



(12) 发明专利

(10) 授权公告号 CN 112231071 B

(45) 授权公告日 2021.06.18

(21) 申请号 202011237860.9

(22) 申请日 2020.05.20

(65) 同一申请的已公布的文献号
申请公布号 CN 112231071 A

(43) 申请公布日 2021.01.15

(62) 分案原申请数据
202010431659.8 2020.05.20

(73) 专利权人 腾讯科技(深圳)有限公司
地址 518057 广东省深圳市南山区高新区
科技中一路腾讯大厦35层

(72) 发明人 李海翔

(74) 专利代理机构 北京三高永信知识产权代理
有限责任公司 11138
代理人 徐立

(51) Int.Cl.

G06F 9/46 (2006.01)

G06F 16/2453 (2019.01)

(56) 对比文件

CN 103019728 A, 2013.04.03

CN 105389161 A, 2016.03.09

CN 111159252 A, 2020.05.15

审查员 崔鑫彤

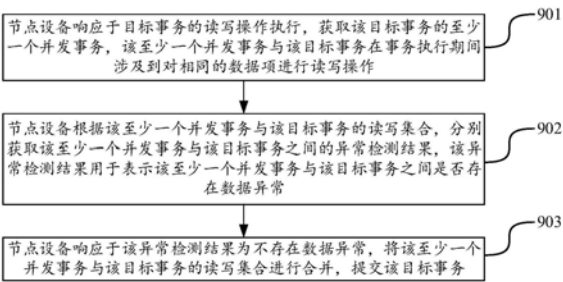
权利要求书5页 说明书56页 附图6页

(54) 发明名称

事务处理方法、装置、计算机设备及存储介质

(57) 摘要

本申请公开了一种事务处理方法、装置、计算机设备及存储介质,属于数据库技术领域。本申请能够全面检测出分布式数据库系统的数据异常,保证数据状态的一致性,这种事务处理机制既不完全依赖于封锁技术也不完全依赖于依赖图技术,避免了限制分布式数据库系统的并发度,有利于提升数据的查询速度和修改速度、提升事务处理效率。



1. 一种事务处理方法,应用于分布式数据库系统,所述分布式数据库系统包括数据节点设备和协调节点设备,其特征在于,所述方法包括:

所述数据节点设备响应于目标事务的读写操作执行,获取所述目标事务的至少一个并发事务,所述至少一个并发事务与所述目标事务在事务执行期间涉及到对相同的数据项进行读写操作;

所述数据节点设备对于所述至少一个并发事务中任一并发事务,根据所述任一并发事务与所述目标事务的读写集合,基于动边交叉与合并算法获取所述任一并发事务与所述目标事务之间的目标检测结果;

所述数据节点设备响应于所述目标检测结果为存在数据异常,将所述至少一个并发事务与所述目标事务之间的异常检测结果确定为存在数据异常;否则,对所述至少一个并发事务中所述任一并发事务的下一个并发事务,执行基于动边交叉与合并算法获取目标检测结果的步骤;

所述数据节点设备响应于所述至少一个并发事务的目标检测结果均为不存在数据异常,将所述异常检测结果确定为不存在数据异常;

所述数据节点设备响应于所述异常检测结果为不存在数据异常,向所述协调节点设备上报所述异常检测结果;

所述数据节点设备响应于所述协调节点设备指示所述目标事务所涉及的各个数据节点设备不存在全局的数据异常,将所述至少一个并发事务与所述目标事务的读写集合进行合并,提交所述目标事务。

2. 根据权利要求1所述的方法,其特征在于,所述方法还包括:

所述数据节点设备对于所述目标事务的任一次读操作,确定所述任一次读操作的快照;

所述数据节点设备根据所述任一次读操作的读取条件以及所述任一次读操作的快照,确定相对于所述目标事务可见的目标数据项,将所述目标数据项添加至所述目标事务的读集中。

3. 根据权利要求2所述的方法,其特征在于,所述确定所述任一次读操作的快照包括:

所述数据节点设备响应于所述分布式数据库系统的隔离级别低于可重复读,在执行所述任一次读操作时获取一次快照;否则,将所述目标事务第一次读操作的快照确定为所述任一次读操作的快照,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

4. 根据权利要求2所述的方法,其特征在于,所述根据所述任一次读操作的读取条件以及所述任一次读操作的快照,确定相对于所述目标事务可见的目标数据项包括:

所述数据节点设备响应于所述分布式数据库系统的隔离级别高于或等于读写已提交,将符合所述读取条件且提交时间戳小于所述任一次读操作的快照的最大版本确定为所述目标数据项,所述读写已提交隔离级别中只有处于已提交状态的数据才可读或可写;

所述数据节点设备响应于所述分布式数据库系统的隔离级别低于读写已提交,若符合所述读取条件的数据项中存在未提交版本,将最大的未提交版本确定为所述目标数据项。

5. 根据权利要求2所述的方法,其特征在于,所述任一次读操作的快照的获取过程包括:

当所述目标事务的第一个结构化查询语言SQL语句执行时,所述协调节点设备为所述目标事务创建所述快照;

随着所述目标事务的SQL语句在所述数据节点设备上执行,所述协调节点设备将所述快照下发至所述数据节点设备。

6.根据权利要求2至5中任一项所述的方法,其特征在于,所述任一次读操作的快照采用混合逻辑时钟HLC,所述HLC采用三元组表示本地物理时钟和混合逻辑时钟,所述本地物理时钟为所述三元组内的第一个元素,所述混合逻辑时钟为所述三元组内的第二个元素和第三个元素。

7.根据权利要求6所述的方法,其特征在于,所述确定相对于所述目标事务可见的目标数据项包括:

所述数据节点设备按所述协调节点设备传入的HLC值,在数据项的版本链上进行遍历,查找和所述HLC的第二个元素相同的数据版本,再往后读一个版本,得到处于全局读一致的所述目标数据项,其中,所述版本链从新到旧有序存储且满足读已提交规则。

8.根据权利要求1所述的方法,其特征在于,所述协调节点设备对所述目标事务的验证过程包括:

所述协调节点设备汇总各个数据节点设备返回的依赖关系和异常检测结果,进行全局异常检测,以确定提交所述目标事务或回滚所述目标事务。

9.根据权利要求8所述的方法,其特征在于,所述依赖关系为依赖图,所述全局异常检测的过程包括:

所述协调节点设备基于各个数据节点设备返回的所述目标事务的多个子事务的依赖图,构造所述目标事务的全局依赖图;

若所述全局依赖图不存在环且没有破坏完整性约束,所述协调节点设备确定所述目标事务符合可串行化隔离级别,所述可串行化隔离级别中无数据异常发生;

若所述全局依赖图不存在环且完整性约束被破坏,所述协调节点设备确定所述目标事务符合可重复读隔离级别,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

10.根据权利要求9所述的方法,其特征在于,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

若所述全局依赖图中首尾相接的子事务由两个读操作组成,所述协调节点设备确定存在读偏序异常或台阶式异常,在所述分布式数据库系统的隔离级别高于或等于快照读历史时,确定回滚所述目标事务,所述快照读历史隔离级别中基于历史上的任何一个时间点,支持读取历史上已经提交的数据。

11.根据权利要求9所述的方法,其特征在于,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

若所述全局依赖图中首尾相接的子事务由读写操作组成,所述协调节点设备确定存在写偏序异常或锯齿波异常,在所述分布式数据库系统的隔离级别高于或等于可重复读时,确定回滚所述目标事务,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

12.一种事务处理装置,其特征在于,所述装置位于分布式数据库系统,所述分布式数

数据库系统还包括协调节点设备,所述装置包括:

第一获取模块,用于响应于目标事务的读写操作执行,获取所述目标事务的至少一个并发事务,所述至少一个并发事务与所述目标事务在事务执行期间涉及到对相同的数据项进行读写操作;

第二获取模块,包括获取子模块和确定子模块:

所述获取子模块,用于对于所述至少一个并发事务中任一并发事务,根据所述任一并发事务与所述目标事务的读写集合,基于动边交叉与合并算法获取所述任一并发事务与所述目标事务之间的目标检测结果;

所述确定子模块,用于响应于所述目标检测结果为存在数据异常,将所述至少一个并发事务与所述目标事务之间的异常检测结果确定为存在数据异常;否则,对所述至少一个并发事务中所述任一并发事务的下一个并发事务,执行基于动边交叉与合并算法获取目标检测结果的步骤;

所述确定子模块,还用于响应于所述至少一个并发事务的目标检测结果均为不存在数据异常,将所述异常检测结果确定为不存在数据异常;

合并提交模块,用于响应于所述异常检测结果为不存在数据异常,向所述协调节点设备上报所述异常检测结果;响应于所述协调节点设备指示所述目标事务所涉及的各个数据节点设备不存在全局的数据异常,将所述至少一个并发事务与所述目标事务的读写集合进行合并,提交所述目标事务。

13. 根据权利要求12所述的装置,其特征在于,所述装置还包括:

第二确定模块,用于对于所述目标事务的任一次读操作,确定所述任一次读操作的快照;

第三确定模块,用于根据所述任一次读操作的读取条件以及所述任一次读操作的快照,确定相对于所述目标事务可见的目标数据项,将所述目标数据项添加至所述目标事务的读集中。

14. 根据权利要求13所述的装置,其特征在于,所述第二确定模块用于:

响应于所述分布式数据库系统的隔离级别低于可重复读,在执行所述任一次读操作时获取一次快照;否则,将所述目标事务第一次读操作的快照确定为所述任一次读操作的快照,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

15. 根据权利要求13所述的装置,其特征在于,所述第三确定模块用于:

响应于所述分布式数据库系统的隔离级别高于或等于读写已提交,将符合所述读取条件且提交时间戳小于所述任一次读操作的快照的最大版本确定为所述目标数据项,所述读写已提交隔离级别中只有处于已提交状态的数据才可读或可写;

响应于所述分布式数据库系统的隔离级别低于读写已提交,若符合所述读取条件的数据项中存在未提交版本,将最大的未提交版本确定为所述目标数据项。

16. 根据权利要求13所述的装置,其特征在于,所述任一次读操作的快照的获取过程包括:

当所述目标事务的第一个结构化查询语言SQL语句执行时,所述协调节点设备为所述目标事务创建所述快照;

随着所述目标事务的SQL语句在所述数据节点设备上执行,所述协调节点设备将所述

快照下发至所述数据节点设备。

17. 根据权利要求13至16中任一项所述的装置,其特征在于,所述任一次读操作的快照采用混合逻辑时钟HLC,所述HLC采用三元组表示本地物理时钟和混合逻辑时钟,所述本地物理时钟为所述三元组内的第一个元素,所述混合逻辑时钟为所述三元组内的第二个元素和第三个元素。

18. 根据权利要求17所述的装置,其特征在于,所述第三确定模块用于:

按所述协调节点设备传入的HLC值,在数据项的版本链上进行遍历,查找和所述HLC的第二个元素相同的数据版本,再往后读一个版本,得到处于全局读一致的所述目标数据项,其中,所述版本链从新到旧有序存储且满足读已提交规则。

19. 根据权利要求12所述的装置,其特征在于,所述协调节点设备对所述目标事务的验证过程包括:

汇总各个数据节点设备返回的依赖关系和异常检测结果,进行全局异常检测,以确定提交所述目标事务或回滚所述目标事务。

20. 根据权利要求19所述的装置,其特征在于,所述依赖关系为依赖图,所述全局异常检测的过程包括:

所述协调节点设备基于各个数据节点设备返回的所述目标事务的多个子事务的依赖图,构造所述目标事务的全局依赖图;

若所述全局依赖图不存在环且没有破坏完整性约束,所述协调节点设备确定所述目标事务符合可串行化隔离级别,所述可串行化隔离级别中无数据异常发生;

若所述全局依赖图不存在环且完整性约束被破坏,所述协调节点设备确定所述目标事务符合可重复读隔离级别,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

21. 根据权利要求20所述的装置,其特征在于,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

若所述全局依赖图中首尾相接的子事务由两个读操作组成,所述协调节点设备确定存在读偏序异常或台阶式异常,在所述分布式数据库系统的隔离级别高于或等于快照读历史时,确定回滚所述目标事务,所述快照读历史隔离级别中基于历史上的任何一个时间点,支持读取历史上已经提交的数据。

22. 根据权利要求20所述的装置,其特征在于,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

若所述全局依赖图中首尾相接的子事务由读写操作组成,所述协调节点设备确定存在写偏序异常或锯齿波异常,在所述分布式数据库系统的隔离级别高于或等于可重复读时,确定回滚所述目标事务,所述可重复读隔离级别中事务内任何两次的读操作所读取到的数据对象是一样的。

23. 一种计算机设备,其特征在于,所述计算机设备包括一个或多个处理器和一个或多个存储器,所述一个或多个存储器中存储有至少一条计算机程序,所述至少一条计算机程序由所述一个或多个处理器加载并执行如权利要求1至权利要求11任一项所述的事务处理方法。

24. 一种存储介质,其特征在于,所述存储介质中存储有至少一条计算机程序,所述至

少一条计算机程序由处理器加载并执行如权利要求1至权利要求11任一项所述的事务处理方法。

事务处理方法、装置、计算机设备及存储介质

[0001] 本申请是申请日为2020年05月20日、申请号为202010431659.8、发明名称为“事务处理方法、装置、计算机设备及存储介质”的分案申请。

技术领域

[0002] 本申请涉及数据库技术领域，特别涉及一种事务处理方法、装置、计算机设备及存储介质。

背景技术

[0003] 随着数据库技术的发展，在数据库系统中如何识别并规避数据异常成为了一个关键问题。目前有两种识别数据异常的方法，其一是利用封锁技术，依赖锁的互斥机制来避免数据异常，其二是利用依赖图技术，通过在并发事务构成的依赖图中确认是否存在环，如果存在环则需要打破环的存在从而消除潜在的数据异常。然而，封锁技术严重限制了数据库系统的并发度，导致事务处理效率低下，而依赖图技术又需要遍历每个并发事务以识别出环的存在，导致事务处理效率仍然不高。因此，亟需一种能够提升事务处理效率的事务处理方法。

发明内容

[0004] 本申请实施例提供了一种事务处理方法、装置、计算机设备及存储介质，能够提升数据库系统的事务处理效率。该技术方案如下：

[0005] 一方面，提供了一种事务处理方法，应用于分布式数据库系统，所述分布式数据库系统包括数据节点设备和协调节点设备，该方法包括：

[0006] 所述数据节点设备响应于目标事务的读写操作执行，获取所述目标事务的至少一个并发事务，所述至少一个并发事务与所述目标事务在事务执行期间涉及到对相同的数据项进行读写操作；

[0007] 所述数据节点设备根据所述至少一个并发事务与所述目标事务的读写集合，获取所述至少一个并发事务与所述目标事务之间的异常检测结果，所述异常检测结果用于表示所述至少一个并发事务与所述目标事务之间是否存在数据异常；

[0008] 所述数据节点设备响应于所述异常检测结果为不存在数据异常，向所述协调节点设备上报所述异常检测结果；

[0009] 所述数据节点设备响应于所述协调节点设备指示所述目标事务所涉及的各个数据节点设备不存在全局的数据异常，将所述至少一个并发事务与所述目标事务的读写集合进行合并，提交所述目标事务。

[0010] 一方面，提供了一种事务处理装置，该装置位于分布式数据库系统，所述分布式数据库系统还包括协调节点设备，该装置包括：

[0011] 第一获取模块，用于响应于目标事务的读写操作执行，获取所述目标事务的至少一个并发事务，所述至少一个并发事务与所述目标事务在事务执行期间涉及到对相同的数

据项进行读写操作；

[0012] 第二获取模块,用于根据所述至少一个并发事务与所述目标事务的读写集合,获取所述至少一个并发事务与所述目标事务之间的异常检测结果,所述异常检测结果用于表示所述至少一个并发事务与所述目标事务之间是否存在数据异常；

[0013] 合并提交模块,用于响应于所述异常检测结果为不存在数据异常,向所述协调节点设备上报所述异常检测结果;响应于所述协调节点设备指示所述目标事务所涉及的各个数据节点设备不存在全局的数据异常,将所述至少一个并发事务与所述目标事务的读写集合进行合并,提交所述目标事务。

[0014] 在一种可能实施方式中,所述第二获取模块包括:

[0015] 获取子模块,用于对于所述至少一个并发事务中任一并发事务,根据所述任一并发事务与所述目标事务的读写集合,获取所述任一并发事务与所述目标事务之间的目标检测结果;

[0016] 确定子模块,用于响应于所述目标检测结果为存在数据异常,将所述异常检测结果确定为存在数据异常;否则,对所述至少一个并发事务中所述任一并发事务的下一个并发事务执行获取目标检测结果的步骤;

[0017] 所述确定子模块,还用于响应于所述至少一个并发事务的目标检测结果均为不存在数据异常,将所述异常检测结果确定为不存在数据异常。

[0018] 在一种可能实施方式中,所述获取子模块包括:

[0019] 第一获取单元,用于获取第一集合、第二集合以及第三集合,所述第一集合为所述目标事务的读集与所述任一并发事务的写集之间的交集,所述第二集合为所述目标事务的写集与所述任一并发事务的读集之间的交集,所述第三集合为所述目标事务的写集与所述任一并发事务的写集之间的交集;

[0020] 第二获取单元,用于响应于所述第三集合不是空集,根据所述任一并发事务的提交情况获取所述目标检测结果;

[0021] 第三获取单元,用于响应于所述第三集合是空集且所述第一集合或者所述第二集合中至少一项不是空集,根据所述第一集合和所述第二集合获取所述目标检测结果。

[0022] 在一种可能实施方式中,所述获取子模块包括:

[0023] 第一获取单元,用于获取第一集合、第二集合、第三集合以及第四集合,所述第一集合为所述目标事务的读集与所述任一并发事务的写集之间的交集,所述第二集合为所述目标事务的写集与所述任一并发事务的读集之间的交集,所述第三集合为所述目标事务的写集与所述任一并发事务的写集之间的交集,所述第四集合为所述目标事务的读集与所述任一并发事务的读集之间的交集;

[0024] 第二获取单元,用于响应于所述第三集合不是空集,根据所述任一并发事务的提交情况获取所述目标检测结果;

[0025] 第四获取单元,用于响应于所述第三集合是空集且所述第一集合、所述第二集合或者所述第三集合中至少一项不是空集,根据所述第一集合、所述第二集合和所述第四集合获取所述目标检测结果。

[0026] 在一种可能实施方式中,所述第二获取单元用于:

[0027] 若所述任一并发事务未提交且所述目标事务的目标参数为一,将所述目标检测结

果获取为存在数据异常且数据异常类型为脏写异常,所述目标参数用于表示所述目标事务的读写集成成分所对应的已提交事务数量;

[0028] 若所述任一并发事务已提交且所述第一集合与所述第三集合之间的交集不是空集,将所述目标检测结果获取为存在数据异常且数据异常类型为丢失更新异常。

[0029] 在一种可能实施方式中,所述第三获取单元包括:

[0030] 第一更新子单元,用于根据所述第一集合和所述第二集合,更新所述任一并发事务与所述目标事务之间的动边交叉值和变量状态值,所述动边交叉值用于表示所述任一并发事务与所述目标事务在数据状态矩阵中操作的不同数据项之间的线段交叉情况,所述变量状态值用于表示所述任一并发事务与所述目标事务涉及操作到处于不同数据状态的变量情况;

[0031] 第一获取子单元,用于基于更新后的动边交叉值和更新后的变量状态值,获取所述目标检测结果。

[0032] 在一种可能实施方式中,所述第一更新子单元用于:

[0033] 响应于所述第一集合不是空集,将动边交叉值更新为已有值加一所得的数值;响应于所述第一集合中存在数据状态不同的变量,将变量状态值更新为已有值加一所得的数值;

[0034] 响应于所述第二集合不是空集,将动边交叉值更新为已有值加一所得的数值;响应于所述第二集合中存在数据状态不同的变量,将变量状态值更新为已有值加一所得的数值。

[0035] 在一种可能实施方式中,所述第一获取子单元包括:

[0036] 确定子子单元,用于响应于所述更新后的动边交叉值大于或等于二,将所述目标检测结果获取为存在数据异常;基于所述更新后的变量状态值,确定数据异常类型;

[0037] 获取子子单元,用于否则,将所述目标检测结果获取为不存在数据异常。

[0038] 在一种可能实施方式中,所述确定子子单元用于:

[0039] 响应于所述更新后的变量状态值为一,确定所述数据异常类型为读异常;

[0040] 响应于所述更新后的变量状态值大于或等于二,确定所述数据异常类型为写异常。

[0041] 在一种可能实施方式中,所述第四获取单元包括:

[0042] 第二更新子单元,用于根据所述第一集合、所述第二集合和所述第四集合,更新所述任一并发事务的布尔型以及所述目标事务的布尔型,所述布尔型用于表示对应事务所构成的动边在数据状态矩阵中的垂直位置关系且所述布尔型的初始值为假;

[0043] 第二获取子单元,用于响应于更新后的所述目标事务的布尔型和更新后的所述任一并发事务的布尔型均为真,将所述目标检测结果获取为存在数据异常;否则,将所述目标检测结果获取为不存在数据异常。

[0044] 在一种可能实施方式中,所述第二更新子单元用于:

[0045] 响应于所述第一集合不是空集,对于所述第一集合中任一变量,若所述变量在所述目标事务的读集中的版本号大于或等于所述变量在所述任一并发事务的写集中的版本号,将所述目标事务的布尔型更新为真;否则,将所述任一并发事务的布尔型更新为真;

[0046] 响应于所述第二集合不是空集,对于所述第二集合中任一变量,若所述变量在所

述目标事务的写集中的版本号大于所述变量在所述任一并发事务的读集中的版本号,将所述目标事务的布尔型更新为真;否则,将所述任一并发事务的布尔型更新为真;

[0047] 响应于所述第四集合不是空集,对于所述第四集合中任一变量,若所述变量在所述目标事务的读集中的版本号大于所述变量在所述任一并发事务的读集中的版本号,将所述目标事务的布尔型更新为真;若所述变量在所述目标事务的读集中的版本号小于所述变量在所述任一并发事务的读集中的版本号,将所述任一并发事务的布尔型更新为真。

[0048] 在一种可能实施方式中,所述装置还包括:

[0049] 检测调整模块,用于响应于所述分布式数据库系统的隔离级别为可串行化,对所述任一并发事务与所述目标事务进行完整性约束检测;响应于所述任一并发事务与所述目标事务之间违反完整性约束,将所述目标检测结果调整为存在数据异常且数据异常类型为谓词写偏序异常。

[0050] 在一种可能实施方式中,所述装置还包括:

[0051] 第一确定模块,用于响应于所述异常检测结果为存在数据异常,根据数据异常类型以及所述分布式数据库系统的隔离级别,确定所述目标事务的执行结果,所述执行结果用于表示提交所述目标事务或者回滚所述目标事务。

[0052] 在一种可能实施方式中,所述第一确定模块用于:

[0053] 响应于数据异常类型包括脏写异常且所述隔离级别不为读写未提交,将所述执行结果确定为回滚所述目标事务;

[0054] 响应于数据异常类型包括丢失更新异常且所述隔离级别高于或等于快照读历史,将所述执行结果确定为回滚所述目标事务;

[0055] 响应于数据异常类型包括读异常且所述隔离级别高于或等于快照读历史,将所述执行结果确定为回滚所述目标事务;

[0056] 响应于数据异常类型包括写异常且所述隔离级别高于或等于可重复读,将所述执行结果确定为回滚所述目标事务;

[0057] 响应于数据异常类型包括谓词写偏序异常且所述隔离级别为可串行化,将所述执行结果确定为回滚所述目标事务。

[0058] 在一种可能实施方式中,所述装置还包括:

[0059] 第二确定模块,用于对于所述目标事务的任一次读操作,确定所述任一次读操作的快照;

[0060] 第三确定模块,用于根据所述任一次读操作的读取条件以及所述任一次读操作的快照,确定相对于所述目标事务可见的目标数据项,将所述目标数据项添加至所述目标事务的读集中。

[0061] 在一种可能实施方式中,所述第二确定模块用于:

[0062] 响应于所述分布式数据库系统的隔离级别低于可重复读,在执行所述任一次读操作时获取一次快照;否则,将所述目标事务第一次读操作的快照确定为所述任一次读操作的快照。

[0063] 在一种可能实施方式中,所述第三确定模块用于:

[0064] 响应于所述分布式数据库系统的隔离级别高于或等于读写已提交,将符合所述读取条件且提交时间戳小于所述任一次读操作的快照的最大版本确定为所述目标数据项;

[0065] 响应于所述分布式数据库系统的隔离级别低于读写已提交,若符合所述读取条件的数据项中存在未提交版本,将最大的未提交版本确定为所述目标数据项。

[0066] 在一种可能实施方式中,所述任一次读操作的快照的获取过程包括:

[0067] 当所述目标事务的第一个结构化查询语言SQL语句执行时,所述协调节点设备为所述目标事务创建所述快照;

[0068] 随着所述目标事务的SQL语句在所述数据节点设备上执行,所述协调节点设备将所述快照下发至所述数据节点设备。

[0069] 在一种可能实施方式中,所述任一次读操作的快照采用混合逻辑时钟HLC,所述HLC采用三元组表示本地物理时钟和混合逻辑时钟,所述本地物理时钟为所述三元组内的第一个元素,所述混合逻辑时钟为所述三元组内的第二个元素和第三个元素。

[0070] 在一种可能实施方式中,所述第三确定模块用于:

[0071] 按所述协调节点设备传入的HLC值,在数据项的版本链上进行遍历,查找和所述HLC的第二个元素相同的数据版本,再往后读一个版本,得到处于全局读一致的所述目标数据项,其中,所述版本链从新到旧有序存储且满足读已提交规则。

[0072] 在一种可能实施方式中,所述协调节点设备对所述目标事务的验证过程包括:

[0073] 汇总各个数据节点设备返回的依赖关系和异常检测结果,进行全局异常检测,以确定提交所述目标事务或回滚所述目标事务。

[0074] 在一种可能实施方式中,所述依赖关系为依赖图,所述全局异常检测的过程包括:

[0075] 基于各个数据节点设备返回的所述目标事务的多个子事务的依赖图,构造所述目标事务的全局依赖图;

[0076] 若所述全局依赖图不存在环且没有破坏完整性约束,确定所述目标事务符合可串行化隔离级别;

[0077] 若所述全局依赖图不存在环且完整性约束被破坏,确定所述目标事务符合可重复读隔离级别。

[0078] 在一种可能实施方式中,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

[0079] 若所述全局依赖图中首尾相接的子事务由两个读操作组成,确定存在读偏序异常或台阶式异常,在所述分布式数据库系统的隔离级别高于或等于快照读历史时,确定回滚所述目标事务。

[0080] 在一种可能实施方式中,若所述全局依赖图存在环,所述全局异常检测的过程还包括:

[0081] 若所述全局依赖图中首尾相接的子事务由读写操作组成,确定存在写偏序异常或锯齿波异常,在所述分布式数据库系统的隔离级别高于或等于可重复读时,确定回滚所述目标事务。

[0082] 一方面,提供了一种计算机设备,该计算机设备包括一个或多个处理器和一个或多个存储器,该一个或多个存储器中存储有至少一条计算机程序,该至少一条计算机程序由该一个或多个处理器加载并执行如上述任一种可能实现方式的事务处理方法。

