(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101889266 B
(45)授权公告日 2013.06.12

(21)申请号 200880107300.2
(22)申请日 2008.09.16
(30)优先权数据
11/901,647 2007.09.18 US
(85)PCT申请进入国家阶段日
2010.03.12
(86)PCT申请的申请数据
PCT/US2008/076563 2008.09.16
(87)PCT申请的公布数据
WO2009/039118 EN 2009.03.26
(73)专利权人 微软公司
地址 美国华盛顿州
(72)发明人 M·M·马格鲁德 D·德特勒夫
J·J·达菲 G·格雷费
V·K·格罗弗
(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 顾嘉运 钱静芳
(51)Int.Cl.
G06F 9/52(2006.01)
G06F 12/00(2006.01)
(56)对比文件
CN 1989490 A,2007.06.27, 全文.
US 6272515 B1,2001.08.07, 全文.
CN 1801097 A,2006.07.12, 全文.
CN 1959641 A,2007.05.09, 全文.
CN 1539110 A,2004.10.20, 全文.
审查员 解欣

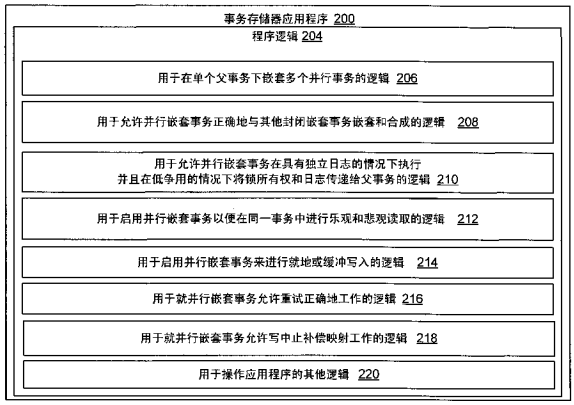
权利要求书1页 说明书12页 附图25页

(54)发明名称

事务存储器中的并行嵌套事务

(57)摘要

公开了用于支持事务存储器系统中的并行嵌套事务的各种技术和方法。为单个父事务创建多个封闭嵌套事务，并且这些封闭嵌套事务作为并行嵌套事务来并发执行。使用各种技术来确保对父事务之外的其他事务隐藏并行嵌套事务的影响直到该父事务提交。例如，就并行嵌套事务使用版本化写锁。当事务存储器字从写锁变为版本化写锁时，在全局版本化写锁映射中形成一条目以存储指向该版本化写锁所替换的写日志条目的指针。当在事务处理期间遇到该版本化写锁时，咨询全局版本化写锁映射以便将该版本化写锁转换成指向写日志条目的指针。



CN 101889266 B

1. 一种用于并行嵌套事务的方法，包括：

为单个父事务创建多个封闭嵌套事务（206）；以及

通过下述步骤将所述封闭嵌套事务作为并行嵌套事务来并发执行（208）：

在所述并行嵌套事务中的每一个的创建时刻，在所述单个父事务的日志中创建一条并行嵌套事务条目（272）；

由所述并行嵌套事务中的相应事务使用对下一个可用并行嵌套事务索引的比较和交换操作来从所述单个父事务的日志中取回相应的并行嵌套事务条目（274）。

2. 如权利要求 1 所述的方法，其特征在于，所述并行嵌套事务是一次性全部创建的（244）。

3. 如权利要求 1 所述的方法，其特征在于，当在提交处理期间在给定并行嵌套事务的日志中遇到锁条目时，将锁的所有权和日志条目传递给所述单个父事务（444）。

4. 如权利要求 1 所述的方法，其特征在于，在创建每一个相应的并行嵌套事务时，在所述单个父事务的日志中为相应的并行嵌套事务条目分配空间（292）。

5. 如权利要求 4 所述的方法，其特征在于，在每一个相应的并行嵌套事务提交时，在所述单个父事务的日志中创建所述相应的并行嵌套事务条目（294）。

6. 如权利要求 4 所述的方法，其特征在于，所述单个父事务的日志在兄弟并行嵌套事务之间同步以便访问（296）。

7. 如权利要求 1 所述的方法，其特征在于，还包括：

对所述父事务之外的其他事务隐藏所述并行嵌套事务的实施直到所述父事务提交（208）。

8. 如权利要求 7 所述的方法，其特征在于，还包括：

对所述父事务的其他并行嵌套事务隐藏给定并行嵌套事务的实施直到所述给定并行嵌套事务被提交至所述父事务中（444）。

9. 一种用于并行嵌套事务的系统，包括：

用于为单个父事务创建多个封闭嵌套事务（206）的装置；以及

用于通过下述装置将所述封闭嵌套事务作为并行嵌套事务来并发执行（208）的装置：

用于在所述并行嵌套事务中的每一个的创建时刻，在所述单个父事务的日志中创建一条并行嵌套事务条目（272）的装置；

用于由所述并行嵌套事务中的相应事务使用对下一个可用并行嵌套事务索引的比较和交换操作来从所述单个父事务的日志中取回相应的并行嵌套事务条目（274）的装置。

事务存储器中的并行嵌套事务

[0001] 背景

[0002] 软件事务存储器(STM)是类似于数据库事务的、用于在并发计算中控制对共享存储器的访问的并发控制机制。事务存储器的上下文中的事务是对共享存储器执行一系列读取和写入的一段代码。换言之,事务访问一个或多个对象中的数据。事务存储器的上下文中的对象是作为一个实体来锁定的一组连接的存储单元。该上下文中的对象也可能是静态变量,或一组这样的变量,或者该对象可能是一组高速缓存线。

[0003] STM 用作传统锁定机制的替换。STM 允许更简单地编写并发程序。事务指定应当如同其隔离执行一样地执行的代码序列。这一隔离错觉可以通过对象的细粒度锁定,以及通过以在发现事务与某一其它事务相冲突的情况下允许回退该事务的副作用的模式执行来实现。如果对于数据访问所生成的代码被修改成包括对这些锁定和回退机制的支持,则可以说该访问被“事务化”。

[0004] 事务可以进行嵌套,并且可被分类为开放或封闭嵌套。如果一线程当前正在执行一事务并且到达一新原子块的开头,则该原子块作为当前执行的父事务的封闭嵌套子事务来执行。该嵌套事务与封闭事务在同一隔离边界内执行,并且就像封闭事务中所访问的其他存储器那样,嵌套事务的实施将只在该封闭事务提交时变得可见。换言之,有效地挂起父事务,并且允许封闭嵌套事务运行至在继续父事务中的处理之前完成。在嵌套事务回退时,其临时实施被撤消并且父事务的状态恢复到该嵌套事务开始的点处。

[0005] 给定线程正在执行的“最外面的”事务是不嵌套的;称之为顶层事务。该顶层事务必须原子地执行,因此嵌套事务变成其一部分。如果某些抽象 A 和 B 各自具有即使在由并发线程使用时它们也想要维护的内部表示不变量,则可能引发嵌套,并且这些抽象由此在实现其方法时使用原子块以确保并发访问不违反这些不变量。现在假设某一更高级的抽象 C 在其实现时使用 A 和 B 的实例,并且具有使这些 A 和 B 实例相关的某些不变量。C 的方法可能使用事务来确保不违反该不变量。如果在 C 的事务内部使用 A 和 B 方法,则 A 和 B 方法中的事务将进行嵌套(在这种情况下)。

[0006] 现有事务存储器系统不允许将在一个事务的隔离边界内执行的工作分布在多个并发执行线程之间。在现有系统中,事务只可具有一个嵌套子事务。这些系统的语义简单地不允许事务内的这样的并行性,并且一次执行不止一个嵌套事务的尝试将导致嵌套事务日志条目在父事务的日志中无序混合和其他错误,以及用于提供隔离错觉的基本的底层细粒度锁定协议的崩溃。

[0007] 概述

[0008] 公开了用于支持事务存储器系统中的并行嵌套事务的各种技术和方法。为单个父事务创建多个封闭嵌套事务,并且将这些封闭嵌套事务作为并行嵌套事务来并发执行。使用各种技术来确保对父事务之外的其他事务隐藏并行嵌套事务的实施直到该父事务提交。

[0009] 在一个实现中,就并行嵌套事务使用版本化写锁。当事务存储器字从写锁变为版本化写锁时,在全局版本化写锁映射中形成一条目以存储指向该版本化写锁所替换的写日志条目的指针。当在事务处理期间遇到该版本化写锁时,咨询全局版本化写锁映射以便将

该版本化写锁转换成指向写日志条目的指针。

[0010] 在另一实现中，对于并行事务支持释放重复写锁以便回退。在并行嵌套事务的回退处理期间，遇到表示写锁的第一写日志条目。如果该写锁被确定为是重复的，则获取全局锁并使用该全局锁来同步对全局版本化写锁映射的访问。

[0011] 在又一实现中，对并行嵌套事务支持乐观读取确认。在乐观读取确认期间，如果版本化写锁指示来自兄弟并行嵌套事务的冲突，则咨询信息以确定是否应破坏并行嵌套事务。在一个实现中，该信息被包含在版本化写锁和全局版本化写锁映射中。

[0012] 在又一其他实现中，对并行嵌套事务支持写锁获取。在试图获取并行嵌套事务的写锁时，读取并分析事务存储器字以确定是否能够获取该写锁。如果事务存储器字指示版本化写锁，则访问全局版本化写锁映射以取回指向第一写日志条目的写日志条目指针。

[0013] 在又一实现中，对并行嵌套事务支持悲观读取。为并行嵌套事务创建悲观重复检测数据结构。将对应于并行嵌套事务中的每一悲观读取的条目形成到该数据结构中。在提交并行嵌套事务时，将新的悲观读锁传递给直接父事务，并且在兄弟事务之间同步的情况下将一条目形成到直接父事务的单独的悲观重复检测数据结构中。该悲观重复检测数据结构还可用于从悲观读取到写锁的升级。

[0014] 在另一实现中，对并行嵌套事务支持重试操作。在作为并行嵌套事务或并行嵌套事务的子事务的事务执行重试时，为该重试注册事务读取集合。当决定将该重试传播通过该事务的并行嵌套父事务时，该读取集合保持注册并成为父事务读取集合的一部分。

[0015] 在又一实现中，写中止补偿映射可以就并行嵌套事务使用以检测和处理错误破坏的父事务。当在回退期间释放并行嵌套事务的新写锁时，创建写中止补偿映射。在并行嵌套事务回退时，在写中止补偿映射中创建对应于所释放的每一个新写锁的条目。

[0016] 提供本概述以便以简化形式介绍将在以下详细描述中进一步描述的一些概念。本概述不旨在标识所要求保护的主题的关键特征或必要特征，也不旨在用于帮助确定所要求保护的主题的范围。

[0017] 附图简述

[0018] 图 1 是一个实现的计算机系统的图示。

[0019] 图 2 是在图 1 的计算机系统上操作的一个实现的事务存储器应用程序的图示。

[0020] 图 3 是图 1 的系统的一个实现的处理流程图，其示出允许在单个父事务下嵌套多个并行事务所涉及的各阶段。

[0021] 图 4 是图 1 的系统的一个实现的处理流程图，其示出在提前知道并行嵌套事务的数量时分配父日志中的并行嵌套事务条目所涉及的各阶段。

[0022] 图 5 是图 1 的系统的一个实现的处理流程图，其示出在不提前知道并行嵌套事务的数量时分配父日志中的并行嵌套事务条目所涉及的各阶段。

[0023] 图 6 示出了一个实现的事务存储器字的示例结构。

[0024] 图 7 是图 1 的系统的一个实现的处理流程图，其示出使用版本化写锁来确保并行嵌套事务适当地嵌套和合成所涉及的各阶段。

[0025] 图 8 和 9 是图 1 的系统的一个实现的处理流程图，其示出适当地处理嵌套事务获取写锁所涉及的各阶段。

[0026] 图 10 是图 1 的系统的一个实现的处理流程图，其示出适当地处理嵌套事务释放任

一种写锁以便提交所涉及的各阶段。

[0027] 图 11 是图 1 的系统的一个实现的处理流程图，其示出适当地处理嵌套事务释放新写锁以便回退所涉及的各阶段。

[0028] 图 12 是图 1 的系统的一个实现的处理流程图，其示出适当地处理嵌套事务释放重复写锁以便回退所涉及的各阶段。

[0029] 图 13 是图 1 的系统的一个实现的处理流程图，其示出执行乐观读锁获取。

[0030] 图 14A-14D 是图 1 的系统的一个实现的处理流程图，这些流程图示出执行乐观读取确认。

[0031] 图 15 是图 1 的系统的一个实现的处理流程图，其示出确保对并行嵌套事务的正确的悲观读取操作所涉及的各阶段。

[0032] 图 16 是图 1 的系统的一个实现的处理流程图，其示出将悲观读锁升级到写锁所涉及的各阶段。

[0033] 图 17A-17D 是图 1 的系统的一个实现的处理流程图，这些流程图示出执行悲观读锁获取。

[0034] 图 18 是图 1 的系统的一个实现的处理流程图，其示出就并行嵌套事务允许重试正确地工作所涉及的各阶段。

[0035] 图 19 是图 1 的系统的一个实现的处理流程图，其示出就并行嵌套事务使用写中止补偿映射所涉及的各阶段。

[0036] 详细描述

[0037] 此处的技术和方法可以在事务存储器系统的一般上下文中描述，但本系统也用作除此之外的其它目的。在一个实现中，此处所描述的一个或多个技术可被实现为诸如微软®.NET 框架等框架程序内的、或来自为开发者提供开发软件应用程序的平台的任何其它类型的程序或服务的特征。在另一实现中，此处所描述的一个或多个技术被实现为处理开发在并发环境中执行的应用程序的其它应用程序的特征。

[0038] 如在背景部分中所描述的，如果嵌套事务的实施是与其包含(即，父)事务相同的隔离边界的一部分，则该嵌套事务被认为是封闭的。使用此处所描述的各种技术和方法，事务可同时具有多个封闭嵌套事务。这些封闭嵌套事务被称为“并行嵌套事务(PNT)”。单个封闭事务下的所有 PNT 都被称为该事务的“并行子事务”，并且该封闭事务被称为“并行父事务”。该并行父事务及其子事务被称为“并行嵌套”。PNT 的“兄弟”是封闭(处在某一嵌套级)在同一并行父事务中的另一 PNT。在一个实现中，每一个 PNT 与普通封闭嵌套事务非常像地执行：其实施被隔离在封闭事务内并且只在并行父事务提交时才在并行嵌套之外变得可见。然而，每一个 PNT 都与其兄弟隔离，就像它是顶层事务那样。PNT 的实施只在它提交时才变得对其兄弟可见。

[0039] 如图 1 所示，用于实现本系统的一个或多个部分的示例性计算机系统包括诸如计算设备 100 等计算设备。在其最基本的配置中，计算设备 100 通常包括至少一个处理单元 102 和存储器 104。取决于计算设备的确切配置和类型，存储器 104 可以是易失性的(如 RAM)、非易失性的(如 ROM、闪存等)或是两者的某种组合。该最基本配置在图 1 中由虚线 106 来示出。

[0040] 另外，设备 100 还可具有附加特征 / 功能。例如，设备 100 还可包含附加存储(可

移动和 / 或不可移动)，包括但不限于磁盘、光盘或磁带。这样的附加存储在图 1 中由可移动存储 108 和不可移动存储 110 示出。计算机存储介质包括以用于存储诸如计算机可读指令、数据结构、程序模块或其它数据等信息的任何方法或技术来实现的易失性和非易失性、可移动和不可移动介质。存储器 104、可移动存储 108 和不可移动存储 110 都是计算机存储介质的示例。计算机存储介质包括但不限于，RAM、ROM、EEPROM、闪存或其它存储器技术、CD-ROM、数字多功能盘（DVD）或其它光存储、磁带盒、磁带、磁盘存储或其它磁存储设备、或者可用于存储所需信息并且可由设备 100 访问的任何其它介质。任何这样的计算机存储介质都可以是设备 100 的一部分。

[0041] 计算设备 100 包括允许计算设备 100 与其它计算机 / 应用程序 115 进行通信的一个或多个通信连接 114。设备 100 还可以具有诸如键盘、鼠标、笔、语音输入设备、触摸输入设备等输入设备 112。还可以包括诸如显示器、扬声器、打印机等输出设备 111。这些设备在本领域中公知且无需在此处详细讨论。在一个实现中，计算设备 100 包括事务存储器应用程序 200。事务存储器应用程序 200 将在图 2 中更详细地描述。

[0042] 现在转向图 2 并继续参考图 1，示出了在计算设备 100 上操作的事务存储器应用程序 200。事务存储器应用程序 200 是驻留在计算设备 100 上的应用程序中的一个。然而，可以理解，事务存储器应用程序 200 可另选地或另外地被具体化为一个或多个计算机上的计算机可执行指令和 / 或与图 1 所示的不同变型。另选地或另外地，事务存储器应用程序 200 的一个或多个部分可以是系统存储器 104 的一部分、可以在其它计算机和 / 或应用程序 115 上、或可以是计算机软件领域的技术人员能想到的其它此类变型。

[0043] 事务存储器应用程序 200 包括负责执行在此描述的技术中的部分或全部的程序逻辑 204。程序逻辑 204 包括用于允许在单个父事务下嵌套多个并行事务的逻辑 206（如以下参考图 3 所描述的）；用于允许并行嵌套事务正确地与其他封闭嵌套事务嵌套和合成的逻辑 208（如以下参考图 7-14 所描述的）；用于允许并行嵌套事务在具有独立日志的情况下执行并且在低争用的情况下将锁所有权和日志传递给父事务的逻辑 210（如以下参考图 4-5 所描述的）；用于启用并行嵌套事务以便在同一事务中进行乐观和悲观读取的逻辑 212（如以下参考图 15-17 所描述的）；用于启用并行嵌套事务来进行就地或缓冲写入的逻辑 214（如以下参考图 7、11 和 12 所描述的）；用于就并行嵌套事务允许重试正确地工作的逻辑 216（如以下参考图 18 所描述的）；用于就并行嵌套事务允许写中止补偿映射工作的逻辑 218（如以下参考图 19 所描述的）；以及用于操作应用程序的其他逻辑 220。

[0044] 现在转向图 3-19 并继续参考图 1-2，更详细地描述了用于实现事务存储器应用程序 200 的一个或多个实现的各阶段。在某些实现中，图 3-19 的过程至少部分地在计算设备 100 的操作逻辑中实现。图 3 示出了允许在单个父事务下嵌套多个并行事务所涉及的各阶段的一个实现。该过程开始于起始点 240，在那里提供将创建准备好执行的一组并行嵌套事务的功能或其他特征（阶段 242）。全部一次性或惰性地在需要时为给定父事务创建一组并行嵌套事务（阶段 244）。使用特殊逻辑来事务性地执行、提交、回退、重新执行和重试并行嵌套事务（阶段 246）。当并行嵌套事务完成时，必须在该并行嵌套事务及其所有并行兄弟都已完成后破坏该事务（阶段 248）。该过程在结束点 250 处结束。

