



(12) 发明专利

(10) 授权公告号 CN 103814362 B

(45) 授权公告日 2016. 06. 15

(21) 申请号 201280045914. 9

(51) Int. Cl.

(22) 申请日 2012. 07. 20

G06F 12/00(2006. 01)

(30) 优先权数据

2011-218145 2011. 09. 30 JP

(56) 对比文件

(85) PCT国际申请进入国家阶段日

2014. 03. 20

US 5408653 A, 1995. 04. 18,
US 2006149701 A1, 2006. 07. 06,
JP 2009525536 A, 2009. 07. 09,
CN 101080714 A, 2007. 11. 28,
CN 102165420 A, 2011. 08. 24,

(86) PCT国际申请的申请数据

PCT/JP2012/068414 2012. 07. 20

审查员 周丹丹

(87) PCT国际申请的公布数据

W02013/046883 JA 2013. 04. 04

(73) 专利权人 国际商业机器公司

地址 美国纽约阿芒克

(72) 发明人 堀井洋

(74) 专利代理机构 北京市金杜律师事务所

11256

代理人 鄢迅 李峥宇

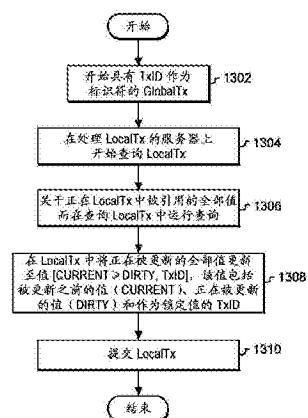
权利要求书2页 说明书13页 附图17页

(54) 发明名称

用于分布式 KVS 系统的处理方法和系统

(57) 摘要

为了在当在另一全局事务中正在被锁定的映像条目在一个本地事务中被查询和更新时,本地事务被提升至分布式 KVS 系统中的全局事务,执行如下步骤:在一个步骤中,在客户端计算机中事务 ID 被确定并且全局事务被开始;在一个步骤中,在多个服务器中的正在处理本地事务的一个服务器,查询本地事务被开始;在一个步骤中,在本地事务查询中正在被查询的全部值在查询本地事务中被查询;在一个步骤中,在本地事务中正在被更新的全部值被分别更新至更新前的值、更新期间的值、使用事务 ID 作为数据表中的值的锁定值;以及在一个步骤中,本地事务被提交。



1. 一种用于分布式KVS系统的处理方法,所述方法具有由所述分布式KVS系统在用于将数据分布至多个服务器的所述分布式KVS系统中执行的步骤,所述数据在所述多个服务器上从客户端计算机被访问,所述方法包括:

在所述多个服务器的每个服务器上放置管理表和数据表的步骤,所述管理表包括事务ID和指示其状态的值,所述数据表包括键值、值和锁定值;所述键值是所述数据表的KEY列中的内容,所述值是所述数据表的VALUE列中的内容,所述锁定值是所述数据表的LOCK列中的内容;

确定所述客户端计算机上的事务ID以开始全局事务的步骤;

在所述多个服务器中在处理本地事务的一服务器上开始查询本地事务的步骤;

在所述查询本地事务中运行关于在所述本地事务中被引用的全部值的查询的步骤;

将所述本地事务中正在被更新的全部值更新至被更新之前的值、正在被更新的值、以及作为所述数据表上的锁定值的所述事务ID的组合的步骤;以及

提交所述本地事务的步骤。

2. 根据权利要求1所述的处理方法,其中所述查询本地事务在提交所述本地事务的点处没有被提交。

3. 根据权利要求1所述的处理方法,其中所述锁定具有:S锁定,指示正在由全局事务或者本地事务引用;LX锁定,指示正在由本地事务更新;以及GX锁定,指示正在由全局事务更新。

4. 根据权利要求1所述的处理方法,其中所述开始所述全局事务的步骤包括:

使得所述客户端计算机生成所述事务ID的步骤;

基于所述事务ID确定保持表示所述全局事务的状态的管理映像的服务器的步骤;以及

针对表示所述全局事务的所述状态的所述管理映像上的映像条目开始本地事务以获取锁定的步骤。

5. 根据权利要求4所述的处理方法,进一步包括:

使用所述本地事务以将表示所述全局事务的所述状态的所述管理映像上的所述映像条目的值改变至被提交、并且提交的步骤;以及

将正在被更新的数据改变至在所述全局事务中被更新的全部映像条目的当前数据的步骤,以及执行删除正在被更新的数据以及事务ID以终止所述全局事务的步骤。

6. 根据权利要求4所述的处理方法,进一步包括:

使用所述本地事务以将表示所述全局事务的所述状态的所述管理映像上的所述映像条目的值改变至被回滚、并且提交的步骤;以及

在所述全局事务中被更新的全部映像条目上执行删除正在被更新的数据以及事务ID以终止所述全局事务的步骤。

7. 一种用于分布式KVS系统的处理系统,所述分布式KVS系统将数据分布至多个服务器,所述数据在所述多个服务器上从客户端计算机被访问,所述系统包括:

在所述多个服务器的每个服务器上放置管理表和数据表的装置,所述管理表包括事务ID和指示其状态的值,所述数据表包括键值、值和锁定值;所述键值是所述数据表的KEY列中的内容,所述值是所述数据表的VALUE列中的内容,所述锁定值是所述数据表的LOCK列中的内容;

确定所述客户端计算机上的事务ID以开始全局事务的装置；
在所述多个服务器中在处理本地事务的一服务器上开始查询本地事务的装置；
在所述查询本地事务中运行关于在所述本地事务中被引用的全部值的查询的装置；
将所述本地事务中正在被更新的全部值更新至被更新之前的值、正在被更新的值、以及作为所述数据表上的锁定值的所述事务ID的组合的装置；以及
提交所述本地事务的装置。

8. 根据权利要求7所述的处理系统，其中所述查询本地事务在提交所述本地事务的点处没有被提交。

9. 根据权利要求7所述的处理系统，其中所述锁定具有：S锁定，指示正在由全局事务或者本地事务引用；LX锁定，指示正在由本地事务更新；以及GX锁定，指示正在由全局事务更新。

10. 根据权利要求7所述的处理系统，其中所述开始所述全局事务的装置包括：
使得所述客户端计算机生成所述事务ID的装置；
基于所述事务ID确定表示所述全局事务的状态的管理映像的服务器的装置；以及
针对表示所述全局事务的所述状态的所述管理映像上的映像条目开始本地事务以获取锁定的装置。

11. 根据权利要求10所述的处理系统，进一步包括：
使用所述本地事务以将表示所述全局事务的所述状态的所述管理映像上的所述映像条目的值改变至被提交、并且提交的装置；以及
将正在被更新的数据改变至所述全局事务中被更新的全部映像条目的当前数据的装置，以及执行删除正在被更新的数据以及事务ID以终止所述全局事务的装置。

12. 根据权利要求10所述的处理系统，进一步包括：
使用所述本地事务以将表示所述全局事务的所述状态的所述管理映像上的所述映像条目的值改变至被回滚、并且提交的装置；以及
在所述全局事务中被更新的全部映像条目上执行删除正在被更新的数据以及事务ID以终止所述全局事务的装置。

用于分布式KVS系统的处理方法和系统

技术领域

[0001] 本发明涉及分布式处理系统,具体地,涉及在分布式数据库系统上的事务处理,以及更具体地,涉及在键-值存储(在下文中缩写为KVS)系统中的事务处理。

背景技术

[0002] 分布式数据库系统是广泛已知的。例如,日本专利申请公开号2007-188518公开了使用所有权分组的分布式数据库系统,其中改变指示数据项所有权的数据的步骤是原子操作。

[0003] 分布式数据库系统典型地实现关系数据库并且使用诸如SQL的查询语法。

[0004] 最近,已经使用了被称为键-值存储(KVS)的数据库管理软件来通过将键与值相关联来写入值,并且通过指定与值相关联的键来读取值。根据服务器的数量,简单的接口特征产生读取和写入值的高的吞吐率和高的可伸缩性。由此,还已经实现了支持将数据分布至多个服务器的分布式的KVS。

[0005] 在分布式数据库系统中,分布式事务通常使用两个阶段的提交(commit)被处理。事务状态由每个资源管理器和事务监视器来管理,以实现跨越多个分布式资源的事务。然而,如果此类机制被引入到KVS,则KVS的简单属性将会丢失,导致损害管理方便性和可伸缩性。由此,优选地向分布式KVS应用一种用于使用分布式锁定管理器的技术以实现全局事务,如在日本专利申请公开(PCT申请的译文)No.2009-525536中所公开。由此,在普通的分布式KVS中,需要客户端仅请求每个服务器中的事务(本地事务),而用于由多个服务器管理的数据的事务应当被处理,以通过组合本地事务而实现分布式事务(全局事务)。

[0006] 然而,在简单实现的分布式事务KVS中,没有实现全局事务。例如,当一个客户端计算机针对两个本地事务向两个服务器做出请求以构成一个全局事务时,如果在服务器上提交本地事务之一后在客户端计算机中出现故障,则不能确定在服务器上的另一本地事务可以被提交。

[0007] 由此,在Slim3on Google App Engine for Java:Development of cloud applications with Slim3,Yasuo Higa和Shinich Ogawa,Shuwa System Co.Ltd., pp.241-251中公开了在Google应用引擎上以本地事务来协调全局事务的方法。在此方法中,在KVS上,管理映像(map)被定义为用以管理全部全局事务的专用映像,并且数据映像通过应用定义为映像,以便不但存储被提交的值、还存储正在利用更新全局事务的ID被更新的脏值。管理映像作为两阶段的提交机制中的事务监视器管理哪个全局事务被提交或者未被提交,并且数据映像作为两阶段的提交机制中资源管理而管理哪个该数据准备被提交,由此它们在仅支持本地事务的分布式KVS上实现两阶段提交。数据操作的并发性由数据映像中的事务ID和管理映像中的全局事务状态来控制。换言之,并发性控制机制中,由KVS提供的用于本地事务的并发性控制机制(本地并发性控制机制)从未被使用。

[0008] 当分布式KVS上的全局事务由此类传统已知的技术来实现时,全局事务和本地事务不能混合,这是因为用于本地事务的并发性控制机制不能与用于全局事务的并发性控制

机制一起工作。例如,当客户端计算机正在利用协调全局事务更新由两个服务器管理的值以自动更新它们更新时,其他客户端可以读取并更新在本地事务中正在被更新的值,这是因为用于全局事务的并发性控制不能从KVS上的服务器的本地并发性控制机制获取任何锁定。

[0009] 由此,即使传统已知技术中的可以以本地事务进行的处理,也需要由全局事务来执行。由于全局事务的开销大于本地事务,存在的降低处理速度的问题。

[0010] [现有技术参考文献]

[0011] [专利文献]

[0012] [专利文档1]

[0013] 日本专利申请公开号2007-188518。

[0014] [专利文档2]

[0015] 日本专利申请(PCT申请的译文)公开号2009-525536。

[0016] [非专利文献]

[0017] [非专利文献1]

[0018] Slim3on Google App Engine for Java:Development of cloud applications with Slim3, Yasuo Higa and Shinich Ogawa, Shuwa System Co.Ltd., pp.241-251.

