



(21) 申请号 201810764278.4

(22) 申请日 2018.07.12

(65) 同一申请的已公布的文献号

申请公布号 CN 110196760 A

(43) 申请公布日 2019.09.03

(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 康俊彬

(74) 专利代理机构 北京励诚知识产权代理有限公司

公司 11647

专利代理师 贾玉姣

(51) Int.Cl.

G06F 9/46 (2006.01)

G06F 9/52 (2006.01)

(56) 对比文件

US 2018075083 A1,2018.03.15

CN 101706811 A,2010.05.12

CN 102037463 A,2011.04.27

审查员 吕蒙宇

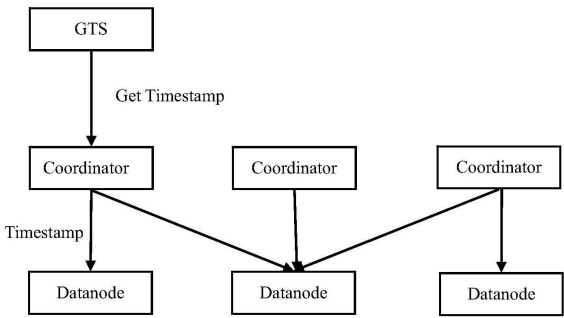
权利要求书2页 说明书14页 附图4页

(54) 发明名称

分布式事务一致性实现方法及装置

(57) 摘要

本发明公开了一种分布式事务一致性实现方法和装置,用于分布式事务系统。包括全局时间戳生成服务器、协调节点和数据节点,分布式事务一致性实现方法包括以下步骤:在协调节点提交事务时向全局时间戳生成服务器申请提交时间戳;提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;在事务开启时向所述全局时间戳生成服务器申请所述事务的开始时间戳以进行事务可见性判断。和在事务开启时向全局时间戳生成服务器申请事务的开始时间戳。本发明实施方式的分布式事务一致性实现方法和装置,通过全局时间戳生成服务器给分布式数据库的事务分配时间戳,保证了事务在各个单节点内部以及多个节点之间的事务的一致性和事务隔离性。



1. 一种分布式事务一致性实现方法,用于分布式事务系统,其特征在于,所述分布式事务系统包括全局时间戳生成服务器、协调节点和数据节点,所述协调节点用于将事务分布到数据所在的相关数据节点执行,所述全局时间戳生成服务器采用可扩展读写锁机制,生成原子递增的全局时间戳,所述分布式事务一致性实现方法包括以下步骤:

在所述协调节点提交事务时向所述全局时间戳生成服务器申请提交时间戳;

将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;和

在事务开启时向所述全局时间戳生成服务器申请所述事务的开始时间戳以进行所述事务可见性判断;

所述协调节点或所述数据节点在任意连续两次向所述全局时间戳生成服务器申请时间戳时,在先申请的所述时间戳小于在后申请的所述时间戳;

所述分布式事务系统包括第一事务和第二事务,所述第一事务执行分布于多个所述数据节点,所述第二事务与所述第一事务并行,将两阶段提交协议中的投票阶段作为所述多个数据节点和协调节点的同步点;

在所述第二事务读表数据时,进行所述事务可见性判断,包括:

若所述第一事务仍处于所述投票阶段未投票的状态或者在快照中没有查到事务状态,则所述第一事务的修改对所述第二事务不可见;

若所述第一事务的所述投票阶段已投票成功的状态,并且所述第一事务的投票时间戳大于或等于所述第二事务的开始时间戳,则所述第一事务的修改对所述第二事务不可见;

若所述第一事务的第二阶段提交完成,则将所述第一事务的提交时间戳和所述第二事务的开始时间戳进行比较,当且仅当所述第一事务的提交时间戳小于所述第二事务的开始时间戳,所述第一事务的修改对所述第二事务可见。

2. 如权利要求1所述的分布式事务一致性实现方法,其特征在于,所述将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中的步骤包括:

在本地事务日志和所述时间戳存储中存储所述事务的提交时间戳。

3. 如权利要求2所述的分布式事务一致性实现方法,其特征在于,所述在本地事务日志和在所述时间戳存储中存储所述事务的提交时间戳的步骤包括:

在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

4. 如权利要求2所述的分布式事务一致性实现方法,其特征在于,所述在本地事务日志中和在所述时间戳存储中存储所述事务的提交时间戳步骤包括:

在所述分布式系统崩溃后,重做所述本地事务日志恢复所述时间戳存储中的数据。

5. 如权利要求1所述的分布式事务一致性实现方法,其特征在于,所述分布式事务系统采用多版本并发机制,包括多个数据行版本,所述事务在数据节点执行,每个所述数据节点包括多个进程,每个进程记录所述进程在执行所述事务的开始时间戳,所述数据节点包括一全局范围变量,所述全局范围变量记录最大的事务提交时间戳,所述分布式事务一致性实现方法还包括步骤:

根据记录的所述提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

6. 一种分布式事务一致性实现装置,用于分布式事务系统,其特征在于,所述分布式事务系统包括全局时间戳生成服务器、协调节点和数据节点,所述协调节点用于将事务分布到数据所在的相关数据节点执行,所述全局时间戳生成服务器采用可扩展读写锁机制,生

成原子递增的全局时间戳,所述分布式事务一致性实现装置包括:

申请模块,用于在所述协调节点提交事务时向所述全局时间戳生成服务器申请提交时间戳;和

写入模块,用于将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;

所述申请模块还用于在事务开启时向所述全局时间戳生成服务器申请所述事务的开始时间戳以进行所述事务可见性判断;

所述协调节点或所述数据节点在任意连续两次向所述全局时间戳生成服务器申请时间戳时,在先申请的所述时间戳小于在后申请的所述时间戳;

所述分布式事务系统包括第一事务和第二事务,所述第一事务执行分布于多个所述数据节点,所述第二事务与所述第一事务并行,将两阶段提交协议中的投票阶段作为所述多个数据节点和协调节点的同步点;

所述申请模块还用于在所述第二事务读表数据时,进行所述事务可见性判断,包括:

若所述第一事务仍处于所述投票阶段未投票的状态或者在快照中没有查到事务状态,则所述第一事务的修改对所述第二事务不可见;

若所述第一事务的所述投票阶段已投票成功的状态,并且所述第一事务的投票时间戳大于或等于所述第二事务的开始时间戳,则所述第一事务的修改对所述第二事务不可见;

若所述第一事务的第二阶段提交完成,则将所述第一事务的提交时间戳和所述第二事务的开始时间戳进行比较,当且仅当所述第一事务的提交时间戳小于所述第二事务的开始时间戳,所述第一事务的修改对所述第二事务可见。

7.如权利要求6所述的分布式事务一致性实现装置,其特征在于,所述写入模块用于在本地事务日志和所述时间戳存储中存储所述事务的提交时间戳。

8.如权利要求7所述的分布式事务一致性实现装置,其特征在于,所述写入模块还用于在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

9.如权利要求6所述的分布式事务一致性实现装置,其特征在于,所述分布式事务系统采用多版本并发机制,包括多个数据行版本,所述事务在数据节点执行,每个所述数据节点包括多个进程,每个进程记录所述进程在执行所述事务的开始时间戳,所述数据节点包括一全局范围变量,所述全局范围变量记录最大的事务提交时间戳,所述分布式事务一致性实现装置还包括:

回收模块,用于根据记录的所述提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

分布式事务一致性实现方法及装置

技术领域

[0001] 本发明涉及计算机技术领域,特别涉及一种分布式事务一致性实现方法及装置。

背景技术

[0002] 在分布式数据库中,事务和查询会分布到多个节点上执行。因此,在设计分布式一致性分布式事务算法和协议时,既要保证各个单节点内部的数据一致性和事务隔离性以外,还要保证多个节点之间的数据一致性和事务隔离性,如何实现节点内以及节点间一致性和隔离性成为亟待解决的问题。

发明内容

[0003] 本发明提供了一种分布式事务一致性实现方法及装置。

[0004] 本发明的实施方式的分布式事务一致性实现方法,用于分布式事务系统,所述分布式事务系统包括全局时间戳生成服务器、协调节点和数据节点,所述协调节点用于将事务分布到数据所在的相关数据节点执行,所述全局时间戳生成服务器采用可扩展读写锁机制,生成原子递增的全局时间戳,所述分布式事务一致性实现方法包括以下步骤:

[0005] 在所述协调节点提交事务时向所述全局时间戳生成服务器申请提交时间戳;

[0006] 将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;
和

[0007] 在事务开启时向所述全局时间戳生成服务器申请所述事务的开始时间戳以进行所述事务可见性判断。

[0008] 在某些实施方式中,所述协调节点或所述数据节点在任意连续两次向所述全局时间戳生成服务器申请时间戳时,在先申请的所述时间戳小于在后申请的所述时间戳。

[0009] 在某些实施方式中,所述分布式事务系统包括两个事务,在所述第二事务读表数据时,当且仅当第一事务的提交时间戳小于第二事务的开始时间戳,所述第一事务的修改对所述第二事务可见。