[0045] 图 4 和 5 示出了如何分配父日志中的并行嵌套事务条目。并行嵌套事务能够在具有独立日志的情况下执行并且在根据图 4 和 5 所描述的分配技术中的一种来分配后在低争

用的情况下将锁所有权、日志和其他数据传递给父事务。现在转向图 4，示出了用于在提前知道并行嵌套事务的数量时分配父日志中的并行嵌套事务条目的一个实现。该过程开始于起始点 270，在那里因为提前知道并行嵌套事务的数量，所以在任何并行嵌套事务（PNT）开始执行之前为嵌套中的每一个 PNT 创建一个并行嵌套事务条目（PNTE）（阶段 272）。当在回退或提交处理期间需要时，并行嵌套事务在进行最小同步的情况下从父事务中取回并行嵌套事务条目。。在一个实现中，预先分配的 PNTE 被保持在父事务中的阵列中。PNT 可通过对下一可用并行嵌套事务索引的简单的比较和交换操作来获取指向下一可用 PNTE 的指针（阶段 274）。比较和交换（CAS）是原子地执行给定值和给定存储单元的内容之间的比较并且如果它们匹配，则将给定的新值存储在该存储单元中的操作。如果它们不匹配，则不采取动作。存在许多用于在许多不同的 CPU 和操作系统上执行 CAS 操作的方法，一些是硬件，一些是软件。如此处所使用的，术语 CAS 意指一般覆盖所有这些方法。

[0046] 并行嵌套事务用信息来填充父日志中的并行嵌套事务条目（阶段 276）。在一个实现中，该信息包括指向子事务的日志的指针和指向子事务的写中止补偿映射和 / 或悲观读取对象表（在适当时）的指针。该写中止补偿映射在图 19 中更详细地描述。悲观读取对象表在图 15 中更详细地描述。该过程在结束点 278 处结束。

[0047] 图 5 示出了在不提前知道并行嵌套事务的数量时分配父日志中的并行嵌套事务条目所涉及的各阶段的一个实现。该过程开始于起始点 290，在那里在创建并行嵌套事务时为新并行嵌套事务条目分配父日志中的空间（阶段 292）。当并行嵌套事务提交时，在父日志中创建并行嵌套事务条目（阶段 294）。同步对父日志的访问以移动该父事务的当前日志点以便获取对并行嵌套事务条目的下一可用空间的访问（阶段 296）。并行嵌套事务用信息来填充父日志中的并行嵌套事务条目（阶段 298）。在一个实现中，该信息包括指向子事务的日志的指针和指向子事务的写中止补偿映射和 / 或悲观读取对象表（在适当时）的指针。该写中止补偿映射在图 19 中更详细地描述。悲观读取对象表在图 15 中更详细地描述。该过程在结束点 300 处结束。

[0048] 图 6 示出了一个实现的事务存储器字（TMW）的示例结构 320，该结构具有用于标记各种锁定状态的各个位 322。在一个实现中，TMW 与每一对象相关联。在一个实现中，使用单个硬件字表示；但允许存储锁定信息的对 TMW 的众多其他表示是可能的。在此处所描述的示例结构中，TMW 可指示相关联的对象是写锁定的或者不是写锁定的。TMW 中的一个位专用于该区别。如果 TMW 不是写锁定的，则它包含被称为版本 / 计数对 324（V/C 对）的版本号和悲观读取者计数。在该状态中，TMW 中的某数量的位记录对象的版本号，而其余位表示当前保持该对象上的悲观读锁的事务数量的计数。当事务保持对象中的写锁时，该锁可具有两种类型（通过 TMW 中的另一位来进行区分）。通常，写锁定的 TMW 中的其余位包含指向锁定事务的写日志中的条目 326；该写日志条目（WLE）包含关于对象和锁定的其他信息。例如，WLE 可能包含在获取写锁之前的对象的 TMW 值；在另一实现中，该 WLE 可能包含对其作出未提交修改的对象的“影子副本”。在另一状态中，写锁定的 TMW 包含版本化写锁（VWL）。此处，TMW 中的其余位（被称为 VWL[V/C]328）表示在该 TMW 仍旧是写锁定时的对象的版本号以及悲观读取者的计数，这类似于 V/C 对。

[0049] 在继续至关于如何使用版本化写锁的更多详细讨论之前，让我们首先探查将帮助示出对版本化写锁（VWL）的需求的示例。假设存在就地 STM 系统，以及获取对对象 O1 的写

锁的顶层事务 Tx1。O1 的 TMW 被设为 WLE1，其是 Tx1 的日志中表示 Tx1 的对 O1 的写锁的写日志条目。现在，假设引入两个 PNT，即 Tx2 和 Tx3，且 Tx1 作为父并行事务。Tx2 和 Tx3 是并行嵌套中的兄弟事务并且并发执行。Tx2 和 Tx3 可访问被 Tx1 锁定的数据，但它们必须彼此隔离。因此，虽然 Tx2 和 Tx3 都可访问 O1，但它们必须不被允许同时这样做。现在，假设 Tx3 希望从 O1 读取。它执行乐观读操作，从而在其日志中创建将 O1 的值记录为 WLE1 的乐观读日志条目。接着，假设 Tx2 对 O1 写入。Tx2 将获取对 O1 的写锁，并将 O1 的 TMW 设为 WLE2，这作为 Tx2 的日志中的写日志条目。WLE2 记录 WLE1 是 O1 的 TMW 的前一值。Tx2 现在可对 O1 的各字段写入，并且以就地写入的方式这样做。当 Tx2 继续执行时，它读取包含来自 Tx2 的未提交写入的 O1 的各字段。Tx3 通过定义来破坏并且应当回退。然而，如果出于任何原因 Tx2 在 Tx3 试图提交之前回退，则 Tx2 必须释放其对 O1 的写锁。为此，Tx2 通常将 O1 的 TMW 设回为 WLE1。但现在当 Tx3 试图提交时，它将看到 O1 的 TMW 包含在 Tx3 首先读取 O1 时所设置的相同的值。在这种情况下，Tx3 将表现为有效，并且将无法识别它从 Tx2 读取了未提交的写入。因此，当 Tx2 回退时它必须将 O1 的 TMW 设为除 WLE1 之外的某个值，并且它必须以确保该系统中的其他事务（PNT 兄弟或其他顶层事务）识别出 O1 仍旧被 Tx1 写锁定的方式这样做。这通过将 O1 的 TMW 设为版本化写锁（VWL）并在全局版本化写锁映射（VWLM）中形成指示 Tx1 持有对 O1 的写锁的条目来实现。VWL 和 VWLM 的细节和用途在下文中描述。该示例示出了其中 VWL 是必需的一种情况。然而，本领域的普通技术人员将会理解，存在众多其中可使用 VWL 的情况，如随着在本章节的其余部分中详细描述用于锁获取和释放的各过程将变得显而易见的。

[0050] 图 7 示出了使用版本化写锁来确保并行嵌套事务适当地嵌套和合成所涉及的各阶段的一个实现。该过程开始于起始点 340，在那里提供可以是版本 / 计数对、写日志条目（WLE）或版本化写锁（VWL）中的一个的事务存储器字（阶段 342）。当事务存储器字（TMW）变为版本化写锁时，在全局版本化写锁映射（VWLM）中形成涉及版本化写锁所替换的旧写日志条目指针的条目，该条目通过对象地址来索引（阶段 344）。当看到版本化写锁时，咨询全局版本化写锁映射以便将该版本化写锁转换成写日志条目指针，并且对写日志条目指针进行普通处理（阶段 346）。无论是保持在 V/C 对还是版本化写锁中，事务存储器字中的版本号在提交或中止处理期间都始终递增（阶段 348）。就在将版本化写锁放置在事务存储器字中之前在全局版本化写锁映射中形成一条目，并且只要该事务存储器字包含该版本化写锁，该条目就保留（阶段 350）。在任何时刻，事务都必须获取访问全局版本化写锁映射的全局锁（阶段 352）。该过程在结束点 354 处结束。

[0051] 图 8 和 9 示出了适当地处理嵌套事务获取写锁所涉及的各阶段的一个实现。新写锁是在事务嵌套中首次获取的锁，而重复写锁是嵌套事务可获取的对先辈事务当前已写锁定的对象的写锁。该过程开始于起始点 370，在那里嵌套事务读取事务存储器字（阶段 372）。如果该事务存储器字不是 V/C 对（判定点 374），则该过程继续至在下一节中描述的图 9 阶段 404。如果事务存储器字是 V/C 对且 C（悲观读取者的计数）大于 0（判定点 375），则执行图 16 的过程以处理悲观读取到写锁的升级。如果事务存储器字是 V/C 对且 C 等于 0（判定点 375），则这指示该事务存储器字未被悲观地锁定以供任何事务读取或写入，因此允许获取新写锁。为此，该系统形成新的写日志条目以记录版本号并将该写日志条目附加到该事务的写日志（阶段 376）。然后执行比较和交换以便将事务存储器字值从 V/C 对切换

至写日志条目指针(例如,WLE*)(阶段 378)。如果比较和交换成功(判定点 380),则成功获取锁(阶段 382)。如果比较和交换不成功(判定点 380),则存在冲突(阶段 384)并且未成功获取锁。该过程在结束点 386 处结束。

[0052] 继续至图 9,如果 TMW 不是 V/C 对(图 8 上的判定点 374),并且如果事务存储器字不是版本化写锁(判定点 404),则它是指向 WLE_A 的 WLE* 并且该过程在阶段 408 处继续。如果事务存储器字是版本化写锁(判定点 404),则使用全局版本写锁映射来取回指向 WLE_A 的底层 WLE* (阶段 406)。如果 WLE_A 不被任何先辈事务所拥有(判定点 408),则存在冲突(阶段 410)并且该过程在结束点 420 处结束。如果 WLE_A 被一先辈拥有(判定点 408),则形成新 WLE_B 以记录 WLE_A 并且将 WLE_B 附加到事务的写日志(阶段 412)。然后执行比较和交换以便将事务存储器字值从 WLE_A* 切换至 WLE_B* (阶段 414)。如果比较和交换成功(判定点 416),则成功获取锁(阶段 418)。如果比较和交换不成功(判定点 416),则存在冲突(阶段 410)并且未成功获取锁。该过程在结束点 420 处结束。

[0053] 图 10 示出了适当地处理嵌套事务释放任一种写锁以便提交所涉及的各阶段的一个实现。该过程开始于起始点 440,在那里在提交处理期间在事务写日志中遇到 WLE_X (阶段 442)。简单地将写锁的所有权和 WLE_X 传递给直接父事务(阶段 444)。通过在提交时将所有权从并行嵌套事务传递给父事务,其他兄弟事务现在发现它们现在能够为自己获取写锁。同样,获取重复写锁的动作防止兄弟事务能够为它们自己获取写锁。该过程在结束点 446 处结束。

[0054] 图 11 示出了适当地处理嵌套事务释放新写锁以便回退所涉及的各阶段的一个实现。该过程开始于起始点 460,在那里在回退处理期间在事务写日志中遇到 WLE_X(阶段 462)。系统检查以查看 WLE_X 表示什么类型的写锁(阶段 464)。如果 WLE_X 不表示新写锁(判定点 466),则执行用于释放重复写锁的逻辑(阶段 471),如在图 12 中参考一个实现所描述的。如果 WLE_X 表示新写锁(判定点 466),则取回存储在 WLE_X 中的前一版本号(阶段 468)。从 WLE_X 中取回事务存储器字的位置并且使用普通存储操作来释放写锁以便将事务存储器字值改为适用于该系统类型(就地或缓冲)的 V/C 对(阶段 470)。在缓冲系统的一个实现中,将事务存储器字值改回到表示原始版本号。在就地系统的一个实现中,则将事务存储器字值改为表示原始版本号加一。该过程在结束点 472 处结束。

[0055] 图 12 示出了适当地处理嵌套事务释放重复写锁以便回退所涉及的各阶段的一个实现。在一个实现中,该过程仅用于在回退时递增对象的版本号的系统,即就地系统。在缓冲系统的某些实现中,不在回退期间增加版本号。在这些系统中,用于释放新写锁的过程(图 11)可用于释放重复写锁。该过程开始于起始点 490,在那里在回退处理期间在事务写日志中遇到 WLE_X(阶段 492)。系统检查以查看 WLE_X 表示什么类型的写锁(阶段 494)。如果锁不是重复写锁(判定点 496),则执行用于释放新写锁的逻辑(阶段 511),如在图 11 中参考一个实现所描述的。如果锁是重复写锁,则获取用于同步对全局版本化写锁映射的访问的全局锁(阶段 498)。从 WLE_X 中取回原始写日志条目指针 WLE_Y* 和事务存储器字的位置(阶段 500)。形成对应于涉及 WLE_Y 的对象的新的全局版本化写锁映射条目(阶段 502)。然后释放用于同步对全局版本化写锁映射的访问的全局锁(阶段 504)。从 WLE_X 中取回原始版本号(阶段 506),并且形成作为原始版本号 +1 的新版本化写锁值(阶段 508)。使用普通存储操作来释放写锁以便将事务存储器字值从 WLE_X* 改为新版本化写锁(阶段 510)。该过程在

结束点 512 处结束。

[0056] 图 13 是图 1 的系统的一个实现的处理流程图，其示出执行乐观读锁获取。该过程开始于起始点 530，在那里在事务执行对对象的乐观读取时读取对象的当前 TMW 值(阶段 532)。创建乐观读日志条目(阶段 534)并且用当前 TMW 值和该 TMW 的位置来填充该读日志条目(阶段 536)。将读日志条目附加到事务的读日志(阶段 538)。在一个实现中，该过程对于以下所有类型的事务都是相同的：顶层、简单嵌套或者并行嵌套事务。该过程在结束点 540 处结束。

[0057] 图 14A-14D 是图 1 的系统的一个实现的处理流程图，这些流程图示出执行乐观读取确认。该过程开始于起始点 560，在那里在试图提交事务或以其他方式确定该事务是否有效时考虑该事务的读日志中的每一个乐观读日志条目(阶段 562)。从读日志条目中取回原始 TMW 值(阶段 564)，并且读取当前 TMW 值(阶段 566)。注意，在每一种情况下，如果系统正在使用写中止补偿映射(WACM)(在图 19 中描述)，则只要在两个版本号中存在差异，就咨询当前聚集 WACM (在确认或提交处理期间形成)。如果原始 TMW 是 V/C 对(判定点 568)，则执行图 14B 中所描述的过程。如果原始 TMW 是 WLE* (判定点 570)，则执行图 14C 中所描述的过程。如果原始 TMW 既不是 V/C 对也不是 WLE*，则该原始 TMW 是 VWL(阶段 572)，并且执行图 14D 的过程。让我们更详细地看一下这些情况中的每一种。

[0058] 图 14B 涵盖了关于当原始 TMW 是 V/C 对时(判定点 568)的一个实现中的在乐观读取确认期间执行的示例性过程的更多细节。如果当前 TMW 是 V/C 对(判定点 590)，并且原始 TMW 和当前 TMW 中的版本号匹配(判定点 592)，则事务有效(阶段 594)。如果当前 TMW 是 V/C 对(判定点 590)，并且原始 TMW 和当前 TMW 中的版本号不匹配(判定点 592)，则事务无效(阶段 596)。

[0059] 如果当前 TMW 改为是 WLE*(判定点 598)，并且如果事务拥有 WLE，并且 WLE 中的已保存的版本号匹配旧 TMW (判定点 600)，则该事务有效(阶段 594)。如果当前 TMW 不是 V/C 对(判定点 590)，并且当前 TMW 不是 WLE* (判定点 598)，则 TMW 是 VWL (阶段 602)。使用已锁定对象的地址来在 VWLM 中进行同步查找。如果不存在 VWLM 条目(判定点 604)，则事务无效(阶段 596)。如果存在条目(判定点 604)，则使用该 VWLM 条目来取回 VWL 所替换的 WLE*。如果事务拥有 WLE，并且 WLE 中的已保存的版本号匹配旧 TMW (判定点 606)，则事务有效(阶段 594)。否则，事务无效(阶段 596)。该过程在结束点 608 处结束。

[0060] 现在转向图 14C，示出了当原始 TMW 是 WLE* 时的一个实现中的在乐观读取确认期间执行的示例性过程。如果当前 TMW 是 V/C 对(判定点 620)，则当前事务无效(阶段 630)。如果当前 TMW 不是 V/C 对(判定点 620)，而改为是 WLE*(判定点 624)，则系统检查以查看原始和当前 TMW 是否匹配，以及当前事务或任何先辈事务是否拥有 WLE (判定点 626)。如果满足这两个准则，则事务有效(阶段 628)。否则，事务无效(阶段 630)。

[0061] 如果当前 TMW 不是 V/C 对(判定点 620)，并且不是 WLE* (判定点 624)，则当前 TMW 是 VWL(阶段 632)。如果该事务或任何先辈事务拥有来自原始 TMW 的 WLE，并且如果保存在该 WLE 中的版本号匹配 VWL 中的版本号(判定点 634)，则事务有效(628)。否则，事务无效(阶段 630)。该过程在结束点 636 处结束。

[0062] 现在转向图 14D，示出了当原始 TMW 是 VWL 时的一个实现中的在乐观读取确认期间执行的示例性过程。如果当前 TMW 是 V/C 对(判定点 650)，则事务由于冲突而无效(阶段

660)。如果当前 TMW 不是 V/C 对(判定点 650),而改为是 WLE*(判定点 654),则系统检查以查看当前事务或任何先辈事务是否拥有 WLE,以及存储在该 WLE 中的版本号是否匹配 VWL 中的版本(判定点 656)。如果满足这两个准则,则事务有效(阶段 658)。否则,事务无效(阶段 660)。

[0063] 如果当前 TMW 不是 V/C 对(判定点 650),并且当前 TMW 不是 WLE*(判定点 654),则当前 TMW 是 VWL (阶段 662)。在 VWLM 中执行查找以便将 VWL 转换成 WLE* (阶段 664)。如果未找到条目(判定点 666),则事务无效(阶段 660)。否则,如果找到条目(判定点 666),则系统检查以查看原始和当前 VWL 的版本号是否匹配,以及该事务或任何先辈事务是否拥有在 VWLM 中找到的对应于 TMW 的 WLE (判定点 668)。如果满足这两个准则,则事务有效(阶段 658)。否则,事务无效(阶段 660)。该过程在结束点 670 处结束。

[0064] 具有简单的封闭嵌套事务的系统中的正确的悲观读取操作需要使用被称为悲观读取对象表(PROT)的重复检测数据结构。每一个顶层事务在事务开始时或者惰性地在首次悲观读取操作时创建 PROT。当该事务或任何后代事务试图获取对一对象的悲观读锁时,该事务咨询 PROT 以确定是否已经获取悲观读锁。如果该对象在 PROT 中,则该对象已被事务嵌套读锁定。如果该对象不在 PROT 中,并且如果该对象当前未被另一事务写锁定,则使用 CAS 操作递增存储在该对象的 TMW 中的 V/C 对中的 C (悲观读取者的计数)来获取悲观读锁。如果该 CAS 操作成功,则在 PROT 中形成一条目以记录该嵌套现在已锁定该对象以便进行悲观读取的事实。当释放悲观读锁时,在顶层提交或回退期间,递减 C (再次用 CAS),并且移除 PROT 条目。让我们现在看一下如何就并行嵌套事务使用 PROT。