发明内容

[0019] [将由本发明解决的问题]

[0020] 本发明的目的在于使得用于本地事务的并发性控制机制与用于全局事务的并发性控制机制一起工作,以便实现高速的本地事务来用于更新和运行关于在一个服务器上的映像条目的查询,同时实现分布式事务(全局事务)用于更新和运行在仅支持本地事务的分布式KVS系统中的多个服务器上的映像条目的查询。

[0021] 用于解决问题的方式

[0022] 本发明通过在本地事务和全局事务之间以如下方式实现并发性控制来解决上述问题:在作为由本地并发性控制机制允许的本地事务的针对映像条目的查询/更新处理期间,检查将被引用和更新的映像条目是否正在作为另一全局事务被更新,并且当其正在被更新时,过程中的本地事务被升级至全局事务,以将后续的处理切换至通过全局事务(全局并发性控制机制)来使用并发性控制的处理。

[0023] 换言之,本发明的系统在另一本地事务中重新获取,在本地事务处理期间由本地并非控制机制获取的读取锁定(共享锁定),并且在现有本地事务中修改所获取的映像条目以将映像条目的写入锁定(排他锁定)改变至针对全局事务的,从而切换至全局并发性控制机制的写入锁定,由此升级至并发性控制。

[0024] 就此,本发明的系统将具有全局事务的事务ID及其状态的配对的管理映像分布至每个服务器作为KVS上的映像,以管理全局事务的状态。通过以此方式放置管理映像,在本地事务被升级到全局事务之后,在本地事务之间、在全局事务之间、在本地事务和全局事务之间的读写冲突通过本地并发性控制机制来解决,在全局事务之间的写写冲突通过使用管理映像来解决,在本地事务和全局事务之间的写写冲突通过本地并发性控制机制来解决,在全局事务和本地事务之间的写读冲突和写写冲突通过使用管理映像来解决。

[0025] 根据本发明,KVS事务机制被用于针对每个映像条目管理四个状态,即,S(获取读取锁定并且正在被全局事务或者本地事务引用),Init(没有访问),LX(获取写入锁定并且正在被本地事务更新)以及GX(获取写入锁定并且正在被全局事务更新)。

[0026] 继而,根据本发明,在尝试针对特定映像条目获取S锁定(LX锁定)时,当认识到由另一全局事务针对该映像条目的锁定是GX时,开始从本地事务升级至全局事务的处理。

[0027] 在第一步骤中,开始执行针对全局事务的处理以首先确定事务ID。

[0028] 接着,更新前的值CURRENT、正在被更新的值DIRTY、以及生成的事务ID作为值来被插入到当前正在针对其获取LX锁定的全部映像条目中。

[0029] 接着,针对目前正在针对其获取S锁定的全部映像条目而开始另一本地事务,以冗余地获取S锁定。因为在原始本地事务中已经获取了S锁定,这总是成功地执行。

[0030] 接着,原始本地事务被提交。作为结果,LX锁定被升级至GX锁定。

[0031] 接着,过程等待已经获取GX的全局事务的终止。

[0032] 在作为等待目标的全局事务的终止之后,S锁定(GX锁定)的获取被尝试为全局事务。

[0033] [本发明的优势]

[0034] 根据本发明,提供了根据环境而将本地事务升级至全局事务的功能,以实现高速的本地事务,用于更新和运行关于在KVS系统中的一个服务器上的映像条目的查询,同时实现全局事务,用于更新和运行关于多个服务器上的映像条目的查询。

附图说明

[0035] 图1是用于执行本发明的整体系统的示意图;

[0036] 图2是客户端计算机的硬件的示意框图;

[0037] 图3是服务器的硬件的示意框图;

[0038] 图4是客户端计算机和服务器的功能框图;

[0039] 图5是示出传统KVS系统的概要的框图;

[0040] 图6是由传统KVS系统执行的处理的示例的框图;

[0041] 图7是由用于执行全局事务的传统KVS系统执行的处理的示例的框图;

[0042] 图8是示出根据本发明的KVS系统的概要的图示;

[0043] 图9是示出锁定的状态转换的图示;

[0044] 图10是示出当事务第一次执行查询/更新处理时的处理的流程图;

[0045] 图11是示出当事务执行第二次和后续的查询/更新处理时的处理的流程图;

[0046] 图12是示出用于全局事务的开始处理的流程图;

[0047] 图13是示出从本地事务向全局事务升级的处理的流程图;

[0048] 图14是示出全局事务中的查询处理的流程图;

[0049] 图15是示出在全局事务中的更新处理的流程图;

[0050] 图16是示出在全局事务中的提交处理的流程图;

[0051] 图17是示出用于全局事务的等待处理的流程图;

[0052] 图18是示出用于全局事务的终止处理的流程图;以及

[0053] 图19是示出根据本发明的KVS系统的操作的示例的图示。

具体实施方式

[0054] 现在将参考附图来描述本发明的优选实施方式。贯穿附图，除非特别指出，相同的参考数字表示相同的目标。应当注意，下文说明书是本发明的优选实施方式，并且本发明并不限于在此实施方式中描述的内容。

[0055] 图1是示出用于执行本发明的整体系统的示意图。在图1中，多个客户端计算机102a、102b、…、102z通过互联网104来根据诸如HTTP协议访问分布式处理系统106。

[0056] 分布式处理系统106具有经由LAN、WAN等互联的多个服务器106a、106b、…、106z。分布式处理系统106是用于通过使用键值存储(KVS)来创建分布式数据库的系统。换言之，ID被赋予每个服务器106a、106b、…、106z，并且用于键的哈希值mod被优选地计算(尽管不限于此方法)以便唯一地确定保持键的服务器。

[0057] 由此，将由客户端计算机102a、102b、…、102z中的每一个访问的服务器106a、106b、…、106z中的任意一个是通过将被引用的键来确定的。优选地，服务器106a、106b、…、106z中的一个被称为目录服务器的服务器，其中键存储在其他服务器上并且其他信息被如此存储，以便客户端计算机102a、102b、…、102z中的每一个将一次访问目录服务器，以获取用于确定客户端计算机将访问服务器106a、106b、…、106z中的哪一个的信息，以便与所指令的服务器建立连接。备选地，还可以使用从由客户端计算机访问的任何服务器向其他多个服务器广播、用以获取信息的方法。为方便起见，下文将描述在找到目标服务器并且建立连接后的情况。

[0058] 客户端计算机102a、102b、…、102z中的每一个生成用于访问分布式处理系统106的唯一全局事务ID，并且使用全局事务ID来用于与分布式处理系统106的后续事务。

[0059] 接着参见图2，将描述图1中被示出为参考数字102a、102b、…、102z所指示的客户端计算机的硬件配置。在图2中，客户端计算机具有主存储器206、CPU 204以及IDE控制器208，并且这些组件被连接至总线202。显示器控制器214、通信接口218、USB接口220、音频接口222、以及键盘/鼠标控制器228也被连接至总线202。硬盘驱动(HDD)210和DVD驱动212被连接至IDE控制器208。DVD驱动212被用于在必要时从CD-ROM或者DVD引入程序。显示器设备216具有LCD屏，其优选地连接至显示器控制器214。应用屏通过web浏览器被显示在显示设备216上。

[0060] 在必要时诸如扩展硬盘驱动的设备可以被连接至USB接口220。

[0061] 键盘230和鼠标232被连接至键盘/鼠标控制器228。键盘230被用于键入键数据或者密码来执行搜索。

[0062] CPU 204可以基于任何类型，例如，可以使用基于32位架构或者64位架构，以及Intel Pentium(Intel公司的注册商标)4或者Core(注册商标)2Duo、或者AMD Athlon(商标)等。

[0063] 至少用于访问分布式处理系统106的操作系统和客户端侧的程序402a(图4)被存储在硬盘驱动210中。操作系统在系统启动时被加载至主存储器206。Window XP(微软公司的注册商标)、Windows Vista(微软公司的注册商标)、Windows(微软公司的注册商标)7、Linux(Linus Torvalds的注册商标)等可以作为操作系统来使用。将在下文中参考图4的框图和图9至图14的流程图来更详细地描述客户端侧的应用程序402a。

[0064] 通信接口218使用由操作系统提供的TCP/IP通信功能,以便在以太网协议(注册商标)等下通过互联网104来与分布式处理系统106进行通信。

[0065] 图3是分布式处理系统106中的服务器106a等的硬件配置的示意框图。如所示出,服务器106a、106b、…、106z通过互联网104连接。由于服务器106a、106b、…、106z基本具有相同的配置,服务器106a在此作为服务器的代表而示出。如图3所示,客户端计算机102a、102b、…、102z经由互联网104被连接至服务器106a的通信接口302。通信接口302进一步连接至总线304、CPU 306、主存储器(RAM)308,而硬盘驱动(HDD)310被连接至总线。

[0066] 尽管未示出,键盘、鼠标、和显示器还被连接至服务器106a,以便维护人员将使用这些组件来进行服务器106的常规管理和维护工作。

[0067] 操作系统被存储在服务器106a的硬盘驱动310中。

[0068] 在硬盘驱动310中还存储了软件,诸如用于使得服务器106a充当web服务器的Apache、用于实现Java虚拟环境的Java EE、以及根据本发明的应用程序402a(将在下文描述),其运行在Java虚拟环境中。这些软件程序在服务器106a启动时被加载到主存储器308中并且被执行。这使得客户端计算机102a、102b、…、102z通过TCP/IP协议来访问服务器106。

[0069] 此外,在服务器106a的硬盘驱动310中存储有诸如IBM(R)Web Sphere eXtreme Scale的用于实现KVS的软件。另外,在硬盘驱动310中,存储根据本发明的用于KVS的事务处理程序406a(图4)。在下文中将参考附图4的框图以及附图9-图14的流程图,来更详细地描述此事务处理程序406a的功能。

[0070] 可以使用如上的服务器106a,服务器型号诸如IBM(国际商业机器公司的注册商标)系统X、系统i或者系统p,其从国际商业机器公司可获得。在此情况下的可用操作系统的示例包括AIX(国际商业机器公司的注册商标)、UNIX(Open Group的注册商标)、Linux(注册商标)、Windows(注册商标)2003服务器等。

[0071] 图4是示出分别在客户端计算机102a、102b、…、102z中的每一个和服务器106a、106b、…、106z中的每一个中的处理程序的示意框图。在此,客户端计算机102a和服务器106a被示出为客户端计算机和服务器的代表。

[0072] 客户端计算机侧的应用程序402a被存储在硬盘驱动210中,被加载到主存储器202中并且由客户端计算机上的用户的预定的操作执行,其具有从客户端计算机向服务器上的KVS系统给出指令的功能,诸如事务启动、数据查询、数据更新、提交和事务终止。

[0073] 应用程序402a具有功能404a,用以在整个系统内生成唯一的全局事务ID(Tx ID)。作为生成全局事务ID的方法的一个示例,给出对于客户端计算机102a、102b、…、102z和服务器106a、106b、…、106z中的每一个唯一的ID,使得每次每个客户端计算机开始一个事务时,客户端计算机的ID加上在客户端计算机上增量的序列号将被设置为全局事务ID。然而,还可以使用在整个系统内设置唯一全局事务ID的任何其他方法。

[0074] 尽管应用程序402a可以生成用以访问服务器106a的全局事务ID,还可以生成用以同时访问多个服务器的其他全局事务ID,以便处理多个全局事务。

[0075] 在服务器106a的硬盘驱动310中,事务处理程序406a、KVS程序408a(诸如IBM(R) WebSphere eXtreme Scale)和将由KVS程序408a引用的键值对被存储。事务处理程序406a和KVS程序408a被加载到主存储器308中以在服务器106a启动时运行。

[0076] 响应于请求伴随有来自客户端计算机102a的全局事务的事务ID,事务处理程序406a控制KVS程序408a以执行处理,该处理涉及获取用于映像条目的锁定、用于提交或者回滚的处理以及优选地在主存储器308中创建具有条目的管理映像412a(该条目包括全局事务ID、状态、排队的全局事务ID),维护用于每个服务器的管理映像412a。

[0077] 在描述根据本发明的KVS系统的配置和操作之前,将描述某些典型传统KVS系统的配置和操作。应当注意,通过参考这些传统系统而使得根据本发明的系统的特征变得更加清晰。

[0078] 图5是示出典型传统KVS的配置的图示。同样在此,KVS配置使得数据被划分为如所示出的数据502a、502b、502c和502d,并且被分布至多个服务器106a、106b、…、106z。客户端计算机102b向一个服务器做出针对事务处理的请求,但是类似于客户端计算机102a,客户端计算机102b还可以向两个服务器做出针对事务处理的请求。在此情况下,数据如此分布以使得任何两个数据集将是互斥的(disjoint)。优选地,数据被放置在其上的每个服务器是通过计算针对键的哈希值mod基于数据来确定的。

[0079] 客户端计算机102a和102b向由键唯一确定的服务器发送命令以做出用于处理的请求,该命令诸如begin(开始事务)、put(将键与值相关联)、get(获取与键相关联的值)以及commit(使得更新持久)。

[0080] 图6是示出在典型传统KVS系统中的客户端计算机102a和客户端计算机102b以及服务器106a和服务器106b之间的事务处理的示例的图示。Tx1、Tx2、和Tx3分别是本地事务的事务ID。在此示例中,客户端1即客户端计算机102a指令服务器1即服务器106a来在数据映像上执行put(K1,U1),指令服务器2即服务器106b来在数据映像上执行get(K4)之后在数据映像上执行put(K3,U3),指令服务器1来执行提交,继而指令服务器2来执行提交。

[0081] 另一方面,客户端2即客户端计算机102b指令服务器1来在数据映像上执行处理,以按照如下顺序来连续地执行put(K2,U2)、get(K5)、put(K1,U1')、以及提交。

[0082] 在此情况下,如果在服务器1完成提交处理中的提交之后、并且在服务器2上的提交处理之前、在客户端1中出现故障,则由于服务器2不能确定是否提交事务,客户端1不能自动地更新K1和K3,并且由此不能实现全局事务。

[0083] 为了解决这一问题,已经开发出基于两个阶段的提交的KVS系统来支持如图7中所示的全局事务。在此系统中,在本地事务中保持读取锁定,并且写入锁定被保持为映像条目值[CURRENT->DIRTY,LOCK],其包括获取写入锁定的全局事务的事务ID(LOCK)、连同在被更新之前被提交的值(CURRENT)以及正在被更新的值(DIRTY)。出于方便起见,下文假定每个映像条目包括KEY列、VALUE列、LOCK列,以及(CURRENT->DIRTY)作为CURRENT和DIRTY值被存储在VALUE列中,而LOCK值被存储在LOCK列中。当没有DIRTY值时,仅有CURRENT被存储在VALUE列中。此外,在图7中,Tx1-1、Tx1-2、Tx2-1、Tx2-2、Tx2-3、GTx1-1和GTx1-2是本地事务的事务ID,并且GTx1和GTx2是全局事务的事务ID。由Tx1-1、Tx1-2、GTx1-1、和GTx1-2指示的本地事务是用于处理由GTx1指示的全局事务的本地事务过程。