[0010] 在某些实施方式中,所述第一事务执行分布于多个所述数据节点,所述第二事务与所述第一事务并行,将两阶段提交协议中的投票阶段作为所述多个数据节点和协调节点的同步点。

[0011] 在某些实施方式中,所述将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中的步骤包括:

[0012] 在本地事务日志中所述时间戳存储中存储所述事务的提交时间戳。

[0013] 在某些实施方式中,所述在本地事务日志中和在所述时间戳存储中存储所述事务的提交时间戳的步骤包括:

[0014] 在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

[0015] 在某些实施方式中,所述在本地事务日志中和在所述时间戳存储中存储所述事务的提交时间戳步骤包括:

[0016] 在所述分布式系统崩溃后,重做所述本地事务日志恢复所述时间戳存储中的数据。

[0017] 在某些实施方式中,所述分布式事务系统采用多版本并发机制,包括多个数据行版本,所述事务在数据节点执行,每个所述数据节点包括多个进程,每个进程记录所述进程在执行所述事务的开始时间戳,所述数据节点包括一全局范围变量,所述全局范围变量记录最大的事务提交时间戳,所述分布式事务一致性实现方法还包括步骤:

[0018] 根据记录的所述提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

[0019] 本发明实施方式的分布式事务一致性实现装置,用于分布式事务系统,所述分布式事务系统包括全局时间戳生成服务器、协调节点和数据节点,所述协调节点用于将事务分布到数据所在的相关数据节点执行,所述全局时间戳生成服务器采用可扩展读写锁机制,生成原子递增的全局时间戳,所述分布式事务一致性实现装置包括:

[0020] 申请模块,用于在所述协调节点提交事务时向所述全局时间戳生成服务器申请提交时间戳;和

[0021] 写入模块,用于将所述提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;

[0022] 所述申请模块还用于在事务开启时向所述全局时间戳生成服务器申请所述事务的开始时间戳以进行所述事务可见性判断。

[0023] 在某些实施方式中,所述协调节点或所述数据节点在任意连续两次向所述全局时间戳生成服务器申请时间戳时,在先申请的所述时间戳小于在后申请的所述时间戳。

[0024] 在某些实施方式中,所述分布式事务系统包括两个事务,在所述第二事务读表数据时,当且仅当第一事务的提交时间戳小于第二事务的开始时间戳,所述第一事务的修改对所述第二事务可见。

[0025] 在某些实施方式中,所述第一事务执行分布于多个所述数据节点,所述第二事务与所述第一事务并行,将两阶段提交协议中的投票阶段作为所述多个数据节点和协调节点的同步点。

[0026] 在某些实施方式中,所述写入模块用于在本地事务日志中所述时间戳存储中存储所述事务的提交时间戳。

[0027] 在某些实施方式中,所述写入模块还用于在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

[0028] 在某些实施方式中,所述写入模块还用于在所述分布式系统崩溃后重做所述本地事务日志恢复所述时间戳存储中的数据。

[0029] 在某些实施方式中,所述分布式事务系统采用多版本并发机制,包括多个数据行版本,所述事务在数据节点执行,每个所述数据节点包括多个进程,每个进程记录所述进程在执行所述事务的开始时间戳,所述数据节点包括一全局范围变量,所述全局范围变量记录最大的事务提交时间戳,所述分布式事务一致性实现装置还包括:

[0030] 回收模块,用于根据记录的所述提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

[0031] 本发明实施方式的分布式事务一致性实现方法及装置,通过全局时间戳生成服务器给分布式数据库的事务分配时间戳,保证了事务在各个单节点内部以及多个节点之间的

事务的一致性和事务隔离性。

[0032] 本发明的附加方面和优点将在下面的描述中部分给出,部分将从下面的描述中变得明显,或通过本发明的实践了解到。

附图说明

[0033] 本发明的上述和/或附加的方面和优点从结合下面附图对实施方式的描述中将变得明显和容易理解,其中:

[0034] 图1是本发明实施方式的分布式事务一致性实现方法的流程示意图;

[0035] 图2是本发明实施方式的分布式事务一致性实现装置的模块示意图;

[0036] 图3是本发明实施方式的分布式事务系统架构示意图;

[0037] 图4是本发明实施方式的分布式事务一致性实现方法的另执行时序图;

[0038] 图5是本发明实施方式的分布式事务一致性实现方法的TBase多版本并发控制机制示意图;

[0039] 图6是本发明实施方式的分布式事务一致性实现方法的提交时间戳存储系统示意图;

[0040] 图7是本发明实施方式的分布式事务一致性实现方法的空间回收示意图;

[0041] 图8是本发明实施方式的分布式事务一致性实现方法的GTS架构示意图;

[0042] 图9是本发明实施方式的分布式事务一致性实现方法的一个事务处理能力示意图;

[0043] 图10本发明实施方式的分布式事务一致性实现方法的另一个事务处理能力示意图。

具体实施方式

[0044] 下面详细描述本发明的实施方式,所述实施方式的示例在附图中示出,其中自始至终相同或类似的标号表示相同或类似的元件或具有相同或类似功能的元件。下面通过参考附图描述的实施方式是示例性的,仅用于解释本发明,而不能理解为对本发明的限制。

[0045] 在本发明的描述中,需要理解的是,术语“中心”、“纵向”、“横向”、“长度”、“宽度”、“厚度”、“上”、“下”、“前”、“后”、“左”、“右”、“竖直”、“水平”、“顶”、“底”、“内”、“外”、“顺时针”、“逆时针”等指示的方位或位置关系为基于附图所示的方位或位置关系,仅是为了便于描述本发明和简化描述,而不是指示或暗示所指的装置或元件必须具有特定的方位、以特定的方位构造和操作,因此不能理解为对本发明的限制。此外,术语“第一”、“第二”仅用于描述目的,而不能理解为指示或暗示相对重要性或者隐含指明所指示的技术特征的数量。由此,限定有“第一”、“第二”的特征可以明示或者隐含地包括一个或者更多个所述特征。在本发明的描述中,“多个”的含义是两个或两个以上,除非另有明确具体的限定。

[0046] 在本发明的描述中,需要说明的是,除非另有明确的规定和限定,术语“安装”、“相连”、“连接”应做广义理解,例如,可以是固定连接,也可以是可拆卸连接,或一体地连接。可以是机械连接,也可以是电连接。可以是直接相连,也可以通过中间媒介间接相连,可以是两个元件内部的连通或两个元件的相互作用关系。对于本领域的普通技术人员而言,可以根据具体情况理解上述术语在本发明中的具体含义。

[0047] 请参阅图1,本发明实施方式提供一种分布式事务一致性实现方法,用于分布式事务系统,分布式事务系统包括全局时间戳生成服务器(Global Timestamp Server,GTS)、协调节点(Coordinator,CN)和数据节点(Datanode,DN),协调节点用于将事务分布到数据所在的相关数据节点执行,全局时间戳生成服务器采用可扩展读写锁机制,生成原子递增的全局时间戳,分布式事务一致性实现方法包括以下步骤:

[0048] S10:在协调节点提交事务时向全局时间戳生成服务器申请提交时间戳;

[0049] S20:将提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中;
和

[0050] S30:在事务开启时向全局时间戳生成服务器申请事务的开始时间戳以进行事务可见性判断。

[0051] 请参阅图2,本发明实施方式提供一种用于分布式事务系统的分布式事务一致性实现装置100,可用于实现上述分布式事务一致性实现方法。分布式事务系统包括全局时间戳生成服务器、协调节点和数据节点。协调节点用于将事务分布到数据所在的相关数据节点执行,全局时间戳生成服务器采用可扩展读写锁机制。分布式事务一致性实现装置100包括申请模块10和写入模块20。步骤S10和步骤S30可以由申请模块10实现,步骤S20可以由写入模块20实现。或者说,申请模块10用于在协调节点提交事务时向全局时间戳生成服务器申请提交时间戳。写入模块20用于将提交时间戳写入参与的协调节点和数据节点的事务日志和时间戳存储中。申请模块10还用于在事务开启时向全局时间戳生成服务器申请事务的开始时间戳以进行事务可见性判断。

[0052] 事务(Transaction)是指作为单个逻辑工作单元执行的一系列操作,要么完全地执行,要么完全不执行。事务处理可以确保除非事务性单元内的所有操作都成功完成,否则不会永久更新面向数据的资源。通过将一组相关操作组合为一个要么全部成功要么全部失败的单元,可以简化错误恢复并使应用程序更加可靠。一个逻辑工作单元要成为事务,必须满足所谓的ACID(原子性、一致性、隔离性和持久性)属性。

[0053] 分布式事务(Distributed Transaction)是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上。

