(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 114328613 B
(45)授权公告日 2022.07.05

(21)申请号 202210200793.6

(22)申请日 2022.03.03

(65)同一申请的已公布的文献号
申请公布号 CN 114328613 A

(43)申请公布日 2022.04.12

(73)专利权人 阿里云计算有限公司
地址 310024 浙江省杭州市西湖区转塘科技经济区块12号

(72)发明人 赵建伟 张纪杨 马国庆

(74)专利代理机构 北京润泽恒知识产权代理有限公司 11319
专利代理师 赵娟

(51)Int.Cl.
G06F 16/2453(2019.01)
G06F 16/2458(2019.01)
G06F 16/27(2019.01)
G06F 16/28(2019.01)

(56)对比文件
CN 109933412 A,2019.06.25
WO 2021107988 A1,2021.06.03
US 2017255668 A1,2017.09.07
WO 2016078423 A1,2016.05.26
CN 106445644 A,2017.02.22
CN 111259083 A,2020.06.09
CN 110716793 A,2020.01.21
奚军庆 等.分布式系统设计中NewSQL数据库技术的应用.《长江信息通信》.2012,第64-67页.
Thamir M. Qadah.A queue-oriented transaction processing paradigm.《ACM》.2019,第26-30页.

审查员 王璐

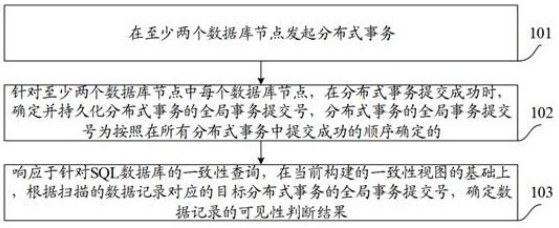
权利要求书3页 说明书19页 附图3页

(54)发明名称

SQL数据库中分布式事务的处理方法、装置及系统

(57)摘要

本发明实施例提供了SQL数据库中分布式事务的处理方法、装置及系统,SQL数据库包括多个数据库节点,所述方法包括:在至少两个数据库节点发起分布式事务;针对至少两个数据库节点中每个数据库节点,在分布式事务提交成功时,确定并持久化分布式事务的全局事务提交号,分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的;响应于针对SQL数据库的一致性查询,在当前构建的一致性视图的基础上,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定数据记录的可见性判断结果。通过本发明实施例,实现了对SQL数据库中分布式一致性查询的优化,能够根据分布式事务的提交顺序来进行可见性判断。



CN 114328613 B

1.一种SQL数据库中分布式事务的处理方法,其特征在于,所述SQL数据库包括多个数据库节点,所述方法包括:

在至少两个数据库节点发起分布式事务;

针对所述至少两个数据库节点中每个数据库节点,在所述分布式事务提交成功时,确定并持久化所述分布式事务的全局事务提交号,所述分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的;

响应于针对所述SQL数据库的一致性查询,在当前构建的一致性视图的基础上,查询扫描的数据记录对应的目标分布式事务的事务状态,在所述目标分布式事务的事务状态为未提交成功状态时,控制所述一致性查询进入等待状态,直至所述目标分布式事务的事务状态更新为提交成功状态时,查询所述目标分布式事务的全局事务提交号;

根据所述扫描的数据记录对应的目标分布式事务的全局事务提交号,确定所述数据记录的可见性判断结果;

所述数据记录存储有用于指向所述目标分布式事务对应的事务槽的事务槽地址,所述事务槽用于持久化所述分布式事务的全局事务提交号和所述分布式事务的事务状态。

2.根据权利要求1所述的方法,其特征在于,所述当前构建的一致性视图对应有一第一全局事务提交号,所述根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定所述数据记录的可见性判断结果,包括:

在所述目标分布式事务的全局事务提交号小于或等于所述第一全局事务提交号的情况下,判定所述数据记录为可见状态;

在所述目标分布式事务的全局事务提交号大于所述第一全局事务提交号的情况下,判定所述数据记录为不可见状态。

3.根据权利要求1所述的方法,其特征在于,所述分布式事务的全局事务提交号为单调递增的时序值。

4.根据权利要求1-3任一项所述的方法,其特征在于,还包括:

在所述分布式事务提交过程中,持久化所述分布式事务的事务状态;

在所述根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定所述数据记录的可见性判断结果之前,还包括:在所述目标分布式事务的事务状态为提交成功状态时,查询所述目标分布式事务的全局事务提交号。

5.根据权利要求1所述的方法,其特征在于,所述分布式事务为基于两阶段提交协议进行提交,所述事务状态与所述分布式事务在两阶段提交协议中所处的阶段相对应。

6.根据权利要求1所述的方法,其特征在于,还包括:

在所述分布式事务启动时,在预置的事务表中分配所述分布式事务对应的事务槽。

7.根据权利要求6所述的方法,其特征在于,还包括:

当对目标事务槽进行复用时,判断所述目标事务槽对应的分布式事务是否已结束,并在所述目标事务槽对应的分布式事务结束的情况下,允许对所述目标事务槽进行复用。

8.根据权利要求1所述的方法,其特征在于,还包括:

在存在至少两个分布式事务修改同一个数据记录时,通过事务锁阻塞除在先进行修改的分布式事务外的其他分布式事务,直至所述在先进行修改的分布式事务获得全局事务提交号并提交。

9.根据权利要求1所述的方法,其特征在于,还包括:

在对数据记录的历史版本进行清理时,确定第二全局事务提交号;

在当前历史版本对应的分布式事务的全局事务提交号小于或等于所述第二全局事务提交号的情况下,对所述当前历史版本进行清理;

在当前历史版本对应的分布式事务的全局事务提交号大于所述第二全局事务提交号的情况下,对所述第二全局事务提交号进行增大。

10.一种SQL数据库中分布式事务的处理装置,其特征在于,所述SQL数据库包括多个数据库节点,所述装置包括:

分布式事务发起模块,用于在至少两个数据库节点发起分布式事务;

全局事务提交号持久化模块,用于针对所述至少两个数据库节点中每个数据库节点,在所述分布式事务提交成功时,确定并持久化所述分布式事务的全局事务提交号,所述分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的;

一致性查询模块,用于响应于针对所述SQL数据库的一致性查询,在当前构建的一致性视图的基础上,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定所述数据记录的可见性判断结果;

其中,所述装置还包括:

事务状态查询模块,用于查询所述扫描的数据记录对应的目标分布式事务的事务状态;

第二全局事务提交查询模块,用于在所述目标分布式事务的事务状态为未提交成功状态时,控制所述一致性查询进入等待状态,直至所述目标分布式事务的事务状态更新为提交成功状态时,查询目标分布式事务的全局事务提交号;

所述数据记录存储有用于指向所述目标分布式事务对应的事务槽的事务槽地址,所述事务槽用于持久化所述分布式事务的全局事务提交号和所述分布式事务的事务状态。

11.一种SQL数据库中分布式事务的处理系统,其特征在于,所述SQL数据库包括多个数据库节点,所述处理系统包括服务组件和处理模块;

所述服务组件,用于在所述SQL数据库中分布式事务提交成功时,生成所述分布式事务的全局事务提交号;

所述处理模块,用于在所述SQL数据库中至少两个数据库节点发起分布式事务的情况下,针对所述至少两个数据库节点中每个数据库节点,在所述分布式事务提交成功时,确定并持久化所述分布式事务的全局事务提交号;响应于针对所述SQL数据库的一致性查询,在当前构建的一致性视图的基础上,查询扫描的数据记录对应的目标分布式事务的事务状态,在所述目标分布式事务的事务状态为未提交成功状态时,控制所述一致性查询进入等待状态,直至所述目标分布式事务的事务状态更新为提交成功状态时,查询所述目标分布式事务的全局事务提交号;根据所述扫描的数据记录对应的目标分布式事务的全局事务提交号,确定所述数据记录的可见性判断结果;其中,所述分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的;

所述数据记录存储有用于指向所述目标分布式事务对应的事务槽的事务槽地址,所述事务槽用于持久化所述分布式事务的全局事务提交号和所述分布式事务的事务状态。

12.一种电子设备,其特征在于,包括处理器、存储器及存储在所述存储器上并能够在

所述处理器上运行的计算机程序,所述计算机程序被所述处理器执行时实现如权利要求1至9中任一项所述的SQL数据库中分布式事务的处理方法。

13.一种计算机可读存储介质,其特征在于,所述计算机可读存储介质上存储计算机程序,所述计算机程序被处理器执行时实现如权利要求1至9中任一项所述的SQL数据库中分布式事务的处理方法。