[0083] 一方面,提供了一种存储介质,该存储介质中存储有至少一条计算机程序,该至少一条计算机程序由处理器加载并执行如上述任一种可能实现方式的事务处理方法。

[0084] 一方面,提供了一种计算机程序产品或计算机程序,该计算机程序产品或计算机程序包括至少一条程序代码,该至少一条程序代码存储在计算机可读存储介质中,计算机设备的一个或多个处理器从该计算机可读存储介质中读取该至少一条程序代码,由该一个或多个处理器执行该至少一条程序代码,使得该计算机设备执行上述任一种可能实现方式的事务处理方法。

[0085] 本申请实施例提供的技术方案带来的有益效果至少包括:

[0086] 通过在目标事务的每个操作执行时全面检测出分布式数据库系统内各种各样的数据异常,保证数据状态的一致性,在保证数据状态一致性的基础上,这种事务处理机制既不完全依赖于封锁技术也不完全依赖于依赖图技术,避免了限制分布式数据库系统的并发度,能够提升分布式数据库系统的事务处理效率。

附图说明

[0087] 为了更清楚地说明本申请实施例中的技术方案,下面将对实施例描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本申请的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其他的附图。

[0088] 图1是本申请实施例提供的一种事务处理方法的实施环境示意图;

[0089] 图2是本申请实施例提供的一种垂边的示意图;

[0090] 图3是本申请实施例提供的一种天边的示意图;

[0091] 图4是本申请实施例提供的一种读写斜边的示意图;

[0092] 图5是本申请实施例提供的一种只读斜边的示意图;

[0093] 图6是本申请实施例提供的一种一致性状态边的原理图;

[0094] 图7是本申请实施例提供的一种静边的示意图;

[0095] 图8是本申请实施例提供的一种锯齿波式异常的原理图;

[0096] 图9是本申请实施例提供的一种事务处理方法流程图;

[0097] 图10是本申请实施例提供的一种获取目标检测结果的流程图;

[0098] 图11是本申请实施例提供的一种获取目标检测结果的流程图;

[0099] 图12是本申请实施例提供的一种基于HLC建立全局快照的原理图;

[0100] 图13是本申请实施例提供的四个事务在两个节点上构成的分布式事务读异常的原理图;

[0101] 图14是本申请实施例提供的三个事务在两个节点上构成的分布式事务写异常的原理图;

[0102] 图15是本申请实施例提供的一种事务处理装置的结构示意图;

[0103] 图16是本申请实施例提供的一种计算机设备的结构示意图。

具体实施方式

[0104] 为使本申请的目的、技术方案和优点更加清楚,下面将结合附图对本申请实施方式作进一步地详细描述。

[0105] 本申请中术语“第一”“第二”等字样用于对作用和功能基本相同的相同项或相似

项进行区分,应理解,“第一”、“第二”、“第n”之间不具有逻辑或时序上的依赖关系,也不对数量和执行顺序进行限定。

[0106] 本申请中术语“至少一个”是指一个或多个,“多个”的含义是指两个或两个以上,例如,多个第一位置是指两个或两个以上的第一位置。

[0107] 在介绍本申请实施例之前,需要引入一些云技术领域内的基本概念:

[0108] 云技术(Cloud Technology):是指在广域网或局域网内将硬件、软件、网络等系列资源统一起来,实现数据的计算、储存、处理和共享的一种托管技术,也即是基于云计算商业模式应用的网络技术、信息技术、整合技术、管理平台技术、应用技术等的总称,可以组成资源池,按需所用,灵活便利。云计算技术将变成云技术领域的重要支撑。技术网络系统的后台服务需要大量的计算、存储资源,如视频网站、图片类网站和更多的门户网站。伴随着互联网行业的高度发展和应用,将来每个物品都有可能存在自己的识别标志,都需要传输到后台系统进行逻辑处理,不同程度级别的数据将会分开处理,各类行业数据皆需要强大的系统后盾支撑,均能通过云计算来实现。

[0109] 云存储(Cloud Storage):是在云计算概念上延伸和发展出来的一个新的概念,分布式云存储系统(以下简称存储系统)是指通过集群应用、网格技术以及分布存储文件系统等功能,将网络中大量各种不同类型的存储设备(存储设备也称之为存储节点)通过应用软件或应用接口集合起来协同工作,共同对外提供数据存储和业务访问功能的一个存储系统。

[0110] 数据库(Database):简而言之可视为一种电子化的文件柜——存储电子文件的处所,用户可以对文件中的数据进行新增、查询、更新、删除等操作。所谓“数据库”是以一定方式储存在一起、能与多个用户共享、具有尽可能小的冗余度、与应用程序彼此独立的数据集合。

[0111] 本申请实施例所涉及的数据库系统,可以是单机数据库系统、单机以事务为主的数据库系统、单机以分析型为主但需要事务处理能力的数据库系统,可以是NoSQL (Non-relational SQL,泛指非关系型数据库)体系,还可以是分布式数据库系统、分布式大数据处理系统,对于分布式数据库系统而言,当不同变量分布存储在不同的物理节点上时,对应于数据状态一致性模型中存在两个及两个以上变量的情况,数据状态一致性模型以及相应的事务处理流程均将在后文的各个实施例中进行详细说明,这里不做赘述。

[0112] 在数据库系统中可以包括至少一个节点设备,每个节点设备的数据库中存储有多个数据表,每个数据表可以用于存储一个或多个数据项(也称为变量版本)。其中,节点设备的数据库可以为任一类型的分布式数据库,可以包括关系型数据库或者非关系型数据库中至少一项,例如SQL (Structured Query Language,结构化查询语言)数据库、NoSQL、NewSQL (泛指各种新式的可拓展/高性能数据库)等,在本申请实施例中对数据库的类型不作具体限定。

[0113] 在一些实施例中,本申请实施例还可以应用于一种基于区块链技术的数据库系统(以下简称为“区块链系统”),上述区块链系统在本质上属于一种去中心化式的分布式数据库系统,采用共识算法保持区块链上不同节点设备所记载的账本数据一致,通过密码算法保证不同节点设备之间账本数据的加密传送以及不可篡改,通过脚本系统来拓展账本功能,通过网络路由来进行不同节点设备之间的相互连接。

[0114] 在区块链系统中可以包括一条或多条区块链,区块链是一串使用密码学方法相关联产生的数据块,每一个数据块中包含了一批网络交易的信息,用于验证其信息的有效性(防伪)和生成下一个区块。

[0115] 区块链系统中节点设备之间可以组成点对点(Peer To Peer,P2P)网络,P2P协议是一个运行在传输控制协议(Transmission Control Protocol,TCP)协议之上的应用层协议。在区块链系统中,任一节点设备可以具备如下功能:1)路由,节点设备具有的基本功能,用于支持节点设备之间的通信;2)应用,用于部署在区块链中,根据实际业务需求而实现特定业务,记录实现功能相关的数据形成账本数据,在账本数据中携带数字签名以表示数据来源,将账本数据发送至区块链系统中的其他节点设备,供其他节点设备在验证账本数据来源以及完整性成功时,将账本数据添加至临时区块中,其中,应用实现的业务可以包括钱包、共享账本、智能合约等;3)区块链,包括一系列按照先后的时间顺序相互接续的区块,新区块一旦加入到区块链中就不会再被移除,区块中记录了区块链系统中节点设备提交的账本数据。

[0116] 在一些实施例中,每个区块中可以包括本区块存储交易记录的哈希值(本区块的哈希值)以及前一区块的哈希值,各区块通过哈希值连接形成区块链,另,区块中还可以包括有区块生成时的时间戳等信息。

[0117] 在介绍本申请实施例之前,由于数据库系统中事务并发控制的正确程度可以通过一致性和隔离性来描述,下面对一致性和隔离性进行解释说明:

[0118] 一、隔离性

[0119] 事务隔离级别通过能否规避某种数据异常而进行定义,可能涉及到的数据异常包括:1)脏读,指一个事务读取到了另一事务尚未提交的数据项;2)不可重复读,指一个事务对同一数据项重复读取两次却得到了不同的结果;3)幻读,指事务在操作过程中进行两次谓词查询(范围查询),第二次查询的结果包含了第一次查询的结果中未出现的数据项或者缺少了第一次查询的结果中出现的数据项。

[0120] 基于能够解决上述三种数据异常,数据库国际标准ANSI(American National Standards Institute,美国国家标准学会)SQL提出四种隔离级别,以对上述三种已知的数据异常加以区分,目的是在允许某些数据异常存在的情况下提高事务处理效率。

[0121] 这四种隔离级别分别包括:1)读未提交:允许如上三种数据异常发生;2)读已提交:不允许脏读发生;3)可重复读:不允许脏读、不可重复读发生;4)可串行化:如上三种数据异常均不能发生。ANSI SQL定义的四隔离级别与数据异常之间的关系可以采用下表1进行直观展示:

[0122] 表1

[0123]

隔离级别	脏写	脏读	不可重复读	幻读
读未提交	不允许	允许	允许	允许
读已提交	不允许	不允许	允许	允许
可重复读	不允许	不允许	不允许	允许
可串行化	不允许	不允许	不允许	不允许

[0124] 可以看出,这四种隔离级别均不允许脏写异常发生,脏写异常是指两个未提交事务修改了同一个数据项。在ANSI SQL制定标准时已知的数据异常种类不多,后续不断有新

的数据异常被发现,除了上述几种数据异常之外,目前已知的数据异常还包括:丢失更新异常、读偏序异常、写偏序异常、串行并发现象 (Serial-Concurrent-Phenomenon) 异常、交叉现象 (Cross-Phenomenon) 异常等。

[0125] 在ANSI SQL标准的基础上,Jim Grey等人重新定义了六种隔离级别,采用更多的异常和更多的层级对隔离级别进行定义,这六种隔离级别与对应数据异常之间的关系可以采用下表2进行直观展示:

[0126] 表2

[0127]	隔离级别	脏写	脏读	游标丢失 更新	丢失更新	不可重复 读	幻读	读偏序	写偏序
	读未提交	不允许	允许	允许	允许	允许	允许	允许	允许
	读已提交	不允许	不允许	允许	允许	允许	允许	允许	允许
	游标稳定	不允许	不允许	不允许	有时允许	有时允许	允许	允许	有时允许
	可重复读	不允许	不允许	不允许	不允许	不允许	允许	不允许	不允许
	快照	不允许	不允许	不允许	不允许	不允许	有时允许	不允许	允许
	可串行化	不允许	不允许	不允许	不允许	不允许	不允许	不允许	不允许

[0128] 二、一致性

[0129] 数据库的一致性定义为:在事务的操作下,数据库的数据状态从一个一致的状态变迁为另一个一致的状态。上述“一致的状态”是指满足数据库系统预先定义的一些规则的数据状态,比如,这些规则可以包括约束、级联、触发器以及三者之间任意的组合(属于数据的逻辑语义),写偏序异常违反的就是特定数据之间的约束,这里的约束属于用户语义所限定的数据的一致性。

[0130] 对于整个数据库系统而言,一致性还包括一层系统级的含义,是指要想保证数据在数据库系统中保持一致,还要求数据库系统符合两个特性,一个是可串行性 (Serializability),另一个是可恢复性 (Recoverability)。可串行性也即是说上述在隔离性中所定义到的可串行化隔离级别,可串行性保证了数据不会被并发操作改坏,而可恢复性是指已经提交的事务未曾读过被回滚的事务写过的数据(指不会发生脏读异常),可恢复性保证了事务被回滚后数据回到之前的一致状态,被回滚的事务不会对数据的一致性造成影响,数据库系统的一致性状态是可恢复的。

[0131] 基于上述概念可知一致性和隔离性与数据异常是息息相关的,目前,在数据库系统中如何识别并规避数据异常是一个尤为关键的问题。数据库的并发访问控制算法即可用于提供ANSI SQL标准定义的隔离级别,但在不同的数据库系统中其实现技术各不相同。

[0132] 例如,DB2、Informix等数据库系统的可串行化隔离级别都是使用封锁技术,依赖锁的互斥机制来避免数据异常的发生;在MySQL数据库系统中,也是使用封锁技术实现可串行化隔离级别,此外,还使用SI (Snapshot Isolation,快照隔离) 技术实现可重复度和读已提交隔离级别;在PostgreSQL数据库系统中,使用SSI (Serializable Snapshot Isolation,可串行化快照隔离) 技术实现可串行化隔离级别,使用SI技术实现可重复度和读已提交隔离级别。又例如,在Spanner分布式数据库系统中,使用Truetime (真实时间) 机

制实现线性可串行化(线性可串行化是指分布式系统下结合了分布式一致性和事务一致性的一种与隔离级别有关联的级别,本质上包括了可串行化);在CockroachDB分布式数据库系统中,使用因果一致性和类似SSI的技术实现了可串行化和因果一致的结合。但是,上述这些数据库系统的隔离级别层次不够丰富,导致数据库系统的灵活度和事务处理效率不够好。

[0133] 根据异常的定义、隔离级别的定义,构建并发访问控制算法,是每个数据库事务处理技术的基础,但不管是ANSI SQL标准定义的四种隔离级别,还是后续发展的六种隔离级别,在此基础上构建的事务处理体系仍然存在下述四类问题:

[0134] 第一,数据异常类型不完整,定义隔离级别时采用的数据异常类型具有较大局限性。

[0135] 第二,数据异常的形式化定义不严格,在本申请实施例中将对各类数据异常给出严格的定义,并分析了传统的定义中不严格之处。

[0136] 第三,传统的隔离级别体系大部分都依赖于封锁(Locking)技术,但封锁技术只是并发访问控制技术的一种,还有很多其他的并发访问控制技术,且封锁技术严重限制了数据库系统的并发度,会导致事务处理效率低下,故传统的隔离级别体系已经不合时宜。

[0137] 第四,对于分布式事务和隔离级别没有系统化的分析和描述,只是沿用传统的ANSI SQL标准。对于分布式事务场景缺乏系统地研究隔离级别和分布式事务的关联关系,缺乏对分布式事务的隔离级别的定义和实现技术。

[0138] 有鉴于此,本申请实施例提供一种事务处理方法,给出了在事务处理过程中并发访问控制算法,分别包括应用于单机数据库系统的单机并发访问控制算法以及应用于分布式数据库系统的分布式并发访问控制算法,通过数据状态模型完善地、系统化地针对各种数据异常提供一种通用形式定义,提供了完整的隔离级别,下面将进行详述。

[0139] 图1是本申请实施例提供的一种事务处理方法的实施环境示意图。参见图1,本实施例可以应用于分布式数据库系统,该系统中可以包括网关服务器101、全局时间戳生成集群102、分布式存储集群103以及分布式协调系统104(例如ZooKeeper),在分布式存储集群103中可以包括数据节点设备和协调节点设备。

[0140] 其中,网关服务器101用于接收外部的读写请求,并将读写请求对应的读写事务分发至分布式存储集群103,比如,用户在登录终端上的应用客户端之后,触发应用客户端生成读写请求,调用分布式数据库系统提供的API(Application Programming Interface,应用程序编程接口)将该读写请求发送至网关服务器101,比如,该API可以是MySQL API(一种关系型数据库系统提供的API)。

[0141] 在一些实施例中,该网关服务器101可以与分布式存储集群103中的任一个数据节点设备或任一协调节点设备合并并在同一个物理机上,也即是,让某个数据节点设备或协调节点设备充当网关服务器101。

[0142] 全局时间戳生成集群102用于生成全局事务的全局提交时间戳(Global Timestamp,Gts),该全局事务又称为分布式事务,是指涉及到多个数据节点设备的事务,例如全局读事务可以涉及到对多个数据节点设备上存储数据的读取,又例如,全局写事务可以涉及到对多个数据节点设备上的数据写入。全局时间戳生成集群102在逻辑上可以视为一个单点,但在一些实施例中可以通过一主三从的架构来提供具有更高可用性的服务,采

用集群的形式来实现该全局提交时间戳的生成,可以防止单点故障,也就规避了单点瓶颈问题。

[0143] 可选地,全局提交时间戳是一个在分布式数据库系统中全局唯一且单调递增的时间戳标识,能够用于标志每个事务全局提交的顺序,以此来反映出事务之间在真实时间上的先后关系(事务的全序关系),全局提交时间戳可以采用物理时钟、逻辑时钟、混合物理时钟或者混合逻辑时钟(Hybrid Logical Clock,HLC)中至少一项,本申请实施例不对全局提交时间戳的类型进行具体限定。

[0144] 在一个示例性场景中,全局提交时间戳可以采用混合物理时钟的方式生成,全局提交时间戳可以由八字节组成,其中,前44位可以为物理时间戳的取值(也即Unix时间戳,精确到毫秒),这样共计可以表示 2^{44} 个无符号整数,因此理论上一共可以表示约为

$557.8 \left(\frac{2^{44}}{1000 * 60 * 60 * 24 * 365} = 557.8 \right)$ 年的物理时间戳,其中,后20位可以为在某一毫秒内的单调递增计数,这样每毫秒有 2^{20} 个(约100万个)计数,基于上述数据结构,如果

单机(任一数据节点设备)的事务吞吐量为10w/s,理论上可以支持包含1万个节点设备的分布式存储集群103,同时,全局提交时间戳的数量代表了系统理论上所能支持的总事务数,基于上述数据结构,理论上系统可以支持 $(2^{44}-1) * 2^{20}$ 个事务。这里仅仅是对一种全局提交时间戳的定义方法的示例性说明,根据业务需求的不同,可以对全局提交时间戳的位数进行扩展,以满足对更多的节点数、事务处理数的支持,本申请实施例不对全局提交时间戳的定义方法进行具体限定。

[0145] 在一些实施例中,该全局时间戳生成集群102可以是物理独立的,也可以和分布式协调系统104(例如ZooKeeper)合并到一起。

[0146] 其中,分布式存储集群103可以包括数据节点设备和协调节点设备,每个协调节点设备可以对应于至少一个数据节点设备,数据节点设备与协调节点设备的划分是针对不同事务而言的,以某一全局事务为例,全局事务的发起节点可以称为协调节点设备,全局事务所涉及的其他节点设备称为数据节点设备,数据节点设备或协调节点设备的数量可以是一个或多个,本申请实施例不对分布式存储集群103中数据节点设备或协调节点设备的数量进行具体限定。由于本实施例所提供的分布式数据库系统中缺乏全局事务管理器,因此在该系统中可以采用XA(eXtended Architecture,X/Open组织分布式事务规范)/2PC(Two-Phase Commit,二阶段提交)技术来支持跨节点的事务(全局事务),保证跨节点写操作时数据的原子性和一致性,此时,协调节点设备用于充当2PC算法中的协调者,而该协调节点设备所对应的各个数据节点设备用于充当2PC算法中的参与者。

[0147] 可选地,每个数据节点设备或协调节点设备可以是单机设备,也可以采用主备结构(也即是为了一主多备集群),如图1所示,以节点设备(数据节点设备或协调节点设备)为一主两备集群为例进行示意,每个节点设备中包括一个主机和两个备机,可选地,每个主机或备机都对应配置有代理(agent)设备,代理设备可以与主机或备机是物理独立的,当然,代理设备还可以作为主机或备机上的一个代理模块,以节点设备1为例,节点设备1包括一个主数据库及代理设备(主database+agent,简称主DB+agent),此外还包括两备数据库及代理设备(备database+agent,简称备DB+agent)。

[0148] 在一个示例性场景中,每个节点设备所对应的主机或备机的数据库实例集合称为

一个SET(集合),例如,假设某一节点设备为单机设备,那么该节点设备的SET仅为该单机设备的数据库实例,假设某一节点设备为一主两备集群,那么该节点设备的SET为主机数据库实例以及两个备机数据库实例的集合,此时可以基于云数据库的强同步技术来保证主机的数据与备机的副本数据之间的一致性,可选地,每个SET可以进行线性扩容,以应付大数据场景下的业务处理需求,在一些金融业务场景下,全局事务通常是指跨SET的转账。

[0149] 分布式协调系统104可以用于对网关服务器101、全局时间戳生成集群102或者分布式存储集群103中至少一项进行管理,可选地,技术人员可以通过终端上的调度器(scheduler)访问该分布式协调系统104,从而基于前端的调度器来控制后端的分布式协调系统104,实现对各个集群或服务器的管理。例如,技术人员可以通过调度器来控制ZooKeeper将某一个节点设备从分布式存储集群103中删除,也即是使得某一个节点设备失效。

[0150] 上述图1仅是提供了一种轻量级的全局事务处理的架构图,是一种类分布式数据库系统。整个分布式数据库系统可以看作是共同维护一个逻辑上的大表,这个大表中存储的数据通过主键被打散到分布式存储集群103中的各个节点设备中,每个节点设备上存储的数据是独立于其他节点设备的,从而实现了节点设备对逻辑大表的水平切分。由于在上述系统中能够将各个数据库中各个数据表水平切分后进行分布式地存储,因此,这种系统也可以形象地称为具有“分库分表”的架构。

[0151] 在上述分布式数据库系统中,已经基于XA/2PC算法实现了写操作时数据的原子性和一致性,而读操作的数据一致性问题,需要通过构造一个轻量的、去中心化的分布式事务处理机制来改善,从技术的角度来看,分布分表架构缺乏一个全局事务管理器,也就缺乏分布式事务处理能力,通过构造上述轻量的、去中心化的分布式事务处理机制,能够为分布式数据库系统提供水平扩展等能力,并且保证分布式数据库系统简单易推广、事务处理效率更高,必将对传统并发控制方式所设计的分布式数据库架构产生极大冲击,具体的分布式事务处理机制将在下个实施例中进行详述。

[0152] 本申请实施例提供的事务处理方法,可以应用于上述采用了分库分表架构的分布式系统中,例如,该分布式系统为分布式事务型数据库系统,当然也可以是分布式关系型数据库系统,此外,本申请实施例提供的事务处理方法也可以应用于一些单机数据库系统中,对于分布式系统而言,需要分布式事务处理能力,且为了提高事务处理效率以应对不同用户在不同场景的应用需求,还需要提供丰富灵活的隔离级别。

[0153] 在一些实施例中,上述网关服务器101、全局时间戳生成集群102、分布式存储集群103以及分布式协调系统104所构成的分布式数据库系统,可以视为一种向用户终端提供数据服务的服务器,该服务器可以是独立的物理服务器,也可以是多个物理服务器构成的服务器集群或者分布式系统,还可以是提供云服务、云数据库、云计算、云函数、云存储、网络服务、云通信、中间件服务、域名服务、安全服务、CDN(Content Delivery Network,内容分发网络)、以及大数据和人工智能平台等基础云计算服务的云服务器。可选地,上述用户终端可以是智能手机、平板电脑、笔记本电脑、台式计算机、智能音箱、智能手表等,但并不局限于此。终端以及服务器可以通过有线或无线通信方式进行直接或间接地连接,本申请在此不做限制。

[0154] 在介绍本申请实施例之前,首先将对数据库系统中所涉及到的一些基本术语以及

符号表示进行介绍:

[0155] 事务:事务是数据库管理系统在执行操作的过程中的一个逻辑单位,由一个有限的数据库操作序列构成,是数据库系统操作的最小执行单位。

[0156] 变量:事务是数据库关系系统中的一个数据单位,变量是数据库操作的作用者(或者说操作对象),一个变量可以包含若干变量版本(在后文中也简称为“版本”),每当事务对变量进行更新时,则会添加新的变量版本,变量的各个变量版本可以以自然数作为版本号标识,版本号越大,则表示变量版本越新。

[0157] 操作:一个数据库操作由操作类型、事务、变量版本三部分构成,其中,操作类型可以包括读(R)和写(W)两种。例如,事务T对变量x进行更新,生成了变量x的新版本i,上述读操作则可以记为 $W_T(x_i)$;又比如,事务T读取变量x的版本i的值,上述写操作可以记为 $R_T(x_i)$ 。

[0158] 事务数据集合:数据集合是由若干个变量版本所构成的集合,且集合中每个变量至多仅包含一个版本,可以记为数据集合 $DS(T) = \{x_i, y_j, z_k, \dots | x, y, z \text{ 为变量}, i, j, k \text{ 为版本号}\}$ 。

[0159] 数据库系统中的每个事务拥有两个事务数据集合,分别为事务的写集和事务的读集,其含义为:写集 $DS_W(T)$ 用于存储事务T所写入的新的数据版本,读集 $DS_R(T)$ 用于存储事务T所读取的数据版本。

[0160] 版本读取集合:版本读取集合是由若干个事务所构成的集合,表示读取某一变量版本的全部事务,可以记为版本读取集合 $TS(x_i) = \{T_i, T_j, T_k, \dots | i, j, k \text{ 为事务ID}\}$,版本读取集合中既可以包括已提交事务,也可以包括未提交事务,其中,事务ID也即是事务标识(Identification)。

[0161] 本申请实施例所提供的事务处理方法,是在数据状态一致性模型(简称为“数据状态模型”)的基础上运行的,而本申请所提供的数据库事务处理技术范围内第一次、全面、系统地提出一个模型,该模型揭示了各种数据异常的本质,而且统一了各种数据异常的描述和表达,解释了数据异常和并发事务之间的关联关系,并且,还能够发现出更多新的、未知的数据异常,能够更好地保证数据库系统的一致性。

[0162] 需要说明的是,在数据状态一致性模型中包括两重基本规则,其一,禁止脏写数据异常,也即禁止并发写写同一个变量,可以采取锁的方式,禁止写写冲突。其二,禁止脏读数据异常,通过采取读已提交的规则,禁止脏读异常发生,即保证所读取到的数据一定是已经提交后的数据。

[0163] 在上述数据状态一致性模型中,可以将事务与事务操作的变量版本之间的关系绘制成图形表示,下面,将对数据状态一致性模型所涉及的几个图形概念进行定义:

[0164] 1、结点:每个变量版本为一个结点,由于一个变量可以包括若干版本,在图形表示中同一变量的若干版本按照由旧至新的顺序呈纵向排列,也即是说,在图形表示中,同一变量的若干版本呈一纵列,并且位于顶端的版本最旧,位于底端的版本最新。

[0165] 2、垂边:同一个变量的两个相邻版本之间所存在的垂直边,称为“垂边”。垂边表示同一变量中版本的变迁,更新操作(比如DML的UPDATE语句)可以造成数据状态的改变,从而生成一条垂边。图2是本申请实施例提供的一种垂边的示意图,如图2所示, t_0 时刻存在变量版本 x_0 ,在 t_1 时刻事务 T_1 将变量版本 x_0 更新为变量版本 x_1 ,记为 $W(x_1)$,在变量版本 x_0 与变

量版本x1之间的垂线段200即为一条“垂边”。

[0166] 需要注意的是,垂边包含“从变量的某一版本更新至另一版本”的含义,因此对于任一个事务而言,其垂边上端连接的变量版本,应加入到当前事务的读集中(无论是否实际对该变量版本执行过读操作)。另外,版本更新只能由单个事务进行,不能由并发事务进行,这样可以禁止写写并发,从而避免发生脏写数据异常。此外,垂边和垂边不能重合,垂边只能是两个相邻版本之间的边,从而可以避免发生丢失更新数据异常。

[0167] 3、斜边:两个不同变量的版本之间存在的边,称为“斜边”。斜边表示两个变量之间的一致性状态,斜边可以划分为写写斜边(又称为“WW斜边”或“天边”)、读写斜边(又称为“RW斜边”)、读读斜边(“RR斜边”或“只读斜边”),其含义分别为:

[0168] 1) 天边:同一个事务修改了两个或两个以上不同的变量,则这些变量在两两之间可以构成一条天边。换言之,在事务的写集中,任意两变量之间一定存在一条天边,天边的两端分别与两结点的上端相连。图3是本申请实施例提供的一种天边的示意图,如图3所示,t0时刻存在变量版本x0和变量版本y0,在t1时刻事务T1将变量版本x0更新为变量版本x1,将变量版本y0修改为变量版本y1,记为W(x1)和W(y1),在变量版本x1与变量版本y1的上顶点之间的连线300即为一条“天边”。