[0054] 在第一类相关技术中,分布式数据库通过分库分表来增加数据库的存储和计算能力,但是不提供一致性分布式事务支持(提供全局一致的隔离性)。基于PostgreSQL的分布式数据库PGXC和PGXL通过全局快照隔离(snapshot isolation)机制来提供全局一致的分布式事务。具体的,PGXC和PGXL通过一个全局节点GTM(Global Transaction Management)管理全局事务信息,在每个节点上通过snapshot isolation机制来做数据可见性判断。

[0055] PGXC和PGXL由GTM,Coordinator(CN)和Datanode(DN)组成。表数据实际存储在DN上,CN负责将查询和事务分布到数据所在的DN上执行,同时在启动事务时向GTM申请全局xid,在事务提交或终止(abort)时向GTM汇报。GTM维护了全局的活动事务列表,在每个CN启动事务时,GTM分配一个全局xid,同时将该xid放入全局活动事务列表中,在CN提交或终止事务时,GTM会把相应的事务从活动事务列表中移除。

[0056] CN在事务(repeatable read isolation)或语句(read committed)开启时,会向GTM申请全局快照做数据可见性判断(从而保证隔离性和一致性),GTM遍历活动事务列表,生成正在运行事务全局快照。CN再将全局快照发送给执行语句(事务)参与的DN节点,DN在

扫描表数据行 (tuple) 时, 根据全局快照来判断一个 tuple 是否可见。具体的, PostgreSQL 采用 MVCC 机制做并发访问控制, 一个 tuple 会有多个版本 (形成版本链), 每个版本用 xmin 代表插入这个版本的事务 xid, 用 xmax 代表删除或更新这个 tuple 的事务 xid。在用全局快照做可见性判断时, 如果 xmin 已经提交, 并且不在快照中 (说明在本查询或事务开启前已经提交), 同时 xmax 无效或者在快照中, 则这个 tuple 版本对本查询可见。

[0057] 然而, 全局快照隔离技术开销极大, 同时很容易导致 GTM 单点瓶颈。CN 在向 GTM 获取快照时, GTM 需要加锁遍历活动事务列表。分布式数据库中并发事务越大, 活动事务列表就越长, 这增加了两方面的开销: (1) GTM 计算资源开销: GTM 加锁遍历所有活动事务计算开销; 锁冲突造成的等待开销; (2) 网络资源开销, 快照大小和活动事务列表大小成正比, CN 频繁获取快照以及传送快照到 DN 会造成集群网络资源的极大消耗。

[0058] 具体地, 全局快照的开销如下: 假设集群有 N 个并发事务, GTM 每次计算一个全局快照的 CPU 开销为 N, 每个 snapshot 占用的网络带宽最小为 $N*4$ 字节 (一个 xid 四字节)。

[0059] 对于 read-committed isolation, 事务中每个语句需要获取一个快照, 假设每个事务平均有 m 条语句, 在 N 个并发事务场景下, 则每次快照的 GTM 的计算开销为 $N*N*m$, GTM 网络带宽为 $N*N*m*4$ 字节。

[0060] 在第二类相关技术中, 通过精确的 GPS 和原子时钟在集群范围内提供时钟偏移特别小的分布式时钟服务, 通过 true time API 获取精确的物理时钟, 实现了外部一致性的分布式事务。但是该类方法需要昂贵并不普遍的硬件配置 (GPS 和原子时钟), 对于由通用的机器组成的数据中心并不适用。

[0061] 此外, 在第三类相关技术中, 在多节点同步一致性状态采用的是在第一阶段提交的时候, 将事务每个修改的数据单元写入锁, 再在第二阶段提交的时候, 将锁替换成写入成功的记录。当读事务打算读取一个被锁的数据单元的时候需要等待写入事务锁的释放。该方法通过在第一阶段写入锁的方法来同步读写事务的一致性, 保证所有开始时间戳大于写事务提交时间戳的读事务都可以看到该写事务的修改内容。

[0062] 然而, 该类方法在提交事务时开销比较大, 第一阶段提交需要遍历所有修改数据单元, 并写入锁信息到数据单元, 第二阶段提交又需要重新再遍历所有修改数据单元, 将锁释放, 写入提交时间戳等信息。对于修改了大量数据单元的事务而言, 事务提交的同步开销比较大, 开销和修改数据量成正比。

[0063] 请参阅图 3, 本发明实施方式的分布式事务一致性实现方法, 应用于基于 PostgreSQL 的 TBase 分布式数据库中, 为 TBase 分布式数据库提供一致性的分布式事务支持。

[0064] 具体地, 在事务提交时向 GTS 申请提交时间戳 (commit timestamp), 并写入各参与节点的事务日志和时间戳日志存储中。每个事务 (repeatable read isolation) 开启时向 GTS 申请该事务的开始时间戳 (start timestamp)。

[0065] 本发明实施方式的分布式事务一致性实现方法可以支持 repeatable-read 的事务隔离级别。Repeatable-read 隔离级别表示的是, 一个事务 T1 的修改对另一个事务 T2 可见的前提是, T1 在 T2 事务开启前已经提交。

[0066] 当然, 本发明实施方式的分布式事务一致性实现方法还可以支持 read-committed 的事务隔离级别, read-committed 隔离级别表示的是, 一个事务 T1 的修改对另一个事务 T2

中的语句可见的前提是, T1在T2中的语句开始之前已经提交, read-committed的事务隔离与repeatable-read的事务隔离相类似, 此处不在赘述。

[0067] 在某些实施方式中, GTS分配时间戳(timestamp) 满足原子递增的特性: CN或者DN在任意连续两次向GTS申请分配时间戳, 其中连续的意思是第一次申请到返回时间戳后再立即发起第二次申请, 两次申请可以是同一CN或DN发起, 也可以是任意不同CN或DN发起的, GTS保证第一次申请到的时间戳一定小于第二次申请的时间戳。

[0068] 在这样的实施方式中, 通过分配到的时间戳来保证事务隔离性。对于任意两个事务T1和T2, 事务T2在读表数据时, T1的修改对T2是否可见的条件是, 当且仅当T1提交时间戳小于T2开始时间戳, 否则不可见。

[0069] 但是从CN上申请到时间戳, 到事务到达每个DN上执行, 有一定的延迟和乱序。需要保证T2在每个DN上都能一致的看到T1的修改, 即对与任意两个T1和T2, 如果T2的开始时间戳大于T1的提交时间戳, 保证T2在所有节点一定能看到T1的修改内容, 同时, 如果T2的开始时间戳小于或等于T1的提交时间戳, 保证T2在所有节点一定看不到T1的修改内容。

[0070] 请参阅图4, 在这样的实施方式中, 利用两阶段提交协议的投票阶段(prepare) 作为同步点, 结合GTS的原子递增性来保证T2在每个DN上都能一致的看到T1的修改。具体地, 假定任意T1和T2两个事务, T1为分布式事务, 执行分布在多个节点, 并发事务T2会访问到T1的修改数据, 假定T1在CN1上开启, T2在CN2上开启。需要说明的是, CN1和CN2可以是相同的CN, 也可以是不同的CN。T1事务在CN上开启, 向GTS申请开始时间戳(start_ts), 并发送到执行的DN上; 用户提交事务T1时, CN开启两阶段提交。Prepare阶段, CN向GTS申请prepare时间戳(prepare_ts), 将时间戳和prepare请求一同发送给所有参与的DN, DN将事务涉及的修改写日志, 返回CN结果。在收到所有DN的返回结果后, 如果均prepare成功, CN进入Commit阶段, 向GTS申请提交时间戳(commit_ts), 并将提交时间戳和commit请求一同发送给所有参与的DN, DN将事务提交, 并将提交时间戳写入时间戳存储中以及事务重做(redo) 日志中; 如果prepare失败, CN向所有DN发起回滚事务请求。T1 on CN1时间线表示的是T1在CN1上执行的时间线, CN1将T1的执行(start, prepare和commit) 发送到参与的DN上(图中只示出了DN1和DN2)。

[0071] 并行事务T2在启动时, 从GTS获取开始时间戳, 并将时间戳发送给参与DN节点。

[0072] T2在DN上读取T1的修改数据时, 根据T2的开始时间戳和T1的状态来决定T1的修改数据是否对T2可见。T2 on CN2的时间线代表的是, 假设CN2在不同的时间点开启T2事务(不同的startTS, 三种情况), 任意DN上(图中示例为DN2) 上T1的修改(数据S2) 对T2是否可见。

[0073] TBase在每个节点都会用一个全局活动事务列表记录当前正在运行的事务的状态。本发明实施方式的分布式事务一致性实现方法给活动事务列表中的事务项增加一个是否已经prepare的状态和一个存储prepare时间戳字段。这个状态会在该事务已经prepare成功后立即设置(在向CN返回prepare结果之前), 同时用CN传过来的prepare时间戳存储到事务项中。

