兴趣集的超集。此外，网络层应避免形成数据孤岛。数据孤岛 是指一个由对等节点构成的连通子图，其中的节点无法连接到其他具有重叠兴趣集的对等节点。

图 1 展示了多个对等节点组之间形成的数据孤岛。左上方的组包含三个节点，它们的兴趣集都能由其连接的对等节点满足。由于该组中每个节点的兴趣集

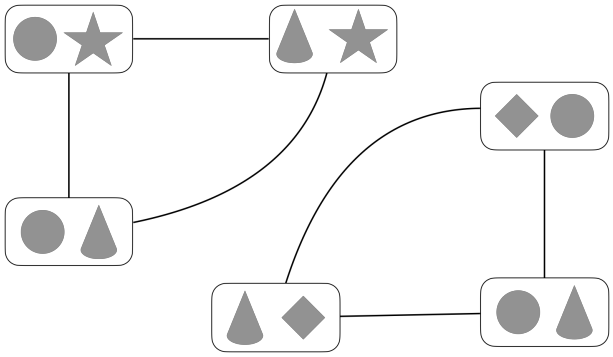


图 1. 由多个组件构成的对等网络图，其中兴趣集 均得到满足，但仍然存在网络分区。对等节点以矩形表示。对等节点之间的连接用箭头表示。不同形状代表不同的兴趣集。

都得到覆盖，每个节点都认为自己接收到了其兴趣集中的所有更新。然而，另一个组也拥有与该组节点兴趣集匹配的数据。由于这些组之间的连接不足，它们形成了数据孤岛。请注意，单个节点不会导致数据孤岛，因为它可以推断出自己需要从某处接收数据。系统中的节点数量及其兴趣集都是动态变化的，这给检测这些孤岛带来了挑战。

应用程序保证： 如§ 2所述，协作应用程序受益于原子性、因果一致性和收敛保证。

遵循 Burckhardt 等人 [4]、Viotti 和 Vukolić [16]的做法，我们将给定节点上的执行建模为一种抽象执行

$$A = (H, vis, ar)$$

其中 H 是构成该节点历史 的一组操作，可见性关系 vis 与仲裁 ar共同定义了这些操作之间的关系。考虑两个操作o_1 和 o_2。我们用 $o_1 \xrightarrow{vis} o_2$ 表示 o_1 对 o_2可见。 ar 是历史上操作的全序，用于描述系统如何解决并发操作导致的冲突。节点执行操作的顺序称为程序顺序。先发生于 关系（hb）是可见性与程序顺序之并集的传递闭包。

如果在某个时间点，操作 o 被记录为在节点 N 可见于 $(\rightsquigarrow)$，那么对于之后发生的任何操作 o' $o \xrightarrow{vis} o'$，均如此。

事务将必须一起应用的操作组合在一起。通俗地说，当任何给定节点上的任何事务要么被原子地应用，要么完全不被应用时，就满足原子性保证；也就是说，这是一种全有或全无的保证。形式上，事务 T 的原子性可以

定义为

$$Atomicity(T) \triangleq \forall N.\forall o_1, o_2 \in T.o_1 \rightsquigarrow N \implies o_2 \rightsquigarrow N \quad (1)$$

换句话说，如果操作 o_1 和 o_2 属于事务 T，并且 o_1 在系统中给定节点 N 上可见，那么 o_2 在同一节点上也可见。

因果一致性保留更新之间的先发生关系，而强收敛确保拥有相同更新集合的节点收敛到相同状态。形式上，因果一致性可以定义为

$$CC \triangleq \forall N.\forall o_1, o_2 \in O.o_1 \xrightarrow{\text{hb}} o_2 \wedge o_2 \rightsquigarrow N \implies o_1 \rightsquigarrow N \quad (2)$$

换句话说，如果两个操作按先发生关系排序，$o_1 \xrightarrow{\text{hb}} o_2$，并且如果 o_2 在给定节点 N 上可见，那么 o_1 也应在同一节点上可见。