SQL数据库中分布式事务的处理方法、装置及系统

技术领域

[0001] 本发明涉及数据库技术领域，特别是涉及SQL数据库中分布式事务的处理方法、装置及系统。

背景技术

[0002] 面对日益增长的海量数据，传统的集中式数据库的弊端日益显现，分布式数据库相比于传统的集中式数据库主要有如下优点：

[0003] 1、更强的扩展性。分布式数据库可以通过增添存储节点来实现存储容量的线性扩展，而集中式数据库的可扩展性十分有限。

[0004] 2、更高的并发访问能力。分布式数据库由于采用多台主机组成存储集群，所以相对集中式数据库，它可以提供更高的用户并发访问量。

[0005] 3、更高的可靠性和更好的可用性。由于数据分布在多个场地并有许多副本数据，在个别场地或个别通信链路发生故障时，不致于导致整个系统的崩溃，而且系统的局部故障不会引起全局失控。

[0006] 对于SQL数据库，如MySQL数据库，可以通过XA(eXtended Architecture，由X/Open组织提出的分布式交易处理的规范)协议发起XA事务来实现分布式事务，进而可以允许多个MySQL实例共同参与到一个全局事务中，通过XA事务使得MySQL有能力成为分布式数据库。

[0007] 但在事实上，MySQL仍然是一个单机数据库管理系统，所有基于MySQL的分布式数据库，在产品力上相比于业界上原生的分布式数据库来说，有所欠缺，特别是，现有的MySQL要在内核层面实现高效的分布式一致性查询，几乎是没有手段的。

发明内容

[0008] 鉴于上述问题，提出了以便提供克服上述问题或者至少部分地解决上述问题的SQL数据库中分布式事务的处理方法、装置及系统，包括：

[0009] 一种SQL数据库中分布式事务的处理方法，SQL数据库包括多个数据库节点，所述方法包括：

[0010] 在至少两个数据库节点发起分布式事务；

[0011] 针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号，分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的；

[0012] 响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果。

[0013] 可选地，当前构建的一致性视图对应有一第一全局事务提交号，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果，包括：

[0014] 在目标分布式事务的全局事务提交号小于或等于第一全局事务提交号的情况下，判定数据记录为可见状态；

[0015] 在目标分布式事务的全局事务提交号大于第一全局事务提交号的情况下，判定数据记录为不可见状态。

[0016] 可选地，分布式事务的全局事务提交号为单调递增的时序值。

[0017] 可选地，还包括：

[0018] 在分布式事务提交过程中，持久化分布式事务的事务状态；

[0019] 在根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果之前，还包括：

[0020] 查询扫描的数据记录对应的目标分布式事务的事务状态；

[0021] 在目标分布式事务的事务状态为提交成功状态时，查询目标分布式事务的全局事务提交号；

[0022] 在目标分布式事务的事务状态为未提交成功状态时，控制一致性查询进入等待状态，直至目标分布式事务的事务状态更新为提交成功状态时，查询目标分布式事务的全局事务提交号。

[0023] 可选地，分布式事务为基于两阶段提交协议进行提交，事务状态与分布式事务在两阶段提交协议中所处的阶段相对应。

[0024] 可选地，数据记录存储有用于指向目标分布式事务对应的事务槽的事务槽地址，还包括：

[0025] 在分布式事务启动时，在预置的事务表中分配分布式事务对应的事务槽，事务槽用于持久化分布式事务的全局事务提交号和分布式事务的事务状态。

[0026] 可选地，还包括：

[0027] 当对目标事务槽进行复用时，判断目标事务槽对应的分布式事务是否已结束，并在目标事务槽对应的分布式事务结束的情况下，允许对目标事务槽进行复用。

[0028] 可选地，还包括：

[0029] 在存在至少两个分布式事务修改同一个数据记录时，通过事务锁阻塞除在先进行修改的分布式事务外的其他分布式事务，直至在先进行修改的分布式事务获得全局事务提交号并提交。

[0030] 可选地，还包括：

[0031] 在对数据记录的历史版本进行清理时，确定第二全局事务提交号；

[0032] 在当前历史版本对应的分布式事务的全局事务提交号小于或等于第二全局事务提交号的情况下，对当前历史版本进行清理；

[0033] 在当前历史版本对应的分布式事务的全局事务提交号大于第二全局事务提交号的情况下，对第二全局事务提交号进行增大。

[0034] 一种SQL数据库中分布式事务的处理装置，SQL数据库包括多个数据库节点，所述装置包括：

[0035] 分布式事务发起模块，用于在至少两个数据库节点发起分布式事务；

[0036] 全局事务提交号持久化模块，用于针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号，分布式事务的全

局事务提交号为按照在所有分布式事务中提交成功的顺序确定的；

[0037] 一致性查询模块，用于响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果。

[0038] 一种SQL数据库中分布式事务的处理系统，SQL数据库包括多个数据库节点，处理系统包括服务组件和处理模块；

[0039] 服务组件，用于在SQL数据库中分布式事务提交成功时，生成分布式事务的全局事务提交号；

[0040] 处理模块，用于在SQL数据库中至少两个数据库节点发起分布式事务的情况下，针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号；响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果；其中，分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的。

[0041] 一种电子设备，包括处理器、存储器及存储在存储器上并能够在处理器上运行的计算机程序，计算机程序被处理器执行时实现如上所述的SQL数据库中分布式事务的处理方法。

[0042] 一种计算机可读存储介质，计算机可读存储介质上存储计算机程序，计算机程序被处理器执行时实现如上所述的SQL数据库中分布式事务的处理方法。

[0043] 本发明实施例具有以下优点：

[0044] 在本发明实施例中，通过在至少两个数据库节点发起分布式事务，然后针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号，响应于针对所述SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果，实现了对SQL数据库中分布式一致性查询的优化，能够根据分布式事务的提交顺序来进行可见性判断，保证了一致性查询的准确性、高效性。

附图说明

[0045] 为了更清楚地说明本发明的技术方案，下面将对本发明的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发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动性的前提下，还可以根据这些附图获得其他的附图。

[0046] 图1是本发明一实施例提供的一种SQL数据库中分布式事务的处理方法的步骤流程图；

[0047] 图2是本发明一实施例提供的另一种SQL数据库中分布式事务的处理方法的步骤流程图；

[0048] 图3是本发明一实施例提供的一种系统结构的示意图；

[0049] 图4是本发明一实施例提供的一种SQL数据库中分布式事务的处理装置的结构框图；

[0050] 图5是本发明一实施例提供的一种SQL数据库中分布式事务的处理系统的结构框图。

具体实施方式

[0051] 为使本发明的上述目的、特征和优点能够更加明显易懂，下面结合附图和具体实施方式对本发明作进一步详细的说明。显然，所描述的实施例是本发明一部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其他实施例，都属于本发明保护的范围。

[0052] 对于SQL数据库，如MySQL数据库，其可以通过XA协议发起XA事务来实现分布式事务，进而可以允许多个MySQL实例共同参与到一个全局事务中。在MySQL数据库中，通过XA事务实现分布式事务具体如下：

[0053] XA START xid(XID是用于唯一标识全局的XA事务)：开启一个事务，并将事务置于ACTIVE状态，此后执行的SQL语句都将置于该是事务中。

[0054] XA END xid：将事务置于IDLE状态，表示事务内的SQL操作完成。

[0055] XA PREPARE xid：实现事务提交的准备工作，事务状态置于PREPARED状态。事务如果无法完成提交前的准备操作，该语句会执行失败。

[0056] XA COMMIT xid：事务最终提交，完成持久化。

[0057] XA ROLLBACK xid：事务回滚终止。

[0058] XA RECOVER：查看MySQL中存在的PREPARED状态的XA事务。

[0059] 对于执行分布式事务的数据库，需要进行分布式的一致性查询，一致性查询最核心的问题在于：如何在多个分布式节点之间采用同一个视图，即看到一致性的结果。而从上文可见，在多个MySQL分布式节点之间，同一个全局分布式事务的标识号是XID，会存在以下问题：

[0060] 1、分布式事务并发执行时，分布式事务因为网络、负载、调度等原因，在多个分布式MySQL节点的执行顺序不同。

[0061] 例如，有两个分布式事务DT_A、DT_B，分布式事务DT_A在两个分布式节点(Node1，Node2)上执行的本地事务是LT_A1、LT_A2，分布式事务DT_B在两个分布式节点(Node1，Node2)上执行的本地事务是LT_B1，LT_B2，它们的执行顺序可能是：

[0062] Node1：LT_A1，LT_B1

[0063] Node2：LT_B2，LT_A2

[0064] 可见，在任何时候看到的数据库全局状态，都不应该是：LT_A1，LT_B1，LT_B2，而由于执行顺序的不同，确实会出现这样的情况，因为此时看到的DT_A事务缺失了在Node2的修改LT_A2，看到了一个不一致的数据库状态。

