[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 12/00 (2006.01)
G06F 17/30 (2006.01)
G06F 11/14 (2006.01)

[12] 发明专利说明书

专利号 ZL 02806161.6

[45] 授权公告日 2007 年 5 月 9 日

[11] 授权公告号 CN 1315055C

[22] 申请日 2002.3.7 [21] 申请号 02806161.6

[30] 优先权

[32] 2001. 3. 7 [33] US [31] 60/274,270

[32] 2002. 3. 4 [33] US [31] 10/092,247

[86] 国际申请 PCT/US2002/006981 2002.3.7

[87] 国际公布 WO2002/073416 英 2002.9.19

[85] 进入国家阶段日期 2003.9.8

[73] 专利权人 甲骨文国际公司

地址 美国加利福尼亚州

[72] 发明人 萨什坎斯·钱德拉塞克拉恩

罗杰·班福德 威廉·布里奇

大卫·布劳尔 尼尔·麦克诺顿

威尔逊·尚 维纳伊·斯瑞哈瑞

[56] 参考文献

US5193162A 1993.3.9

EP0499422A2 1992.8.19

WO99/41664A1 1999.8.19

US5327556A 1994.6.5

WO91/03024A1 1991.3.7

MEMORY QUEUE PRIORITY MECHNISM FOR A RISCPROCESSOR IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, Vol. 37 No. 6 1994

审查员 解 欣

[74] 专利代理机构 北京康信知识产权代理有限责任公司

代理人 余 刚

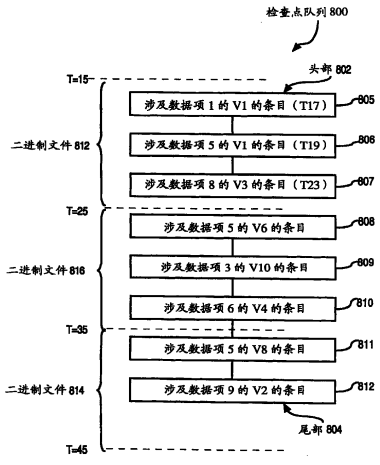
权利要求书 5 页 说明书 38 页 附图 14 页

[54] 发明名称

对多节点系统中的检查点队列进行管理

[57] 摘要

提供了用于管理一个系统中的缓存区的方法，该系统具有多个缓存区，其中多个缓存中可以具有同一数据项的不同拷贝。 具体的说，提供了用于对在这种数据项上执行的磁盘写入操作进行协调的方法，以确保数据项的较旧版本不会改写较新版本，并且减少故障之后进行恢复所需要的处理量。 并且提供了各种方法，其中使用了一个主管来与多个缓存区进行协调，以便将数据项写入永久性存储器。此外还提供了技术方法来管理那些与缓存区相关联的检查点，其中使用检查点来确定一个位置，在出现故障时，对恢复日志进行的处理始于这个位置。



1. 一种用于对将数据项写入到永久性存储器的操作进行协调的方法，所述方法包括步骤：

在第一节点内部为需要写入永久性存储器的脏数据项保持一个第一队列；

在第一节点内部为需要写入永久性存储器的脏数据项保持一个第二队列；

当对应于条目的脏数据项需要转移到所述第一节点之外的一个节点时，将条目从所述第一队列移动到所述第二队列；以及

在对写入到永久性存储器中的数据项进行选择的时候，为与所述第二队列中的条目相对应的数据项提供优先级。

2. 根据权利要求 1 所述的方法，其中移动条目的步骤包括响应于所述第一节点接收的消息而将一个条目从所述第一队列移动到所述第二队列，其中所述消息指示另一个节点已经请求了对应于所述条目的数据项。