[0074] T2在DN2上执行开始时, 会扫描所有当前正在活动事务列表, 记录所有事务的状态, 保存为快照, 在下面算法做可见性判断时, 会去快照中查找相应事务的状态以及prepare时间戳。

[0075] 当T2在DN2上读取到T1的修改数据时, 有三种情况, 分别进行讨论:

[0076] I. 如果T1仍处于未prepare的状态或者是在快照中没有查到事务状态,则T1的修改对T2不可见;

[0077] II. 如果T1是已经prepare的状态,并且T1.prepare_ts大于或等于T2.start_ts,则也T1的修改对T2不可见;否则T2需要等待T1第二阶段提交成功,再进行可见性检查;

[0078] III. 如果T1已经第二阶段提交完成,则根据T1的xid从时间戳日志存储中读取T1的提交时间戳,并将T1的提交时间戳和T2的开始时间戳进行比较,如果T2.start_ts大于T1.commit_ts,则T2的修改对T1可见,否则不可见。

[0079] 如果T1是已经prepare状态,T2需要等待T1第二阶段提交完成。这个是通过加事务锁来实现的。在TBase中每个事务开启的时候均会创建一个事务xid标识的锁,并加锁,在事务结束时,会将锁释放(唤醒所有等待事务)。

[0080] 对于I,由于T2在扫描到T1修改的数据时,T1仍未过prepare阶段,则说明T2在DN2上开始执行时,T1还未向CN申请提交时间戳(CN需要在收到所有DN的prepare返回结果后,才会向GTS申请提交时间戳),而T2已经在CN上申请到了开始时间戳才被调度到DN2上开始执行的。如果是在快照中未查到T1的状态,说明T2在DN2上扫描活动事务列表时T1还未开始执行。在这种情况下,由GTS的原子递增性,可以推导出T2的开始时间戳一定小于T1的提交时间戳,因此T1的修改对T2不可见。

[0081] 对于II,由于T1在DN2上已经返回prepare结果给CN1了,如果T2的开始时间戳小于或等于T1的prepare时间戳,则T2对T1不可见,因为T1的提交时间戳在prepare时间戳之后申请的,T1.commit_ts>T1.prepare_ts>=T2.start_ts。如果T1的prepare时间戳小于T2的开始时间戳,则T2需要等待T1在DN上第二阶段提交结束,之后根据DN上收到的T1的提交时间戳和T2的开始时间戳做比较来判断T2的修改对T1是否可见。

[0082] 对于III,直接根据T1的提交时间戳和T2的开始时间戳判断T2对T1是否可见。

[0083] 因此,本发明实施方式的分布式事务一致性实现方法能够保证事务隔离性和一致性。

[0084] 对于T1只有一个参与DN的情况,也即是说。不需要两阶段提交,事务可见性判断方法和上述一致,只有事务提交的流程不一样。

[0085] 具体地,CN在提交事务时,直接向DN发送提交请求,DN收到提交请求时,将事务T1的状态修改为已经prepare状态,prepare时间戳设置为一个预定最小值1(小于所有事务的开始时间戳),DN向GTS申请提交时间戳,提交事务,将提交时间戳写入事务日志和时间戳存储中。

[0086] 对于TBase支持的用户显示两阶段事务,本发明实施方式的分布式事务一致性实现方法将两阶段事务变成三阶段事务,可见性判断算法和两阶段事务一致,只有事务提交流程不一样:用户发起两阶段提交的prepare请求后,CN不申请prepare时间戳,直接向DN发送prepare请求,DN完成prepare请求后,不标志已经prepare的状态,用户发起commit请求后,CN向GTS申请prepare时间戳,同时向参与DN发送precommit请求,并将prepare时间戳发送至各个参与DN,DN上该事务的状态修改为已经prepare,CN向GTS申请提交时间戳,向所有参与DN发送commit请求,并将提交时间戳发送至各个参与DN。

[0087] 请参阅图5,TBase采用PostgreSQL的MVCC多版本并发控制来提供高并发数据访问(读写不互斥)。数据库表中的行由tuple存储,tuple中的元数据中记录了xmin和xmax,xmin

表示的是插入(生成)这个版本的事务xid,xmax表示删除(更新)这个版本的事务xid.TBase在插入一个行数据到表中时,tuple的xmin记录了插入事务的xid,同时xmax设置为invalid状态。删除一个表的行时,不是直接删除tuple,而是将tuple的xmax设置为删除事务的xid.TBase在更新一个表的行时,不是直接更新这个行的tuple,而是插入一个新的版本tuple,并将原来的tuple版本中的xmax字段和新插入tuple版本的xmin字段设置为更新事务的xid。

[0088] 这样,MVCC机制允许数据库对同一行的读写事务互相不阻塞:读查询可以读到最大的对它可见的版本,写事务可以直接在该行的tuple多版本链后面增加一个新的版本。同时,TBase通过空间回收(vacuum)的流程来回收tuple多版本链中失效的版本。

[0089] MVCC机制下,会导致表中的每个行可能有多个tuple版本,组成tuple链,需要进行空间回收,将失效的不会再被访问的tuple版本占的空间释放出来。在MVCC中,一个tuple可以被回收的前提条件是它处于dead状态,即它已经失效(被更新并且下一个版本已经提交),并且当前以及之后没有其它事务会访问它,即对正在活跃和后续到来的事务不可见,或者时候它的下一版本对所有当前事务和后续到来的事务均可见.TBase会在扫描tuple页时对页内的tuple链进行热回收,将页面进行压缩,以及会在空间回收开启时进行回收,也即是说,就是将一个页内的一个tuple的多个版本,将不会被访问到的(确定不会有事务再访问了),从链表中去掉,减少查找开销。其中,热回收是在事务扫描数据时会开启,冷回收是在一定时刻(用户开启或者周期性开启)开启,热回收减少了每个tuple的版本链的长度,冷回收会将失效tuple占的空间回收回来(具体的将一个页里面有效的数据拷贝到新的页,再将原来的页标记为空的状态)

[0090] 如上所述,事务T在扫描一个tuple版本时,该版本是否对T可见的条件是,这个tuple版本的xmin对事务T可见,而xmax对事务T不可见或者xmax是无效(invalid)状态。

[0091] 根据上述可见性算法,对于一个tuple,根据如下流程来具体判断一个tuple是否可见:读取tuple的xmin字段,读取xmin的提交时间戳,如果提交时间戳存在,则直接根据事务T.start_ts和xmin.commit_ts做比较,根据上述方法来判定xmin是否可见。如果xmin的提交时间戳不存在,则判断xmin事务是否已经过了prepare阶段,如果没有过,则直接判断xmin不可见。如果过了Prepare阶段,则对比xmin.prepare_ts和T.start_ts,来判定xmin是否可见。如果xmin.prepare_ts $\geq$ T.start_ts,则tuple对事务T不可见,结束。如果xmin.prepare_ts $<$ T.start_ts,则事务T等待事务xmin结束。等待完成后,xmin如果已经abort,则不可见。否则根据xmin.commit_ts和T.start_ts来判断xmin可见性。如果xmin可见,则继续上述同样的流程判断xmax是否可见。如果xmax为invalid或者根据上述同样流程判断xmax不可见,则这个tuple版本对事务T可见。

[0092] 如此,可以保证分布式数据库内部一致性。

[0093] 本发明实施方式的分布式事务一致性实现方法同时也可以保证外部一致性。外部一致性也即是对于任意用户客户端来说,如果事务T1已经提交并返回成功,后续发起的任意事务T2均可以看到T1的修改(T1和T2可以是在同一个客户端发起的,也可以是不同客户端)。

[0094] 具体地,假设当T1提交成功并返回给客户端结果时物理世界真实精确时间是t1。在T1返回提交成功后,假设客户端发起T2事务时物理世界真实精确时间是t2,t2 $\geq$ t1。假

设T1的提交时间戳申请到时的物理世界时间是 t_3 ,T2的开始时间戳申请发起的物理世界时间是 t_4 。 t_3 必然小于 t_1 (考虑还有网络延迟), t_4 必然大于 t_2 。因此 $t_3 < t_1 \leq t_2 < t_4$,按照GTS的原子递增性,T2的开始时间戳必然大于T1的提交时间戳。

[0095] 如此,可以保证分布式数据库外部一致性。

[0096] 请参阅图6,在某些实施方式中,步骤S20包括:

[0097] 在本地事务日志和时间戳存储中存储事务的提交时间戳。

[0098] 在某些实施方式中,在本地事务日志和时间戳存储中存储事务的提交时间戳的步骤可以由写入模块20实现,或者说,写入模块20用于在本地事务日志和时间戳存储中存储事务的提交时间戳。

[0099] 在这样的实施方式中,在本地事务日志和时间戳存储中存储事务的提交时间戳的步骤包括:

[0100] 在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

[0101] 在这样的实施方式中,在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问的步骤可以由写入模块20实现,或者说,写入模块20用于在表数据记录头部元数据中建立LRU高速页缓存以加速所述提交时间戳的访问。

