



(12)发明专利申请

(10)申请公布号 CN 108337911 A

(43)申请公布日 2018.07.27

(21)申请号 201680031962.0

(74)专利代理机构 北京林达刘知识产权代理事

(22)申请日 2016.03.22

务所(普通合伙) 11277

(30)优先权数据

代理人 刘新宇

62/141,388 2015.04.01 US

(51)Int.Cl.

(85)PCT国际申请进入国家阶段日

G06F 17/30(2006.01)

2017.11.30

(86)PCT国际申请的申请数据

PCT/US2016/023554 2016.03.22

(87)PCT国际申请的公布数据

W02016/160416 EN 2016.10.06

(71)申请人 起元技术有限责任公司

地址 美国马萨诸塞州

(72)发明人 C·W·斯坦菲尔

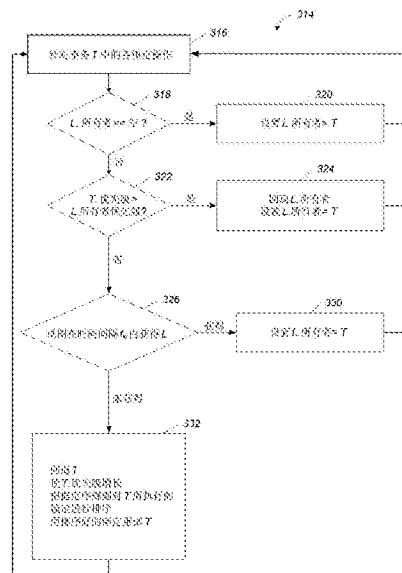
权利要求书5页 说明书13页 附图5页

(54)发明名称

在分布式计算系统中处理数据库事务

(57)摘要

在包括多个处理模块(110)的分布式计算系统(102)中处理事务包括：将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统(112)中，其中所述数据项是根据定序规则而完全定序的，并且所述进程中的至少一些运行在不同的处理模块上；以及使用多个多进程来处理事务。使用多个多进程来处理事务包括：接收用于访问所述数据存储系统中所存储的数据项的请求的集合(其中所述请求按照第一顺序排列)；如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁；以及如果在所述第一时间间隔内未获得所述锁中的任一锁，则重新开始正处理的事务。



1. 一种用于在包括多个处理模块的分布式计算系统中处理事务的方法，所述方法包括：

将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中，其中所述数据项是根据定序规则而完全定序的，并且所述进程中的至少一些运行在不同的处理模块上；以及

使用多个所述多进程来处理事务，其中使用多个所述多进程之一来处理事务包括：

接收用于访问所述数据存储系统中所存储的数据项的请求的集合，其中所述请求按照第一顺序排列，

如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁，

如果在所述第一时间间隔内未获得所述锁中的任一锁，则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序，以及

重新开始正处理的事务，包括以所述第二顺序顺次获得数据项上的锁。

2. 根据权利要求1所述的方法，其中，重新开始正处理的事务包括回退该事务，从而释放所述数据项上的任何现有锁。

3. 根据权利要求1至2中任一项所述的方法，其中，所述第二顺序至少与针对在所述第一时间间隔内获得了锁的数据项的所述定序规则一致。

4. 根据权利要求1至3中任一项所述的方法，其中，所述处理还包括：基于在所述第一时间间隔内获得了锁的数据项的定序规则来存储标识位置的信息，以及确定所述第二顺序包括至少部分地基于所存储的标识位置的信息来确定所述第二顺序。

5. 根据权利要求1至4中任一项所述的方法，其中，确定所述第二顺序包括根据所述定序规则来对用于获得锁的操作的列表进行排序。

6. 根据权利要求1至5中任一项所述的方法，其中，所述处理还包括判断所述第一顺序是否与所述定序规则一致，以及在所述第一顺序与所述定序规则一致的情况下等待比所述第一时间间隔更长的时间，以获得在所述第一时间间隔内未获得锁的任何数据项上的锁。

7. 根据权利要求1至6中任一项所述的方法，其中，所述处理还包括确定接收到所述请求的集合的进程相对于多个所述多进程中的其它进程的优先级。

8. 根据权利要求7所述的方法，其中，在确定所述优先级之后重新开始正处理的事务。

9. 根据权利要求1至8中任一项所述的方法，其中，在判断为所述请求的集合请求访问的至少一个数据项上的锁的状态已经改变之后，重新开始正处理的事务。

10. 根据权利要求1至5中任一项所述的方法，其中，所述处理还包括判断所述第一顺序是否与所述定序规则一致，以及在从判断为所述第一顺序与所述定序规则不一致起的第二时间间隔之后重新开始正处理的事务。

11. 根据权利要求10所述的方法，其中，所述第二时间间隔比所述第一时间间隔长。

12. 根据权利要求1至11中任一项所述的方法，其中，所述第一时间间隔小于1秒。

13. 根据权利要求1至12中任一项所述的方法，其中，所述处理还包括：保持所述数据项上获得的任何锁，直到提交、中止或重新开始正处理的事务为止；以及在提交、中止或重新开始正处理的事务的情况下释放所述数据项上的锁。

14. 根据权利要求1至13中任一项所述的方法，其中，所述数据项上的锁具有包括一个

未锁定状态以及一个或多个锁定状态的至少两个状态。

15. 根据权利要求14所述的方法,其中,获得数据项上的锁包括将与该数据项相关联的锁的状态改变为所述锁定状态之一。

16. 根据权利要求14至15中任一项所述的方法,其中,释放数据项上的锁包括将与该数据项相关联的锁的状态从所述锁定状态之一改变为所述未锁定状态。

17. 根据权利要求14至16中任一项所述的方法,其中,所述数据项上的锁具有仅许可对锁定的数据项的单处理完全访问的至少一个独占锁定状态。

18. 根据权利要求14至17中任一项所述的方法,其中,所述数据项上的锁具有仅许可对锁定的数据项的多进程只读访问的至少一个共享锁定状态。

19. 根据权利要求1至18中任一项所述的方法,其中,所述多进程中的至少一些彼此异步地运行。

20. 根据权利要求1至19中任一项所述的方法,其中,所述事务包括数据库事务,并且所述数据项是数据库中的记录。

21. 根据权利要求20所述的方法,其中,所述数据库是内存数据库。

22. 根据权利要求20至21中任一项所述的方法,其中,所述数据存储系统分布在所述数据库的多个节点中。

23. 根据权利要求1至22中任一项所述的方法,其中,多个所述多进程中的至少一些运行在所述处理模块中的不同处理模块上。

24. 一种计算机可读存储介质,其存储在由一个或多个计算机执行时致使一个或多个计算机进行用于在包括多个处理模块的分布式计算系统中处理事务的操作的指令,所述操作包括:

将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中,其中所述数据项是根据定序规则而完全定序的,并且所述进程中的至少一些运行在不同的处理模块上;以及

使用多个所述多进程来处理事务,其中使用多个所述多进程之一来处理事务包括:

接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,

如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,

如果在所述第一时间间隔内未获得所述锁中的任一锁,则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序,以及

重新开始正处理的事务,包括以所述第二顺序顺次获得数据项上的锁。

25. 一种用于处理事务的分布式计算系统,所述分布式计算系统包括:

多个处理模块;以及

数据存储系统,其被配置为存储数据项,所述数据存储系统是所述分布式计算系统中运行的多进程能够访问的,其中所述数据项根据定序规则而完全定序,并且所述进程中的至少一些运行在不同的处理模块上;

其中,多个所述多进程被配置为处理事务,并且使用多个所述多进程之一来处理事务包括:

接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,

如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,

如果在所述第一时间间隔内未获得所述锁中的任一锁,则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序,以及

重新开始正处理的事务,包括以所述第二顺序顺次获得数据项上的锁。

26.一种用于在包括多个处理模块的分布式计算系统中处理事务的方法,所述方法包括:

将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中,其中所述数据项是根据定序规则而完全定序的,并且所述进程中的至少一些运行在不同的处理模块上;以及

使用多个所述多进程来处理事务,其中使用多个所述多进程之一来处理事务包括:

接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,

如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,

如果在所述第一时间间隔内未获得所述锁中的任一锁,则判断所述第一顺序是否与所述定序规则一致,以及

如果所述第一顺序与所述定序规则不一致,则重新开始正处理的事务。

27.根据权利要求26所述的方法,其中,重新开始正处理的事务包括回退该事务,释放所述数据项上的任何现有锁,以及以与所述第一顺序不同的第二顺序顺次获得数据项上的锁。

28.根据权利要求26至27中任一项所述的方法,其中,所述第二顺序至少与针对在所述第一时间间隔内获得了锁的数据项的所述定序规则一致。

29.根据权利要求26至28中任一项所述的方法,其中,所述处理还包括:基于在所述第一时间间隔内获得了锁的数据项的定序规则来存储标识位置的信息,以及至少部分地基于所存储的标识位置的信息来确定所述第二顺序。

30.根据权利要求26至29中任一项所述的方法,其中,所述第二顺序是基于根据所述定序规则对用于获得锁的操作的列表进行排序来确定的。

31.根据权利要求26至30中任一项所述的方法,其中,所述处理还包括在所述第一顺序与所述定序规则一致的情况下等待比所述第一时间间隔更长的时间,以获得在所述第一时间间隔内未获得锁的任何数据项上的锁。

