软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(5):2392−2412 [doi: 10.13328/j.cnki.jos.006460]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

分布式数据库多级一致性统一建模理论研究*

水治禹[1,2], 卢 卫[1,2], 赵展浩[1,2], 何粤阳[1,2], 张 孝[1,2], 杜小勇[1,2]

[1](数据工程与知识工程教育部重点实验室 (中国人民大学), 北京 100872)

[2](中国人民大学 信息学院, 北京 100872)

通信作者: 张孝, E-mail: zhangxiao@ruc.edu.cn; 杜小勇, E-mail: duyong@ruc.edu.cn

摘 要: 分布式数据库系统出现了支持多协调器和多副本存储的新架构, 这给事务调度的正确性带来了新的挑战, 包括缺少中心协调器带来的新数据异常以及多副本机制带来的读取数据一致性等问题. 基于事务隔离级别和分布式系统一致性协议的定义, 为多协调器多副本分布式数据库的事务多级一致性构建了一个混合依赖图模型. 该形式化模型为事务的正确调度提供具有鲁棒性的评价标准, 可以方便地对数据库事务调度情况进行动态或静态分析检验.

关键词: 分布式数据库; 一致性; 隔离级别; 混合依赖图

中图法分类号: TP311

中文引用格式: 水治禹, 卢卫, 赵展浩, 何粤阳, 张孝, 杜小勇. 分布式数据库多级一致性统一建模理论研究. 软件学报, 2023, 34(5): 2392–2412. http://www.jos.org.cn/1000-9825/6460.htm

英文引用格式: Shui ZY, Lu W, Zhao ZH, He YY, Zhang X, Du XY. Theoretical Study on Multi-level Consistency Modeling in Distributed Databases. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2392–2412 (in Chinese). http://www.jos.org.cn/1000-9825/6460.htm

Theoretical Study on Multi-level Consistency Modeling in Distributed Databases

SHUI Zhi-Yu[1,2], LU Wei[1,2], ZHAO Zhan-Hao[1,2], HE Yue-Yang[1,2], ZHANG Xiao[1,2], DU Xiao-Yong[1,2]

[1](Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

Abstract: A new architecture that supports multiple coordinators and multi-replica storage has emerged in distributed database systems, which brings new challenges to the correctness of transaction scheduling. The challenges are represented by new data anomalies caused by the lack of a central coordinator and data inconsistency caused by the multi-replica mechanism. Based on the definition of transaction isolation levels and consistency protocols for distributed systems, this study constructs a unified hybrid dependency graph model for transactional multi-level consistency in multi-coordinator and multi-replica distributed databases. The model provides a robust standard for evaluating the correctness of transaction scheduling, which can facilitate dynamic or static analysis of transaction scheduling in databases.

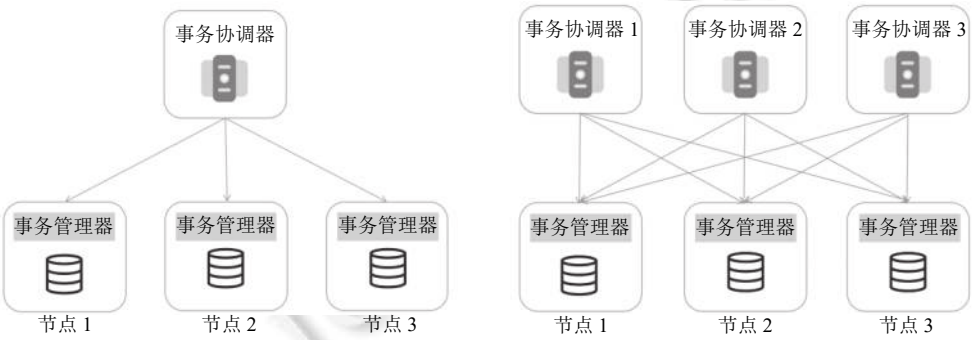
Key words: distributed database; consistency; isolation levels; hybrid dependency graph

1 引 言

大数据具有容量大、变化快的特征, 这些特征要求分布式数据库具有高可扩展性; 同时, 节点规模的增加会导致系统出现节点故障的概率提升, 因此分布式数据库还应当具备高可用性, 即在出现节点故障时系统仍然能够对外提供服务[1]. 为了实现这两个目标, 从数据一致性的角度来看, 分布式数据库在系统架构上有两点新的需求[2].

* 基金项目: 国家自然科学基金 (61972403,61732014); 中央高校基本科研业务费专项资金 (20XNLG22); 中国人民大学-腾讯联合实验室联合项目基金
本文由“数据库系统新型技术”专题特约编辑李国良教授、于戈教授、杨俊教授和范举教授推荐.
收稿时间: 2021-07-01; 修改时间: 2021-07-31; 采用时间: 2021-09-13; jos 在线出版时间: 2022-10-14
CNKI 网络首发时间: 2022-11-15

● 需求 1. 事务处理要能支持多协调器. 传统的分布式数据库采用单事务协调器系统架构 (如图 1(a) 所示). 事务协调器接收查询执行引擎分发的事务处理请求, 逻辑上, 协调器把该事务拆成若干个子事务, 并分发每个子事务到相应的存储节点 (每个存储节点都是独立的数据库实例, 具有独立的事务管理器, 分配给存储节点的子事务由涉及该存储节点数据的操作组成), 由存储节点中事务管理器独立完成子事务的处理, 最后, 由协调器通过 2PC 协议协调各个存储节点完成事务的提交或回滚. 在该架构下, 所有的事务处理都经由单事务协调器调度, 协调器容易成为性能瓶颈, 系统可扩展性弱; 此外, 一旦该协调器发生故障, 整个系统的可用性会受到影响, 可用性差. 为了消除单协调器的性能瓶颈, 谷歌于 2010 年提出了 Percolator 系统[3], 该系统采用多协调器架构 (如图 1(b) 所示). 在该架构下, 每个事务协调器单独接收和处理事务的请求, 处理的任务与其在单事务协调器架构下的任务相同. 目前, 谷歌最新一代的分布式数据库系统 Spanner[4]、阿里系的 OceanBase[5]、腾讯的 TDSQL[6], 以及 Spanner 的两个开源实现: TiDB[7]与 CockroachDB[8], 都是采用多事务协调器系统架构.



(a) 单事务协调器系统架构 (b) 多事务协调器系统架构

图 1 单事务协调器与多事务协调器架构

● 需求 2. 存储管理要支持多副本存储. 为了支持存储系统的高可扩展性, 分布式数据库中的数据是分片存储的 (例如水平分片), 每个分片根据一定的策略 (例如基于哈希函数的分片策略), 被分发到不同的存储节点来管理. 这个思想跟目前 NoSQL 存储系统的思想是一致的, 数据量的增加会导致分片的分裂和迁移, 通过迁移热点分片到非瓶颈存储节点, 以提高存储系统的性能. 为了提高系统的可用性, 每个分片会设置若干个数量的副本, 这些副本会根据不同的策略, 存储在不同的节点, 甚至是不同的数据中心. 通过引入数据分片和多副本机制, 即使数据库系统出现部分节点故障、网络故障, 系统仍然具备对外提供数据服务的能力.

分布式数据库在系统架构上的新需求, 给事务处理系统的正确性带来了新的挑战.

● 挑战 1. 在多事务协调器架构下, 由于缺少统一的全局时钟, 而每个协调器又单独为事务分配开始时间戳、提交时间戳等, 由不同协调器发起的事务, 在时钟上不能进行两两比较. 这跟原单事务协调器系统架构下的事务处理具有明显的不同; 在单事务协调器系统架构下, 所有事务均由单一协调器统一分配时钟, 因而任意两个事务的时间戳是可以比较的. 事务处理的正确性理论并没有对这一点进行特别的说明, 并且我们还发现, 在缺少统一全局时钟的情况下, 事务处理仍然会出现新的各类数据异常, 即使是过去我们认为的事务调度的正确性理论, 即事务的可串行化调度.

● 挑战 2. 在多副本机制下, 出于性能上的考虑, 同一数据项在不同副本的更新不要求强同步, 这就导致事务从不同副本读取相同数据项时, 无法保证数据的一致性. 例如, 同一个事务内读取相同数据项的两个操作, 由于这两个读操作分别读取了来自两个副本的相同数据项, 由于副本之间数据可能存在的不一致, 导致出现了“不可重复读”异常.

应对上述两个挑战, 分布式数据库在进行并发控制设计时, 不仅需要考虑原先事务的隔离性问题, 还需要考虑多协调器和多副本引起的数据不一致问题. 前一个问题一直是传统数据库事务处理的重点, 事务并发控制已经发展出了一套比较完善的多级隔离级别理论体系; 而后一个问题, 从 20 世纪 80 年代开始 Herlihy 等人[9]、Lamport[10]就对缺少中心协调器的分布式系统中的数据一致性进行了一系列研究, 形成了线性一致性 (linearizability)[9]、顺

序一致性 (sequential consistency)[10]、因果一致性 (causal consistency)[11]、读己之所写一致性 (read your writes)[12]等多个不同强度的分布式系统一致性. 在此基础上, 面临这两个挑战的新型数据库系统 NewSQL[13], 既具备了分布式一致性的架构, 又需要满足事务 ACID 属性, 迫切需要一套结合事务多级隔离级别和分布式一致性的新型理论解决数据一致性问题.

结合两套理论以解决分布式系统下的一致性问题, 在现有工作中已有部分体现. 例如, SLOG[14]结合线性一致性和可串行化隔离级别, 定义了严格可串行化; Transaction chains[15]探究在可串行化级别上保留一致性模型读己之所写一致性的偏序; Daudjee 等人[16]提出强会话可串行化 (strong session serializability), 结合了会话一致性 (session consistency)[12]与可串行化. 但是, 这些工作都是对隔离级别和一致性模型进行的逐案分析定义, 是将某一隔离级别和一致性模型的某些特性进行结合, 缺乏对两套理论的系统性分析和建模.

一些系统性的工作虽然考虑了隔离级别与一致性的联系, 但其目标及方法与本文不同. 例如, Cerone 等人[17]在代数层面探讨了弱一致性模型的两种不同类型规范之间的联系. 文献 [17] 对数据库系统模型以及一致性级别做了较多约束, 并且其目标在于建立两套体系之间的联系和转化方式. Szekeres 等人[18]提出新的统一框架来定义一致性和隔离保证; Shapiro 等人[19]的工作通过将多级隔离级别和一致性模型建模在 3 个维度上来讨论. 相比之下, 本文工作采用了新的定义方式——混合依赖图和环来进行分析定义, 同时给出了推理过程和方法, 增强可读性的同时也增强了模型的可扩展性. 另外, 本文的工作涵盖了更弱的一致性级别, 且致力于结合两套体系以弥补原有定义的不足.

本文通过深入分析多级隔离级别和一致性模型, 对两套理论体系进行了统一建模, 并基于统一的模型探究两个体系之间结合的可行性与效果. 传统的分布式数据库系统隐含单一协调器的假设, 而事务的隔离级别正是在这样的假设下定义的. 然而, 由于架构调整, 现有分布式数据库由单协调器调整为多协调器、单副本调整为多副本后, 事务的隔离级别定义需要扩展. 本文提出的分布式数据库多级一致性统一建模理论, 既保证了数据库事务隔离性, 同时结合了分布式系统也就是多协调器、多副本环境下的一致性考虑, 为新型分布式数据库系统, 如 NewSQL 提供了一套完善的一致性理论. 同时, 多级一致性模型也可为分布式数据库中事务的正确调度提供具有鲁棒性的评价标准, 可以借助理论方便地对数据库运行中事务调度的情况进行动态或静态分析和检验.