[0102] 在这样的实施方式中,在本地事务日志和时间戳存储中存储事务的提交时间戳的步骤包括:

[0103] 在分布式系统崩溃后,重做本地事务日志恢复所述时间戳存储中的数据。

[0104] 在这样的实施方式中,写入模块20还用于在分布式系统崩溃后重做本地事务日志恢复时间戳存储中的数据。

[0105] 具体地,对于分布式数据库的每个节点,在事务提交的时候,需要将从GTS申请到的提交时间戳持久化存储,并使用缓存加速对事务提交时间戳的查找操作。在每个分布式节点通过本地时间戳日志存储(Tlog)来存储事务的提交时间戳。Tlog实现为一个从xid到该事务xid的提交时间戳的一个索引映射存储。具体的,Tlog逻辑上为一个固定长度的数组,数组中每个项(slot)长度固定(假设为W,为8个字节),存储一个时间戳或者为空(0),数组以xid作为下标进行索引。

[0106] TBase中正常事务xid为一个从3开始连续递增的32位整形数字,因此,Tlog的长度为 $(2^{32}-3)*W$ 。在查找某个xid的提交时间戳时,根据偏移 $xid*W$ 找到Tlog中的相应slot,再从slot中读取值,如果为0,则说明该事务xid还未提交或已经abort。每个节点提交事务xid时,从GTS申请到的提交时间戳写入 $xid*W$ 偏移处的Tlog的slot中。为支持Tlog在系统崩溃或者断电的情况下可恢复,在提交事务时,将xid的提交时间戳也写进了事务redo日志中的事务提交记录中。在数据库恢复时,将从redo日志中扫描到的事务提交记录中的提交时间戳写入Tlog中。

[0107] 为了加速查找,Tlog在内存中分配了LRU的页缓存来缓存磁盘上的数据。为了避免LRU缓存全局锁竞争,将Tlog进行均匀分区,每个分区创建一个LRU页缓存,同时建立一个以Tlog物理块号为key的hash表来索引页缓存(xid映射到Tlog在磁盘上的块号,再去查找该块是否被LRU页缓存缓存,如果缓存则查找出对应的缓存位置)。

[0108] 每次可见性判断时都从Tlog中读取提交时间戳开销会比较大。进一步将从Tlog中读取到提交时间戳写入扫描到的tuple的头部元数据,来加速下一次访问同一tuple时做可

见性的性能。具体的,每个Tuple的头部元数据部分预留了一个xmin的提交时间戳字段和xmax的提交时间戳字段。在扫描该Tuple做可见性判断时,首先从tuple中读取xmin和xmax的提交时间戳来做可见性判断,如果tuple中不存在,则会去Tlog中读取xmin或(和)xmax的提交时间戳。如果Tlog中有提交时间戳,则同时将读到的提交时间戳写入tuple头部元数据中。

[0109] 在扫描每个缓存(buffer)上的tuple时,加的是buffer的共享锁,但是会对tuple头部的xmin和xmax的提交时间戳进行并发的读写操作,可能会导致不一致的状态,比如读到部分写入的提交时间戳(一个扫描进程将读出Tlog的提交时间戳写入tuple头部元数据,另一个进程同时从tuple头部读这个时间戳)。为了保证tuple头部提交时间戳的数据一致性,使用了tuple头部元数据的两个标志位来分别标记xmin的提交时间戳的状态和xmax的提交时间戳的状态。当一个进程打算读取一个tuple头部元数据的提交时间戳时,先去检查相应的标志位是否已经置位,如果已经置位,则读取tuple头部元数据中相应的提交时间戳;如果没有置位,则从Tlog中读取提交时间戳,如果Tlog中有时间戳,则存入tuple头部元数据中,再将相应的标志位置位。由于置位和读取位操作只涉及一个bit位,CPU会保证这两个操作的原子性。对于写写冲突,例如两个扫描进程同时将提交时间戳写入同一个tuple头部,由于对于一个xid而言,它的提交时间戳是从GTS分配到的,写入Tlog后是确定不变的,同时写入相同值不会导致tuple头部的元数据出现不一致的状态。

[0110] 请再次参阅图1和图2,在某些实施方式中,分布式事务一致性实现方法还包括步骤:

[0111] S40:根据记录的提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

[0112] 在某些实施方式中,分布式事务一致性实现装置100还包括回收模块30。步骤S40可以由回收模块30实现,或者说,回收模块30用于根据记录的提交时间戳回收失效数据记录以回收被更新或删除的失效记录。

[0113] 具体地,在TBase的每个单节点上,运行了多个活动session(进程),每个活动session里运行着事务。

[0114] 在每个DN节点内维护了一个全局范围变量RecentCommitTs,该变量记录了该数据库节点内部最大的事务提交时间戳。每个事务提交时,都会去更新这个变量。该变量随着时间变化递增,并且被持久化存储。在事务开始时或者vacuum开始时,扫描所有正在活动的session,获取每个活跃session的Tmin(session运行的事务(或语句)从GTS申请到的开始时间戳),计算出当前session的全局oldestTmin= $\min\{\text{所有活动session的Tmin, RecentCommitTs}\}$ 。最后将当前事务或语句在CN上从GTS申请到的开始时间戳赋值给当前session的Tmin变量。在回收tuple版本(热回收或vacuum)时,如果tuple的xmax具有提交时间戳,并且xmax小于oldestxmin,以及xmax的提交时间戳小于oldestTmin减去常量Delta(系统启动时可以配置),则该tuple可以回收,否则暂时不可以被回收。DN在执行被CN调度到该DN节点的事务时,如果该事务的开始时间戳大于或者等于RecentCommitTs减去Delta,则可以执行;否则,abort该事务或语句(用户或应用程序重试该事务)。

[0115] 请参阅图7,假设任意分布式数据库中任意两个事务T1(空间回收)和T2,在某个节点DN1上执行。定义 $R1 = \min\{\text{所有活动session的Tmin, RecentCommitTs}\}$ 为T1的session中计算出的局部变量值, $R2 = \text{RecentCommitTs}$ 为T2的session中计算出的局部变量值。

[0116] 假设tuple A是T1在回收过程中尝试回收的任意对象,假设tuple A的两个任意连续的版本v1和v2。假设T1回收了v1版本,则按照上述说明,v1可以回收,说明v1.xmax已经提交,由于v1.xmax=v2.xmin,因此v1.xmax.commit_ts=v2.xmin.commit_ts,并且v1.xmax.commit_ts<R1-Delta。

[0117] 假设在T2比T1在DN1上后扫描活动session列表(用全局锁串行化多个session在事务开始时扫描当前活动session列表),则 $R2 \geq R1$,因为RecentCommitTs是随着时间递增的。T2如果被接受执行,则 $T2.start_ts \geq R2 - Delta \geq R1 - Delta > v1.xmax.commit_ts = v2.xmin.commit_ts$ 。因此被回收的v1对T2不可见(T2可以看到v1的xmax的修改),同时T2可以看到tuple A的v1的后续版本,即便v1被回收。由于T2是任意的活动事务,因此被回收的v1对所有活动事务均不可见。

[0118] 假设在T2比T1在DN1上先扫描活动session列表(则T2.start_ts在T1扫描session列表之前已经赋值给T2的session的Tmin), $R1 = \min\{\text{所有活动session的Tmin}, \text{RecentCommitTs}\} \leq \min\{\text{所有活动session的Tmin}\} \leq T2.start_ts$ 。则 $v2.xmin.commit_ts = v1.xmax.commit_ts < R1 - Delta \leq T2.start_ts$,则说明在这种情况下被回收的v1对T2不可见(T2可以看到v1的xmax的修改)。同时T2可以看到tuple A的v1版本的后续版本,即便v1被回收了。由于T2是任意的活动事务,因此被回收的v1对所有活动的事务均不可见。

[0119] 由于TBase采用32位的连续递增的xid来给每个事务分配唯一的xid号,xid可能会在一段时间后被用光。因此,TBase有一个xid翻转的机制,在xid翻转回来从3开始重新计数之前,会将tuple里面的xmin进行冻结,即赋值一个冻结的xid(为2),从而回收之前分配的xid。每个tuple的xmin可以被冻结的前提是,xmin对所有事务可见了,而不需要读取它的提交时间戳进行可见性判断。

[0120] 因此在执行tuple冻结的时候,如果 $xmin.commit_ts < R1 - delta$,则该tuple的xmin对所有事务已经可见,可以被冻结。

[0121] 本发明实施方式的分布式事务一致性实现方法,采用GTS服务器来提供全局单调递增的时钟服务。GTS保证两次连续的申请时间戳是递增的,同时GTS会持久化和备份当前时钟,从而可以容灾。