3. 一种用于对将数据项写入到永久性存储器的操作进行协调的方法，所述方法包括步骤：

为每个所述数据项保持一个强制写入计数；

每当一个节点将一个数据项写入永久性存储器，以使所述数据项转移到另一个节点时，则递增所述数据项的强制写入计数；

基于与数据项相关联的强制写入计数高于某个阈值，来选择写入永久性存储器中的脏数据项。

4. 根据权利要求 3 所述的方法，还包括步骤：

将那些强制写入计数超出某个阈值的脏数据项存入某个特定队列；以及

在对写入到永久性存储器中的脏数据项进行选择的时候，为存储在所述特定队列中的数据项提供优先级。

5. 一种用于在故障之后进行恢复的方法，所述方法包括步骤：

确定所述故障是否仅涉及一个节点；以及

如果所述故障仅涉及所述一个节点，则通过从恢复日志中的第一个点开始应用所述节点的恢复日志来执行恢复；以及

如果所述故障涉及所述一个节点之外的一个或多个节点，则通过从恢复日志中的第二个点开始应用所述节点的恢复日志来执行恢复；

其中所述第一个点不同于所述第二个点。

6. 根据权利要求 5 所述的方法，其中：

第一点至少部分是由驻留在其他节点缓存区中并由所述节点弄脏的数据项来确定的；以及

第二点至少部分是由被永久存储的所述节点弄脏的数据项来确定的。

7. 一种用于在系统出现一次故障之后恢复数据项当前版本的方法，其中所述系统包含多个缓存区，所述方法包括步骤：

修改所述多个缓存区的第一节点中的数据项，以便创建一个经过修改的数据项；

将修改过的数据项从所述第一节点发送到所述多个缓存区的第二节点，而不将来自所述第一节点的已修改数据项持久保存在永久性存储器中；

在已经从所述第一节点将所述已修改数据项发送到所述第二节点之后，并且在所述第一节点中的所述数据项由一个写入磁盘操作所覆盖之前，丢弃所述第一节点中的所述数据项；

在所述故障之后，基于与所有所述多个缓存区相关联的合并的重做日志而将改变应用于永久性存储器上的数据项，由此重新构建所述数据项的当前版本；

为所述多个缓冲区的每一个都保持一个全局变脏的检查点队列以及一个本地变脏的检查点队列；

其中，在由写入磁盘操作覆盖之前，与全局变脏的检查点队列中的条目相关联的全局变脏的数据项是不会保留的；

基于位于本地变脏的检查点队列头部的条目的首次变脏时间以及位于全局变脏的检查点队列头部的条目的首次变脏时间中较低的一个来为每个缓存区确定一个检查点；以及

在所述故障之后，基于为所述缓存区确定的检查点来判定从何处开始处理与每个缓存区相关联的重做日志。

8. 一种用于在系统出现一次故障之后恢复数据项当前版本的方法，其中所述系统包含多个缓存区，所述方法包括步骤：

修改所述多个缓存区的第一节点中的数据项，以便创建一个经过修改的数据项；

将修改过的数据项从所述第一节点发送到所述多个缓存区的第二节点，而不将来自所述第一节点的已修改数据项持久保存在永久性存储器中；

在已经从所述第一节点将所述已修改数据项发送到所述第二节点之后，并且在所述第一节点中的所述数据项由一个写入磁盘操作所覆盖之前，丢弃所述第一节点中的所述数据项；

在所述故障之后，基于与所有所述多个缓存区相关联的合并的重做日志而将改变应用于永久性存储器上的数据项，由此重新构建所述数据项的当前版本；

为所述多个缓冲区的每一个都保持一个全局变脏的检查点队列以及一个本地变脏的检查点队列；

其中，在由写入磁盘操作覆盖之前，与全局变脏的检查点队列中的条目相关联的全局变脏的数据项是不会保留的；

为每个缓存区保持一个涉及本地变脏的检查点队列的第一检查点记录，所述记录指示了一个第一时间，其中在第一时间之前对缓存区中当前变脏的数据项所进行的所有改变都被记录到位于永久性存储器上的数据项的一个版本中；

为每个缓存区保持一个涉及全局变脏的检查点队列的第二检查点记录，其中第二检查点记录包括了数据项的一个列表，这些数据项在缓存中曾经变脏，但是此后已被转移出去并且不会写入永久性存储器；以及