[0065] 2、XID只是一个XA事务的唯一标识，在XA事务启动的时候指定，然而事务的可见性仅仅取决于事务的提交顺序，而多个分布式事务在不同的分布式MySQL节点上的提交顺序也不同。

[0066] 具体而言，事务的可见性由事务的提交顺序决定，而XID是XA事务启动时分配的唯一标识。和提交顺序没有关系(即XID不能表示事务的提交顺序)，而MySQL只有XID做不到全局一致性读。

[0067] 3、XID虽然持久化在Undo表空间中，但在XA事务结束后(提交或回滚)，就会被Purge系统(不会再有查询需要看的历史版本没有存在的必要，Purge系统会定期按需清理掉这些历史版本数据)清理掉。

[0068] 4、对于多个分布式MySQL节点，通过维护XID信息来构建视图是困难的，而这样的视图在多个分布式节点以及Client节点之间共享也是困难的。

[0069] 具体而言，由于可见性是由事务的提交顺序决定的，XID不能代表事务的提交顺序，只能表示事务的起始标识，如果只有XID，却要用XID来表示事务可见性工程实现比较复杂且性能不好，在分布式场景下会放大问题，几乎处于不可用状态，具体的可见性工程实现如下：

[0070] A、需要维护当前的活跃事务的XID链表，有多少个活跃事务，就有多少个元素。XID链表的读取和修改，都需要在全局锁保护下。

[0071] B、读事务启动的时候需要拿到当前的活跃事务的XID链表。

[0072] C、分布式事务的协调者需用统筹各个分布式节点的活跃XID链表信息。需要知道每个XID代表的分布式事务在各个节点上的执行状态：未启动，执行中，提交，回滚等。

[0073] 基于此，本发明通过系统化、结构化的改造，让如MySQL这种传统的单机数据库具有更强大、更内聚的分布式能力，相比于MySQL原生的事务系统，主要区别如下：

[0074] 1、引入Transaction Table(事务表)，用于持久化事务状态。在事务启动的时候，会在Transaction Table分配一个Transaction Slot(事务槽)，在事务提交的时候，会将事务状态信息回填到Transaction Slot上。

[0075] 作为一示例，事务状态信息可以包括XID、Trx ID(本地事务的唯一标识，在事务开始的时候分配的全局唯一递增号)、State(事务的状态，如：启动、提交、回滚、清理等)、TCN(Transaction Commit Number，本地事务提交号)、GCN(Global Commit Number，全局事务提交号)等。

[0076] 2、引入本地事务提交号TCN、全局事务提交号GCN，来表示本地、全局事务的提交顺序。

[0077] 3、数据记录在格式上除了Trx ID以及Rollptr(回滚段指针，存在于每一个InnoDB数据行上，通过回滚段指针，能够回溯该数据行的最近的一个历史版本)两个系统字段外，还额外引入系统字段：GCN、TCN以及TSA(Transaction Slot Address，事务槽地址)，TSA字段上记录了修改本记录的事务所对应的Transaction Slot的地址。

[0078] 其中，数据记录可以为记录(record)、数据行、行记录等，都可以看做是同一个概念，即表示一行数据在物理上的实际存储内容，当用户插入一行数据，就会在数据库上持久化一条数据记录。

[0079] 4、丢弃掉Read View上用Trx ID来做可见性判断的机制，而采用TCN、GCN做可见性判断的机制。

[0080] 对于可见性判断，可以通过如下例子理解：

[0081] 有两个事务先后发生了：Trx_A以及Trx_B

[0082] 时间线为：读事务R1-->写事务Trx_A提交-->读事务R2-->写事务Trx_B提交-->读事务R3。

[0083] 显然：读事务R1不能看到Trx_A和Trx_B的修改，原因在于这个时候Trx_A和Trx_B

都还没有提交,R2可以看到Trx_A的修改,但不能看到Trx_B的修改,R3可以看到Trx_A和Trx_B的修改。

[0084] 所以,一个视图能否看到某个事务的修改:关键在于视图建立时,该事务是否已经提交。

[0085] 对于用Trx ID来做可见性判断的机制,可以通过如下例子理解:

[0086] 在T1时刻,有两个活跃事务TrxA,TrxB。它们的Trx ID分别是50,100。同时,事务系统中还维护两个边界:30和110。所有Trx ID < 30的事务都已经提交,而Trx ID > 110的事务都未提交。

[0087] 在T2时刻,读事务R1启动,构建了它的视图ReadView。该视图实际上代表本时刻数据库的状态。即“Trx ID为50,100的事务处于活跃状态,而Trx ID < 30的事务都已经提交,而Trx ID > 110的事务都未提交”。

[0088] 当读事务R1扫描到行记录rec_1,发现修改rec_1的事务的Trx ID的是15,则该行记录对于R1是可见的。当读事务发现修改rec_1的事务的Trx ID是120,则该行记录不可见。当读事务发现修改rec_1的事务的Trx ID是75,通过查询R1视图的活跃事务链表信息,发现Trx ID为75的事务不在活跃事务范围内。则判断该行记录可见。当读事务发现修改rec_1的事务的Trx ID是50,通过查询R1视图的活跃事务链表信息,发现Trx ID为50的事务在活跃事务范围内。则判断该行记录不可见。

[0089] 对于用Trx ID来做可见性判断的机制存在以下问题:

[0090] A、写事务和写事务、读事务和读事务,以及读事务和写事务之间都因为一把全局的事务系统大锁,而造成了相互干扰。特别是读写混合场景,写事务较多会导致Active Trx Set比较大,在构造Read View的时候需要持有锁更长的时间,来拷贝Active Trx Set,从而造成了更严重的锁争抢问题。而MySQL MVCC机制设计的初衷是:读操作只读取历史版本,不会与写操作相互阻塞。但是因为这把全局大锁的原因,读写干扰难以避免。

[0091] B、由于真实业务场景中,大部分写事务都是小事务,先进入Active Trx Set的一般也会先出来。但是Active Trx Set的数据结构是数组,移走前面的元素,需要把后面的元素全部往前挪,整个过程都需持有全局事务系统大锁,进一步加剧了问题的严重程度。

[0092] C复杂的数据结构以及复杂的工程实现,使得MySQL无法充分利用单机的多核能力。而在分布式场景下,分布式一致性查询,需要共享一个全局的Read View。如果Read View本身的设计就很复杂,那么各个分布式节点之间需要保持一致性就会变得更加复杂和困难。

[0093] 而对于采用TCN、GCN做可见性判断的机制,可以通过如下例子理解:

[0094] 在T1时刻,事务系统维护了全局事务提交号200。

[0095] 在T2时刻,读事务R1启动,构建了它的视图Vision。认为所有事务提交号小于或等于200的事务都已经提交,而大于200的事务仍未提交。

[0096] 当读事务R1扫描到行记录rec_1,发现修改rec_1的事务的TCN是150。通过和Vision上记录的视图信息(200)比较,发现R1启动时,修改rec_1的事务已经提交,则可判断该行记录可见。当读事务R1扫描到行记录rec_1,发现修改rec_1的事务的TCN是250。通过和Vision上记录的视图信息(200)比较,发现R1启动时,修改rec_1的事务仍未提交,则可判断该行记录不可见。

[0097] 5、原生支持Native Flashback Query(数据库领域中一种特殊的查询,它可以指定过往的某个时间点,查找该时间点时的数据库数据),用户可以通过SELECT * FROM AS OF [TIMESTAMP | SCN] 任意查询过往的历史版本。

[0098] 6. 引入Undo(数据库数据的历史版本保存在Undo当中,通过undo,用户可以查询到数据过往的版本,也可以让数据回滚到某个历史版本) Reservation机制。允许MySQL能够从时间和空间两个维度上,控制Purge系统推进的程度,让数据的历史版本得以保留。

[0099] 以下进行详细说明:

[0100] 参照图1,示出了本发明一实施例提供的一种SQL数据库中分布式事务的处理的步骤流程图,SQL数据库可以为MySQL数据库,SQL数据库可以包括多个数据库节点,如多个MySQL实例。

[0101] 具体的,可以包括如下步骤:

[0102] 步骤101,在至少两个数据库节点发起分布式事务。

[0103] 在XA事务的基础上,可以在多个数据库中至少两个数据库节点中发起分布式事务,即同一个事务在该至少两个数据库节点中执行。

[0104] 步骤102,针对至少两个数据库节点中每个数据库节点,在分布式事务提交成功时,确定并持久化分布式事务的全局事务提交号,分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的。