[0122] 请参阅图8,GTS可以有一主一备,备机同步主机的状态,在主机崩溃后可以接管主机的工作。GTS生成的时间戳是由base_clock和从Intel TSC或者Hpet等高精度计数器获得的时间流逝绝对值clock_delta组成。Intel TSC或者Hpet可以返回自系统某个点(一般是启动)以来流逝的时钟数,可以精确到纳秒的级别。GTS采用64位的时间戳(精度到微秒)。Linux操作系统提供clock_gettime系统调用来获取(指定参数为CLOCK_MONOTONIC_RAW)。GTS初始化(数据库初始化的时候)的时候会设置一个base_clock,并持久化存储。GTS在启动的时候,会将存储中的base_clock读出来,通过操作系统接口中读取当前的clock(即启动以来流逝的时钟数)记录在last_cycle变量中。GTS收到申请时间戳请求后,从系统读取当前的时钟clock,根据last_cycle计算出delta(clock-last_cycle),将base_clock+delta返回给客户端(CN或者DN),即当前时间戳=base_clock+(clock-last_cycle)。同时GTS会周期性的更新并持久化base_clock,每隔一段时间(周期为T),GTS通过操作系统接口中读取当前的clock,减去last_cycle算出delta,将base_clock更新为base_clock加上delta,并将读取到的当前的clock赋值给last_cycle。

[0123] 为了避免时钟反转,GTS在每次启动的时候,都将base_clock加上一个固定的值(safe_delta)并持久化。base_clock同步周期T远小于safe_delta。同时GTS在同步更新base_clock的时候会去记录整个更新花费的时间(包括存储base_clock的I/O时间),如果更新花费时间超过了一定的预订的值(远小于safe_delta),GTS会报错停止工作,下次启动起来的时候会再次加上safe_delta以避免时钟反转。

[0124] GTS主机将base_clock周期同步给备机。备机在收到base_clock后,加上safe_delta再持久化存储。

[0125] GTS有一个更新的线程来周期性更新并持久化base_clock。为了避免和GTS的时间戳服务线程(计算当前时间戳返回给客户端)的读写冲突,GTS采用可扩展读写锁机制。GTS分配一个共享的锁数组(bitmap),锁实际为一个整形变量,每个整形变量占一个CPU缓存行(Cache Line),每个服务线程对应一个锁数组中的一个锁。每个服务线程一个锁(bit)。服务线程在读取时钟时(base_clock,last_cycle),通过compare and swap(CAS)原子操作来读取数组中相应的锁,如果为锁值0,则设置为1。如果锁值为1,则循环等待到变为0。处理器提供的CAS指令可以将整形变量的检查(是否为0)和设置两个操作原子化。而更新线程在要用delta去更新base_clock以及重置last_cycle变量时,需要去依次检查每个服务线程的锁是否为0,如果为0,则设置为1。如果为1,则等待变为0。上述操作同样用CAS指令来保证原子性。多个服务线程在并发读时钟时是免锁的,因为各自读取和设置各自的位,互相不冲突。服务线程和更新线程彼此互斥。由于更新线程是周期性更新,锁竞争开销较小。

[0126] 由于GTS采用的是精度到微秒的时间戳,而数据中心网络延迟大于一微秒,通常数十微秒,因此GTS分配的时间戳不会出现反转的情况。

[0127] 假设客户端发起申请时间戳,GTS收到请求后分配了时间戳,经过T时间返回到客户端,客户端再发起申请时间戳请求,经过T时间(假设延迟相同)到达GTS,则此时GTS已经过去了2T时间(T为至少数微秒),则由于GTS的精度是微秒,GTS不可能生成等于甚至小于之前分配的时间戳。

[0128] 综上所述,本发明实施方式的分布式事务一致性实现方法,事务每个语句只需要从GTS(Global Timestamp Server)获取一个时间戳(8个字节),同时GTS采用可扩展读写锁维护全局时钟,实现了多核可扩展的时间戳处理能力。采用检查写事务是否过了两阶段提交中的第一阶段来决定读事务是否需要等待写事务结束进行可见性判断,从而保证在分布式多个节点上所有开始时间戳大于写事务提交时间戳的读事务都可以看到该写事务的修改内容,从而保证分布式一致性。并且,在各个节点事务提交时只需要往各节点的时间戳日志存储(存储了<xid,commit_ts>映射关系)中简单的写入一次事务提交时间戳。为了加速可见性判断时读取提交时间戳的性能,从时间戳日志存储中读取到的事务提交时间戳缓存到tuple头部记录中,来加速下次访问同一个tuple做可见性判断的性能,直接从tuple头部中读取到。

[0129] 此外,本发明实施方式的分布式事务一致性实现方法,可以让分布式数据库的OLTP(Online Transaction Processing)处理能力随着集群规模增长而接近线性可扩展。

[0130] 请参阅图9和图10,下面以在60个节点规模集群上的TPCC实验结果为例进行说明。

[0131] 以两个纬度对TBase的OLTP性能和可扩展性进行测试。

[0132] 第一个纬度是集群规模固定,不断增加TPCC client数目,从1逐步增加到30(1,2,

5,10,15,20,25,30),每个client有100个连接,最大时有3000个连接,测试TBase的吞吐量。同时将DN进行分组,分为3组,每10台DN一组,TPCC client也相应的进行分组,前10个Client的表创建在第一组,中间10个创建在第二组,最后10个Client分布到第三组。

[0133] 第二个纬度是逐步增大集群的规模,从2增加到60 (1CN+1DN,2CN+2DN,5CN+5DN,10CN+10DN,15CN+15DN,20CN+20DN,25CN+25DN,30CN+30DN),同时相应的增加TPCC client的数目(每增加一组CN+DN,则相应的增加一个TPCCclient)。我们通过Group来控制集群规模大小,分别设置1个group (1CN+1DN到10CN+10DN),2个group (15CN+15DN到20CN+20DN)和3个group (25CN+25DN到30CN+30DN)。每个group最多容纳10个DN,比如15CN+15DN时,创建了两个Group,第一个Group包括10个DN,第二个Group包括剩下的5个DN。

[0134] 如此,本发明实施方式的分布式事务一致性实现方法实现在TBase分布式数据库中,TBase可以提供随着节点规模增长而吞吐量线性增长的事务处理能力。

[0135] 在本说明书的描述中,参考术语“一个实施方式”、“一些实施方式”、“示意性实施方式”、“示例”、“具体示例”或“一些示例”等的描述意指结合实施方式或示例描述的具体特征、结构、材料或者特点包含于本发明的至少一个实施方式或示例中。在本说明书中,对上述术语的示意性表述不一定指的是相同的实施方式或示例。而且,描述的具体特征、结构、材料或者特点可以在任何的一个或多个实施方式或示例中以合适的方式结合。

[0136] 流程图中或在此以其他方式描述的任何过程或方法描述可以被理解为,表示包括一个或更多个用于实现特定逻辑功能或过程的步骤的可执行指令的代码的模块、片段或部分,并且本发明的优选实施方式的范围包括另外的实现,其中可以不按所示出或讨论的顺序,包括根据所涉及的功能按基本同时的方式或按相反的顺序,来执行功能,这应被本发明的实施例所属技术领域的技术人员所理解。

[0137] 在流程图中表示或在此以其他方式描述的逻辑和/或步骤,例如,可以被认为用于实现逻辑功能的可执行指令的定序列表,可以具体实现在任何计算机可读介质中,以供指令执行系统、装置或设备(如基于计算机的系统、包括处理模块的系统或其他可以从指令执行系统、装置或设备取指令并执行指令的系统)使用,或结合这些指令执行系统、装置或设备而使用。就本说明书而言,“计算机可读介质”可以是任何可以包含、存储、通信、传播或传输程序以供指令执行系统、装置或设备或结合这些指令执行系统、装置或设备而使用的装置。计算机可读介质的更具体的示例(非穷尽性列表)包括以下:具有一个或多个布线的电连接部(控制方法),便携式计算机盘盒(磁装置),随机存取存储器(RAM),只读存储器(ROM),可擦除可编程只读存储器(EPROM或闪速存储器),光纤装置,以及便携式光盘只读存储器(CDROM)。另外,计算机可读介质甚至可以是可在其上打印程序的纸或其他合适的介质,因为可以例如通过对纸或其他介质进行光学扫描,接着进行编辑、解译或必要时以其他合适方式进行处理来以电子方式获得程序,然后将其存储在计算机存储器中。

[0138] 应当理解,本发明的实施方式的各部分可以用硬件、软件、固件或它们的组合来实现。在上述实施方式中,多个步骤或方法可以用存储在存储器中且由合适的指令执行系统执行的软件或固件来实现。例如,如果用硬件来实现,和在另一实施方式中一样,可用本领域公知的下列技术中的任一项或他们的组合来实现:具有用于对数据信号实现逻辑功能的逻辑门电路的离散逻辑电路,具有合适的组合逻辑门电路的专用集成电路,可编程门阵列(PGA),现场可编程门阵列(FPGA)等。