本文第 2 节对数据隔离级别进行了介绍. 第 3 节对分布式一致性理论进行了介绍. 第 4 节对分布式一致性进行建模, 借助抽象依赖图建立了分布式一致性的新型环定义. 第 5 节进行了统一建模并详细阐述了模型效果、实验方法与可视化呈现. 最后, 第 6 节总结, 对未来发展方向进行展望.

2 数据异常与隔离级别

ACID[20]是数据库系统对事务处理正确性的判断准则, 即要求数据库事务具备原子性 (atomicity)、一致性 (consistency)、隔离性 (isolation)、持久性 (durability) 这 4 个特性. 本文聚焦于隔离性.

隔离性的保证依赖于所采用的并发控制技术. 严格的隔离性, 如可串行化, 在实际应用中会造成并发事务之间较多的冲突和阻塞, 从而造成性能上的损失. 为了获得更佳的性能, 隔离性会根据不同程度被定义为不同级别, ANSI/SQL-92[21]基于数据异常来定义事务隔离级别, 将数据库隔离级别划分为 4 个等级, 包括读未提交 (read uncommitted)、读已提交 (read committed)、可重复读 (repeatable read) 和可串行化 (serializable) 这 4 个级别. 随后, Berenson 等人[22]进一步在 SQL-92 标准基础上, 使用形式化的符号对隔离级别进行重新定义, 消除了 SQL-92 隔离级别定义的二义性. 然而, Adya 等人[23]认为文献 [20] 对数据异常的定义过于严格, 并且不适用于后来流行的乐观并发控制 (OCC)[24]和多版本并发控制 (MVCC) 方法. 因此, Adya 等人提出了基于环的数据异常定义方法, 即, 将事务的执行序列构建为数据依赖图, 通过识别数据依赖图中是否存在环来判断异常. 具体地, Adya 的定义根据数据库事务历史构建数据依赖图, 将事务作为顶点、事务之间的数据依赖关系作为有向边, 不同数据异常对应不同结构的环, 不同隔离级别需要禁止依赖图中出现不同结构的环 (后文也称环定义).

2.1 数据依赖图

在引出数据依赖图之前, 先介绍一些基本概念. 本文假设分布式数据库管理系统 (RDBMS) 是基于多版本并发控制 (MVCC) 机制的. 在实际中, 在实际系统中, 主流的开源或商用 RDBMS (例如, Oracle[25], DB2[26], SQL Server[27]) 都基于 MVCC 机制, 因此可以认为假设是合理的. 在基于 MVCC 的 RDBMS 中, 一个数据对象 (也称为数据项) 可以具有多个版本.

● 版本. 使用小写字符 (例如 x) 代表数据项, 用下标代表数据项的版本, 如 x_i 代表数据项 x 的第 i 个版本. 数据项每次更新会产生一个新版本, 为了简化模型中, 将插入和删除也视为一次更新操作.

● 事务 (transaction). 每个事务由确定顺序的若干读、写操作 (operation) 组成, 隐式或显式地以提交 (commit) 或回滚 (abort) 操作结尾. 使用 T 来表示事务, 下标表示事务序号, 比如 T_i 表示第 i 个事务.

● 事务历史 (transaction history). 给定一组事务, 事务历史是该组事务在数据库中实际执行的操作序列, 它反映了不同事务数据操作之间的顺序, 使用 H 来表示. 与大部分研究工作一致, 在本文中, $r_i(x_j)$ 代表事务 T_i 读取了数据项 x 的版本 j, $w_i(x_j)$ 代表事务 T_i 对数据项 x 写了版本 j. 对于一个元组 x 的两个版本 i 和 j, x_i 和 x_j, 若 $i<j$, 则表示 x_i 在 x_j 之前创建.

● 数据依赖 (data dependency). 当不同的事务在同一个数据对象上进行操作且至少有一个操作是写操作时, 事务之间就会出现数据依赖. 数据依赖的类型取决于操作的类型和事务提交的顺序被分为 3 种, 定义如表 1.

表 1 数据依赖关系

数据依赖	描述	符号表示
写写依赖 (write-write/ww dependency)	事务T_i提交了数据项x上的写版本x_m, 之后事务T_j提交了x的下一写版本x_n, $m<n$, 则T_j写写依赖于T_i	$T_i \overset{ww}{\rightarrow} T_j$
写读依赖 (write-read/wr dependency)	事务T_i提交了数据项x上的写版本x_m, 事务T_j读取到了x_m, 则T_j写读依赖于T_i	$T_i \overset{wr}{\rightarrow} T_j$
读写依赖 (read-write/rw dependency)	事务T_i读到数据项x的版本x_m, 而后事务T_j提交了x的下一个写版本x_n, $m<n$, 则T_j读写依赖于T_i	$T_i \overset{rw}{\rightarrow} T_j$

● 数据依赖图. 根据事务历史我们可以定义出数据依赖图. 对于一个事务历史 H, 依赖图按照以下方法生成: 每个提交事务对应图中的一个节点. 每对数据依赖关系对应图中的一条边, 若事务 T_j 写读或写写或读写依赖于 T_i, 那么依赖图中就有一条从 T_i 指向 T_j 的 wr/ww/rw 边. 另外, 环定义在依赖图中仅显示已提交事务的信息, 对于回滚事务会额外说明.

2.2 基于环的数据异常与隔离级别定义

本节回顾了文献 [23] 中基于环的数据异常定义方法, 并根据这些数据异常, 对目前数据库系统中常见的隔离级别, 包括: 写已提交、读已提交、可重复读、快照隔离、可串行化进行了重新定义.

(1) 写已提交

写已提交级别可以避免出现脏写异常. 脏写表示在 T_1 尚未提交时, 不会被来自其他事务 (如 T_2) 的写操作所覆盖. 定义如下.

定义 1. 脏写. 如果在数据依赖图中存在一个环, 环全部由写写依赖边构成, 那么我们就说历史序列 H 中存在一个脏写异常.

数据库隐式要求所有运行事务不允许出现脏写异常. 在写已提交级别, 可以保证禁止脏写异常.

定义 2. 写已提交. 禁止脏写异常环. 在数据依赖图中, 若不存在脏写, 那么就称历史序列达到了写已提交级别.

达到写已提交级别, 所写的数据可以看作是已提交的数据. 在环定义中, 并不要求所写数据一定是严格已提交的, 即使是并发的写, 只要不存在写写环, 那么写操作所在的事务就等价于一个串行化的序列 (仅考虑写操作).

(2) 读已提交

在 ANSI 定义中, 禁止脏写异常、脏读异常达到读已提交级别. 脏读异常被描述为在 T_1 尚未提交时, T_1 所做的修改不能被其他事务比如 T_2 读到. 读已提交包含 2 个定义.

定义 3. 脏读.

① 读回滚. 如果历史记录 H 中 T_2 读取了 T_1 写过的数据, 并且 T_1 回滚, 则历史序列 H 中存在“读回滚”的脏读异常.

② 读中间. 如果历史记录 H 中 T_2 读取了 T_1 写过但并不是最终提交版本的数据, 则历史序列 H 中存在“读中间”的脏读异常.

③ 写读环. 如果在数据依赖图中存在一个环, 环全部由写写依赖边或写读依赖边构成 (即, ww*∪wr*), 那么就说历史序列 H 中存在一个脏读异常.

定义 4. 读已提交. 禁止脏读异常. 在数据依赖图中, 不存在脏读异常, 那么称历史序列达到了读已提交级别.

(3) 可重复读

在可重复读级别, 我们对不可重复读这一数据异常进行了扩充, 加以考虑与其本质原因相同异常, 如丢失更新、读偏斜等.

例 1: 图 2 表示了产生不一致异常的几个情况. 图 2(a) 是不可重复读异常, P_1、P_2 表示数据库系统中的两个进程, 左图是事务历史的执行示意图, 右边是根据事务历史得到的数据依赖图. 事务 T_1 两次读取数据项 x 得到了不同的版本, 产生了不可重复读异常. 图 2(a) 右侧的数据依赖图对应左侧的进程, 由一条 rw 边和一条 wr 边构成的环, 这个环可以理解为 T_j 接受了 T_2 的修改 (ww 边) 的同时, 又错过了 T_i 的修改 (rw 边), 揭示了不可重读异常环的本质. 将图 1(b) 环中的 wr 边改为 ww 变后得到图 2(b), 此时环表示出现了丢失更新 (lost update) 异常, 对应图 2(b) 左侧示意图. 之所以考虑转换, 是因为写写依赖与写读依赖都会对事务的正确串行顺序产生影响. 在图 2(b) 所示的场景中, 事务 T_1 读取到的数据版本先于 T_2, 而后续的写又位于 T_2 修改后, T_2 的修改丢失了.

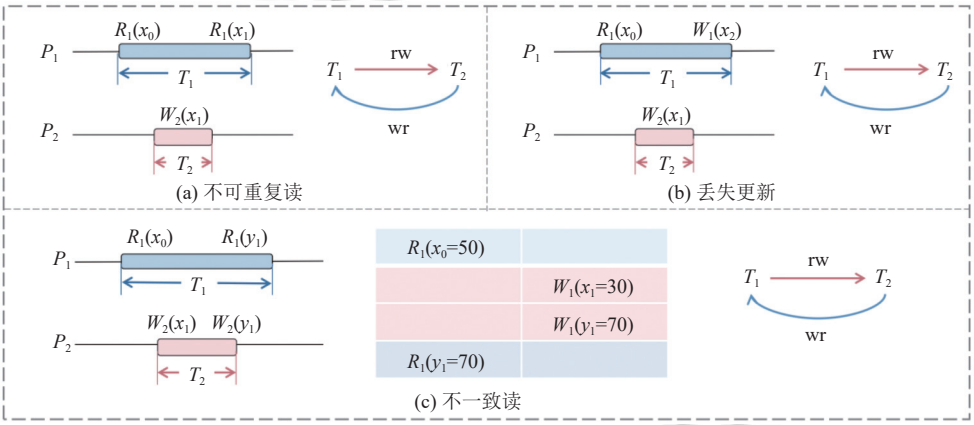


图 2 不可重复读及丢失更新等异常图

究其本质, 不可重复读和丢失更新都是读/写到了其他事务修改前后的不同状态. 更进一步, 考虑读取或修改的不是同一数据项 , 而是有不变量约束的不同数据项, 如图 2(c). 对于数据项 x, y 有 $x+y$=100 的不变量约束, 当事务 T_1 两次分别读取到事务 T_2 修改前、后的不一致状态时, T_1 读到了 $x+y$=130 的异常结果. 此时, 异常产生的本质仍然是由读/写到了其他事务修改前后的不一致性状态. 综上, 本文使用对不可重复读异常进行了扩充.

可重复读级别的定义如下.

定义 5. 不可重复读. 如果在数据依赖图中存在一个环, 环由一条读写依赖边和任意条写写依赖边、写读依赖边构成, 那么就说历史序列 H 中存在一个不可重复读异常.

定义 6. 可重复读. 禁止脏读异常、不可重复读和丢失更新异常. 在数据依赖图中, 不存在脏读、不可重复读和丢失更新, 那么称历史序列达到了可重复读级别.

在这一级别, 我们将不可重复读异常扩充到了不可重复读和丢失更新、读偏斜等异常, 同时将定义自然地扩充到多变量约束上.

(4) 快照隔离与并行快照隔离

在 Adya 等人[23,28]对常见隔离级别进行的环定义的基础上, Saeida Ardekani 等人[29]和 Sovran 等人[30]的工作进一步对快照隔离和并行快照隔离进行了定义, 其定义如下:

定义 7. 快照隔离. 禁止出现环, 环由零条、一条或多条不连续的 rw 边, 以及任意条 ww、wr 边构成 (即 ((ww∪wr); rw?)+), 那么称历史序列达到了快照隔离级别.