[0169] 需要说明的是,在一些实施例中,如果一个事务仅修改了一个变量,那么该事务的垂边的终端点可以称为天点,天点是天边的一种特殊形式,相当于天边的两端点重合,以图2为例,由于事务T1仅修改了一个变量版本x0,其垂边的终端点为变量版本x1,因此图2中的变量版本x1就是一个天点。此外,一个垂边不能跨越天点而存在,如果某一垂边跨越天点,则意味着两个并发事务在同时写同一个变量,会造成写写冲突即产生脏写数据异常,按照基本规则需要禁止脏写数据异常,因此垂边不能跨越天点。

[0170] 2) 读写斜边:某个事务在任何两个不同的变量间,先后存在读、写操作,则该事务所读的变量版本和所写的变量版本之间可以构成一条读写斜边。读写斜边的读端与结点下端相连,写端与结点上端相连。图4是本申请实施例提供的一种读写斜边的示意图,如图4所示,t0时刻存在变量版本x0和变量版本y0,在t1时刻事务T1将变量版本x0更新为变量版本x1,并读取变量版本y0,记为W(x1)和R(y0),在变量版本y0的下顶点与变量版本x1的上顶点之间的连线400即为一条“读写斜边”。

[0171] 3) 只读斜边:同一个事务读取两个或两个以上不同的变量,则这些变量在两两之间可以构成一条只读斜边。换言之,在事务的读集中,任意两变量之间一定存在一条只读斜边。只读斜边的两端分别与两结点的下端相连,也即是说只有当变量版本已经处于提交状态时才可以被事务读取到,此时执行读已提交规则。图5是本申请实施例提供的一种只读斜边的示意图,如图5所示,t0时刻存在变量版本x0和变量版本y0,随后某一事务将变量版本x0更新为变量版本x1,在t1时刻事务T1分别读取变量版本x1和变量版本y0,记为R(x1)和R(y0),在变量版本y0的下顶点与变量版本x1的下顶点之间的虚线500即为一条“只读斜边”。

[0172] 4、一致性状态边:

[0173] 一致性状态边由一个或多个斜边或垂边首尾相连构成,用于描述多个变量之间的一致性状态。例如,图3中变量版本x0与变量版本x1之间构成的垂边、变量版本x1与变量版本y1之间构成的天边、变量版本y0与变量版本y1之间构成的垂边,这三条边共同构成一个一致性状态边;又比如,图4中变量版本x0与变量版本x1之间构成的垂边、变量版本y0与变

量版本x1之间构成的读写斜边,这两条边共同构成一个一致性状态边;又比如,图5中变量版本y0与变量版本x1之间构成的只读斜边,该只读斜边可以独自构成一个一致性状态边。

[0174] 一致性状态边可能涉及多个事务的垂边或天边,也即是说,处于一致性状态边上的结点之间保持数据一致性,但这些结点并不一定是单个事务操作的,有可能涉及到多个并发事务。例如,图5中变量版本x0与变量版本x1之间构成的垂边以及变量版本y0与变量版本x1之间构成的只读斜边可以共同构成一致性状态边,但垂边是由某一事务执行更新操作而生成的,而只读斜边是由另一事务执行读取操作而生成的,虽然两者对应于不同的事务,但可以共同构成一个一致性状态边。

[0175] 在一些实施例中,垂边可以被一致性状态边忽略,因此垂边不会和其他斜边构成动边交叉(动边交叉意味着存在数据异常),但在一些特定情况下,垂边是一致性状态边的组成部分,此时垂边不可以被一致性状态边忽略。

[0176] 在一个示例中,仍以图5为例,变量版本x0与变量版本x1之间构成的垂边作为一致性状态边中最外层的边可以被忽略,而变量版本y0与变量版本x1之间构成的只读斜边可以作为一致性状态边。

[0177] 在另一个示例中,图6是本申请实施例提供的一种一致性状态边的原理图,请参考图6,事务T1读取变量版本z0并写入变量版本x1,记为“R(z0)W(x1)”,事务T2读取变量版本y0写入变量版本z1,记为“R(y0)W(z1)”,如600所示,变量版本x0与变量版本x1之间构成的垂边、变量版本z0与变量版本x1之间构成的RW斜边、变量版本z0与变量版本z1之间构成的垂边、变量版本y0与变量版本z1之间构成的RW斜边可以共同构成一致性状态边,这是由2个事务T1和T2构成的一致性状态边,其中变量版本x0与变量版本x1之间构成的垂边作为一致性状态边的最外层的边可省略,而变量版本z0与变量版本z1之间构成的垂边作为一致性状态边的内层的边不可省略,这是由于内层的垂边起着联系其他斜边的作用,因此有可能会和其他斜边构成动边交叉,这种情况下垂边不可省略。

[0178] 按照事务提交与否,一致性状态边可以划分为两种,下面分别进行介绍:

[0179] 1) 静边:指静态一致性状态边,静边上的结点由已经提交的事务生成,已提交事务的相关点(某个变量的状态)之间的边,是一条或多条RR斜边构成的一致性状态边,也即称为静边。静边反映了历史时刻上数据状态发生变化后,各个变量版本之间的一致性关系,历史态数据一旦在事务提交后形成,只能被读取而不能被篡改,在读取历史态数据时,静边可以保证读取到的变量之间是处于相同一致性状态的。

[0180] 图7是本申请实施例提供的一种静边的示意图,如图7所示,3个变量的初始状态分别为r11,r21,r31,随着时间的推移(图7中从上至下代表时间增长),不同事务修改了变量状态,变量状态发生变化,比如T1事务将r11修改为r12,此时r12、r21和r31之间的虚线构成一条静边701,T2事务将r21和r31分别修改为r22和r32,则此时r22、r32和r12之间的连线构成一条静边702,T3事务将r12和r32分别修改为r13和r33,则此时r13、r33和r22之间的虚线构成一条静边703。如果在介于t1和t2时刻中间的任一个时间点读取保持一致性的变量版本,由于r22、r32的修改尚未提交,因此此时保持一致性的只能是r12、r21和r31间的虚线静边701,因此读取到的变量版本为r12、r21和r31。

[0181] 2) 动边:指动态一致性状态边,动边是并发事务集合中的一部分事务共同读写变量而构成的边,在并发事务集合中至少有一个事务没有提交。动边至少由一个事务构成,也

可由多个事务构成,多个并发事务合并后可构成一条动边(并发事务合并条件在将后文进行介绍,这里不做赘述)。动边反映出了并发事务对数据状态的影响。

[0182] 静边和动边协同反映了时态数据库中全态数据(包括历史态数据和当前态数据)的获取方式和技术(历史态数据只可读),还反映了数据状态的形成方式和技术,也即是说,当前态数据和并发事务是否不存在数据异常使得某个事务可以提交。

[0183] 5、并发事务:

[0184] 在数据状态一致性模型中,每个已经完成的事务可以按逻辑时钟有序,即事务的提交时间戳全局有序。如图7中的时刻 $t_0 \sim t_3$ 可以采用逻辑时钟定序,旧时刻从 t_0 开始,按 t_0 、 t_1 、 t_2 、 t_3 等顺序逐步增长。

[0185] 在保证全局有序的情况下,并发事务是指并发执行的两个或两个以上的事务在事务执行期间涉及到对相同的数据项进行读写操作,以两事务为例进行说明,若两事务对相同的变量在同一段时间内进行了读写操作,则认为两事务并发执行,假设存在事务 T_1 和事务 T_2 ,若两事务满足 $t_s(T_1) < t_c(T_2)$ 且 $T_s(T_2) < t_c(T_1)$,并对相同的变量执行过读写操作,也即是说 T_1 的事务开始时刻 t_s 小于 T_2 的事务提交时刻 t_c ,且 T_2 的事务开始时刻 t_s 小于 T_1 的事务提交时刻 t_c ,且两者对相同的变量执行过读写操作,此时可以认为 T_1 和 T_2 互为并发事务。

[0186] 在一些实施例中,上述事务开始时刻有2种不同的含义:一种是事务启动时刻,另一种是事务的第一个读操作或写操作开始的时刻,两种含义均可以作为判定并发事务的条件,本申请实施例不对事务开始时刻的具体含义进行限定。

[0187] 在同一时间段内一系列并发事务可以构成一个并发事务集合 TS ,并发事务集合 $TS = \{T_1, T_2, \dots, T_n\}$,其中 $n \geq 2$, n 个事务中每个事务在该并发事务集合中均至少存在一个与其并发的任务,也即是说每个事务至少在该并发事务集合中存在“两事务并发执行”的情况, TS 中每个元素(即事务)至少能找到另一元素(即另一事务)与其满足两事务并发执行,并发事务集合也可以简称并发事务,一个并发事务集合中至少存在一条“动态一致性状态边”(即动边)。

[0188] 而事务提交算法,在本质上来说,是用于验证目标事务与并发事务之间是否存在动边交叉情况,如果不存在动边交叉,则认为不存在数据异常,目标事务可以提交,否则,认为存在数据异常,需要根据数据库系统的隔离级别,判断当前产生的数据异常在当前的隔离级别下是否禁止发生,从而决定目标事务进入提交阶段还是进入回滚阶段。

[0189] 6、动边交叉:

[0190] 动边交叉的定义如下:若两个并发事务操作两个不同变量,且每个事务分别操作不同变量的不同状态,且两条动边在不同状态之间互相跨越,这种情况称为动边交叉。上述2个不同的变量,其含义可以指2个不同的物理变量,如 x 、 y ,也可以是 x_1 、 x_2 ,当然,其含义也可以指2个不同的逻辑变量,比如 $\{x\}$ 、 $\{y_1, z_0\}$,其中, $\{x\}$ 和 $\{y_1, z_0\}$ 都是逻辑变量,代表若干个变量的组合。

[0191] 从本质上来说,动边从图形上看是在一个矩阵图(数据状态矩阵)上事务的操作在不同变量版本之间的线段,从集合的角度看,就是事务的读集和写集。所以动边交叉本质是两个并发事务 T_i 和 T_j 之间的读写集合是否符合某种条件,对于动边交叉的检测,在本申请实施例中提供了两种检测方式,下面分别进行说明。

[0192] 方式一:

[0193] 在初始化过程中,将动边交叉值 (Dynamic Line Intersection, DLI) 的初始值置为0,将变量状态值 (Variable State, VS) 的初始值置为0。其中,该动边交叉值用于表示事务Ti与事务Tj在数据状态矩阵中操作的不同数据项之间的线段交叉情况,该变量状态值用于表示事务Ti与事务Tj涉及操作到处于不同数据状态的变量情况。

[0194] 进一步地,若事务Ti的读集和事务Tj的写集存在交集且两者的交集不为空集,则令 $DLI = DLI++$,也即将DLI置为已有值加一所得的数值(这种操作可以俗称为“DLI自增1”);如两者的交集中某个非空元素的数据状态不同,则令 $VS = VS++$,也即将VS置为已有值加一所得的数值(VS自增1);

[0195] 若事务Ti的写集和事务Tj的读集存在交集且两者的交集不为空集,则令 $DLI = DLI++$,也即将DLI置为已有值加一所得的数值(DLI自增1);如两者的交集中某个非空元素的数据状态不同,则令 $VS = VS++$,也即将VS置为已有值加一所得的数值(VS自增1);

[0196] 在事务Ti的写集和事务Tj的写集不存在交集(或者说两者的交集为空集)的情况下,可以保证满足基本规则中禁止写写冲突发生的条件,也即禁止脏写数据异常,如果动边交叉值DLI大于或等于2,且VS大于或等于1,则认为存在动边交叉,也即存在数据异常。在满足动边交叉的判定条件的基础上,可以根据变量状态值VS判断数据异常类型,VS=1时数据异常类型是读异常,包括1个变量的不可重复读异常、2个变量的读偏序异常、3个及3个以上变量的台阶式异常,VS=2时数据异常类型是写异常,包括2个变量的写偏序异常、3个及3个以上的锯齿波式异常。

[0197] 方式二:

[0198] 在初始化过程中,为每个事务各自分配一个布尔型,称之为upper,并将upper值初始化为false。该布尔型表示的是事务所构成的动边在数据状态矩阵中的垂直位置关系,以任意两个并发事务Ti和Tj为例,若 $Ti.upper = true$,则表明事务Ti在某变量上构成的动边相对另一事务Tj较新,也即是说,针对同一变量而言,事务Ti在该变量上操作了比事务Tj较新的变量版本。

[0199] 进一步地,若事务Ti的读集和事务Tj的写集存在交集,则对于交集的每一个变量x,若事务Ti读变量x的版本 $\geq$ 事务Tj写变量x的版本,比如事务Ti读取变量版本x2、事务Tj写入变量版本x1,则置 $Ti.upper = true$,否则,若事务Ti读变量x的版本 $<$ 事务Tj写变量x的版本,比如事务Ti读取变量版本x1、事务Tj写入变量版本x2,则置 $Tj.upper = true$ 。该规则对Ti的写集和Tj的读集比较的情况,可对事务Ti以及事务Tj的upper值进行同理赋值。

[0200] 若事务Ti的读集和事务Tj的读集存在交集,则对于交集的每一个变量x,若事务Ti读变量x的版本 $>$ 事务Tj读变量x的版本,比如事务Ti读取变量版本x2、事务Tj读取变量版本x1,则置 $Ti.upper = true$;若事务Ti读变量x的版本 $<$ 事务Tj读变量x的版本,比如事务Ti读取变量版本x1、事务Tj读取变量版本x2,则置 $Tj.upper = true$;若事务Ti读变量x的版本=事务Tj读变量x的版本,比如事务Ti和事务Tj均读取了变量版本x1,则不做任何操作。

[0201] 在事务Ti的写集和事务Tj的写集不存在交集(或者说二者的交集为空集)的情况下,可以保证满足基本规则中禁止写写冲突发生的条件,也即禁止脏写数据异常。在上述规则中,分别对事务Ti的写集和事务Tj的读集进行比较、对事务Ti的读集和事务Tj的写集进行比较、对事务Ti的读集和事务Tj的读集进行比较之后,若如果 $Ti.upper = Tj.upper = true$,则认为存在动边交叉,也即存在数据异常。

[0202] 动边交叉即构成数据异常,分析如下:动边由某个事务构成,如果该事务能够提交成功则该事务形成的动边处于一致性状态,由于两条动边各自保持一个一致性状态边,而交叉意味着两条动边互相跨越了对方的一致性状态边,分别操作(读写)了不同状态的数据,因而必定存在数据异常,发生动边交叉的并发事务集合中至少其中一个事务需要回滚。所以,数据异常的产生原因在本质上是:并发事务操作了不处于同一数据状态下的数据,即造成数据异常。

[0203] 在动边交叉中,至少有一条动边包含一个未提交事务,需要说明的是,已提交的事务也有可能属于并发事务。

[0204] 在并发事务集合中,如果在2个相同的变量上存在至少2条动边,以2条动边为例,按照排列组合,每条边有RR斜边、RW斜边、WR斜边这三种可能(其中WW斜边被禁止写写冲突的基本规则所禁止,所以读和写组合四种可能中只能取三种),故一共有“ $C_3^1 * C_3^1$ ”9种组合。因是动边,每个操作对都是针对不同变量,比如,RW斜边针对变量X和Y,则展开式为“R1 (X0) W1 (Y1)”。而在一个组合对中,比如RW-RW组合,假设事务T1的第一个RW对包括变量X和Y,那么事务T2的第二个RW对则必须针对变量Y和X,这是因为要构成一条动边(而不是一条静边),所以一定要跨越不同数据对象的不同状态,则第一个RW对展开式为“R1 (X0) W1 (Y1)”的情况下,第二个RW对的展开式只能是“R2 (Y0) W2 (X1)”,所以RW-RW组合其完整的操作序列应为:R1 (X0) R2 (Y0) W1 (Y1) W2 (X1)。但注意操作序列不只是这么一种,在此仅以这一种为例进行说明,除此以外还包括多种操作序列可以构成动边交叉,后面会针对每种组合均进行具体讨论。

[0205] 上面提到的9种组合中的WR-WR、WR-RR、WR-RW三种关系,其中第一个“WR”操作对的展开式为“W1 (X1) R2 (X1)”,在读已提交规则下,其展开式还可能为“W1 (X1) C1 R2 (X1)”表明写事务已经提交结束,所以该关系不可能和后续的任何一个组合(RR、RW、WR三种之一)中的某个事务成为同一个事务,所以这三种组合不可能构成动边交叉。

[0206] 进一步分析,9种组合中的RW-RW、RR-RR、RW-RR、RW-WR、RR-WR、RR-RW关系,可以构成动边交叉。这些模式之间的关系如下表3所示,其中RW-RW、RW-WR构成基本模式,其余四种模型是从这两种基本模式演化而来,可以称为一级扩展模式。

[0207] 表3

[0208]

编号	模式名	操作序列	等价模式（加粗的部分表示可以合并为一个事务）	数据异常类型
1	RW-RW	R1(X0) R2(Y0) W1(Y1) W2(X1)	基本模式	写偏序
2	RW-RR	R1(X0) R2(Y0) W1(Y1) R2(X1)	一级扩展模式 R1(X0) R2(Y0) W1(Y1) W3(X1) C3 R2(X1) == R1(X0) R2(Y0) W1(Y1) W2(X1)	写偏序
3	RR-RW	R1(X0) R2(Y0) W2(X1) R1(Y1)	一级扩展模式 R1(X0) R2(Y0) W2(X1) W3(Y1) C3 R1(Y1) == R1(X0) R2(Y0) W2(X1) W1(Y1)	写偏序
4			一级扩展模式 R1(X0) R2(Y0) W2(X1) W3(Y1) C3 R1(Y1) == R1(X0) R2(Y0) W2(X1) W2(Y1) R1(Y1)	读偏序
5	RW-WR	R1(X0) W2(X1) W2(Y1) R1(Y1)	基本模式	读偏序

[0209]

6	RR-WR	R1(X0) R2(X1) W2(Y1) R1(Y1)	一级扩展模式 R1(X0) W3(X1) C3 R2(X1) W2(Y1) C2 R1(Y1) == R1(X0) W2(X1) W2(Y1) C2 R1(Y1)	读偏序
7	RR-RR	R1(X0) R2(Y0) R1(Y1) R2(X1) == R1(X0) R2(Y0) R2(X1) R1(Y1)	一级扩展模式 R1(X0) R2(Y0) W3(Y1) W3(X1) C3 R2(X1) R1(Y1) == R1(X0) W2(Y1) W2(X1) R1(Y1) R1(X0) R2(Y0) W4(Y1) C4 W3(X1) C3 R2(X1) R1(Y1) == R1(X0) W2(Y1) W2(X1) R1(Y1)	读偏序

[0210] 一级扩展模式本质上就是基本模式，在一级扩展模式的基础上，在一个读操作前扩展出一个其他的、并发的、已经提交完成的事务的写操作，并通过事务合并即可转化为一个基本模式，因此识别一级扩展模式和基本模式，这为识别复杂的并发事务操作序列提供了理论基础，而识别基本模式又为识别一级扩展模式提供了理论基础，这2个基础称为“模式等价原理”。

[0211] 根据模式等价原理，在一级扩展模式中扩展出的写操作，可以在同一个变量的两个不同事务的读操作之间的任何一个操作位置出现，这称为“模式扩展等价原理”。如表3中第二个写偏序模式“R1 (X0) R2 (Y0) W1 (Y1) W3 (X1) C3R2 (X1)”，事务T3的写操作和提交操作“W3 (X1) C3”可以在“R1 (X0)”和“R2 (X1)”之间的任何位置出现，比如，上述表3中第二个写偏

序模式等价的形式还可以包括：“R1 (X0) W3 (X1) R2 (Y0) W1 (Y1) R2 (X1)”、“R1 (X0) R2 (Y0) W3 (X1) W1 (Y1) R2 (X1)”。

[0212] 通过模式等价原理和模式扩展等价原理,表明任何一个基于两个变量的操作序列,如果存在表3中描述的任一种基本模式或者一级扩展模式(当然一级扩展模式本质上也是基本模式),则必定存在数据异常。否则不存在数据异常。

[0213] 下面将分别对上述六种模式(包括两种基本模式和四种一级扩展模式)的操作序列的可能实现方式进行说明:

[0214] 1) 2个变量上构成RW-RW关系:R1 (X0) R2 (Y0) W1 (Y1) W2 (X1),在RW-RW操作序列中,读操作和读操作可互换位置,写操作和写操作可互换位置。本动边交叉直接构成数据异常,对应于传统的写偏序异常。

[0215] 2) 2个变量上构成RW-RR关系:R1 (X0) R2 (Y0) W1 (Y1) R2 (X1),在RW-RR操作序列中,前2个读操作和读操作可互换位置,后2个写操作和读操作在不涉及同一个变量的情况下可互换位置。具体地,变量版本X1必然被别的事务写造成,所以必定有1个事务T3在R1 (X0) 和R2 (X1) 之间执行了W3 (X1)。此3个事务T1~T3是并发事务,构成写偏序异常。

[0216] 3) 2个变量上构成RR-RW关系:R1 (X0) R2 (Y0) W2 (X1) R1 (Y1),在RR-RW操作序列中,前2个读操作和读操作可互换位置。具体地,变量版本Y1必然被别的事务写造成,所以必定有1个事务T3在R2 (Y0) 之后、R1 (Y1) 之前执行了W3 (Y1)。此3个事务T1~T3是并发事务。如果将事务T1和事务T3合并为一个逻辑事务{T1,T3},由于其模式变为RW-RW,那么逻辑事务{T1,T3}与事务T2之间构成传统的写偏序异常,也即 $\{T1,T3\} \cap T2 = \text{传统的写偏序异常}$;如果将事务T2和事务T3合并为一个逻辑事务{T2,T3},由于其模式变为RW-WR,那么逻辑事务{T2,T3}与事务T1之间构成传统的读偏序异常,也即 $\{T2,T3\} \cap T1 = \text{传统的读偏序异常}$ 。

[0217] 4) 2个变量上构成RW-WR关系:R1 (X0) W2 (X1) W2 (Y1) R1 (Y1)。这一操作序列RW-WR构成传统的读偏序异常。在读已提交规则下,该操作序列实际上为“R1 (X0) W2 (X1) W2 (Y1) C2 R1 (Y1)”才能确保R1 (Y1) 成功执行。

[0218] 5) 2个变量上构成RR-WR关系:R1 (X0) W2 (Y1) R2 (X1) R1 (Y1),在RR-WR操作序列中,前2个读操作与写操作可互换位置,后2个读操作和读操作也可互换位置。具体地,变量版本X1必然被别的事务写造成,所以必定有1个事务T3在R1 (X0) 之后、R2 (X1) 之前执行了W3 (X1)。此3个事务T1~T3是并发事务,构成传统的读偏序异常。

[0219] 6) 2个变量上构成RR-RR关系:R1 (X0) R2 (Y0) R1 (Y1) R2 (X1),在RR-RR操作序列中,前2个读操作与读操作可互换位置,后2个读操作与读操作可互换位置。具体地,变量版本X1和变量版本Y1必然被别的事务写造成,所以可能有1个事务T3在R1 (X0) R2 (Y0) 之后、R1 (Y1) R2 (X1) 之前执行了W3 (Y1) W3 (X1);或者有两个事务分别执行了W3 (Y1) 和W4 (X1)。此3个或4个事务(T1~T3或者T1~T4),是并发事务,也对应于传统的读偏序异常。

[0220] 在发生动边交叉时,至少一条动边中的未提交事务需要回滚,从而可解决数据异常的产生原因,保证数据库系统中的数据状态一致性。

[0221] 7、数据状态历史:

[0222] 一个数据对象用X表示,X值的变化,即可构成一个历史序列,从变量的角度来说,变量版本的变迁构成一个历史序列。变量X的历史序列如下:

[0223] Data History=DH=X0 X1...Xi Xi+1...X

[0224] 其中, X_i 称为一个变量版本, $0 \leq i \leq m$, m 为变量 X 的最大版本号。

[0225] 其中, X_0 是数据的初始版本,表明数据对象 X 存在,值为 X_0 。

[0226] 其中, X_i 和 X_{i+1} 是相邻的变量版本, X_i 称为 X_{i+1} 的before-image(前像)。

[0227] 版本变迁是指当发生写操作,将变量版本 X_i 变迁为变量版本 X_{i+1} 。

[0228] 一个读操作或写操作发生,必定在数据状态的历史序列上,有一个与其对应的before-image。

[0229] 在一个示例中,以 $W1(X_0-X_1)$ 为例, X_0-X_1 表示事务 T_1 中,before-image是 X_0 ,产生的新版本是 X_1 ;同一个事务的连续多次写,不构成多个变量版本,即before-image相同时不构成多个变量版本。

[0230] 在一个示例中,以 $R2(X_2)$ 为例,表示事务 T_2 中,before-image是 X_2 。

[0231] 8、数据状态矩阵:

[0232] 水平方向: n 个变量,从左至右水平排列,分别称为 X 、 Y 、 Z ……,或者用 a_1 、 a_2 、 a_3 ……表示。

[0233] 垂直方向:单个变量的数据状态历史序列。

[0234] 由水平方向和垂直方向构成的一个由 n 个数据、最多 m 个数据状态的数据版本,构成一个 $n*m$ 的矩阵,称为数据状态矩阵。图7所示的静边图就是数据状态矩阵的一部分。

[0235] 并发事务之间所构成的斜边,在形式上是数据状态矩阵中不同变量版本间的连线,也是该数据状态矩阵的一部分。并发事务中每个并发事务构成的动边,是一个动态的、尚待验证是否能够保持合法状态的一致性状态边,如果事务能够成功提交,则该动边合法,动边也变为了静边。

[0236] 数据状态矩阵上有一组或多组并发事务正在操作变量版本,从而可以形成一条或多条动边。在某组并发事务中,如果只有一条动边,则该动边表示的事务必可提交,这是由于单个动边不违反一致性状态,即不跨越一致性状态边,也即没有动边交叉;如果有两条动边构成动边交叉,则至少有一条动边对应的事务需要回滚,以避免对另外一条动边构成的一致性数据状态造成动态破坏。

[0237] 本质上来讲,数据状态矩阵是并发事务的操作和生命空间,历史的事务在这个空间中沉淀(动边转化为静边),有冲突或无冲突的并发事务在这个空间中发生(当前存在的动边)。

[0238] 9、动边合并:

[0239] 两个并发的任务,每个任务的一致性状态边(即动边)要么存在数据异常(动边交叉),要么可以合并(即不存在数据异常)。也即是说,如果两个并发任务存在动边交叉,那么一定存在数据异常,否则,这两个任务的动边一定可以合并。

[0240] 针对两任务合并而言,可以划分为三个维度上的合并,即任务合并、变量合并以及数据状态合并,下面分别进行介绍:

[0241] 1) 任务合并:并发任务集合中,若任务 T_1 (表示为 $\{T_1\}$)和任务 T_2 (表示为 $\{T_2\}$)可以构成一条动边,则任务 T_1 和任务 T_2 可合并为一个逻辑任务(表示为 $\{T_1, T_2\}$),任务 T_1 和任务 T_2 的合并是指分别将任务 T_1 和任务 T_2 的读写集合进行合并,意味着任务 T_1 的读集和任务 T_2 的读集合并,任务 T_1 的写集和任务 T_2 的写集合并。任务合并的一个重要条件是合并在一起的任务之间不存在动边交叉,如果两任务存在动边交叉那么将无法合并。动边合并,形式