[0084] 在客户端计算机102a作为客户端1时,本地事务GTx1-1首先指令服务器3(服务器106c)来在put(GTxA,working)之后在管理映像上执行提交。

[0085] 继而,客户端1指令服务器1(服务器106a)来在本地事务Tx1-1中在put(K1,V1->U1,GTxA)之后在数据映像上执行提交。

[0086] 接着,客户端1在本地事务Tx2-1中指令服务器2(服务器106b)来在数据映像上执行get(K4)。

[0087] 接着,客户端1在本地事务Tx2-2中指令服务器2来在数据映像上执行put(K3,V3->U3,GTxA)并且提交。

[0088] 接着,客户端1在本地事务GTx1-2中指令服务器3来在管理映像中执行put(GTxA,committed),并且提交。

[0089] 接着,客户端1在本地事务Tx1-2中指令服务器1来在数据映像上执行put(K1,U1,NULL)并且提交。

[0090] 接着,客户端1指令服务器2来在本地事务Tx2-1中提交。

[0091] 接着,客户端1指令服务器2来在本地事务Tx2-3中在数据映像上执行put(K1,U3,NULL)并且提交。

[0092] 另一方面,客户端2(即,客户端计算机102b)指令服务器1来在本地事务Tx3中在数据映像上执行put(K2,U2,NULL)、get(K5)、put(K1,U1',NULL),并且提交。

[0093] 在此配置中,客户端1可以自动地更新K1和K3值,然而客户端2不被允许在本地事务中更新K1值。这是因为客户端1对于正在被更新的K1的映像条目在服务器1上没有保持本地并发性控制机制的锁定,并且由此客户端2可以在K1值上更新数据。然而,在全局事务过程中,由于客户端1使得GTxA条目成为LOCK值,该条目不应当被首先更新。为了防止这样,全部事务必须被升级到全局事务。然而,由于由多个本地事务实现的全局事务比本地事务具有更大的开销,整个系统的性能被降级。

[0094] 图8示出了本发明的配置。图8中的参考数字对应于图4的功能框图中的参考数字。如所示出,管理映像412a、412b、412c和412d中的每一个具有事务ID(TxID)的列,而事务状态分别在服务器106a、106b、…、106z中的每一个上单独地提供,其中事务状态被存储在VALUE列中。

[0095] 此外,用于存储KVS数据的表(数据映像)410a、410b、410c和410d中的每一个在服务器106a、106b、和106d中的每一个上被提供。数据映像410a、410b、410c和410d中的每一个具有:KEY列,作为用于制作键的条目的列;VALUE列,作为用于制作被提交的值或者正被更新的值的条目;以及LOCK列用于存储锁定状态,即正在被更新的事务ID。

[0096] 服务器106a、106b、106c和106d中的每一个使用针对数据映像410a、410b、410c和410d中的每个映像条目的KVS事务机制,以管理四个锁定状态。四个锁定状态是S(正在由全局事务或者本地事务引用)、Init(没有访问)、LX(正在由本地事务更新)以及GX(正在由全局事务更新)。四个锁定状态根据图9的转换图示执行转换。

[0097] 在图8中,请求全局事务的客户端计算机102a做出用于处理多个本地事务的请求,并且请求本地事务的客户端计算机102b做出用于处理单一本地事务的请求。

[0098] 服务器106a、106b、106c和106d中的每一个包括本地并发性控制机制(未示出),并且管理映像412a、412b、412c和412d中的每一个被放置在服务器106a、106b、106c和106d中的每一个上,由此在本地事务被升级到全局事务之后,在本地事务之间、在全局事务之间、在本地事务和全局事务之间的读写冲突通过本地并发性控制机制来解决,在全局事务之间的写写冲突通过使用管理映像来解决,在本地事务和全局事务之间的写写冲突通过本地并发性控制机制来解决,在全局事务和本地事务之间的写读冲突和写写冲突通过使用管理映

像来解决。假定本地并发性控制机制在当Get被请求时向请求本地事务的客户端给出读取锁定,在Put被请求时给出写入锁定,在GetForUpdate被请求时给出写入锁定。

[0099] 接着,将描述四个锁定状态,即,S(正在由全局事务或者本地事务引用)、Init(没有访问)、LX(正在由本地事务更新)以及GX(正在由全局事务更新)。如图9所示,从Init向S、LX和GX中的任一项做出转换。

[0100] 从S向Init、LX和GX中的任一项做出转换。LX和GX仅可以返回Init。

[0101] 当本地事务请求S锁定时,本地事务被开始以执行查询处理(Get)。在确认锁定状态不是GX之后,查询处理被执行。当锁定状态是GX时,本地事务被更新至全局事务以作为全局事务等待,直到做出更新的全局事务被终止,并且其后请求S作为全局事务。

[0102] 当本地事务请求LX锁定时,本地事务被开始以执行查询处理(GetForUpdate),其涉及获取写入锁定。在确认锁定状态不是GX时,更新处理被执行。当锁定状态是GX时,本地事务被更新至全局事务以等待直到做出更新的全局事务被终止,并且其后其请求GX作为全局事务。

[0103] 当本地事务请求提交/回滚时,提交/回滚在本地事务上被执行。

[0104] 当全局事务请求S锁定时,本地事务被开始,并且其后确认锁定状态不是GX,查询处理被执行。当锁定状态是GX时,本地事务被提交,等待直到做出更新的全局事务被终止。

[0105] 当全局事务请求GX锁定时,本地事务被开始,并且在通过在数据映像上使用GetForUpdate确认锁定状态为Init时,针对映像条目中的VALUE列的DIRTY以及LOCK列的更新处理被执行,并且本地事务被提交。当锁定状态为GX时,本地事务被提交,等待直到做出更新的全局事务被终止。

[0106] 当全局事务请求从GX来提交时,本地事务被开始以更新VALUE列中的CURRENT->DIRTY到DIRTY,删除LOCK列并且提交本地事务。

[0107] 当全局事务请求从GX回滚时,本地事务被开始以删除VALUE列中的DIRTY,删除LOCK列并且提交本地事务。

[0108] 接着,将参见流程图来描述根据来自客户端计算机的指令在服务器上执行的处理。在下文中,为方便描述流程图起见,术语可以缩写如下:将事务ID缩写为TxID、将本地事务缩写为LocalTx、以及将全局事务缩写为GlobalTx。

[0109] 图10是示出当事务第一次执行查询/更新处理时的处理的流程图。

[0110] 在步骤1002中,响应于来自事务的请求,服务器基于将被引用和更新的键而被指定,例如一旦访问目录服务器。就此例如,服务器ID是根据作为通过将用于键的哈希值除以服务器的数量而获得的余数的值而指定的。

[0111] 在步骤1004中,指定的服务器开始本地事务。在步骤1006中,指定的服务器引用对应于键的值,并且在步骤1008中,确定在被引用的映像条目中的LOCK列中是否存在另一事务ID。如果存在,则查询处理在步骤1010中被升级到全局事务之后作为全局事务被执行。用于将事务升级到全局事务的具体处理将在下文中参考图13来描述。

[0112] 在步骤1008中确定在被引用的值的LOCK列中不存在其他事务ID时,继而在步骤1012中确定处理是否是查询处理,或者如果不是,则更新处理在步骤1014中作为本地事务被执行。在此,LOCK意味着在数据表410a、…等中的每一个数据表的LOCK列中的值。

[0113] 图11是示出当事务执行用于第二次和后续次数的查询/更新处理时的处理。

[0114] 在步骤1102中,确定事务是否正作为全局事务而工作。如果是,则查询/更新处理在步骤1104中被执行为全局事务。

[0115] 如果事务不是作为全局事务工作,则服务器在步骤1106中基于将被引用和升级的键而被指定,例如,通过一次访问目录服务器而进行。就此,例如,服务器ID是根据作为通过将用于键的哈希值除以服务器的数量而获得的余数的值而指定的。

[0116] 在步骤1108中,确定服务器与先前处理中的服务器是否相同,以及如果相同,则指定的分区引用对应于步骤1110中的键的值。在此情况下,当处理是更新处理时,查询处理(GetForUpdate)涉及写入锁定。

[0117] 在步骤1112中,指定的服务器确定在被引用的值的锁定是否存在另一事务ID(TxID)。如果是,则查询处理在被升级到全局事务之后,在步骤1114中作为全局事务而被执行。

[0118] 当在步骤1112中确定在被引用的值的锁定中不存在其他事务ID,则在步骤1116中确定处理是否是查询处理,以及如果是,则更新处理在步骤1118中作为本地事务而被执行。

[0119] 返回步骤1108,当确定服务器与先前处理中的不相同时,查询处理在被升级到全局事务之后在步骤1114中作为全局事务而被执行。

[0120] 图12是示出用于全局事务的开始处理。

[0121] 在步骤1202中,作为全局事务的标识符的事务ID在客户端中被生成。

[0122] 在步骤1204中,服务器基于事务ID而被指定。在此情况下,例如,服务器ID根据作为将用于事务ID的哈希值除以服务器的数目而获得的余数的值而被指定。

[0123] 在步骤1206中,管理本地事务在指定服务器上开始。继而,在步骤1208中,由事务ID作为键的映像条目的写入锁定在管理本地事务中被获取。

[0124] 图13是示出用于从本地事务向全局事务升级的处理的流程图。

[0125] 在步骤1302中,具有事务ID作为其标识符的全局事务被开始。

[0126] 在步骤1304中,查询本地事务在处理本地事务的服务器上被开始。

[0127] 在步骤1306中,在查询本地事务中的查询关于在本地事务(即,针对其而正在获取S锁定)中正在被引用的全部值而运行。

[0128] 在步骤1308中,在本地事务(即,针对其而正被获取的LX锁定)中正在被更新的全部值被更新至值[CURRENT->DIRTY,LOCK],该值通过组合被更新之前的值(CURRENT)、正在被更新的值(DIRTY)和针对其正在获取写入锁定的事务ID(LOCK)而获得。在此,LOCK值意味着诸如在数据表410a、…等上的LOCK列中的值。

[0129] 在步骤1310中,原始本地事务被提交。由此,所获取的LX锁定被升级至GX锁定。注意,查询本地事务在此时没有被提交。

[0130] 图14是示出在全局事务中的查询处理的流程图。在步骤1402中,服务器基于键被指定。就此,例如,服务器ID时基于作为通过将用于键的哈希值除以服务器的数目而获取的余数的值而被指定的。

[0131] 在步骤1404中,查询本地事务在指定的服务器上开始。

[0132] 在步骤1406中,对应于键的值在查询本地事务中被引用。

[0133] 在步骤1408中,确定在被引用的值的LOCK值中是否存在另一事务ID,以及如果存在,则用于另一事务ID的等待处理在步骤1410中被指定,以及过程返回步骤1402。

[0134] 当在被引用的值的LOCK值中不存在其他事务ID时,处理在此时不提交查询本地事务的情况下结束。

[0135] 图15是示出在全局事务中的更新处理的流程图。在步骤1502中,服务器基于键而被指定。就此,例如,服务器ID是基于作为通过将用于键的哈希值除以服务器的数目而获取的余数的值而被指定的。

[0136] 在步骤1504中,更新本地事务在指定服务器上开始。

[0137] 在步骤1506中,对应于键的值在更新本地事务中被引用。

[0138] 在步骤1508中,确定在被引用的值的LOCK值中是否存在另一事务ID,以及如果存在,则用于另一事务ID的等待处理在步骤1510中被执行,以及过程返回至步骤1502。

[0139] 当在被引用的值的LOCK值中不存在其他事务ID时,在更新本地事务中在步骤1512中执行对值[CURRENT->DIRTY,TxID]的更新,该值通过组合被更新之前的值(CURRENT)、正在被更新的值(DIRTY)以及针对其正在获取写入锁定的事务ID(TxID)而获取。

[0140] 在步骤1514中,更新本地事务被提交。

[0141] 图16是示出在全局事务中的提交处理的流程图。在步骤1602中,相应服务器将由管理映像的事务ID作为键的值更新到管理本地事务中的“被提交”。

[0142] 在步骤1604中,服务器提交管理本地事务。

[0143] 在步骤1606中,确定是否成功执行提交,以及如果没有,则在步骤1608中执行回滚。

[0144] 当在步骤1606中确定提交成功执行,则全部就绪的本地事务在步骤1610中被提交。

[0145] 在步骤1612中,确定正在被更新的全部VALUE是否被选择,以及如果是,则处理结束。

[0146] 当正在被更新的全部VALUE还没有被选择时,正在被更新的KEY及其值[CURRENT->DIRTY,TxID]在步骤1614中被选择。

[0147] 在步骤1616中,服务器基于键而被指定。就此,例如,服务器ID是根据作为通过将用于键的哈希值除以服务器的数量而获得的余数的值而指定的。

[0148] 在步骤1618中,更新本地事务是在指定的服务器上开始的。

[0149] 在步骤1620中,对应于键的值在更新本地事务中被引用。

[0150] 在步骤1622中,确定被引用的值是否是[CURRENT->DIRTY,TxID],并且如果是,则在更新本地事务中在步骤1624中,对应于该键的值被更新至[DIRTY,NULL],更新本地事务在步骤1626中被提交,并且过程返回至步骤1612。