定义 8. 并行快照隔离. 禁止出现环, 环由零条或一条的 rw 边, 以及任意条 ww、wr 边构成 (即 (ww∪wr)+; rw?), 那么称历史序列达到了并行快照隔离级别.

(5) 可串行化

可串行化级别需要禁止所有数据依赖中的环产生. 当数据依赖图中无环时, 事务的执行历史必有等价的串行执行序列. 可串行化级别的定义如下.

定义 9. 可串行化. 禁止脏读, 同时在数据依赖图中, 禁止出现环, 环由任意条 ww、wr、rw 边构成, 那么称历史序列达到了可串行化级别.

综上, 隔离级别的换定义如表 2 所示.

表 2 隔离级别及对应禁止的环结构

隔离级别	环结构
写已提交	ww+
读已提交	(ww∪wr)+
可重复读	(ww∪wr)+;rw?
并行快照隔离	(ww∪wr)+;rw?
快照隔离	((ww∪wr);rw?)+
可串行化	(ww∪wr∪rw)+

此外, 在环定义中谓词在所有隔离级别中被灵活处理. 处理的方式是, 在数据依赖边中增加了基于谓词的读写依赖边和基于谓词的写读依赖边. 由于本文讨论的重点不在于此, 因此, 在后文中本文默认写读依赖、读写依赖包括同时谓词和数据项上的依赖, 不作分类讨论.

3 抽象执行与分布式系统一致性

类似多级隔离级别理论的研究历程, 分布式系统中对一致性模型的研究也经历了漫长的发展. 在 CAP 理论中, 为了得到更佳的可用性和分区容错性, 分布式系统牺牲部分正确性形成了多种多样从强到弱的一致性模型.

早在 20 世纪 80 年代, 分布式系统, 尤其是存储系统就对一致性模型进行了研究. Herlihy 等人[9]首先提出了线性一致性 (linearizability), 随后 Lamport[10]又提出了顺序一致性 (sequential consistency). 线性一致性和顺序一致性属于强一致性模型 (strong consistency), 对数据的一致性状态的要求严格. 随着数据库系统规模和处理数据规模的极速增大, 主流需求变为追求大规模、高可用, 于是因果一致性 (causal consistency)、PRAM[31]一致性等牺牲部分正确性换取高性能的一致性模型被广泛采用. Viotti 等人[32]使用抽象关系提出了一套体系化地定义方式, 整理、定义了多种级别的一致性模型. 本文主要基于文献 [32] 进行, 下面对他的工作进行回顾和介绍.

3.1 仲裁、可见性与会话

根据 Burckhardt[33]的工作 (后文中也用抽象执行 (abstract executions) 代指), Viotti 等人[32]根据历史序列 (history)、可见性 (visibility) 和仲裁 (arbitration) 对一致性模型系统进行建模. 其中历史序列是给定执行的一组操作, 另外两个组成部分, 可见性和仲裁, 则直观地捕获了分布式环境下的不确定性 (例如, 消息传递顺序) 以及实际实现中的约束. 可见性和仲裁定义了历史序列中事务对之间的关系, 而这个关系可以解释并证明了操作对的输出. 具体有如下定义.

- 会话顺序 (session order/so): 会话顺序为同一进程调用的两个操作 (我们说这些操作属于同一个会话) 顺序

依次执行的自然偏序. 同一进程上的任意两个操作一定有会话顺序的先后关系 .

● 可见性顺序 (visibility/vis): 可见性顺序是一种用于解释写入操作的可见传播顺序. 直观地, 如果操作 a 的结果对操作 b 可见, 即, $a \overset{\text{vis}}{\rightarrow} b$, 意味着 a 的修改对调用 b 的进程可见 (例如, b 可以读取 a 写入的值). 如果两个写操作之间如果没 vis 顺序, 那么它们彼此不可见.

● 仲裁顺序 (arbitration/ar): 仲裁顺序是操作历史中的事务上的一个全序关系, 用于指定系统中并发的冲突和不可见事务之间的顺序. 这是实际实现中的先后顺序, 不同的实现机制有不同的方式来获得仲裁顺序, 如, 分布式时间戳方式[34] , 或是共识协议 [35−37], 集中调度或者确定性地冲突解决策略.

3.2 符　号

作为参考, 表 3 列出了使用到的一些符号及其含义.

表 3　符号及其含义

符号	含义
∪	并集. 如(ww∪wr) 表示集合中ww边和wr边的并集
+	前面的内容在集合中出现一次或多次. +等价于{1, }
*	前面的内容在集合中出现任意次. *等价于{0, }
?	前面的内容在边集合中出现0次或1次. ?等价于{0, 1}
;	组合. 如 (ww; wr) 表示在集合中出现ww边后面连接wr边, 即$T_1 \overset{\text{ww}}{\rightarrow} T_2 \overset{\text{wr}}{\rightarrow} T_3$
→	依赖边. 如$T_1 \rightarrow T_2$指的就是从事务T_1指向事务T_2的依赖边
⊆	包含于, $A \subseteq B$即A是B的子集
so	分布式一致性会话顺序
vis	分布式一致性可见性顺序
ar	分布式一致性仲裁顺序
hb	分布式一致性发生顺序
rt	分布式一致性实时顺序

3.3 分布式系统一致性

下面对文献 [32] 中定义的分布式系统一致性级别进行回顾.

(1) 读己之所写 (read your writes)

读己之所写表示一个客户端会话会首先执行对某个数据项的写入操作, 然后再读取该数据项, 也就意味着读到的结果不旧于自己写入的数据. 基于前文中的基础概念, 我们将其抽象执行定义为: $\text{so}|_{\text{w}\rightarrow\text{r}} \subseteq \text{vis}$, 即具有会话顺序的写操作与读操作, 也一定具有可见性顺序.

(2) 单调写 (monotonic writes)

单调写一致性保证了在一个进程上, 两个写操作对同一数据项顺次进行了写 w_1 和 w_2, 那么所有进程都应该观察到 w_1 在 w_2 之前. 单调写对不同进程上的写入没有约束, 仅要求同一进程上的写被顺序执行. 其抽象执行定义概括为: $\text{so}|_{\text{w}\rightarrow\text{w}} \subseteq \text{ar}$.

(3) 单调读 (monotonic reads)

单调读一致性是指如果一个进程从系统中读取出一个数据项的某个值后, 那么系统对于该进程后续的任何数据访问都不应该返回更旧的值. 单调对同一进程上的读取进行了约束. 单调读的抽象执行定义为: $(\text{vis}; \text{so}|_{\text{r}\rightarrow\text{r}}) \subseteq \text{vis}$.

(4) 写在读之后 (writes follow reads, session causality)

写在读之后保证如果同一个进程上的事务读取到了 x_i, 随后进行了对 x 的写, 如 x_j, 那么所有进程都应该观测到 x_j 在 x_i 之后. 写在读之后意味着, “不能改变读取操作的过去”, 也就是说, 当一个版本的值被读取, 那么任何操作不会影响到早于这个读取的版本. 其抽象执行定义为: $(\text{vis}; \text{so}|_{\text{r}\rightarrow\text{w}}) \subseteq \text{ar}$.

(5) PRAM (pipeline random access memory, FIFI consistency)

PRAM 要求: 同一进程上的两个写操作, 在任意进程上观测到的它们之间的顺序都应当与它们在进程上执行的顺序一致; 而来自不同进程上的写操作, 在不同的进程上观测到的顺序可以不一致. PRAM 的抽象执行定义为: so⊆vis.

(6) 因果一致性 (causality)

因果一致性是保证具有因果关系的操作的顺序在任意进程上观测都应该是一致的, 但是不具备因果关系的操作在不同的进程上可以不一致. 其抽象执行定义如下:

CAUSALVISIBILITY = hb ⊆ vis;

CAUSALARBITRATION = hb ⊆ ar;

CAUSALITY(F) = CAUSALVISIBILITY∧CAUSALARBITRATION∧RVAL(F).

RVAL(F), 表示返回值一致性, 返回值一致性是抽象执行的谓词, 它保证执行任何给定操作的返回值属于预期返回值的集合. 即使在数据库上的数据有多副本, 也会返回一致地最后写入的值 (last write win).

(7) 顺序一致性 (sequential consistency)

顺序一致性要求顺序之间形成一个全序, 并且该全序要与各个进程上原本的操作顺序保持一致. 顺序一致性的抽象执行定义如下:

SEQUENTIALCONSISTENCY(F) = SINGLEORDER ∧ PRAM ∧ RVAL(F), SINGLEORDER 施加了同时定义 vis 和 ar 的单个全局顺序.

(8) 线性一致性 (linearizablility)

线性一致性是指, 所有操作都原子性地发生, 形成一个全序, 且该全序保留了操作的实时序. 粗略地讲, 线性化是一个正确性条件, 它确定每个操作应该在调用和响应之间的某个时间点立即应用. 线性一致性的抽象执行定义如下:

LINEARIZABILITY(F) = SINGLEORDER ∧ REALTIME ∧ RVAL(F).

同样, 线性一致性要求一个 SINGLEORDER 全序, 同时 REALTIME 约束仲裁 (ar) 以遵守实时约束.

4 基于环的分布式系统一致性建模

Viotti 等人在文献 [32] 中对非事务系统中的一致性进行了详尽的阐述. 图 3 展示了多种分布式一致性之间的关系, 图中的有向边将两个操作一致性关联起来, 强的一方需要保证弱的一方的所有特性. 在图 3 中, 由底部到顶部, 一致性级别也越来越高.

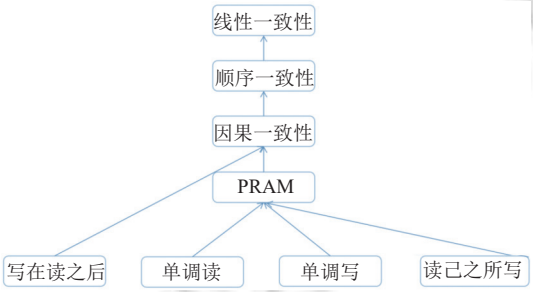


图 3 分布式一致性级别强弱划分

根据图 3 中的强弱级别, PRAM 所禁止的环应当包含单调读、单调写、读己之所写, 线性一致性禁止的环也应当包含顺序一致性禁止的环, 以此类推, 分布式一致性级别的强作划分可以作为建模过程中的一项理论依据. 下面, 我们将根据抽象执行定义对分布式一致性进行建模.

4.1 抽象依赖图

为了解决由分布式架构带来的数据不一致问题, 本节参照隔离级别的环定义, 通过构建事务的抽象依赖图, 并

通过识别抽象依赖图中的环来定义分布式系统一致性.

根据事务历史和抽象执行顺序, 可以定义抽象依赖图. 对于一个事务历史 H, 抽象依赖图按照以下方法生成: 抽象依赖图以事务为节点, 每个提交事务都对应图中的一个节点. 每一对抽象执行顺序都对应图中的一条边. 有事务 T_i 的某一操作 T_j 可见, 那么抽象依赖图中就有一条从 T_i 指向 T_j 的 vis 边. 同理, ar 边、so 边亦然. 下面对 so 边、vis 边以及 ar 边的定义进行事务粒度下的重写:

● 会话顺序 (so 边): so 边抽象为在一个进程上的事务顺序依次执行的顺序. 同一进程上的任意两个事务 一定有会话顺序的先后关系 .