32.根据权利要求26至31中任一项所述的方法,其中,所述处理还包括确定接收到所述请求的集合的进程相对于多个所述多进程中的其它进程的优先级。

33.根据权利要求32所述的方法,其中,在确定所述优先级之后进行在所述第一顺序与所述定序规则不一致的情况下重新开始正处理的事务。

34.根据权利要求26至33中任一项所述的方法,其中,在判断为所述请求的集合请求访问的至少一个数据项上的锁的状态已经改变之后,进行在所述第一顺序与所述定序规则不

一致的情况下重新开始正处理的事务。

35. 根据权利要求26至33中任一项所述的方法,其中,在从判断为所述第一顺序与所述定序规则不一致起的第二时间间隔之后,进行在所述第一顺序与所述定序规则不一致的情况下重新开始正处理的事务。

36. 根据权利要求35所述的方法,其中,所述第二时间间隔比所述第一时间间隔长。

37. 根据权利要求26至36中任一项所述的方法,其中,所述第一时间间隔小于1秒。

38. 根据权利要求26至37中任一项所述的方法,其中,所述处理还包括:保持所述数据项上获得的任何锁,直到提交、中止或重新开始正处理的事务为止;以及在提交、中止或重新开始正处理的事务的情况下释放所述数据项上的锁。

39. 根据权利要求26至38中任一项所述的方法,其中,所述数据项上的锁具有包括一个未锁定状态以及一个或多个锁定状态的至少两个状态。

40. 根据权利要求39所述的方法,其中,获得数据项上的锁包括将与该数据项相关联的锁的状态改变为所述锁定状态之一。

41. 根据权利要求39至40中任一项所述的方法,其中,释放数据项上的锁包括将与该数据项相关联的锁的状态从所述锁定状态之一改变为所述未锁定状态。

42. 根据权利要求39至41中任一项所述的方法,其中,所述数据项上的锁具有仅许可对锁定的数据项的单处理完全访问的至少一个独占锁定状态。

43. 根据权利要求39至42中任一项所述的方法,其中,所述数据项上的锁具有仅许可对锁定的数据项的多进程只读访问的至少一个共享锁定状态。

44. 根据权利要求26至43中任一项所述的方法,其中,所述多进程中的至少一些彼此异步地运行。

45. 根据权利要求26至44中任一项所述的方法,其中,所述事务包括数据库事务,并且所述数据项是数据库中的记录。

46. 根据权利要求45所述的方法,其中,所述数据库是内存数据库。

47. 根据权利要求45至46中任一项所述的方法,其中,所述数据存储系统分布在所述数据库的多个节点中。

48. 根据权利要求26至47中任一项所述的方法,其中,多个所述多进程中的至少一些运行在所述处理模块中的不同处理模块上。

49. 一种计算机可读存储介质,其存储在由一个或多个计算机执行时致使一个或多个计算机进行用于在包括多个处理模块的分布式计算系统中处理事务的操作的指令,所述操作包括:

将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中,其中所述数据项是根据定序规则而完全定序的,并且所述处理中的至少一些运行在不同的处理模块上;以及

使用多个所述多进程来处理事务,其中使用多个所述多进程之一来处理事务包括:

接收用于访问所述数据存储系统中所存储的数据项的请求的集合,

其中所述请求按照第一顺序排列,

如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,

如果在所述第一时间间隔内未获得所述锁中的任一锁，则判断所述第一顺序是否与所述定序规则一致，以及

如果所述第一顺序与所述定序规则不一致，则重新开始正处理的事务。

50. 一种用于处理事务的分布式计算系统，所述分布式计算系统包括：

多个处理模块；以及

数据存储系统，其被配置为存储数据项，所述数据存储系统是所述分布式计算系统中运行的多进程能够访问的，其中所述数据项是根据定序规则而完全定序的，并且所述处理中的至少一些运行在不同的处理模块上；

其中多个所述多进程被配置为处理事务，并且使用多个所述多进程之一来处理事务包括：

接收用于访问所述数据存储系统中所存储的数据项的请求的集合，其中所述请求按照第一顺序排列，

如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁，

如果在所述第一时间间隔内未获得所述锁中的任一锁，则判断所述第一顺序是否与所述定序规则一致，以及

如果所述第一顺序与所述定序规则不一致，则重新开始正处理的事务。

在分布式计算系统中处理数据库事务

[0001] 相关申请的交叉引用

[0002] 本申请要求2015年4月1日提交的美国申请序列62/141,388的优先权。

技术领域

[0003] 本申请涉及在分布式计算系统中处理事务。特别地，本申请还涉及特别被配置为在多个数据处理模块和数据库的分布式网络中处理数据库事务的分布式计算系统、计算机实现方法和计算机可读存储介质，其中所述数据库例如可以分布在多个数据库节点中。

背景技术

[0004] 数据库是可以使用软件程序进行管理和查询的结构化的持久性数据集。事务数据库管理系统可以包括使用数据库“事务”对数据库中的数据进行操作（例如，存储和操纵）的关系数据库系统。一般来说，数据库事务代表数据库管理系统对数据库所进行的工作（包括一个或多个操作的）单个单元。为了确保数据库事务处理的可靠性，数据库事务必须是原子性的（即，事务，包括其一个或多个操作中的全部操作，必须完整地完成或者对任何事物都没有影响）、一致性的（即，事务必须将数据库从一个有效状态移动至另一有效状态）、隔离性的（即，并行执行的事务导致数据库中出现与串行执行事务的情况下将会导致的状态相同的状态），以及持久性的（即，与系统崩溃、错误和其它问题无关地，提交的事务将保持提交）。数据库事务的这组性质有时被称为“ACID”。数据库事务可以用于诸如在线事务处理（OLTP）系统等的产业系统，其可以用于对来自一个或多个服务器计算机处的客户端计算机的请求作出应答，或者可以用作建立诸如包括数据仓库系统的在线分析处理（OLAP）系统等的其它系统的基础。多种类型的产业可以使用这样的系统，其中这些产业包括例如涉及研究、医疗保健或信息服务等的产业。

[0005] 在一些情况下，在某些条件下，特别是在对于ACID性质的严格遵守具有挑战性的系统（例如，分布式系统）上，可以放宽ACID性质中的一个或多个。在其它情况下，事务保持ACID性质是很重要的，甚至在数据库分布在多个数据库节点中的情况下也是如此。如以下更详细地描述，在诸如分布式数据库等的分布式系统中，需要以避免节点中的死锁的方式实现用于支持任何ACID性质的任何分布式算法。然而，对于某些分布式算法（例如，用于原子提交的两阶段提交协议、或者用于并行控制的两阶段锁定协议），提供用以检测实际死锁情况的能力或者始终保证严格的死锁避免对系统功能性能有影响。

[0006] 传统地，用于检测、防止和/或避免诸如上述死锁等的死锁的集中式事务数据库方法是众所周知的。然而，在分布式数据库中，寻找用于检测和避免死锁的解决方案变得越来越困难，这是因为多个并行执行的事务的情况下存在较高的发生死锁的可能性。一些分布式数据库操作与分布式计算中的难以解决的问题相对应，特别是容错算法。对于某些类型的故障，分布式计算中的一些问题已被证明是不可能用容错分布式算法来解决的。

发明内容

[0007] 在一方面,一般来说,一种用于在包括多个处理模块的分布式计算系统中处理事务的方法包括:将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中,其中所述数据项是根据定序规则而完全定序的,并且所述进程中的至少一些运行在不同的处理模块上;以及使用多个所述多进程来处理事务,其中使用多个所述多进程之一来处理事务包括:接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,如果在所述第一时间间隔内未获得所述锁中的任一锁,则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序,以及重新开始正处理的事务,包括以所述第二顺序顺次获得数据项上的锁。

[0008] 方面可以包括以下特征中的一个或多个特征。

[0009] 重新开始正处理的事务包括回退该事务,从而释放所述数据项上的任何现有锁。

[0010] 所述第二顺序至少与针对在所述第一时间间隔内获得了锁的数据项的所述定序规则一致。

[0011] 所述处理还包括:基于在所述第一时间间隔内获得了锁的数据项的定序规则来存储标识位置的信息,以及确定所述第二顺序包括至少部分地基于所存储的标识位置的信息来确定所述第二顺序。

[0012] 确定所述第二顺序包括根据所述定序规则来对用于获得锁的操作的列表进行排序。

[0013] 所述处理还包括判断所述第一顺序是否与所述定序规则一致,以及在所述第一顺序与所述定序规则一致的情况下等待比所述第一时间间隔更长的时间,以获得在所述第一时间间隔内未获得锁的任何数据项上的锁。

[0014] 所述处理还包括确定接收到所述请求的集合的进程相对于多个所述多进程中的其它进程的优先级。

[0015] 在确定所述优先级之后重新开始正处理的事务。

[0016] 在判断为所述请求的集合请求访问的至少一个数据项上的锁的状态已经改变之后,重新开始正处理的事务。

[0017] 所述处理还包括判断所述第一顺序是否与所述定序规则一致,以及在从判断为所述第一顺序与所述定序规则不一致起的第二时间间隔之后重新开始正处理的事务。

[0018] 所述第二时间间隔比所述第一时间间隔长。

[0019] 所述第一时间间隔小于1秒。

[0020] 所述处理还包括:保持所述数据项上获得的任何锁,直到提交、中止或重新开始正处理的事务为止;以及在提交、中止或重新开始正处理的事务的情况下释放所述数据项上的锁。