[0151] 在步骤1622中,当被引用的值不是[CURRENT->DIRTY,TxID]时,更新本地事务在步骤1626中立即被执行,并且过程返回至步骤1612。

[0152] 图17是示出用于具有事务ID TxID'作为其标识符的全局事务的等待处理的流程图。

[0153] 在步骤1702中,用于将由事务ID TxID作为键的值更新至[Waiting,TxID']]的处理在管理本地事务中被执行。

[0154] 在步骤1704中,管理本地事务被提交。

[0155] 在步骤1706中,确定是否成功执行了提交,如果没有则在步骤1708中执行回滚。

[0156] 当在步骤1706中确定成功执行了提交时,则服务器在步骤1710中基于TxID'而被指定。例如,这是通过将用于TxID'的哈希值除以服务器数量而获得值来确定的。

[0157] 在步骤1712中,等待本地事务在指定服务器上开始。

[0158] 在步骤1714中,TxID'的值在等待本地事务中的管理映像上被引用。

[0159] 在步骤1716中,确定TxID'的值是被提交还是被回滚。当该值被提交时,服务器在步骤1718中基于TxID被指定,由TxID作为键的值在管理本地事务中在步骤1720中被更新至Working,并且过程结束。

[0160] 另一方面,当TxID'的值被回滚时,在步骤1722中确定TxID是否包括在TxID'的值中,并且如果没有,则过程前进至步骤1724中以回滚处理。

[0161] 当在步骤1722中确定TxID被包括在TxID'的值中时,等待本地事务在步骤1726中被提交。继而在步骤1728中,服务器基于TxID而被指定,以及管理本地事务在步骤1730中在指定的服务器上开始。

[0162] 继而在步骤1732中,针对具有TxID'的全局事务的等待处理被开始。

[0163] 图18是示出了用于全局事务的终止处理的流程图。

[0164] 在步骤1802中,在全局事务启动时开始本地事务,表示全局事务状态的用于映像条目的值被更新至“被提交”或者“被回滚”,并且被提交。

[0165] 在步骤1804中,过程依赖于全局事务的状态是否被提交或者被回滚而产生分支。当被提交时,在步骤1806中,针对全局事务中被更新的全部映像条目,VALUE列中的CURRENT->DIRTY被更新至DIRTY,并且用于删除LOCK列的处理被执行(本地提交)。另一方面,当被回滚时,针对在全局事务中被更新的全部映像条目,用于删除VALUE列中的DIRTY和LOCK列的处理在步骤1808中被执行(本地回滚)。

[0166] 接着,将描述客户端故障期间的处理。

[0167] 首先,当在全局事务被提交之前在客户端中出现故障时,每个映像条目保持GX状态不变。继而,表示全局事务状态的管理映像上的映像条目的本地事务由服务器来回滚,变为其中不再存储状态的状态。下次用于引用和更新映像条目的事务可以检查管理映像,以检查事务是否被回滚。如果被回滚,则本地回滚处理由已经检查到其被回滚的事务来执行。

[0168] 接着,当在本地提交处理之前在客户端中出现故障时,则每个映像条目保持GX状态未改变。下次用于引用和更新映像条目事务可以检查管理映像,以检查事务是否被提交。当被提交时,本地提交处理由已经检查到其被提交的事务来执行。

[0169] 由此,当在全局事务被提交之前在客户端中出现故障、以及当在本地提交处理之前在客户端中出现故障的两种情况下,处理并发性可以根据本发明来得以保持。

[0170] 接着参见图19,将描述本发明的此实施方式的操作的示例。首先,在图19中,客户端1的全局事务GTXA-1(客户端计算机102a)指令服务器3(服务器106c)来执行put(GTxA, working),并继而提交。

[0171] 接着,客户端1的本地事务Tx1-1指令服务器1(服务器106a)来执行put(K1,V1->U1,GTxA),并继而提交。

[0172] 接着,客户端1的本地事务Tx2-1指令服务器2(服务器106b)来执行put(K3,V3->U3,GTxA),并继而提交。

[0173] 接着客户端1的本地事务Tx1-2指令服务器1来执行put(GTx1,committed)并继而

提交。同时,客户端1的全局事务GTXA-1指令服务器3来执行put(GTXA,committed),并继而提交。

[0174] 在此时段,客户端2的本地事务Tx3(客户端计算机102a)尝试在服务器1上执行GetForUpdate(K1)。在此情况下的处理依赖于如下而改变:任何其他全局事务不采取写入锁定,或者任何其他全局事务采取写入锁定。

[0175] 当任何其他全局事务不采取写入锁定时,执行如下处理:

[0176] Tx3-1'.getForUpdate(K2)

[0177] Tx3-2'.put(K2,U2,NULL)

[0178] Tx3-3'.get(K5)

[0179] Tx3-4'.getForUpdate(K1)

[0180] Tx3-5'.put(K1,U1',NULL)

[0181] Tx3-6'.commit

[0182] 当任何其他全局事务采取写入锁定时,执行如下处理:

[0183] Tx3-1'.getForUpdate(K2)

[0184] Tx3-2'.put(K2,U2,NULL)

[0185] Tx3-3'.get(K5)

[0186] Tx3-4'.getForUpdate(K1)

[0187] Tx3-5'.put(K2,V2->U2,GTXB)//从Tx3升级到GTXB

[0188] Tx4-1'.get(K5)//开始Tx4以再次查询正在被查询的映像条目

[0189] Tx3-6'.Commit

[0190] Wait for GTXA commit/rollback//等待GTXA终止

[0191] Tx5-1'.put(K1,U1->U1',GTXB)//在现存事务中,针对全局事务来

[0192] 修改正在被更新的映像条目的写入锁定

[0193] 尽管本发明的实施方式是基于特定的硬件和软件的平台来描述的,本领域技术人员应当理解,本发明可以在任何计算机硬件和软件平台中实现。

[0194] [参考数字的描述]