[0105] 如前文所述,事务的可见性取决于事务提交成功的顺序,而XID是XA事务启动时分配的唯一标识,其和提交顺序没有关系,则本发明实施例可以在当前的分布式事务提交成功时,按照当前的分布式事务在所有分布式事务中提交成功的顺序,确定当前的分布式事务的全局事务提交号GCN,并可以将该全局事务提交号进行持久化存储。

[0106] 具体而言,可以通过以下两种方法在提交成功的时候指定外部的全局事务提交号GCN:

[0107] 1、SET SESSION innodb_commit_seq = [GCN];

[0108] XA COMMIT XID

[0109] 2、XA COMMIT XID $GCN

[0110] 在本发明一实施例中,分布式事务的全局事务提交号可以为单调递增的时序值,可以通过服务组件TSO(Timestamp Oracle)方案来保证分布式事务的时序。即存在一个服务组件TSO,TSO 授时服务可以保证按照递增的方式分配时间戳,任何一次申请得到的时间戳都不会重复,在分布式系统中用于给事件定序,最常见和重要的作用即保证事务版本号的单调递增,确保分布式事务的时序,不断产生递增的单调时序值,该值将会作为全局的分布式事务提交GCN持久化到各个MySQL节点中。

[0111] 步骤103,响应于针对SQL数据库的一致性查询,在当前构建的一致性视图的基础上,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定数据记录的可见性判断结果。

[0112] 其中,一致性视图可以为数据库的状态(包括数据、数据与数据之间的关系),数据库的状态会随着用户的操作而发生改变,在读事务发生时,需要拿到一个视图,来表示读事务观察到的当时的数据库的状态,一致性视图是通过一个全局事务提交号来表示(即:视图起来的时候,从全局事务提交号中拿当时的事务提交号构成自己)。

[0113] 由于全局事务提交号可以表征分布式事务提交成功的顺序,且在事务提交成功时进行了持久化,则在进行全局一致性查询时,在扫描到某个数据记录时,可以查询该数据记录对应的目标分布式事务的全局事务提交号,进而可以根据目标分布式事务的全局事务提交号,判断该数据记录是否可见,得到在当前构建的一致性视图下的可见性判断结果。

[0114] 在本发明一实施例中,当前构建的一致性视图可以对应有一第一全局事务提交号,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定数据记录的可见性判断结果,可以包括:

[0115] 在目标分布式事务的全局事务提交号小于或等于第一全局事务提交号的情况下,判定数据记录为可见状态;在目标分布式事务的全局事务提交号大于第一全局事务提交号的情况下,判定数据记录为不可见状态。

[0116] 在具体实现中,可以在MySQL节点上都采用Flashback Query的方式,即:用户可以任意指定GCN,查询所有全局事务提交号小于或等于该GCN的分布式事务带来的修改(需要说明的是,在数据记录存在多个历史版本的情况下,如undo链中存在GCN分别为200、100、80的分布式事务对应的历史版本,则当输入的全局事务提交号为GCN=120,则只会看到GCN为100的分布式事务对应的历史版本,而不是GCN为80的分布式事务对应的历史版本,即在数据记录存在多个历史版本的情况下,查询全局事务提交号小于或等于指定的GCN对应的最近一个历史版本),其提供了两种使用方式:

[0117] 1、SET SESSION innodb_snapshot_seq = [GCN]

[0118] 2、SELECT ... FROM tablename AS OF GCN

[0119] 其中,方式1可以通过改变innodb_snapshot_seq 这个会话级的变量,来指定分布式一致性查询所需要的GCN,方式2可以通过扩展的SQL(社区MySQL版本没有这样的语法)来指定分布式一致性查询所需要的GCN。

[0120] 基于此,一致性查询可以对应有一第一全局事务提交号,即指定的GCN,在判断数据记录是否可见时,可以判断数据记录对应的分布式事务的全局事务提交号是否小于或等于第一全局事务提交号的情况下,可以判定该数据记录为可见状态,在目标分布式事务的全局事务提交号大于第一全局事务提交号的情况下,可以判定该数据记录为不可见状态。

[0121] 在本发明实施例中,通过在至少两个数据库节点发起分布式事务,然后针对至少两个数据库节点中每个数据库节点,在分布式事务提交成功时,确定并持久化分布式事务的全局事务提交号,响应于针对SQL数据库的一致性查询,在当前构建的一致性视图的基础上,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定数据记录的可见性判断结果,实现了对SQL数据库中分布式一致性查询的优化,能够根据分布式事务的提交顺序来进行可见性判断,保证了一致性查询的准确性、高效性。

[0122] 参照图2,示出了本发明一实施例提供的另一种SQL数据库中分布式事务的处理的步骤流程图,具体可以包括如下步骤:

[0123] 步骤201,在至少两个数据库节点发起分布式事务。

[0124] 步骤202,针对至少两个数据库节点中每个数据库节点,在分布式事务提交过程中,持久化分布式事务的事务状态,并在分布式事务提交成功时,确定并持久化分布式事务的全局事务提交号,分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的。

[0125] 其中,分布式事务可以为基于两阶段提交(2PC,Two-phase Commit)协议进行提交,事务状态可以与分布式事务在两阶段提交协议中所处的阶段相对应,其可以包括未提交成功状态和提交成功状态,未提交成功状态即可以为2PC的第一阶段中的PREPARED状态。需要说明的是,未提交成功的事务状态有多种,如Active状态表示事务处于活跃状态。这样的事务带来的修改,只有本事务才可见。也就是说,自己的修改,自己必须能看见。否则,其它视图对于这样的都是不可见的。另外还有一阶段的状态,也就是说所谓的prepare状态,只有遇到这样的状态,才需要等待这个事务提交掉。

[0126] 具体的,MySQL在XA事务中采用2PC来保证事务的一致性,具体的,分布式系统中存在两种角色,一种是协调者,一种是参与者。

[0127] 事务提交的时候有两个阶段:

[0128] 第一阶段(也被称为Prepare阶段):

[0129] 1、协调者节点向所有参与者节点询问是否可以执行提交操作,并开始等待各参与者节点的响应。

[0130] 2、参与者节点执行完所有的操作后,将修改持久化掉。

[0131] 3、参与者节点根据具体情况,响应协调者是"同意"还是"中止"。

[0132] 第二阶段:

[0133] 1、协调者根据各个节点反馈,决定分布式事务是提交还是回滚。然后将这个决议下发给所有的节点。

[0134] 2、各个节点根据协调者的指令,对本事务进行提交或回滚操作。完成后向协调者反馈。

[0135] 3.、协调者收到所有节点的反馈后,最终该事务进入完结状态(提交或回滚)。

[0136] 在本发明实施例中,一个XA事务在数据库节点上执行的过程如下:

[0137] 1、XA START $XID: 开启一个事务,并将事务置于ACTIVE状态,同时分配一个Transaction Slot。

[0138] 2、XA END $XID: 将事务置于IDLE状态,表示事务内的SQL操作完成。

[0139] 3、XA PREPARE $XID: 2PC的第一个阶段,事务状态置于PREPARED状态,PREPARED状态表示事务已经完成2PC的第一个阶段。

[0140] 4、XA COMMIT $XID: 2PC的第二个阶段,从TSO获取GCN并持久化,事务最终提交。

[0141] 从上述过程可知,如果分布式一致性查询发现当前查询的行记录,所对应的事务处于PREPARED状态,该事务虽然处于提交的过程中,但事实上在这个节点上并没有完成提交,无法查询其对应的全局事务提交号(在2PC的第二阶段提交成功才确定GCN),则读视图无法判断出该记录是否可见。

[0142] 具体而言,如果认为该事务仍处于活跃状态,则判断为不可见,但这个分布式事务可能在别分布式节点上已经提交,并被查询出结果。这样会导致同一个查询在一个节点上能查询到结果,而在另外的节点上查询不到结果。如果认为该事务处于完结状态,而事实上该事务需要在2PC的第二个阶段才能获知本事务的外部提交号GCN。

[0143] 基于上述分析,可以得出结论:处于PREPARED状态的事务是无法进行可见性判断,需要进行进一步操作,不能直接进行可见性判断,则可以在分布式事务采用2PC进行提交过程中,根据其在两阶段提交协议中所处的阶段确定事务状态,进而可以对事务状态进行持

久化。

[0144] 步骤203,响应于针对SQL数据库的一致性查询,查询扫描的数据记录对应的目标分布式事务的事务状态;

[0145] 在进行全局的一致性查询时,由于对事务状态进行了持久化,则可以查询扫描的数据记录对应的目标分布式事务的事务状态。

[0146] 步骤204,在目标分布式事务的事务状态为提交成功状态时,查询目标分布式事务的全局事务提交号;