强收敛属性指出，观察到相同更新集合的两个节点会收敛到相同状态[16]。实际上，使用无冲突复制数据类型[12]可以轻松实现强收敛。当节点接收到远程对等节点的状态时，它会将传入状态合并到本地状态中。如果合并函数具有幂等性、结合性和交换性，并且是两个状态的最小上界，就能保证强收敛[12]。

因果+ 一致性结合了因果一致性和强收敛[16]。事务因果+ 一致性（TCC+）结合了原子性和因果+一致性。

4.1 应用保证与兴趣集

如前所述，节点的兴趣集取决于其在应用程序中的角色（包括其订阅的数据以及节点的权限）。下面讨论兴趣集的定义如何影响这些保证。

节点连接到对等节点时，双方兴趣集的组合可分为以下几种情况：

1. 远程端和本地兴趣集不相交 2. 远程端的兴趣集与本地兴趣集相同 3. 远程端的兴趣集是本地兴趣集的超集 4. 远程端的兴趣集是本地兴趣集的子集 5. 远程端的兴趣集与本地兴趣集重叠

对于条目 1，没有需要复制的内容，因此会话可以关闭。对于条目 2，本地和远程对等节点对相同的数据感兴趣，并且本地节点所具备的所有保证在远程端也都得到维持。其余情况最值得关注。在条目 3 中，对于仅包含与本地兴趣集匹配的对象的所有事务，原子性都得以维持。假设从本地和远程来源收到的所有更新都捆绑在单个原子复制更新中，因果一致性也得以保留。在条目 4 中，原子性和一致性得以维持，因为所有内容都是本地内容的子集。在条目 5 中，对于仅包含与本地兴趣集的交集相匹配的对象的所有事务，原子性和因果一致性都得以维持。

对于不符合这些条件的其余数据，可以确保最终一致性和收敛[1]。

请注意，强收敛要求观察到相同更新操作集的两个节点最终达到相同状态。由于我们要求与节点兴趣集相关的所有更新最终都传递到该节点，因此使用无冲突复制数据类型可轻松满足数据部分视图的强收敛保证。

由于每个节点所持数据的部分视图这一固有特性，我们认为有必要对系统中节点能够满足的保证施加一些限制。在第 5 节中，我们将定义每个节点必须基于此部分视图在本地遵守的保证，从而使整个系统满足事务因果一致性（TCC+）。

4.2 应用程序保证与网络拓扑

让我们讨论网络拓扑与保证之间的关系，以处理开发者可能定义网络拓扑的情况。

考虑一种森林状网络拓扑，其中兴趣集最小的边缘节点位于叶节点，而兴趣集随着向根节点靠近而逐渐扩大，最终到达订阅所有内容的云。在这种层次模型中，源自叶节点的事务会复制到规模更大的节点，因此可以轻易地保持其原子性和因果性。源自根节点的事务在向层次结构中较低的节点传输时，可能会被拆分成若干部分。尽管如此，由于较低层节点对事务中其余的操作不感兴趣，因此可以认为整个系统维持了 TCC+ 保证。

当两个节点之间的路径上存在兴趣集更小的另一个节点（或一组节点）时，就会出现这种问题场景。考虑三个节点 N_1、N_2 和 N_3。假设 N_1 和 N_3 的兴趣集为 $s_1 \cup s_2$，并通过兴趣集为 s_2 的 N_2 相连。如果一个源自 N_1 的事务同时变更了 s_1 和 s_2 中的数据，并将该事务复制到 N_2，那么事务将被缩减为仅包含对 s_2 的变更。此时，当 N_2 与 N_3 进行复制时，N_3 将只能获得部分知识，从而破坏原子性保证。同样的推理也适用于一致性保证。