[0021] 所述数据项上的锁具有包括一个未锁定状态以及一个或多个锁定状态的至少两个状态。

[0022] 获得数据项上的锁包括将与该数据项相关联的锁的状态改变为所述锁定状态之一。

[0023] 释放数据项上的锁包括将与该数据项相关联的锁的状态从所述锁定状态之一改变为所述未锁定状态。

[0024] 所述数据项上的锁具有仅许可对锁定的数据项的单处理完全访问的至少一个独占锁定状态。

[0025] 所述数据项上的锁具有仅许可对锁定的数据项的多进程只读访问的至少一个共享锁定状态。

[0026] 所述多进程中的至少一些彼此异步地运行。

[0027] 所述事务包括数据库事务，并且所述数据项是数据库中的记录。

[0028] 所述数据库是内存数据库。

[0029] 所述数据存储系统分布在所述数据库的多个节点中。

[0030] 多个所述多进程中的至少一些运行在所述处理模块中的不同处理模块上。

[0031] 在一方面，一般来说，一种计算机可读存储介质存储在由一个或多个计算机执行时致使一个或多个计算机进行用于在包括多个处理模块的分布式计算系统中处理事务的操作的指令，所述操作包括：将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中，其中所述数据项是根据定序规则而完全定序的，并且所述进程中的至少一些运行在不同的处理模块上；以及使用多个所述多进程来处理事务，其中使用多个所述多进程之一来处理事务包括：接收用于访问所述数据存储系统中所存储的数据项的请求的集合，其中所述请求按照第一顺序排列，如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁，如果在所述第一时间间隔内未获得所述锁中的任一锁，则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序，以及重新开始正处理的事务，包括以所述第二顺序顺次获得数据项上的锁。

[0032] 另一方面，一种用于处理事务的分布式计算系统，所述分布式计算系统包括：多个处理模块；以及数据存储系统，其被配置为存储数据项，所述数据存储系统是所述分布式计算系统中运行的多进程能够访问的，其中所述数据项根据定序规则而完全定序，并且所述进程中的至少一些运行在不同的处理模块上；其中，多个所述多进程被配置为处理事务，并且使用多个所述多进程之一来处理事务包括：接收用于访问所述数据存储系统中所存储的数据项的请求的集合，其中所述请求按照第一顺序排列，如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁，如果在所述第一时间间隔内未获得所述锁中的任一锁，则针对所述锁中的至少两个锁确定与所述定序规则一致的第二顺序，以及重新开始正处理的事务，包括以所述第二顺序顺次获得数据项上的锁。

[0033] 另一方面，一种用于在包括多个处理模块的分布式计算系统中处理事务的方法，所述方法包括：将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中，其中所述数据项是根据定序规则而完全定序的，并且所述进程中的至少一些运行在不同的处理模块上；以及使用多个所述多进程来处理事务，其中使用多个所述多进程之一来处理事务包括：接收用于访问所述数据存储系统中所存储的数据项的请求的集合，其中所述请求按照第一顺序排列，如果在第一时间间隔内获得所述数据项上的锁中的各锁，则以所述第一顺序顺次获得这些锁，如果在所述第一时间间隔内未获得所述锁中的任一锁，则判断所述第一顺序是否与所述定序规则一致，以及如果所述第一顺序与所述定序规则不一致，则重新开始正处理的事务。

[0034] 方面可以包括以下特征中的一个或多个。

[0035] 重新开始正处理的事务包括回退该事务，释放所述数据项上的任何现有锁，以及

以与所述第一顺序不同的第二顺序顺次获得数据项上的锁。

[0036] 所述第二顺序至少与针对在所述第一时间间隔内获得了锁的数据项的所述定序规则一致。

[0037] 所述处理还包括：基于在所述第一时间间隔内获得了锁的数据项的定序规则来存储标识位置的信息，以及至少部分地基于所存储的标识位置的信息来确定所述第二顺序。

[0038] 所述第二顺序是基于根据所述定序规则对用于获得锁的操作的列表进行排序来确定的。

[0039] 所述处理还包括在所述第一顺序与所述定序规则一致的情况下等待比所述第一时间间隔更长的时间，以获得在所述第一时间间隔内未获得锁的任何数据项上的锁。

[0040] 所述处理还包括确定接收到所述请求的集合的进程相对于多个所述多进程中的其它进程的优先级。

[0041] 在确定所述优先级之后进行在所述第一顺序与所述定序规则不一致的情况下重新开始正处理的事务。

[0042] 在判断为所述请求的集合请求访问的至少一个数据项上的锁的状态已经改变之后，进行在所述第一顺序与所述定序规则不一致的情况下的重新开始正处理的事务。

[0043] 在从判断为所述第一顺序与所述定序规则不一致起的第二时间间隔之后，进行在所述第一顺序与所述定序规则不一致的情况下的重新开始正处理的事务。

[0044] 所述第二时间间隔比所述第一时间间隔长。

[0045] 所述第一时间间隔小于1秒。

[0046] 所述处理还包括：保持所述数据项上获得的任何锁，直到提交、中止或重新开始正处理的事务为止；以及在提交、中止或重新开始正处理的事务的情况下释放所述数据项上的锁。

[0047] 所述数据项上的锁具有包括一个未锁定状态以及一个或多个锁定状态的至少两个状态。

[0048] 获得数据项上的锁包括将与该数据项相关联的锁的状态改变为所述锁定状态之一。

[0049] 释放数据项上的锁包括将与该数据项相关联的锁的状态从所述锁定状态之一改变为所述未锁定状态。

[0050] 所述数据项上的锁具有仅许可对锁定的数据项的单处理完全访问的至少一个独占锁定状态。

[0051] 所述数据项上的锁具有仅许可对锁定的数据项的多进程只读访问的至少一个共享锁定状态。

[0052] 所述多进程中的至少一些彼此异步地运行。

[0053] 所述事务包括数据库事务，并且所述数据项是数据库中的记录。

[0054] 所述数据库是内存数据库。

[0055] 所述数据存储系统分布在所述数据库的多个节点中。

[0056] 多个所述多进程中的至少一些运行在所述处理模块中的不同处理模块上。

[0057] 另一方面，一般来说，一种计算机可读存储介质存储在由一个或多个计算机执行时致使一个或多个计算机进行用于在包括多个处理模块的分布式计算系统中处理事务的

操作的指令,所述操作包括:将数据项存储在所述分布式计算系统中运行的多进程能够访问的数据存储系统中,其中所述数据项是根据定序规则而完全定序的,并且所述处理中的至少一些运行在不同的处理模块上;以及使用多个所述多进程来处理事务,其中使用多个所述多进程之一来处理事务包括:接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,如果在所述第一时间间隔内未获得所述锁中的任一锁,则判断所述第一顺序是否与所述定序规则一致,以及如果所述第一顺序与所述定序规则不一致,则重新开始正处理的事务。

[0058] 另一方面,一般来说,一种用于处理事务的分布式计算系统包括:多个处理模块;以及数据存储系统,其被配置为存储数据项,所述数据存储系统是所述分布式计算系统中运行的多进程能够访问的,其中所述数据项是根据定序规则而完全定序的,并且所述处理中的至少一些运行在不同的处理模块上;其中多个所述多进程被配置为处理事务,并且使用多个所述多进程之一来处理事务包括:接收用于访问所述数据存储系统中所存储的数据项的请求的集合,其中所述请求按照第一顺序排列,如果在第一时间间隔内获得所述数据项上的锁中的各锁,则以所述第一顺序顺次获得这些锁,如果在所述第一时间间隔内未获得所述锁中的任一锁,则判断所述第一顺序是否与所述定序规则一致,以及如果所述第一顺序与所述定序规则不一致,则重新开始正处理的事务。

[0059] 方面可以包括以下优点中的一个或多个优点。

[0060] 在其它优点中,这些方面防止分布式数据库中的死锁,同时保持系统性能。

[0061] 方面确保事务不会由于被连续回退而饥饿(starve)。

[0062] 方面允许依赖于分布式数据库中的高效事务数据处理的物流过程或产业过程可以直接受益于减少的死锁的数量或更快的事务数据处理。

[0063] 方面允许根据事务所指定的顺序来获得数据项上的锁,并且在未成功获得数据项上的锁的情况下恢复为已知的无死锁锁定顺序。

[0064] 根据以下描述和权利要求书,本发明的其它特征和优点将变得明显。

附图说明

[0065] 图1是包括分布式数据库系统的数据处理系统的框图。

[0066] 图2示出两个数据库事务之间的死锁。

[0067] 图3是先占式死锁避免算法的流程图。

[0068] 图4示出图3中的算法在第一种潜在死锁情况中的应用。

[0069] 图5示出图3中的算法在第二种潜在死锁情况中的应用。

具体实施方式

[0070] 1概述

[0071] 图1示出可以使用死锁避免技术的数据处理系统100的示例。系统100包括通过通信网络106(例如,WAN、LAN或者多处理器系统网络或片上网络等)与M个数据库客户端104进行通信的分布式数据库系统102。