上是将并发事务之间合并为一个逻辑事务,实质上是将并发事务的读写集合分别进行合并。

[0242] 2) 变量合并:在事务合并的过程中,若合并后事务的读集中包含变量版本 X_i 和 Y_j ,合并后写集中包含变量版本 $X_{(i+1)}$ 和 $Y_{(j+1)}$,此时进一步地进行变量合并,可以将变量 X 和 Y 合并为一个逻辑变量 $\{X, Y\}$,即合并后事务的读集中包含变量版本 $\{X_i, Y_j\}$,合并后事务的写集中包含变量版本 $\{X_{(i+1)}, Y_{(j+1)}\}$ 。

[0243] 3) 数据状态合并:若事务合并的过程中,涉及某个变量的多个状态,如 X_i 和 $X_{(i+1)}$,则变量纵向合并时,可以变为 $\{X(i, i+1)\}$ 。

[0244] 动边合并定义:分别隶属于两个事务的每两条动边,如果不存在动边交叉,则两个事务合并为一个逻辑事务,遵循前述的事务合并规则;在事务合并后,两个事务的读集和写集遵循变量合并规则;但是对于同一个变量而言,不同变量版本即不同状态的数据版本之间不合并,即不进行数据状态合并。

[0245] 在此基础上,提出一种缩减规则,用于形式化地解释复杂的并发操作序列向基础操作序列化简的方式,以判断一个并发操作序列,上述表3中将扩展后的一级扩展模式向基本模式进行化简时,即采用了缩减规则。下面以一级扩展模式向基本模式合并为例,对两种不同的缩减规则进行说明。

[0246] 在一些实施例中,一级扩展模式向基本模式合并时,形如“W3(Y1)C3R1(Y1)”格式,其缩减规则为:如果是同一个状态的变量,写操作保留(去掉读操作),事务号保留读事务的事务号(去掉写操作的事务号),这相当于事务T1和事务T3之间的两条动边合并。如表3中编号为2、3、6的模式,都符合本条缩减规则。对于编号为3和4的即RR-RW模式,是由于采用了不同的缩减规则导致最终在化简后对应于不同的数据异常,具体地,如果采用前述规则进行合并,即视异常为写偏序(编号3)而不认为是读偏序(编号4)。

[0247] 在一些实施例中,在一级扩展模式向基本模式合并时,形如“R2(Y0)W3(Y1)”格式,其缩减规则为:如果是同一个变量,写操作保留(去掉读操作),事务号保留读事务的事务号(去掉写操作的事务号),变量状态保留最新的变量状态,这相当于事务T2和事务T3之间的两条动边合并。如表3中编号为7的两个一级扩展模式在化简时,都符合本条缩减规则,当然,编号4的RR-RW模式恰好是因为采用了本条缩减规则,所以被视为读偏序异常。

[0248] 在上述描述中,提供了数据状态一致性模型,满足上述各种约束条件的并发事务,也即是符合数据状态一致性模型。

[0249] 基于上述数据状态一致性模型,可以基于读已提交规则,对数据异常进行如下定义:

[0250] 1、异常点:

[0251] 当某个操作(读、写、提交、回滚)发生,动边交叉形成,则异常形成,称为异常形成点,简称异常点。异常点的形成,包括下述两种类型。

[0252] 1) 读写操作级:当读或写操作发生时,异常即形成,此时称为读写操作级异常异常点,简称操作级异常点。

[0253] 2) 事务级:非操作级异常,且当事务提交或回滚时,异常即形成,此时称为事务级异常异常点,简称事务级异常点。

[0254] 这两种不同类型的异常点所对应的异常,分别称为操作级异常和事务级异常。

[0255] 此外,这两种不同类型的异常点,还影响着事务的并发度,将在下述针对并发度的介绍中进行详述。

[0256] 2、并发度:

[0257] 两事务满足 $t_s(T_1) < t_c(T_2)$ 且 $t_s(T_2) < t_c(T_1)$, 则称为两事务并行,或并行事务。

[0258] 在两事务并行的基础上,如果两事务操作过至少同一个数据对象,称为事务并发,或并发事务。

[0259] 两个并发事务的读写集合的交集不为空集,获取共同操作的数据对象(实则是数据的某个版本对象),这些数据对象在每个事务中对应的第一个操作和最后一个操作可以构成一个区间,该区间称为冲突区,也称为第一异常区间。冲突区的上界(最大值点)到事务级异常点之间的区间,称为第二异常区间。

[0260] 冲突区/第一异常区间异常形成,即形成操作级异常,操作级异常对应的事务生命周期短,与其他事务的并发度低。此种情况称为并发度一度,简称一度。这里的生命周期短,并非用户的事务语义导致该事务的生命周期短,而是在异常已经发生导致该事务面临被迫回滚的风险,因此造成了事务的生命周期短。

[0261] 事务级异常对应的事务能够在第二异常区间继续存在(事务的生命期长),与其他事务的并发度高。此种情况称为并发度二度,简称二度。一度的并发度小于二度的并发度。

[0262] 3、异常分类:

[0263] (3.1) 按照数据异常涉及的变量个数,可以将数据异常划分为单变量异常和多变量异常。单变量异常是指只涉及1个变量的数据异常,多变量异常是指涉及2个及2个以上变量的数据异常。

[0264] (3.2) 按照数据异常涉及的变量是否存储在同一个数据节点设备(也称为子节点)上,可以将数据异常划分为分布式异常和局部异常。

[0265] A、分布式异常:如果一个多变量异常涉及的所有变量不都在一个子节点上存储,该异常称为分布式异常,一个分布式异常一定是多变量异常,但一个多变量异常也可能发生在同一个子节点上(也即涉及的所有变量都在一个子节点上存储),此时多变量异常为局部异常。

[0266] B、局部异常:一个单变量异常,该异常又称为局部异常(单变量异常必定是局部异常)。而幻读异常特殊,虽涉及到多个变量,但是以谓词为单位,读取到的一批数据在可以视为“一个”变量,而符合谓词被新加入的数据,是依附谓词的,因此幻读异常也属于局部异常。非分布式异常的数据异常,即是局部异常。

[0267] (3.3) 按照并发事务的读写操作是否直接针对数据状态矩阵中存在的变量版本以及异常发生的时机等因素,可以将数据异常划分广义异常和狭义异常。本申请实施例中提出的数据状态矩阵所能检测到的异常,属于狭义异常。

[0268] (3.3.1) 狭义异常:包含实体数据异常、虚体数据异常和约束异常。

[0269] A、实体数据异常:基于数据状态矩阵,并发事务的读写操作直接针对数据状态矩阵中已存在的变量版本而造成的异常,称为实体类数据异常。比如,脏写、脏读、中间读、不可重复读、丢失更新、读偏序、读写偏序、台阶式等异常,均属于实体数据异常。这类异常的特征是并发事务读写操作的数据对象已经在操作序列中存在,而不是通过谓词把相关数据对象包含进来。

[0270] B、虚体数据异常：基于数据状态矩阵，并发事务的读写操作通过谓词把相关数据对象进行包含而造成的异常，且异常的操作语义在操作发生时已经明确，称为虚体类数据异常。比如，幻读异常，在第二次谓词读操作发生时，即已经构成幻读异常，属于操作级语义异常。

[0271] C、约束异常：在事务提交节点，数据库数据的一致性状态违反了预定义的数据约束，称为约束异常。比如，谓词写偏序、写偏序、锯齿波式等异常，该类异常的操作语义在操作发生时尚不明确，只有等到结果形成后，才能根据某种预定义好的语义验证结果，属于结果级语义异常。由于该类异常和用户定义的约束有关，故本申请实施例中不将约束异常纳入数据状态一致性模型的考量范围内。

[0272] (3.3.2) 广义异常：除了狭义异常所描述的异常外，还包括了读半已提交异常。

[0273] 读半已提交异常：该异常是分布式场景下所发生的异常，其发生原因是分布式事务在各个节点设备上的提交时间存在不一致，当并发的读事务访问数据时，部分节点设备已经提交，而部分节点设备尚未提交，从而造成读取数据的不一致。由于本申请实施例对异常的讨论聚焦于单机场景，故本申请实施例中不将读半已提交异常纳入数据状态一致性模型的考量范围内。

[0274] 4、数据异常类型：

[0275] 按照上述数据状态一致性模型，在本申请实施例中定义了12种类型的数据异常，分别包括脏写数据异常、脏读数据异常、中间读数据异常、不可重复读数据异常、幻读数据异常、丢失更新数据异常、读偏序数据异常、台阶式数据异常、写偏序数据异常、谓词写偏序数据异常以及锯齿波式数据异常，下面进行详述。

[0276] 1) 脏写数据异常 (Dirty Write)：

[0277] 定义：操作序列形如 $w1[x1] \dots w2[x2] \dots$ 的数据异常，脏写数据异常属于操作级异常。

[0278] 但脏写数据异常的操作序列定义不能是： $w1[x] \dots w2[x] \dots ((c1 \text{ or } a1) \text{ and } (c2 \text{ or } a2) \text{ in any order})$ ，其中有序对 $(a2, c1)$ 或 $(a2, a1)$ 的组合，不会造成数据不一致，但由于执行写操作和回滚操作，都将耗费CPU (Central Processing Unit, 中央处理器) 资源，因此与其执行写操作后还得执行回滚操作，不如在定义中直接禁止发生。

[0279] 其中，对任一事务 T_i ($i \geq 1$) 而言， c_i 是指提交事务 T_i ， a_i 是指回滚事务 T_i ，比如 c_1 是指提交事务 T_1 ， a_2 是指回滚事务 T_2 ，后面将不做赘述。

[0280] 2) 脏读数据异常 (Dirty Read)：

[0281] 定义：操作序列形如 $w1[x1] \dots r2[x1] \dots (a1 \text{ and } c2)$ 的数据异常，脏读数据异常属于事务级异常。

[0282] 但脏读数据异常的操作序列定义不能是： $w1[x1] \dots r2[x1] \dots (a1 \text{ and } c2 \text{ in either order})$ ，这是由于如果 c_2 先完成，而 c_1 发生时，无论是允许事务 T_1 提交还是回滚事务 T_1 ，都会造成数据异常。

[0283] 在脏读数据异常的定义中，如果 a_1 变为 c_1 ，则没有数据异常，故只能是 a_1 。如果 c_2 变为 a_2 ，则不会有异常存在，故只能是 c_2 。如果 a_1 先发生，则 c_2 发生也必须回滚才能消除异常。所以只有执行到事务的提交或回滚操作时，才能有效识别是否存在数据异常，所以脏读数据异常是事务级异常。

[0284] 3) 中间读数据异常 (Intermediate Reads) :

[0285] 定义:操作序列形如 $w1(x1) \dots r2(x1) \dots w1(x2) \dots (c1 \text{ and } c2)$ 的数据异常,中间读数据异常属于事务级异常。

[0286] 但中间读数据异常的操作序列定义不能是: $w1(x1) \dots r2(x1) \dots w1(x2) \dots (c1 \text{ and } c2 \text{ in either order})$,这是由于如果 $c2$ 先完成,而 $c1$ 发生时,无论是允许事务 $T1$ 提交还是回滚事务 $T1$,都会造成数据异常。

[0287] 在中间读数据异常的定义中,如果 $c1$ 变为 $a1$,则该异常是脏读,故只能是 $c1$ 。如果 $c2$ 变为 $a2$,则不会有异常存在,故只能是 $c2$ 。

[0288] 4) 不可重复读数据异常 (Fuzzy or Non-Repeatable Read) :

[0289] 定义:操作序列形如 $r1[x0] \dots w2[x1] \dots c2 \dots r1[x1]$ 的数据异常,不可重复读数据异常属于操作级异常。

[0290] 但不可重复读数据异常的操作序列定义不能是: $r1[x0] \dots w2[x1] \dots c2 \dots r1[x1] \dots c1$,因为事务 $T2$ 已经提交,事务 $T1$ 即使发生 $c1$,为消除异常最终结果只能是事务 $T2$ 回滚。

[0291] 5) 幻读数据异常 (Phantom) :

[0292] 定义:操作序列形如 $r1[P] \dots w2[x \text{ in } P] \dots c2 \dots r1[P]$ 的数据异常,幻读数据异常属于操作级异常。

[0293] 但幻读数据异常的操作序列定义不能是: $r1[P] \dots w2[x \text{ in } P] \dots c2 \dots r1[P] \dots c1$,因为事务 $T2$ 已经提交,事务 $T1$ 即使发生 $c1$,为消除异常最终结果只能是事务 $T1$ 回滚。

[0294] 6) 丢失更新数据异常 (Lost Update) :

[0295] 定义:操作序列形如 $r1[x0] \dots w2[x1] \dots c2 \dots w1[x2] \text{ or } rc1[x0] \dots w2[x1] \dots c2 \dots w1[x2]$ 的数据异常,丢失更新数据异常属于操作级异常。

[0296] 但丢失更新数据异常的操作序列定义不能是: $r1[x0] \dots w2[x1] \dots c2 \dots w1[x2] \text{ or } rc1[x0] \dots w2[x1] \dots c2 \dots w1[x2] \dots c1$,因为事务 $T2$ 已经提交,事务 $T1$ 即使发生 $c1$,为消除异常最终结果只能是事务 $T1$ 回滚。

[0297] 7) 读偏序数据异常 (Read Skew) :

[0298] 定义:操作序列形如 $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots r1[y1]$ 的数据异常,读偏序数据异常属于操作级异常。

[0299] 但读偏序数据异常的操作序列定义不能是: $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots r1[y1] \dots (c1)$,事务 $T1$ 事实上只能回滚,没有别的选择。

[0300] 在一些实施例中,如果未定义有读已提交规则,则读偏序的定义可以修改为: $r1[x0] \dots w2[x1] \dots w2[y1] \dots r1[y1] \dots (c1 \text{ and } c2 \text{ in either order})$ 。

[0301] 8) 读写偏序数据异常 (Read and Write Skew) :

[0302] 定义:操作序列形如 $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots w1[y2]$ 的数据异常,读写偏序数据异常属于操作级异常。

[0303] 但读写偏序数据异常的操作序列定义不能是: $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots w1[y2] \dots (c1)$,事务 $T1$ 事实上只能回滚,没有别的选择。

[0304] 在一些实施例中,如果未定义有读已提交规则,则读写偏序数据异常的定义可以修改为: $r1[x0] \dots w2[x1] \dots w2[y1] \dots w1[y2] \dots (c1 \text{ and } c2 \text{ in either order})$ 。

[0305] 9) 台阶式数据异常 (Step Read Skew):

[0306] 定义:操作序列形如 $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots w3[y2] \dots w3[z1] \dots c3 \dots r1[z1]$ 的数据异常,台阶式数据异常又称阶梯式数据异常,属于操作级异常。

[0307] 但台阶式数据异常的操作序列定义不能是: $r1[x0] \dots w2[x1] \dots w2[y1] \dots c2 \dots w3[y2] \dots w3[z1] \dots c3 \dots r1[z1] \dots c1$,原因一是 $r1[z1]$ 一旦执行,那么动边交叉立即形成;二是该事务不可能被成功提交,除了选择回滚之外,没有别的方式可以解决异常,因此 $c1$ 出现在定义中无价值。

[0308] “ $w2[y1] \dots c2 \dots w3[y2]$ ”为一个台阶,该台阶模式可以扩展或变换出任意多个非 x 和 z 的变量和事务。比如再扩展出一个变量 a 和一个事务 $T4$ (事务情况如表4所示)、再扩展出三个变量 a 、 b 、 d 和三个事务 $T4$ 、 $T5$ 、 $T6$ (事务情况如表5所示)等。

[0309] 表4

事务T1	事务T2	事务T3	事务T4
R1-X0			
	W2-X1 W2-Y1 C2		
		W3-Y2 W3-a1 C3	
			W4-a2 W4-Z1 C4
R1-Z1			

[0311] 表5

事务 T1	事务 T2	事务 T3	事务 T4	事务 T5	事务 T6
R1-X0					
	W2-X1 W2-Y1 C2				
		W3-Y2			
		W3-a1 C3			
			W4-a2 W4-b1 C4		
				W5-b2 W5-d1 C5	
					W6-d2 W6-Z1
R1-Z1					

[0314] 需要说明的是,台阶式数据异常包括3个及3个以上的变量,扩展形式至少包括两个事务写同一个变量且其中一个事务已经提交。

[0315] 10) 写偏序数据异常 (Write Skew):

[0316] 定义:操作序列形如 $r1[x0] \dots r2[y0] \dots w1[y1] \dots w2[x1]$ 的数据异常,写偏序数据异常属于操作级异常。

[0317] 但写偏序数据异常的操作序列定义不能是： $r1[x0] \dots r2[y0] \dots w1[y1] \dots w2[x1] \dots$ ($c1$ and $c2$ in either order), 定义中第二个写操作完成, 即形成异常, 且可选择事务T1或者事务T2其中之一进行回滚。写偏序数据异常在变量x和y之间不必有约束定义, 并且也能被动边交叉识别。

[0318] 11) 谓词写偏序数据异常 (Predicate Write Skew):

[0319] 定义: 操作序列形如 $r1[P] \dots r2[P] \dots w1[y1 \text{ in } P] \dots w2[x1 \text{ in } P] \dots$ ($c1$ and $c2$ in either order) 的数据异常, 谓词写偏序数据异常属于事务级异常。在第二个事务提交时, 谓词写偏序数据异常发生; 消除异常只能通过回滚后提交的事务的方式进行。

[0320] 但谓词写偏序数据异常的操作序列定义不能是: $r1[P] \dots r2[P] \dots w1[y1 \text{ in } P] \dots w2[x1 \text{ in } P]$ 。为了提高异常识别效率, 在事务提交时候, 可以通过用户显式定义的约束条件来进行验证。

[0321] 该谓词写偏序数据异常还具备扩展能力, 其扩展形式为: $ri[P] \dots wi[v1 \text{ in } P]$, 可以扩展0到任意多个。该谓词写偏序数据异常具备类似写偏序异常到锯齿波式异常的扩展方式的扩展能力, 也即该谓词写偏序数据异常的扩展方式类似于锯齿波式数据异常的扩展, 锯齿波式数据异常将在下一项中进行详述。

[0322] 该谓词写偏序数据异常不能通过动边交叉识别, 需要通过完整性约束来进行识别, 比如 $r1[P]$ 不包含x对象, 而 $w2[x1 \text{ in } P]$ 包含了x对象, 形成不了同一个事务的垂边和斜边连接, 即没有动边, 因此强制定义为事务级异常。

[0323] 12) 锯齿波式数据异常 (Sawtooth Write Skew):

[0324] 定义: 操作序列形如 $r1[x0] \dots r2[y0] \dots r3[z0] \dots w1[y1] \dots w2[z1] \dots w3[x1]$ 的数据异常, 锯齿波式数据异常属于操作级异常。

[0325] 但锯齿波式数据异常的操作序列定义不能是: $r1[x0] \dots r2[y0] \dots r3[z0] \dots w1[y1] \dots w2[z1] \dots w3[x1] \dots$ ($c1$ and $c2$ and $c3$ in any order), 在写操作发生时即构成异常。锯齿波式数据异常, 包括3个及3个以上的变量。

[0326] 图8是本申请实施例提供的一种锯齿波式异常的原理图, 请参考图8, 示出了由事务操作序列构成三变量的锯齿波式数据异常。事务T1读取变量版本z0并写入变量版本x1, 记为“R(z0)W(x1)”, 事务T2读取变量版本y0写入变量版本z1, 记为“R(y0)W(z1)”, 事务T3读取变量版本x0写入变量版本y1, 记为“R(x0)W(y1)”, 事务T1~T3的一致性状态边如800所示, 在三个变量的锯齿波式数据异常的基础上, 还可以扩展到任意多个变量的锯齿波式数据异常。

[0327] 扩展方式如801所示, 方框802内的变量和事务关系可以在被复制任意次数后将复制得到的内容放入该方框802内, 达到对变量个数及事务个数的扩展, 方框802内的操作模式可以抽象为 $Ri(k0)Wi(a1)$, 该模式为一个锯齿, $k0$ 可为x0或任意其他变量, 但最终扩展的锯齿与x0通过读操作衔接, 事务T3要读取到a0。本质上, 图8中的变量y相关的部分也是一个锯齿, 但作为锯齿波区别于写偏序的有效组成部分 (至少包含3个变量), 因此在描述锯齿波式异常的扩展方式时, 不应被提及。

[0328] 需要说明的是, 上述操作模式 $Ri(k0)Wi(a1)$, 还可以变形为如下模式: $Ri(k0)W(a1)Ri(a1)$, 并且在扩展时, 锯齿波式数据异常可以扩展0到任意多个锯齿。

[0329] 显然, 写偏序数据异常和锯齿波式数据异常, 在本质上是同一类异常 (写异常), 但

从数学形式上描述,根据存在变量个数不同的差异,将涉及2个变量的写异常称为写偏序异常,将涉及3个及3个以上变量的写异常称为锯齿波式异常,也利于区分于传统的数据异常定义,在传统定义中仅描述了写偏序异常,只有两个变量,且不考虑该类异常的扩展性,而本申请实施例中考虑了其具备的无穷扩展能力。读偏序数据异常和台阶式数据异常之间的关系,与写偏序数据异常和锯齿波式数据异常之间的关系同理,这里不做赘述。

[0330] 在上述针对数据异常类型进行了详尽定义的基础上,本申请实施例中针对数据库系统定义了新的隔离级别,定义隔离级别遵循如下原则:按照并发度从高到低的过程,其数据的一致性从弱到强,允许发生的数据异常从最多变为没有,具体地,包括读写未提交、读写已提交、快照读历史、可重复读和可串行化五种隔离级别,这五种隔离级别与数据异常之间的关系如下表6所示。

[0331] 表6

异常 \ 级别					
	读写未提交	读写已提交	快照读历史	可重复读	可串行化
脏写	允许	不允许	不允许	不允许	不允许
脏读	允许	不允许	不允许	不允许	不允许
中间读	允许	不允许	不允许	不允许	不允许
不可重复读	允许	允许	允许	不允许	不允许
幻读	允许	允许	允许	不允许	不允许
丢失更新	允许	允许	不允许	不允许	不允许
读偏序	允许	允许	不允许	不允许	不允许
读写偏序	允许	允许	不允许	不允许	不允许
台阶式	允许	允许	不允许	不允许	不允许
写偏序	允许	允许	允许	不允许	不允许
谓词写偏序	允许	允许	允许	允许	不允许
锯齿波式	允许	允许	允许	不允许	不允许

[0333] 1、读写未提交 (Read/Write Uncommitted,RWU) 级别:对于没有提交的事务的中间状态,可读也可写,可读使得脏读可能发生,可写使得脏写可能发生。此RWU隔离级别和传统的“读未提交”级别的不同之处在于,RWU隔离级别没有对并发写操作进行屏蔽。也即,传统的读未提交不允许脏写异常发生,RWU隔离级别允许脏写异常发生。

[0334] 2、读写已提交 (Read/Write Committed,RWC) 级别:只有处于已提交状态的数据,才可读才可写,可读避免了脏读异常,即不能读事务执行的过程中的数据状态,可写避免了脏写异常,即不能写/覆盖事务执行的过程中的数据状态。此RWC隔离级别和RWU隔离级别的不同之处在于,RWC隔离级别禁止了并发的写操作。

[0335] 3、快照读历史 (Snapshot History, SH): 快照读历史基于历史上的任何一个时间点 (该时间点可以是系统最新的时刻, 但不能是未来的某一个时刻), 可读取历史上已经提交的数据, 即从数据状态矩阵中找到一条静边, 该静边表示了数据在历史的某个时刻的一致性状态, 所以能保持读数据的一致性。快照读历史和读写已提交的差异在于, 快照读历史禁止了丢失更新异常、读偏序异常、读写偏序异常和台阶式异常。快照读历史和Jim Grey给定的传统“快照”级别的差异在于, 本定义中的快照读历史级别允许发生幻读异常和不可重复读异常, 这是因为快照技术本质上就是静边技术, 静边总是代表着一个既成事实, 即“不可变更的历史”, 而幻读异常和不可重复读异常对应的操作序列中有两个历史的两次读操作发生, 他们是相似操作, 理应归为一类而不是被分离开。

[0336] 4、可重复读 (Repeatable Read, RR): 对于不带有谓词的读操作, 即事务内任何两次的读操作所读取到的数据对象总是一样的 (由于没有被其他事务修改, 所以系统的并发度降低)。RR隔离级别是建立在SH隔离级别基础之上的一个级别, 两者的差异在于, RR隔离级别禁止除了带有谓词语义相关的数据异常之外的其他异常发生, 也即相对于SH隔离级别, RR隔离级别还禁止了不可重复读异常、幻读异常、写偏序异常以及锯齿波式异常。

[0337] 5、可串行化 (serializable, S): 无一切数据异常发生。不管是实体数据对象上发生的异常, 还是在抽象语义下发生的异常, 都被禁止。S隔离级别是最高数据一致性级别。

[0338] 需要说明的是, Jim Grey等人在表2中定义的游标稳定 (Cursor Stability) 隔离级别, 不在本申请实施例提供的隔离级别之外, 因为该游标稳定级别是基于特定的实现不同隔离级别的封锁技术而定义的, 把特定的实现技术融合在隔离级别理论中不适宜。本申请实施例提供的隔离级别体系完全是基于已知异常和并发度的关系进行定义, 所允许的数据异常类型越多, 则系统的并发度越高, 但数据一致性的强度越低; 所允许的数据异常类型越少, 则系统的并发度越低, 但数据一致性的强度越高。所以隔离级别的数据一致性强度具有如下关系: $RWU < RWC < SH < RR < S$ 。

[0339] 在提供了上述针对数据异常的定义、动边交叉的定义以及隔离级别的定义的基础上, 图9是本申请实施例提供的一种事务处理方法流程图。参见图9, 该实施例应用于数据库系统中任一节点设备, 对于单机数据库系统而言, 该节点设备可以是单机设备, 对于分布式数据库系统而言, 该节点设备可以是协调节点设备或者数据节点设备, 该实施例包括下述步骤:

[0340] 901、节点设备响应于目标事务的读写操作执行, 获取该目标事务的至少一个并发事务, 该至少一个并发事务与该目标事务在事务执行期间涉及到对相同的数据项进行读写操作。

[0341] 其中, 该目标事务可以是全局事务, 也可以是局部事务, 其中, 该全局事务是指涉及到跨节点操作的事务, 全局事务也称为分布式事务, 而局部事务则是指仅涉及单个节点操作的事务, 局部事务也称为本地事务, 本申请实施例不对目标事务的类型进行具体限定。

[0342] 在上述过程中, 节点设备可以在开始执行目标事务时, 将该目标事务的读集和写集初始化为空集, 可选地, 目标事务可以由终端发起, 此时终端与节点设备建立用于处理目标事务的会话, 终端向节点设备发送目标事务的执行请求, 节点设备响应于目标事务的执行请求, 开始执行该目标事务。在一些实施例中, 如果终端与节点设备已经建立过一个会话, 那么无需在两者之间建立新的会话, 只需要复用当前已建立的会话即可。

[0343] 在读写集合的初始化过程中,节点设备可以在数据库系统启动时,向操作系统申请一块内存空间,该内存空间用于维护至少一个事务的读写集合(包括读集和写集),当目标事务开始执行时,节点设备从该内存空间中申请一块内存,该内存用于管理该目标事务的读写集合(包括读集和写集),从而在节点设备上完成了目标事务的读写集合的创建,并且,将创建出的读集和写集均初始化为空集。

[0344] 在对读写集合初始化完毕后,节点设备根据目标事务的执行语句,确定该目标事务是否涉及对变量进行更新操作,响应于该目标事务对任一变量进行更新,节点设备将该变量添加至该目标事务的写集中,为该变量分配版本号。例如,当某一事务T对变量x进行更新时,将变量x添加至事务T的写集中,并为变量x分配版本号。不同事务对同一个变量的写操作的版本号按整数递增。新发生的读操作也因此可获得变量的版本号。即数据库系统为每个变量维护一个递增的版本号。另外,节点设备响应于该目标事务对任一变量进行读取,所读取的数据是最新的且满足读已提交规则的数据。