● 可见性顺序 (vis 边): vis 边是一种用于解释写入操作的可见传播顺序. 直观地, 如果事务 a 对事务 b 可见, 即, $a \overset{\text{vis}}{\rightarrow} b$, 则表示事务 a 的影响 (如 updates) 是事务 b 可以观测到, 比如说, 事务 b 读到了一个事务 a 写的数据项版本. 两个写事务之间如果没有 vis 边, 那么它们之间是不可见的.

● 仲裁顺序 (ar 边): ar 边是事务历史中的事务上的一个全序关系, 用于指定系统中并发的冲突和不可见事务之间的顺序.

构建抽象依赖图后, 以此为基础, 我们需要考虑分布式系统一致性的环定义构建, 这里主要有两个难点. 其一, 分布式系统一致性从操作粒度变为事务粒度后, 部分定义将不再适用, 必须抽象出分布式系统一致性抽象执行定义的核心思想, 以构建正确合理的模型. 其二, 抽象执行定义并不是以环的方式来定义分布式一致性级别的, 即使构建了抽象依赖图也不能从中找到环. 如何构建环并且构建不同结构的环, 来定义多级分布式系统一致性是本节的重点.

此外, 抽象执行根据客观规律, 满足 $\text{vis} \subseteq \text{ar}$, $\text{ar}^{+} \subseteq \text{ar}$ 以及 ar 自身不成环. 除此之外, 抽象执行中还定义了发生顺序、实时顺序以及返回值一致性等. 发生顺序 (happens-before order/hb): 表示发生先后关系, 定义 hb = (so ∪ vis)+. 实时顺序 (realtime order/rt) 表示真实时间下的先后关系.

4.2 分布式系统一致性建模

下面将详细介绍分布式系统一致性从抽象执行到环的建模方法:

(1) 读己之所写 (read your writes)

读己之所写的抽象执行定义为: $\text{so}|_{\text{w}\rightarrow\text{r}} \subseteq \text{vis}$, 即: 存在事务 T_i、T_j, 若有 $T_i \overset{\text{so}|_{\text{w}\rightarrow\text{r}}}{\rightarrow} T_j$, 必有 $T_i \overset{\text{vis}}{\rightarrow} T_j$. 其中, 事务 T_i、T_j 操作同一数据对象, T_i 为读操作、T_j 为写操作.

沿着环定义的思路, 当抽象依赖图中出现某类环 (我们定义为 RYW 环) 时, 该历史序列违背读己之所写一致性. RYW 环应该等价于: 存在事务 T_i、T_j, 满足 $T_i \overset{\text{so}|_{\text{w}\rightarrow\text{r}}}{\rightarrow} T_j$, 但不满足 $T_i \overset{\text{vis}}{\rightarrow} T_j$. 转变为图的形式, 以事务作为顶点, 相当于当 T_i 到 T_j 间存在 $\text{so}|_{\text{w}\rightarrow\text{r}}$ 边时, 不存在 vis 边, 如图 4.

为了构建首尾相连的环, 我们尝试对 $T_i \overset{\text{vis}}{\rightarrow} T_j$ 边进行变形, 使其与 $T_i \overset{\text{so}|_{\text{w}\rightarrow\text{r}}}{\rightarrow} T_j$ 边成环. 我们引入与 vis 相对立的边¬vis, 表示“不可见”, 不满足“$T_i \overset{\text{vis}}{\rightarrow} T_j$”相当于“$T_i \overset{\neg\text{vis}}{\rightarrow} T_j$”, 表示 T_i 对 T_j 不可见, 在图 5 中表示为一条从 T_i 指向 T_j 的边. 此时两条边方向一致, 尚不成环. 为使成环, 我们对¬vis 做一个代换, 令 $T_i \overset{\neg\text{vis}}{\rightarrow} T_j$ 等价于 $T_j \overset{\neg\text{vis}^{-1}}{\rightarrow} T_i$, 从而改变边的方向, 如图 5 所示.

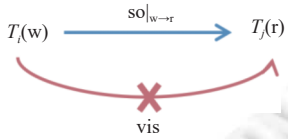


图 4 违背读己之所写

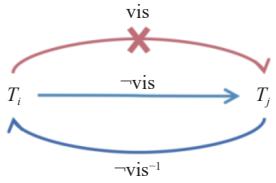


图 5 $\neg\text{vis}^{-1}$ 边

此时得到了构建 RYW 环的两类边: $T_i \overset{\text{so}|_{\text{w}\rightarrow\text{r}}}{\rightarrow} T_j$, 在图 6 中用从 T_i 到 T_j 的 so 边表示. so 表示在同一进程上的事务先后执行顺序, 客观地, so 具有传递性, 因此 RYW 环中可能存在一条或多条连续的 so. 另一类边, 违背 $T_i \overset{\text{vis}}{\rightarrow} T_j$,

则被转化为从 T_j 指向 T_i 的$\neg vis^{-1}$ 边. 由于 $T_j \xrightarrow{\neg vis^{-1}} T_i$ 的边一定是从读到写, 我们额外标记为$\neg vis^{-1}(rw)$ 边. 综上, 从 $so|_{w\to r} \subseteq vis$, 可以建模出如下定义.

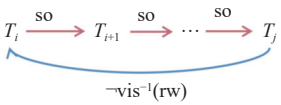


图 6 RYW 环

定义 10. 读己之所写. 在一致性依赖图中不存在 RYW 环, 即满足读己之所写一致性.

定义 11. RYW 环. 环由一条或多条 so 边及一条$\neg vis^{-1}(rw)$ 边构成.

在此环中, 我们不对 so 施加关于数据对象、操作的限制, 而将约束放到$\neg vis^{-1}$ 上. 一方面, 由于此级别是针对同一进程上写、读之间的操作, 成环时必然有一条$\neg vis^{-1}(rw)$ 来约束, 将约束放在这条边上是充分的; 另一方面, 这有利于在下一步的多级结合中将$\neg vis^{-1}$ 与数据依赖建立联系.

本文规定 so 之间必须是对同一数据对象进行操作, 放宽到同一进程 (so 边), 而对数据对象的限制由$\neg vis^{-1}$ 边来完成. $\neg vis^{-1}$ 边在此小节中作为 vis 边的变形引出, 意义为“不可见”边的逆向边, 与实际对应的意义不明确, 但$\neg vis^{-1}$ 边作为一致性模型与隔离级别结合的基础边, 将在第 5 节中进一步讨论.

在实际系统中, 如果某一进程调用的读取操作, 不能保证是从响应过自己之前写操作的进程上获取的, 就可能会发生此种异常.

(2) 单调写 (monotonic writes)

单调写一致性保证了在一个进程上, 两个事务顺次进行了写操作 w_1 和 w_2, 那么所有进程都应该观察到 w_1 在 w_2 之前. 单调写对不同进程上的写入没有约束, 仅要求同一进程上的写被顺序执行.

根据单调写的抽象表达 $so|_{w\to w} \subseteq ar$, 使用相似的转化思路, 单调写意味着统一进程上的两个事务不能违背仲裁顺序. 与可见性不同, 仲裁顺序在一致性中是一个严格的全序, 任意两个操作 A、B 之间要么 $A \xrightarrow{ar} B$ 要么 $B \xrightarrow{ar} A$. 在同一进程上事务顺次进行, 仲裁顺序严格需要满足要么 $T_i \xrightarrow{ar} T_j$, 要么 $T_j \xrightarrow{ar} T_i$. 因此, 对于两个事务 T_i、T_j 而言, 违背单调写一致性, 即为: 存在事务 T_i、T_j, 有 $T_i \xrightarrow{so|_{w\to w}} T_j$, 但不满足 $T_i \xrightarrow{ar} T_j$. 即满足 $T_j \xrightarrow{ar} T_i$, 构建成环. 综上, 单调写的环定义如下.

定义 12 . 单调写. 在一致性依赖图中不存在 MW 环, 即满足单调写一致性.

定义 13. MW 环. 环由一条或多条 so 边及一条 ar(ww) 边构成, 如图 7.

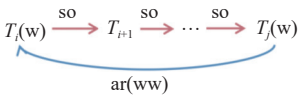


图 7 MW 环

同样, 对于事务写操作同一数据项的约束, 我们从 so 边转移到 ar 边上, 同时对连续条 so 边数目不做约束. 单调写对同一进程上的写入顺序进行了约束, 要求不能违背同一进程上对同一数据项写入的先后顺序.

(3) 单调读 (monotonic reads)

单调读的抽象执行定义为 $(vis; so|_{r\to r}) \subseteq vis$, 与读己之所写、单调写不同, 单调读抽象执行定义的前半部分是一个组合, 即, 一条连续的 vis 边和 $so|_{r\to r}$ 边. 同样的思路, 我们可以构建出单调读的定义.

定义 14. 单调读. 在一致性依赖图中不存在 MR 环, 即满足单调读一致性.

定义 15. MR 环. 环由一条或多条 so 边, 一条连续的$\neg vis^{-1}(rw)$ 边和一条 vis 边构成, 如图 8.

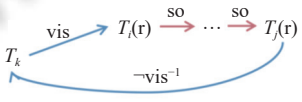


图 8 MR 环

从抽象执行定义到环定义, 不仅需要考虑边、环的构建, 还需要考虑由操作粒度变为事务粒度的差异. 在读己之所写与单调写中仅需考虑同一进程上先后的任意两个事务是否违背 vis 或 ar, 而在单调读中, 有两个以上关键事务, 这就需要考虑在多数据约束下的情况了. 举个例子, 有历史序列 H_{mr}, 事务 T_i 和 T_j 在同一进程上, 有 $T_i \overset{\text{so}|_{\text{r}\to r}}{\to} T_j$: H_{mr}=$W_0(x_0)$ $W_0(y_0)$ C_0 $W_k(x_1)$ $W_k(y_1)$ C_k $R_i(x_1)$ C_i $R_j(y_0)$ C_j.

我们认为 H_{mr} 也违反了单调读一致性. 数据库以事务为一个单位, 而非以单个操作或数据项为单位, 例说, x 和 y 分别代表员工缴纳的公积金和公司缴纳的公积金, x=y. 在某次涨薪涨公积金 (即, 事务 T_k) 后, T_i 读到了涨薪后的数值, 但 T_j 却读到涨薪前的数值, 仿佛时光倒流. 因此, 上述单调读及 MR 的环的定义正确地捕获了单调读一致性的特征和事务多数据项约束的特点.

单调读对同一进程上的读取进行了约束, 如果用户从不同从库进行多次读取, 就可能发生违背单调读的情况. 在实现中, 达到单调读的一种方式是确保每个用户 (即进程) 总是从同一个节点进行读取 (不同的用户可以从不同的节点读取), 而不是随机选择节点.

(4) 写在读之后 (writes follow reads, session causality)

写在读之后的抽象执行定义为 (vis; $\text{so}|_{\text{r}\to\text{w}}$) ⊆ ar, 类似地, 可以得到如下定义.

定义 16. 写在读之后. 在一致性依赖图中不存在 WFR 环, 即满足写在读之后一致性.

定义 17. WFR 环. 环由一条或多条 so 边, 以及连续的一条 ar(ww) 边和一条 vis(wr) 边构成, 如图 9.

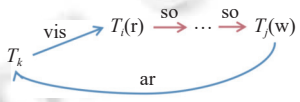


图 9 WFR 环

出现 WFR 环表明在满足抽象表达 (vis; $\text{so}|_{\text{r}\to\text{w}}$) 的情况下, 违背了 ar 顺序.

(5) PRAM (pipeline random access memory, FIFI consistency)

PRAM 的抽象定义 PRAM 的抽象定义为 so ⊆ vis, so ⊆ vis 也被称为强回话保证, 保证了具有会话顺序的事务之间也有可见性顺序.