[0072] 分布式数据库系统102包括分配了数据库D的片段D_n的N个节点108。各节点108包

括存储了数据库D的片段的数据存储装置112、以及作为(例如,由服务器计算机、处理器或处理器核控制的)处理模块的用于管理数据存储装置112上的数据库的片段的数据库管理器110。针对给定节点108的数据库管理器110还用作数据存储装置112上的数据库的片段与该节点108外部的诸如客户端104和其它节点108等的实体之间的接口。尽管图1中没有明确示出,但在一些示例中,各数据库管理器110包括负责管理由数据库管理器110管理的数据存储装置112上所存储的数据库的片段的数据处理器、用于处理要求访问多于一个节点108上的数据库片段的请求的应用处理器、以及用于与客户端104和其它节点108进行通信的通信软件。

[0073] 在操作中,客户端104指定用于在数据库D上执行的一个或多个数据库事务。客户端104所指定的事务通过通信网络106而被发送至节点108的一个或多个数据库管理器110。在事务到达N个节点108中的第n个数据库管理器110的情况下,第n个数据库管理器110评价该事务,并且生成用于在由第n个数据库管理器110管理的第n个数据存储装置112上所存储的数据库的片段上执行该事务的访问计划(例如,查询计划)。

[0074] 在一些示例中,在事务访问多个节点108上所存储的数据库的多个片段的情况下,第n个数据库管理器110将事务和/或访问计划转发至多个节点108的数据库管理器110。在其它示例中,事务所源自的客户端104将事务发送至完成事务所需的适当节点108。在又一示例中,事务所源自的客户端104将事务发送至被指定为领导节点的一个节点108,并且该领导节点将适当的访问计划发送至完成该事务所需的适当节点108。

[0075] 利用针对适当节点108的适当位置中的一个或多个数据库事务的访问计划,一个或多个事务可以执行并且访问数据库。与传统的集中式事务数据库的情况一样,一个或多个事务可能彼此冲突,从而导致一些事务成功地完成而其它事务失败,此时这些失败的事务被迫撤销其更改并重新开始。

[0076] 2死锁

[0077] 这些事务之间出现的一个潜在的冲突是死锁。一般来说,死锁是两个或更多个并行执行的事务各自需要另一事务已独占访问的资源、并且在完成之前必须等待另一事务释放资源的情况。如果不加限制,则这两个或更多个并行执行的事务由于永远无法访问它们所需的资源因而都不会完成。

[0078] 参考图2,示出发生死锁的情况的典型示例。在该示例中,两个事务T₁和T₂在事务数据库中所存储的数据元素x和y上并行操作。T₁包括用于读取数据元素y的值并且将值写到数据元素x的操作。T₂包括用于读取数据元素x的值并且将值写到数据元素y的操作。如这里所使用的,“锁”是用于防止除获取特定数据元素上的锁的实体以外的任何实体并行访问该特定数据元素的任何机制。根据正进行/防止的访问的类型,可以存在多种类型的锁。例如,“读锁”是用于防止任何其它实体读取数据元素的锁,以及“写锁”是用于防止任何其它实体写入数据元素的锁。

[0079] 在第一时间步(1)处,T₁发出read_lock(y)命令以获得数据元素y上的读锁R_{T1} 113。在第二时间步(2)处,T₁发出read(y)命令以读取y的值。在第三时间步(3)处,T₂发出read_lock(x)命令以获得数据元素x上的读锁R_{T2} 115。在第四时间步(4)处,T₂发出read(x)命令以读取x的值。

[0080] 在第五时间步(5)处,T₁发出write_lock(x)命令以尝试获得数据元素x上的写锁

$W_{T1} 117$ 。然而, T_2 已经具有数据元素 x 上的读锁, 因此 T_1 在其可以获得写锁 $W_{T1} 117$ 之前必须等待, 直到 T_2 已经释放数据元素 x 上的读锁为止。同样, 在第六时间步(6)处, T_2 发出 `write_lock(y)` 命令以尝试获得数据元素 y 上的写锁 $W_{T2} 119$ 。然而, T_1 已经具有数据元素 y 上的读锁, 因此 T_2 在其可以获得写锁 $W_{T2} 119$ 之前必须等待, 直到 T_1 已经释放数据元素 y 上的读锁为止。

[0081] 此时, 发生了死锁, 这是因为 T_1 和 T_2 都不会释放读锁直到已经完成其事务为止, 但是为了使任一事务取得进展, 另一事务必须释放其读锁。一旦发生了诸如图2所示的死锁等的死锁, 唯一能够取得进展的方式是通过使两个事务之一回退并且使得另一事务能够完成来打破死锁。

[0082] 3死锁避免

[0083] 如上所述, 传统地, 用于检测、防止和/或避免诸如上述的死锁等的死锁的集中式事务数据库方法是众所周知的。然而, 在分布式数据库中, 寻找用于检测和避免死锁的解决方案变得越来越困难, 这是因为在多个并行执行的事务的情况下存在较高的发生死锁的可能性, 其中多个并行执行的事务各自在分布式数据库系统102的多个不同节点108所存储的数据上操作。一些分布式数据库操作与分布式计算中的难以解决的问题相对应, 特别是容错算法。对于某些类型的故障, 分布式计算中的一些问题已被证明是不可能用容错分布式算法来解决的。

[0084] 参考图3, 在一些示例中, 分布式数据库系统102使用“先占式死锁避免”算法314来避免死锁, 其中该先占式死锁避免算法314在存在发生死锁或者将发生死锁(即使实际尚未发生死锁)的情况下撤销(或回退)事务。另外, 如以上更详细地描述, 算法314能够通过将要求与定序规则一致的需求仅限制于不快速获得锁的特定情况, 来减少否则将施加在定序锁定方案中的一些开销。以这种方式, 依赖于分布式数据库中的高效事务数据处理的物流过程或产业过程可以直接受益于减少的死锁的数量或更快的事务数据处理。

[0085] 先占式死锁避免算法314的先决条件是分布式数据库中所存储的数据项根据定序规则进行完全定序。“完全定序”(或“线性定序”)的项目集合是如下的一种项目集合, 其中在该项目集合中, 集合的项目对中的每个项目根据定序规则(也称为“可比较性”和“总体性”)相对于该对中的另一项目具有定义好的顺序。在“部分定序”的集合中, 相比之下, 并非每个项目对都必须相对于彼此而具有定义好的顺序。完全定序的集合还具有其它性质(例如, 反对称性和传递性), 其中这些性质对于部分定序的集合也是如此。

[0086] 这种完全定序的定序规则的一个简单示例是将一系列号码中的不同号码分配至数据库中的各数据项。例如, 数据库中的第一数据元素将被分配号码1, 数据库中的第二数据元素将被分配号码2, 等等。当然, 可以使用其它更复杂的定序规则。例如, 各节点108可以独立地将号码分配给其数据项, 以使得任何给定节点上的号码在各自的数据库片段内是唯一的, 然后进行对于不同节点之间的重复的检查。如果发现任何重复, 则使用打破僵局规则(`tie breaker rule`)来改变这些号码中的一些或全部以确保不同的号码(或者完全定序的字符串)被分配给所有节点上的不同的数据项。在发现重复的情况下可以使用的打破僵局规则的示例是用于将所有号码重新分配在N个节点的各节点上的以下规则, 其中‘ i ’是被分配给数据项的原始本地号码, 并且‘ j ’是节点的号码($1 \sim N$): $(i+j) \bmod N$ 。

[0087] 具有完全定序的数据项的数据库的一个性质是: 如果仅以与数据项的定序规则一

致的顺序获取数据项上的锁，则不可能发生死锁。即，如果事务获取具有号码1的数据项上的锁、然后获取具有号码2的数据项上的锁，则不可能发生死锁（假设所有其它事务正以与定序规则一致的顺序获取锁）。然而，如果事务获取具有号码2的数据项上的锁、然后获取数据项1上的锁，则可能发生死锁（但不能保证），这取决于事务的相对定时。

[0088] 一般来说，先占式死锁避免算法314首先使得事务能够尝试以任何顺序获得数据项上的锁，并且不会迫使事务尝试以与定序规则一致的顺序获得锁。如果事务尝试获得的任何锁都不是在预定时间间隔或时间延迟内获得的，则由于死锁已经发生或者将会发生的可能性、因此事务回退。在一些示例中，时间间隔或时间延迟足够长，使得正以与定序规则一致的顺序获取锁的任何两个事务有可能（例如，可能性在50%以上、75%以上或者90%以上）在该时间间隔或时间延迟内获得它们的锁。

[0089] 对于回退的任何事务，以与定序规则一致的第二顺序来对该事务的锁定操作进行排序。然后重新开始（例如，重试或再次处理）该事务，并且以排序好的第二顺序来进行锁定操作。重新开始的事务将至少在获取锁的方面取得无死锁的进展，这是因为该事务以与定序规则一致的顺序获取其锁、并且与正进行第一次尝试的事务相比具有更高的优先级。即，由于重新开始的事务与正进行第一次尝试的事务相比具有更高的优先级，因此以下描述的示例性算法314将回退正进行第一次尝试的任何事务。结果，重新开始的事务只与先前已被回退并且正以与锁定顺序一致的顺序获取至少一些锁的事务竞争锁。在算法的其它示例中，仍使用时间间隔之后的定序判断，但是不必使用事务之间的相对优先级，因此仍防止死锁，但取得进展的要求不必是严格的。