[0345] 目标事务的读集和写集随着事务操作的进行而实时维护。如发生读操作,则被读取的数据进入读集;如发生写操作,则被读取的数据进入写集。同一个事务写同一个变量版本多次则有多个不同的新版本,在写集中用最新的版本替代旧的变量版本,上述初始化过程及维护策略可以称为读写阶段事务读写集合的形成算法。

[0346] 在上述读写集合的维护策略下,对于目标事务的每一个操作(该操作可以是读操作也可以是写操作),在该操作发生时,进入准备阶段,节点设备进入临界区,遍历目标事务的读集和写集,获取目标事务的至少一个并发事务(比如所有并发事务)的事务标识,并将该至少一个并发事务的事务标识存入到一个链表TS中。

[0347] 在获取并发事务的过程中,内存中的每个数据项上都可以记载曾经读写过本数据项但尚未提交的事务的事务标识(简称为活跃事务标识),每个数据项上记载的活跃事务标识可以是一个,也可以是多个,还可以为空。节点设备只需要获取读集和写集中各个数据项上所记载的各个活跃事务标识,即可得到该至少一个并发事务的事务标识。

[0348] 在上述过程中,相当于在目标事务的每个操作发生时,开始做准备工作,进入目标事务的准备阶段,准备阶段用于为事务T是否可以提交做好事务并发访问控制异常验证前的准备工作,节点设备在临界区中找到至少一个并发事务的事务标识之后,退出临界区。

[0349] 902、节点设备根据该至少一个并发事务与该目标事务的读写集合,分别获取该至少一个并发事务与该目标事务之间的异常检测结果,该异常检测结果用于表示该至少一个并发事务与该目标事务之间是否存在数据异常。

[0350] 在上述过程中,对该至少一个并发事务中任一并发事务,节点设备都可以根据该任一并发事务与该目标事务的读写集合,获取该任一并发事务与该目标事务之间的一个目标检测结果,响应于该任一并发事务的目标检测结果为存在数据异常,将最终的异常检测结果直接确定为存在数据异常,否则,对该至少一个并发事务中该任一并发事务的下一个并发事务执行获取目标检测结果的步骤,响应于该至少一个并发事务的目标检测结果均为不存在数据异常,将最终的异常检测结果确定为不存在数据异常。由于节点设备利用检测动边交叉的原理来进行数据的一致性检测,因此其一致性检测算法也可以称为动边交叉与合并算法。

[0351] 在目标事务的每个操作发生时,都可以执行获取至少一个并发事务的步骤,以及

对各个并发事务执行动边交叉与合并算法,假设当前进入验证阶段的待判断的目标事务为T,目标事务T最开始是一个物理事务,在通过动边交叉与合并算法的检验之后,如果最终的异常检测结果为不存在数据异常,那么将会变成一个合并后的逻辑事务。

[0352] 在验证阶段,提供两种不同的动边交叉与合并算法,第一种是基于各事务的动边交叉值DLI和变量状态值VS进行异常检测,第二种是基于各事务的布尔型upper进行异常检测。

[0353] 在第一种异常检测方式中,节点设备需要为DLI以及VS配置初始值,可以将两者的初始值都配置为0,也即令 $DLI=0$; $VS=0$ 。接着执行进入下述循环操作:从链表TS中取出第一个事务定为TL,对TL执行基于DLI和VS的异常检测方法,当存在动边交叉时,报告异常发生,将目标检测结果获取为存在数据异常,退出循环,将最终的异常检测结果获取为存在数据异常,否则,当不存在动边交叉时,将TL与目标事务T进行动边合并,得到新的事务T-new,并将T-new赋值给目标事务T,从链表TS中取出第二个事务定为TL,执行下一次循环(判断新的T-new与新的TL之间是否存在动边交叉),重复执行上述操作直到链表为空,结束循环,如果循环结束时仍没有异常报告,将最终的异常检测结果获取为不存在数据异常。上述基于DLI和VS的异常检测方法将在下个实施例中进行详述,这里不做展开。

[0354] 在第二种异常检测方式中,节点设备为每个事务各自分配一个布尔型,称之为upper,并在初始化过程中将各事务的upper值初始化为false。接着执行下述循环操作:从链表TS中取出第一个事务定为TL,对TL执行基于upper的异常检测方法,当存在动边交叉时,报告异常发生,将目标检测结果获取为存在数据异常,退出循环,将最终的异常检测结果获取为存在数据异常,否则,当不存在动边交叉时,将TL与目标事务T进行动边合并,得到新的事务T-new,并将T-new赋值给目标事务T,从链表TS中取出第二个事务定为TL,执行下一次循环(判断新的T-new与新的TL之间是否存在动边交叉),重复执行上述操作直到链表为空,结束循环,如果循环结束时仍没有异常报告,将最终的异常检测结果获取为不存在数据异常。上述基于upper的异常检测方法将在下个实施例中进行详述,这里不做展开。

[0355] 903、节点设备响应于该异常检测结果为不存在数据异常,将该至少一个并发事务与该目标事务的读写集合进行合并,提交该目标事务。

[0356] 在上述过程中,当目标事务是局部事务时,如果该异常检测结果为不存在数据异常,也即循环结束仍没有异常报告,则该目标事务可以直接提交,当目标事务是分布式事务时,此时该节点设备还需要向协调节点设备上报本节点设备的异常检测结果,由协调节点设备根据各个节点设备上报的异常检测结果,全局判断是否提交目标事务,如果所有节点设备上报的异常检测结果均为不存在数据异常,那么协调节点设备向各个节点设备下发相应的提交指令,节点设备在接收到提交指令后提交目标事务,对目标事务进行数据落盘,否则,只要存在任一节点设备上报的异常检测结果为存在数据异常,协调节点设备向各个节点设备下发相应的回滚指令,节点设备在接收到回滚指令后回滚目标事务。

[0357] 本申请实施例提供的方法,通过响应于目标事务的读写操作执行,获取该目标事务的至少一个并发事务,根据该至少一个并发事务与该目标事务的读写集合,获取该至少一个并发事务与该目标事务之间的异常检测结果,响应于该异常检测结果为不存在数据异常,将该至少一个并发事务与该目标事务的读写集合进行合并,提交该目标事务,这种方式能够在目标事务的每个操作执行时全面检测出数据库系统内各种各样的数据异常,保证数

据状态的一致性,在保证数据状态一致性的基础上,这种事务处理机制既不完全依赖于封锁技术也不完全依赖于依赖图技术,避免了限制数据库系统的并发度,能够提升数据库系统的事务处理效率。

[0358] 上述所有可选技术方案,可以采用任意结合形成本公开的可选实施例,在此不再一一赘述。

[0359] 下面,将详细介绍面向数据状态一致性的动边交叉与合并算法(Dynamic Line Intersection-Merge,DLI-M),DLI-M算法基于数据状态一致性模型进行数据异常识别,从而判断并发事务是否符合数据一致性,DLI-M算法没有消除出带有谓词的幻读、谓词写偏序数据异常的解决方式,也即仅实现了可串行化(S)隔离级别。

[0360] DLI-M算法中包括读写阶段的事务读写集形成算法、准备阶段算法以及一致性检验算法(动边交叉与合并算法),在一致性检验算法中,分别提供了基于DLI和VS的异常检测方法以及基于upper的异常检测方法,两种异常检测方法均可以达成针对数据异常的识别与上报,下面将分别进行说明。

[0361] 1、读写阶段事务读写集形成算法

[0362] 在目标事务开始执行时,初始化目标事务的读集和写集为空集。

[0363] 当目标事务对变量x进行更新时,将变量x添加至目标事务的写集中,为变量x分配版本号。不同事务对同一个变量的写操作的版本号按整数递增。新发生的读操作也因此可获得变量的版本号,即为每个变量维护一个递增的版本号。

[0364] 目标事务的读集和写集随着事务操作的进行而实时维护。如发生读操作,则被读取的数据进入读集;如发生写操作,被读取的数据进入写集。同一个事务写同一个变量版本多次则有多个不同新的版本,在写集中用最新的版本替代旧的变量版本。

[0365] 此外,在读取过程中所读取的数据,是最新的满足读已提交规则的数据。

[0366] 上述事务读写集形成算法与上述步骤901所介绍的读写阶段的步骤类似,这里不做赘述。

[0367] 2、每个操作发生时,执行准备阶段算法

[0368] 在目标事务T的每个操作(或读或写)发生时,进入准备阶段并开始做准备工作,准备阶段用于为目标事务T是否可提交做好事务并发访问控制异常验证前的准备工作。

[0369] 在准备阶段中,节点设备进入临界区,遍历目标事务T的读集和写集,找出目标事务T的所有并发事务的事务标识,放入一个链表TS内进行存储,在找到后退出临界区。在内存中,每个数据项上存有曾经读写过本数据项但尚未提交的事务的事务标识。

[0370] 上述准备阶段算法与上述步骤901所介绍的准备阶段的步骤类似,这里不做赘述。

[0371] 3、每个操作发生时,执行一致性检验算法(也即动边交叉与合并算法)

[0372] 对于目标事务的任一并发事务,节点设备可以根据该任一并发事务与该目标事务的读写集合,获取该任一并发事务与该目标事务之间的目标检测结果;响应于该目标检测结果为存在数据异常,将该异常检测结果确定为存在数据异常;否则,对该至少一个并发事务中该任一并发事务的下一个并发事务执行获取目标检测结果的步骤;响应于该至少一个并发事务的目标检测结果均为不存在数据异常,将该异常检测结果确定为不存在数据异常。

[0373] 换言之,节点设备从链表TS中取出第一个事务定为TL,对TL执行动边交叉与合并

算法,当存在动边交叉时,报告异常发生,将目标检测结果获取为存在数据异常,退出循环,将最终的异常检测结果获取为存在数据异常,否则,当不存在动边交叉时,将TL与目标事务T进行动边合并,得到新的事务T-new,并将T-new赋值给目标事务T,从链表TS中取出第二个事务定为TL,执行下一次循环(判断新的T-new与新的TL之间是否存在动边交叉),重复执行上述操作直到链表为空,结束循环,如果循环结束时仍没有异常报告,将最终的异常检测结果获取为不存在数据异常。

[0374] 在本申请实施例中,动边交叉与合并算法包括两种不同的类型,第一种是基于DLI和VS的异常检测方法,第二种是基于upper的异常检测方法,下面将分类讨论。

[0375] (一) 基于DLI和VS的异常检测方法

[0376] 图10是本申请实施例提供的一种获取目标检测结果的流程图,请参考图10,示出了节点设备如何基于一致性检验算法获取目标检测结果,这里针对某一次循环过程进行展开说明。

[0377] 1001、对于至少一个并发事务中任一并发事务,节点设备获取第一集合S1、第二集合S2以及第三集合S3,其中,S1为目标事务的读集与该任一并发事务的写集之间的交集,S2为目标事务的写集与该任一并发事务的读集之间的交集,S3为目标事务的写集与该任一并发事务的写集之间的交集。

[0378] 在上述过程中,节点设备在进入验证阶段的时候,假设当前进入验证阶段的待判断的目标事务为T,目标事务T最开始初始值是一个物理事务,在经过动边合并后变成一个逻辑事务。进一步地,节点设备为动边交叉值DLI以及变量状态值VS配置初始值,可以将两者的初始值都配置为0,也即令 $DLI=0$; $VS=0$ 。

[0379] 在对DLI以及VS初始化完毕后,节点设备对链表TS中每个并发事务循环执行获取目标检测结果的操作,一旦任一目标检测结果为存在数据异常,退出循环并将最终的异常检测结果确定为存在数据异常,否则,继续对链表TS中下一并发事务执行循环,直到链表TS为空,所有的目标检测结果均为不存在数据异常,结束循环并将最终的异常检测结果确定为不存在数据异常。

[0380] 上述步骤1001即为对任一并发事务TL执行的循环操作的第一个步骤,此时节点设备根据目标事务T和该任一并发事务TL的读写集合,将目标事务T的读集与该任一并发事务TL的写集之间的交集确定为第一集合S1,第一集合S1可以表示为“ $S1=DS_R(T) \cap DS_W(TL)$ ”,将目标事务T的写集与该任一并发事务TL的读集之间的交集确定为第二集合S2,第二集合S2可以表示为“ $S2=DS_R(TL) \cap DS_W(T)$ ”,将目标事务T的写集与该任一并发事务TL的写集之间的交集确定为第三集合S3,第三集合S3可以表示为“ $S3=DS_W(T) \cap DS_W(TL)$ ”。

[0381] 可选地,在获取S1、S2、S3时,可以使用哈希表存储TL和T各自的读写集合,从而能够在线性时间复杂度内得到他们的交集和并集。

[0382] 需要说明的是,如果S1、S2、S3三者之间的并集为空集,也即 $S1 \cup S2 \cup S3$ 为空集(简化计算: $S1=\emptyset$ 且 $S2=\emptyset$ 且 $S3=\emptyset$,也即S1、S2、S3均为空集),则说明TL和目标事务T不是并发事务,节点设备需要把TL重新加入链表TS的尾部。否则,如果S1、S2、S3三者之间的并集不是空集,说明TL和目标事务T是并发事务,此时执行下述步骤1002-1003进行异常检测,如果发现异常,回滚目标事务T。

[0383] 1002、节点设备响应于该第三集合S3不是空集,根据该任一并发事务的提交情况

获取该任一并发事务与该目标事务之间的目标检测结果。

[0384] 在一些实施例中,若该任一并发事务TL未提交且该目标事务T的目标参数为一,将该目标检测结果获取为存在数据异常且数据异常类型为脏写异常,其中,该目标参数T.no_committed用于表示该目标事务的读写集合成分所对应的已提交事务数量,由于在循环过程中,如果上一个并发事务与目标事务之间的目标检测结果为不存在数据异常,那么在执行本次循环之前,需要将上一个并发事务与目标事务进行动边合并,并将动边合并后得到的新的事务赋值给目标事务,而动边合并的本质是对两事务的读写集合进行合并,因此在任一次循环中,目标事务的读写集合有可能是经过合并后的读写集合,在目标事务的读写集合中会拥有某个已经提交的事务的读写集合的成分,而目标参数T.no_committed则用来描述读写集合中所拥有的其他已提交事务的成分所对应的已提交事务数量。

[0385] 在一些实施例中,若该任一并发事务TL已提交且该第一集合S1与该第三集合S3之间的交集不是空集,将该目标检测结果获取为存在数据异常且数据异常类型为丢失更新异常。

[0386] 也即是说,如果 $S3 \neq \emptyset$,满足如下任一条件时,节点设备将目标检测结果获取为存在数据异常,并将最终的异常检测结果获取为存在数据异常,报告写写异常发生,循环终止:1) 该任一并发事务TL没有提交完成且该目标事务T的目标参数T.no_committed=1,此时构成写写冲突,数据异常类型为脏写异常;2) 该任一并发事务TL已经提交且 $S1 \cap S3$ 不是空集,此时数据异常类型为丢失更新异常。当 $S3 = \emptyset$ 时,执行下述步骤1003。

[0387] 1003、节点设备响应于该第三集合S3是空集且该第一集合S1或者该第二集合S2中至少一项不是空集,根据该第一集合S1和该第二集合S2获取该目标检测结果。

[0388] 在一些实施例中,节点设备可以根据该第一集合和该第二集合,更新该任一并发事务与该目标事务之间的动边交叉值DLI和变量状态值VS,基于更新后的动边交叉值DLI和更新后的变量状态值VS,获取该目标检测结果。

[0389] 在 $S3 = \emptyset$ 的基础上,如果满足 $S1 \neq \emptyset$ 或者 $S2 \neq \emptyset$ 中至少一项,节点设备根据如下更新规则对DLI和VS进行更新:

[0390] 1) 节点设备响应于该第一集合S1不是空集,将动边交叉值DLI更新为已有值加一所得的数值,也即: $S1$ 不为空,则 $DLI = DLI++$;响应于该第一集合S1中存在数据状态不同的变量,将变量状态值VS更新为已有值加一所得的数值,也即:如S1中存在某个非空元素的数据状态不同,则 $VS = VS++$ 。

[0391] 2) 节点设备响应于该第二集合S2不是空集,将动边交叉值DLI更新为已有值加一所得的数值,也即: $S2$ 不为空,则 $DLI = DLI++$;响应于该第二集合S2中存在数据状态不同的变量,将变量状态值VS更新为已有值加一所得的数值,也即:如S2中存在某个非空元素的数据状态不同,则 $VS = VS++$ 。

[0392] 在一些实施例中,在对DLI和VS进行更新后,节点设备可以基于下式方式来获取该目标检测结果:节点设备响应于该更新后的动边交叉值DLI大于或等于二,将该目标检测结果获取为存在数据异常,并基于该更新后的变量状态值VS,确定数据异常类型;否则,将该目标检测结果获取为不存在数据异常。

[0393] 在一些实施例中,节点设备在基于更新后的VS确定数据异常类型时,可以响应于

该更新后的变量状态值为一,确定该数据异常类型为读异常,也即是 $VS=1$,报告读异常,读异常包括1个变量的不可重复读异常、2个变量的读偏序异常、3个及3个以上变量的台阶式异常;响应于该更新后的变量状态值 VS 大于或等于二,确定该数据异常类型为写异常,也即是 $VS \geq 2$,报告写异常,写异常包括2个变量的写偏序异常、3个及3个以上变量的锯齿波式异常。

[0394] 在上述过程中,如果 $DLI \geq 2$,说明目标事务 T 和 TL 构成动边交叉,则报告异常发生,将目标检测结果以及异常检测结果确定为存在数据异常,循环终止($VS=1$ 时数据异常类型为读异常, $VS \geq 2$ 时数据异常类型为写异常);否则,没有任何异常,将 TL 和目标事务 T 进行动边合并,设定得到的新的事务为 $T\text{-new}$,并将 $T\text{-new}$ 赋值给目标事务 T ($T=T\text{-new}$),从链表 TS 中取出第二个事务定为 TL ,执行下一次循环(判断新的 $T\text{-new}$ 与新的 TL 之间是否存在动边交叉)。

[0395] 在上述动边合并的过程中,节点设备可以将该任一并发事务 TL 的读集并入目标事务 T 的读集中,将该任一并发事务 TL 的写集并入目标事务 T 的写集中,此外,如果事务 TL 没有提交,则令 $T\text{-new}$ 的目标参数自增1,也即令 $T\text{-new.no_committed++}$ (自增1),表示合并得到的新事务中拥有某个已经提交的事务的读写集合的成分。

[0396] 进一步地,如果循环结束时,仍没有异常报告,则代表目标事务可提交。可选地,如果隔离级别参数= S ,也即隔离级别是可串行化级别,此时即可满足可串行化级别,否则,系统可以在允许某些类型的数据异常发生的情况下,满足所设定的隔离级别。

[0397] (二)基于upper的异常检测方法

[0398] 图11是本申请实施例提供的一种获取目标检测结果的流程图,请参考图11,示出了节点设备如何基于一致性检验算法获取目标检测结果,这里针对某一次循环过程进行展开说明。

[0399] 1101、对于至少一个并发事务中任一并发事务,节点设备获取第一集合 $S1$ 、第二集合 $S2$ 、第三集合 $S3$ 以及第四集合 $S4$,其中, $S1$ 为目标事务的读集与该任一并发事务的写集之间的交集, $S2$ 为目标事务的写集与该任一并发事务的读集之间的交集, $S3$ 为目标事务的写集与该任一并发事务的写集之间的交集, $S4$ 为目标事务的读集与该任一并发事务的读集之间的交集。

[0400] 在上述过程中,节点设备在进入验证阶段的时候,假设当前进入验证阶段的待判断的目标事务为 T ,目标事务 T 最开始初始值是一个物理事务,在经过动边合并后变成一个逻辑事务。

[0401] 进一步地,节点设备为目标事务 T 以及该任一并发事务 TL 的布尔型upper配置初始值,可以将两者的初始值都配置为false,也即令 $T.upper=false$; $TL.upper=false$ 。该布尔型upper表示的是事务所构成的动边在数据状态矩阵中的垂直位置关系,以任意两个并发事务 Ti 和 Tj 为例,若 $Ti.upper=true$,则表明事务 Ti 在某变量上构成的动边相对另一事务 Tj 较新,也即是说,针对同一变量而言,事务 Ti 在该变量上操作了比事务 Tj 较新的变量版本。

[0402] 在对两事务的upper值初始化完毕后,节点设备对链表 TS 中每个并发事务循环执行获取目标检测结果的操作,一旦任一目标检测结果为存在数据异常,退出循环并将最终的异常检测结果确定为存在数据异常,否则,继续对链表 TS 中下一并发事务执行循环,直到

链表TS为空,所有的目标检测结果均为不存在数据异常,结束循环并将最终的异常检测结果确定为不存在数据异常。

[0403] 上述步骤1101即为对任一并发事务TL执行的循环操作的第一个步骤,此时节点设备根据目标事务T和该任一并发事务TL的读写集合,将目标事务T的读集与该任一并发事务TL的写集之间的交集确定为第一集合S1,第一集合S1可以表示为“ $S1 = DS_R(T) \cap DS_W(TL)$ ”,将目标事务T的写集与该任一并发事务TL的读集之间的交集确定为第二集合S2,第二集合S2可以表示为“ $S2 = DS_R(TL) \cap DS_W(T)$ ”,将目标事务T的写集与该任一并发事务TL的写集之间的交集确定为第三集合S3,第三集合S3可以表示为“ $S3 = DS_W(T) \cap DS_W(TL)$ ”,将目标事务T的读集与该任一并发事务TL的读集之间的交集确定为第四集合S4,第四集合S4可以表示为“ $S4 = DS_R(TL) \cap DS_R(T)$ ”。

[0404] 可选地,在获取S1、S2、S3、S4时,可以使用哈希表存储TL和T各自的读写集合,从而能够在线性时间复杂度内得到他们的交集和并集。

[0405] 需要说明的是,如果S1、S2、S3、S4三者之间的并集为空集,也即 $S1 \cup S2 \cup S3 \cup S4$ 为空集(简化计算: $S1 = \emptyset$ 且 $S2 = \emptyset$ 且 $S3 = \emptyset$ 且 $S4 = \emptyset$,也即S1、S2、S3、S4均为空集),则说明TL和目标事务T不是并发事务,节点设备需要把TL重新加入链表TS的尾部。否则,如果S1、S2、S3、S4三者之间的并集不是空集,说明TL和目标事务T是并发事务,此时执行下述步骤1102-1103进行异常检测,如果发现异常,回滚目标事务T。

[0406] 1102、节点设备响应于S3不是空集,根据该任一并发事务的提交情况获取该任一并发事务与该目标事务之间的目标检测结果。

[0407] 上述步骤1102与上述步骤1002类似,这里不做赘述。

[0408] 1103、节点设备响应于S3是空集且S1、S2或者S4中至少一项不是空集,根据S1、S2以及S4获取该目标检测结果。

[0409] 在一些实施例中,节点设备可以根据S1、S2和S4,更新该任一并发事务TL的布尔型TL.upper与该目标事务T的布尔型T.upper,基于更新后的TL.upper和更新后的T.upper,获取该目标检测结果。其中,该布尔型用于表示对应事务所构成的动边在数据状态矩阵中的垂直位置关系且该布尔型的初始值为假(false)。

[0410] 具体地,在基于更新后的TL.upper和更新后的T.upper,获取该目标检测结果时,可以执行如下处理:节点设备响应于更新后的该目标事务的布尔型和更新后的该任一并发事务的布尔型均为真 $T.upper = TL.upper = true$,将该目标检测结果获取为存在数据异常;否则,将该目标检测结果获取为不存在数据异常。

[0411] 在 $S3 = \emptyset$ 的基础上,如果满足 $S1 \neq \emptyset$ 、 $S2 \neq \emptyset$ 或者 $S4 \neq \emptyset$ 中至少一项,节点设备根据如下更新规则对TL.upper和T.upper进行更新:

[0412] 1) 节点设备响应于S1不是空集,对于S1中任一变量,若该变量在目标事务的读集中的版本号大于或等于该变量在该任一并发事务的写集中的版本号,将该目标事务的布尔型更新为真;否则,将该任一并发事务的布尔型更新为真。

[0413] 也即是说,对于S1中的每一个变量x,若该目标事务T读变量x的版本 $\geq$ 该任一并发事务TL写变量x的版本,比如该目标事务T读取变量版本x2、该任一并发事务TL写入变量版本x1,则置 $T.upper = true$,否则,若该目标事务T读变量x的版本 $<$ 该任一并发事务TL写变量x的版本,比如该目标事务T读取变量版本x1、该任一并发事务TL写入变量版本x2,则置

TL.upper=true。

[0414] 2) 节点设备响应于S2不是空集,对于S2中任一变量,若该变量在目标事务的写集中的版本号大于该变量在该任一并发事务的读集中的版本号,将该目标事务的布尔型更新为真;否则,将该任一并发事务的布尔型更新为真。

[0415] 也即是说,对于S2中的每一个变量x,若该目标事务T写变量x的版本>该任一并发事务TL读变量x的版本,比如该目标事务T写入变量版本x2、该任一并发事务TL读取变量版本x1,则置T.upper=true,否则,若该目标事务T写变量x的版本≤该任一并发事务TL读变量x的版本,比如该目标事务T写入变量版本x1、该任一并发事务TL读取变量版本x2,则置TL.upper=true。

[0416] 3) 节点设备响应于S4不是空集,对于S4中任一变量,若该变量在目标事务的读集中的版本号大于该变量在该任一并发事务的读集中的版本号,将该目标事务的布尔型更新为真;若该变量在目标事务的读集中的版本号小于该变量在该任一并发事务的读集中的版本号,将该任一并发事务的布尔型更新为真。

[0417] 也即是说,对于S4中的每一个变量x,若该目标事务T读变量x的版本>该任一并发事务TL读变量x的版本,比如该目标事务T读取变量版本x2、该任一并发事务TL读取变量版本x1,则置T.upper=true;若该目标事务T读变量x的版本<该任一并发事务TL读变量x的版本,比如该目标事务T读取变量版本x1、该任一并发事务TL读取变量版本x2,则置TL.upper=true;若该目标事务T读变量x的版本=该任一并发事务TL读变量x的版本,比如该目标事务T和该任一并发事务TL均读取了变量版本x1,则不做任何操作。

[0418] 在S3为空集的情况下,可以保证满足基本规则中禁止写写冲突发生的条件,也即禁止脏写数据异常。在以上述规则1)-3)中,分别对该目标事务T的读集和该任一并发事务TL的写集进行比较(分析S1中的元素)、对该目标事务T的写集和该任一并发事务TL的读集进行比较(分析S2中的元素)、对该目标事务T的读集和该任一并发事务TL的读集进行比较(分析S4中的元素)之后,分别对该目标事务T的布尔型T.upper和该任一并发事务TL的布尔型TL.upper进行更新,若在更新完毕后,满足如下条件:T.upper=TL.upper=true,则认为存在动边交叉,也即存在数据异常,此时将该目标检测结果获取为存在数据异常;否则,将该目标检测结果获取为不存在数据异常。

[0419] 在上述过程中,如果最终T.upper=TL.upper=true,说明该目标事务T和该任一并发事务TL构成动边交叉,则报告异常发生,将目标检测结果以及异常检测结果确定为存在数据异常,循环终止;否则,没有任何异常,将该任一并发事务TL和该目标事务T进行动边合并,设定得到的新的事务为T-new,并将T-new赋值给目标事务T(T=T-new),从链表TS中取出第二个事务定为TL,执行下一次循环(判断新的T-new与新的TL之间是否存在动边交叉)。