按照 Brzezinski 等人[38]所述, 如果系统提供读己之所写、单调读和单调写保证, 则达到了 PRAM 一致, 反之亦然. 但是, 代入条件 so ⊆ vis, 3 类一致性并不能直接推出: (1) $\text{so}|_{\text{w}\to\text{r}}$ ⊆ vis, (2) $\text{so}|_{\text{w}\to\text{w}}$ ⊆ ar, (3) (vis;$\text{so}|_{\text{r}\to\text{r}}$) ⊆ vis. 具体地: (1) 满足; (2) 有 $\text{so}|_{\text{w}\to\text{r}}$ ⊆ so ⊆ vis, 由于客观存在 vis ⊆ ar, 因此满足; (3) 有 (vis; $\text{so}|_{\text{r}\to\text{r}}$) ⊆ (vis;vis), 此前并没有规定 (vis;vis) ⊆ vis, 故不满足. 因此, 本文认为在对 PRAM 进行环定义时, 不能直接使用 so ⊆ vis 来推导环, 而是选择, 得出定义如下.

定义 18. PRAM. 在一致性依赖图中不存在 PRAM 环, 即满足 PRAM 一致性.

定义 19. PRAM 环. 包括 RYW 环、 MR 环 、MW 环. 即, 环由一条或多条 so、零或一条 vis 边, 和一条$\neg\text{vis}^{-1}$ 边构成, 如图 10.

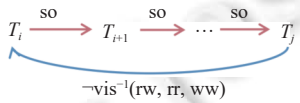


图 10 PRAM 环示意图

那么依照 so ⊆ vis 来对环进行建模是否可行? 从概念上来说是可行的, 但是需要对单调读进行处理. 单调读可以抽象为由 so 边和一条$\neg\text{vis}^{-1}$(rr) 边构成, 使用$\neg\text{vis}^{-1}$(rr) 来表示两个读事务间的$\neg\text{vis}^{-1}$.

直观来看, PRAM 定义了同一进程上的读写事务、读读事务、写写事务的可见性约束. 在成环结果上, 是读己之所写、单调读和单调写 3 类一致性的合集.

(6) 因果一致性 (causality)

因果一致性决定了所有进程在因果相关操作的顺序上都是一致的. 它包含了 PRAM 和写在读之后一致性, 其

抽象定义如下:

CAUSALVISIBILITY = hb ⊆ vis;

CAUSALARBITRATION = hb ⊆ ar;

CAUSALITY(F) = CAUSALVISIBILITY ∧ CAUSALARBITRATION ∧ RVAL(F).

RVAL(F) 表示返回值一致性 (与 replicated data type 对应), 保证执行任何给定操作的返回值属于预期返回值的集合. 即使在数据库上的数据有多副本, 也会返回一致地返回可见集中最后写入的值 (last write win). 将因果一致性扩展到事务粒度, hb 顺序自然的可以抽象为提交顺序. 按照定义, 提交顺序 hb 既不能违背可见性顺序, 也不能违背仲裁顺序, 同时, 提交顺序本身不能成环 (这在一致性模型中有要求). 因此, 因果一致性定义如下.

定义 20. 因果一致性. 禁止因果环, 即满足因果一致性.

定义 21. 因果环. 环由任意条 hb 边, 0 或 1 条 ar 边, 以及 0 或 1 条$\neg vis^{-1}$ 边构成. 可表示为 (hb+∪ar?∪$\neg vis^{-1}$?).

因果一致性相比于 PRAM, 施加了尊重因果的条件: vis+⊆vis, 在因果一致性级别以上, 将对 vis 的条数不再进行约束.

此外, 对因果环施加 REALTIME 约束得到实时因果一致性 (real-time causality), 其抽象定义如下.

REALTIMECAUSALITY(F) = CAUSALITY(F) ∧ REALTIME.

定义 22. 实时因果一致性. 禁止实时因果环, 即满足实时因果一致性.

定义 23. 实时因果环. 环由任意条 hb 边, 零或一条 ar 边, 以及零或一条$\neg vis^{-1}$ 边构成. 可表示为 (hb+∪rt+∪ar?∪$\neg vis^{-1}$?).

(7) 顺序一致性 (sequential consistency)

顺序一致性[10]要求事务之间形成一个全序关系, 并且需要满足 (1) 各进程上事务之间的偏序关系要求在全序关系中保持, 即强会话保证; (2) 不违背仲裁顺序一致; (3) 不违背可见性顺序一致. 顺序一致性不需要对跨进程的事务进行实时排序, 仅保留由同一进程调用的事务排序 (如 PRAM 一致性一样), 顺序一致性的抽象定义如下.

SEQUENTIALCONSISTENCY(F) = SINGLEORDER∧PRAM∧RVAL(F).

SINGLEORDER 施加了同时定义 vis 和 ar 的单个全局顺序. 顺序一致性包括了上述 (1)–(6) 个级别的一致性, 同时需要满足由全局序. 按照这一包含关系我们定义顺序一致性如下.

定义 24. 顺序一致性. 在一致性依赖图中不存在顺序环, 即满足顺序一致性.

定义 25. 顺序环. 环由任意条 so, vis, ar, $\neg vis^{-1}$ 边构成.

so, vis, ar, $\neg vis^{-1}$ 这 4 类边不成环可以保证顺序一致性满足会话保证, 同时可见性顺序和仲裁顺序都不会被违背.

(8) 线性一致性 (linearizablility)

线性一致性是指, 所有操作都原子性地发生, 形成一个全序, 且该全序保留了事务的可见性顺序、仲裁顺序和实时序. 粗略地讲, 线性化是一个正确性条件, 它确定每个事务应该在调用和响应之间的某个时间点立即应用. 线性化长期以来一直被视为分布式存储实现应该达到的理想正确性条件. 线性一致性的抽象定义如下.

LINEARIZABILITY(F) = SINGLEORDER∧REALTIME∧RVAL(F).

同样, 线性一致性要求一个 SINGLEORDER 全序, 同时 REALTIME 约束仲裁 (ar) 以遵守实时约束. 我们使用实时顺序 rt, 来表示在真实时间提交事务的先后顺序 (注意, 在一致性模型中 rt 为一个操作的发生时间, 在事务中抽象为事务提时间). 例如, 事务 T_1 提交的真实时间早于 T_2 提交的真实时间, 则有 $T_1 \xrightarrow{rt} T_2$.

类似顺序一致性, 我们定义线性一致性如下.

定义 26. 线性一致性. 在一致性依赖图中不存在线性环, 即满足线性一致性.

定义 27. 线性环. 环由任意条 rt, vis, ar, $\neg vis^{-1}$ 边构成.

由于 so ⊆ rt, 因此线性一致性强于顺序一致性, 在线性一致性下, 读、写事务全局有序且不会成环.

至此, 我们对分布式一致性理论的环定义进行一个小结, 每个级别在抽象依赖图中所需要禁止的成环情况在表 4 中列出.

表 4 分布式一致性定义成环结构

分布式一致性级别	成环情况	对应图
读己之所写	so+∪¬vis^{-1}(rw)	图6
单调写	so+∪ar(ww)	图7
单调读	so+∪ (¬vis^{-1}(rw);vis)	图8
写在读之后	so+∪(ar(ww);vis(wr))	图9
PRAM	so+∪¬vis^{-1};vis?	图10
因果一致性	hb+∪ar?∪¬vis^{-1}?	返回值一致
实时因果一致性	hb+∪rt+∪ar?∪¬vis^{-1}?	返回值一致
顺序一致性	(so∪vis∪ar∪¬vis^{-1})+	返回值一致
线性一致性	(rt∪vis∪ar∪¬vis^{-1})+	返回值一致

5 去中心化环境下的多级一致性统一建模

5.1 混合依赖图

我们构建混合依赖图来定义同时支持对隔离级别和分布式系统一致性的一致性级别. 构建混合依赖图并不直接加入所有类型边, 而是需要对它进行精简. 混合依赖图主要由 ww、wr、rw、so、rt 这 5 类边构成, 而 vis、ar 以及¬vis^{-1} 边则在与隔离级别的结合过程中, 按照规则被替换. 关于边替换规则, 我们有如下讨论:

(1) ar 边. 在对隔离级别的 3 类边的划分中, 会涉及版本的概念. 隔离级别 3 类边的方向也都建立在版本的设定上. 而在多级一致性中没有版本的概念, 取而代之的是仲裁顺序 ar 和可见性顺序 vis. 仲裁顺序用于指定系统中的并发冲突和不可见事务之间的顺序, 随系统的实现机制不同, 包括时间戳方式、共识协议或集中调度等有不同的判定方法. 仲裁顺序的核心思想在于标识出操作的顺序, 这与隔离级别中的版本思想非常相似. 在隔离级别中, ww 边指示了系统判定的两个操作的先后, 也是版本先后. 因此, 可以认为 ar 顺序等同于 ww 依赖.

(2) vis 边. 可见性顺序 vis 是用于解释写入操作的可见传播顺序, 表示一事务的影响是另一事务可以观测到的. wr 依赖根据定义直观地属于可见性, ww 依赖则需要进一步考虑. 在一致性模型中, 对于不可见的写操作之间, 是没有 vis 关系的, 因为他们之间互不影响, 只存在一个真实的 ar 顺序; 但是, 在事务隔离级别中, 当数据库运行在写已提交级别以上时, 不考虑读事务, 我们可以认为写事务之间是可以串行执行的, 后面的写会受到前面的写的影响. 因此, 在写已提交级别以上, vis 也应当包含 ww 依赖.

(3) ¬vis^{-1} 边. 下面对引入的¬vis^{-1} 边进行分析. 假设有 $T_1 \overset{\neg vis^{-1}}{\rightarrow} T_2$, 那么 T_2 对 T_1 是不可见的. 考虑, 排除 T_2 对 T_1 可见的情形, 就是 T_2 对 T_1 不可见的情形. 根据上文, 当 $T_2 \overset{wr}{\rightarrow} T_1$ 时, 或者写已提交级别以上 $T_2 \overset{ww}{\rightarrow} T_1$ 时, T_2 对 T_1 可见, 即 $T_2 \overset{vis}{\rightarrow} T_1$. 反过来, 当 $T_1 \overset{rw}{\rightarrow} T_2$ 时, 或 $T_1 \overset{ww}{\rightarrow} T_2$ 时, 满足 T_2 对 T_1 不可见, 即 $T_2 \overset{\neg vis}{\rightarrow} T_1$. 综上, ¬vis^{-1} 包含 rw 边以及在写已提交级别以上的 ww 边, 方向与¬vis^{-1} 一致.

综上, 构建混合依赖图只需 5 类边, 大大简化了模型的复杂程度, 也减轻了实现中构建混合依赖图的性能压力. 在结合过程中, 表 5 列出来不同级别结合时边的替换规则, −表示此时不满足任何级别. 在相应隔离级别下, √所在行对应的边包含√所在列对应的边, 例如, 在写已提交级别, vis 边的含义包含 ww 边及 wr 边. 表 6 是不同一致性级别下一致性边含有的特殊属性.

表 5 不同隔离级别下边的转化关系

边	−				写已提交				读已提交/可重复读/快照隔离				可串行化			
	ar	vis	¬vis^{-1}	hb	ar	vis	¬vis^{-1}	hb	ar	vis	¬vis^{-1}	hb	ar	vis	¬vis^{-1}	hb
ww	√				√	√	√	√	√	√	√	√	√	√	√	√
wr		√				√				√	√	√		√	√	√
rw			√				√				√				√	√
so				√				√				√				√

表 6 不同一致性级别下边的属性

一致性级别	边关系
—	so+ ⊆ so, ar+⊆ ar, vis ⊆ ar
PRAM以上 (会话保证)	so ⊆ vis
因果一致性以上 (尊重因果)	vis+ ⊆ vis

5.2 基于环的多级一致性统一建模

下面, 我们将对去中心化环境下基于环的隔离级别和分布式系统一致性进行统一建模, 分级讨论不同级别间结合的可能性与效果.