[0065] 图 15 示出了确保对于并行嵌套事务的正确的悲观读取操作所涉及的各阶段的一个实现。该过程开始于起始点 690,在那里并行嵌套事务在初始化期间,或者惰性地在并行嵌套事务获取第一个悲观读锁期间创建悲观重复检测数据结构(被称为 PROT,如上所述)(阶段 692)。在事务试图将悲观读锁升级到写锁时,系统使用该数据结构。将对应于并行嵌套事务或任何后续子事务所作出的对对象的首次悲观读取的条目形成到 PROT 中(阶段 694)。在提交时,将新的悲观读锁传递给直接父事务,并且形成适当的父 PROT 条目(阶段 696)。释放重复读锁以允许兄弟事务在并行子事务提交后获取写访问(阶段 698)。系统然后破坏与重复的悲观读锁相关联的日志条目(阶段 700)。在提交后,可以用其余事务来破坏子事务的 PROT (阶段 702)。该过程在结束点 704 处结束。

[0066] 图 16 示出了将悲观读锁升级到写锁所涉及的各阶段的一个实现。该过程开始于起始点 720,在那里发现所需要的对对象的写锁已经对悲观读取开放(阶段 722)。查询子事务的 PROT (阶段 724)以查看读取者的当前计数并且决定当前事务是否能够计入(account for)所有这些读取(阶段 726)。如果结果计入所有悲观读取者,则子事务可试图照常升级至写锁(阶段 728)。如果仍旧存在未计入的读取者,则子事务必须查询所有先辈事务的 PROT 以确定其是否可以升级到写锁(阶段 730)。如果任何先辈事务是并行父事务,则必须考虑该并行父事务的 PROT 以及已经提交的任何并行兄弟事务的 PROT。这些兄弟事务的 PROT 经由 PNTE 保持在并行父事务的日志中。需要适当的同步来确保存在对这些兄弟事务的 PROT 的无竞争访问。该同步通过确保 PNT 不在将其 PROT 放置在相关联的 PNTE 中后访问该 PROT 来实现。如果先辈事务和任何已提交的并行兄弟事务补偿额外的悲观读取者,则子事务可尝试照常升级至写锁(阶段 732)。在一个实现中,升级如下实现。形成新的写日志并将其

添加到 PNT 的日志。将当前 TMW 值放置在 WLE 中以便在回退和提交处理期间使用。如果当前 TMW 值是 VWL，则在进行适当的同步的情况下使用 VWLM 来首先将 VWL 转换成 WLE*。使用 CAS 来获取写锁。如果 CAS 成功，则升级奏效，否则存在冲突。该过程在结束点 734 处结束。

[0067] 现在是时候描述图 17A-17D 所示的用于在处于并行嵌套事务中时执行悲观读锁获取的示例性过程。该过程开始于 17A，在那里读取当前 TMW（阶段 752）。如果 TMW 是 V/C 对（判定点 754），则执行图 17B 中所描述的过程。如果 TMW 是 WLE*（判定点 756），则执行图 17C 中所描述的过程。如果 TMW 不是 V/C 对（判定点 754），并且不是 WLE*（判定点 756），则 TMW 是 VWL（阶段 758），并且执行图 17D 中所描述的过程。现在将更详细地看一下这些情况中的每一种。

[0068] 现在转向图 17B，如果 TMW 是 V/C 对，则咨询事务的 PROT 以确定该事务是否已经持有对对象的悲观读锁（判定点 769）。如果存在 PROT 条目，则该过程在结束点 776 处结束。如果不存在 PROT 条目（判定点 769），在使用比较和交换（CAS）来递增悲观读取者的计数 C（阶段 770）。如果 CAS 成功（判定点 772），则获取锁并且形成 PROT 条目（阶段 774）。如果 CAS 失败（判定点 772），则重试，这通过参考图 17A 的阶段 752 示出。

[0069] 如图 17C 所示，如果 TMW 是 WLE*，并且当前事务或任何先辈事务拥有该 WLE（判定点 790），则系统确定拥有该 WLE 的事务在并行父事务之下还是之上（判定点 792）。如果拥有该 WLE 的事务在并行父事务之下（判定点 792），则悲观读取成功（阶段 794）。如果拥有该 WLE 的事务在并行父事务之上（判定点 792），则系统将 TMW 切换至 VWL 以便与兄弟事务进行协调。为此，在进行适当同步的情况下形成 VWLM 条目以记录 WLE*（阶段 796），并且从存储在 VWL 中的版本号中形成 VWL 且将 C（悲观读取者的计数）设为 1（阶段 798）。用 CAS 来在 TMW 中设置新 VWL（阶段 800）。如果 CAS 成功，则形成 PROT 条目并且获取悲观读锁（阶段 806）。如果 CAS 不成功，则移除 VWLM 条目（阶段 804），并且通过返回到图 17A 的阶段 752 来重试。该过程在结束点 810 处结束。

[0070] 如图 17D 所示，如果 TMW 是 VWL，则在进行适当同步的情况下经由 VWLM 将 VWL 转换成 WLE*（阶段 820）。如果该事务或任何先辈事务拥有该 WLE（判定点 822），则使用 CAS 来递增 TMW 中的 VWL 中的 C（悲观读取者的计数）（阶段 824）。如果 CAS 成功（判定点 826），则形成 PROT 条目并且获取锁。如果 CAS 失败，则通过继续至图 17A 的阶段 752 来重试。如果该事务或任何先辈事务不拥有该 WLE（判定点 822），则存在冲突（阶段 830）。该过程在结束点 832 处结束。

[0071] 图 18 示出了就并行嵌套事务允许重试正确地工作所涉及的各阶段的一个实现。在深入研究图 18 的细节以及讨论就并行嵌套事务允许重试工作之前，首先有必要提供关于一个实现的重试操作的某些背景信息。重试操作允许在事务之间进行基本通信。在事务执行重试操作时，回退该事务的实施并挂起执行直到该事务读取的某物改变。当检测到改变时，重新执行事务。重试操作可用于某些非常常见的数据结构，如阻塞队列。例如，事务可检查以查看队列是否为空并且如果该队列为空，则重试，或者如果该队列不为空，则移除一元素。该事务将在队列保持不变时阻塞并且在队列状态改变时重新执行，这给予了该事务另一次完成机会。

[0072] 在一个实现中，在事务执行重试操作时，系统为对待重试事务的读取集合中的每一读取注册。重试事务等待读取集合中的某物已改变的通知。从释放写锁的特定事务发出

等待通知。事务以两种方式中的一种知道是否需要通知。以第一种方式，如果事务存储器字在写锁获取期间包含等待者位，则在释放期间在对象等待者映射中查找事务存储器字，并发信号通知每一等待事务。以第二种方式，如果写事务在释放所有写锁后发现等待事务的全局计数大于 0，则该写事务将使用事务等待者映射来确定哪些事务（如果有的话）正在等待写入位置并需要发信号通知。在每一种情况下，使用普通存储操作来释放写锁。

[0073] 在另一实现中，以回退仅重试嵌套事务和等待其读取集合来开始渐进式重试操作。在等待某一特定时间或者满足某一其他条件后，执行退避过程以回退重试事务的直接父事务，从而增加原始等待集合的大小。重复该退避过程直到回退最顶层的父事务，以便增加对每一个下一父事务的附加等待。该聚集等待集合与最顶层的父事务相关联并且任何通知都将导致重新执行该最顶层的父事务。

[0074] 现在参考图 18，现在将讨论对如何能够就并行嵌套事务使用重试的解释。该过程开始于起始点 850，当并行嵌套事务或任何顺序后代执行重试时，系统照常为重试注册事务的读取集合（阶段 852）。该系统包括用于确定在将重试操作扩展成包括事务的先辈事务中的某一子集之前重试事务应等待多久的试探法（阶段 854）。调整这些对于并行嵌套事务的试探法并允许延长的等待时间，只要其他并行兄弟事务是活动的。如果超出该延长的等待时间，则将重试操作扩展成包括父事务（阶段 856）。在进一步传播之前考虑兄弟事务的状态（阶段 858）。当决定将重试传播通过并行嵌套父事务时，破坏所有子事务，并且然后必须在回退重试父事务之前完成（阶段 860）。当决定将重试传播通过父事务时，并行嵌套事务的读取集合保持注册并成为父事务读取集合的一部分（阶段 862）。注意，将已被提交到父事务中的任何并行兄弟事务的读取集合与该父事务的读取集合一起成为等待集合的一部分。被将重试传播至父事务的决定中止的任何并行兄弟事务不对该等待集合作出贡献。该过程在结束点 864 处结束。

[0075] 图 19 示出了在就并行嵌套事务使用写中止补偿映射所涉及的各阶段的一个实现。在深入研究图 19 的细节以及如何能够就并行嵌套事务使用写中止补偿映射之前，有必要提供关于一个实现的写中止补偿映射的某些背景信息。

[0076] 写中止补偿映射可用于检测使用就地写入的事务存储器系统中的嵌套子事务的遭错误破坏的父事务。写中止补偿映射（或其他存储机制）跟踪为回退的每一嵌套事务所释放的每一个锁的释放计数。嵌套事务释放写锁定的次数被记录在它们相应的写中止补偿映射中。可以在父事务的确认期间使用释放计数来确定明显无效的乐观读取实际上是否是有效的。

[0077] 在一个实现中，在处理父事务日志时，所看到的对于嵌套子事务的任何写中止补偿映射被组合到父事务中的聚集的写中止补偿映射中。如果乐观读取由于版本号不匹配而未能确认，则咨询聚集的写中止补偿映射来取回对于嵌套子事务的特定变量的写锁释放计数。如果版本号差与嵌套子事务的写锁释放计数正好匹配，则乐观读取是有效的。

[0078] 现在返回到图 19，让我们看一下如何能够就并行嵌套事务使用写中止补偿映射。该过程开始于起始点 880，在那里在回退并行嵌套事务期间释放新写锁时创建写中止补偿映射（WACM）（阶段 882）。当嵌套事务回退时，它为所释放的每一个新写锁创建 WACM 条目（阶段 884）。当在提交期间将并行嵌套事务的日志的所有权传递给父事务时，在逻辑上将 WACM 放在该日志的前面（阶段 886）。如上所述，就像任何其他嵌套事务的 WACM 那样将在父事务

回退期间使用这些 PNT WACM，并且这些 PNT WACM 将确保对父事务乐观读取的正确确认（阶段 888）。该过程在结束点 890 处结束。

[0079] 尽管用结构特征和 / 或方法动作专用的语言描述了本主题，但可以理解，所附权利要求书中定义的主题不必限于上述具体特征或动作。相反，上述具体特征和动作是作为实现权利要求的示例形式公开的。落入在此所述和 / 或所附权利要求所描述的实现的精神的范围内的所有等效方案、更改和修正都期望受到保护。

[0080] 例如，计算机软件领域普通技术人员将认识到，此处所讨论的示例可以在一个或多个计算机上不同地组织来包括比这些示例中所描绘的更少或更多选项或特征。

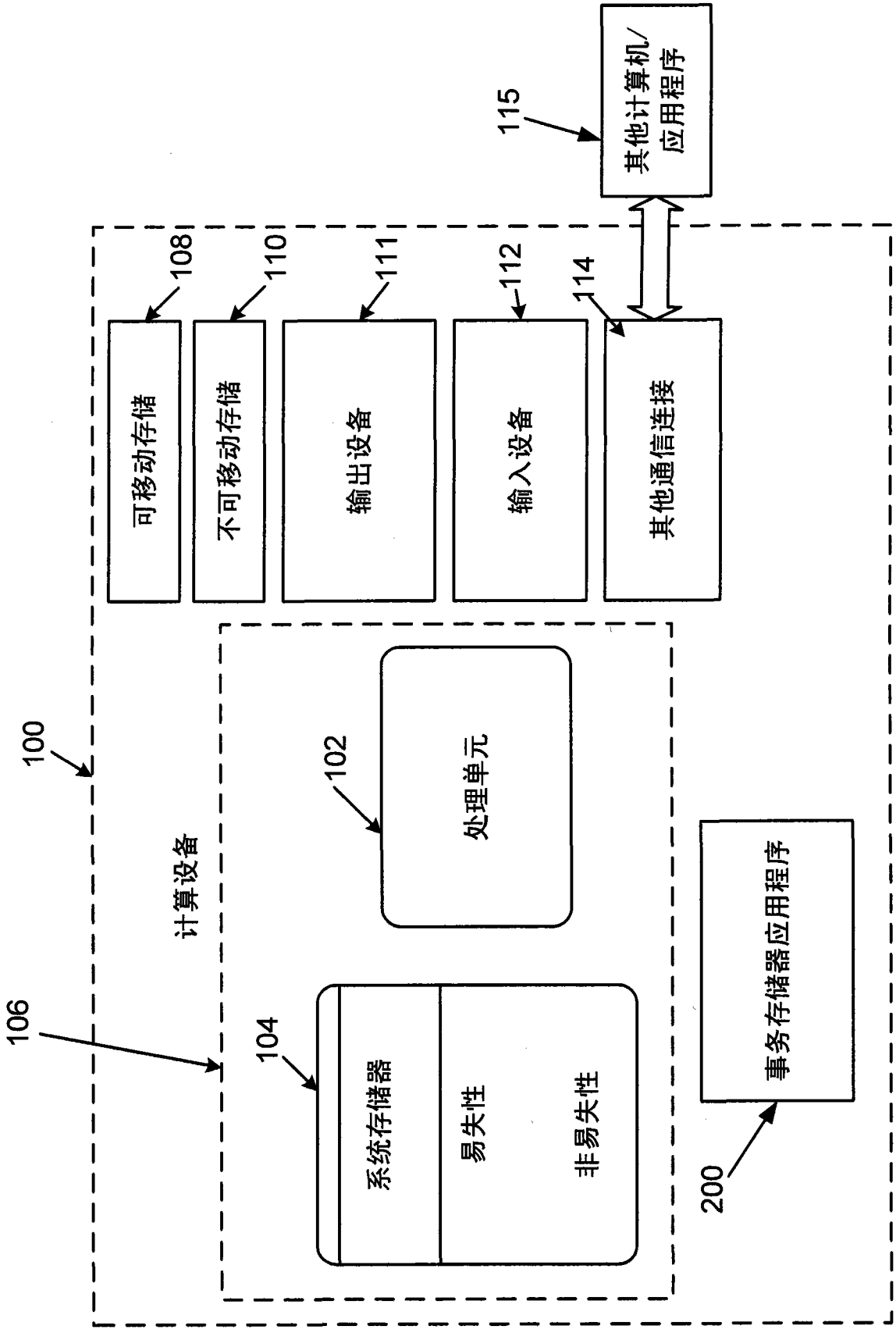

其他计算机/
应用程序
115
108
110
111
112
114
可移动存储
不可移动存储
输出设备
输入设备
其他通信连接
100
计算设备
102
处理单元
106
104
系统存储器
易失性
非易失性
事务存储器应用程序
200

图 1

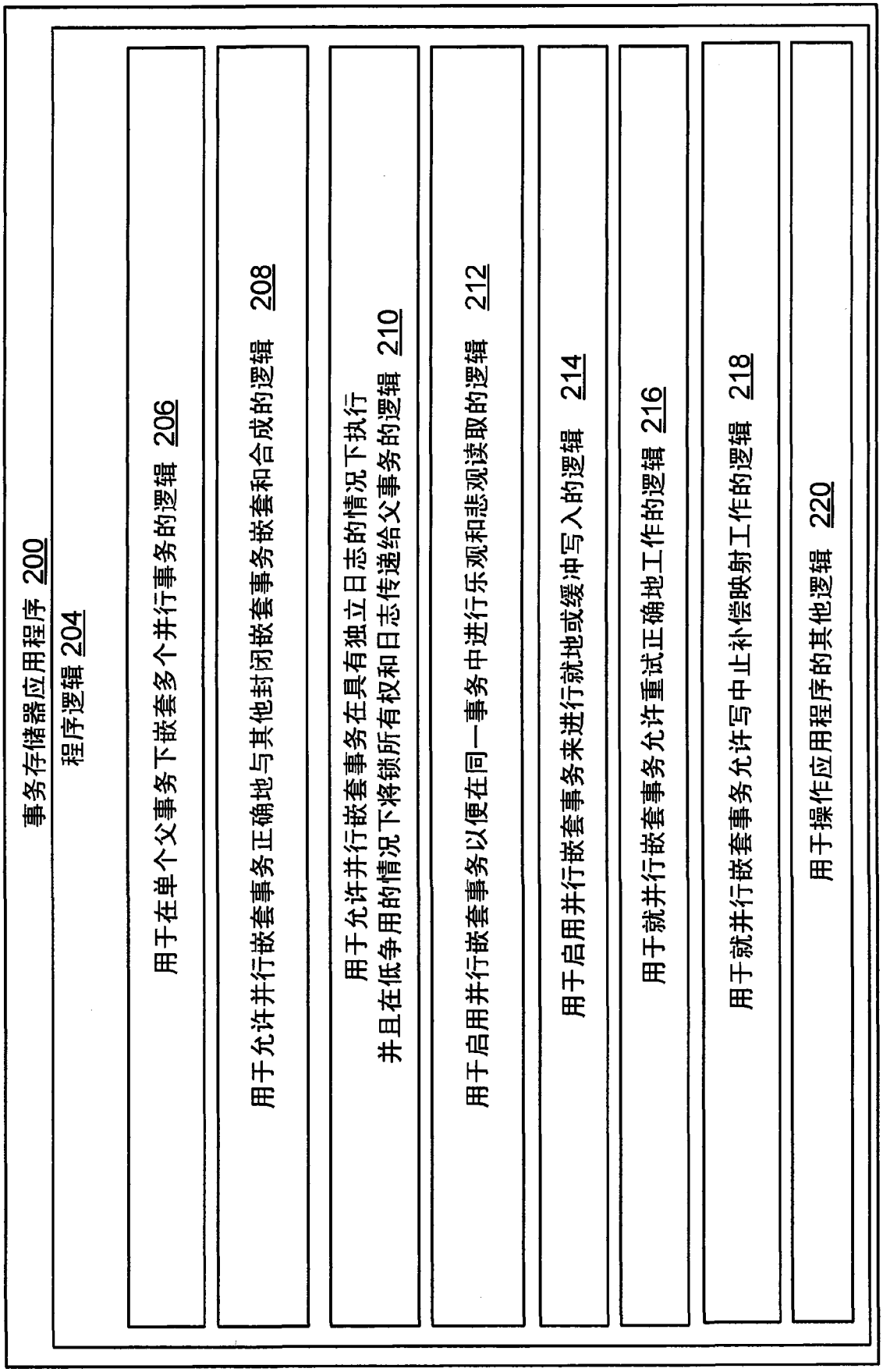

事务存储器应用程序 200
程序逻辑 204
用于在单个父事务下嵌套多个并行事务的逻辑 206
用于允许并行嵌套事务正确地与其他封闭嵌套事务嵌套和合成的逻辑 208
用于允许并行嵌套事务在具有独立日志的情况下执行
并且在低争用的情况下将锁所有权和日志传递给父事务的逻辑 210
用于启用并行嵌套事务以便在同一事务中进行乐观和悲观读取的逻辑 212
用于启用并行嵌套事务来进行就地或缓冲写入的逻辑 214
用于就并行嵌套事务允许重试正确地工作的逻辑 216
用于就并行嵌套事务允许写中止补偿映射工作的逻辑 218
用于操作应用程序的其他逻辑 220