[0420] 在上述动边合并的过程中,节点设备可以将该任一并发事务TL的读集并入该目标事务T的读集中,将该任一并发事务TL的写集并入该目标事务T的写集中,此外,如果该任一并发事务TL没有提交,则令T-new的目标参数自增1,也即令T-new.no_committed++(自增1),表示合并得到的新事务中拥有某个已经提交的事务的读写集合的成分。

[0421] 进一步地,如果循环结束时,仍没有异常报告,则代表该目标事务可提交。可选地,如果隔离级别参数=S,也即隔离级别是可串行化级别,此时即可满足可串行化级别,否则,

系统可以在允许某些类型的数据异常发生的情况下,满足所设定的隔离级别。

[0422] 本申请实施例提供的动边交叉与合并算法(DLI-M),使用到多次集合的运算,对于这些运算,可以使用哈希表作为集合的数据结构,从而可在线性时间复杂度内得到他们的交集和并集,时间复杂度为 $O(m+n)$ 。DLI-M算法再每次循环中,通过5次的集合并交计算,即能识别并发事务之间是否存在数据异常。而循环次数最多达到并发事务的个数 k ,因此总的算法复杂度应为: $O(k*(m+n))$ 。最坏的情况是,每一个事务只有1到2个并发事务,且循环执行了 $k-1$ 次,才找到可合并的并发事务,这导致复杂度变为 $O(k^2*(m+n))$ 。但实际情况通常是,随着事务动边合并的推进,余下的事务个数在减少,因此复杂度为 $O(k*\lg k*(m+n))$ 。

[0423] 上述所有可选技术方案,可以采用任意结合形成本公开的可选实施例,在此不再一一赘述。

[0424] 在本申请实施例中,在上述DLI-M算法的基础上,采用快照技术、融合静边和动边概念,可以实现数据库系统的多种隔离级别,这一算法称为基于快照的数据状态一致性动边交叉与合并算法(Snapshot-based DLI-M,SDLI-M),该SDLI-M算法可以考虑耦合或者解耦乐观并发控制算法(Optimistic Concurrency Control,OCC),此外,在本实施例中的SDLI-M算法还可以针对上述实施例中DLI-M算法的一些细节进行优化,从而达到对DLI-M算法的改进,下面进行详述。

[0425] 1、读写阶段事务读写集成算法

[0426] 在目标事务开始执行时,初始化目标事务的读集和写集为空集。

[0427] 当目标事务对变量 x 进行更新时,将变量 x 添加至目标事务的写集中,为变量 x 分配版本号。不同事务对同一个变量的写操作的版本号按整数递增。新发生的读操作也因此可获得变量的版本号,即为每个变量维护一个递增的版本号。

[0428] 目标事务的读集和写集随着事务操作的进行而实时维护。如发生读操作,则被读取的数据进入读集;如发生写操作,被读取的数据进入写集。同一个事务写同一个变量版本多次则有多个不同新的版本,在写集中用最新的版本替代旧的变量版本。

[0429] 上述初始化过程以及读写集合的维护策略与DLI-M算法类似,这里不做赘述。

[0430] SDLI-M算法与DLI-M算法的区别在于,SDLI-M算法在读取过程中所读取的数据,并不是DLI-M算法中“最新的满足读已提交规则的数据”,而是距离快照 St 时间点最近且最合适的静边上的相关数据,可以满足数据的一致性状态。

[0431] 具体地,在数据读取过程中遵循数据项的可见性判断算法:基于时间戳,建立一个快照Snapshot Set= $S_s = \{St\}$, St 是一个时间点。在每个数据项的基本数据结构中,存储有一个产生该数据项的事务的事务号 tid 和一个产生该数据项的事务的提交时间戳 tc (若该数据版本未提交则 $tc=NULL$)。如果一个数据项的 tc 非NULL且 $tc < St$,则该版本可被快照读取,也即该数据项为相对于目标事务可见的目标数据项,但是所有的版本之间存在一个从最新版本到最老版本之间的一个版本链(也即历史序列),读取版本的时候从最新版本开始。

[0432] 需要说明的是快照时间点 St 、事务标识 tid 、提交时间戳 tc 三者都属于同一个数据类型,如都是同一个数值型逻辑时间戳值,或都是类似HLC(混合逻辑时钟)的时间戳值,只要可以进行偏序比较大小即可。这一可见性判断算法保证了一个事务读取的数据是距离 St 时间点最近且最合适的静边上的相关数据,即保证读取的数据满足数据的一致性状态。

[0433] 2、数据读取阶段

[0434] 在一些实施例中,在SDLI-M算法的数据读取阶段,节点设备对于目标事务的任一次读操作,可以确定该读操作的快照;根据该读操作的读取条件以及该读操作的快照,确定相对于该目标事务可见的目标数据项,将该目标数据项添加至该目标事务的读集中。

[0435] 可选地,在确定该读操作的快照时:响应于数据库系统的隔离级别低于可重复读RR,节点设备可以在执行该读操作时获取一次快照;否则,节点设备可以将该目标事务第一次读操作的快照确定为该读操作的快照。也即是说,在读取数据的时候,如果隔离级别参数 $<RR$ (隔离级别低于RR),每执行一次读操作,都将获取一次快照,这种情况下每次读都具有新快照则允许幻读异常发生;否则,如果隔离级别参数 $\geq RR$ (隔离级别高于或等于RR),仅在第一次执行读操作时获取快照,后续每次读操作均使用第一次获取的快照来读取数据,这种情况下只使用第一次的快照,可避免幻读异常和不可重复读异常发生,1个变量的不可重复读异常在此数据读取阶段内已经被识别,因此不用进入到动边交叉与合并算法中进行识别。需要说明的是,此时是避免发生某一类数据异常,而不是在检测发现异常然后回滚事务,所以没有报告异常并终结事务的步骤。

[0436] 在一些实施例中,在可见性判断算法中,节点设备响应于数据库系统的隔离级别高于或等于读写已提交RWC,将符合该读取条件且提交时间戳 t_c 小于该读操作的快照 S_t 的最大版本确定为目标数据项;响应于数据库系统的隔离级别低于读写已提交RWC,若符合该读取条件的数据项中存在未提交版本,将最大的未提交版本确定为目标数据项。

[0437] 也即是说,当隔离级别参数 $\geq RWC$ 时,与上述“读写阶段事务读写集形成算法”中所涉及的可见性判断算法保持一致,这里不做赘述,但当隔离级别参数 $<RWC$ 时,可以对前述可见性判断算法进行修改,使得最新的一个未提交版本可以被读取到,从而实现了RWU级别。

[0438] 在一些实施例中,如果采用OCC框架,在隔离级别参数 $>RR$ 时,则对于每个事务中的第二个读操作,先从读集和写集中验证是否存在要读取的对象,如果存在,再从读写集合中获取;否则,需要从底层的存储层内获取。

[0439] 3、每个操作发生时,执行准备阶段算法

[0440] 在目标事务T的每个操作(或读或写)发生时,进入准备阶段并开始做准备工作,准备阶段用于为目标事务T是否可提交做好事务并发访问控制异常验证前的准备工作。

[0441] 在准备阶段中,节点设备进入临界区,遍历目标事务T的读集和写集,找出目标事务T的所有并发事务的事务标识,放入一个链表TS内进行存储,在找到后退出临界区。在内存中,每个数据项上存有曾经读写过本数据项但尚未提交的事务的事务标识。

[0442] 上述准备阶段算法与上述DLI-M算法所介绍的准备阶段算法类似,这里不做赘述。

[0443] 在一些实施例中,SDLI-M算法与OCC算法耦合时,准备阶段的算法对应于OCC的验证阶段。

[0444] 在一些实施例中,SDLI-M算法可以针对DLI-M算法进行如下改进:如果已有链表oldTS不为空,则把目标事务直接加入到TS中(也即令 $TS = oldTS + T$),不再遍历目标事务的读集和写集,从而可以减少对读写集合的遍历逻辑,从而可提高事务处理效率。

[0445] 4、每个操作发生时,执行一致性检验算法(也即改进的动边交叉与合并算法)

[0446] 在一些实施例中,SDLI-M算法与OCC算法耦合时,一致性检验算法对应于OCC的验证阶段,不再是每个操作发生时执行一次,而是在OCC的验证阶段执行。

[0447] 在SDLI-M算法中改进的动边交叉与合并算法与DLI-M算法中的动边交叉与合并算法相类似,这里对类似的步骤不做详细展开,改进的动边交叉与合并算法仍然包括两种不同的类型,第一种是改进的基于DLI和VS的异常检测方法,第二种是改进的基于upper的异常检测方法,下面将分类讨论。

[0448] (一)改进的基于DLI和VS的异常检测方法

[0449] (4.1)当前进入验证阶段的待判断的目标事务为T,此事务T的初始值是一个物理事务,在经过动边合并之后变成一个逻辑事务。

[0450] (4.2)对DLI和VS进行初始化。也即是设定一些隶属变量的初始值:DLI=0;VS=0。

[0451] (4.3)从链表TS中取出第一个事务定为TL,循环执行下述操作,直到链表为空:

[0452] (4.3.1)获取第一集合 $S1 = DS_R(T) \cap DS_W(TL)$,S1为目标事务T的读集和并发事务TL的写集之间的交集。S1不为空,则DLI=DLI++;如S1中某个非空元素的数据状态不同,则VS=VS++。

[0453] (4.3.2)获取第二集合 $S2 = DS_R(TL) \cap DS_W(T)$,S2为目标事务T的写集和并发事务TL的读集之间的交集。S1不为空,则DLI=DLI++;如S2中某个非空元素的数据状态不同,则VS=VS++。

[0454] (4.3.3)获取第三集合 $S3 = DS_W(T) \cap DS_W(TL)$,S3为目标事务T的写集和并发事务TL的写集之间的交集。

[0455] (4.3.4)如果 $S1 \cup S2 \cup S3$ 为空集(简化计算为: $S1 = \emptyset$ 且 $S2 = \emptyset$ 且 $S3 = \emptyset$),则说明TL和T不是并发事务,把TL重新加入到链表TS的尾部。

[0456] (4.3.5)否则, $S1 \cup S2 \cup S3$ 不为空集,说明S1、S2、S3中至少存在一个集合不是空集,此时TL和T是并发事务,进行异常检测,一旦发现异常,需要报告异常并回滚目标事务T。异常检测过程遵循下述规则:

[0457] (4.3.5.1)如果 $S3 \neq \emptyset$,在满足如下两种条件中任一项时,节点设备则报告写写异常发生,循环终止。

[0458] 条件一、如果隔离级别参数 $\neq RWU$ (只有RWU级别不禁止脏写异常),TL没有提交完成且T.no_committed=1,存在写写冲突,数据异常类型为脏写异常。

[0459] 条件二、如果隔离级别参数 $\geq SH$ (SH级别禁止丢失更新发生,所以高于或等于SH的隔离级别均报告此异常),TL已经提交且 $S1 \cap S3$ 不是空集,上报数据异常,数据异常类型为丢失更新异常。

[0460] (4.3.5.2)否则,在 $S3 = \emptyset$ 的条件下,如果 $S1 \neq \emptyset$ 或者 $S2 \neq \emptyset$,可以分为如下三种情况进行讨论:

[0461] 情况一、如果目标事务T和TL构成动边交叉,则报告异常发生,循环终止。在一些实施例中,DLI ≥ 2 时,认为目标事务T与TL之间构成动边交叉,此时节点设备可以根据隔离级别确定是否报告异常发生并使得循环终止,并根据VS来确定数据异常类型。

[0462] 可选地,如果隔离级别参数 $\geq SH$ (禁止读偏序异常和台阶式异常),且VS=1,则报告异常发生并使得循环终止,此时数据异常类型为读异常,包括2个变量的读偏序异常、3个及3个以上变量的台阶式异常(1个变量的不可重复读异常已经在数据读取阶段中被识别了,在异常检测阶段无需进行重复识别)。

[0463] 可选地,如果隔离级别参数 $\geq RR$ (禁止写偏序异常和锯齿波式异常),且 $VS \geq 2$,则报告异常发生并使得循环终止,此时数据异常类型为写异常,包括2个变量的写偏序异常、3个及3个以上变量的锯齿波式异常。

[0464] 情况二、否则,如果目标事务T和TL没有构成动边交叉,在隔离级别参数=S的情况下,节点设备还需要检查是否存在完整性约束,如果本事务所读写的数据违反了完整性约束,则报告异常发生并使得循环终止,通过完整性约束可以检测出来DLI-M算法中所无法解决的谓词写偏序异常。

[0465] 也即是说,在基于DLI和VS检测出不存在动边交叉之后,本来该并发事务TL与目标事务T之间的目标检测结果被置为“不存在数据异常”,但在可串行化S级别下还需要识别谓词写偏序异常(动边交叉与合并算法无法识别),因此节点设备响应于数据库系统的隔离级别为可串行化S,对该并发事务TL与该目标事务T进行完整性约束检测;响应于该并发事务TL与该目标事务T之间违反完整性约束,将该目标检测结果调整为存在数据异常且数据异常类型为谓词写偏序异常。其中,完整性约束的检测过程将在下一个实施例中进行详述。

[0466] 情况三、否则,如果目标事务T和TL没有构成动边交叉,且符合完整性约束,那么没有任何异常,对目标事务T和TL进行动边合并,设定新的事务为T-new。

[0467] 在动边合并过程中,节点设备把并发事务TL的读集并入目标事务T的读集,将并发事务TL的写集并入目标事务T的写集;如果并发事务TL没有提交,则令T-new的目标参数T-new.no_committed++(自增1),表示合并得到的新事务中拥有某个已经提交的事务的读写集的成分;令 $T=T\text{-new}$,用T-new给T赋值,然后执行下一次循环。

[0468] (4.3.6) 如果循环结束,没有异常报告,则目标事务T可以提交。如果隔离级别参数=S则保证了可串行化。

[0469] (4.3.7) 在SDLI-M算法中,还可以记录本次异常检测的链表TS,从中去除本次事务,供下次使用,使得 $oldTS=TS-T$,达到oldTS的实时维护。可选地,TS里面还可以有很多合并后的逻辑事务,使得TS的长度变小,一方面节约了存储空间,另一方面有助于加速后续每个操作的判断。

[0470] 在一些实施例中,通过使用哈希表作为集合的数据结构,可以在线性时间复杂度内得到不同集合之间的交集或者并集。

[0471] (二)改进的基于upper的异常检测方法

[0472] (4.1) 当前进入验证阶段的待判断的目标事务为T,此事务T的初始值是一个物理事务,在经过动边合并之后变成一个逻辑事务。

[0473] (4.2) 从链表TS中取出第一个事务定为TL,分别对T.upper和TL.upper进行初始化,也即是设定一些隶属变量的初始值: $T.upper=false$; $TL.upper=false$ 。

[0474] (4.3) 对两事务T、TL的upper值初始化完毕后,循环执行下述操作,直到链表为空:

[0475] (4.3.1) 获取第一集合 $S1=DS_R(T) \cap DS_W(TL)$, S1为目标事务T的读集和并发事务TL的写集之间的交集。对于S1中的每一个变量x,若目标事务T读变量x的版本 $\geq$ 并发事务TL写变量x的版本,则置 $T.upper=true$;否则,置 $TL.upper=true$ 。

[0476] (4.3.2) 获取第二集合 $S2=DS_R(TL) \cap DS_W(T)$, S2为目标事务T的写集和并发事务TL的读集之间的交集。对于S2中的每一个变量x,若目标事务T写变量x的版本 $>$ 并发事务TL读变量x的版本,则置 $T.upper=true$;否则,置 $TL.upper=true$ 。

[0477] (4.3.3) 获取第三集合 $S3 = DS_W(T) \cap DS_W(TL)$, $S3$ 为目标事务 T 的写集和并发事务 TL 的写集之间的交集。

[0478] (4.3.4) 获取第四集合 $S4 = DS_R(TL) \cap DS_R(T)$, $S4$ 为目标事务 T 的读集和并发事务 TL 的读集之间的交集。对于 $S4$ 中的每一个变量 x , 若目标事务 T 读变量 x 的版本 $>$ 并发事务 TL 读变量 x 的版本, 则置 $T.upper = true$; 若目标事务 T 读变量 x 的版本 $<$ 并发事务 TL 读变量 x 的版本, 则置 $TL.upper = true$; 若目标事务 T 读变量 x 的版本 $=$ 并发事务 TL 读变量 x 的版本, 则不做任何操作。

[0479] (4.3.5) 如果 $S1 \cup S2 \cup S3 \cup S4$ 为空集 (简化计算为: $S1 = \emptyset$ 且 $S2 = \emptyset$ 且 $S3 = \emptyset$ 且 $S4 = \emptyset$), 则说明 TL 和 T 不是并发事务, 把 TL 重新加入到链表 TS 的尾部。

[0480] (4.3.6) 否则, $S1 \cup S2 \cup S3 \cup S4$ 不为空集, 说明 $S1$ 、 $S2$ 、 $S3$ 、 $S4$ 中至少存在一个集合不是空集, 此时 TL 和 T 是并发事务, 进行异常检测, 一旦发现异常, 需要报告异常并回滚目标事务 T 。异常检测过程遵循下述规则:

[0481] (4.3.6.1) 如果 $S3 \neq \emptyset$, 在满足如下两种条件中任一项时, 节点设备则报告写写异常发生, 循环终止。

[0482] 条件一、如果隔离级别参数 $\neq RWU$ (只有 RWU 级别不禁止脏写异常), TL 没有提交完成且 $T.no_committed = 1$, 存在写写冲突, 数据异常类型为脏写异常。

[0483] 条件二、如果隔离级别参数 $\geq SH$ (SH 级别禁止丢失更新发生, 所以高于或等于 SH 的隔离级别均报告此异常), TL 已经提交且 $S1 \cap S3$ 不是空集, 上报数据异常, 数据异常类型为丢失更新异常。

[0484] (4.3.6.2) 否则, 在 $S3 = \emptyset$ 的条件下, 如果 $S1 \neq \emptyset$ 、 $S2 \neq \emptyset$ 或者 $S4 \neq \emptyset$, 可以分为如下三种情况进行讨论:

[0485] 情况一、如果目标事务 T 和 TL 构成动边交叉, 则报告异常发生, 循环终止。在一些实施例中, $T.upper = TL.upper = true$ 时, 认为目标事务 T 与 TL 之间构成动边交叉, 此时节点设备可以报告异常发生并使得循环终止。

[0486] 情况二、否则, 如果目标事务 T 和 TL 没有构成动边交叉, 在隔离级别参数 $= S$ 的情况下, 节点设备还需要检查是否存在完整性约束, 如果本事务所读写的数据违反了完整性约束, 则报告异常发生并使得循环终止, 通过完整性约束可以检测出来 $DLI-M$ 算法中所无法解决的谓词写偏序异常。

[0487] 也即是说, 对于任一并发事务 TL , 在基于 $upper$ 值检测出不存在动边交叉之后, 本来该任一并发事务 TL 与目标事务 T 之间的目标检测结果被置为“不存在数据异常”, 但在可串行化 S 级别下还需要识别谓词写偏序异常 (动边交叉与合并算法无法识别), 因此节点设备响应于数据库系统的隔离级别为可串行化 S , 对该任一并发事务 TL 与该目标事务 T 进行完整性约束检测; 响应于该任一并发事务 TL 与该目标事务 T 之间违反完整性约束, 将该目标检测结果调整为存在数据异常且数据异常类型为谓词写偏序异常。其中, 完整性约束的检测过程将在下一个实施例中进行详述。

[0488] 情况三、否则, 如果目标事务 T 和 TL 没有构成动边交叉, 且符合完整性约束, 那么没有任何异常, 对目标事务 T 和 TL 进行动边合并, 设定新的事务为 T_new 。

[0489] 在动边合并过程中, 节点设备把并发事务 TL 的读集并入目标事务 T 的读集, 将并发

事务TL的写集并入目标事务T的写集;如果并发事务TL没有提交,则令T-new的目标参数T-new.no_committed++(自增1),表示合并得到的新事务中拥有某个已经提交的事物的读写集的成分;令 $T=T\text{-new}$,用T-new给T赋值,然后执行下一次循环。

[0490] (4.3.7) 如果循环结束,没有异常报告,则目标事务T可以提交。如果隔离级别参数=S则保证了可串行化。

[0491] (4.3.8) 在SDLI-M算法中,还可以记录本次异常检测的链表TS,从中去除本次事务,供下次使用,使得 $oldTS=TS-T$,达到oldTS的实时维护。可选地,TS里面还可以有很多合并后的逻辑事务,使得TS的长度变小,一方面节约了存储空间,另一方面有助于加速后续每个操作的判断。

[0492] 在一些实施例中,通过使用哈希表作为集合的数据结构,可以在线性时间复杂度内得到不同集合之间的交集或者并集。

[0493] 本申请实施例提供的SDLI-M算法的效率与上述实施例中DLI-M算法的效率相同,可以适用于单机数据库系统的事务处理。在SDLI-M算法中,响应于异常检测结果为存在数据异常,节点设备可以根据数据异常类型以及数据库系统的隔离级别,确定目标事务的执行结果,该执行结果用于表示提交该目标事务或者回滚该目标事务,从而实现了多个不同的隔离级别。

[0494] 在一些实施例中,响应于数据异常类型包括脏写异常且该隔离级别不为读写未提交RWU,将该执行结果确定为回滚该目标事务(对应于4.3.5.1的条件一);响应于数据异常类型包括丢失更新异常且该隔离级别高于或等于快照读历史SH,将该执行结果确定为回滚该目标事务(对应于4.3.5.1的条件二);响应于数据异常类型包括读异常且该隔离级别高于或等于快照读历史SH,将该执行结果确定为回滚该目标事务(对应于4.3.5.2的情况一中隔离级别参数 $\geq SH$ 的描述);响应于数据异常类型包括写异常且该隔离级别高于或等于可重复读RR,将该执行结果确定为回滚该目标事务(对应于4.3.5.2的情况一中隔离级别参数 $\geq RR$ 的描述);响应于数据异常类型包括谓词写偏序异常且该隔离级别为可串行化S,将该执行结果确定为回滚该目标事务(对应于4.3.5.2的情况二)。

[0495] 上述所有可选技术方案,可以采用任意结合形成本公开的可选实施例,在此不再一一赘述。

[0496] 上述实施例中提供的SDLI-M算法可以应用于单机数据库系统的事务处理,而在本申请实施例中,将提供一种面向分布式数据库系统的基于快照的数据状态一致性动边交叉与合并算法(Distributed Snapshot-based DLI-M, DSDLI-M),在介绍DSDLI-M算法之前,首先对分布式事务-子事务之间的隔离级别的关系进行说明。

[0497] 在分布式数据库系统中,如果父事务的隔离级别强于子事务的隔离级别,则强制子事务的隔离级别同父事务的隔离级别保持一致,即父事务的隔离级别代表整个系统的隔离级别。如果父事务的隔离级别弱于子事务的隔离级别,对于单变量的事务,由于其不可能构成分布式事务,则强制父事务的隔离级别同子事务的隔离级别保持一致,即子事务的隔离级别代表整个系统的隔离级别。对于多变量的事务,由于有可能构成分布式异常,因此需要定义新的级别,从弱到强分为A、B、C、D四级。分布式事务-子事务之间的隔离级别的关系如下表7所示。

[0498] 表7

[0499]	子事务 父事务	读写未提交	读写已提交	快照历史	可重复读	可串行化
	读写未提交	读写未提交	读写已提交	A-弱 3	A-弱 2	A-弱 1
	读写已提交	读写已提交	读写已提交	B-弱 3	B-弱 2	B-弱 1
	快照历史	快照历史	快照历史	快照历史	C-弱 2	C-弱 1
	可重复读	可重复读	可重复读	可重复读	可重复读	D-弱 1
	可串行化	可串行化	可串行化	可串行化	可串行化	可串行化

[0500] 其中，“快照历史”也即是快照读历史SH级别的简称，后文不做赘述。下面分别针对新定义的隔离级别进行说明。

[0501] 1) A-弱3级别：父事务读写未提交+子事务快照历史(父RWU+子SH)，遵从子快照历史，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0502] 2) A-弱2级别：父事务读写未提交+子事务可重复读(父RWU+子RR)，遵从子可重复读，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0503] 3) A-弱1级别：父事务读写未提交+子事务可串行化(父RWU+子S)，遵从子可串行化，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0504] 4) B-弱3级别：父事务读写已提交+子事务快照历史(父RWC+子SH)，遵从子快照历史，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0505] 5) B-弱2级别：父事务读写已提交+子事务可重复读(父RWC+子RR)，遵从子可重复读，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0506] 6) B-弱1级别：父事务读写已提交+子事务可串行化(父RWC+子S)，遵从子可串行化，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0507] 7) C-弱2级别：父事务快照历史+子事务可重复读(父SH+子RR)，遵从子可重复读，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0508] 8) C-弱1级别：父事务快照历史+子事务可串行化(父SH+子S)，遵从子可串行化，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、分布式锯齿波等数据异常。

[0509] 9) D-弱1级别：父事务可重复读+子事务可串行化(父RR+子S)，遵从子可串行化，整体允许分布式读偏序、分布式读写偏序、分布式台阶式、分布式写偏序、分布式谓词写偏序、

分布式锯齿波等数据异常。

[0510] 分布式事务的隔离级别,可以具有如下2种定义形式:

[0511] 第一种、两级分布式事务隔离级别:如表7定义,父事务和子事务可各自定义各自的隔离级别,但整体事务遵从表7中隔离级别的定义。

[0512] 第二种、统一的分布式事务隔离级别:如表7中加粗字体的方格定义,父事务和子事务不可各自定义各自的隔离级别,整体事务遵从表7的定义中加粗字体的方格定义。

[0513] 针对上述第二种分布式事务隔离级别,根据表6和前述规则(表7中加粗字体所在的方格定义),可定义出等同于表6的表8,只是在表8中标明了分布式异常和隔离级别之间的关系。

[0514] 表8

异常 \ 级别	读写未提交	读写已提交	快照	可重复读	可串行化	
	脏写	脏读	中间读	不可重复读	幻读	丢失更新
[0515]	脏写	允许	不允许	不允许	不允许	不允许
	脏读	允许	不允许	不允许	不允许	不允许
	中间读	允许	不允许	不允许	不允许	不允许
	不可重复读	允许	允许	允许	不允许	不允许
	幻读	允许	允许	允许	不允许	不允许
	丢失更新	允许	允许	不允许	不允许	不允许

[0516]	读偏序（2 变量，分布式）	允许	允许	不允许	不允许	不允许
	读写偏序（2 变量，分布式）	允许	允许	不允许	不允许	不允许
	台阶式（多变量，分布式）	允许	允许	不允许	不允许	不允许
	写偏序（2 变量，分布式）	允许	允许	允许	不允许	不允许
	谓词写偏序（多变量，分布式）	允许	允许	允许	允许	不允许
	锯齿波式（多变量，分布式）	允许	允许	允许	不允许	不允许

[0517] 这标明,在单机系统中,需要依据表6或表8(此2表在本质上完全一样)设定隔离级别;在分布式系统中,需要依据表8、尤其是表8中标注出分布式异常的栏目所对应的异常(包括读偏序、读写偏序、台阶式、写偏序、谓词写偏序以及锯齿波式数据异常),在分布式系统中在不同的级别下解决各自对应的数据异常;且分布式系统的各个子节点上在不同的级别下各自解决表8对应的各种分布式异常。如快照历史级别,读偏序在单个节点上需要解决,在分布式系统上也需要解决。