[0147] 在目标分布式事务的事务状态为提交成功状态时,即其已完成2PC中的第二阶段,则可以直接查询目标分布式事务的全局事务提交号。

[0148] 步骤205,在目标分布式事务的事务状态为未提交成功状态时,控制一致性查询进入等待状态,直至目标分布式事务的事务状态更新为提交成功状态时,查询目标分布式事务的全局事务提交号;

[0149] 在目标分布式事务的事务状态为未提交成功状态时,即处于2PC中的第一阶段,即处于PREPARED状态,则可以应用GP(Global Query) Wait机制,在事务锁的基础上,构建出GP Lock来表示阻塞关系,使得一致性查询进入等待状态,直到该PREPARE状态的事务完成2PC的第二阶段,即分布式事务提交成功,事务状态更新为提交成功状态,则可以查询目标分布式事务的全局事务提交号。

[0150] 步骤206,在当前构建的一致性视图的基础上,根据扫描的数据记录对应的目标分布式事务的全局事务提交号,确定数据记录的可见性判断结果。

[0151] 以下结合2PC来说明数据记录的可见性判断:

[0152] 假设。现在有两个分布式事务DT_A(包括DT_A1、DT_A2)、DT_B(包括DT_B1、DT_B2),它们的提交号分别是50和100。在某个时刻T1,TSO上的GCN生成器已经推高到110,而上面两个分布式事务在两个分布式节点上的状态分别是:

[0153] Node1:

[0154] DT_A1(已提交,GCN=50),DT_B1(已提交,GCN=100)

[0155] Node2:

[0156] DT_B2(已提交,GCN=100),DT_A2(PREPARED)。

[0157] 这个时候一个全局的一致性查询Q1发起,拿到的全局视图R1的第一全局事务提交号GCN=110。

[0158] 在Node1上,根据以GCN为核心的可见性判断机制。DT_A1与DT_B1的修改都被判断为可见。

[0159] 而在Node2上,对于DT_B2的修改,R1可以判断为可见(110 > 100)。但是对于DT_A2带来的修改,假设这些修改作用于行记录rec_1上,则:

[0160] 1、R1是没有办法知道这个修改是由DT_A这个分布式事务带来的,并且这个事务在全局视图下认为已经被提交了。

[0161] 2、在R1看来,这个事务处于PREPARED状态,即不是活跃状态(可以直接判定为不可见),也不是事务的终态(即提交)。所以这个时候必须要陷入GP Wait状态,等待这个事务最终提交。当DT_A2完成提交过程后,R1才能够知道DT_A2的全局事务提交号GCN(GCN=50)。此时,才能够判断为该事务带来的修改,对于R1来说是可见的。

[0162] 在本发明一实施例中,数据记录可以存储有用于指向目标分布式事务对应的事务槽的事务槽地址(TSA),还可以包括:

[0163] 在分布式事务启动时,在预置的事务表中分配分布式事务对应的事务槽,事务槽用于持久化分布式事务的全局事务提交号和分布式事务的事务状态。

[0164] 在具体实现,可以通过XA START $XID开启一个事务,并将事务置于ACTIVE状态,同时分配一个Transaction Slot,后续的分布式事务的全局事务提交号和分布式事务的事务状态都将存储在该Transaction Slot中。

[0165] 在一示例中,本地事务提交号(TCN)也可以持久化至该Transaction Slot中,本地事务提交号可以由数据库节点中的TCN生成器在本地事务提交时生成,并持久化。

[0166] 需要说明的是,分布式事务是由多个本地事务协同完成的,事务槽是属于本地的,如分布式事务需要在两个数据节点上完成,两个数据节点都需要完成各自本地事务,然后都成功提交掉后,这个分布式事务才算提交成功。事务槽是存在每个数据节点上的。

[0167] 在本发明一实施例中,还可以包括:

[0168] 当对目标事务槽进行复用时,判断目标事务槽对应的分布式事务是否已结束,并在目标事务槽对应的分布式事务结束的情况下,允许对目标事务槽进行复用。

[0169] 在具体实现中,可以将分布式事务的包括TCN、GCN等信息持久化到Transaction Table上。由于Transaction Table不能无限膨胀,所以Transaction Table引入了复用机制。然而,如果某个事务的Transaction Slot被复用掉,这个事务的事务状态信息将会丢失掉。

[0170] 为了让查询(包括全局一致性查询和本地查询)能够尽可能地找到行记录对应事务的真实的事务状态信息,可以将被复用的事务状态信息放到一个深度链表上。这意味着,GCN在这个深度链表也有一个时序。如果GCN在深度链表上是乱序的,则被查询的行记录,会获取到一个错误的事务状态信息,从而导致查询得到错误的结果集。

[0171] 针对这种情况,本方案通过控制Transaction Slot的复用机制来保证GCN在这个深度链表的有序性。Transaction Slot在事务启动的时候被分配,在事务完结后才允许被复用。Transaction Slot至少要在事务完结之后才允许被复用,所以能够保证GCN在深度链表上的有序性。

[0172] 在本发明一实施例中,还可以包括:

[0173] 在存在至少两个分布式事务修改同一个数据记录时,通过事务锁阻塞除在先进行修改的分布式事务外的其他分布式事务,直至在先进行修改的分布式事务获得全局事务提交号并提交。

[0174] 在具体实现中,MySQL的历史版本数据都会存在Undo表空间中。对于每一条行记录,它的所有历史版本都会按照先后顺序被组织成一个链表。行记录的最新版本在链表的头部,而行记录的最老记录在链表的末尾。从链表头部沿着链表走到链表的尾部,Trx ID、TCN是降序的。如果GCN在这个Undo链上是乱序的,则可能会导致分布式一致性查询查到错误的结果,因为分布式查询在每个MySQL节点上都采用Flashback Query的方式,而该方式需要不断遍历Undo链表,直到找到第一个历史版本数据(最近一个历史版本数据),它的持久化GCN号比Flashback Query指定的GCN号要小。

[0175] 基于此,本方案通过MySQL事务锁机制,能够保证GCN在Undo链上有正确的先后顺

序。假设事务A以及事务B都修改了同一个行记录，如果事务A先进行修改，则事务A能够持有该行记录的行锁，并且该行锁会一直持有直到事务A从TSO上拿到GCN并完结掉该事务。整个过程事务B都会被阻塞，所以事务B最后获取的GCN一定比事务A的大。

[0176] 在本发明一实施例中，还可以包括：

[0177] 在对数据记录的历史版本进行清理时，确定第二全局事务提交号；在当前历史版本对应的分布式事务的全局事务提交号小于第二全局事务提交号的情况下，对当前历史版本进行清理；在当前历史版本对应的分布式事务的全局事务提交号大于第二全局事务提交号的情况下，对第二全局事务提交号进行增大。

[0178] 在具体实现中，为了防止Undo表空间无限制膨胀，MySQL的Purge系统会按照一定的策略清理掉Undo表空间中历史版本数据。Purge系统会从最老的历史版本开始清理，为了保证Purge系统总是清理当前Undo表空间中最老的历史版本，MySQL引入了History List来组织所有历史事务的Undo段。History List上的每一个结点都是一个Undo段。每一个Undo段都存储了一个事务产生的所有历史版本数据。History List上的Undo段是按照事务提交的本地顺序来排序的。

[0179] 在分布式场景中，由于网络、负载、调度等原因，多个不冲突的分布式事务在各个分布式节点上提交的顺序可能是不同的。具体地说，对于某一个分布式节点，假设有两个分布式事务：Trx_A和Trx_B。在本地提交顺序上，Trx_A先于Trx_B提交，即TCN_A < TCN_B。然而在全局视角上，TSO认为事务B是先提交的，即会先给事务B分配GCN_B，所以GCN_A > GCN_B。也就是说，沿着History List，TCN是递增的，而GCN是乱序的。

[0180] 如果Flashback Query查询已经被Purge系统清理掉的历史版本，则会返回错误告知用户该历史版本已经被清理掉。对于基于TCN的Flashback Query，由于Purge系统是严格按照TCN递增的顺序清理历史版本，所以基于TCN的Flashback Query总是能够正确判断行记录的历史版本是否已经被Purge系统清理掉。

[0181] 然而，由于GCN在History List上是乱序的，同样的保证，对于基于GCN的Flashback Query，是无法得到满足的。为了解决该问题，本方案引入了Purged_GCN，即第二全局事务提交号，Purge系统每次沿着History List清理老事务产生的历史版本时，会回溯该事务的GCN信息，如果小于或等于当前Purged_GCN，则被Purge系统清理掉，如果比当前Purged_GCN要大，则推高Purged_GCN(即如果历史版本已经被清理，那么Purged_GCN就会被推高到至少比这个历史版本的全局提交号大)，并将该Purged_GCN持久化掉。