[0195] 102 客户端计算机

[0196] 106 服务器

[0197] 202 主存储器

[0198] 204 CPU

[0199] 206 主存储器

[0200] 210 硬盘驱动

[0201] 306 CPU

[0202] 308 主存储器

[0203] 310 硬盘驱动

[0204] 404 应用程序

[0205] 406 事务处理程序

[0206] 408 KVS程序

[0207] 410 数据映像

[0208] 412 管理映像

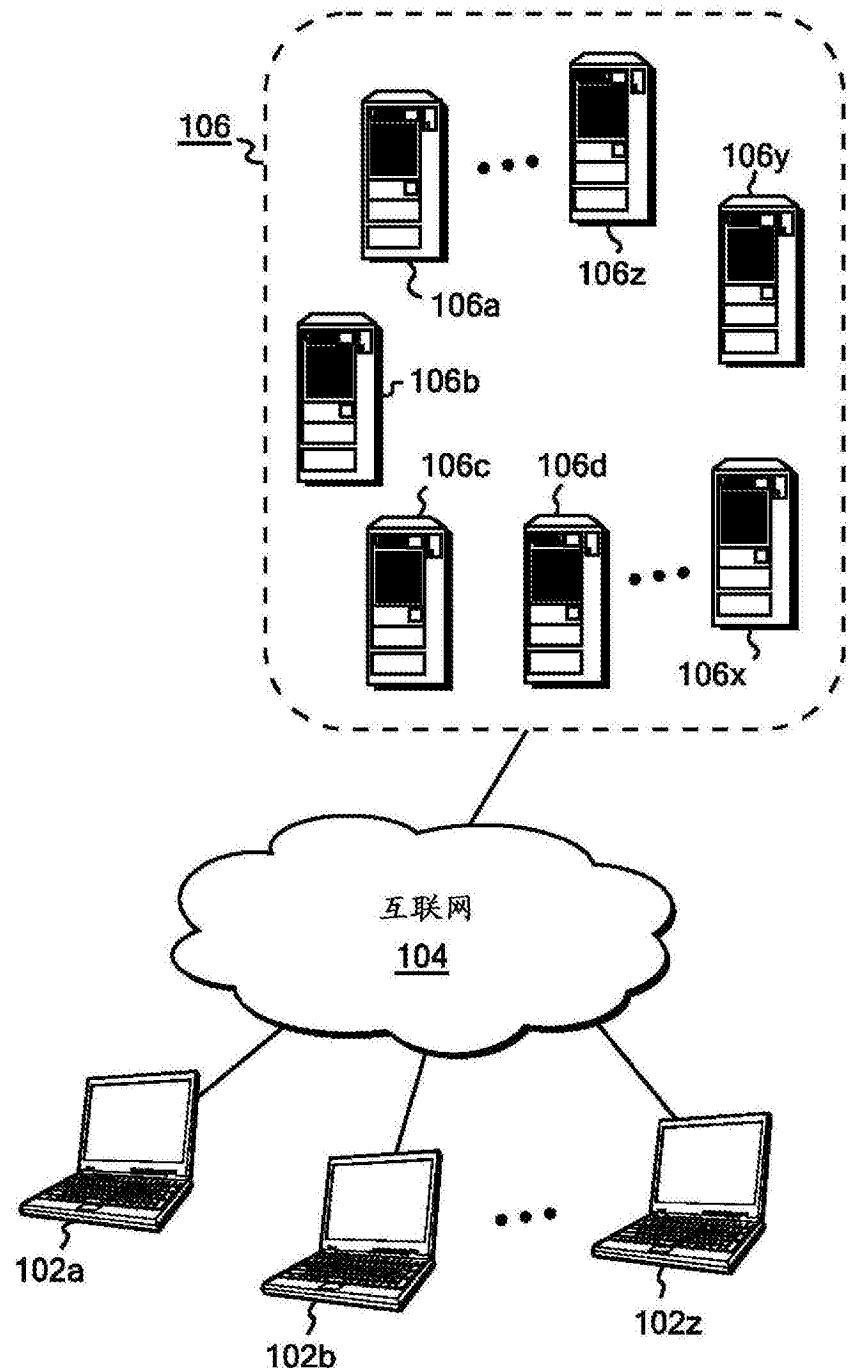


图1