图 2

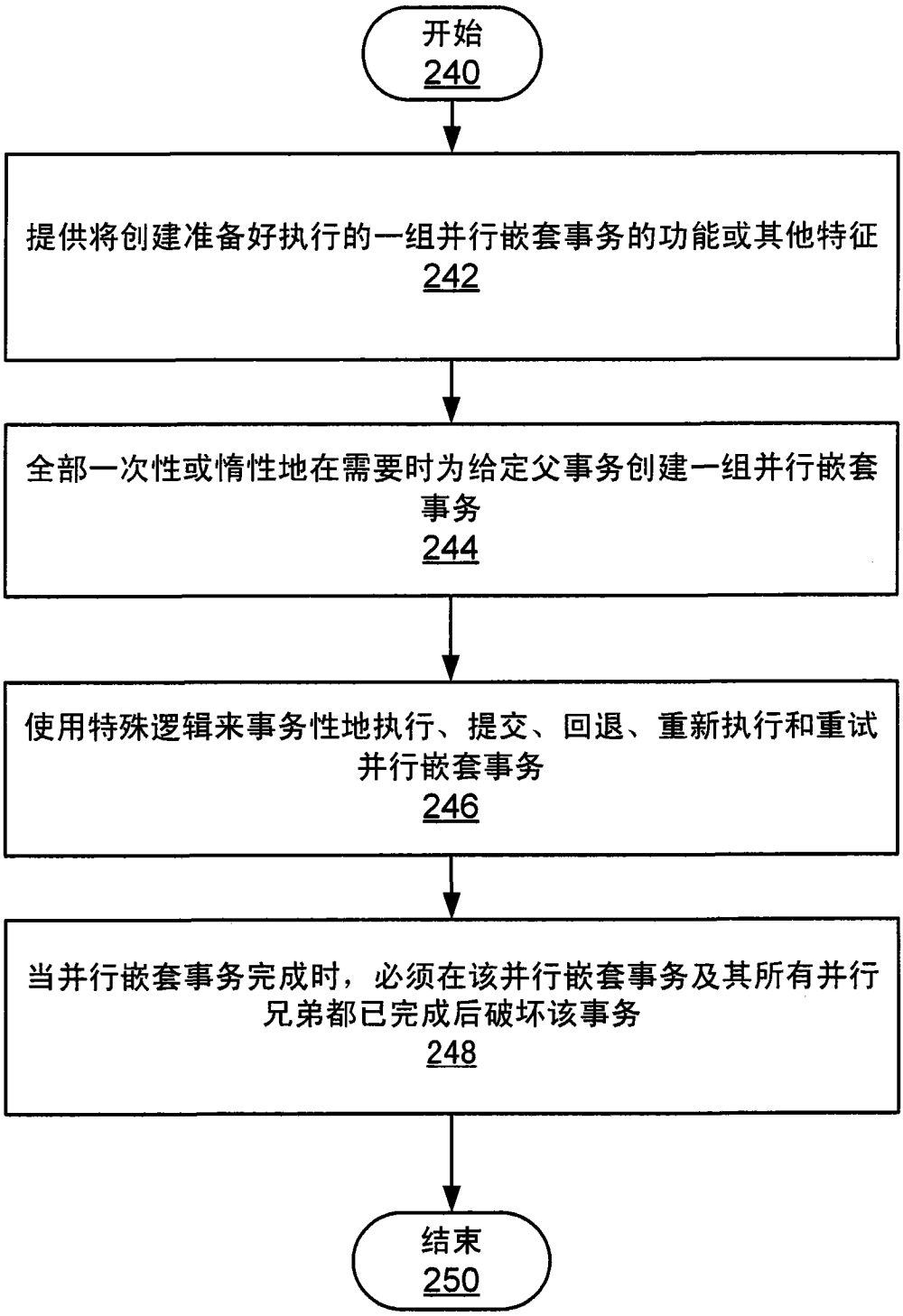

开始
240
提供将创建准备好执行的一组并行嵌套事务的功能或其他特征
242
全部一次性或惰性地在需要时为给定父事务创建一组并行嵌套事务
244
使用特殊逻辑来事务性地执行、提交、回退、重新执行和重试并行嵌套事务
246
当并行嵌套事务完成时，必须在该并行嵌套事务及其所有并行兄弟都已完成后破坏该事务
248
结束
250

图 3

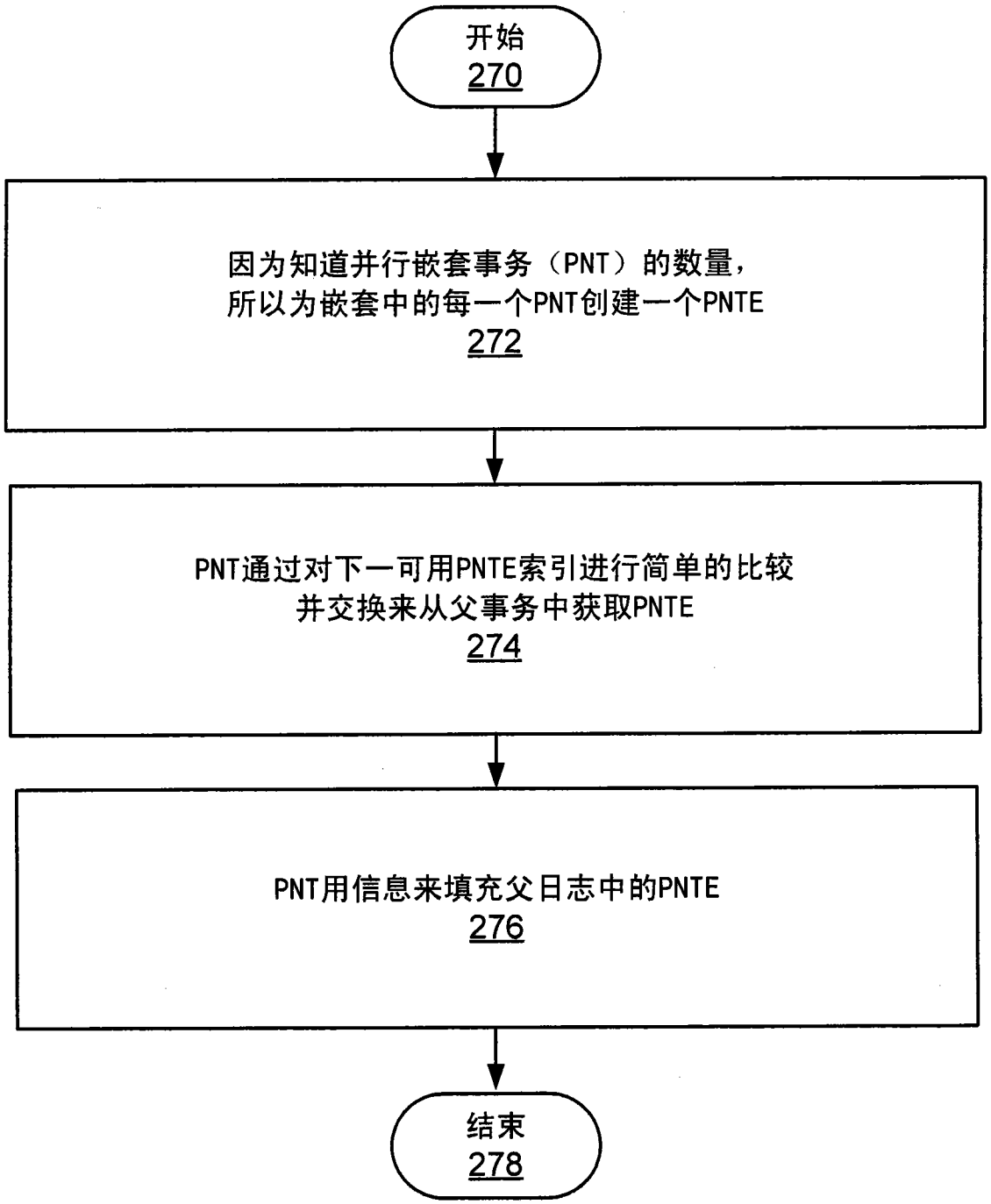

开始
270
因为知道并行嵌套事务（PNT）的数量，
所以为嵌套中的每一个PNT创建一个PNTE
272
PNT通过对下一可用PNTE索引进行简单的比较
并交换来从父事务中获取PNTE
274
PNT用信息来填充父日志中的PNTE
276
结束
278

图 4

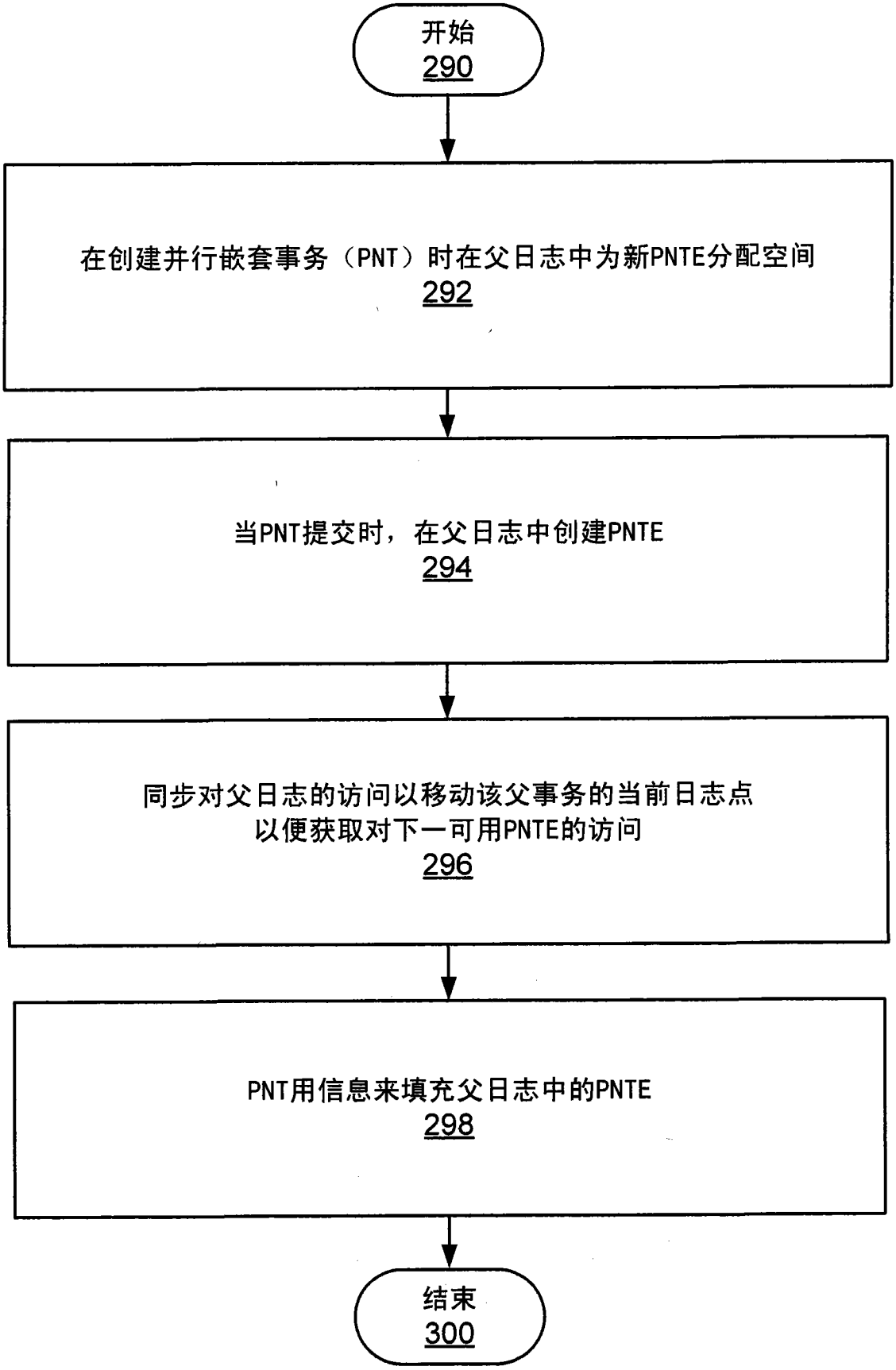

开始
290
在创建并行嵌套事务（PNT）时在父日志中为新PNTE分配空间
292
当PNT提交时，在父日志中创建PNTE
294
同步对父日志的访问以移动该父事务的当前日志点
以便获取对下一可用PNTE的访问
296
PNT用信息来填充父日志中的PNTE
298
结束
300

图 5

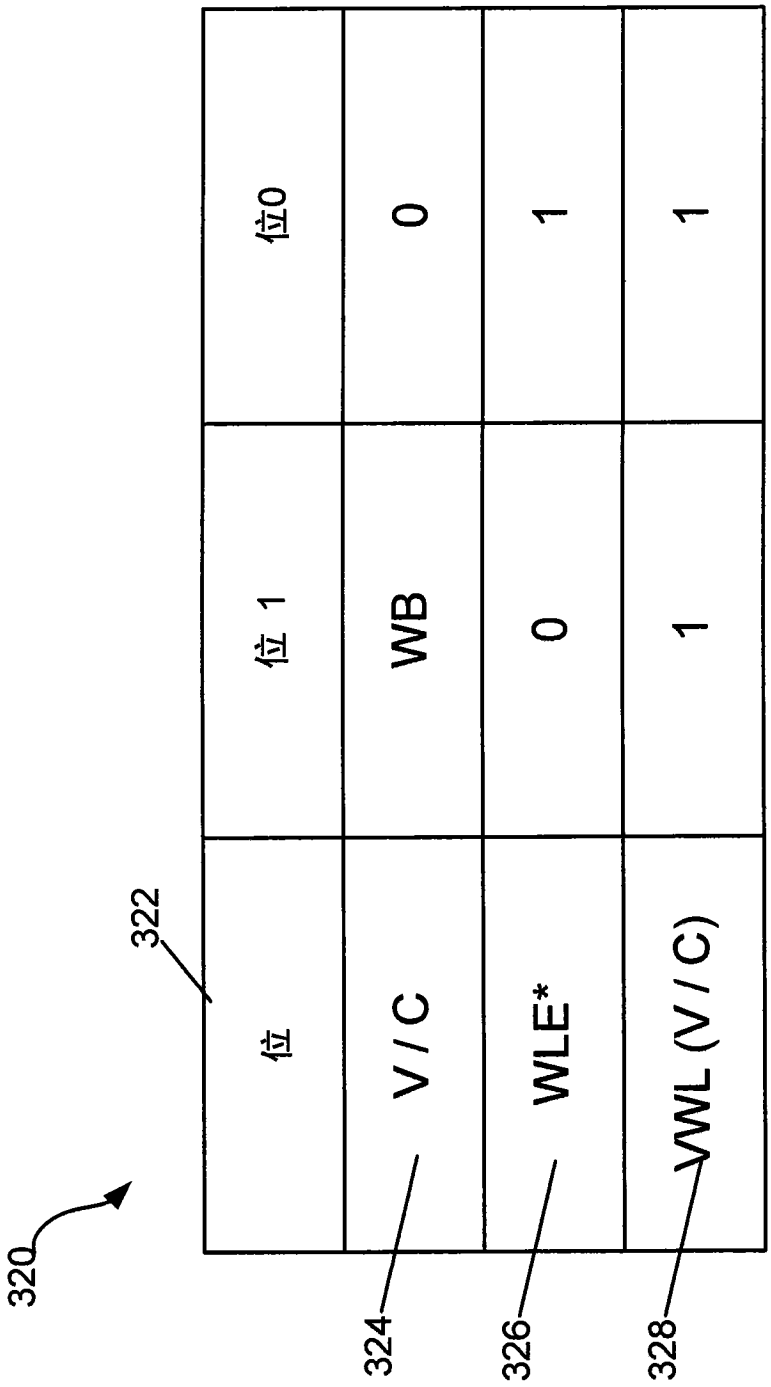


图 6

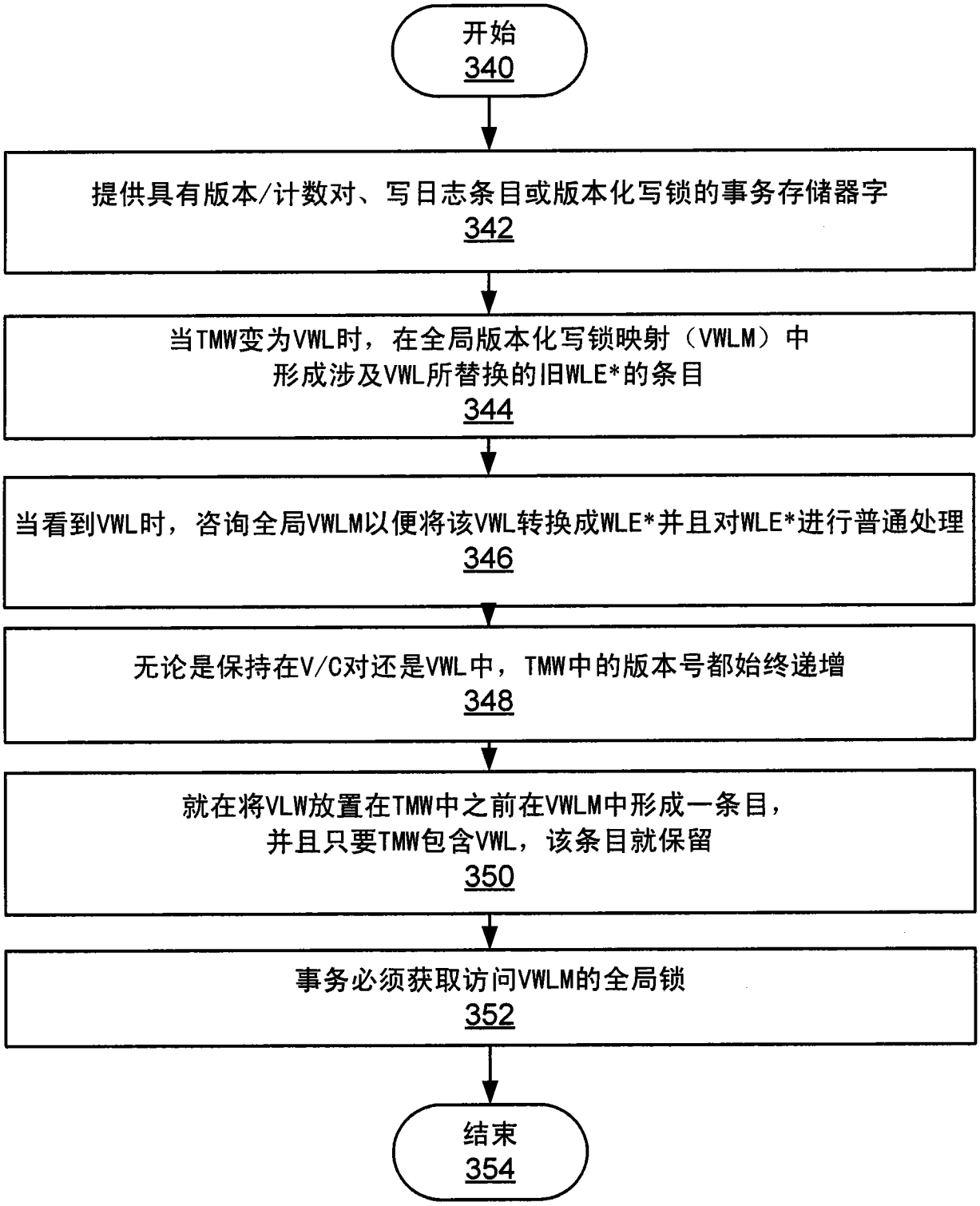

开始
340
提供具有版本/计数对、写日志条目或版本化写锁的事务存储器字
342
当TMW变为VWL时，在全局版本化写锁映射（VWLM）中
形成涉及VWL所替换的旧WLE*的条目
344
当看到VWL时，咨询全局VWLM以便将该VWL转换成WLE*并且对WLE*进行普通处理
346
无论是保持在V/C对还是VWL中，TMW中的版本号都始终递增
348
就在将VLW放置在TMW中之前在VWLM中形成一条目，
并且只要TMW包含VWL，该条目就保留
350
事务必须获取访问VWLM的全局锁
352
结束
354