(1) 弱一致性的结合

弱一致性级别包括读己之所写、单调写、单调读、写在读之后等, 其对同一进程上的写、读以及部分因果(写在读之后) 进行了约束. 在建模过程中, 这些一致性级别都被定义为连续 so 边与某一其他类别边的组合, 环的约束能力较弱. 由于这些一致性约束都很弱, 因此它们可以与任意隔离级别结合, 而不对隔离级别产生影响. 例如, 对于一个可串行化的序列 $w_1(x_1)\ c_1\ w_2(x_2)\ c_2\ w_3(x_3)\ c_3$ ... 当新的写入 $w_k(x_k)$ 到达时, 我们总是可以把 w_k 放到 w_1 之前或某个最近版本之前, 也就是说作为旧版本写入, 而序列总是保持可串行化的. 但是, 增加了单调写约束后, 在同一进程的写版本只能递增, 则不再会出现这个问题.

RYW 环、MW 环、MR 环、WFR 环都可以与各隔离级别的环取并集, 即, 同时禁止两个级别中需禁止的环. 做并集后, 可以在原有隔离级别上额外对这一致性级别进行约束. 包括但不限于以下几项.

写已提交与单调写: 隔离级别中写已提交达到了写事务可串行化, 对读没有做要求; 分布式一致性中, 单调写一致性对同一进程上的写事务有先后约束. 写已提交与单调写一致性结合后, 可以保证了写事务之间不会交错且在每个进程上写事务的版本单调递增.

并行快照隔离与单调读: 并行快照隔离保证了同一个事务的读取都来自同一个快照, 与单调读结合后, 可以保证每个快照点都是单调递增的, 即, 同一进程上的事务, 其快照读到的版本不会早于之前的版本.

读已提交与读己之所写: 读已提交保证了读、写可以操作的是已提交的数据, 而不会操作到中间状态或是回滚状态. 对于读已提交来说, 规定地仍是事务间的交错程度而无关先后顺序, 读己之所写规定了同一进程上的写必须被后续的事务所觉察到. 例如, 对于同一进程上的事务 T_1、T_2, 有历史序列 $w_1(x_1)\ r_2(x_0)$, 在施加读己之所写的一致性约束后, 事务 T_2 必须观察到 T_1 的写, 此序列不被允许.

可以说, 这 4 类基础的一致性要求可以与隔离级别正交地结合, 满足低级别的一致性需求. 对于更高级别的一致性级别, 与隔离级别的结合会变得更加有趣.

(2) 因果一致性的结合

在因果一致性级别, 事务之间具备了强会话保证 so⊆vis 和因果性 vis+⊆vis. 但与上一节的直接取并集不同, 因果一致性与隔离级别结合时需要对 hb 边 (happens-before order) 分类讨论.

hb 边, 在不同隔离级别保证之上有不同含义: 在隔离级别体系中, 我们认为在达到写已提交级别以上, 具有写写依赖 (ww) 的事务之间可以视作先后提交, ww 就具有了 hb 的属性, 因此 hb 在达到写已提交后包含 ww 边; 同理, 再达到读已提交级别后, 具有 wr 依赖的事务之间也具有了 hb 属性, hb 在达到读已提交后包含 wr 边、ww 边; 在可串行化级别, hb 还包含了 rw 边. 同时, 由于因果一致性包含强会话保证和因果性, 因此 hb 也包含 so 边, 并且 hb 边是可传递的, 在环中则体现为可以出现任意条而不必有所限制.

下面解释为什么因果一致性中的因果, 在事务粒度不是 wr 具有传递性, 而是 hb 具有传递性. 在一致性中都是单原子操作, 发生 wr 依赖意味着一定是先写后读的因果关系. 而在事务关系中, 有 wr 关系并不意味着事务的先后次序一定如此. 比如, 对于事务 T_1、T_2, 有序列 $w_1(x_1)\ r_2(x_1)\ r_2(y_0)\ w_1(y_1)$ 这样的多变量操作, 事务 T_2 既读到了 T_1 修改的数据, 又读到了 T_1 修改前的数据, 情况变得比单变量情况更加复杂. 不过, 这样复杂交错过程由隔离级别来定义, 而对于一致性, 我们简化因果关系为在每一级别的 hb 关系. 因此, 我们认为从本质来看, 在结合过程中, 因果关

系并不总包含 wr 依赖, 而是应当结合隔离级别来看. 因果一致性与隔离级别的结合如表 7.

因果快照: 快照隔离机制保证一个事务的读取都来自一个快照, 但是隔离级别没有约束读取的快照是哪一个时间点的, 因此, 这个快照点可以是最新最近的, 也可以是没有任意的, 当然也可以是满足因果的, 这就是因果快照. 结合后, 快照隔离机制在保证了快照读、快照写 (先提交者胜) 的同时加以结合因果一致性, 此时, 每个事务读到的快照都是满足因果的, 每个事务的写也会服从因果.

表 7 因果一致性结合表

因果一致性	隔离级别	hb的含义	结合成环
$hb+\cup ar?\cup\neg vis^{-1}?$	写已提交	hb: so, ww	(so∪ww)+
	读已提交	hb: so, ww, wr	(so∪ww∪wr)+
	快照隔离	hb: so, ww, wr	((so∪ww∪wr)∪rw?)+
	可串行化	hb: so, ww, wr, rw	(so∪ww∪wr∪rw)+

(3) 顺序一致性的结合

顺序一致性要求所有进程都能观测到一个总序, 可串行化级别保证了事务调度等价于一个串行执行顺序, 也就是一个总序. 可串行化级别之下, 事务是没有总序的. 因此, 顺序一致性只能与可串行化级别结合, 不能与可串行化级别之下的级别结合. 顺序一致性与可串行化结合可以得到顺序可串行化, 如下:

顺序可串行化: 禁止出现环, 环由任意条 so、ww、wr、rw 边构成, (so∪ww∪wr∪rw)+.

可串行化级别本身保证了具有数据依赖的关系的事务不成环, 这就保证了事务间一定存在一个总序, 这个保证非常强. 事实上, 因果一致性与可串行化级别结合后, 由于可串行化本身已经提供了总序, 在满足因果关系的约束下, 二者结合已经达到了顺序一致性. 但是, 可串行化对没有数据依赖关系的事务并没有进行约束, 比如同一进程上顺序发生的事务、同一进程上顺序发生的对同一个数据对象的读. 前者是有进程上的先后关系, 但没有操作相同的数据项; 后者是进程上的先后关系且操作了相同数据项, 但操作是两次读, 不具备冲突依赖关系. 缺少这类约束会使分布式数据库产生某些违背因果的异常, 比如说违背单调读一致性. 引入 so 边与可串行化结合, 禁止的环可以完全覆盖因为违背因果的而产生的异常, 达到因果可串行化以上的顺序可串行化.

(4) 线性一致性的结合

线性一致性同样要求总序, 因此只能与可串行化级别结合, 同时需要加入真实时间顺序约束. 线性一致性与可串行化的结合也被称为严格可串行化, 可以推出定义如下.

严格可串行化: 禁止出现环, 环由任意条 so、ww、wr、rw 边构成, (rt∪ww∪wr∪rw)+.

线性一致性包含了顺序一致性, 因为 so 边含于 rt 实时时间, 在同一进程上的先后, 一定在真实时间上也存在先后关系. 同理, 顺序一致性补充了可串行化级别上进程上的因果, 但对于不同进程间先后顺序发生的事务没有做约束. 加入 rt 实时约束后, 环定义包含了线性一致性与可串行化二者的全部语义, 从数据库事务隔的角度来看, 在达到可串行化的基础上, 补充了真实时间中具有先后顺序的事务的约束, 防止了违背真实时间因果顺序的异常发生.

根据上述讨论, 本文通过按照不同隔离级别替换分布式一致性中的 vis, ar 等顺序边, 完成了对事务隔离级别和分布式一致性的统一建模和结合. 图 11 给出了结合的效果图, 详细的结合方法在后文表 8 中给出了总结. 图 11 中左侧为按从强到弱向下的隔离级别, 右侧为按一致性强弱的分布式一致性, 有向边将两个操作一致性关联起来, 强的一方需要保证弱的一方的所有特性.

线性一致性、顺序一致性只能与可串行结合, 其余级别, 即左右方框内的级别之间均可自由组合. 其中, PRAM 一下的 4 个一致性级别互不相干, 可以几个同时与隔离级别结合, 如, 同时满足单调读、读己之所写一致性的读已提交级别.

结合之后达到的一致性级别, 既满足事务属性中的某一隔离级别, 又满足分布式一致性中的一致性约束. 一致性模型的结合穷举了主流隔离级别和分布式一致性的数据异常, 提供了 NewSQL 权衡一致性、可扩展性和性能的级别划分, 并给出了独立于实现的方法定义.

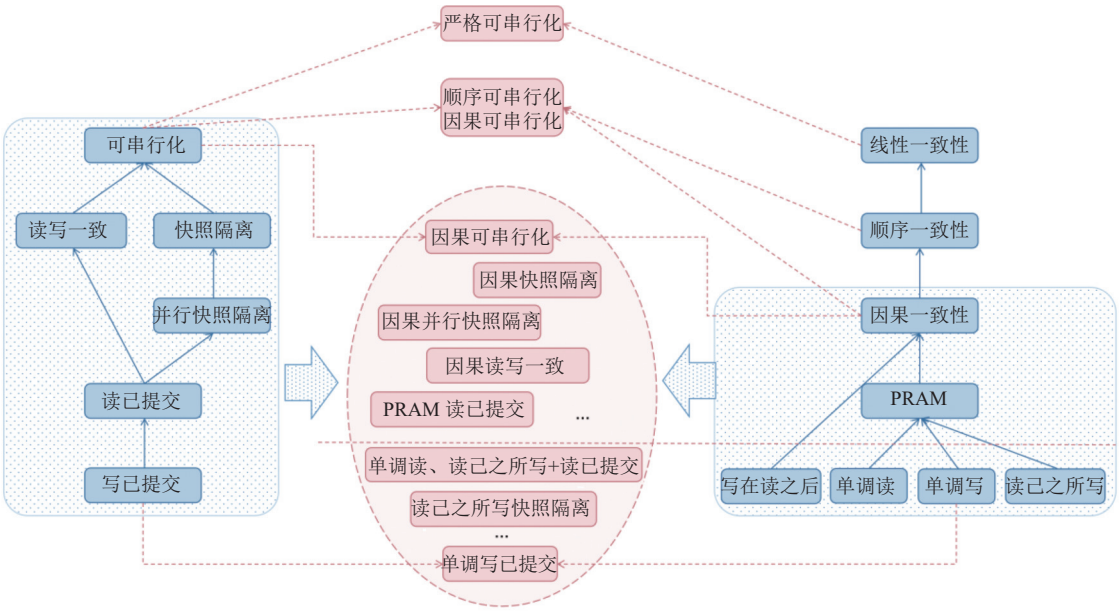


图 11 一致性模型

此外, 表 8 详细给出了一致性级别与隔离级别结合后, 数据依赖图中所需禁止的环结构. 表中一致性强度大致从左上角到右下角由强到弱分布.