[0090] 更具体地，在算法314的第一步骤316中，检查循环条件以查看当循环顺次通过事务T所执行的锁定操作的列表（即，列表所指示的序列中的某一时刻的一个锁定操作）而进展时是否存在所要进行的下一个锁定操作，并且针对各锁定操作，算法314开始尝试获得锁定操作所标识的数据项上的锁L。在第二步骤318中，锁定操作检查L.所有者（即，锁L的表示哪些事务（如果有的话）已拥有锁L的性质）以判断其值是否等于空。如果是，则当前没有事务拥有锁L，并且算法进入事务T拥有锁L的第三步骤320。在第三步骤320中，L.所有者被设置为T，并且循环返回步骤316。

[0091] 如果L.所有者不等于空，则另一事务已拥有锁L，并且算法进入第四步骤322，其中在该第四步骤322中，将T.优先级（即，表示事务T的处理优先级的性质）与L.所有者的优先级（即，表示当前拥有锁L的事务的处理优先级的性质）进行比较。如果T.优先级大于L.所有者.优先级，则算法314进入第五步骤324，其中在该第五步骤324中，当前拥有锁L的事务（即，L.所有者）回退。然后，事务T拥有锁L，这时L.所有者被设置为T，并且循环返回步骤316。如果T.优先级小于或等于L.所有者.优先级，则算法314进入第六步骤326，其中在该第六步骤326中，事务被允许一段时间的间隔（或“时间间隔”或“时间延迟”） t_w ，以试图获得锁L。

[0092] 如果在该段时间的间隔 t_w 内释放锁L、从而使得L.所有者变为等于空，则算法314进入事务T拥有锁L的第七步骤330，并且循环返回步骤316。在第七步骤330中，L.所有者被设置为T。可选地，如果在时间间隔或时间延迟 t_w 期间、其它事务未曾释放锁L，则算法进入第八步骤332。在该第八步骤332中，算法314回退事务T所发出的所有操作。然后第八步骤332以与数据项的定序规则一致的第二顺序对T所执行的锁定操作的列表进行排序并使T.

优先级增长。然后算法314通过返回步骤316来重新开始事务T，其中循环条件被重新初始化到排序好的锁定操作的列表的开始，因此现将以排序好的(第二)顺序顺次获得锁定操作的锁。

[0093] 可以可选地使用与示例性算法314相比应用不同的解决方案的先占式死锁避免算法的多种其它示例，但仍然以如算法314中那样减少开销的方式来避免死锁。例如，代替在特定时间间隔或时间延迟之后针对每个事务进行回退和以第二顺序对锁定操作的排序(在步骤332中)，算法可以作为代替而首先判断获得任何初始锁的顺序是否与定序规则一致。如果初始锁是以与定序规则一致的顺序获得的，则延迟不是由于死锁造成的，并且事务可以在回退事务之前、在更长时间间隔或时间延迟内继续等待锁。然而，如果初始锁不是以与定序规则一致的顺序获得的，则由于已经发生了死锁这一可能性、因此事务被回退并且例如以排序好的列表重新开始(例如，重试或再次处理)。因此，一些算法包括用以判断第一顺序是否与定序规则一致的步骤，并且其它算法(诸如算法314)不包括这样的步骤。

[0094] 3.1示例

[0095] 参考图4，在先占式死锁避免算法314的典型应用中，两个事务T₁和T₂并行地在分布式数据库的数据项(包括数据项x和y)上操作。这些数据项根据定序规则按顺序进行编号，其中x分配了号码1并且y分配了号码2。为了简化以下的讨论，数据项被称为x₁和y₂。

[0096] 在初始时间步(0)处，没有事务拥有x₁或y₂上的锁。在第一时间步(1)处，T₁尝试通过发出read_lock(y₂)命令来获得y₂上的锁。为此，T₁应用先占式死锁避免算法314的第二步骤318，并且判断y₂的L.所有者是否等于空。由于在第一时间步(1)处没有其它事务拥有y₂上的写锁，因此T₁进入算法314的第三步骤320，并且成功获得y₂上的读锁R_{T₁}434。在获得y₂上的读锁R_{T₁}434时，y₂的L.所有者被设置为T₁。

[0097] 在第二时间步(2)处，T₁尝试通过发出read(y₂)命令来读取y₂的值。由于T₁拥有y₂上的读锁，因此T₁成功读取y₂的值。

[0098] 在第三时间步(3)处，T₂尝试通过发出read_lock(x₁)命令来获得x₁上的锁。为此，T₂应用先占式死锁避免算法314的第二步骤318，并且判断x₁的L.所有者是否等于空。由于在第三时间步(3)处没有其它事务拥有x₁上的写锁，因此T₂进入算法314的第三步骤320，并且成功获得x₁上的读锁R_{T₂}436。在获得x₁上的读锁R_{T₂}436时，x₁的L.所有者被设置为T₂。

[0099] 在第四时间步(4)处，T₂尝试通过发出read(x₁)命令来读取x₁的值。由于T₂拥有x₁上的读锁，因此T₂成功读取x₁的值。

[0100] 在第五时间步(5)处，T₁尝试通过发出write_lock(x₁)命令来获得x₁上的写锁。为此，T₁应用先占式死锁避免算法314的第二步骤318，并且判断x₁的L.所有者是否等于空。由于T₂仍然拥有x₁上的写锁(即，x₁的L.所有者等于T₂)，因此T₁进入算法314的第四步骤322，并且判断T₁是否具有比T₂更大的优先级。在第五时间步(5)处，T₁.优先级和T₂.优先级等于零，因此T₁进入算法314的第六步骤326，并且试图在时间间隔或时间延迟t_w内获得x₁上的写锁。

[0101] 在第六时间步(6)处，在T₁仍然等待以获得x₁上的写锁的情况下，T₂尝试通过发出write_lock(y₂)命令来获得y₂上的写锁。为此，事务T₂应用先占式死锁避免算法314的第二步骤318，并且判断y₂的L.所有者是否等于空。由于T₁仍然拥有y₂上的写锁(即，y₂的L.所有者等于T₁)，因此T₂进入算法314的第四步骤322，并且判断T₂是否具有比T₁更大的优先级。在

第五时间步(5)处, T_1 .优先级和 T_2 .优先级等于零, 因此 T_2 进入算法314的第六步骤326, 并且试图在时间间隔或时间延迟 t_w 内获得 y_2 上的写锁。

[0102] 在第七时间步(7)处, 经过了事务 T_1 试图获得 x_1 上的写锁所需的时间间隔 t_w , 并且(由于时间间隔 t_w 期间没有释放读锁 $R_{T_2} 436$ 因而) T_1 未曾获得 x_1 上的写锁。算法314进入第八步骤332, 并且回退事务 T_1 , 包括中止其在第七时间步(7)获得 x_1 上的写锁的尝试并且在第八时间步(8)释放 y_2 上的读锁。通过回退事务 T_1 , 避免了 T_1 和 T_2 之间的死锁。由于 T_1 被回退, 因此 T_1 .优先级通过算法314的第八步骤332从0增加至1(参见图5)。通过使 T_1 的优先级增加, 算法314确保 T_1 不会由于被重复回退因而饥饿。最终, 算法314的第八步骤332检查 T_1 在回退之前发出的锁定操作, 并且将这些锁定操作排序为与数据项的定序规则一致。针对 T_1 , 将锁定操作排序为:

[0103] (1) write_lock (x_1)

[0104] (2) read_lock (y_2)

[0105] 一旦释放 T_1 在 y_2 上的锁, 事务 T_2 就能够成功地获得 y_2 上的写锁 $W_{T_2} 440$ 。在第九时间步(9)处, 事务 T_2 发出write (y_2) 命令并且用新的值 y_2' 来重写 y_2 。在第十时间步(10)处, 事务 T_2 发出commit (T_2) 命令以提交对数据库的改变。在提交 T_2 的改变之后, x_1 上的读锁 $R_{T_2} 436$ 和 y_2' 上的写锁 $W_{T_2} 440$ 释放, 并且事务 T_2 完成。

[0106] 在第十一时间步(11)处, T_1 开始第二次尝试。在第二次尝试中, T_1 以排序好的顺序(即, 以与数据项的定序规则一致的顺序)重新发出在事务的最开始的第一次尝试中发出的锁定操作。即, 在第十一时间步(11)处, T_1 尝试通过发出write_lock (x_1) 命令来获得 x_1 上的写锁。为此, T_1 应用先占式死锁避免算法314的第二步骤318, 并且判断 x_1 的L.所有者是否等于空。由于在第十一时间步(11)处没有其它事务拥有 x_1 上的写锁, 因此 T_1 进入算法314的第三步骤320, 并且成功获得 x_1 上的写锁 $W_{T_1} 442$ 。在获得 x_1 上的写锁 $W_{T_1} 442$ 时, x_1 的L.所有者被设置为 T_1 。

[0107] 在第十二时间步(12)处, T_1 尝试通过发出read_lock (y_2') 命令来获得 y_2 上的读锁。为此, 第一事务 T_1 应用先占式死锁避免算法314的第二步骤318, 并且判断 y_2' 的L.所有者是否等于空。由于在第十二时间步(12)处没有其它事务拥有 y_2' 上的写锁, 因此 T_1 进入算法314的第三步骤320, 并且成功获得 y_2' 上的读锁 $R_{T_1} 444$ 。在获得 y_2' 上的读锁 $R_{T_1} 444$ 时, y_2' 的所有者被设置为 T_1 。