[0182] 这样，所有小于Purged_GCN的事务，都可能已经被Purge系统清理掉。显然，这种策略可能会导致有些事务仍在Undo当中未被实际清理，却被提前定性为Purged状态。但是，这样的方案能够彻底保障正确性。

[0183] 需要说明的是，历史版本的清理不一定是按照GCN的顺序清理的。例如，可能是按照GCN为5、4、6的顺序清理，当5的被清理后，Purged_GCN就会被推高到5，当6被清理，就会被推高。

[0184] 总体而言，本发明实施例可以实现以下：

[0185] 1、通过引入全局事务提交号GCN，将GCN持久化到Transaction Slot。

[0186] 2、在Native Flashback Query的基础上，打造了基于GCN的Flashback Query方案。在分布式一致性查询中，所有在分布式节点上的分布式查询都是基于GCN的Flashback

Query。

[0187] 3、保证了GCN在核心结构上的正确性以及有效性，具体如下：

[0188] A、借助事务锁机制保证Undo链上的GCN的有序性。

[0189] B、控制Transaction Slot的复用机制来保证GCN在深度链表的有序性。

[0190] C、引入了Purged_GCN机制，解决了GCN在History List上乱序的问题，保证查询不会错误查找到已经被清理的历史版本。

[0191] 4、引入了GP Lock机制，解决了处于PREPARE状态的事务的可见性问题，保证查询能够拿到正确事务状态信息。

[0192] 5、生成持久化的全局单调递增号，并用于TSO生成全局事务提交号GCN。

[0193] 以下结合图3和表1进行示例性说明：

[0194] 如图3所示，存在两个数据库节点Node1和Node2，每个数据库节点存在Trx ID生成器和TCN生成器，Trx ID生成器在事务开始(Begin)时生成TRX ID，TCN生成器在事务提交时在事务表(Transaction Table)持久化TRX ID、TCN，数据记录的格式(Record Format)可以包括PK(Primary Key，MySQL InnoDB数据表的主键字段，表上的每一个数据行都有唯一的PK)、TCN、TSA、Trx ID、Rollptr、User Cols。

[0195] 在外部可以设置以TSO组件，其携带有GCN生成器，可以在生成GCN并在事务提交时将GCN持久化至事务表。

[0196] 如表1所示为一分布式事务的执行过程：

[0197]

时间点	TSO	Node1	Node2
T1	GCN=500		
T2	发起分布式事务 DT_A，DT_B		
T3		会话1: XA BEGIN "DT_A1" 分配事务槽N1_TS1，以及Trx ID为100	会话1: XA BEGIN "DT_A2" 分配事务槽N2_TS1，以及Trx ID为25
		会话2: XA BEGIN "DT_B1" 分配事务槽N1_TS2，以及Trx ID为101	会话2: XA BEGIN "DT_B2" 分配事务槽N2_TS2，以及Trx ID为26

[0198]

T4	分发数据操作	会话1: UPDATE T1 SET COL_1=15 WHERE PK=10; 在PK=10的行记录上，修改为行记录的新版本，并在TSA字段上指向了事务槽N1_TS1	会话1: UPDATE T2 SET COL_1=150 WHERE PK=100; 在PK=100的行记录上，修改为行记录的新版本，并在TSA字段上指向了事务槽N2_TS1
		会话2: UPDATE T1 SET COL_1=25 WHERE PK=20; 在PK=20的行记录上，修改为行记录的新版本，并在TSA字段上指向了事务槽N1_TS2	会话2: UPDATE T2 SET COL_1=250 WHERE PK=200; 在PK=200的行记录上，修改为行记录的新版本，并在TSA字段上指向了事务槽N2_TS2
T5	数据操作完结	XA END "DT_A1"	XA END "DT_A2"
		XA END "DT_B1"	XA END "DT_B2"

[0199]

T6	发起一阶段	XA PREPARE "DT_A1" 在事务槽N1_TS1上持久化事务状态为PREPARED状态	XA PREPARE "DT_A2" 在事务槽N2_TS1上持久化事务状态为PREPARED状态
		XA PREPARE "DT_B1" 在事务槽N1_TS2上持久化事务状态为PREPARED状态	XA PREPARE "DT_B2" 在事务槽N2_TS2上持久化事务状态为PREPARED状态
T7	完成DT_A的一阶段， 发起DT_A的二阶段， GCN推高为501	XA COMMIT "DT_A1" 在事务槽N1_TS1上持久化事务状态为COMMIT状态，以及GCN为501	
T8	完成DT_B的一阶段， 发起DT_B的二阶段， GCN推高为502	XA COMMIT "DT_B1" 在事务槽N1_TS2上持久化事务状态为COMMIT状态，以及GCN为502	XA COMMIT "DT_B2" 在事务槽N2_TS2上持久化事务状态为COMMIT状态，以及GCN为502

[0200]

T9	发起了全局分布式一致性查询Q1 ("SELECT XXX AS OF GCN 502")，并构建了全局一致性视图R1（GCN=502） 该查询在Node1上： 扫描到PK=10的记录，通过TSA回查事务槽N1_TS1，发现对应的事务DT_A1已经处于已提交状态，并且GCN=501。通过朴素的数值比较（501<=502），就能够判断PK=10这一个行记录，对于R1来说是可见的。 扫描到PK=20的记录，同样的，该行记录对于R1来说也是可见的。 该查询在Node2上： 扫描到PK=200的记录，发现对于R1来说，该行记录可见。 扫描到PK=100的记录，发现该事务处于PREPARED状态，无法马上判断出可见性。陷入GP Wait等待状态。		
T10			XA COMMIT "DT_A2" 在事务槽N2_TS1上持久化事务状态为COMMIT状态，以及GCN为501
T11	Q1 的GP Wait等待结束在Node2上，对于R1来说，PK=100的行记录是可见的。至此，Q1返回查询的结果。		

[0201] 需要说明的是，对于方法实施例，为了简单描述，故将其都表述为一系列的动作组合，但是本领域技术人员应该知悉，本发明实施例并不受所描述的动作顺序的限制，因为依据本发明实施例，某些步骤可以采用其他顺序或者同时进行。其次，本领域技术人员也应该知悉，说明书中所描述的实施例均属于优选实施例，所涉及的动作并不一定是本发明实施例所必须的。

[0202] 参照图4，示出了本发明一实施例提供的一种SQL数据库中分布式事务的处理装置的结构示意图，SQL数据库可以包括多个数据库节点。

[0203] 具体的，可以包括如下模块：

[0204] 分布式事务发起模块401，用于在至少两个数据库节点发起分布式事务。

[0205] 全局事务提交号持久化模块402，用于针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号，分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的。

[0206] 一致性查询模块403，用于响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果。

[0207] 在本发明一实施例中，当前构建的一致性视图对应有一第一全局事务提交号，一致性查询模块403，可以包括：

[0208] 可见状态判定子模块，用于在目标分布式事务的全局事务提交号小于或等于第一全局事务提交号的情况下，判定数据记录为可见状态。

[0209] 不可见状态判定子模块，用于在目标分布式事务的全局事务提交号大于第一全局

事务提交号的情况下,判定数据记录为不可见状态。

[0210] 在本发明一实施例中,分布式事务的全局事务提交号为单调递增的时序值。

[0211] 在本发明一实施例中,还可以包括:

[0212] 事务状态持久化模块,用于在分布式事务提交过程中,持久化分布式事务的事务状态。

[0213] 在本发明一实施例中,还可以包括:

[0214] 事务状态查询模块,用于查询扫描的数据记录对应的目标分布式事务的事务状态。

[0215] 第一全局事务提交查询模块,用于在目标分布式事务的事务状态为提交成功状态时,查询目标分布式事务的全局事务提交号。

[0216] 第二全局事务提交查询模块,用于在目标分布式事务的事务状态为未提交成功状态时,控制一致性查询进入等待状态,直至目标分布式事务的事务状态更新为提交成功状态时,查询目标分布式事务的全局事务提交号。

[0217] 在本发明一实施例中,分布式事务为基于两阶段提交协议进行提交,事务状态与分布式事务在两阶段提交协议中所处的阶段相对应。

[0218] 在本发明一实施例中,数据记录存储有用于指向目标分布式事务对应的事务槽的事务槽地址,还可以包括:

[0219] 分配事务槽模块,用于在分布式事务启动时,在预置的事务表中分配分布式事务对应的事务槽,事务槽用于持久化分布式事务的全局事务提交号和分布式事务的事务状态。

[0220] 在本发明一实施例中,还可以包括:

[0221] 事务槽复用模块,用于当对目标事务槽进行复用时,判断目标事务槽对应的分布式事务是否已结束,并在目标事务槽对应的分布式事务结束的情况下,允许对目标事务槽进行复用。

[0222] 在本发明一实施例中,还可以包括:

[0223] 事务锁阻塞模块,用于在存在至少两个分布式事务修改同一个数据记录时,通过事务锁阻塞除在先进行修改的分布式事务外的其他分布式事务,直至在先进行修改的分布式事务获得全局事务提交号并提交。

[0224] 在本发明一实施例中,还可以包括:

[0225] 第二全局事务提交号确定模块,用于在对数据记录的历史版本进行清理时,确定第二全局事务提交号。

[0226] 当前历史版本清理模块,用于在当前历史版本对应的分布式事务的全局事务提交号小于第二全局事务提交号的情况下,对当前历史版本进行清理。

[0227] 第二全局事务提交号增大模块,用于在当前历史版本对应的分布式事务的全局事务提交号大于第二全局事务提交号的情况下,对第二全局事务提交号进行增大。

[0228] 参照图5,示出了本发明一实施例提供的一种SQL数据库中分布式事务的处理系统的结构示意图,SQL数据库可以包括多个数据库节点,处理系统可以包括服务组件501和处理模块502。

[0229] 服务组件501,用于在SQL数据库中分布式事务提交成功时,生成分布式事务的全

局事务提交号。

[0230] 处理模块502，用于在SQL数据库中至少两个数据库节点发起分布式事务的情况下，针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号；响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果；其中，分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的。

[0231] 在本发明一实施例中，当前构建的一致性视图对应有一第一全局事务提交号，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果，可以包括：

[0232] 在目标分布式事务的全局事务提交号小于或等于第一全局事务提交号的情况下，判定数据记录为可见状态。

[0233] 在目标分布式事务的全局事务提交号大于第一全局事务提交号的情况下，判定数据记录为不可见状态。

[0234] 在本发明一实施例中，分布式事务的全局事务提交号为单调递增的时序值。

[0235] 在本发明一实施例中，处理模块502还可以用于：

[0236] 在分布式事务提交过程中，持久化分布式事务的事务状态。

[0237] 在本发明一实施例中，处理模块502还可以用于：

[0238] 查询扫描的数据记录对应的目标分布式事务的事务状态。

[0239] 在目标分布式事务的事务状态为提交成功状态时，查询目标分布式事务的全局事务提交号。

[0240] 在目标分布式事务的事务状态为未提交成功状态时，控制一致性查询进入等待状态，直至目标分布式事务的事务状态更新为提交成功状态时，查询目标分布式事务的全局事务提交号。

[0241] 在本发明一实施例中，分布式事务为基于两阶段提交协议进行提交，事务状态与分布式事务在两阶段提交协议中所处的阶段相对应。

[0242] 在本发明一实施例中，数据记录存储有用于指向目标分布式事务对应的事务槽的事务槽地址，处理模块502还可以用于：

[0243] 在分布式事务启动时，在预置的事务表中分配分布式事务对应的事务槽，事务槽用于持久化分布式事务的全局事务提交号和分布式事务的事务状态。

[0244] 在本发明一实施例中，处理模块502还可以用于：

[0245] 当对目标事务槽进行复用时，判断目标事务槽对应的分布式事务是否已结束，并在目标事务槽对应的分布式事务结束的情况下，允许对目标事务槽进行复用。

[0246] 在本发明一实施例中，处理模块502还可以用于：

[0247] 在存在至少两个分布式事务修改同一个数据记录时，通过事务锁阻塞除在先进行修改的分布式事务外的其他分布式事务，直至在先进行修改的分布式事务获得全局事务提交号并提交。

[0248] 在本发明一实施例中，处理模块502还可以用于：

[0249] 在对数据记录的历史版本进行清理时，确定第二全局事务提交号。

[0250] 在当前历史版本对应的分布式事务的全局事务提交号小于或等于第二全局事务提交号的情况下,对当前历史版本进行清理。

[0251] 在当前历史版本对应的分布式事务的全局事务提交号大于第二全局事务提交号的情况下,对第二全局事务提交号进行增大。

[0252] 本发明一实施例还提供了一种电子设备,可以包括处理器、存储器及存储在存储器上并能够在处理器上运行的计算机程序,计算机程序被处理器执行时实现如上SQL数据库中分布式事务的处理方法。

[0253] 本发明一实施例还提供了一种计算机可读存储介质,计算机可读存储介质上存储计算机程序,计算机程序被处理器执行时实现如上SQL数据库中分布式事务的处理方法。

[0254] 对于装置实施例而言,由于其与方法实施例基本相似,所以描述的比较简单,相关之处参见方法实施例的部分说明即可。

[0255] 本说明书中的各个实施例均采用递进的方式描述,每个实施例重点说明的都是与其他实施例的不同之处,各个实施例之间相同相似的部分互相参见即可。

[0256] 本领域内的技术人员应明白,本发明实施例可提供为方法、装置、或计算机程序产品。因此,本发明实施例可采用完全硬件实施例、完全软件实施例、或结合软件和硬件方面的实施例的形式。而且,本发明实施例可采用在一个或多个其中包含有计算机可用程序代码的计算机可用存储介质(包括但不限于磁盘存储器、CD-ROM、光学存储器等)上实施的计算机程序产品的形式。

[0257] 本发明实施例是参照根据本发明实施例的方法、终端设备(系统)、和计算机程序产品的流程图和/或方框图来描述的。应理解可由计算机程序指令实现流程图和/或方框图中的每一流程和/或方框、以及流程图和/或方框图中的流程和/或方框的结合。可提供这些计算机程序指令到通用计算机、专用计算机、嵌入式处理机或其他可编程数据处理终端设备的处理器以产生一个机器,使得通过计算机或其他可编程数据处理终端设备的处理器执行的指令产生用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的装置。

[0258] 这些计算机程序指令也可存储在能引导计算机或其他可编程数据处理终端设备以特定方式工作的计算机可读存储器中,使得存储在该计算机可读存储器中的指令产生包括指令装置的制造品,该指令装置实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能。

[0259] 这些计算机程序指令也可装载到计算机或其他可编程数据处理终端设备上,使得在计算机或其他可编程终端设备上执行一系列操作步骤以产生计算机实现的处理,从而在计算机或其他可编程终端设备上执行的指令提供用于实现在流程图一个流程或多个流程和/或方框图一个方框或多个方框中指定的功能的步骤。

[0260] 尽管已描述了本发明实施例的优选实施例,但本领域内的技术人员一旦得知了基本创造性概念,则可对这些实施例做出另外的变更和修改。所以,所附权利要求意欲解释为包括优选实施例以及落入本发明实施例范围的所有变更和修改。

[0261] 最后,还需要说明的是,在本文中,诸如第一和第二等之类的关系术语仅仅用来将一个实体或者操作与另一个实体或操作区分开来,而不一定要求或者暗示这些实体或操作之间存在任何这种实际的关系或者顺序。而且,术语“包括”、“包含”或者其任何其他变体意

在涵盖非排他性的包含，从而使得包括一系列要素的过程、方法、物品或者终端设备不仅包括那些要素，而且还包括没有明确列出的其他要素，或者是还包括为这种过程、方法、物品或者终端设备所固有的要素。在没有更多限制的情况下，由语句“包括一个……”限定的要素，并不排除在包括所述要素的过程、方法、物品或者终端设备中还存在另外的相同要素。

[0262] 以上对所提供的SQL数据库中分布式事务的处理方法、装置及系统，进行了详细介绍，本文中应用了具体个例对本发明的原理及实施方式进行了阐述，以上实施例的说明只是用于帮助理解本发明的方法及其核心思想；同时，对于本领域的一般技术人员，依据本发明的思想，在具体实施方式及应用范围上均会有改变之处，综上所述，本说明书内容不应理解为对本发明的限制。

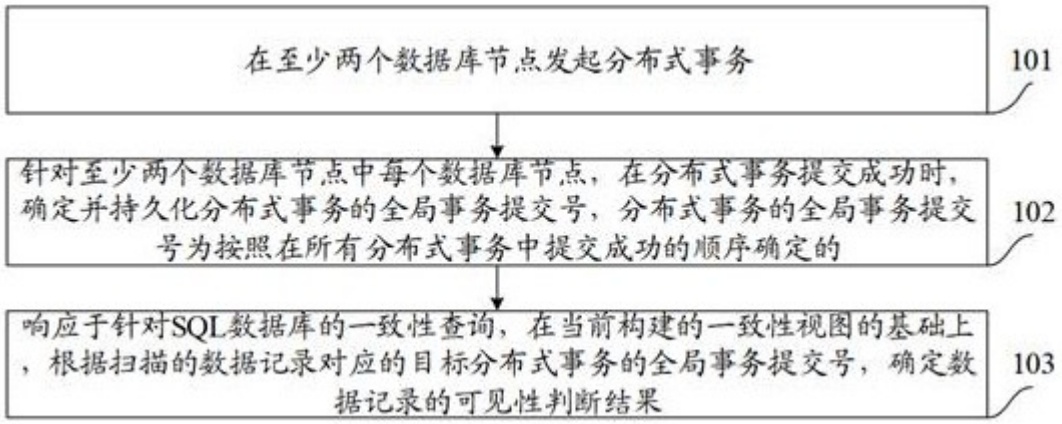

在至少两个数据库节点发起分布式事务
101
针对至少两个数据库节点中每个数据库节点，在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号，分布式事务的全局事务提交号为按照在所有分布式事务中提交成功的顺序确定的
102
响应于针对SQL数据库的一致性查询，在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果
103