图 7

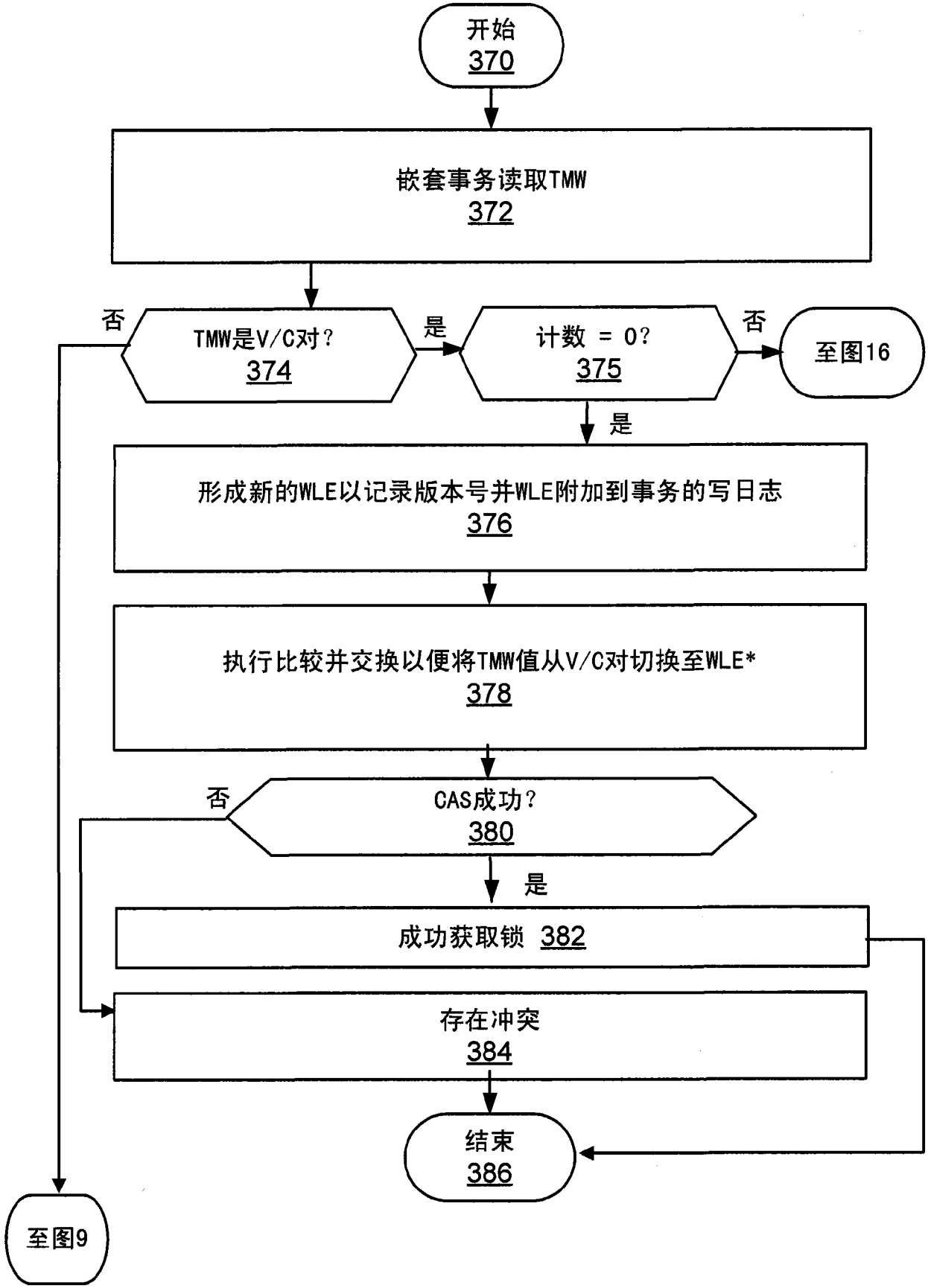

开始
370
嵌套事务读取TMW
372
否
TMW是V/C对？
374
是
计数 = 0？
375
否
至图16
是
形成新的WLE以记录版本号并WLE附加到事务的写日志
376
执行比较并交换以便将TMW值从V/C对切换至WLE*
378
否
CAS成功？
380
是
成功获取锁 382
存在冲突
384
结束
386
至图9

图 8

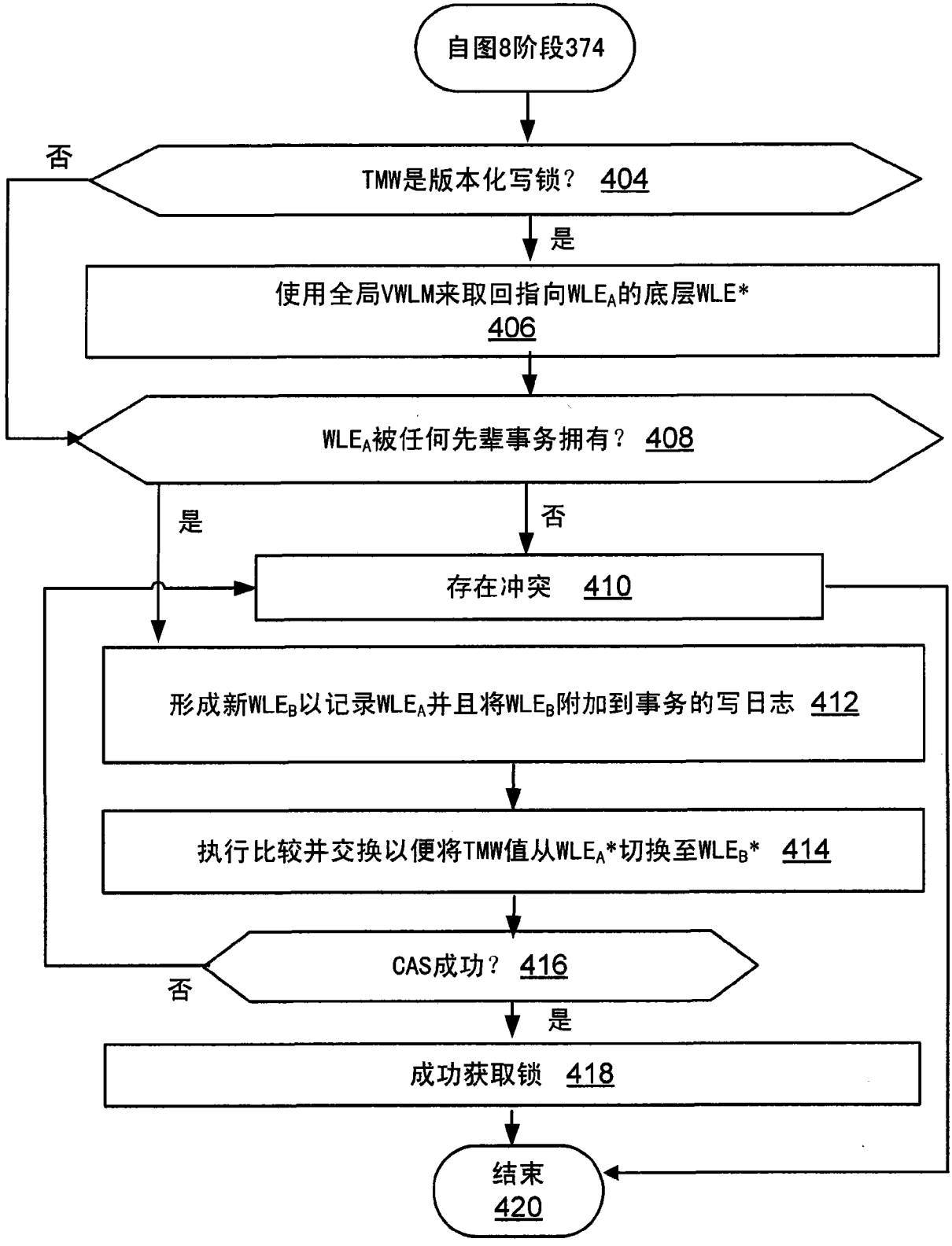

自图8阶段374
否
TMW是版本化写锁？ 404
是
使用全局VWLM来取回指向WLE$_A$的底层WLE* 406
WLE$_A$被任何先辈事务拥有？ 408
是
否
存在冲突 410
形成新WLE$_B$以记录WLE$_A$并且将WLE$_B$附加到事务的写日志 412
执行比较并交换以便将TMW值从WLE$_A$*切换至WLE$_B$* 414
CAS成功？ 416
否
是
成功获取锁 418
结束 420

图 9

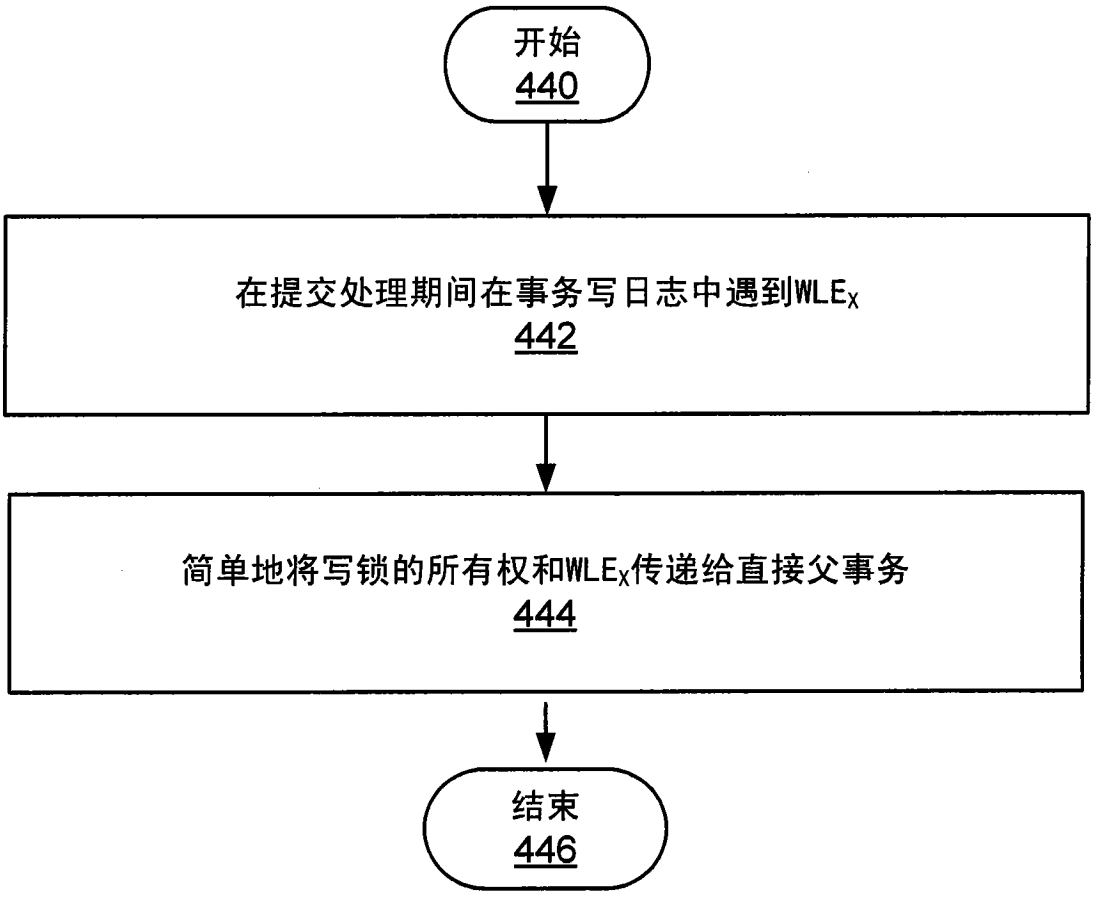

开始
440
在提交处理期间在事务写日志中遇到WLE_X
442
简单地将写锁的所有权和WLE_X传递给直接父事务
444
结束
446

图 10

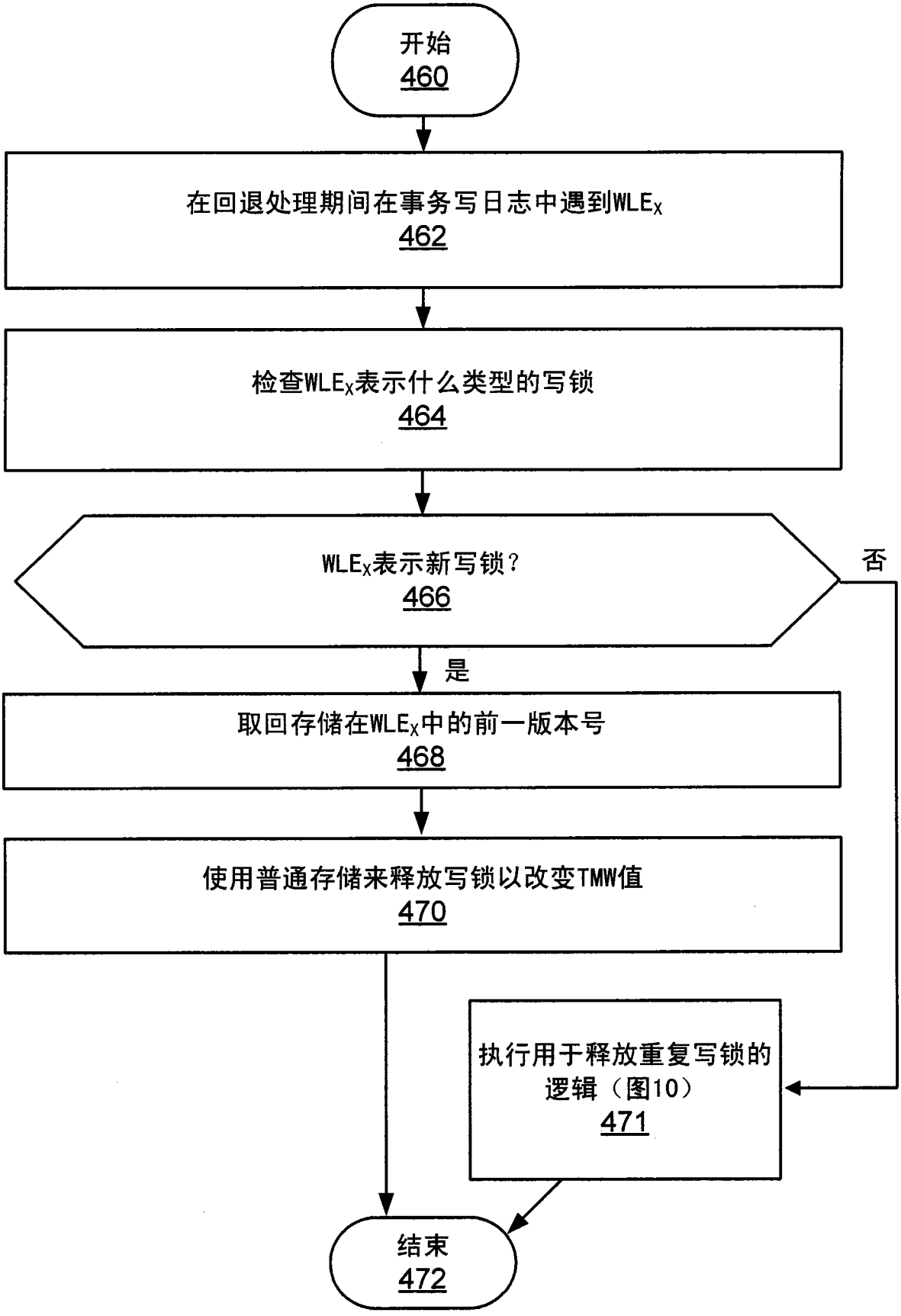

开始
460
在回退处理期间在事务写日志中遇到WLEX
462
检查WLEX表示什么类型的写锁
464
WLEX表示新写锁？
466
否
是
取回存储在WLEX中的前一版本号
468
使用普通存储来释放写锁以改变TMW值
470
执行用于释放重复写锁的逻辑（图10）
471
结束
472