表 8 多级一致性环结构

一致性级别	可串行化 (ww∪wr∪rw)+	快照隔离 ((ww∪wr);rw?)+	并行快照隔离 (ww∪wr)+;rw?	可重复读 (ww∪wr)+;rw?	读已提交 (ww∪wr)+	写已提交ww+
线性一致性 (rt∪vis∪ar∪$\neg vis^{-1}$)+	(rt∪ww∪wr∪rw)+	—	—	—	—	—
顺序一致性 (so∪vis∪ ar∪$\neg vis^{-1}$)+	(so∪ww∪wr∪rw)+	—	—	—	—	—
实时因果一致性 hb+∪rt+∪ar?∪$\neg vis^{-1}$?	(rt∪ww∪wr∪rw)+	((rt∪ww∪wr)∪rw?)+	(rt∪ww∪wr)+∪rw?	(rt∪ww∪wr)+∪rw?	(rt∪ww∪wr)+∪rw?	(rt∪ww)∪rw?
因果一致性 hb+∪ar?∪$\neg vis^{-1}$?	(so∪ww∪wr∪rw)+	((so∪ww∪wr)∪rw?)+	(so∪ww∪wr)+∪rw?	(so∪ww∪wr)+∪rw?	(so∪ww∪wr)+∪rw?	(so∪ww)+∪rw?
PRAMso+$_{\cup}\neg vis^{-1}$;vis?	(ww∪wr∪rw)+ so+∪(rw;(ww∪wr)) so+∪ww so+∪rw	((ww∪wr);rw?)+ so+∪(rw;(ww∪wr)) so+∪ww so+∪rw	(ww∪wr)+;rw? so+∪(rw;(ww∪wr)) so+∪ww so+∪rw	(ww∪wr)+;rw? so+∪(rw;(ww∪wr)) so+∪ww so+∪rw	(ww∪wr)+ so+∪(rw;(ww∪wr)) so+∪ww so+∪rw	ww+ so+∪(rw;(ww∪wr)) so+∪ww so+∪rw
写在读之后 so+∪(ar(ww);vis(wr))	(ww∪wr∪rw)+ so+∪(ww;wr)	((ww∪wr);rw?)+ so+∪(ww;wr)	(ww∪wr)+;rw? so+∪(ww;wr)	(ww∪wr)+;rw? so+∪(ww;wr)	(ww∪wr)+ so+∪(ww;wr)	ww+ so+∪(ww;wr)
单调读 so+∪($\neg vis^{-1}$(rw);vis)	(ww∪wr∪rw)+ so+∪(rw;(ww∪wr))	((ww∪wr);rw?)+ so+∪(rw;(ww∪wr))	(ww∪wr)+;rw? so+∪(rw;(ww∪wr))	(ww∪wr)+;rw? so+∪(rw;(ww∪wr))	(ww∪wr)+ so+∪(rw;(ww∪wr))	ww+ so+∪(rw;(ww∪wr))
单调写so+∪ar(ww)	(ww∪wr∪rw)+ so+∪ww	((ww∪wr);rw?)+ so+∪ww	(ww∪wr)+;rw? so+∪ww	(ww∪wr)+;rw? so+∪ww	(ww∪wr)+ so+∪ww	ww+ so+∪ww
读己之所写 so+∪$\neg vis^{-1}$(rw)	(ww∪wr∪rw)+ so+∪rw	((ww∪wr);rw?)+ so+∪rw	(ww∪wr)+;rw? so+∪rw	(ww∪wr)+;rw? so+∪rw	(ww∪wr)+ so+∪rw	ww+ so+∪rw

5.3 基于 TPC-C 的分布式数据库一致性级别检验

本节介绍了基于通用基准测试 TPC-C, 结合本文提出的统一模型, 来实现对去中心化分布式数据库系统进行一致性级别检验的方法. 首先, 我们利用统一模型对 TPC-C 基准进行静态分析, 分析在 TPC-C 基准中可能出现的异常环; 然后, 我们以 Google Spanner 的开源实现——CockroachDB 为实例, 进行 TPC-C 基准测试实验并进行一致性级别判断. 实验结果显示 CockroachDB 达到因果快照隔离级别, 但达不到实时因果快照隔离级别.

(1) TPC-C 基准静态分析

TPC-C 基准[39]模拟了仓库订单处理场景. TPC-C 包括 5 种事务, 其中 NewOrder、Payment 和 Delivery 为读写事务, Stock-Level 和 Order-Status 为只读事务. 每个仓库的数据量约为 100M, 可以通过设置仓库数来调整数据量大小.

基于本文提出的分布式数据库多级一致性模型, 我们根据 TPC-C 基准中定义的 5 种事务的 SQL 语句, 静态分析了这些事务间存在依赖边, 分析得到的 TPC-C 静态混合依赖图如图 12 所示.

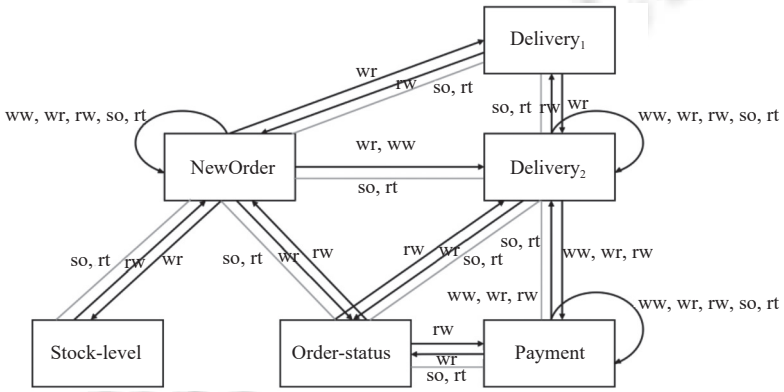


图 12 TPC-C 静态混合依赖图

参考 Fekete 等人[40]的工作, 首先对 wr/rw/ww 依赖进行分析. 我们对事务内的 SQL 语句进行抽象. 当 SQL 语句对同一行数据进行操作时 (即使是不同属性), 我们判断 SQL 语句对同一数据对象进行操作. 例如, 事务 Payment 的存储过程的第 1 句 SQL:

EXEC SQL UPDATE warehouse SET w_ytd = w_ytd + :h_amount WHERE w_id=:w_id;

可抽象为 W(w_id=:w_id), w_id 是仓库 warehouse 的 id 标识, 也就是说, 若有两个处于同一仓库的 Payment 事务 A、B, 则可能发生 $A \overset{ww}{\rightarrow} B$ 或者 $B \overset{ww}{\rightarrow} A$. 在静态分析图中, 表示为一条从 Payment 事务指向自身的 ww 边.

进一步, 我们对 so 边、rt 边进行分析. 对于任意两个事务, 都有可能发生在同一进程上先后进行, 由于 so 边不对操作的数据项做约束, 因此可以在任意两个事务间增加 so 边和 rt 边. 如图 12 中所示, 可以发现, 图中包括了 $((rt \cup ww \cup wr) \cup rw?)+$、$((so \cup ww \cup wr) \cup rw?)+$、$(so \cup ww \cup wr)+ \cup rw?$ 等环, 因此 TPC-C 基准可以检测实时因果快照隔离级别及以下的一致性级别.

(2) CockroachDB 实验分析

我们将 CockroachDB 部署在由 3 台机器组成的内部集群上, 每个节点都包含一个协调器和一个数据节点. 每台机器使用 CentOS 7.4 操作系统, 配备有两个 Intel(R) Xeon(R) Platinum 8276 CPU (28 核×2 HT)、8×128 GB DRAM 和 3 TB NVMe SSD.

我们在 CockroachDB 上进行了 TPC-C 实验, 通过对数据库日志进行分析, 构建混合依赖图. 在实验构建的依赖图中, 通关识别环判定数据库执行处于的一致性级别. 对 CockroachDB 进行基准测试后, 从日志构建的混合依赖图中我们识别了一类环. 此类环由 rt 边及其他边组成的环, 图 13 是我们抽象出的一个典型例子.

例 2. 图 13 展示了 CockroachDB 中出现的一个异常环. 假设有这样一个场景: 一位用户 A 创建了一个新的订单, 新订单调用 NewOrder 事务, 并在新订单表 new_orders 中新增一条记录 NO.oid=1, 同时, 订单的创建也对库存表 Stock 进行了操作, 减少了相应商品的库存 (根据订单明细表, 记录为 OL.oid=1). 此时, 一名仓库管理员查看本

地的库存信息, 调用了 Stock-Level 事务, 本地恰好有因订单 NO.oid=1 商品库存变动. 管理查看完库存信息后. 管理员又对用户 A 的订单状态进行了查看, 却发现用户 A 最新的订单是很久之前的, 也就是说用户 A 近期并没有新订单, 那么刚才查看的库存变动到从何而来? 我们分析由于 CockroachDB 没有对实时时间进行约束, 因此有可能会出现此类问题. 当 Order_Status 事务的协调器时钟较早时, 可能会判断 NewOrder 事务写入的 No.oid=1 对当前不可见, 导致没有读到已经完成的事务, 造成异常.

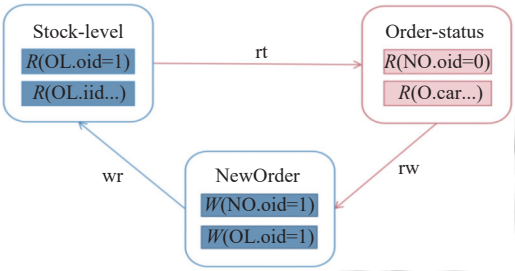


图 13 CockroachDB 异常环

(1) 实验结果: CockroachDB 实验中出现的异常环在因果快照隔离级别是不被禁止的, 在实时因果快照隔离级别则会被禁止, 因此, CockroachDB 在 TPC-C 基准下, 达到了因果快照隔离级别.

(2) 实验结果分析: 一方面, CockroachDB 使用混合逻辑时钟 (hlc) 机制, 缺乏全局时钟, 因此, 在多协调器的分布式系统中, CockroachDB 由于不支持实时约束, 不能达到线性一致性或实时因果一致性; 另一方面, 尽管 CockroachDB 声明达到可串行化级别, 但是由于 TPC-C 负载的限制 (最高只会出现实时因果快照隔离级别以下的异常), CockroachDB 的隔离级别最高只能证明达到快照隔离. 因此, 实验证明: 在 TPC-C 基准下 CockroachDB 达到了因果快照隔离级别, 没有达到实时因果快照隔离级别, 与理论分析结果一致.

5.4 可视化呈现

我们给出了一个可视化展示的例子, 通过对日志进行抽象分析可以来判断数据库运行所处的一致性级别. 分析过程中, 我们将日志中的读写结果抽象转化为读写序列, 结合额外获取的执行事务的进程信息. 可以通过我们开发的可视化分析工具来构建混合依赖图以展示受测数据库的一致性级别.

如图 14 所示, 按照不同进程输入日志中抽象出的历史序列后 (左边输入框), 根据不同进程、数据版本信息, 生成一个混合依赖图 (中间). 根据该混合依赖图, 可以识别图中所包含的环, 从而对数据库事务运行的级别进行判断. 图 14 中给出了一个简单示例, 通过对混合依赖图进行识别, 可以识别出右侧的环. 此环由 wr∪rw;rw 或 so∪rw;rw 构成, 被线性可串行化、顺序可串行化、实时因果可串行化所禁止. 因此, 此运行得到此日志的分布式数据库一致性级别不会超过因果可重复读级别.