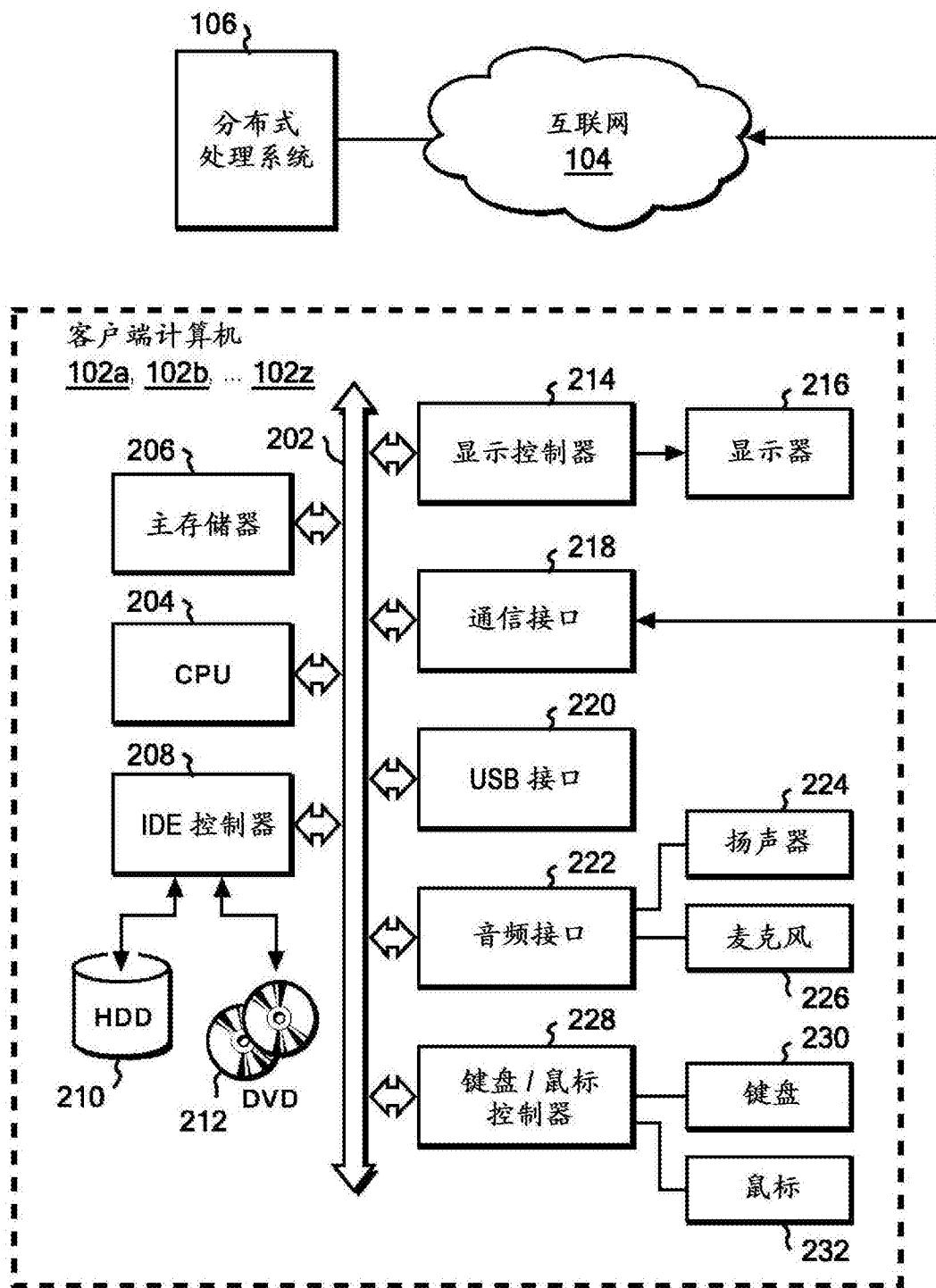


图2

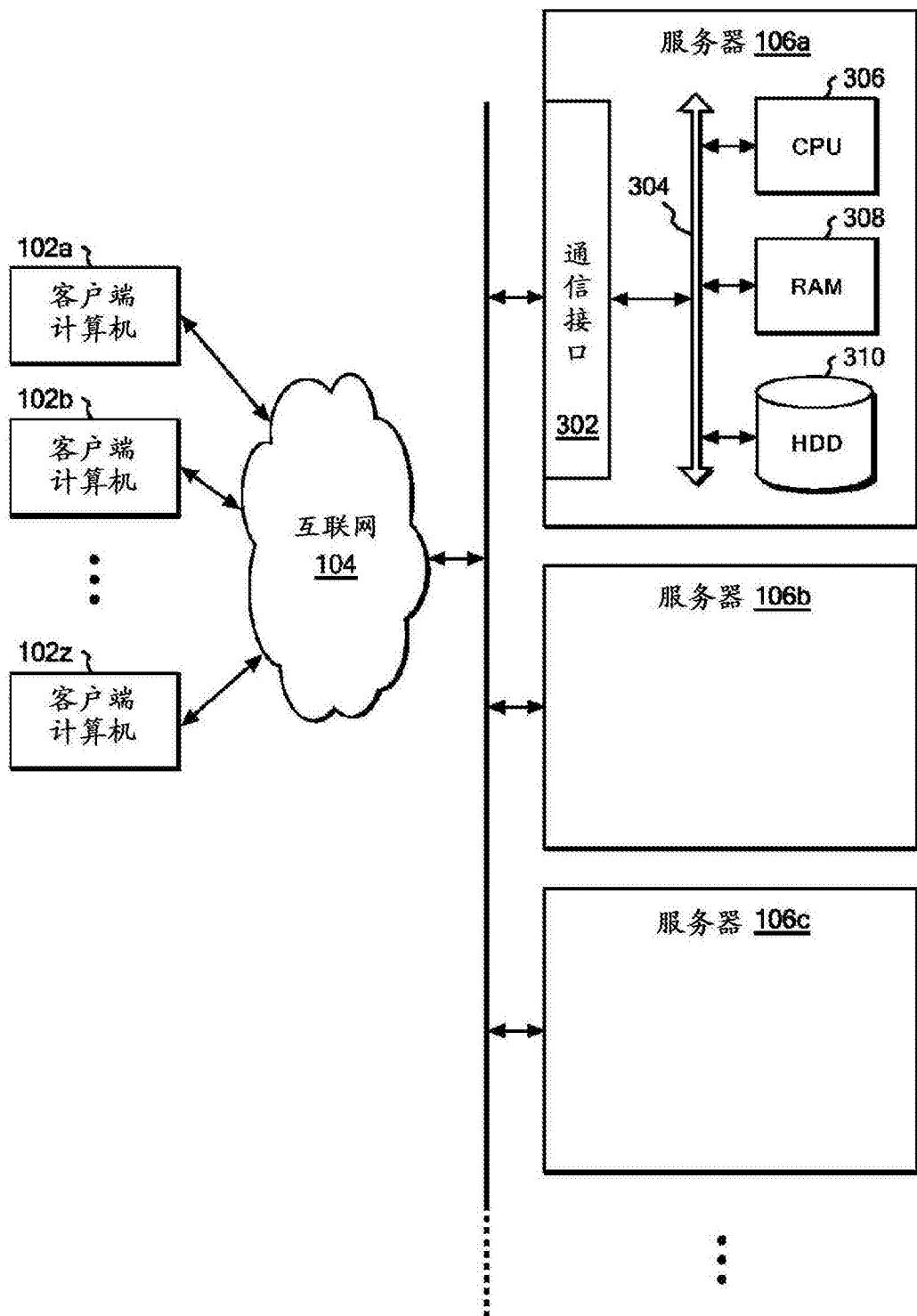


图3

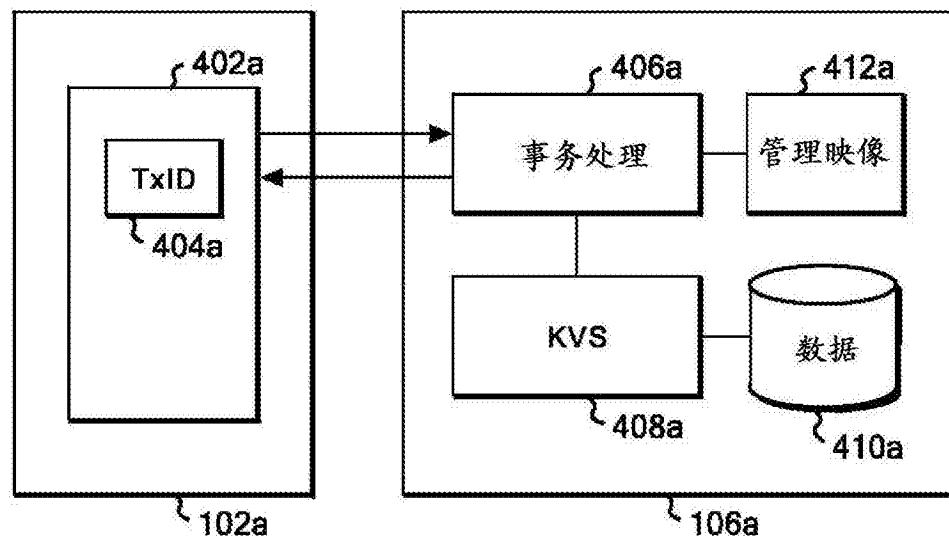


图4

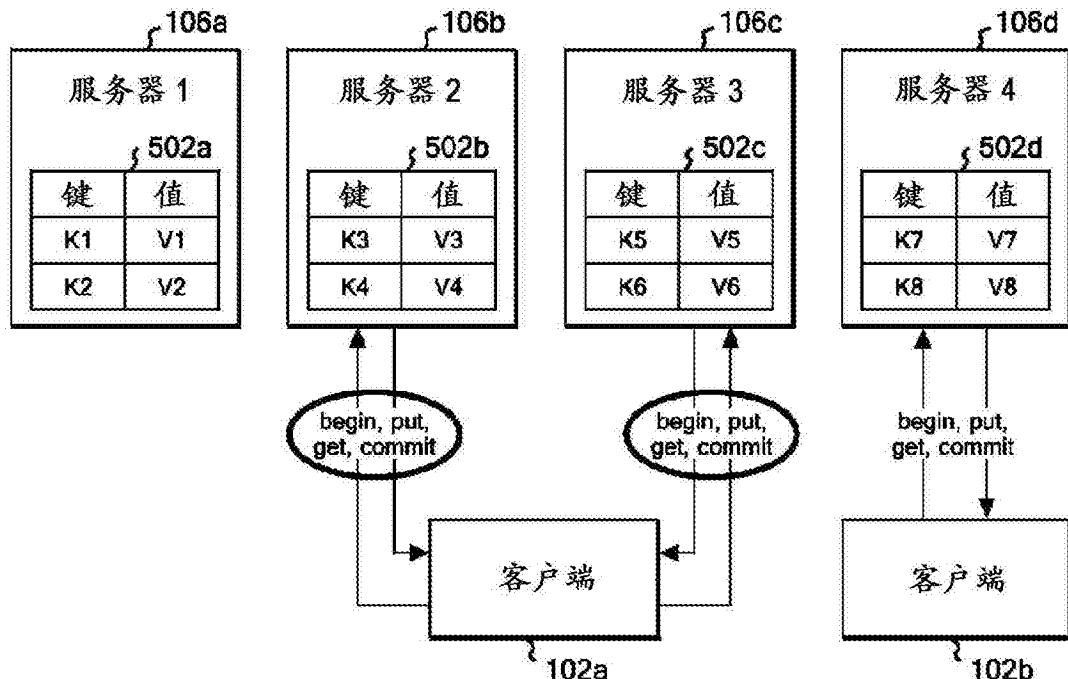


图5

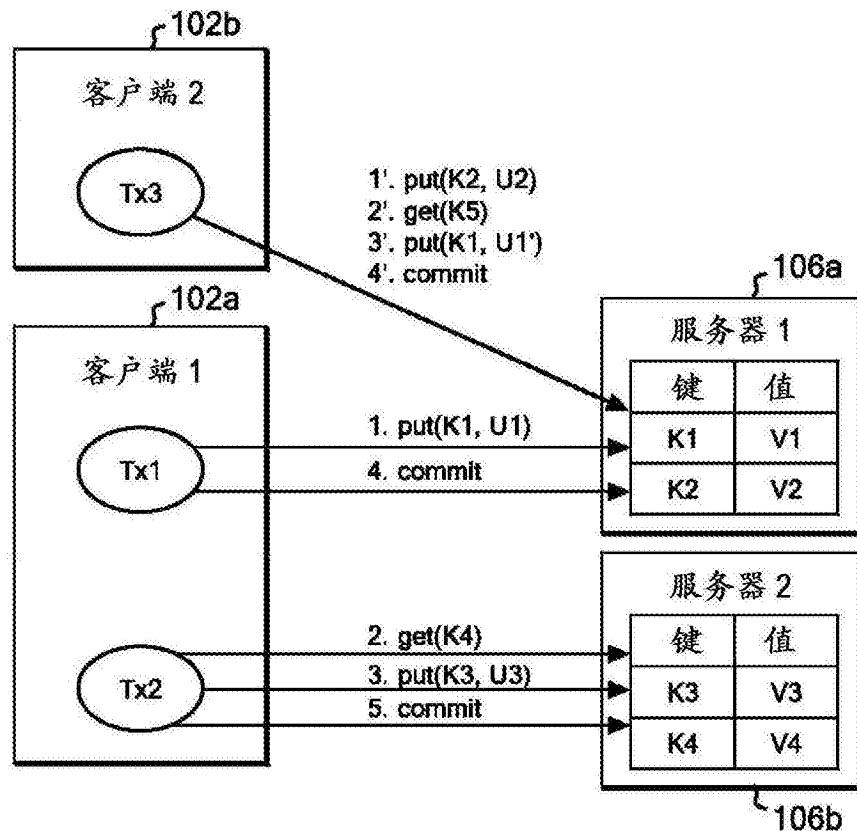


图6

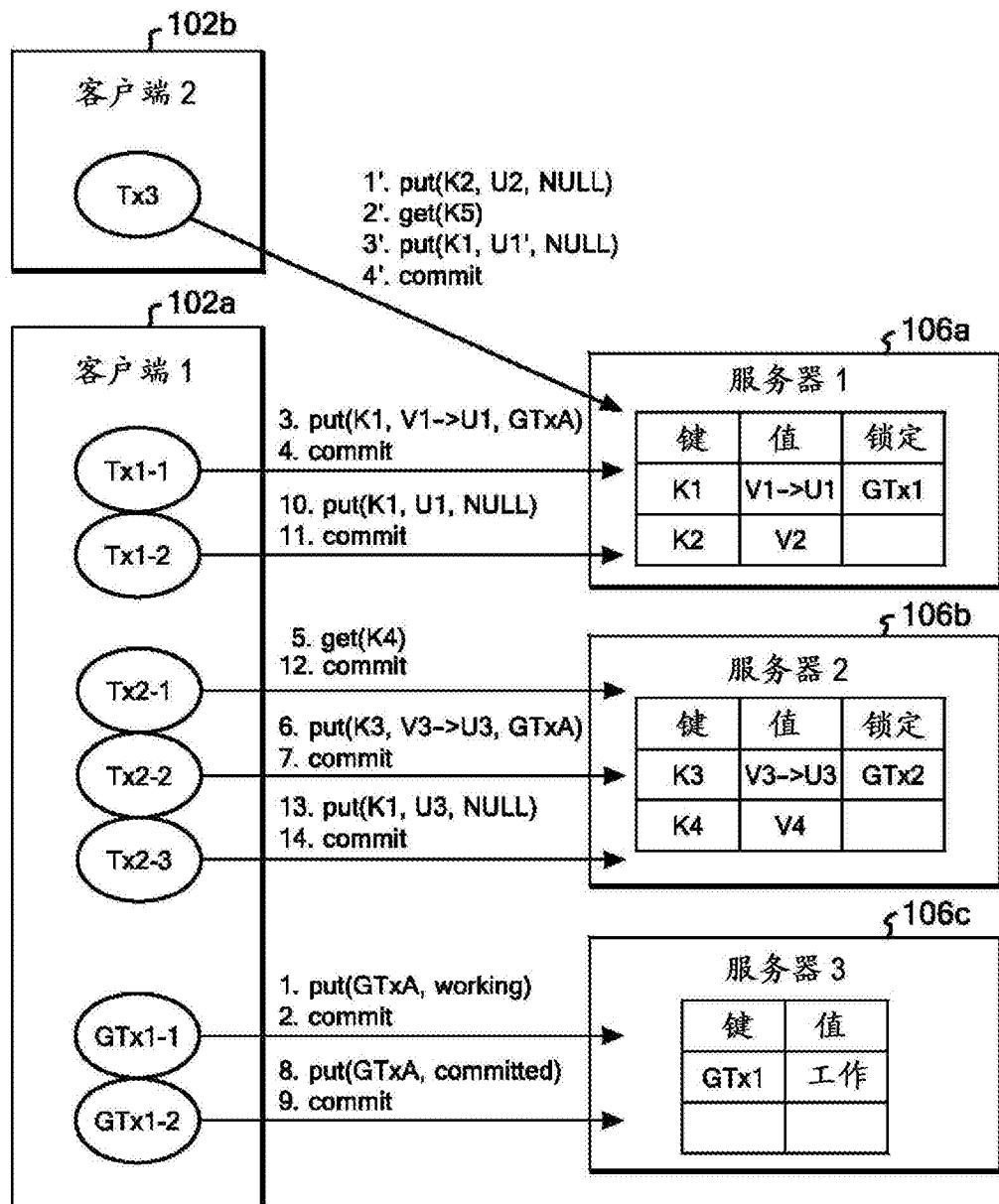


图 7