图 11

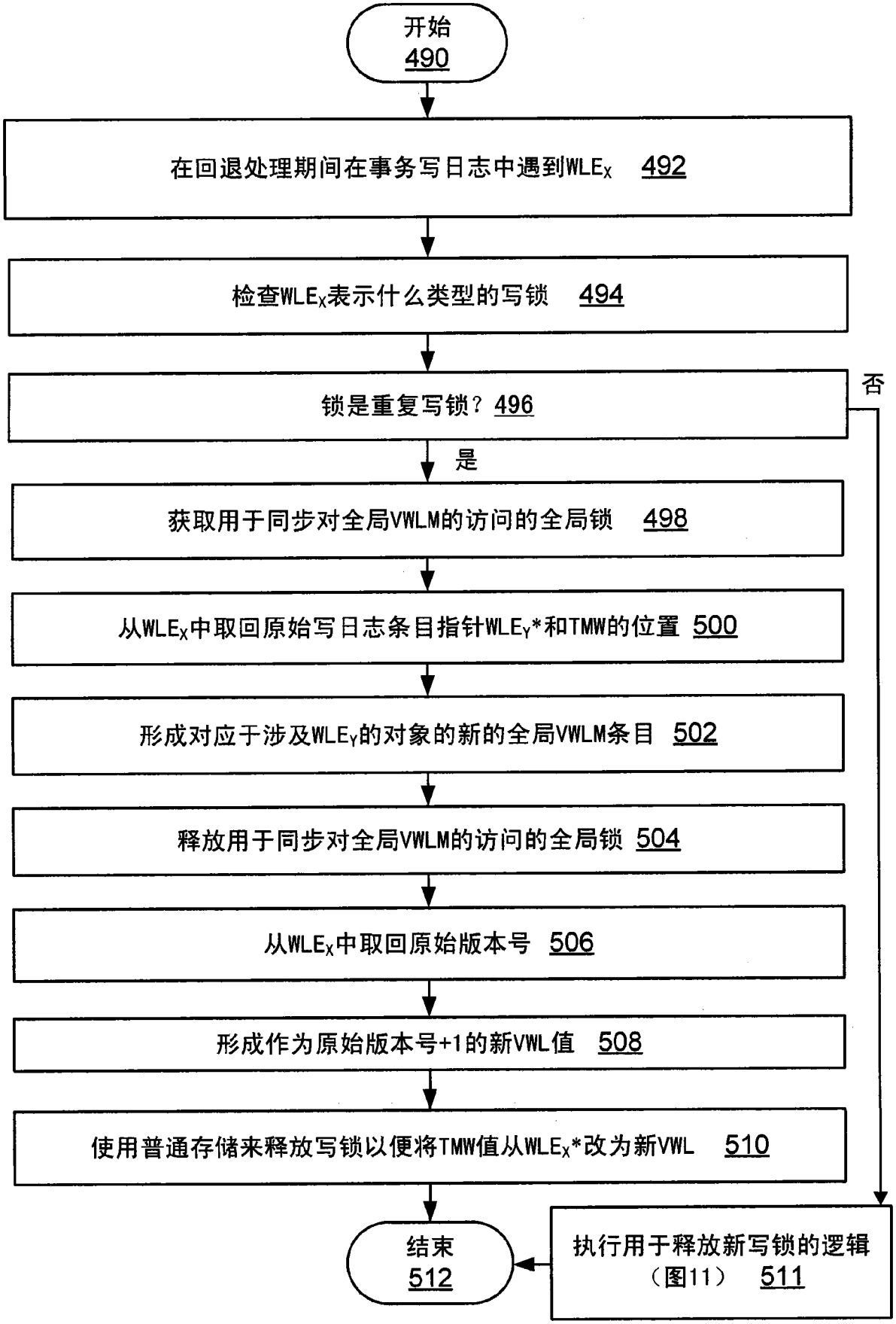

开始
490
在回退处理期间在事务写日志中遇到WLEX 492
检查WLEX表示什么类型的写锁 494
锁是重复写锁？496
否
是
获取用于同步对全局VWLM的访问的全局锁 498
从WLEX中取回原始写日志条目指针WLEY*和TMW的位置 500
形成对应于涉及WLEY的对象的新的全局VWLM条目 502
释放用于同步对全局VWLM的访问的全局锁 504
从WLEX中取回原始版本号 506
形成作为原始版本号+1的新VWL值 508
使用普通存储来释放写锁以便将TMW值从WLEX*改为新VWL 510
结束
512
执行用于释放新写锁的逻辑
（图11） 511

图 12

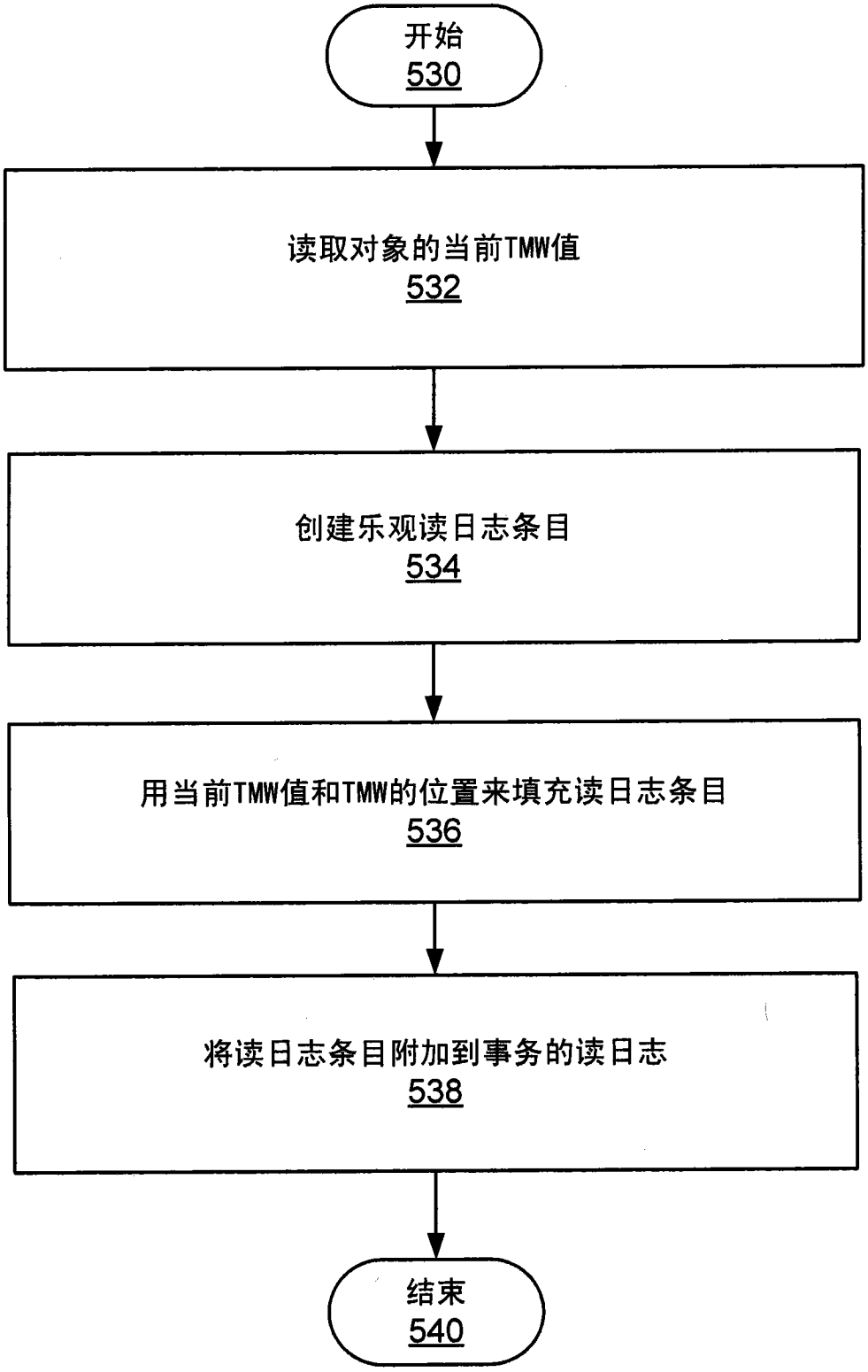

开始
530
读取对象的当前TMW值
532
创建乐观读日志条目
534
用当前TMW值和TMW的位置来填充读日志条目
536
将读日志条目附加到事务的读日志
538
结束
540

图 13

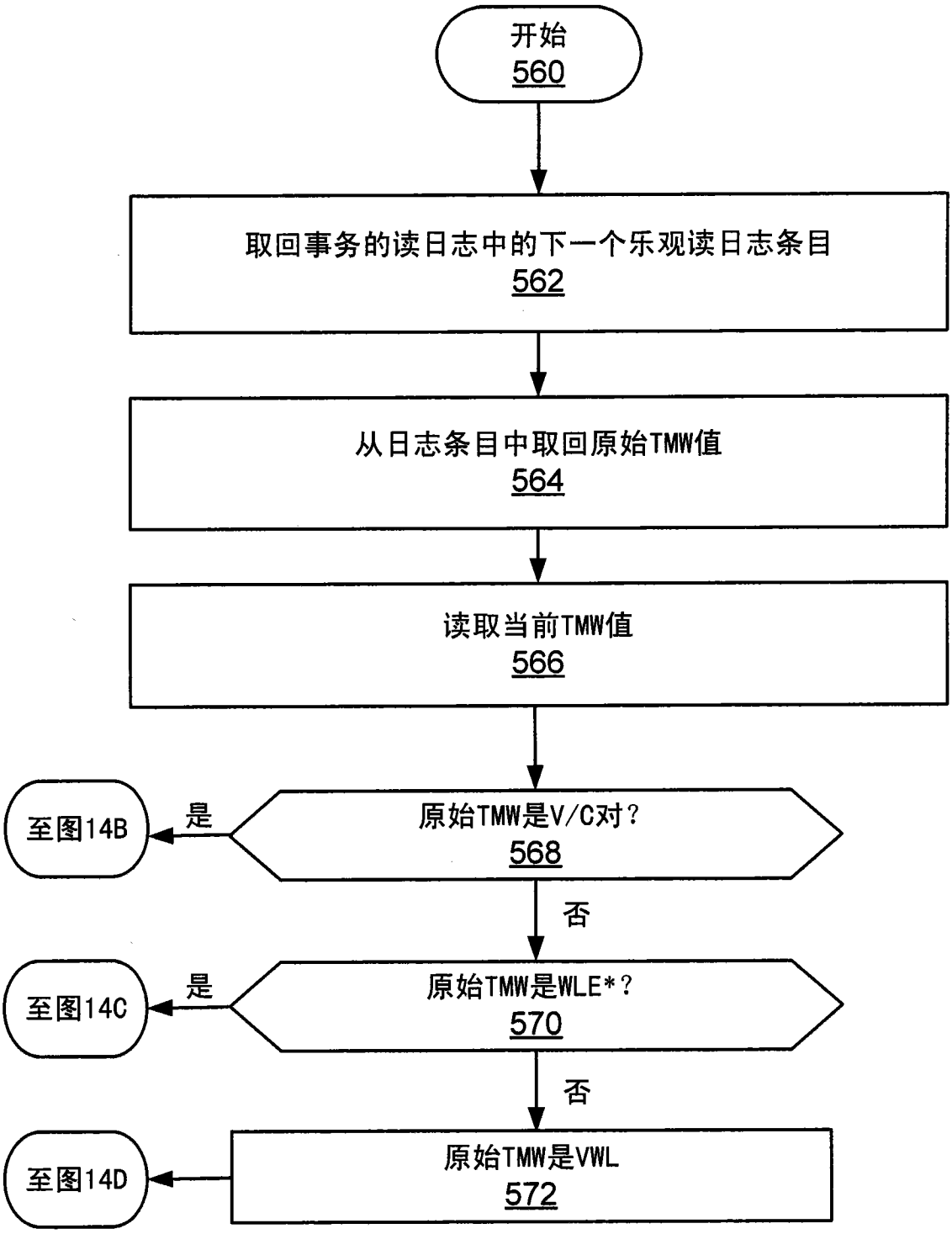

开始
560
取回事务的读日志中的下一个乐观读日志条目
562
从日志条目中取回原始TMW值
564
读取当前TMW值
566
原始TMW是V/C对？
568
是
至图14B
否
原始TMW是WLE*？
570
是
至图14C
否
原始TMW是VWL
572
至图14D

图 14A

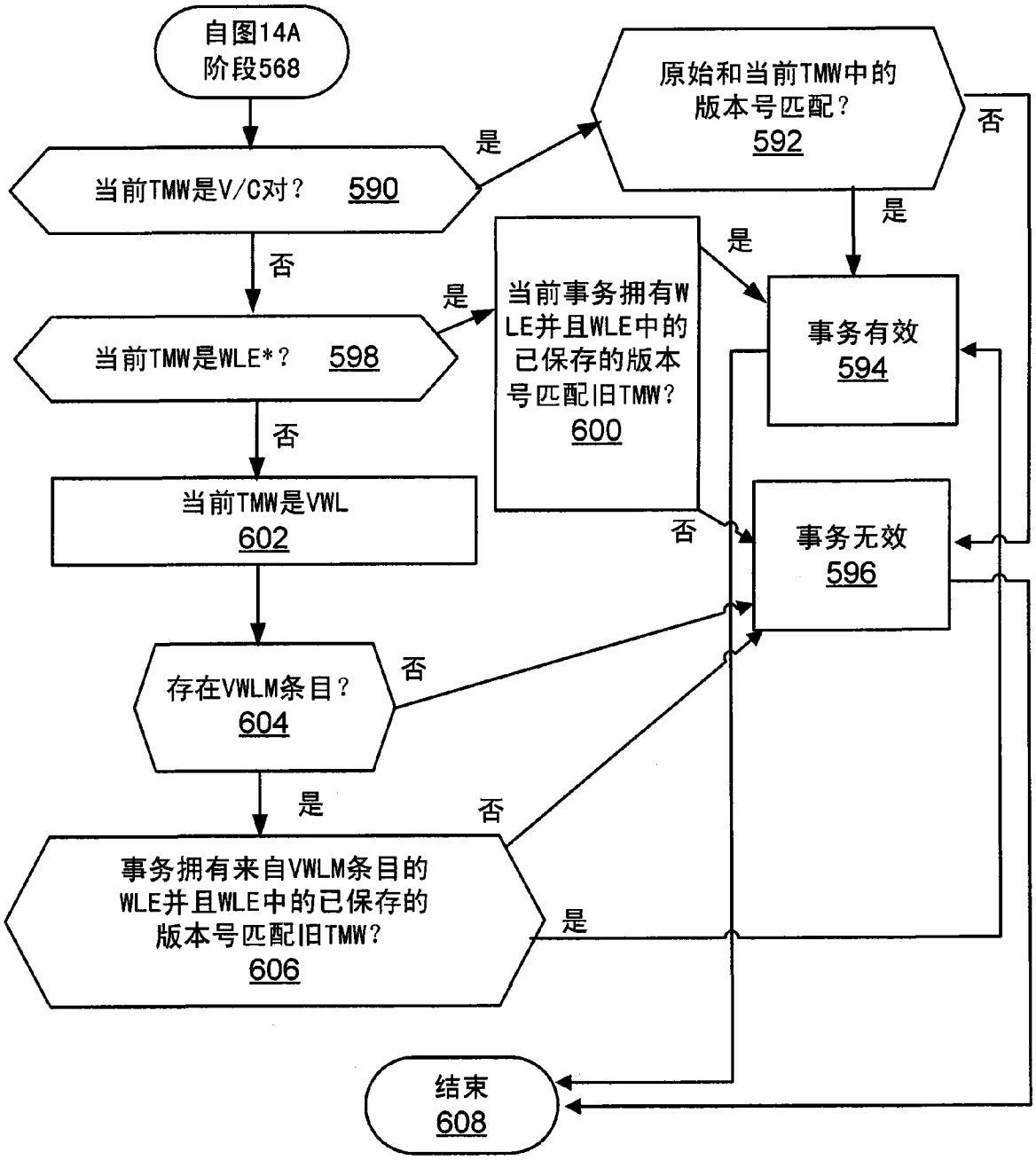

自图14A
阶段568
当前TMW是V/C对？ 590
是
原始和当前TMW中的
版本号匹配？
592
否
是
否
当前TMW是WLE*？ 598
是
当前事务拥有W
LE并且WLE中的
已保存的版本
号匹配旧TMW？
600
是
事务有效
594
否
当前TMW是VWL
602
事务无效
596
存在VWLM条目？
604
否
是
否
事务拥有来自VWLM条目的
WLE并且WLE中的已保存的
版本号匹配旧TMW？
606
是
结束
608

图 14B

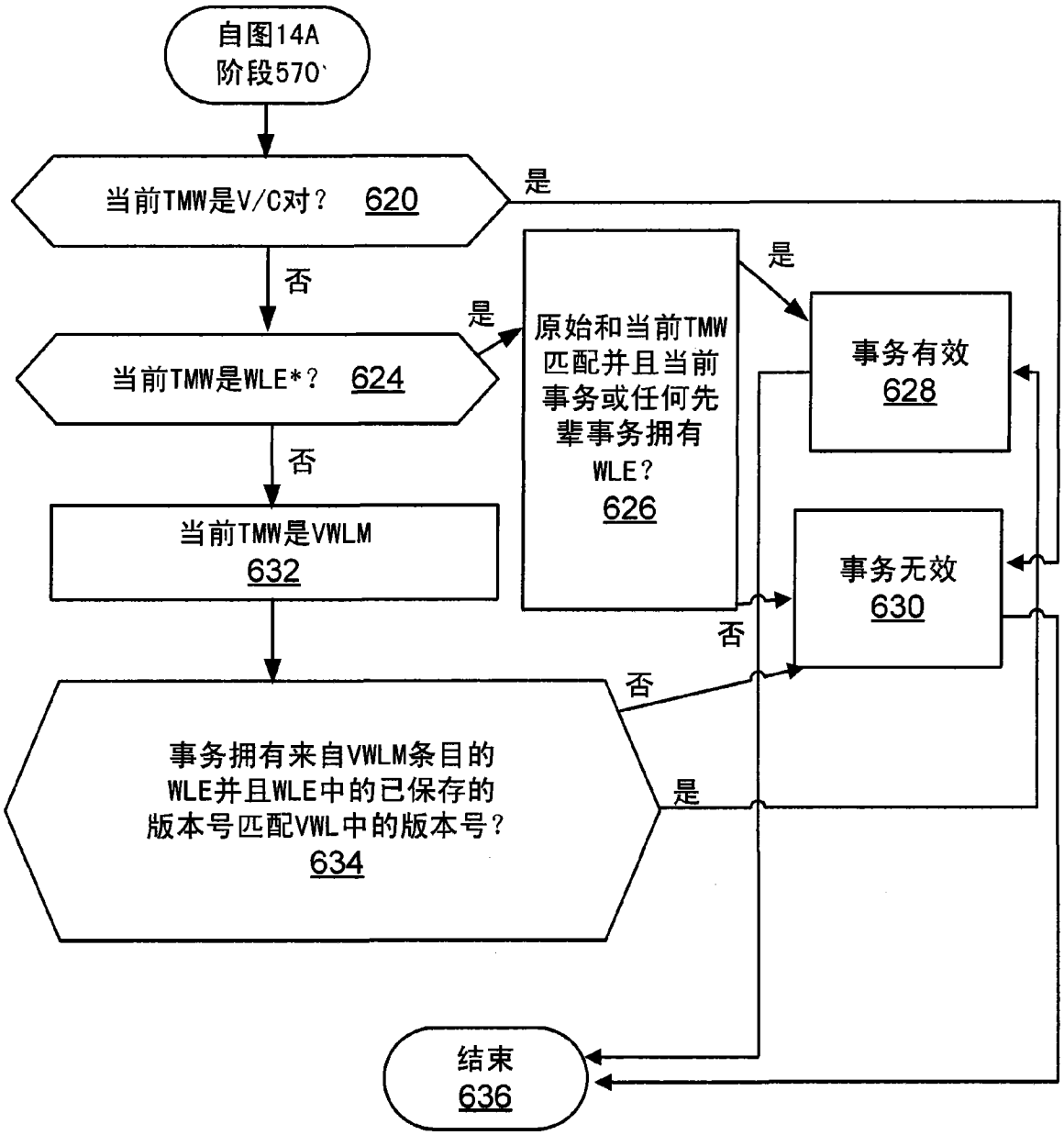

自图14A
阶段570'
当前TMW是V/C对？ 620
是
否
当前TMW是WLE*？ 624
是
否
当前TMW是VWLM
632
原始和当前TMW匹配并且当前事务或任何先辈事务拥有WLE？
626
是
事务有效
628
事务无效
630
否
否
事务拥有来自VWLM条目的WLE并且WLE中的已保存的版本号匹配VWL中的版本号？
634
是
结束
636