图1

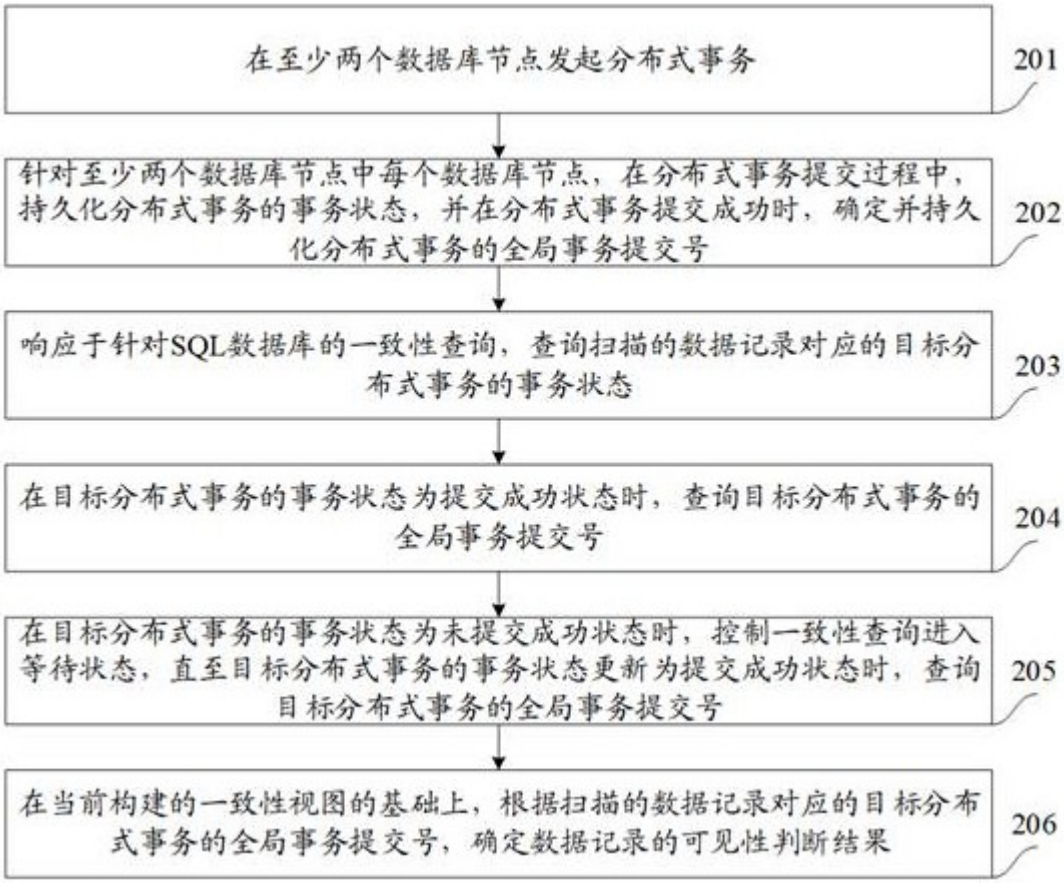

在至少两个数据库节点发起分布式事务
201
针对至少两个数据库节点中每个数据库节点，在分布式事务提交过程中，持久化分布式事务的事务状态，并在分布式事务提交成功时，确定并持久化分布式事务的全局事务提交号
202
响应于针对SQL数据库的一致性查询，查询扫描的数据记录对应的目标分布式事务的事务状态
203
在目标分布式事务的事务状态为提交成功状态时，查询目标分布式事务的全局事务提交号
204
在目标分布式事务的事务状态为未提交成功状态时，控制一致性查询进入等待状态，直至目标分布式事务的事务状态更新为提交成功状态时，查询目标分布式事务的全局事务提交号
205
在当前构建的一致性视图的基础上，根据扫描的数据记录对应的目标分布式事务的全局事务提交号，确定数据记录的可见性判断结果
206

图2

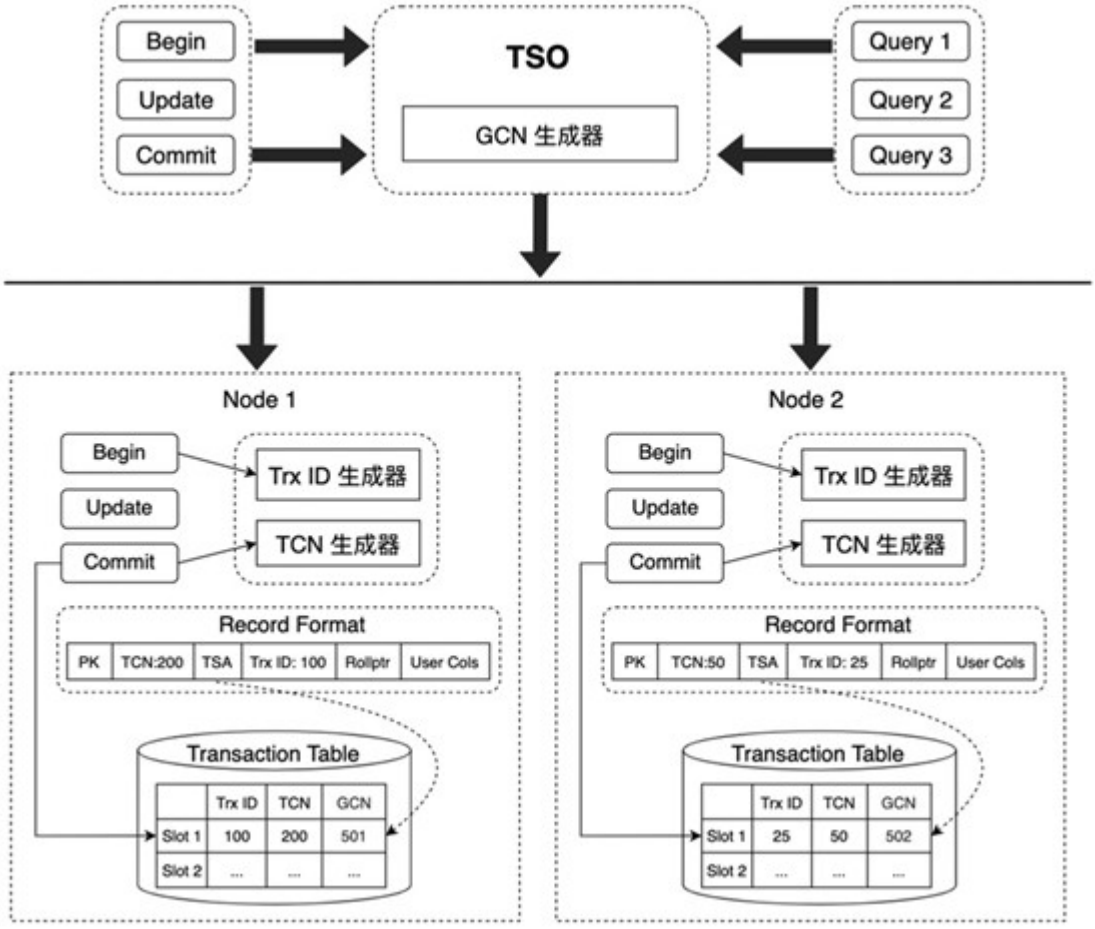

Begin
Update
Commit
TSO
GCN 生成器
Query 1
Query 2
Query 3
Node 1
Begin
Update
Commit
Trx ID 生成器
TCN 生成器
Record Format
PK
TCN:200
TSA
Trx ID: 100
Rollptr
User Cols
Transaction Table
Trx ID
TCN
GCN
Slot 1
100
200
501
Slot 2
...
Node 2
Begin
Update
Commit
Trx ID 生成器
TCN 生成器
Record Format
PK
TCN:50
TSA
Trx ID: 25
Rollptr
User Cols
Transaction Table
Trx ID
TCN
GCN
Slot 1
25
50
502
Slot 2
...

图3


分布式事务发起模块401
全局事务提交号持久化模块402
一致性查询模块403

图4

图5