5 离线优先协作应用程序设计的保证

如第 4.1 小节所述，对等同步会话必须满足特定条件，才能维持原子性和因果一致性保证。我们将这些条件形式化为交集原子性（IntersectionAtomicity）和交集因果一致性（IntersectionCC）。简而言之，如果接收节点能够针对其兴趣集中的数据更新，保留给定

[1]前提是网络能够确保数据最终到达感兴趣的节点。

	LSet ⊂ RSet	LSet ⊃ RSet	LSet =RSet	LSet ∩ RSet ≠ ∅
本地事务的交集原子性	✓	✓	✓	LSet ∩RSet
远程事务的交集原子性	LSet	✓	✓	LSet ∩RSet
单个对象的交集CC	✓	✓	✓	✓
多个对象的交集CC	LSet	✓	✓	LSet ∩RSet
收敛	✓	✓	✓	✓

表 1。 与对等节点兴趣集相比，交集原子性和交集因果一致性保证。每个单元格包含一个谓词，用于表示满足这些保证的数据集。 ✓表示恒为真的谓词。

事务的原子性，则该接收节点就保留了交集原子性。换言之，如果某个节点订阅了事务中所涉及的数据，那么当它观察到该事务中的一项更新时，也会观察到同一事务中的所有其他更新，这样系统就保留了该事务的交集原子性。同样，如果接收节点能够保留其兴趣集中的数据更新的因果一致性，则它就保留了交集因果一致性。开发者在应用程序中理解并运用这些概念将会受益：交集原子性使基于订阅的同步能够提供事务保证，而交集因果一致性则确保系统在各节点之间维持一致性。

请注意，当数据类型本身具有收敛性时（例如无冲突复制数据类型），收敛性会得到平凡的保持。这是因为操作充当更新的原子单元，在给定时间只能修改一个数据对象。

我们将给定节点 N 的兴趣集记为 S_N。如果某个操作修改了属于兴趣集 S_N 的数据，我们记作 $o \between S_N$；否则记作 $o \barwedge S_N$。

根据我们采用的 Viotti 和 Vukolić [16], 的形式化方法（并在第 4 节中进行了讨论），交集原子性和交集因果一致性可定义如下:

如果系统中的任意节点 N 都满足以下条件，则称事务 T 维护交集原子性

$$\textsc{IntersectionAtomicity}\ (T) \triangleq \forall o_1, o_2 \in T.o_1 \rightsquigarrow N \implies o_2 \rightsquigarrow N \vee o_2 \ominus N \tag{3}$$

如果某个更新在节点上可见，则该更新的整个、且与该节点兴趣集匹配的因果历史 均可见；此时称某个操作维护交集因果一致性。

$$\textsc{IntersectionCC} \triangleq \forall o_1, o_2 \in O.o_1 \xrightarrow{\text{hb}} o_2 \wedge o_2 \rightsquigarrow N \implies o_1 \rightsquigarrow N \vee o_1 \ominus N \tag{4}$$

现在让我们考察在对等同步会话中如何维持这些保证。我们将发送更新的本地对等节点的兴趣集称为 LSet，将接收更新的远程对等节点的兴趣集称为 RSet。借助无冲突复制数据类型，可以轻松维持单个对象的交集CC（也称为相干性）和收敛。当 LSet 是 RSet 的超集或与之相等时，本地和远程事务的交集原子性，以及跨多个对象的交集CC都能得到维持。请注意，这是因为发送节点掌握的信息比接收节点更多，因此接收节点能收到其所需的全部信息。当发送对等节点可能不掌握完整信息时，也就是说，当 LSet⊂ RSet 且 LSet ∩ RSet ≠ ∅ 时，这些保证仅适用于属于两个节点兴趣集交集的数据。表 1 总结了在对等节点兴趣集之间不同关系下，各项保证得以维持的条件。