图 14C

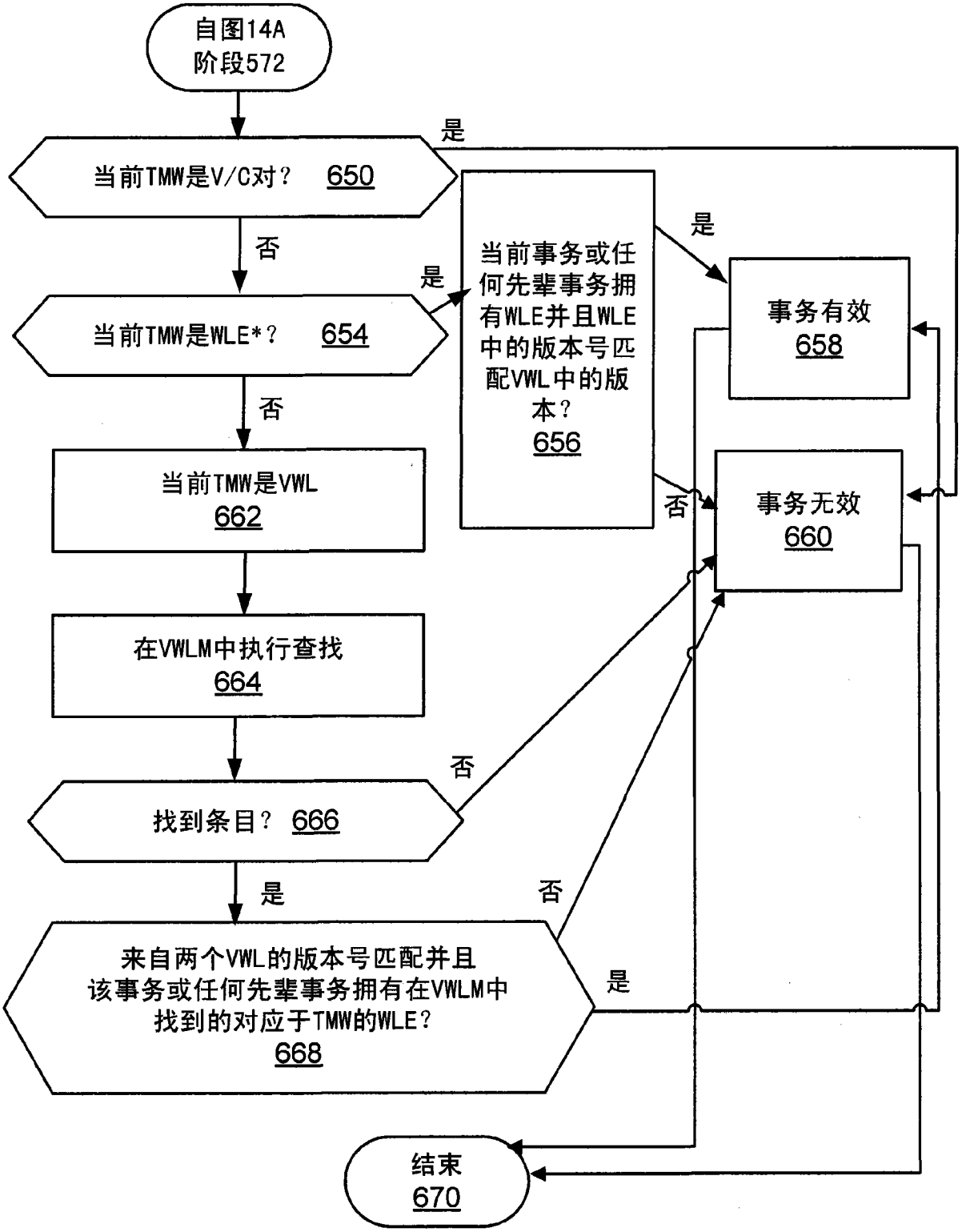

自图14A
阶段572
当前TMW是V/C对？ 650
是
否
当前TMW是WLE*？ 654
是
否
当前TMW是VWL
662
在VWLM中执行查找
664
找到条目？ 666
否
是
来自两个VWL的版本号匹配并且
该事务或任何先辈事务拥有在VWLM中
找到的对应于TMW的WLE？
668
否
是
当前事务或任何先辈事务拥有WLE并且WLE中的版本号匹配VWL中的版本？
656
是
否
事务有效
658
事务无效
660
结束
670

图 14D

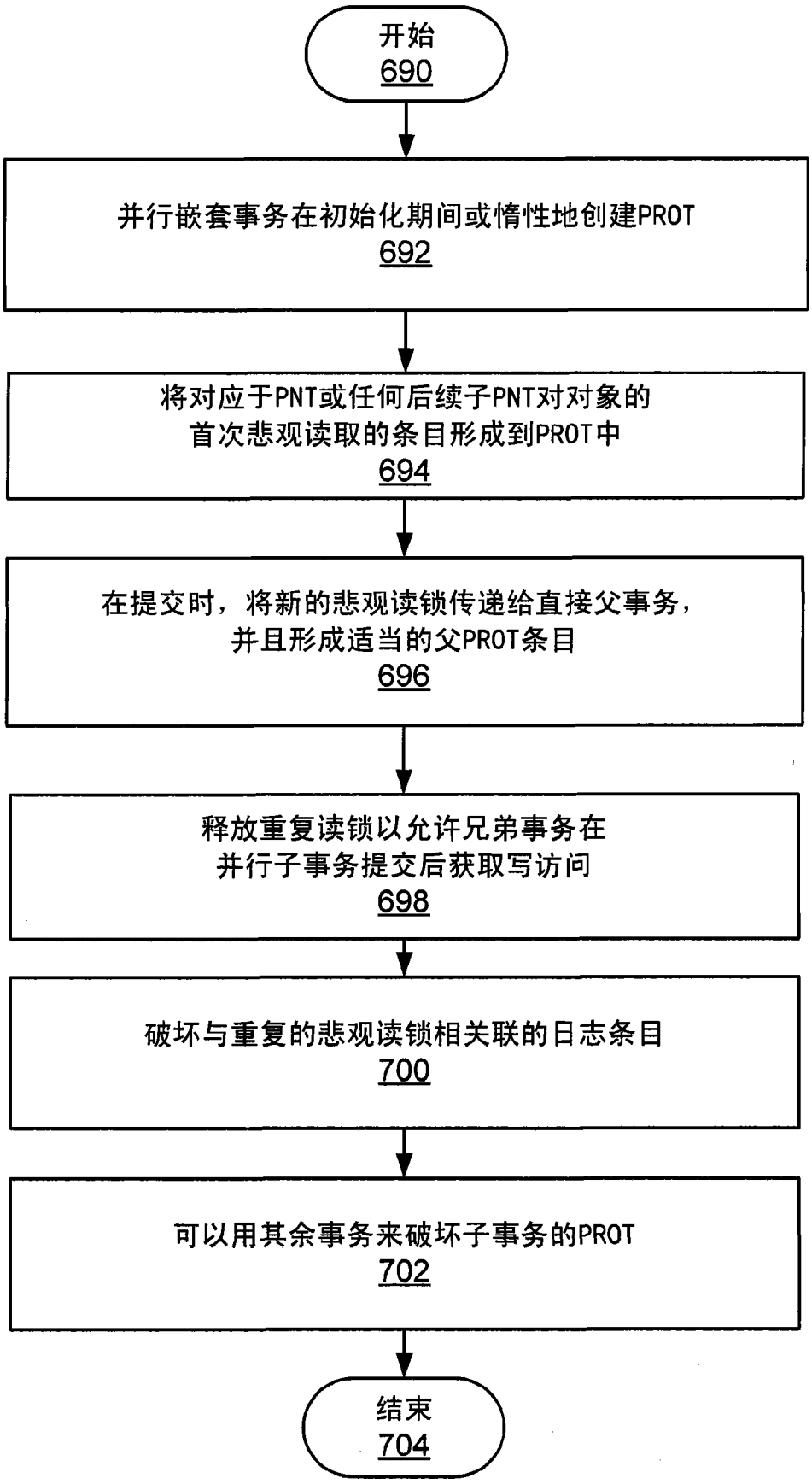

开始
690
并行嵌套事务在初始化期间或惰性地创建PROT
692
将对应于PNT或任何后续子PNT对对象的
首次悲观读取的条目形成到PROT中
694
在提交时，将新的悲观读锁传递给直接父事务，
并且形成适当的父PROT条目
696
释放重复读锁以允许兄弟事务在
并行子事务提交后获取写访问
698
破坏与重复的悲观读锁相关联的日志条目
700
可以用其余事务来破坏子事务的PROT
702
结束
704

图 15

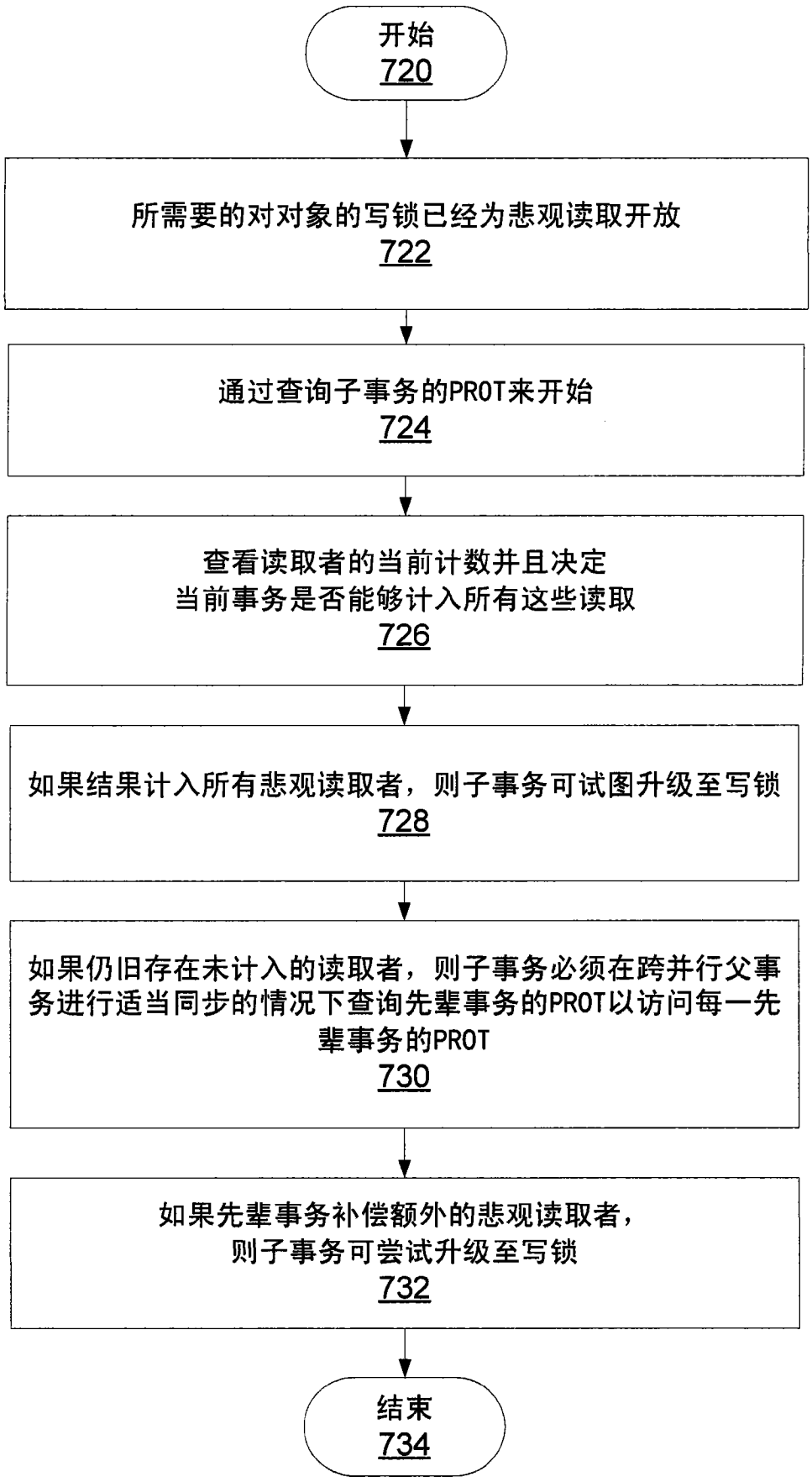

开始
720
所需要的对对象的写锁已经为悲观读取开放
722
通过查询子事务的PROT来开始
724
查看读取者的当前计数并且决定
当前事务是否能够计入所有这些读取
726
如果结果计入所有悲观读取者，则子事务可试图升级至写锁
728
如果仍旧存在未计入的读取者，则子事务必须在跨并行父事务进行适当同步的情况下查询先辈事务的PROT以访问每一先辈事务的PROT
730
如果先辈事务补偿额外的悲观读取者，
则子事务可尝试升级至写锁
732
结束
734

图 16

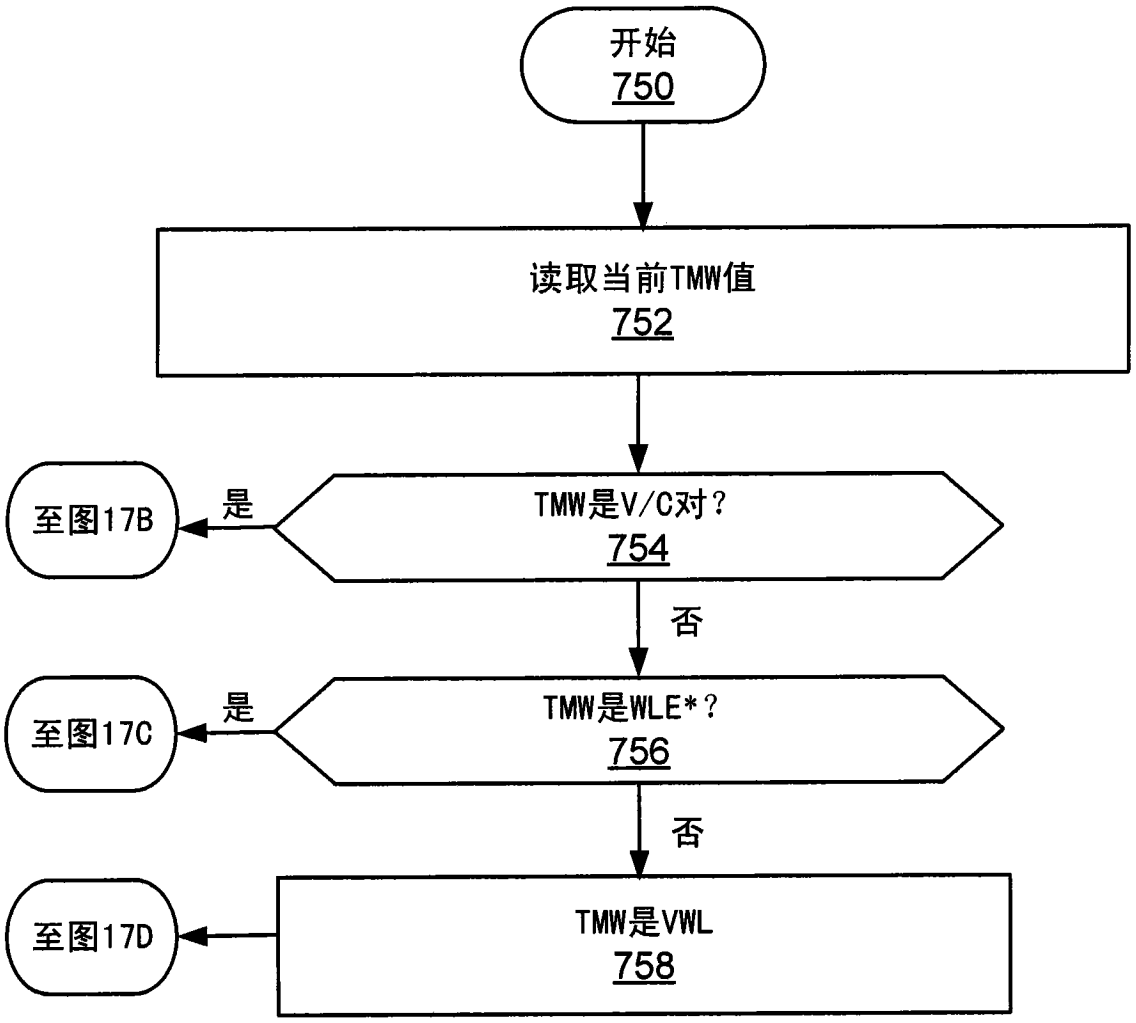

开始
750
读取当前TMW值
752
TMW是V/C对？
754
是
至图17B
否
TMW是WLE*？
756
是
至图17C
否
TMW是VWL
758
至图17D