[0108] 在第十三时间步(13)处, 事务 T_1 发出write (x_1) 命令并且用新的值 x_1' 来重写 x_1 。在第十四时间步(14)处, 事务 T_1 发出commit (T_1) 命令以提交对数据库的改变。在提交 T_1 的改变之后, y_2 上的读锁 $R_{T_1} 444$ 和写锁 $W_{T_1} 442$ 释放, 并且事务 T_1 完成。

[0109] 参考图5, 在先占式死锁避免算法314的另一典型应用中, 两个事务 T_1 和 T_2 并行地在分布式数据库的数据项(包括数据项 x_1 和 y_2)上操作。在该示例中, 第一事务 T_1 先前已被回退, 并且 T_1 .优先级等于1。

[0110] 在初始时间步(0)处, 没有事务拥有 x_1 或 y_2 上的锁。在第一时间步(1)处, T_1 尝试通过发出read_lock (y_2) 命令来获得 y_2 上的锁。为此, T_1 应用先占式死锁避免算法314的第二步骤318, 并且判断 y_2 的L.所有者是否等于空。由于在第一时间步(1)处没有其它事务拥有 y_2 上

的写锁,因此T₁进入算法314的第三步骤320,并且成功获得y₂上的读锁R_{T₁}646。在获得y₂上的读锁R_{T₁}646时,y₂的L.所有者被设置为T₁。

[0111] 在第二时间步(2)处,T₁尝试通过发出read(y₂)命令来读取y₂的值。由于T₁拥有y₂上的读锁,因此T₁成功读取y₂的值。

[0112] 在第三时间步(3)处,T₂尝试通过发出read_lock(x₁)命令来获得x₁上的锁。为此,T₂应用先占式死锁避免算法314的第二步骤318,并且判断x₁的L.所有者是否等于空。由于在第三时间步(3)处没有其它事务拥有x₁上的写锁,因此T₂进入算法314的第三步骤320,并且成功获得x₁上的读锁R_{T₂}648。在获得x₁上的读锁R_{T₂}648时,x₁的L.所有者被设置为T₂。

[0113] 在第四时间步(4)处,T₂尝试通过发出read(x₁)命令来读取x₁的值。由于T₂拥有x₁上的读锁,因此T₂成功读取x₁的值。

[0114] 在第五时间步(5)处,T₁尝试通过发出write_lock(x₁)命令来获得x₁上的写锁。为此,T₁应用先占式死锁避免算法314的第二步骤318,并且判断x₁的L.所有者是否等于空。由于T₂仍然拥有x₁上的读锁(即,x₁的L.所有者等于T₂),因此T₁进入算法314的第四步骤322,并且判断T₁是否具有比T₂更大的优先级。在第五时间步(5)处,T₁.优先级等于1,且T₂.优先级等于0。因此,T₁进入算法314的第五步骤324并且回退T₂,包括在第六时间步(6)处发出unlock(x₁)命令以释放T₂在x₁上的读锁R_{T₂}646。由于T₂被回退,因此,利用算法314的第五步骤324,T₂.优先级从0增加至1。

[0115] 一旦释放T₂在x₁上的锁R_{T₂}648,T₁就能够成功地获得x₁上的写锁W_{T₁}650。在第七时间步(7)处,T₁发出write(x₁)命令并且用新的值x₁'来重写x₁。在第八时间步(8)处,T₁发出commit(T₁)命令以提交对数据库的改变。在提交T₁的改变之后,y₂上的读锁R_{T₁}646和x₁'上的写锁W_{T₁}650释放,并且事务T₁完成。

[0116] 在第九时间步(9)处,T₂开始第二次尝试。在第二次尝试中,T₂尝试通过发出read_lock(x₁)命令来获得x₁'上的读锁。为此,T₂应用先占式死锁避免算法314的第二步骤318,并且判断x₁的L.所有者是否等于空。由于在第九时间步(9)处没有其它事务拥有x₁'上的写锁,因此T₂进入算法314的第三步骤320,并且成功获得x₁'上的读锁R_{T₂}652。在获得x₁'上的读锁R_{T₂}652时,x₁'的L.所有者被设置为T₂。

[0117] 在第十时间步(10)处,T₂尝试通过发出write_lock(y₂)命令来获得y₂上的写锁。为此,第二事务T₂应用先占式死锁避免算法314的第二步骤318,并且判断y₂的L.所有者是否等于空。由于在第十时间步(10)处没有其它事务拥有y₂上的写锁,因此T₂进入算法314的第三步骤320,并且成功获得y₂上的写锁W_{T₂}654。在获得y₂上的写锁W_{T₂}654时,y₂的L.所有者被设置为T₂。

[0118] 在第十一时间步(11)处,T₂发出read(x₁')命令以读取x₁'的值。在第十二时间步(12)处,T₂发出write(y₂)命令并且用新的值y₂'来重写y₂的值。在第十三时间步(13)处,T₂发出commit(T₂)命令以提交T₂对数据库的改变。在提交T₂的改变之后,x₁上的读锁R_{T₂}652和x₁'上的写锁W_{T₂}654释放,并且事务T₂完成。

[0119] 4可选例

[0120] 在一些示例中,当在事务的第一次尝试中对数据项进行锁定操作时,存储基于定

序规则的数据项的位置以供稍后对锁定操作进行排序时使用。例如,可以存储在第一时间延迟内获得锁的数据项的位置。代替要求对事务中的所有锁进行排序,对于确保向前的进展,简单地仅对在第一时间延迟内初始获得的锁进行排序可能是足够的,这对于具有大量锁定操作的事务而言可能不需要那么多的工作。

[0121] 在一些示例中,如果锁定操作与定序规则一致,则用于等待锁定操作完成的时间间隔或时间延迟更长。

[0122] 在一些示例中,事务仅在第二时间间隔或第二时间延迟之后重新开始。在一些示例中,第二时间间隔或时间延迟比锁定操作被允许试图获得锁的时间间隔或时间延迟更长。

[0123] 在一些示例中,锁定操作被允许试图获得锁的时间间隔或时间延迟小于1秒(例如,100ms、1ms、100微秒、1微秒或0.1微秒)。在一些示例中,锁定被允许试图获得锁的时间间隔约为0秒。

[0124] 在一些示例中,针对给定数据项的锁可以具有包括一个未锁定状态以及一个或多个锁定状态等的多个状态。在一些示例中,一个或多个锁定状态包括独占写入锁定状态和非独占多重读取锁定状态。

[0125] 在一些示例中,分布式数据库系统上正执行的事务是异步执行的事务。在一些示例中,数据库是内存数据库。

[0126] 可以使用各种其它可选技术,包括这里描述的各种特征,以及包括以上的发明内容部分中描述的特征。

[0127] 5实现

[0128] 上述的死锁避免方法可以例如使用执行适当软件指令的可编程计算系统来实现,或者可以在诸如现场可编程门阵列(FPGA)等的适当硬件中或者以某种混合形式实现。例如,在编程的方法中,该软件可以包括在一个或多个编程或可编程计算系统(可以具有诸如分布式、客户端/服务器或网格式等各种架构)上执行的一个或多个计算机程序中的过程,其中该一个或多个编程或可编程计算系统各自包括至少一个处理器、至少一个数据存储系统(包括易失性和/或非易失性存储器和/或存储元件)、至少一个用户接口(用于使用至少一个输入装置或端口来接收输入,并且用于使用至少一个输出装置或端口来提供输出)。该软件可以包括例如提供与数据流图的设计、配置和执行有关的服务的较大程序的一个或多个模块。可以将程序的模块(例如,数据流图的元素)实现为符合数据存储库中所存储的数据模型的数据结构或其它有组织数据。

[0129] 软件能够在一段时间(例如,诸如动态RAM等的动态存储器装置的刷新周期之间的时间)内以非暂时性形式(诸如在易失性或非易失性存储介质或任何其它非暂时性介质中体现)、使用介质的物理性质(例如,表面凹区和凸区、磁畴或电荷)存储。在为加载指令所作的准备中,软件可以设置在诸如(利用通用或专用计算机系统或装置可读取的)CD-ROM或其他计算机可读介质等的有形、非暂时性介质上,或者可以经由网络的通信介质(例如,以编码在传播信号中的形式)传递至执行该软件的计算系统的有形、非暂时性介质。可以在专用计算机上、或者使用诸如协处理器或现场可编程门阵列(FPGA)或专用特定用途集成电路(ASIC)等的专用硬件来进行一些处理或所有功能。可以以分布式方式来实现该处理,其中利用不同的计算元件来进行软件所指定的计算的不同部分。优选将每一个这种计算机程序

存储在通用或专用可编程计算机可读取的存储装置的计算机可读存储介质(例如,固态存储器或介质、或者磁性或光学介质)上或者下载至该存储介质,以在利用计算机读取存储装置介质的情况下配置计算机并使该计算机进行工作以进行这里所述的处理。本发明的系统还可被视为作为配置有计算机程序的有形、非暂时性介质来实现,其中如此配置成的介质使计算机以特定的预定义方式进行工作,以进行这里所述的处理步骤中的一个或多个。

[0130] 已经描述了本发明的许多实施例。然而,应当理解,前述描述旨在示出而不是限制本发明的范围,该范围由所附权利要求书的范围限定。因此,其它实施例也在所附权利要求书的范围内。例如,可以在不偏离本发明的范围的情况下作出各种修改。另外,上述步骤中的一些可以是与顺序无关的,因此可以按照与所描述的顺序不同的顺序执行。