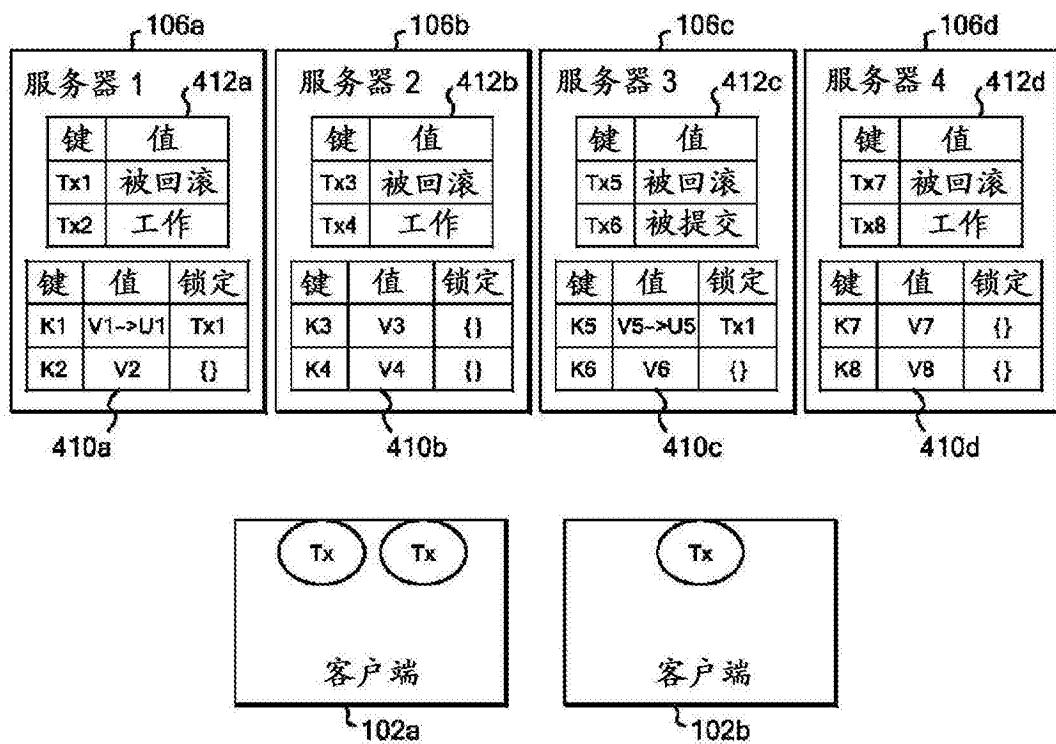


图8

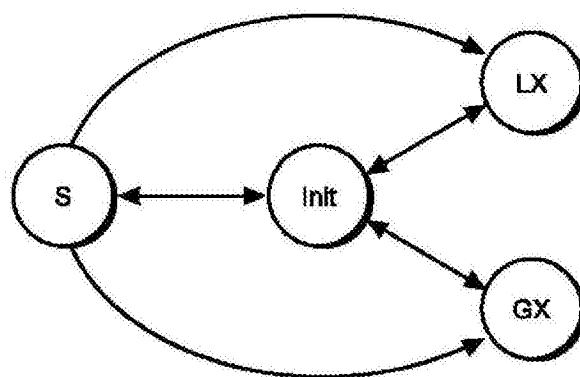


图9

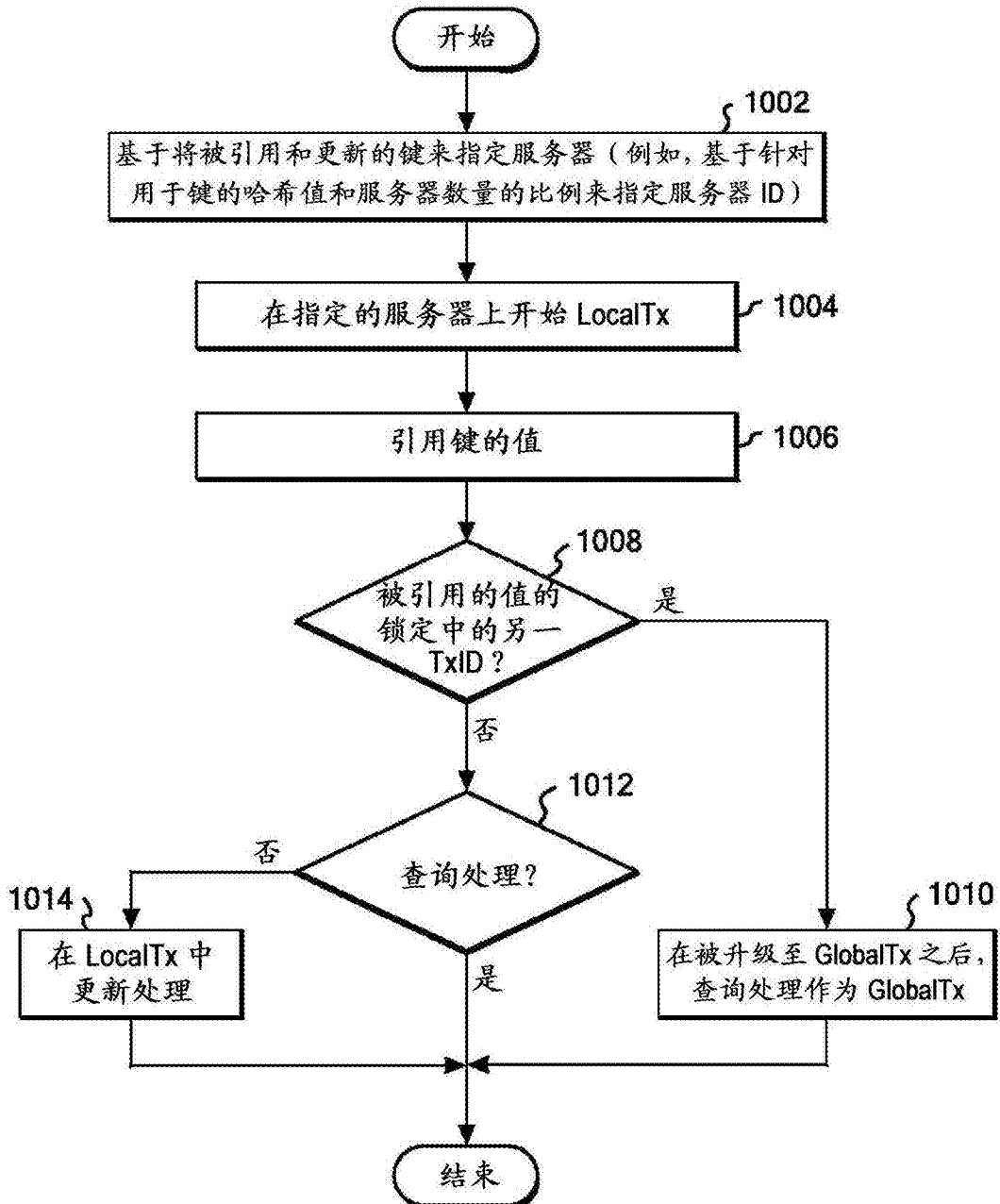


图10

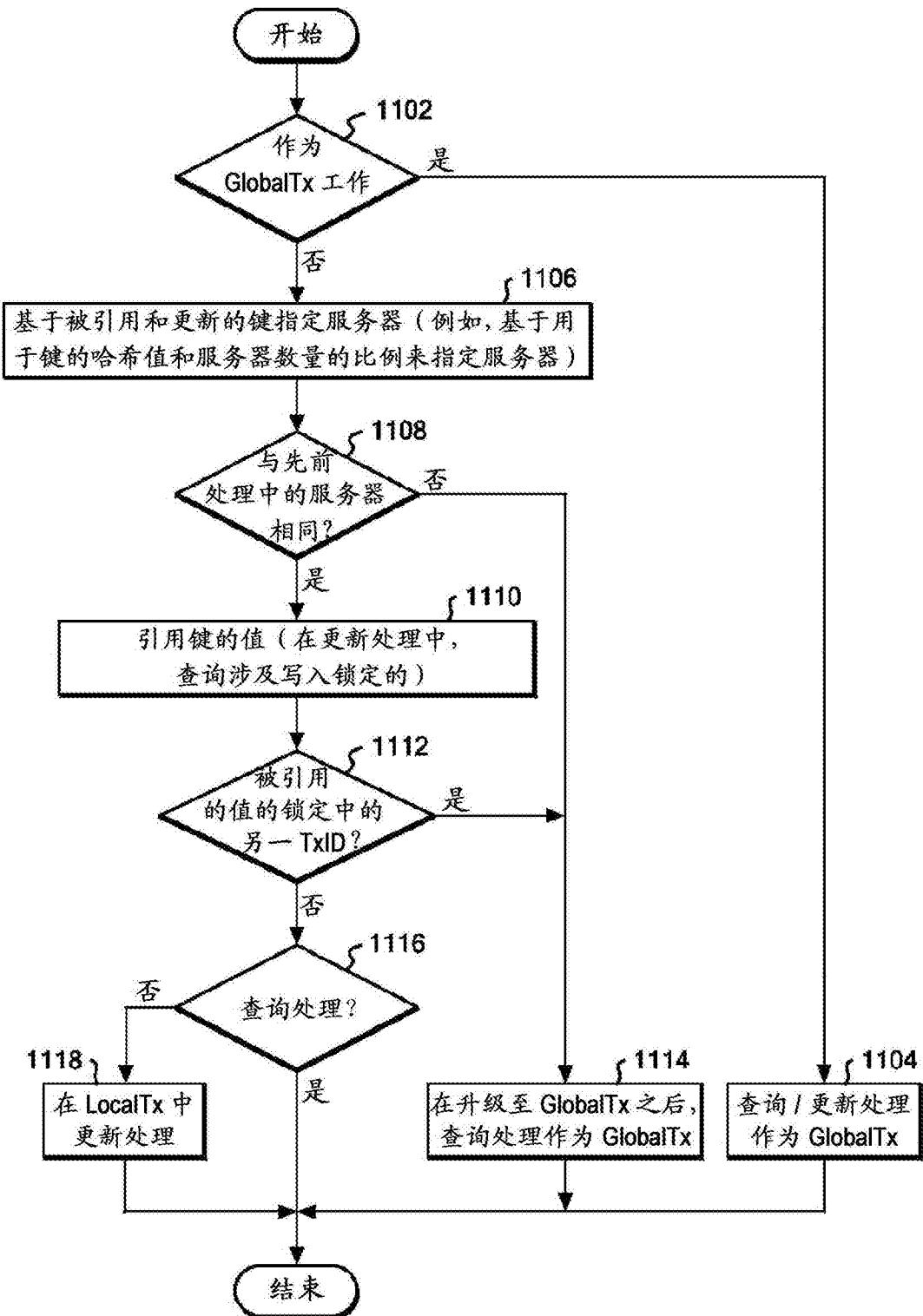


图11

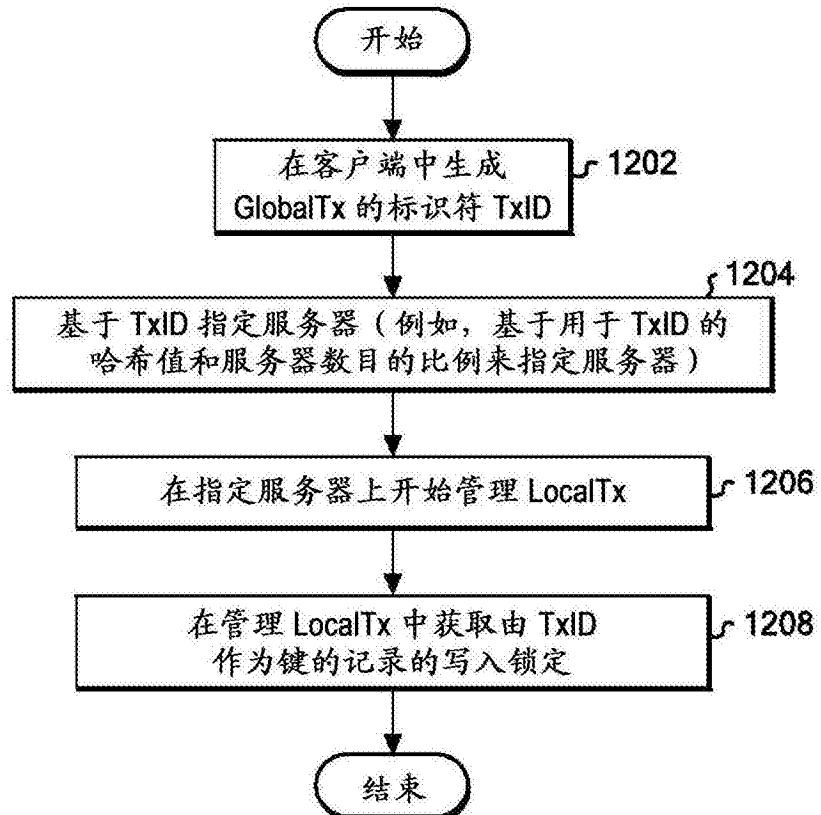


图12

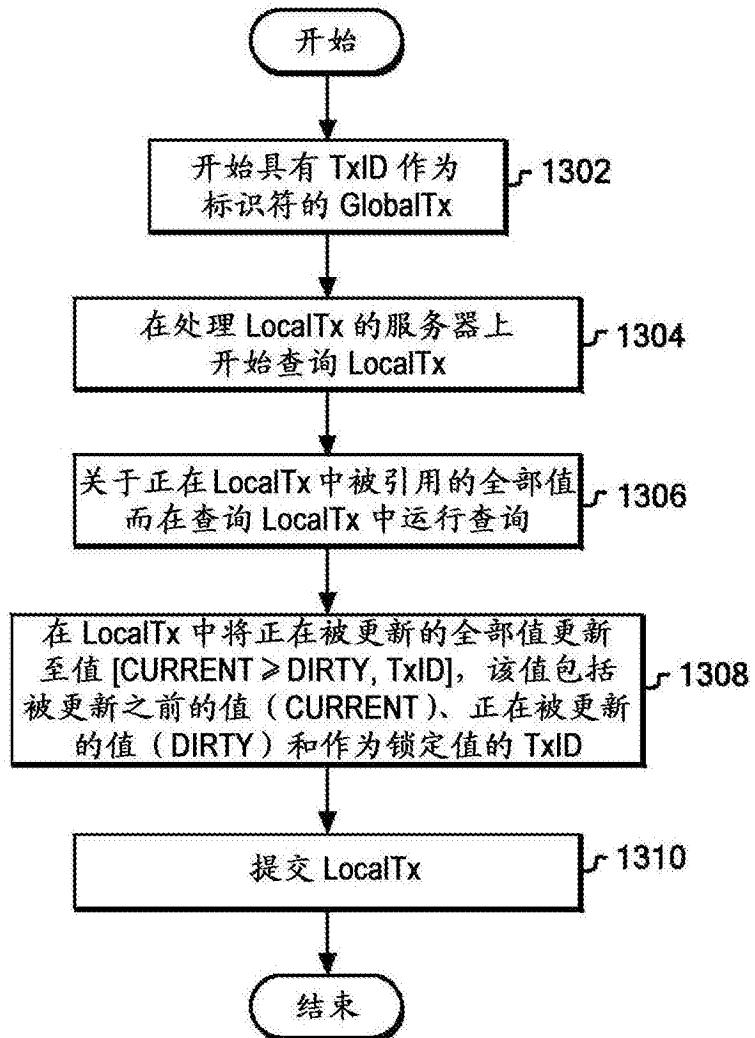


图13