[0139] 本技术领域的普通技术人员可以理解实现上述实施例方法携带的全部或部分步骤是可以通过程序来指令相关的硬件完成, 的程序可以存储于一种计算机可读存储介质中, 该程序在执行时, 包括方法实施例的步骤之一或其组合。

[0140] 此外, 在本发明的各个实施例中的各功能单元可以集成在一个处理模块中, 也可以是各个单元单独物理存在, 也可以两个或两个以上单元集成在一个模块中。上述集成的模块既可以采用硬件的形式实现, 也可以采用软件功能模块的形式实现。集成的模块如果以软件功能模块的形式实现并作为独立的产品销售或使用, 也可以存储在一个计算机可读取存储介质中。

[0141] 上述提到的存储介质可以是只读存储器, 磁盘或光盘等。

[0142] 尽管上面已经示出和描述了本发明的实施方式, 可以理解的是, 上述实施方式是示例性的, 不能理解为对本发明的限制, 本领域的普通技术人员在本发明的范围内可以对上述实施实施进行变化、修改、替换和变型。

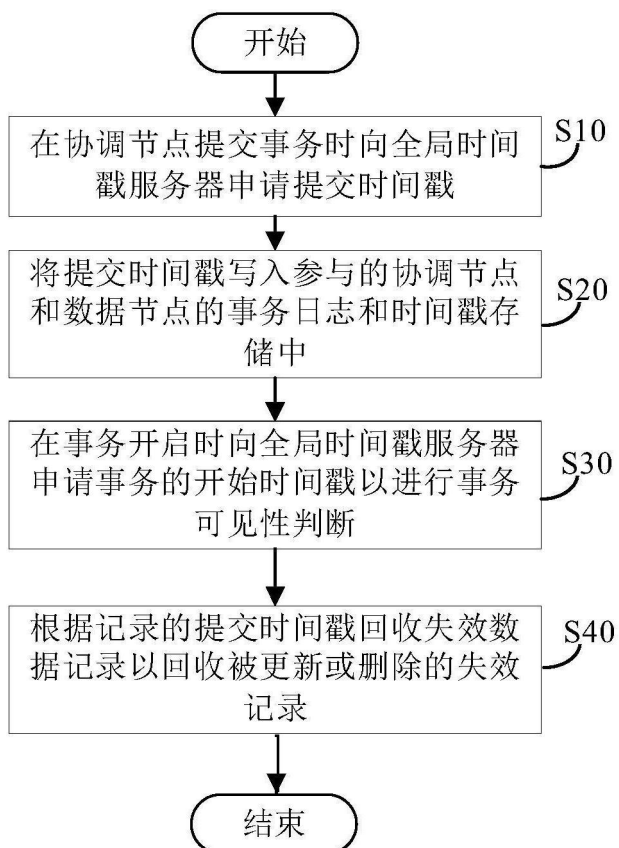


图1

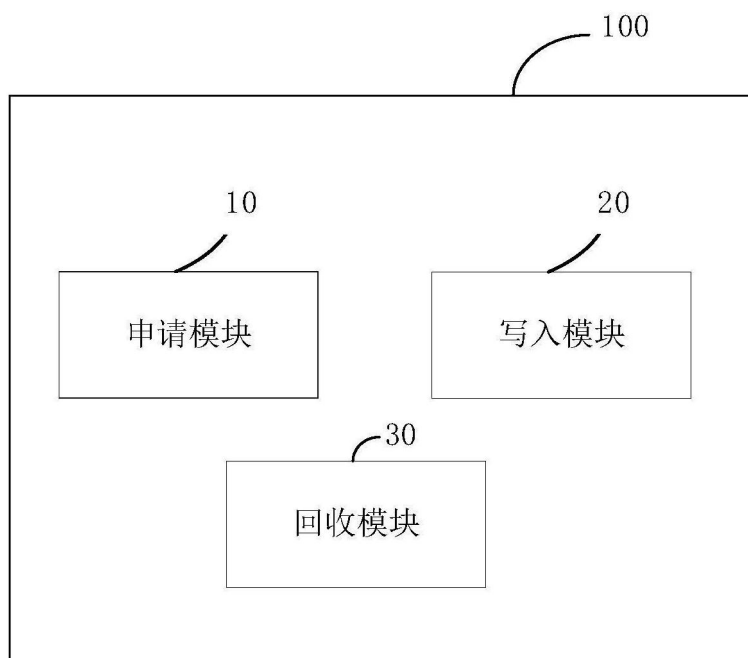


图2

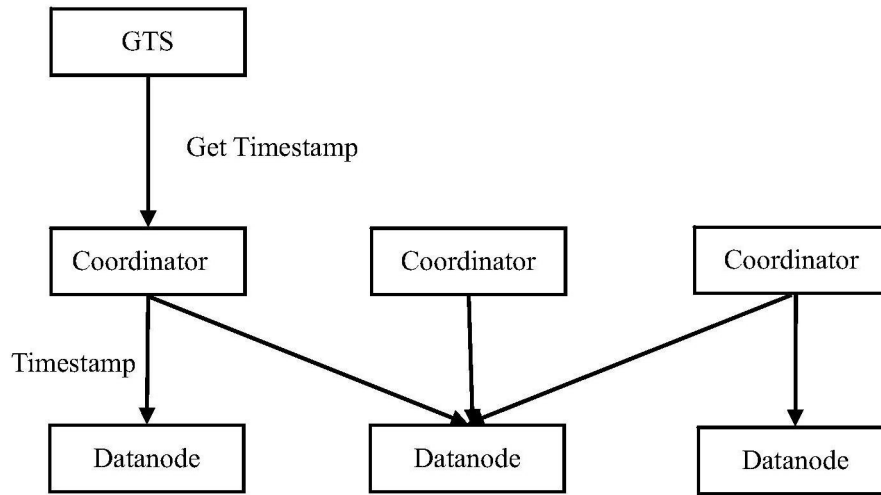


图3

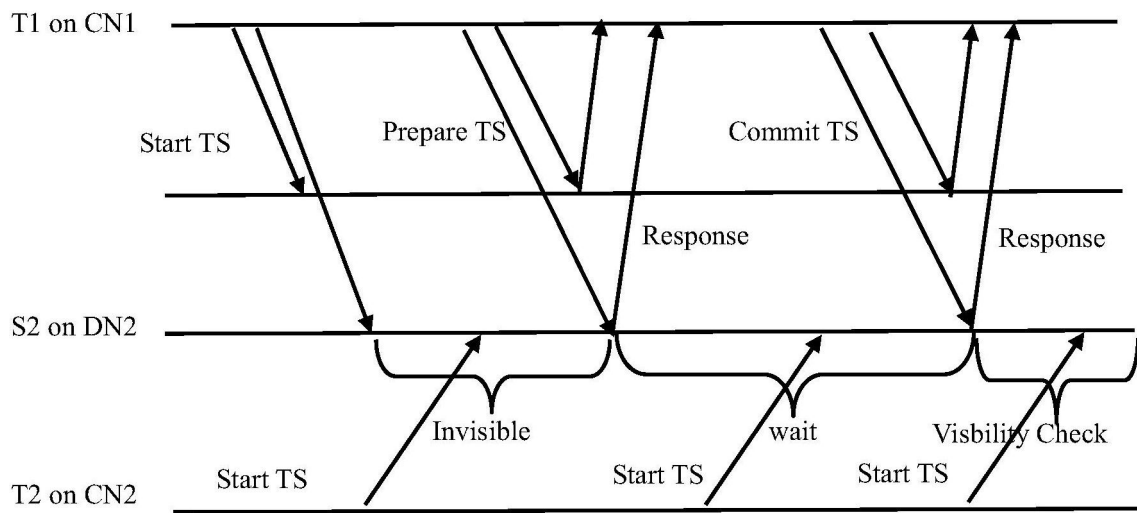


图4

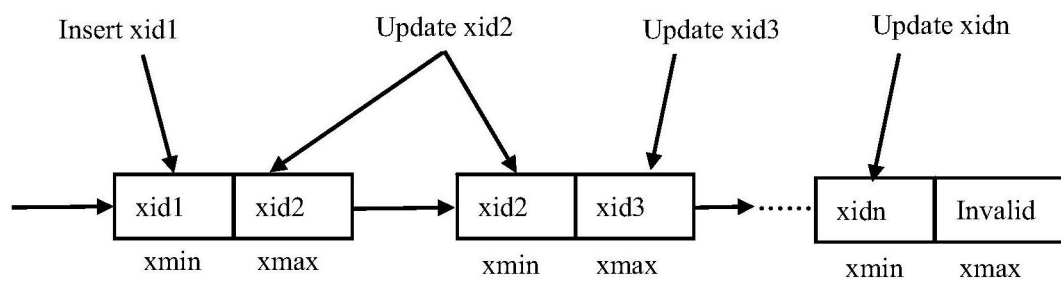


图5

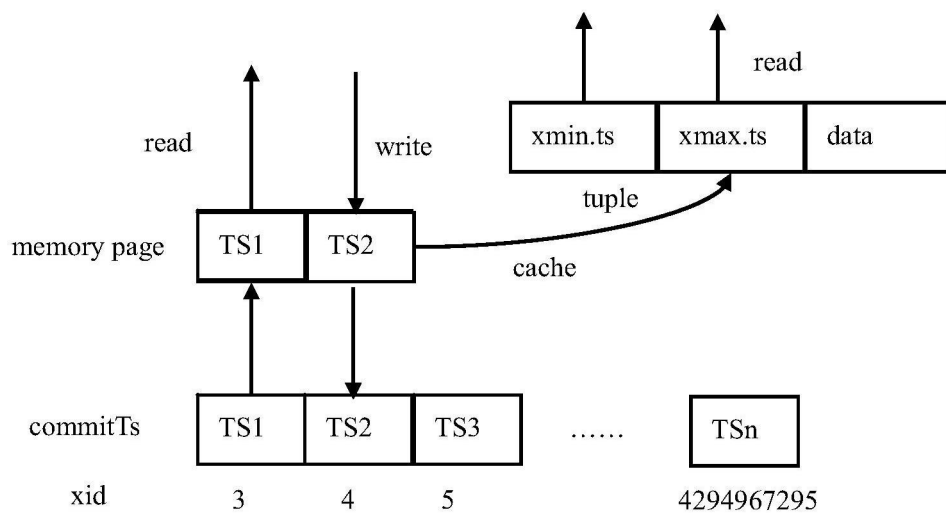


图6

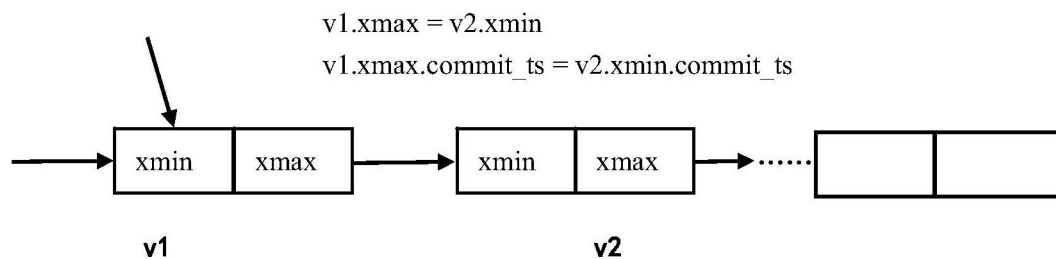


图7

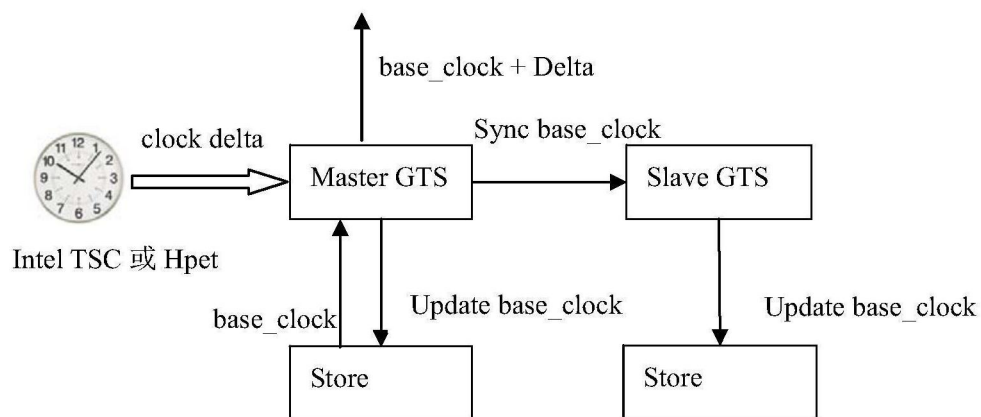


图8

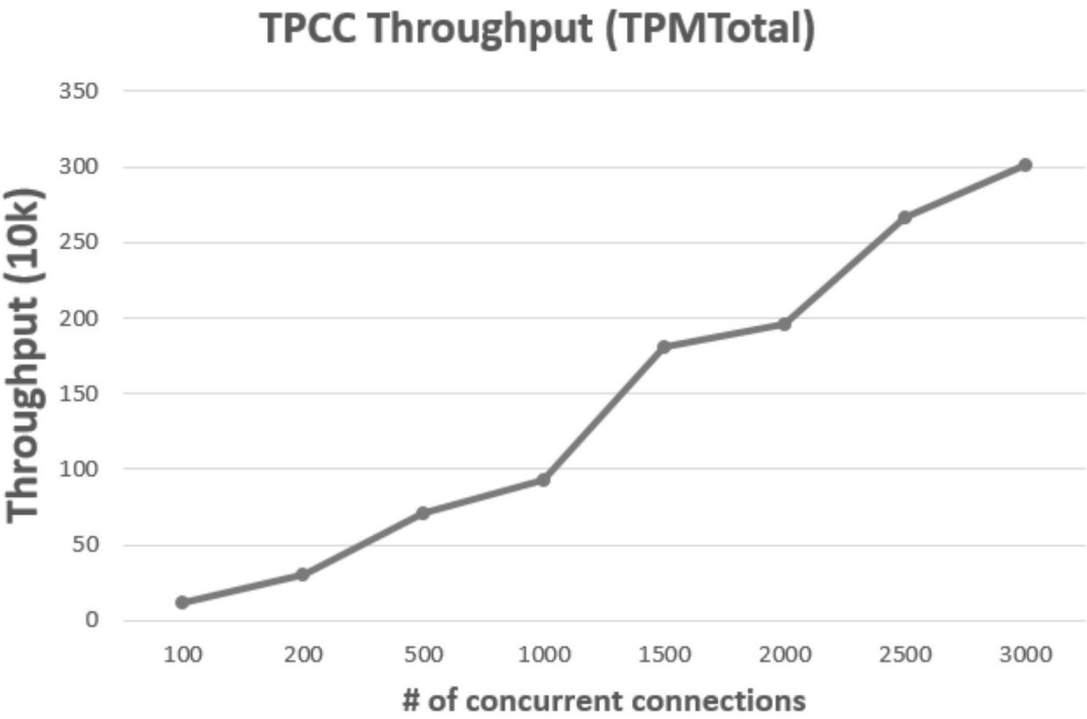


图9

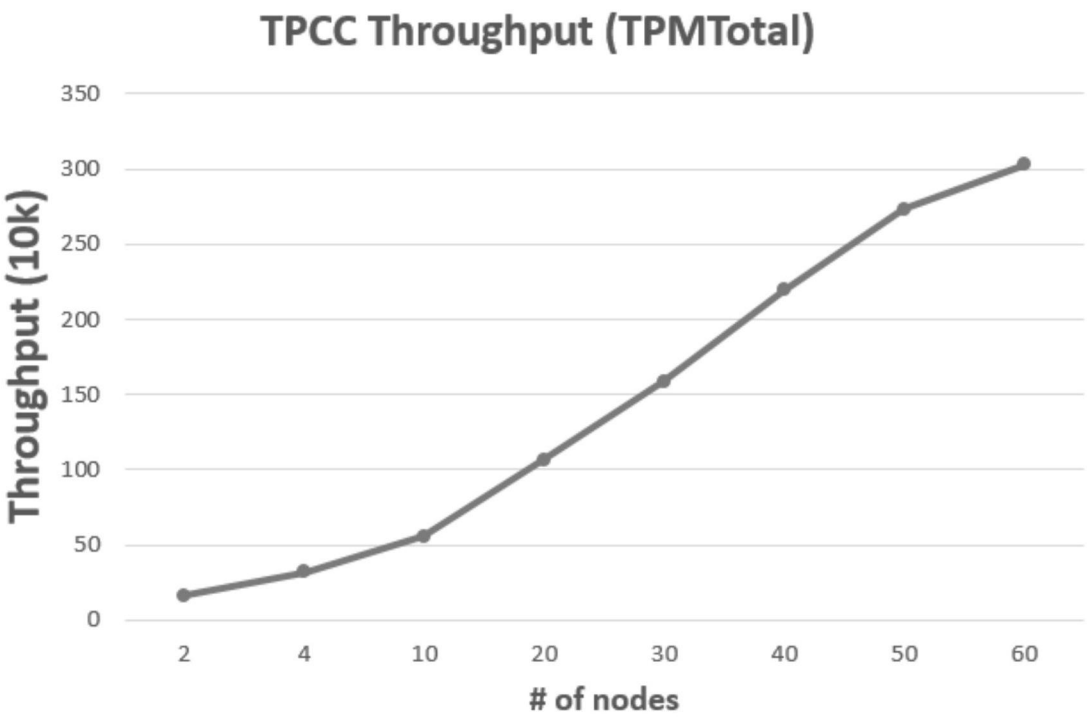


图10