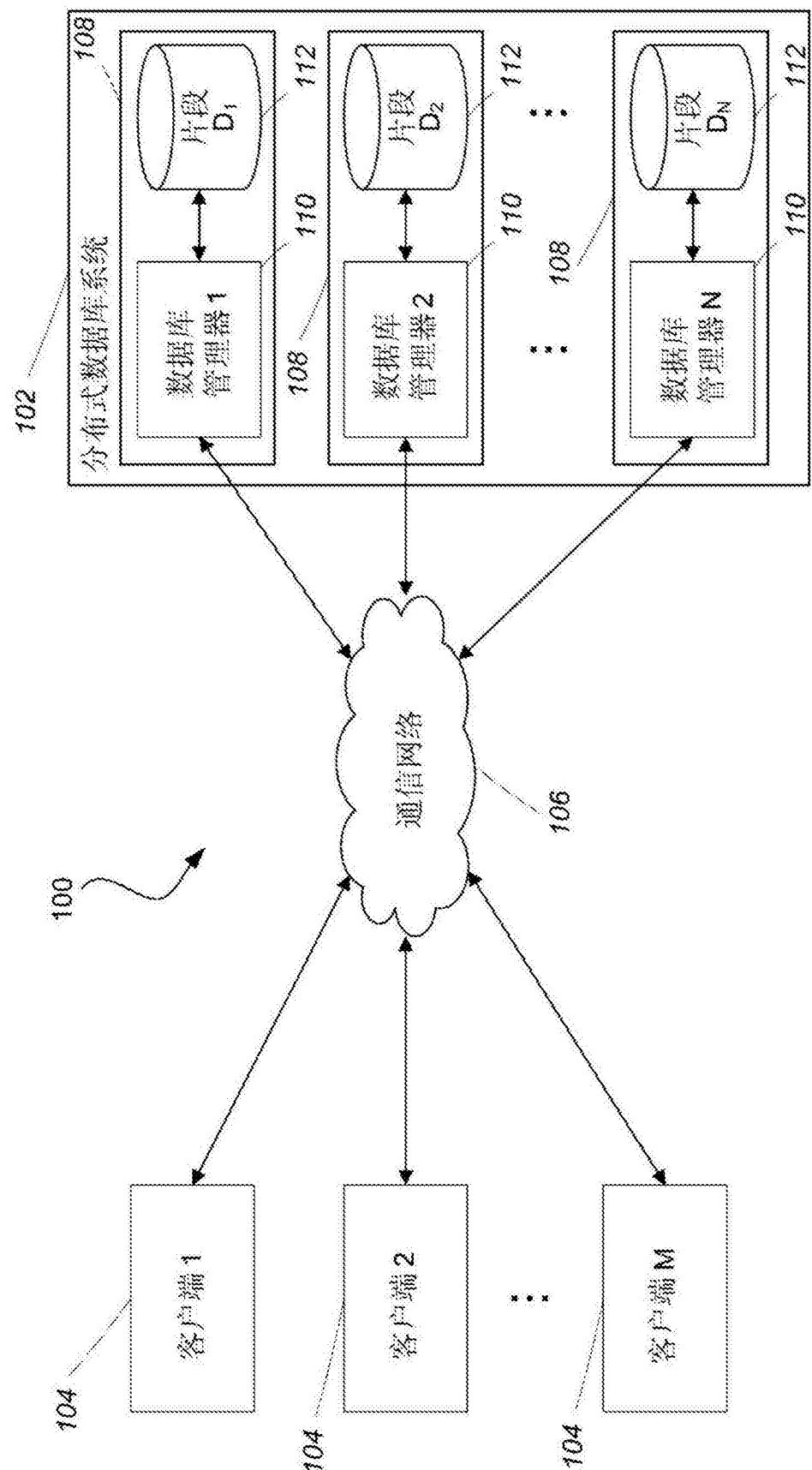


图1

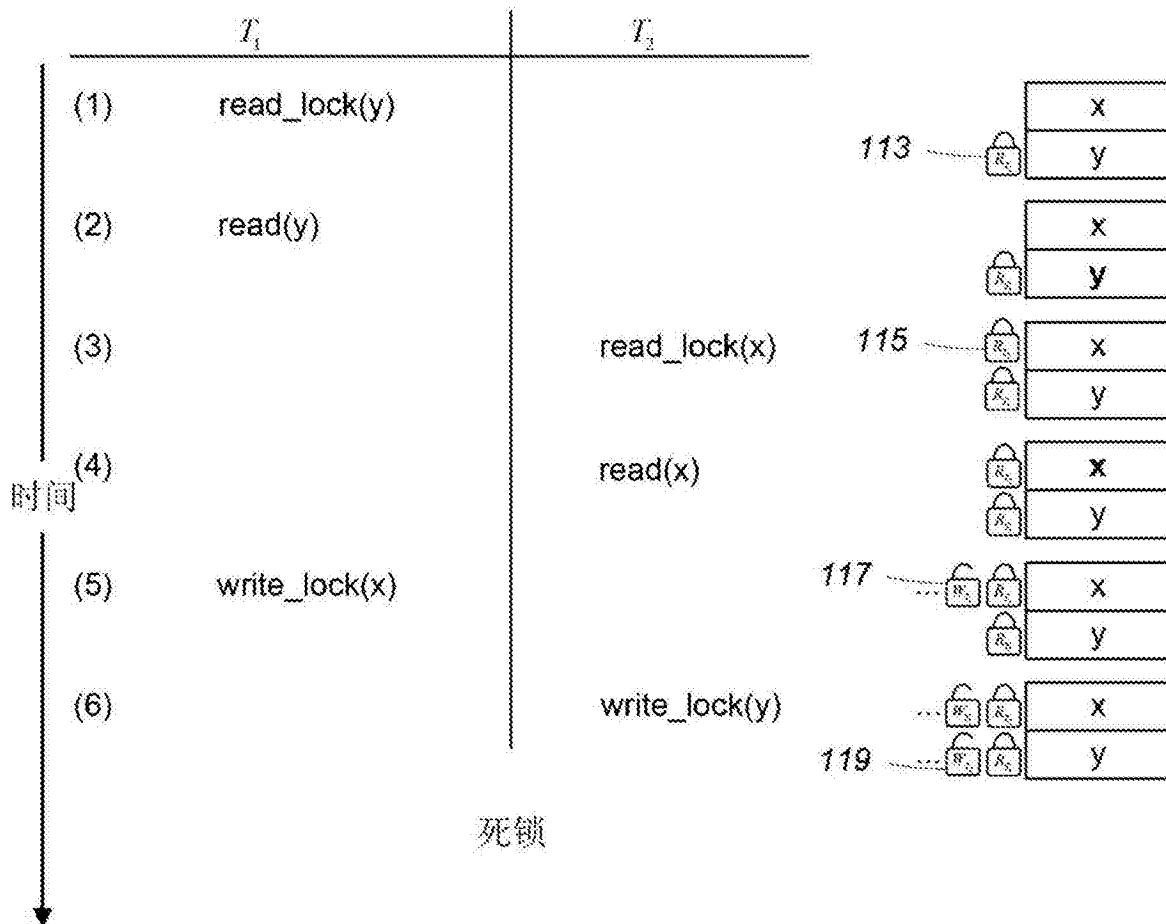


图2

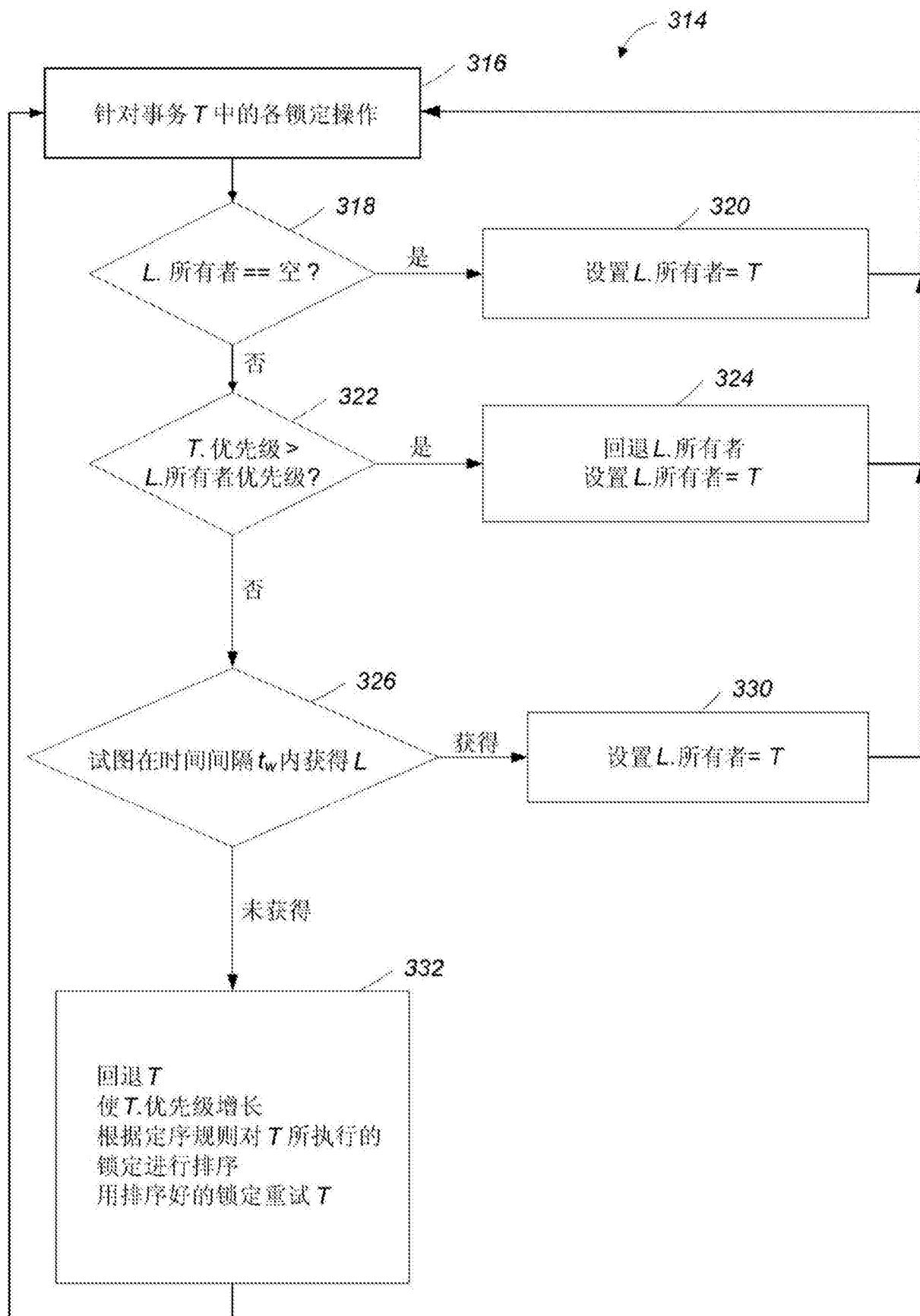