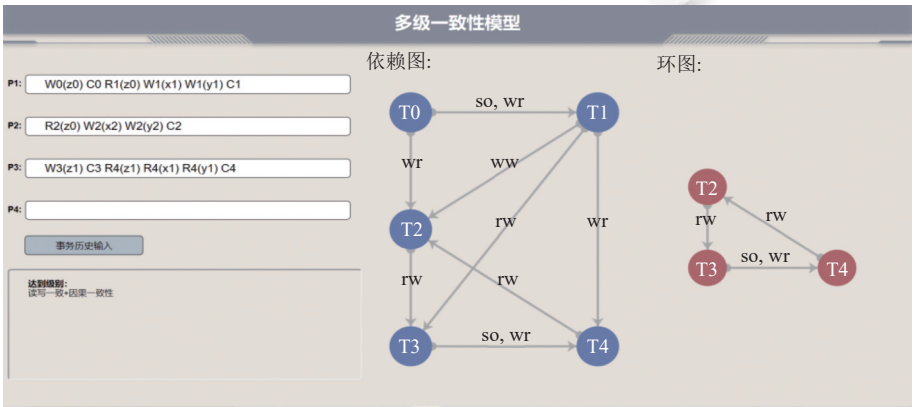


图 14 一致性模型可视化

对于实际中的分布式数据库, 可以分析日志流水得出数据库日常使用中可能产生的异常, 也可以分析业务逻辑对业务中可能出现的异常进行预先判断, 以此作为指标选择合适的数据库运行级别.

6 结 论

在本文中, 我们对分布式一致性理论和事务隔离级别进行了统一建模, 并在理论上给出了它们之间所有可能的组合效果. 我们提出了分布式一致性理论在事务粒度的多级环定义, 这些定义能够在分布式数据库的新架构下弥补原有隔离级别理论的缺失. 据我们所知, 本文是第一个借助环定义来结合多级一致性与隔离级别的理论研究工作. 我们提出了一套完整的统一模型, 根据混合依赖图精确表征新架构下的分布式数据库系统的一致性级别. 我们对 TPC-C 基准进行了静态分析, 发现其存在因果快照隔离下的异常, 然后, 我们使用 TPC-C 负载对分布式数据库 CockroachDB 进行实验测试, 结果表明 CockroachDB 达到了因果快照隔离, 而不满足实时因果快照隔离. 基于本文提出的框架体系, 继续探索最终一致性等更多一致性级别及其特征, 是我们的未来工作.

致谢 对参与本文讨论的老师、同学以及对本文提出宝贵建议的匿名评审老师表示衷心的感谢.

References:

[1] Cui B, Gao J, Tong YX, Xu JQ, Zhang DX, Zou L. Progress and trend in novel data management system. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 164–193 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5646.htm [doi: 10.13328/j.cnki.jos.005646]

[2] Du XY, Lu W, Zhang F. History, present, and future of big data management systems. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1): 127–141 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5644.htm [doi: 10.13328/j.cnki.jos.005644]

[3] Peng D, Dabek F. Large-scale incremental processing using distributed transactions and notifications. In: Proc. of the 9th USENIX Conf. on Operating Systems Design and Implementation. Vancouver: USENIX Association, 2010. 251–264.

[4] Corbett JC, Dean J, *et al*. Spanner: Google's globally distributed database. ACM Trans. on Computer Systems, 2013, 31(3): 8. [doi: 10.1145/2491245]

[5] OceanBase. 2021. https://www.oceanbase.com

[6] Lu W, Zhao ZH, Wang XY, Li HX, Zhang ZM, Shui ZY, Ye S, Pan AQ, Du XY. A lightweight and efficient temporal database management system in TDSQL. Proc. of the VLDB Endowment, 2019, 12(12): 2035–2046. [doi: 10.14778/3352063.3352122]

[7] Huang DX, Liu Q, Cui Q, *et al*. TiDB: A raft-based HTAP database. Proc. of the VLDB Endowment, 2020, 13(12): 3072–3084. [doi: 10.14778/3415478.3415535]

[8] Taft R, Sharif I, Matei A, *et al*. CockroachDB: The resilient geo-distributed SQL database. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: Association for Computing Machinery, 2020. 1493–1509. [doi: 10.1145/3318464.3386134]

[9] Herlihy MP, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Trans. on Programming Languages and Systems, 1990, 12(3): 463–492. [doi: 10.1145/78969.78972]

[10] Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. on Computers, 1979, C-28(9): 690–691. [doi: 10.1109/TC.1979.1675439]

[11] Ahamad M, Neiger G, Burns JE, Kohli P, Hutto PW. Causal memory: Definitions, implementation, and programming. Distributed Computing, 1995, 9(1): 37–49. [doi: 10.1007/BF01784241]

[12] Terry D. Replicated data consistency explained through baseball. Communications of the ACM, 2013, 56(12): 82–89. [doi: 10.1145/2500500]

[13] Pavlo A, Aslett M. What's really new with NewSQL? ACM SIGMOD Record, 2016, 45(2): 45–55. [doi: 10.1145/3003665.3003674]

[14] Ren K, Li D, Abadi DJ. SLOG: Serializable, low-latency, geo-replicated transactions. Proc. of the VLDB Endowment, 2019, 12(11): 1747–1761. [doi: 10.14778/3342263.3342647]

[15] Zhang Y, Power R, Zhou SY, Sovran Y, Aguilera MK, Li JY. Transaction chains: Achieving serializability with low latency in geo-distributed storage systems. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. Pennsylvania: Association for Computing Machinery, 2013. 276–291. [doi: 10.1145/2517349.2522729]

[16] Daudjee K, Salem K. Lazy database replication with ordering guarantees. In: Proc. of the 20th Int'l Conf. on Data Engineering. Boston: IEEE, 2004. 424–435. [doi: 10.1109/ICDE.2004.1320016]

[17] Cerone A, Gotsman A, Yang H. Algebraic laws for weak consistency. In: Proc. of the 28th Int'l Conf. on Concurrency Theory. Berlin: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. 26: 1–26: 18. [doi: 10.4230/LIPIcs.CONCUR.2017.26]

[18] Szekeres A, Zhang I. Making consistency more consistent: A unified model for coherence, consistency and isolation. In: Proc. of the 5th Workshop on the Principles and Practice of Consistency for Distributed Data. Porto: Association for Computing Machinery, 2018. 7. [doi: 10.1145/3194261.3194268]

[19] Shapiro M, Ardekani MS, Petri G. Consistency in 3D. In: Proc. of the 27th Int'l Conf. on Concurrency Theory. Québec: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. 3: 1–3: 14. [doi: 10.4230/LIPIcs.CONCUR.2016.3]

[20] Gray J, Reuter A. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann, 1992.

[21] American National Standards Institute. ANSI X3.135-1992 American national standard for information systems—Database language-SQL. New York: American National Standards Institute, 1992.

[22] Berenson H, Bernstein P, Gray J, Melton J, O'Neil E, O'Neil P. A critique of ANSI SQL isolation levels. ACM SIGMOD Record, 1995, 24(2): 1–10. [doi: 10.1145/568271.223785]

[23] Adya A, Liskov B, O'Neil P. Generalized isolation level definitions. In: Proc. of the 16th Int'l Conf. on Data Engineering. San Diego: IEEE, 2000. 67–78. [doi: 10.1109/ICDE.2000.839388]

[24] Kung HT, Robinson JT. On optimistic methods for concurrency control. ACM Trans. on Database Systems, 1981, 6(2): 213–226. [doi: 10.1145/319566.319567]

[25] Greenwald R, Stackowiak R, Stern J. Oracle Essentials: Oracle Database 12c. O'Reilly Media, Inc. 2013.

[26] Haderle DJ, Jackson RD. IBM database 2 overview. IBM Systems Journal, 1984, 23(2): 112–125.

[27] Anley C. Advanced SQL injection in SQL server applications. Technical Report, NGSSoftware. 2002. https://priv.gg/e/Hacking%20-%20Advanced%20SQL%20Injection.pdf

[28] Adya A. Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions [Ph.D. Thesis]. Cambridge: Massachusetts Institute of Technology, 1999.

[29] Saeida Ardekani M, Sutra P, Shapiro M. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In: Proc. of the 32nd IEEE Int'l Symp. on Reliable Distributed Systems. Braga: IEEE, 2013. 163–172. [doi: 10.1109/SRDS.2013.25]

[30] Sovran Y, Power R, Aguilera MK, Li JY. Transactional storage for geo-replicated systems. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles. Cascais: Association for Computing Machinery, 2011. 385–400. [doi: 10.1145/2043556.2043592]

[31] Lipton RJ, Sandberg JS. PRAM: A Scalable Shared Memory. Research Report, Princeton: Princeton University, 1988.

[32] Viotti P, Vukolić M. Consistency in non-transactional distributed storage systems. ACM Computing Surveys, 2017, 49(1): 19. [doi: 10.1145/2926965]

[33] Burckhardt S. Principles of Eventual Consistency. Hanover: Now Publishers Inc., 2014.

[34] Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978, 21(7): 558–565. [doi: 10.1145/359545.359563]

[35] Birman K, Schiper A, Stephenson P. Lightweight causal and atomic group multicast. ACM Trans. on Computer Systems, 1991, 9(3): 272–314. [doi: 10.1145/128738.128742]

[36] Hadzilacos V, Toueg S. A modular approach to fault-tolerant broadcasts and related problems. Technical Report, TR 94-1425, Ithaca: Cornell University, 1994.

[37] Rajsbaum S. ACM SIGACT news distributed computing column 5. ACM SIGACT News, 2001, 32(4): 34–58. [doi: 10.1145/568425.568433]

[38] Brzezinski J, Sobaniec C, Wawrzyniak D. Session guarantees to achieve PRAM consistency of replicated shared objects. In: Proc. of the 5th Int'l Conf. on Parallel Processing and Applied Mathematics (PPAM). Czestochowa: Springer, 2004. 1–8. [doi: 10.1007/978-3-540-24669-5_1]

[39] Leutenegger ST, Dias D. A modeling study of the TPC-C benchmark. ACM SIGMOD Record, 1993, 22(2): 22–31. [doi: 10.1145/170036.170042]

[40] Fekete A, Liarokapis D, O'Neil EJ, O'Neil P, Shasha D. Making snapshot isolation serializable. ACM Trans. on Database Systems, 2005, 30(2): 492–528. [doi: 10.1145/1071610.1071615]

附中文参考文献:

[1] 崔斌, 高军, 童咏昕, 许建秋, 张东祥, 邹磊. 新型数据管理系统研究进展与趋势. 软件学报, 2019, 30(1): 164–193. http://www.jos.org.

cn/1000-9825/5646.htm [doi: 10.13328/j.cnki.jos.005646]
[2] 杜小勇, 卢卫, 张峰. 大数据管理系统的历史、现状与未来. 软件学报, 2019, 30(1): 127−141. http://www.jos.org.cn/1000-9825/5644.htm [doi: 10.13328/j.cnki.jos.005644]

水治禹(1997−), 女, 硕士生, CCF 学生会员, 主要研究领域为分布式数据库系统, 事务处理.

何粤阳(1997−), 女, 硕士生, CCF 学生会员, 主要研究领域为数据库并发控制技术, 数据库隔离级别, 分布式系统一致性.

卢卫(1981−), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为数据库基础理论, 大数据系统研制, 时空背景下的查询处理, 云数据库系统和应用.

张孝(1972−), 男, 博士, 教授, 博士生导师, CCF 高级会员. 主要研究领域为数据库体系, 大数据管理与分析, 基准测试.

赵展浩(1995−), 男, 博士生, CCF 学生会员, 主要研究领域为分布式数据库系统, 事务处理.

杜小勇(1963−), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为高性能数据库, 智能信息检索, 非结构化数据管理.