图 17A

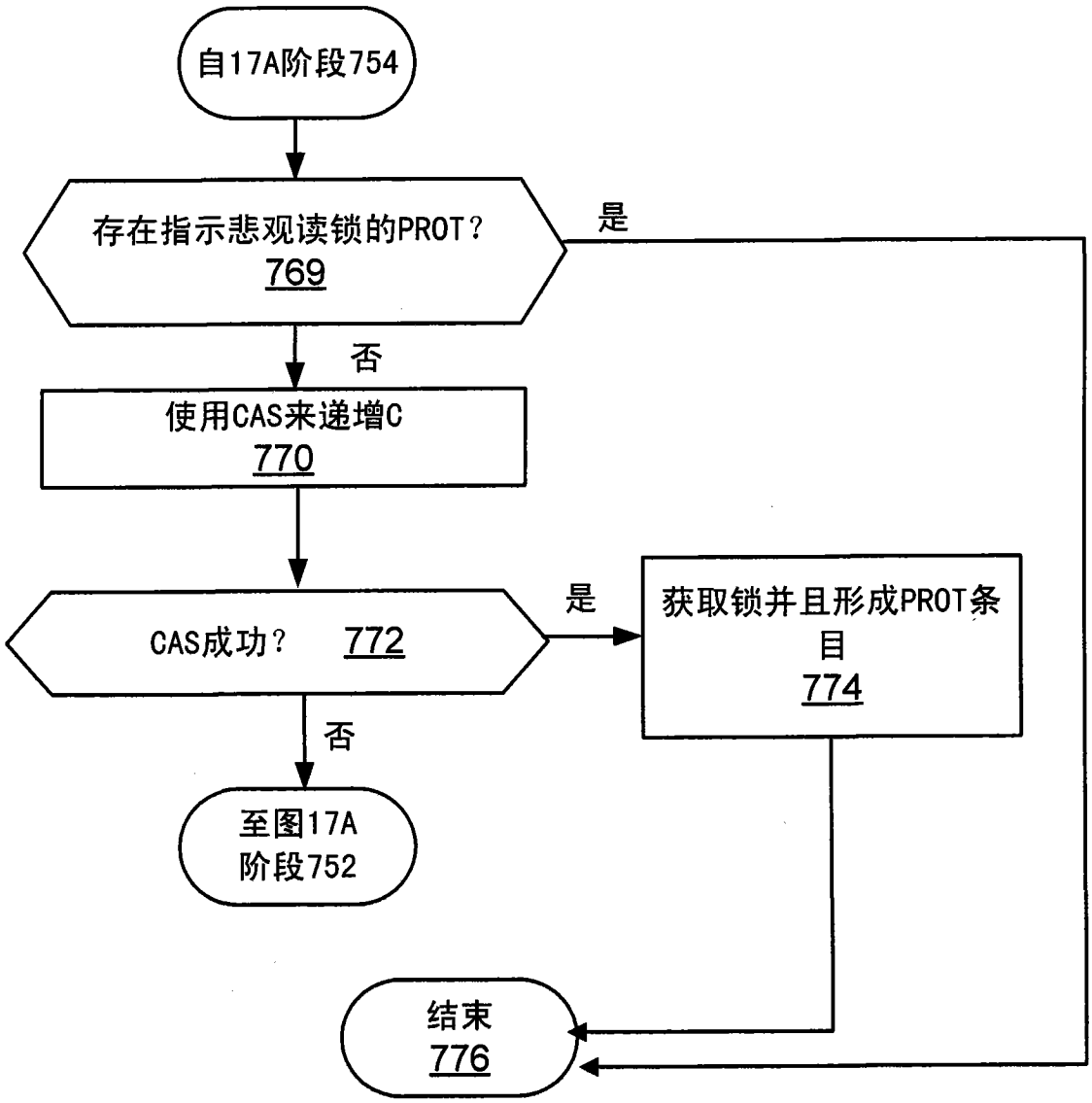

自17A阶段754
存在指示悲观读锁的PROT？
769
是
否
使用CAS来递增C
770
CAS成功？ 772
是
获取锁并且形成PROT条目
774
否
至图17A
阶段752
结束
776

图 17B

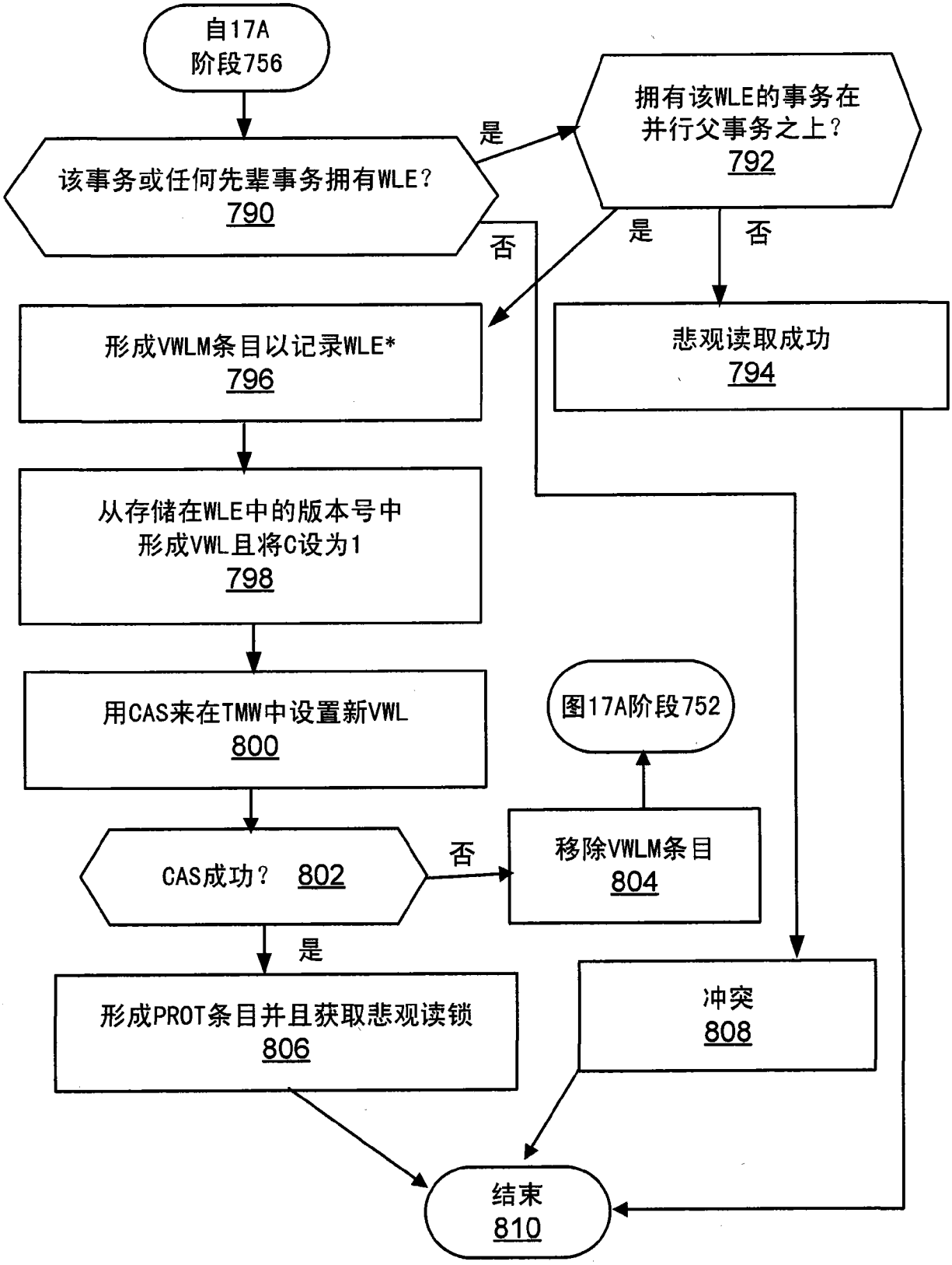

自17A
阶段756
该事务或任何先辈事务拥有WLE?
790
是
拥有该WLE的事务在
并行父事务之上?
792
是
否
否
形成VWLM条目以记录WLE*
796
悲观读取成功
794
从存储在WLE中的版本号中
形成VWL且将C设为1
798
用CAS来在TMW中设置新VWL
800
图17A阶段752
CAS成功? 802
否
移除VWLM条目
804
是
形成PROT条目并且获取悲观读锁
806
冲突
808
结束
810

图 17C

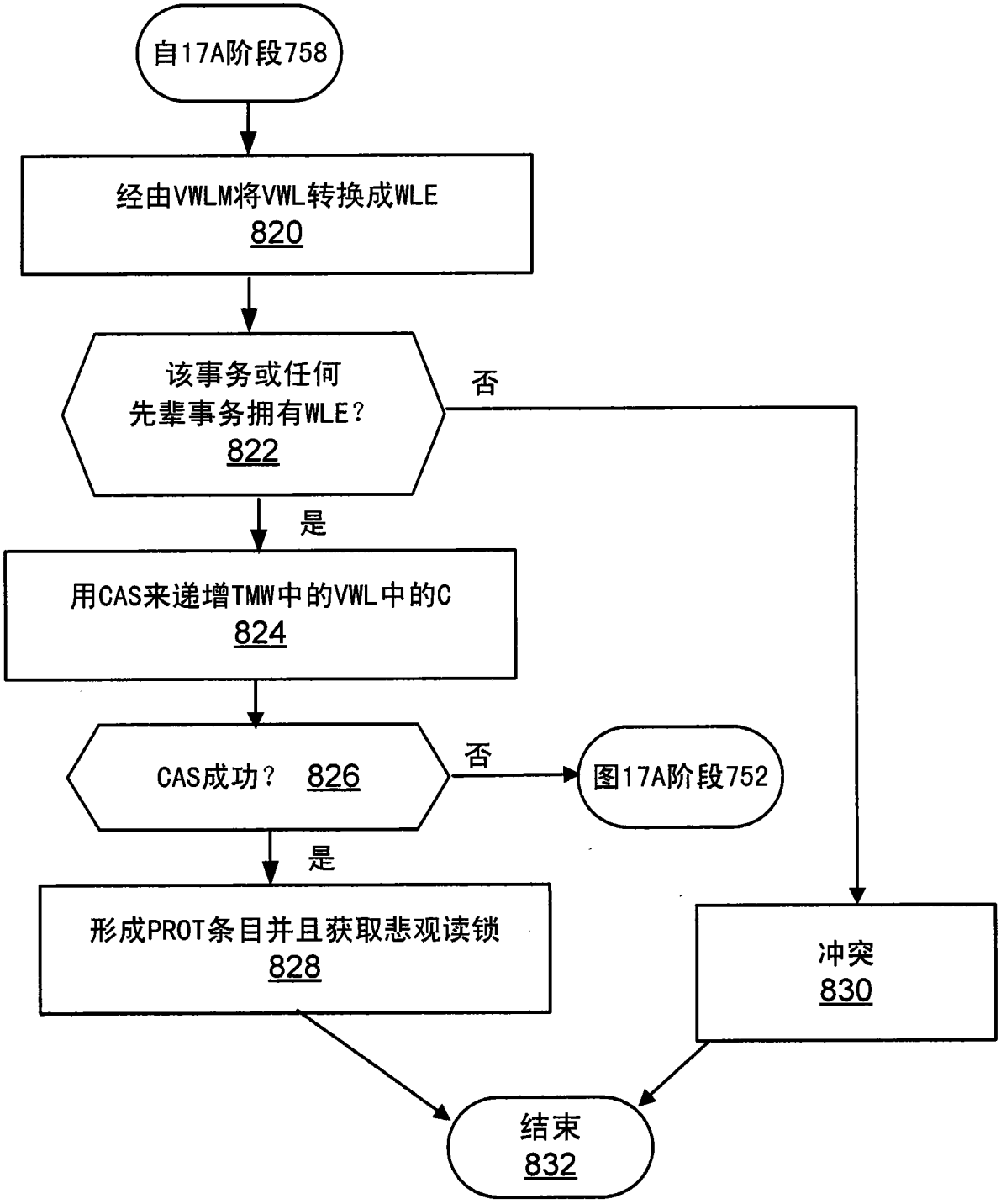

自17A阶段758
经由VWLM将VWL转换成WLE
820
该事务或任何
先辈事务拥有WLE?
822
否
是
用CAS来递增TMW中的VWL中的C
824
CAS成功? 826
否
图17A阶段752
是
形成PROT条目并且获取悲观读锁
828
冲突
830
结束
832

图 17D

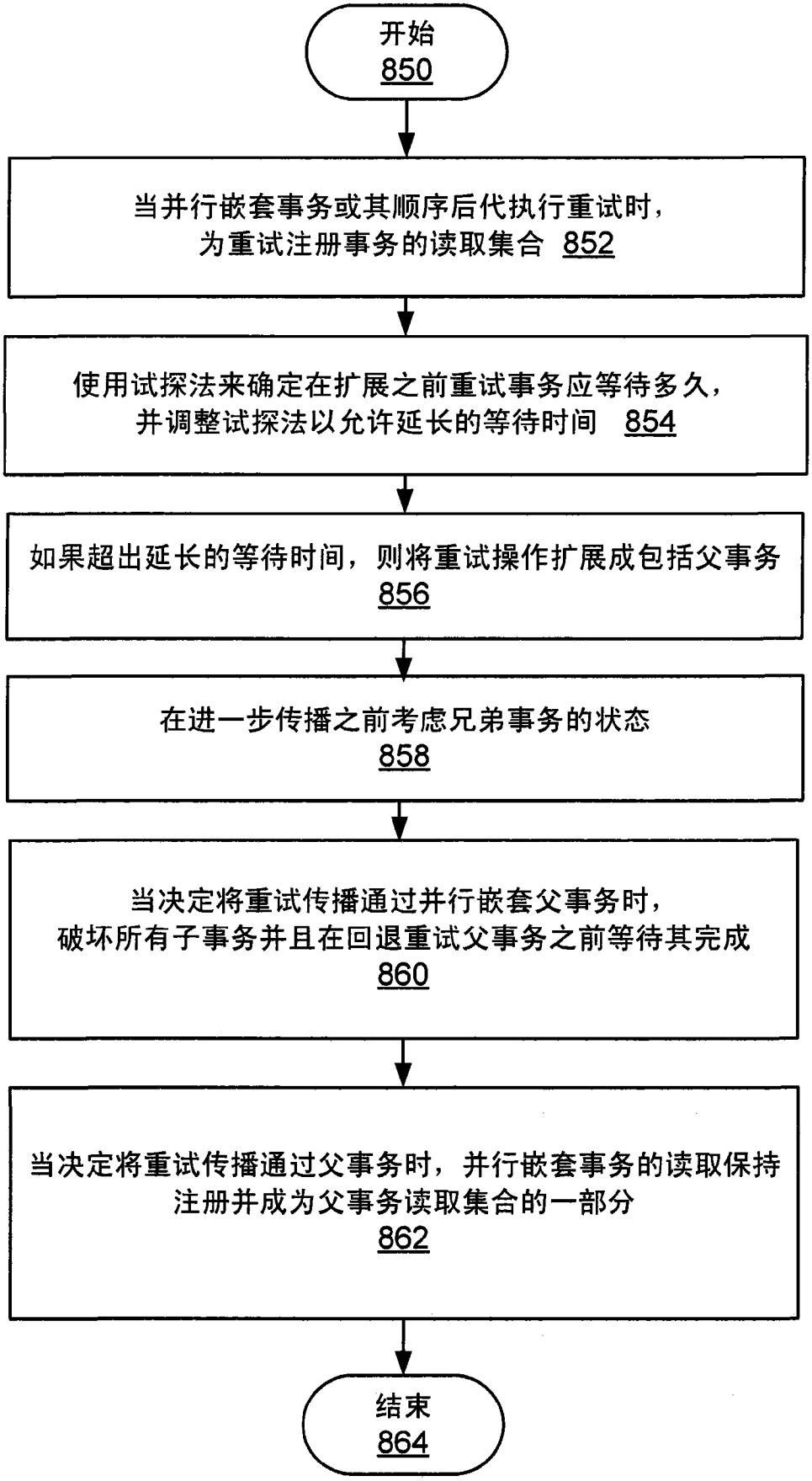

开始
850
当并行嵌套事务或其顺序后代执行重试时，
为重试注册事务的读取集合 852
使用试探法来确定在扩展之前重试事务应等待多久，
并调整试探法以允许延长的等待时间 854
如果超出延长的等待时间，则将重试操作扩展成包括父事务
856
在进一步传播之前考虑兄弟事务的状态
858
当决定将重试传播通过并行嵌套父事务时，
破坏所有子事务并且在回退重试父事务之前等待其完成
860
当决定将重试传播通过父事务时，并行嵌套事务的读取保持
注册并成为父事务读取集合的一部分
862
结束
864

图 18

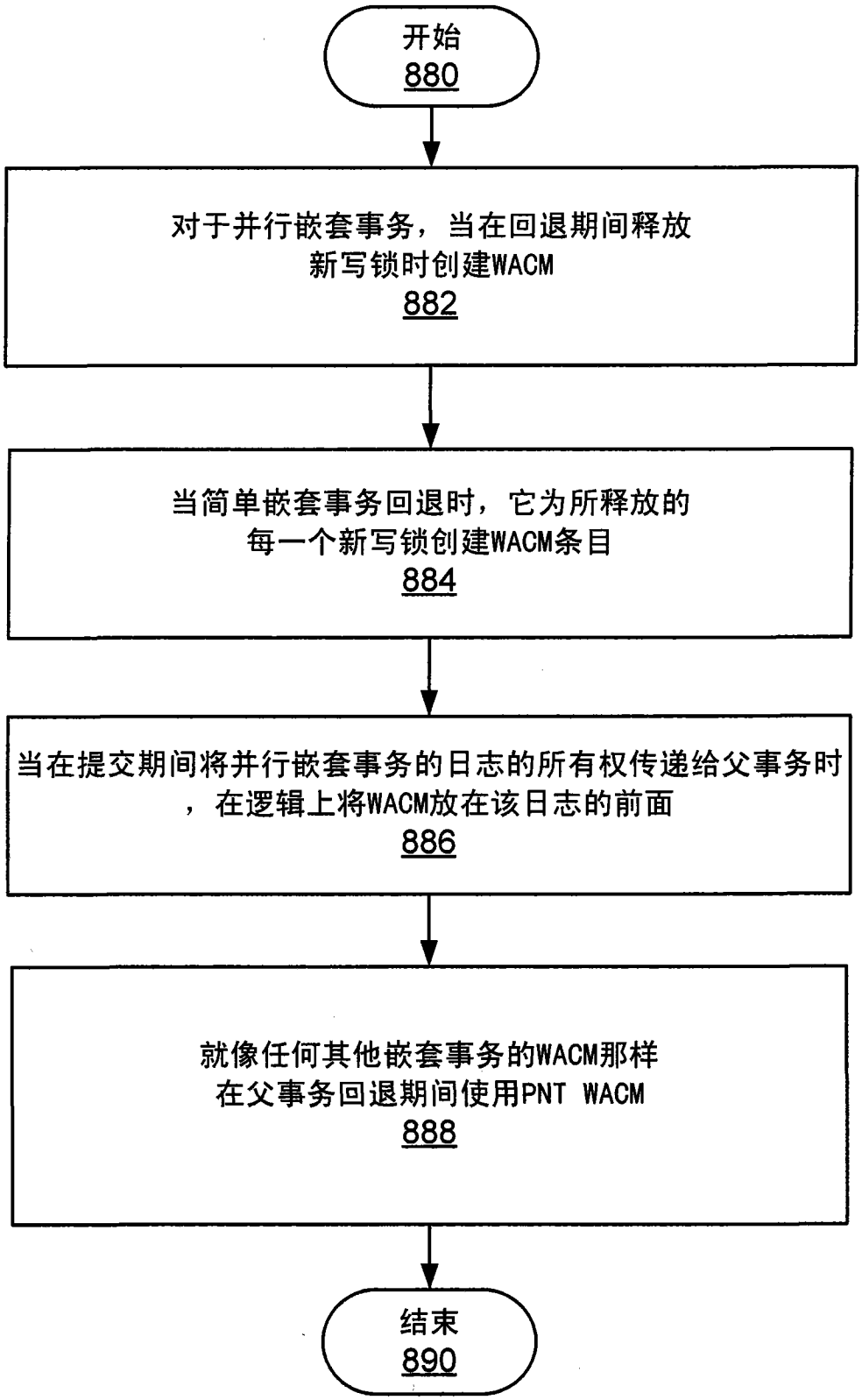

开始
880
对于并行嵌套事务，当在回退期间释放
新写锁时创建WACM
882
当简单嵌套事务回退时，它为所释放的
每一个新写锁创建WACM条目
884
当在提交期间将并行嵌套事务的日志的所有权传递给父事务时
，在逻辑上将WACM放在该日志的前面
886
就像任何其他嵌套事务的WACM那样
在父事务回退期间使用PNT WACM
888
结束
890

图 19