图3

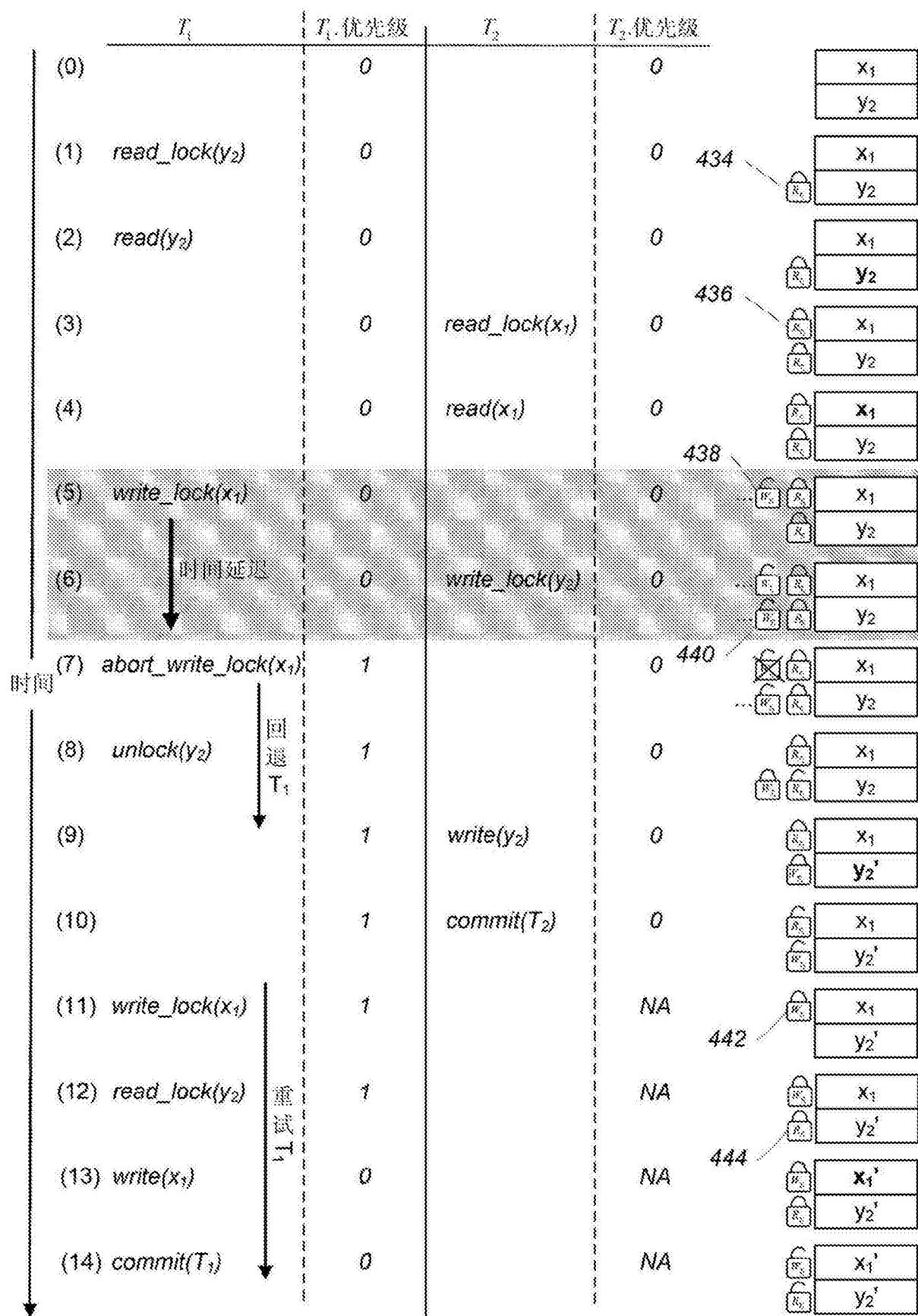


图4

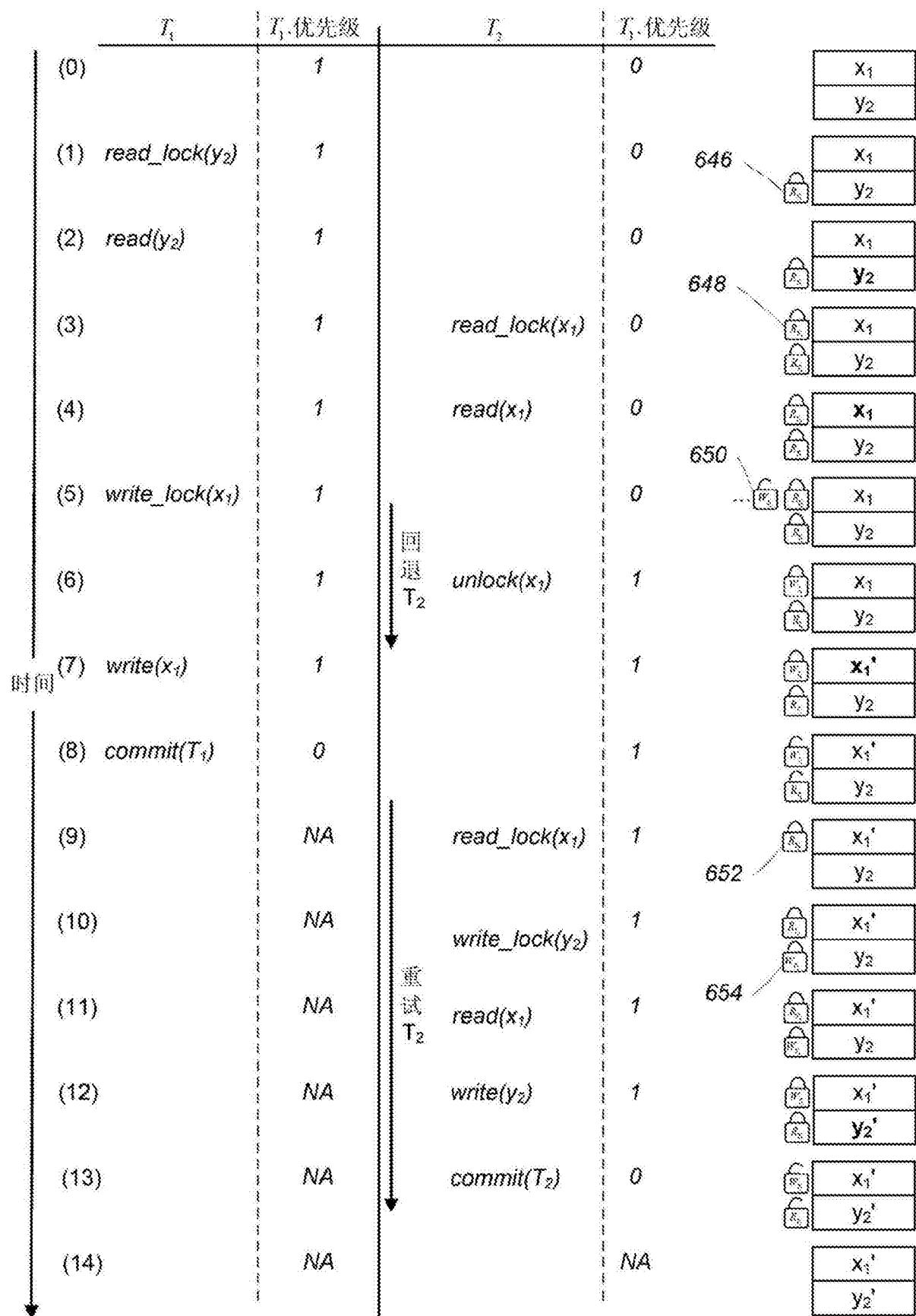


图5