如果系统中所有对等节点间连接都维持交集原子性和交集因果一致性保证，并配合使用无冲突复制数据类型，则可保证整个系统始终维持原子性和因果+一致性。这是因为这些条件确保每个节点所关注的数据都维持原子性和因果一致性，从而确保系统中的任何操作都能在全局范围内维持 TCC+ 保证。

5.1 维护这些保证

如果所有节点都订阅全部数据，交集原子性和交集因果一致性保证便可平凡地确保系统中的 TCC+ 保证。由于我们的系统模型受到限制，我们必须确保这些保证始终由各个节点维持。

如果我们能够灵活选择网络的拓扑，那么当节点构成分层网络、底部订阅较小且越往顶部订阅越大时，便可确保系统中的 TCC+ 。将拓扑限制为森林状架构、以云节点为根节点并以边缘节点为叶节点的分层复制模型[14]遵循这一模型。

开发者也可以选择配置系统中各节点的兴趣集，以确保 TCC+保证。在这种情况下，兴趣集的配置应确保事务及其全部因果历史不会落在系统中任何一对兴趣集的交集之外。例如，考虑一个包含以下事务的系统

- T_1 修改集合 s_1，并在因果上依赖集合 s_2,
- T_2 修改集合 s_3，并在因果上依赖同一集合，且
- T_3 变更集合 s_3 和 s_4，并在因果上依赖于 s_4。

节点的兴趣集可以是 $s_1 \cup s_2$、 $s_3 \cup s_4$，也可以是它们的任意超集。

另一种方法是在所有节点上复制全部元数据，并在实际数据属于某个节点的兴趣集时复制这些数据。[2]

6 相关工作

由于涉及的权衡考量十分复杂，推理分布式系统是一项挑战。Viotti 和 Vukolić [16] 提出了非事务一致性模型分类，扩展了 Burckhardt 等人 [4] 为定义最终一致性模型所提出的基本机制。然而，这两项工作都假设系统中的节点存储了完整的数据副本。此外，尽管 Shapiro 等人 [11] 提出了一个用于推理一致性保证的框架，但他们假设兴趣集彼此不相交。

Akkoorath 等人 [1] 提出了一种提供 TCC+的复制协议。该协议针对云环境，其中数据中心配备了多台物理机，负责对数据进行分片和存储。Toumlilt 等人 [14] 将这些保证扩展到了边缘，但要求节点采用森林状拓扑。Belaramani 等人[3] 也通过发送经过优化的元数据，解决在边缘提供因果一致性的问题。两个系统都采用了我们讨论过的权衡方式：前者通过调整网络拓扑，后者则以某种方式调整“兴趣集”。

7 结论

本文介绍了两种模型，即交集原子性和交集因果一致性，用于分析对等节点仅订阅受限数据集的点对点系统。我们表明，如果每个对等节点都保持这些模型所提供的保证，则整个系统就能维持事务因果+ 一致性，这是在无需协调的高可用分布式系统中所能期望的最高保证。我们还列出了任意给定对等节点要维持这些保证所必须满足的条件。这些模型及其所需条件使开发者能够分析稀疏连接、部分复制的点对点系统中的一致性与原子性保证。