[0518] 在提供了上述分布式事务的隔离级别的基础上,针对上述统一的分布式事务隔离级别(第二种分布式事务隔离级别),在本申请实施例,基于SDLI-M算法还提供了DSDLI-M算法,采用全局快照技术,可以应用于分布式事务型数据库系统,下面进行详述。

[0519] 1、快照点的建立方法,全局快照点构建法

[0520] (1.1) 采用HLC算法构造全局的混合逻辑时钟。这意味着每个子节点上各自的逻辑

时钟,通过HLC保持了因果序的同步(全局递增),而单个节点上局部事务的时钟可自增(局部递增)。在HLC时间戳中,可以采用三元组 $\{pt, l, c\}$ 表示本地物理时钟(pt ,三元组内的第一个元素)和混合逻辑时钟(l 和 c ,分别三元组内的第二、三个元素)。

[0521] (1.2)在事务协调器上,事务的第一个SQL语句执行时,通过HLC算法分配快照点(获取本协调器上最新的HLC值作为快照点)。也即获取Snapshot Set= $S_s = \{St\}$, St 是一个全局HLC分配的时间值。

[0522] (1.3)分布式事务的SQL语句执行时,随着SQL语句在子节点上执行,事务协调器将快照 St 下发给各个子节点。

[0523] (1.4)每个子节点上,执行基于HLC的数据项的可见性判断算法:

[0524] (1.4.1)满足读已提交规则。

[0525] (1.4.2)数据项的版本链从新到旧有序存储,遍历该版本链,找到第一个较新的且满足如下(1.4.3)中算法的数据项为止。

[0526] (1.4.3)按给定的HLC值(从事务协调器传入),假定为 $local_HLC$,其三元组为 $\{10, 10, 0\}$,在各个子节点的版本链上进行遍历,找到和 $local_HLC.l$ 值(该值为10)相同的数据版本,然后再往后读一个版本,该版本就是处于全局读一致的版本。

[0527] 图12是本申请实施例提供的一种基于HLC建立全局快照的原理图,如1200所示,较粗的实线沿着逻辑时钟 c 值为0从节点0到节点1,一直到节点3,三元组的变化分别为:节点0 $\{10, 10, 0\}$ 、节点1 $\{0, 0, 0\}$ 、节点2 $\{1, 1, 0\}$ 、节点3 $\{2, 2, 0\}$,这表明在这样的一个状态下,四个节点处于一致的状态,也即是说,逻辑时钟 c 值相同的最大版本之间符合一致性状态。在图12中,实线箭头表示两个节点间发生了Send(发送)和Receive(接收)事件,同一个节点上的多个三元组相连表示本地事件发生。

[0528] (1.4.4)该数据项的可见性判断算法,要求数据版本上的提交时间戳 tc 的数据结构,和HLC保持一致,也即要求提交时间戳 tc 也为基于HLC算法分配的时间戳。

[0529] (1.5)事务协调器在一个集群中可以有多,事务协调器对应于协调节点设备,子节点对应于数据节点设备,当事务协调器只有一个时,对应于中心化的事务处理技术,当事务协调器包括多个时,对应于去中心化的事务处理技术。

[0530] (1.6)事务在提交时,从全局HLC获取提交时间戳 tc ,然后修改各个子节点上事务提交标识的时间戳值。

[0531] 2、事务管理器

[0532] 在一个集群中,事务管理器(每个物理节点的数据库实例上存在一个事务管理器)扮演着事务协调器的角色。事务协调器(协调节点设备)发起一个事务,称为父事务。在不同的子节点(数据节点设备)上执行的,称为子事务。父子事务之间,通过全局事务标识 tid 进行识别,每个子事务,也有自己的子事务的标识 $ctid$,在子节点上一个事务的完整标识为 $\{tid, ctid\}$ 。

[0533] 在协调节点设备上执行目标事务 T 的第一个SQL语句时,为目标事务 T 创建快照;协调节点设备将该快照下发到目标事务 T 所涉及的各个数据节点设备。也即是说,在分布式数据库系统中可以采用HLC形式的快照,该HLC形式的快照由协调节点设备下发。

[0534] 当目标事务 T 提交时,各个数据节点设备采用DLI-M算法各自判断本节点上是否存在数据异常,也即获取本节点的异常检测结果。如果本节点的异常检测结果为存在数据异

常,则回滚本地的子事务,然后发送消息通知协调节点设备的父事务,由父事务再通知其他相关数据节点设备的子事务,从而进行全局的事务回滚。

[0535] 否则,如果本节点的异常检测结果为不存在(单节点的)数据异常,但无法保证其他数据节点设备上是否存在数据异常,因此协调节点设备需要在汇总各个数据节点设备上报的相关信息并上报父事务,由父事务仲裁是否存在全局的数据异常,如果存在全局的数据异常,则发送消息给各个数据节点设备,各个数据节点设备进行全局回滚;否则,则事务提交成功。

[0536] 3、数据读取阶段

[0537] 在一些实施例中,在DSDLI-M算法的数据读取阶段,如果隔离级别参数 $<RR$ (隔离级别低于 RR),每执行一次读操作,都将获取一次快照,这种情况下每次读都具有新快照则允许幻读异常发生;否则,如果隔离级别参数 $\geq RR$ (隔离级别高于或等于 RR),仅在第一次执行读操作时获取快照,后续每次读操作均使用第一次获取的快照来读取数据,这种情况下只使用第一次的快照,可避免幻读异常和不可重复读异常发生,1个变量的不可重复读异常在此数据读取阶段内已经被识别,因此不用进入到动边交叉与合并算法中进行识别。需要说明的是,此时是避免发生某一类数据异常,而不是在检测发现异常然后回滚事务,所以没有报告异常并终结事务的步骤。

[0538] 在一些实施例中,当隔离级别参数 $\geq RWC$ 时,与上述“读写阶段事务读写集形成算法”中所涉及的基于HLC的可见性判断算法保持一致,这里不做赘述,但当隔离级别参数 $< RWC$ 时,可以对前述基于HLC的可见性判断算法进行修改,使得最新的一个未提交版本可以被读取到,从而实现了RWU级别。

[0539] 在一些实施例中,如果采用OCC框架,在隔离级别参数 $> RR$ 时,则对于每个事务中的第二个读操作,先从读集和写集中验证是否存在要读取的对象,如果存在,再从读写集合中获取;否则,需要从底层的存储层内获取。

[0540] 这里DSDLI-M算法的数据读取阶段与上一个实施例中SDLI-M算法的数据读取阶段类似,这里不做赘述。

[0541] 4、每个操作发生时,执行准备阶段算法

[0542] 在目标事务T的每个操作(或读或写)发生时,进入准备阶段并开始做准备工作,准备阶段用于为目标事务T是否可提交做好事务并发访问控制异常验证前的准备工作。

[0543] 在准备阶段中,节点设备进入临界区,遍历目标事务T的读集和写集,找出目标事务T的所有并发事务的事务标识,放入一个链表TS内进行存储,在找到后退出临界区。在内存中,每个数据项上存有曾经读写过本数据项但尚未提交的事务的事务标识。

[0544] 在一些实施例中,DSDLI-M算法与OCC算法耦合时,准备阶段的算法对应于OCC的验证阶段。

[0545] 在一些实施例中,如果已有链表oldTS不为空,则把目标事务直接加入到TS中(也即令 $TS = oldTS + T$),不再遍历目标事务的读集和写集,从而可以减少对读写集合的遍历逻辑,从而可提高事务处理效率。

[0546] 上述准备阶段算法与上述SDLI-M算法所介绍的准备阶段算法类似,这里不做赘述。

[0547] 5、子节点事务验证算法

[0548] DSDLI-M算法所提供的子节点事务验证算法,与SDLI-M算法所提供的改进的动边交叉与合并算法相类似,但仍存在一些差异。

[0549] 在一些实施例中,DSDLI-M算法与OCC算法耦合时,子节点事务验证算法对应于OCC的验证阶段,即当操作发生时,协调节点设备立刻开始进行验证,而不是等到进入OCC的验证阶段,这一过程称为“子节点去除”过程,此规则的目的是加快子事务的执行速度,从而提高集群系统的整体事务响应效率。

[0550] 对每个数据节点设备(子节点)来说,其自身的链表TS称为cTS,在子节点上进行实时维护,没有事务发生,即维护新发生的子事务或局部事务进入cTS。每个子节点cTS的维护方式,同单机系统中描述的TS的维护方式。

[0551] 在DSDLI-M算法中,针对子节点的事务验证算法仍然包括两种不同的类型,第一种是改进的基于DLI和VS的异常检测方法,第二种是改进的基于upper的异常检测方法,下面将分类讨论。

[0552] (一)子节点上基于DLI和VS的事务验证方法

[0553] (5.1)每个操作发生时,即设定一些隶属操作级的变量的初始值:DLI=0;VS=0。

[0554] (5.2)从子节点自身的链表cTS中取出第一个事务定为TL,循环执行下述操作,直到链表为空:

[0555] (5.2.1)获取第一集合 $S1 = DS_R(T) \cap DS_W(TL)$,S1为目标事务(父事务)所对应子事务T的读集和并发事务TL的写集之间的交集。如果S1不为空,则DLI=DLI++;如果S1中某个非空元素的数据状态不同,则VS=VS++。

[0556] (5.2.2)获取第二集合 $S2 = DS_R(TL) \cap DS_W(T)$,S2为子事务T的写集和并发事务TL的读集之间的交集。如果S1不为空,则DLI=DLI++;如果S2中某个非空元素的数据状态不同,则VS=VS++。

[0557] (5.2.3)获取第三集合 $S3 = DS_W(T) \cap DS_W(TL)$,S3为子事务T的写集和并发事务TL的写集之间的交集。

[0558] (5.2.4)如果 $S1 \cup S2 \cup S3$ 为空集(简化计算为: $S1 = \emptyset$ 且 $S2 = \emptyset$ 且 $S3 = \emptyset$),则说明TL和T不是并发事务,把TL重新加入到链表cTS的尾部。

[0559] (5.2.5)否则, $S1 \cup S2 \cup S3$ 不为空集,TL和T是并发事务,进行异常检测,一旦发现异常,回滚本地的子事务T,并立即向父事务返回目标事务的回滚消息。异常检测过程遵循下述规则:

[0560] (5.2.5.1)如果 $S3 \neq \emptyset$,在满足如下两种条件中任一项时,子节点则报告写写异常发生,循环终止。

[0561] 条件一、如果隔离级别参数 $\neq RWU$ (只有RWU级别不禁止脏写异常),TL没有提交完成且T.no_committed=1,存在写写冲突,数据异常类型为脏写异常。

[0562] 条件二、如果隔离级别参数 $\geq SH$ (SH级别禁止丢失更新发生,所以高于或等于SH的隔离级别均报告此异常),TL已经提交且 $S1 \cap S3$ 不是空集,上报数据异常,数据异常类型为丢失更新异常。

[0563] (5.2.5.2)否则,在 $S3 = \emptyset$ 的条件下,如果 $S1 \neq \emptyset$ 或者 $S2 \neq \emptyset$,可以分为如下三种情况进行讨论:

[0564] 情况一、如果子事务T和TL构成动边交叉,则报告异常发生,循环终止。在一些实施例中, $DLI \geq 2$ 时,认为子事务T与TL之间构成动边交叉,此时节点设备可以根据隔离级别确定是否报告异常发生并使得循环终止。

[0565] 可选地,如果隔离级别参数 $\geq SH$ (禁止读偏序异常和台阶式异常),且 $VS=1$,则报告异常发生并使得循环终止,此时数据异常类型为读异常,包括2个变量的读偏序异常、3个及3个以上变量的台阶式异常(1个变量的不可重复读异常已经在数据读取阶段中被识别了,在异常检测阶段无需进行重复识别)。

[0566] 可选地,如果隔离级别参数 $\geq RR$ (禁止写偏序异常和锯齿波式异常),且 $VS \geq 2$,则报告异常发生并使得循环终止,此时数据异常类型为写异常,包括2个变量的写偏序异常、3个及3个以上变量的锯齿波式异常。

[0567] 情况二、否则,如果子事务T和TL没有构成动边交叉,在隔离级别参数=S的情况下,节点设备还需要检查是否存在完整性约束,如果本事务所读写的数据违反了完整性约束,则报告异常发生并使得循环终止,通过完整性约束可以检测出来DLI-M算法中所无法解决的谓词写偏序异常。

[0568] 情况三、否则,如果子事务T和TL没有构成动边交叉,且符合完整性约束,那么没有任何异常,对子事务T和TL进行动边合并,设定新的事务为T-new。

[0569] 在动边合并过程中,节点设备把并发事务TL的读集并入子事务T的读集,将并发事务TL的写集并入子事务T的写集;如果并发事务TL没有提交,则令T-new的目标参数T-new.no_committed++(自增1),表示合并得到的新事务中拥有某个已经提交的事务的读写集的成分;令 $T=T\text{-new}$,用T-new给T赋值,然后执行下一次循环。

[0570] (5.2.6) 如果循环结束,没有异常报告,则子事务T局部可以提交。子事务T向父事务发送本子事务T的依赖关系和无局部异常信息(也即单节点的异常检测结果)。

[0571] (5.2.7) 记录本次异常检测的链表cTS,从中去除本次事务,供下次使用,使得 $oldTS=cTS-T$,达到oldTS的实时维护。可选地,cTS里面还可以有很多合并后的逻辑事务,使得cTS的长度变小,一方面节约了存储空间,另一方面有助于加速后续每个操作的判断。

[0572] 在一些实施例中,通过使用哈希表作为集合的数据结构,可以在线性时间复杂度内得到不同集合之间的交集或者并集。

[0573] (二) 子节点上基于upper的事务验证方法

[0574] (5.1) 每个操作发生时从子节点自身的链表cTS中取出第一个事务定为TL,分别对T.upper和TL.upper进行初始化,也即是设定一些隶属变量的初始值: $T.upper=false$; $TL.upper=false$ 。

[0575] (5.2) 对两事务T、TL的upper值初始化完毕后,循环执行下述操作,直到链表为空:

[0576] (5.2.1) 获取第一集合 $S1=DS_R(T) \cap DS_W(TL)$,S1为目标事务(父事务)所对应子事务T的读集和并发事务TL的写集之间的交集。对于S1中的每一个变量x,若子事务T读变量x的版本 $\geq$ 并发事务TL写变量x的版本,则置 $T.upper=true$;否则,置 $TL.upper=true$ 。

[0577] (5.2.2) 获取第二集合 $S2=DS_R(TL) \cap DS_W(T)$,S2为子事务T的写集和并发事务TL的读集之间的交集。对于S2中的每一个变量x,若子事务T写变量x的版本 $>$ 并发事务TL读变量x的版本,则置 $T.upper=true$;否则,置 $TL.upper=true$ 。

[0578] (5.2.3) 获取第三集合 $S3=DS_W(T) \cap DS_W(TL)$,S3为子事务T的写集和并发事务TL的

写集之间的交集。

[0579] (5.2.4) 获取第四集合 $S4 = DS_R(TL) \cap DS_R(T)$, $S4$ 为子事务 T 的读集和并发事务 TL 的读集之间的交集。对于 $S4$ 中的每一个变量 x , 若子事务 T 读变量 x 的版本 $>$ 并发事务 TL 读变量 x 的版本, 则置 $T.upper = true$; 若子事务 T 读变量 x 的版本 $<$ 并发事务 TL 读变量 x 的版本, 则置 $TL.upper = true$; 若子事务 T 读变量 x 的版本 $=$ 并发事务 TL 读变量 x 的版本, 则不做任何操作。

[0580] (5.2.5) 如果 $S1 \cup S2 \cup S3 \cup S4$ 为空集 (简化计算为: $S1 = \emptyset$ 且 $S2 = \emptyset$ 且 $S3 = \emptyset$ 且 $S4 = \emptyset$), 则说明 TL 和 T 不是并发事务, 把 TL 重新加入到 c 链表 TS 的尾部。

[0581] (5.2.6) 否则, $S1 \cup S2 \cup S3 \cup S4$ 不为空集, TL 和 T 是并发事务, 进行异常检测, 一旦发现异常, 回滚本地的子事务 T , 并立即向父事务返回目标事务的回滚消息。异常检测过程遵循下述规则:

[0582] (5.2.6.1) 如果 $S3 \neq \emptyset$, 在满足如下两种条件中任一项时, 节点设备则报告写写异常发生, 循环终止。

[0583] 条件一、如果隔离级别参数 $\neq RWU$ (只有 RWU 级别不禁止脏写异常), TL 没有提交完成且 $T.no_committed = 1$, 存在写写冲突, 数据异常类型为脏写异常。

[0584] 条件二、如果隔离级别参数 $\geq SH$ (SH 级别禁止丢失更新发生, 所以高于或等于 SH 的隔离级别均报告此异常), TL 已经提交且 $S1 \cap S3$ 不是空集, 上报数据异常, 数据异常类型为丢失更新异常。

[0585] (5.2.6.2) 否则, 在 $S3 = \emptyset$ 的条件下, 如果 $S1 \neq \emptyset$ 、 $S2 \neq \emptyset$ 或者 $S4 \neq \emptyset$, 可以分为如下三种情况进行讨论:

[0586] 情况一、如果子事务 T 和 TL 构成动边交叉, 则报告异常发生, 循环终止。在一些实施例中, $T.upper = TL.upper = true$ 时, 认为子事务 T 与 TL 之间构成动边交叉, 此时节点设备可以报告异常发生并使得循环终止。

[0587] 情况二、否则, 如果子事务 T 和 TL 没有构成动边交叉, 在隔离级别参数 $= S$ 的情况下, 节点设备还需要检查是否存在完整性约束, 如果本事务所读写的数据违反了完整性约束, 则报告异常发生并使得循环终止, 通过完整性约束可以检测出来 $DLI-M$ 算法中所无法解决的谓词写偏序异常。

[0588] 情况三、否则, 如果子事务 T 和 TL 没有构成动边交叉, 且符合完整性约束, 那么没有任何异常, 对子事务 T 和 TL 进行动边合并, 设定新的事务为 T_new 。

[0589] 在动边合并过程中, 节点设备把并发事务 TL 的读集并入子事务 T 的读集, 将并发事务 TL 的写集并入子事务 T 的写集; 如果并发事务 TL 没有提交, 则令 T_new 的目标参数 $T_new.no_committed++$ (自增1), 表示合并得到的新事务中拥有某个已经提交的事务的读写集的成分; 令 $T = T_new$, 用 T_new 给 T 赋值, 然后执行下一次循环。

[0590] (5.2.7) 如果循环结束, 没有异常报告, 则子事务 T 局部可以提交。子事务 T 向父事务发送本子事务 T 的依赖关系和无局部异常信息 (也即单节点的异常检测结果)。

[0591] (5.2.8) 记录本次异常检测的链表 cTS , 从中去除本次事务, 供下次使用, 使得 $oldTS = cTS - T$, 达到 $oldTS$ 的实时维护。可选地, cTS 里面还可以有很多合并后的逻辑事务, 使得 cTS 的长度变小, 一方面节约了存储空间, 另一方面有助于加速后续每个操作的判断。

[0592] 在一些实施例中,通过使用哈希表作为集合的数据结构,可以在线性时间复杂度内得到不同集合之间的交集或者并集。

[0593] 父节点(也即事务协调器、协调节点设备)在汇总各个子节点(也即数据节点设备)的依赖关系和异常检测结果之后,可以调用父节点事务验证算法进行全局异常检测,从而确定最终提交目标事务还是回滚目标事务,在本申请实施例中,提供了两种父节点事务验证算法,分别包括动边交叉合并算法和依赖关系检测算法,动边交叉合并算法可以检测出除了谓词写偏序异常外的分布式异常,而依赖关系检测算法可以检测出谓词写偏序异常,下面将分别在第6点和第7点中进行介绍。

[0594] 6、父节点事务验证算法1(动边交叉合并算法)

[0595] 依据分布式事务的隔离级别(表8),在不同的隔离级别下,分布式事务在事务协调器上,还需要分别各自解决分布式的读偏序、台阶式、写偏序、谓词写偏序、锯齿波这5类异常。而该5类异常除了谓词写偏序异常外,其他四种异常可以在父事务即协调器节点上采用类似的DLI-M算法加以解决。而谓词写偏序异常(对应可串行化S级别),需要通过在父事务所在的协调节点设备上执行完整性约束验证加以解决。

[0596] 在父事务即协调器上,构造DLI-M算法需要的信息:父事务收集子事务的读集和写集,以及数据项/变量版本上记录的冲突的并发事务(至少一个并发事务)这些信息,然后调用DLI-M算法,在事务的提交阶段,实现不同隔离级别的分布式事务验证,主要是验证分布式事务中的读偏序、台阶式、写偏序、锯齿波异常,并通过完整性约束检查验证谓词写偏序异常。

[0597] 7、父节点事务验证算法2(依赖关系检测算法)

[0598] (7.1)子事务识别完毕本地节点不存在任何数据异常的情况下,在子事务准备返回消息给父事务之前,子事务需要根据依赖图,构造并发事务的依赖关系,将依赖关系返回给父事务。

[0599] 图13是本申请实施例提供的四个事务在两个节点上构成的分布式事务读异常的原理图,如1300所示,在两个物理节点Node1和Node2上,分别有四个事务T1~T4在并发执行,Node1上存在T4→T1→T3构成依赖图,Node2上存在T3→T2→T4构成依赖图。

[0600] 图14是本申请实施例提供的三个事务在两个节点上构成的分布式事务写异常的原理图,如1400所示,在两个物理节点Node1和Node2上,分别有三个事务T1~T3在并发执行,Node1上存在T3→T1→T2构成依赖图,Node2上存在T2→T3构成依赖图。

[0601] (7.2)父事务根据各个子节点返回的多个子事务的依赖图,构造全局的依赖图,如果全局依赖图不存在环,则整个分布式事务不存在依赖环,且也没有破坏完整性约束,因此是符合可串行化S隔离级别的。

[0602] (7.3)否则,如果完整性约束被破坏,则符合可重复读RR隔离级别。

[0603] (7.4)在上述(7.3)中有环的情况下:

[0604] (7.4.1)如前述,在图13所示的示例中,父事务收集Node1的T4→T1→T3依赖图和Node2的T3→T2→T4依赖图,合并得到全局依赖图为T4→T1→T3→T2→T4,这样首尾都是T4,因此全局依赖图中存在环,所以不可串行化,且T4事务是由两个读操作组成的,所以检测到的是读偏序或台阶式异常。如果隔离级别参数 $\geq$ SH(隔离级别高于或等于快照读历史SH),则回滚父事务及所有子事务(即回滚目标事务),报告异常。

[0605] (7.4.2) 如前述,在图14所示的示例中,父事务收集Node1的 $T3 \rightarrow T1 \rightarrow T2$ 依赖图和Node2的 $T2 \rightarrow T3$ 依赖图,合并得到全局依赖图为 $T3 \rightarrow T1 \rightarrow T2 \rightarrow T3$,这样首尾都是 $T3$,因此全局依赖图中存在环,所以不可串行化,且 $T3$ 事务分别是由读写操作组成的,所以检测到的是写偏序或锯齿波式异常。如果隔离级别参数 $\geq RR$ (隔离级别高于或等于可重复读 RR),则回滚父事务及所有子事务(即回滚目标事务),报告异常。

[0606] (7.4.3) 依赖关系检测算法只在父子事务之间传递依赖图相关信息,读写集等并不在父子事务之间传递,所以对于大结果集事务的可串行化隔离级别的实现,有较大的优势。

[0607] 本申请实施例提供的DSDLI-M算法可以适用于分布式事务型数据库系统的事务处理。在DSDLI-M算法中,通过子节点和父节点处理隔离级别的技术,描绘了不同隔离级别和分布式事务之间的关联关系,使得分布式事务也具备多种隔离级别的能力。

[0608] 上述所有可选技术方案,可以采用任意结合形成本公开的可选实施例,在此不再一一赘述。

[0609] 在本申请实施例中,完善了数据状态一致性模型,根据数据状态一致性模型,形式化定义了各种数据异常,使得数据异常的定义标准化、数学化;根据本申请实施例提供的数据异常定义,提出了与传统隔离级别技术存在差异的新的、单机事务型数据库的五级隔离级别体系,并明确提出隔离级别建立的标准;提出2种适合于单机系统的并发访问控制基础算法(DLI-M算法以及SDLI-M算法),并结合数据异常的定义和隔离级别的定义,在算法中给出如何实现不同的隔离级别;提出分布式事务的隔离级别和单机系统的隔离级别的关联关系,并定义了分布式事务系统的五级隔离级别体系;基于分布式隔离级别和前述的基础算法,提出适合分布式事务处理的并发访问控制技术(DSDLI-M算法),DSDLI-M算法完整地描述了事务处理的各个阶段的具体过程,并给出若干个不同的子算法实现父事务的并发控制。

[0610] 图15是本申请实施例提供的一种事务处理装置的结构示意图,请参考图15,该装置包括:

[0611] 第一获取模块1501,用于响应于目标事务的读写操作执行,获取该目标事务的至少一个并发事务,该至少一个并发事务与该目标事务在事务执行期间涉及到对相同的数据项进行读写操作;

[0612] 第二获取模块1502,用于根据该至少一个并发事务与该目标事务的读写集合,获取该至少一个并发事务与该目标事务之间的异常检测结果,该异常检测结果用于表示该至少一个并发事务与该目标事务之间是否存在数据异常;

[0613] 合并提交模块1503,用于响应于该异常检测结果为不存在数据异常,将该至少一个并发事务与该目标事务的读写集合进行合并,提交该目标事务。

[0614] 本申请实施例提供的装置,通过响应于目标事务的读写操作执行,获取该目标事务的至少一个并发事务,根据该至少一个并发事务与该目标事务的读写集合,获取该至少一个并发事务与该目标事务之间的异常检测结果,响应于该异常检测结果为不存在数据异常,将该至少一个并发事务与该目标事务的读写集合进行合并,提交该目标事务,这种方式能够在目标事务的每个操作执行时全面检测出数据库系统内各种各样的数据异常,保证数据状态的一致性,在保证数据状态一致性的基础上,这种事务处理机制既不完全依赖于封

锁技术也不完全依赖于依赖图技术,避免了限制数据库系统的并发度,能够提升数据库系统的事务处理效率。

[0615] 在一种可能实施方式中,基于图15的装置组成,该第二获取模块1502包括:

[0616] 获取子模块,用于对于该至少一个并发事务中任一并发事务,根据该任一并发事务与该目标事务的读写集合,获取该任一并发事务与该目标事务之间的目标检测结果;

[0617] 确定子模块,用于响应于该目标检测结果为存在数据异常,将该异常检测结果确定为存在数据异常;否则,对该至少一个并发事务中该任一并发事务的下一个并发事务执行获取目标检测结果的步骤;

[0618] 该确定子模块,还用于响应于该至少一个并发事务的目标检测结果均为不存在数据异常,将该异常检测结果确定为不存在数据异常。

[0619] 在一种可能实施方式中,基于图15的装置组成,该获取子模块包括:

[0620] 第一获取单元,用于获取第一集合、第二集合以及第三集合,该第一集合为该目标事务的读集与该任一并发事务的写集之间的交集,该第二集合为该目标事务的写集与该任一并发事务的读集之间的交集,该第三集合为该目标事务的写集与该任一并发事务的写集之间的交集;

[0621] 第二获取单元,用于响应于该第三集合不是空集,根据该任一并发事务的提交情况获取该目标检测结果;

[0622] 第三获取单元,用于响应于该第三集合是空集且该第一集合或者该第二集合中至少一项不是空集,根据该第一集合和该第二集合获取该目标检测结果。

[0623] 在一种可能实施方式中,基于图15的装置组成,该获取子模块包括:

[0624] 第一获取单元,用于获取第一集合、第二集合、第三集合以及第四集合,该第一集合为该目标事务的读集与该任一并发事务的写集之间的交集,该第二集合为该目标事务的写集与该任一并发事务的读集之间的交集,该第三集合为该目标事务的写集与该任一并发事务的写集之间的交集,该第四集合为该目标事务的读集与该任一并发事务的读集之间的交集;

[0625] 第二获取单元,用于响应于该第三集合不是空集,根据该任一并发事务的提交情况获取该目标检测结果;

[0626] 第四获取单元,用于响应于该第三集合是空集且该第一集合、该第二集合或者该第三集合中至少一项不是空集,根据该第一集合、该第二集合和该第四集合获取该目标检测结果。

[0627] 在一种可能实施方式中,该第二获取单元用于:

[0628] 若该任一并发事务未提交且该目标事务的目标参数为一,将该目标检测结果获取为存在数据异常且数据异常类型为脏写异常,该目标参数用于表示该目标事务的读写集合成分所对应的已提交事务数量;

[0629] 若该任一并发事务已提交且该第一集合与该第三集合之间的交集不是空集,将该目标检测结果获取为存在数据异常且数据异常类型为丢失更新异常。

[0630] 在一种可能实施方式中,基于图15的装置组成,该第三获取单元包括:

[0631] 第一更新子单元,用于根据该第一集合和该第二集合,更新该任一并发事务与该目标事务之间的动边交叉值和变量状态值,该动边交叉值用于表示该任一并发事务与该目

标事务在数据状态矩阵中操作的不同数据项之间的线段交叉情况,该变量状态值用于表示该任一并发事务与该目标事务涉及操作到处于不同数据状态的变量情况;

[0632] 第一获取子单元,用于基于更新后的动边交叉值和更新后的变量状态值,获取该目标检测结果。

[0633] 在一种可能实施方式中,该第一更新子单元用于:

[0634] 响应于该第一集合不是空集,将动边交叉值更新为已有值加一所得的数值;响应于该第一集合中存在数据状态不同的变量,将变量状态值更新为已有值加一所得的数值;

[0635] 响应于该第二集合不是空集,将动边交叉值更新为已有值加一所得的数值;响应于该第二集合中存在数据状态不同的变量,将变量状态值更新为已有值加一所得的数值。

[0636] 在一种可能实施方式中,基于图15的装置组成,该第一获取子单元包括:

[0637] 确定子子单元,用于响应于该更新后的动边交叉值大于或等于二,将该目标检测结果获取为存在数据异常;基于该更新后的变量状态值,确定数据异常类型;

[0638] 获取子子单元,用于否则,将该目标检测结果获取为不存在数据异常。

[0639] 在一种可能实施方式中,该确定子子单元用于:

[0640] 响应于该更新后的变量状态值为一,确定该数据异常类型为读异常;

[0641] 响应于该更新后的变量状态值大于或等于二,确定该数据异常类型为写异常。

[0642] 在一种可能实施方式中,基于图15的装置组成,该第四获取单元包括:

[0643] 第二更新子单元,用于根据该第一集合、该第二集合和该第四集合,更新该任一并发事务的布尔型以及该目标事务的布尔型,该布尔型用于表示对应事务所构成的动边在数据状态矩阵中的垂直位置关系且该布尔型的初始值为假;

[0644] 第二获取子单元,用于响应于更新后的该目标事务的布尔型和更新后的该任一并发事务的布尔型均为真,将该目标检测结果获取为存在数据异常;否则,将该目标检测结果获取为不存在数据异常。

[0645] 在一种可能实施方式中,该第二更新子单元用于:

[0646] 响应于该第一集合不是空集,对于该第一集合中任一变量,若该变量在该目标事务的读集中的版本号大于或等于该变量在该任一并发事务的写集中的版本号,将该目标事务的布尔型更新为真;否则,将该任一并发事务的布尔型更新为真;

[0647] 响应于该第二集合不是空集,对于该第二集合中任一变量,若该变量在该目标事务的写集中的版本号大于该变量在该任一并发事务的读集中的版本号,将该目标事务的布尔型更新为真;否则,将该任一并发事务的布尔型更新为真;

[0648] 响应于该第四集合不是空集,对于该第四集合中任一变量,若该变量在该目标事务的读集中的版本号大于该变量在该任一并发事务的读集中的版本号,将该目标事务的布尔型更新为真;若该变量在该目标事务的读集中的版本号小于该变量在该任一并发事务的读集中的版本号,将该任一并发事务的布尔型更新为真。

[0649] 在一种可能实施方式中,基于图15的装置组成,该装置还包括:

[0650] 检测调整模块,用于响应于数据库系统的隔离级别为可串行化,对该任一并发事务与该目标事务进行完整性约束检测;响应于该任一并发事务与该目标事务之间违反完整性约束,将该目标检测结果调整为存在数据异常且数据异常类型为谓词写偏序异常。

[0651] 在一种可能实施方式中,基于图15的装置组成,该装置还包括:

[0652] 第一确定模块,用于响应于该异常检测结果为存在数据异常,根据数据异常类型以及数据库系统的隔离级别,确定该目标事务的执行结果,该执行结果用于表示提交该目标事务或者回滚该目标事务。

[0653] 在一种可能实施方式中,该第一确定模块用于:

[0654] 响应于数据异常类型包括脏写异常且该隔离级别不为读写未提交,将该执行结果确定为回滚该目标事务;

[0655] 响应于数据异常类型包括丢失更新异常且该隔离级别高于或等于快照读历史,将该执行结果确定为回滚该目标事务;

[0656] 响应于数据异常类型包括读异常且该隔离级别高于或等于快照读历史,将该执行结果确定为回滚该目标事务;

[0657] 响应于数据异常类型包括写异常且该隔离级别高于或等于可重复读,将该执行结果确定为回滚该目标事务;

[0658] 响应于数据异常类型包括谓词写偏序异常且该隔离级别为可串行化,将该执行结果确定为回滚该目标事务。

[0659] 在一种可能实施方式中,基于图15的装置组成,该装置还包括:

[0660] 第二确定模块,用于对于该目标事务的任一次读操作,确定该任一次读操作的快照;

[0661] 第三确定模块,用于根据该任一次读操作的读取条件以及该任一次读操作的快照,确定相对于该目标事务可见的目标数据项,将该目标数据项添加至该目标事务的读集中。

[0662] 在一种可能实施方式中,该第二确定模块用于:

[0663] 响应于数据库系统的隔离级别低于可重复读,在执行该任一次读操作时获取一次快照;否则,将该目标事务第一次读操作的快照确定为该任一次读操作的快照。

[0664] 在一种可能实施方式中,该第三确定模块用于:

[0665] 响应于数据库系统的隔离级别高于或等于读写已提交,将符合该读取条件且提交时间戳小于该任一次读操作的快照的最大版本确定为该目标数据项;

[0666] 响应于数据库系统的隔离级别低于读写已提交,若符合该读取条件的数据项中存在未提交版本,将最大的未提交版本确定为该目标数据项。

[0667] 在一种可能实施方式中,在分布式数据库系统中采用混合逻辑时钟形式的快照,该混合逻辑时钟形式的快照由协调节点设备下发。

[0668] 上述所有可选技术方案,可以采用任意结合形成本公开的可选实施例,在此不再一一赘述。

[0669] 需要说明的是:上述实施例提供的事务处理装置在处理事务时,仅以上述各功能模块的划分进行举例说明,实际应用中,可以根据需要而将上述功能分配由不同的功能模块完成,即将计算机设备的内部结构划分成不同的功能模块,以完成以上描述的全部或者部分功能。另外,上述实施例提供的事务处理装置与事务处理方法实施例属于同一构思,其具体实现过程详见事务处理方法实施例,这里不再赘述。

[0670] 图16是本申请实施例提供的一种计算机设备的结构示意图。该计算机设备1600可因配置或性能不同而产生比较大的差异,可以包括一个或一个以上处理器(Central

Processing Units,CPU) 1601和一个或一个以上的存储器1602,其中,该存储器1602中存储有至少一条计算机程序,该至少一条计算机程序由该处理器1601加载并执行以实现上述各个实施例提供的事务处理方法。当然,该计算机设备1600还可以具有有线或无线网络接口、键盘以及输入输出接口等部件,以便进行输入输出,该计算机设备1600还可以包括其他用于实现设备功能的部件,在此不做赘述。

[0671] 在示例性实施例中,还提供了一种计算机可读存储介质,例如包括至少一条计算机程序的存储器,上述至少一条计算机程序可由终端中的处理器执行以完成上述实施例中事务处理方法。例如,该计算机可读存储介质可以是ROM(Read-Only Memory,只读存储器)、RAM(Random-Access Memory,随机存取存储器)、CD-ROM(Compact Disc Read-Only Memory,只读光盘)、磁带、软盘和光数据存储设备等。

[0672] 在示例性实施例中,还提供了一种计算机程序产品或计算机程序,该计算机程序产品或计算机程序包括至少一条程序代码,该至少一条程序代码存储在计算机可读存储介质中,计算机设备的一个或多个处理器从该计算机可读存储介质中读取该至少一条程序代码,由该一个或多个处理器执行该至少一条程序代码,使得该计算机设备执行上述各个实施例提供的事务处理方法。

[0673] 本领域普通技术人员可以理解实现上述实施例的全部或部分步骤可以通过硬件来完成,也可以通过程序来指令相关的硬件完成,该程序可以存储于一种计算机可读存储介质中,上述提到的存储介质可以是只读存储器,磁盘或光盘等。

[0674] 以上所述仅为本申请的可选实施例,并不用以限制本申请,凡在本申请的精神和原则之内,所作的任何修改、等同替换、改进等,均应包含在本申请的保护范围之内。

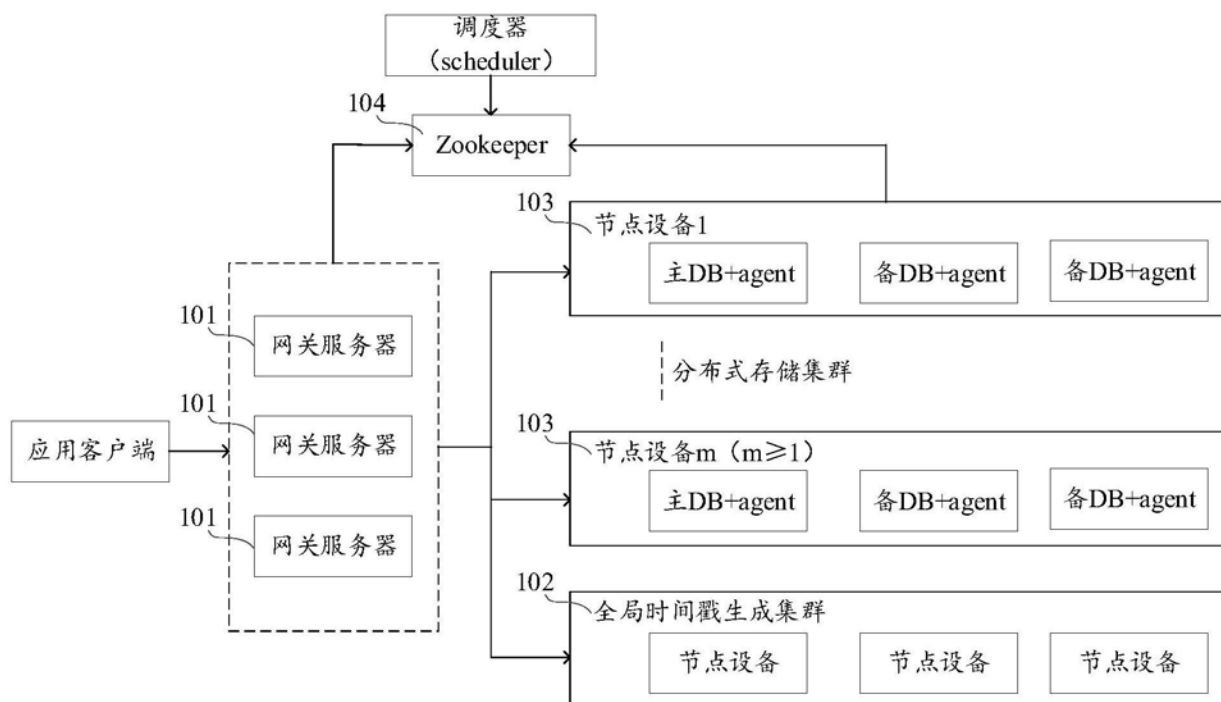


图1

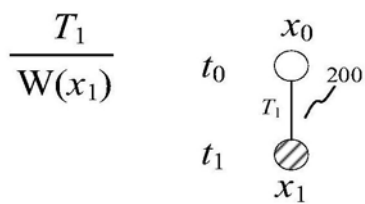


图2

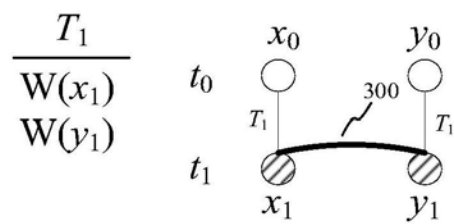


图3

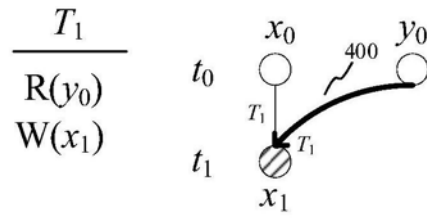


图4

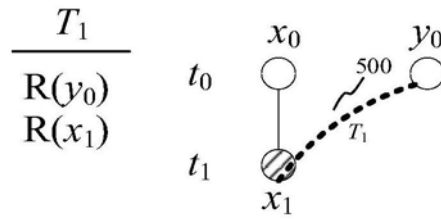


图5

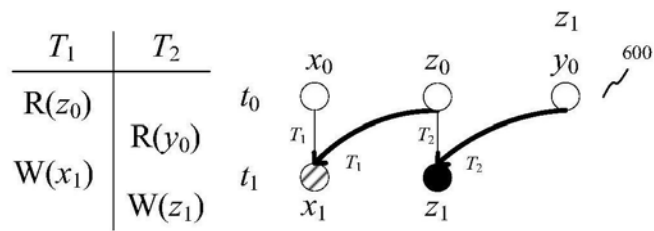


图6

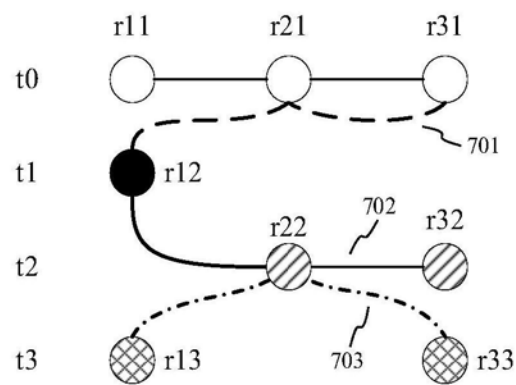


图7

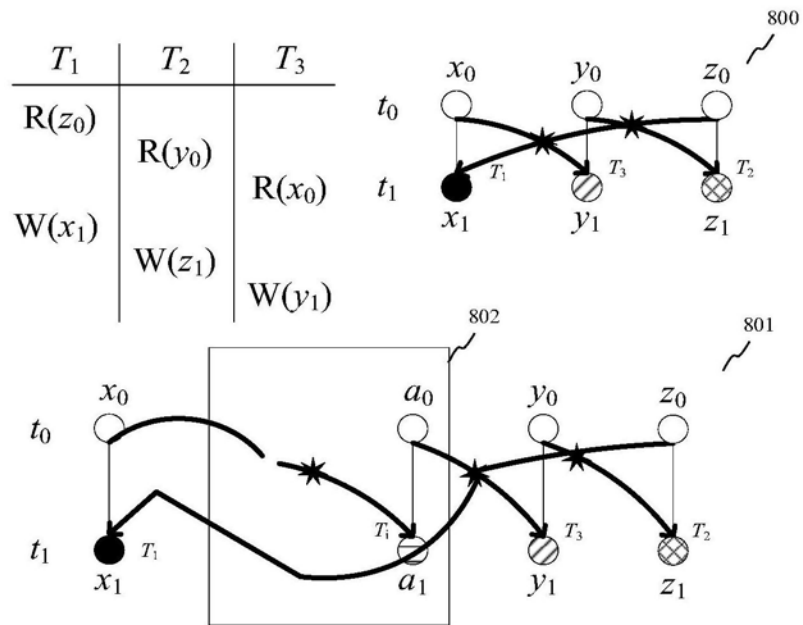


图8

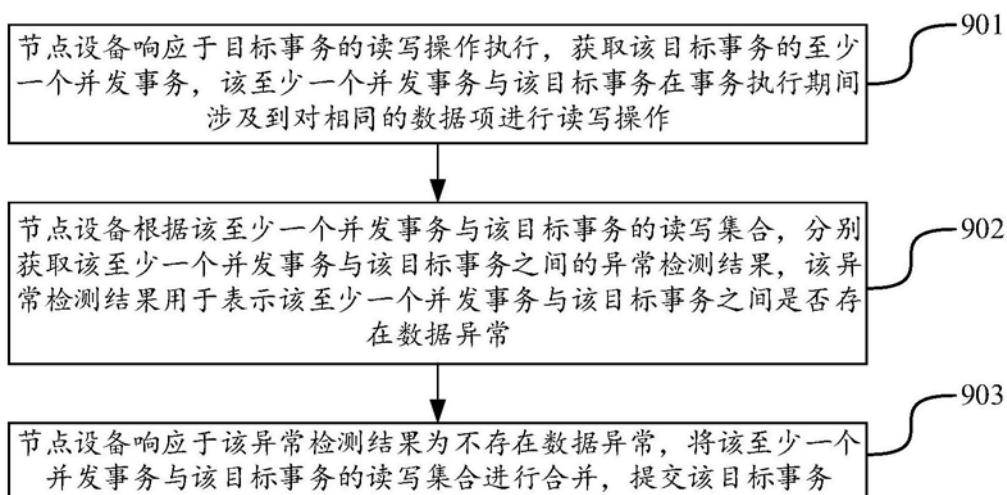


图9

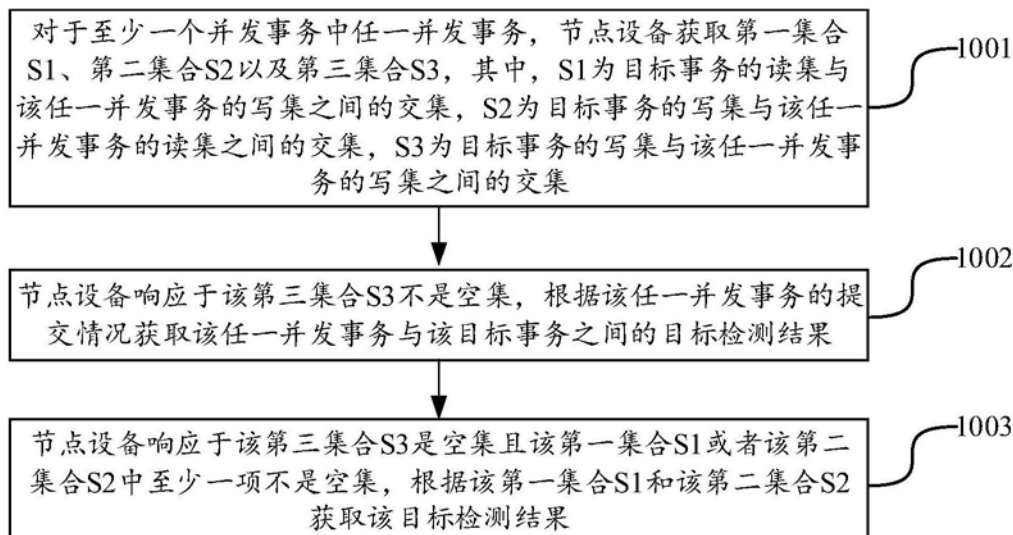


图10

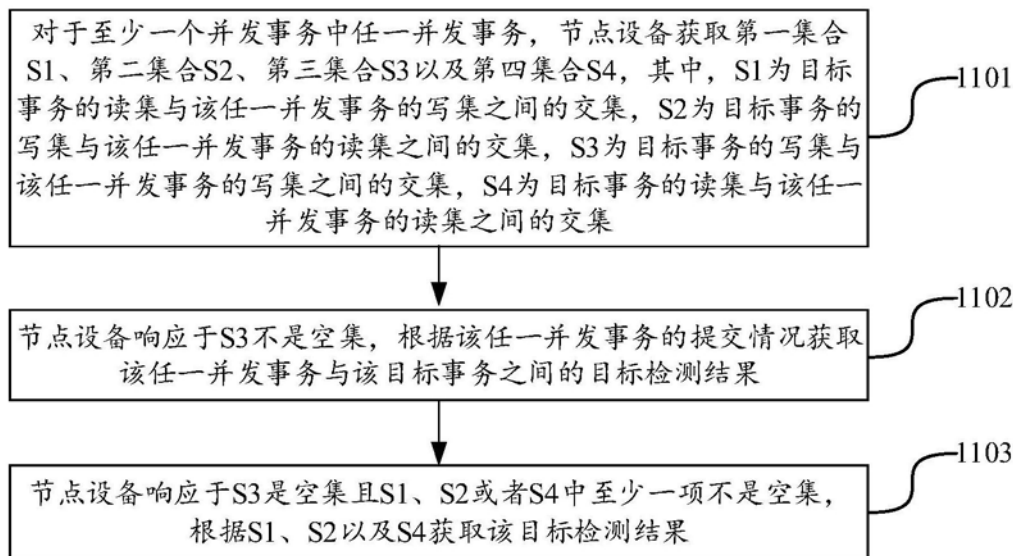


图11

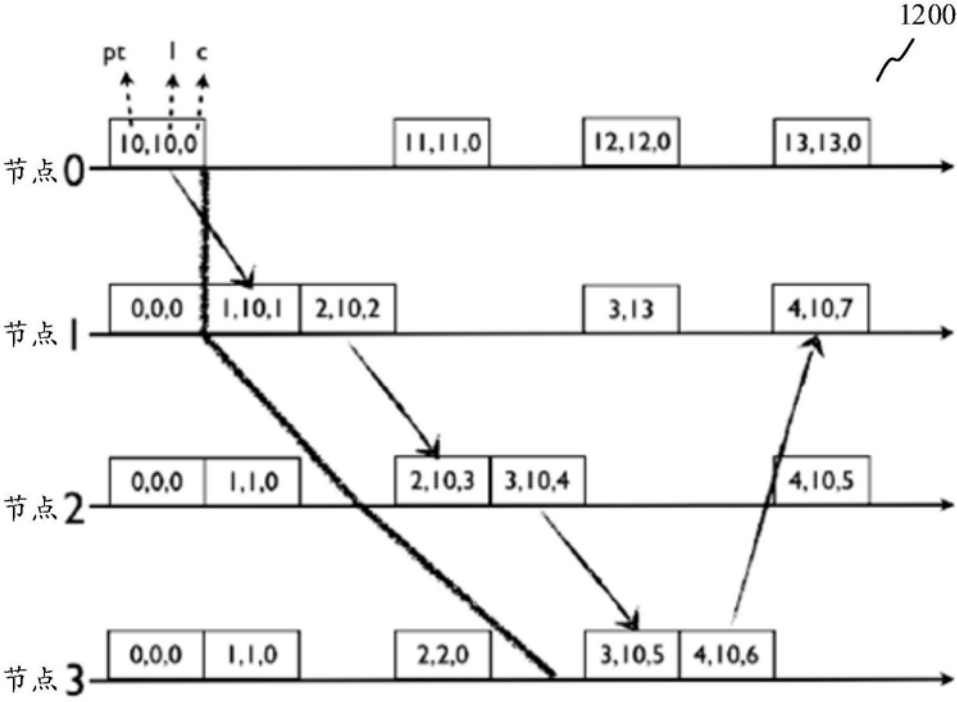


图12

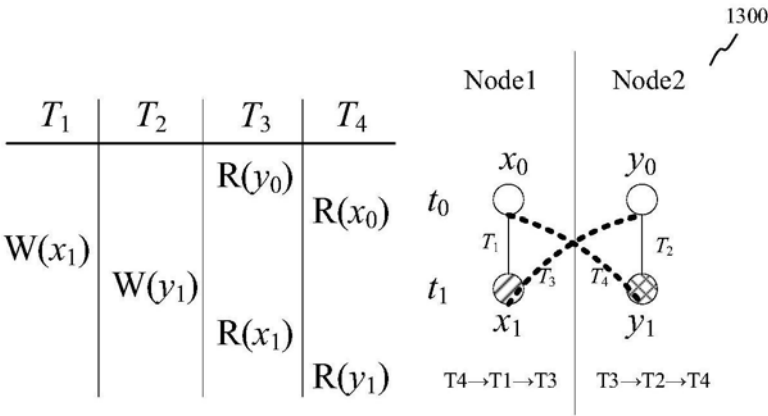


图13

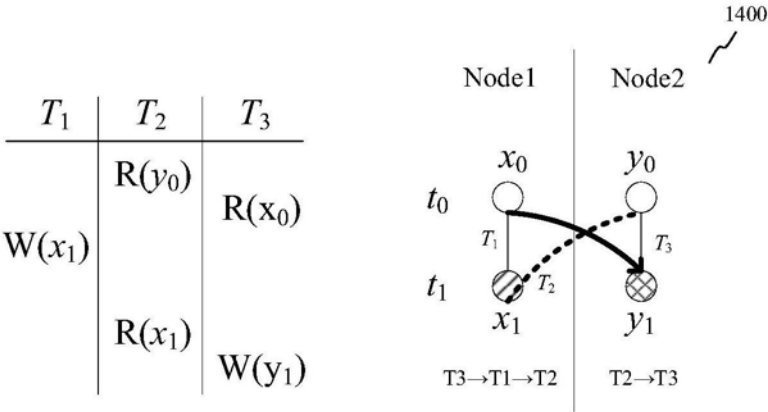


图14

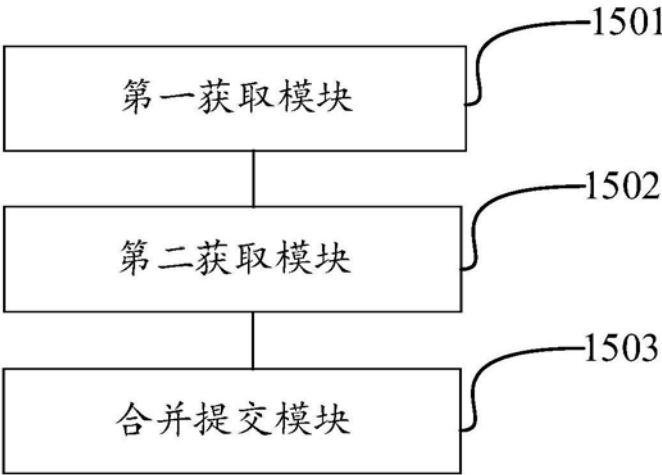


图15

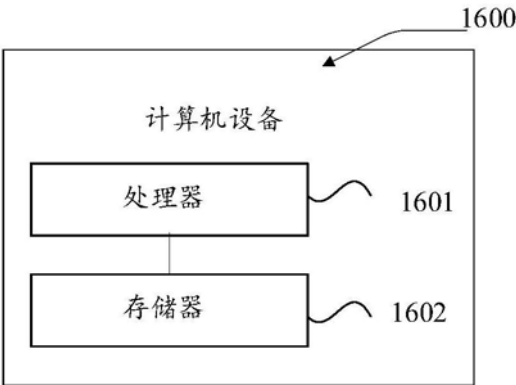


图16