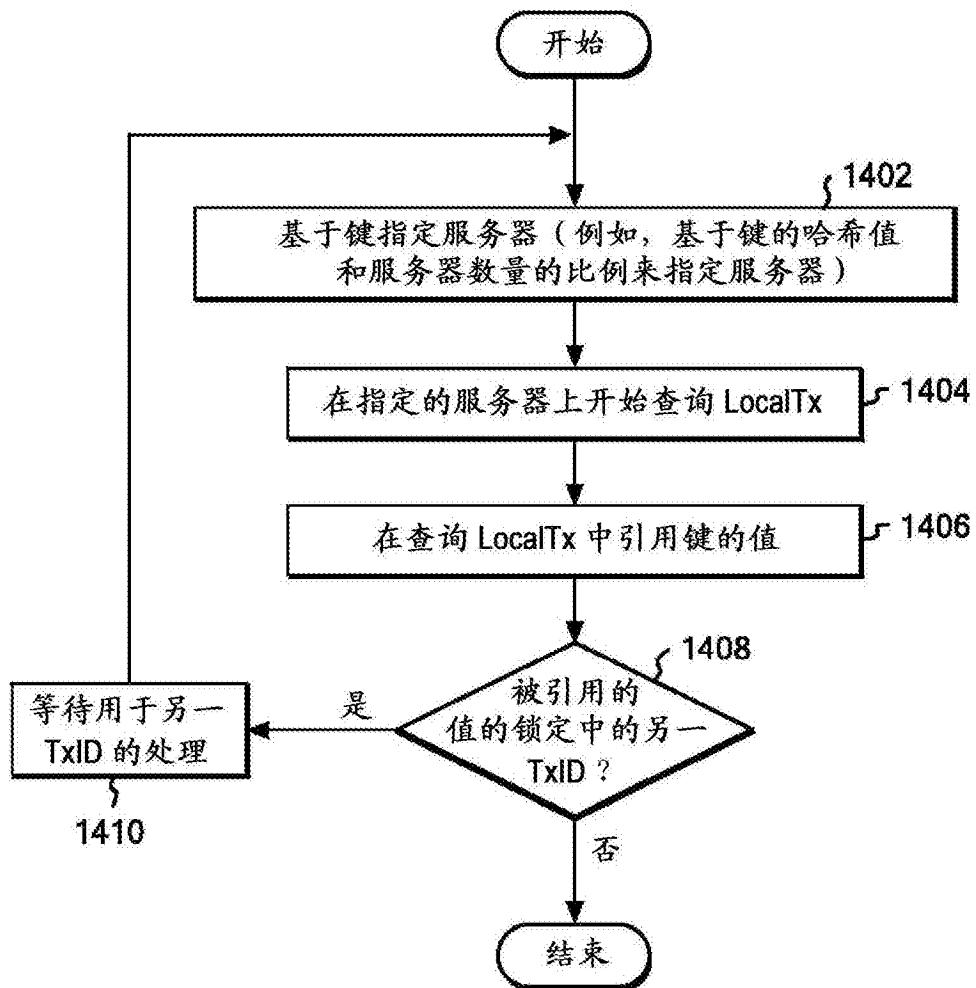


图14

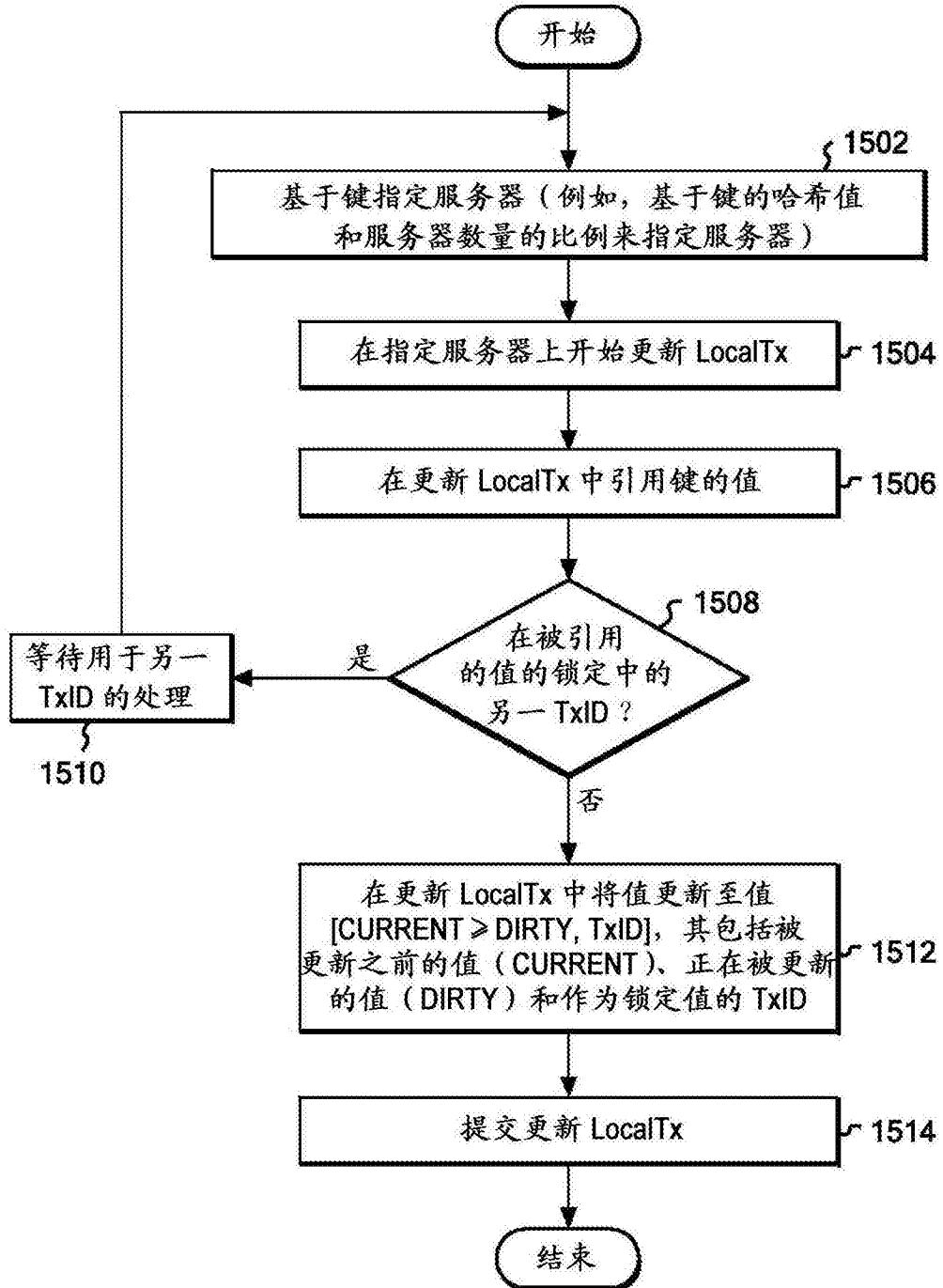


图15

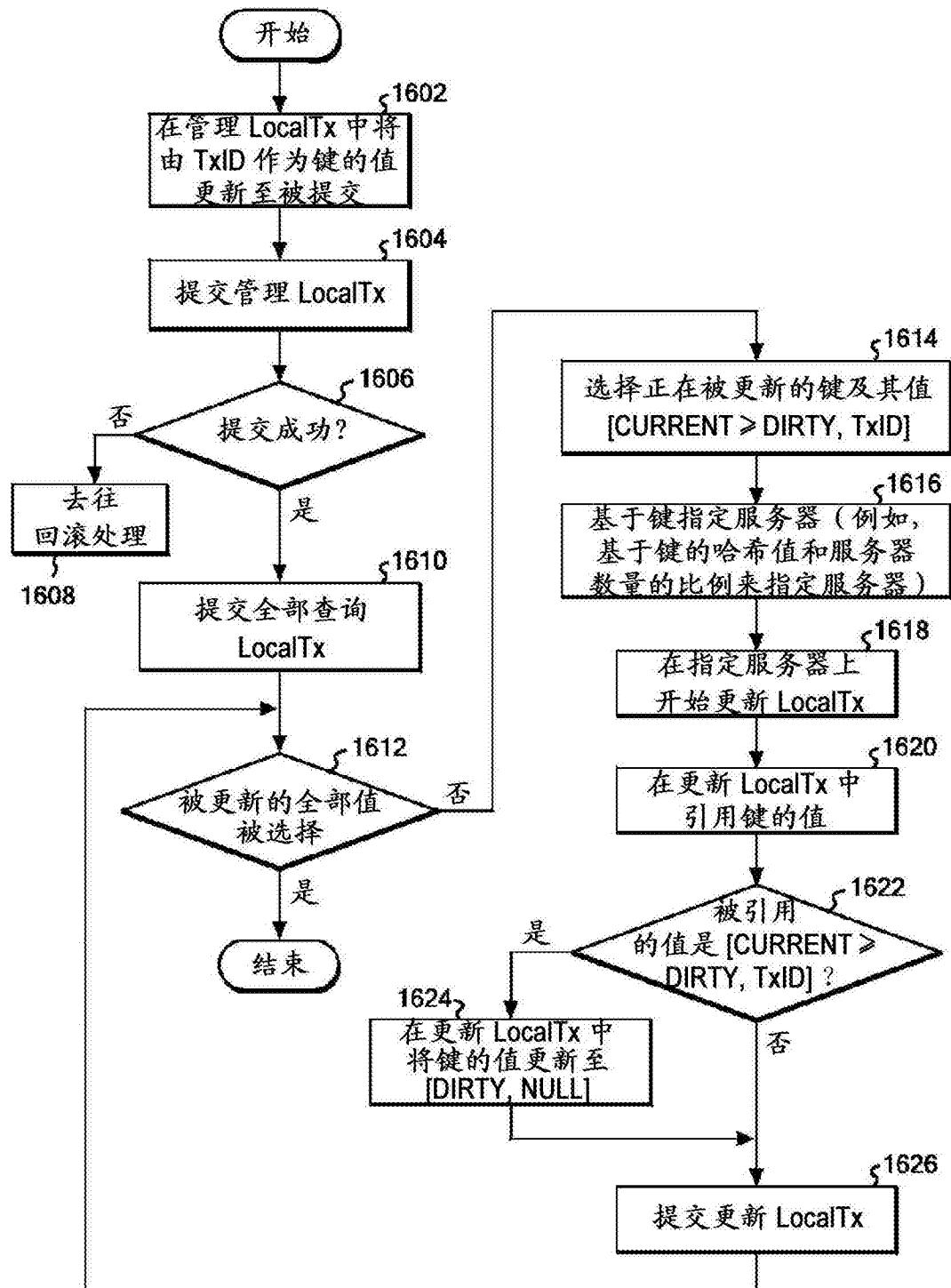


图16

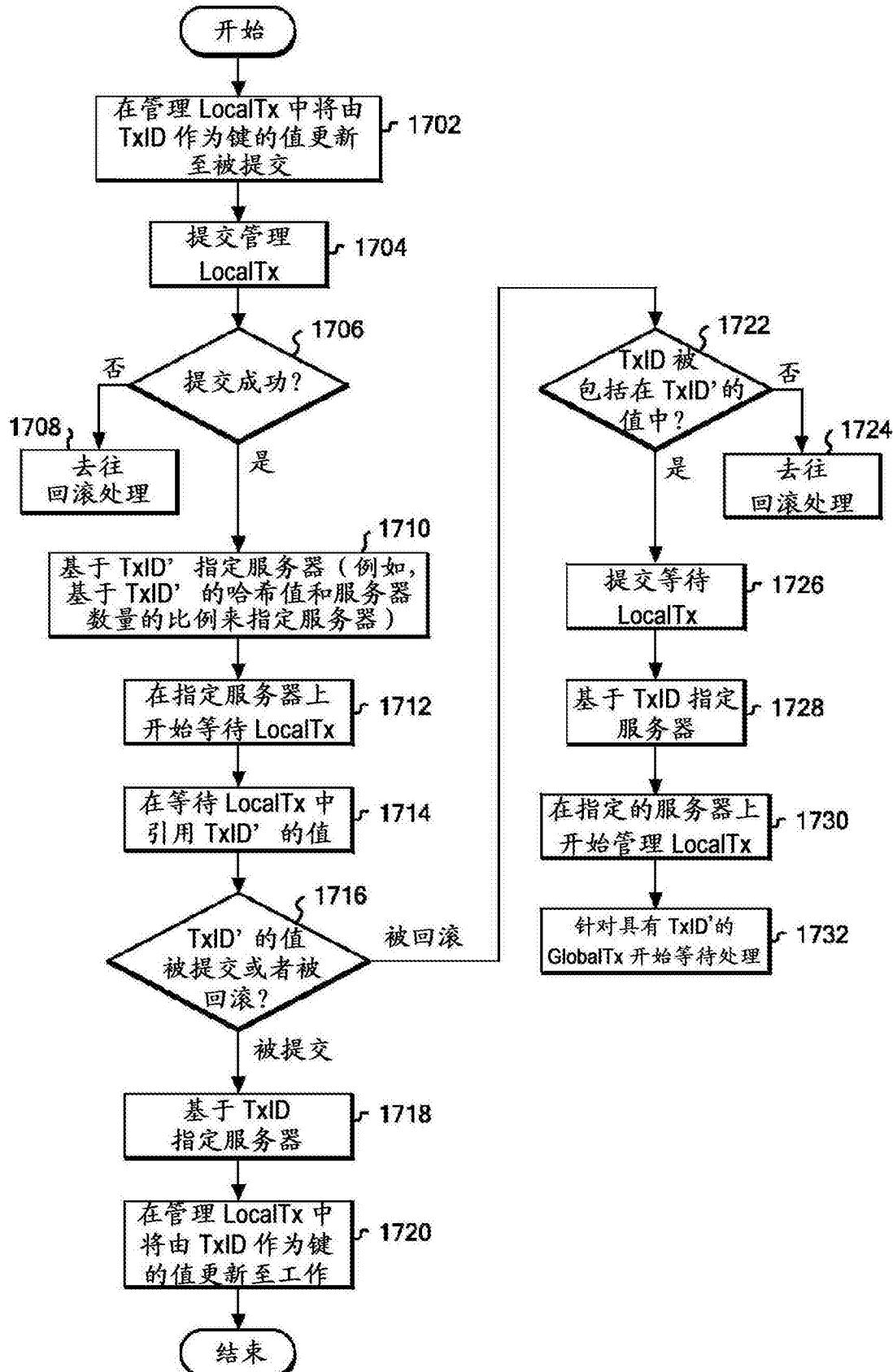


图17

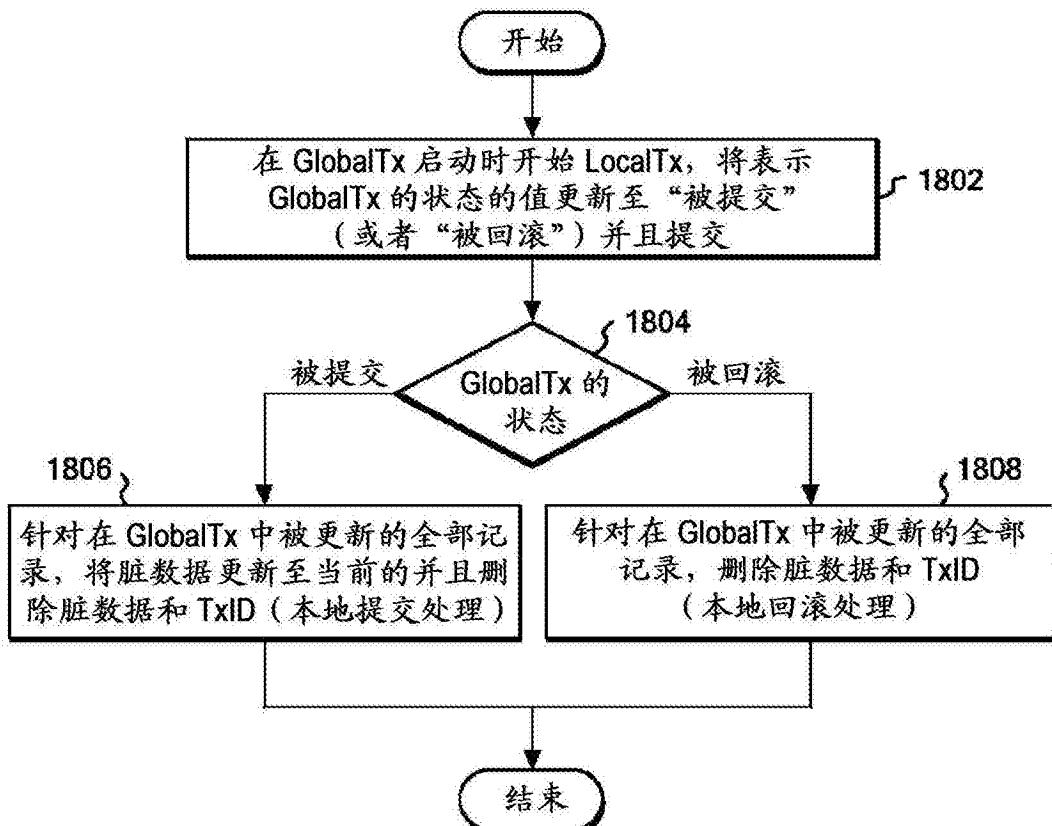


图18

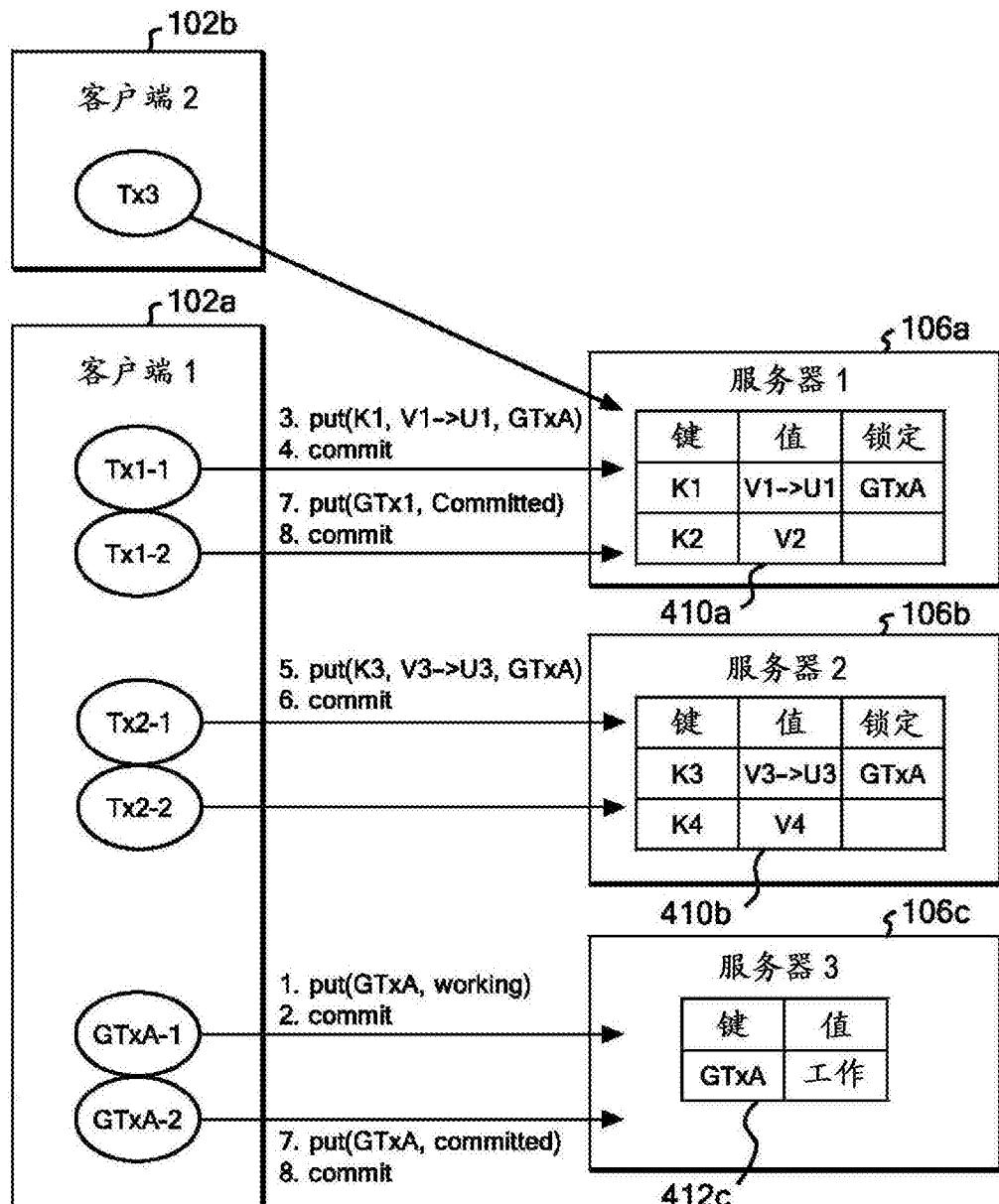


图19