致谢

作者感谢 Toni Kemp 提供的编辑支持，以及 Matheus Cardoso 带来的所有有趣讨论。

参考文献

[1] Deepthi Devaki Akkoorath、Alejandro Z. Tomsic、Manuel Bravo、Zhongmiao Li、TylerCrain、AnnetteBieniusa、NunoPreguiça 和 Marc Shapiro。2016年。Cure：强语义兼具高可用性与低延迟。载于《2016年 IEEE 第36届国际分布式计算系统会议（ICDCS）》。IEEE，皮斯卡塔韦，美国新泽西州，405–414。doi:10.1109/ICDCS.2016.98[2] Wi-Fi 联盟。2024年。Wi-Fi Direct。https://www.wi-fi.org/discover-wi-fi/wi-fi-direct [在线；访问日期：2月9日-2024]。[3] Nalini Belaramani、Mike Dahlin、Lei Gao、Amol Nayate、Arun Venkataramani、Praveen Yalagandula 和 Jiandan Zheng。2006年。PRACTI 复制。载于《第3届网络系统设计与实现会议论文集——第3卷》（加利福尼亚州圣何塞）（NSDI’06）。USENIX 协会，美国，5。[4] Sebastian Burckhardt 等。2014年。最终一致性原理。《编程语言基础与趋势》1，1-2（2014年），1–150。[5] John Day-Richter。2010年。新版 Google Docs 有何不同：冲突解决。https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_22.html [在线；访问日期：2月14日-2024]。[6] C. A. Ellis 和 S. J. Gibbs。1989年。群件系统中的并发控制。SIGMOD Rec. 18，2（1989年6月），399–407。doi:10.1145/66926.66963[7] Basho Technologies Inc. 2017年。Riak 文档——数据类型。https: //docs.riak.com/riak/kv/2.2.3/learn/concepts/crdts/index.html [在线；访问日期：2月14日-2024]。[8] DittoLive Inc. 2024年。Ditto——无论是否联网，都能同步应用。https://ditto.live/ [在线；访问日期：2月9日-2024]。[9] Redis Ltd. 2024年。深入解析：Redis CRDT。https://redis.com/docs/under-the-hood/ [在线；访问日期：2月14日-2024]。[10] D. S. Parker、G. J. Popek、G. Rudisin、A. Stoughton、B. J. Walker、E. Walton、J. M. Chow、D. Edwards、S. Kiser 和 C. Kline。1983年。分布式系统中相互不一致的检测。《IEEE 软件工程汇刊》9，3（1983年5月），240–247。doi:10.1109/TSE.1983.236733[11] Marc Shapiro、Karthikeyan Bhargavan、Yek Chong 和 Youssef Hamadi。2004年。简短公告：一致性与部分复制的形式化体系。载于 DISC 2004——第18届国际分布式计算会议，第3274卷。[12] Marc Shapiro、Nuno Preguiça、Carlos Baquero 和 Marek Zawirski。2011年。无冲突复制数据类型。载于《分布式系统的稳定性、安全性与保障：第13届国际研讨会，SSS 2011，法国格勒诺布尔，2011年10月10日至12日。论文集》第13卷。施普林格，386–400。[13] Dharma Shukla。2018年。Azure Cosmos DB：推动全球分布式数据库的前沿。https://azure.microsoft.com/en-us/blog/azure-cosmos-db-pushing-the-frontier-of-globally-distributed-databases/ [在线；访问日期：2月14日-2024]。[14] Ilyas Toumlilt、Pierre Sutra 和 Marc Shapiro。2021年。在边缘借助 colony 实现高可用、一致的群组协作。载于《第22届国际中间件会议论文集》（加拿大魁北克市）（Middleware ’21）。计算机协会，纽约，美国纽约州，336–351。doi:10.1145/3464298.3493405[15] Albert van der Linde、João Leitão 和 Nuno Preguiça。2016年。 Δ-CRDT：使 δ-CRDT 基于增量。载于《分布式数据一致性的原理与实践研讨会第2届会议论文集》（英国伦敦）（PaPoC ’16）。计算机协会，纽约，美国纽约州，文章 12, 4 10 1145 2911151 2911163 页码。[16] doi: . / . Paolo Viotti 和 Marko Vukolić。2016. . 非事务型分布式存储系统中的一致性。《ACM 计算调查》49，1，文章 19 2016 34 10 1145 2926965 （6月），页码。doi: . / [17] David Wang 、 Alex Mah 和 Soren Lassen。2010. . Google Wave 操作转换。https://svn.apache.org/repos/asf/incubator/wave/whitepapers/operational-transform/operational-transform.html [在线；访问日期：2月14日-2024]。

[2]处理这种情况的优化解决方案已在文献中提出[3]。