在所述故障之后，基于涉及所述缓存的所述第一检查点记录以及所述第二检查点记录来确定从何处开始处理与每个缓存区相关联的重做日志。

9. 根据权利要求 8 所述的方法，其中处理重做日志的步骤包括以下步骤：

确定一个起始位置，以便基于第一检查点记录所确定的重做日志中的一个位置以及对第二检查点记录中的数据项的列表所进行的较早改变而确定的日志中的位置中较小的一方来扫描重做日志；以及

在恢复过程中，对于全局检查点记录所指示位置到本地检查点记录所指示位置之间的重做日志的位置来说，为了可能

的重做而仅仅对那些与全局检查点记录中所标识数据项相对应的日志记录加以考虑。

对多节点系统中的检查点队列进行管理

相关申请；优先权声明

本专利申请涉及 2001 年 3 月 7 日提交的美国临时专利申请 60/274,270 并且要求享有该专利申请的优先权，所述专利申请的名称是 METHODS TO PERFORM DISK WRITES IN A DISTRIBUTED SHARED DISK SYSTEM NEEDING CONSISTENCY ACROSS FAILURES，其内容在此全面引入作为参考。

技术领域

本发明涉及磁盘写入操作的执行，尤其涉及对多个系统中的脏数据项的写入进行协调，其中这些系统允许一个数据项的脏数据版本驻留在多个节点的缓存区中。

背景技术

一种改进数据库系统中的可扩展性的方法是允许多个节点同时读取和修改共享存储器中的数据。每个节点都具有一个缓存区，以便将数据保持在易失主存储器中，并且使用非易失共享磁盘存储器来进行备份。一个全局锁管理器（GLM）或一个分布式锁管理器（DLM）用来维持节点之间的缓存区相关性。为了提供从清除主存储器内容的节点故障中进行的恢复，使用了通用的预写式日志（WAL）协议。出于性能原因，每个节点都具有一个记录各种改变的私有重做日志。而为了减少节点故障之后需要在重做日志中被扫

描的改变的数量，通常会采用递增或周期性的检查点，由此确保不必将检查点之前对数据项所做的全部改变重新应用于非易失存储器中的数据项。

并发控制

在运行于相同或不同节点的事务之间执行的并发控制是借助于全局事务页级锁或行级锁来实施的。事务系统可以使用强制策略或非强制性策略，在强制策略中，经过事务修改的数据项（例如页面/数据块）在提交事务的过程中写入到稳定存储器中，而在非强制策略中，只有重做日志中的事务变化是在提交事务时强制执行的。将强制策略与页级锁一起使用，这意味着数据块仅仅由一个节点（实际上仅仅由一个事务）进行修改，并且在任何时刻都只能在一个系统缓存区中变脏。在所有其他组合中（也就是行级锁与强制策略一起，页级锁与非强制策略一起，以及行级锁与非强制策略一起），可以在多个系统中修改这些数据项并且还需要一种缓存区一致机制。

最普通的范例是具有非强制数据项管理策略的行级锁。出于说明目的，以下给出的例子是在使用了具有非强制数据项管理策略的行级锁的系统环境中提供的。然而，这里描述的技术并不局限于这种环境。

检查点队列

在提交事务时，反映事务所进行的改变的数据必须保存在永久性存储器中。在某些系统中，在提交的时候必须持续保存那些指示事务所进行的改变的重做记录，但是可以延迟那些经过修改的数据项自身的实际写入。那些（1）包含变化和（2）没有持续保存的数据项称为“脏数据项”。通常，节点中的脏数据项越多，如果节点

出现故障，则恢复时间也就越长。因此，为了确保恢复时间不会长到难以接受，节点可以保持一个检查点队列。

检查点队列包含了标识脏数据项的条目。队列中的条目是根据持续保存的重做日志中的相应重做记录顺序来排列的。如果发生故障，则必须从一条重做记录开始而对重做日志进行处理，其中该记录与位于检查点队列头部的条目相对应。

在把一个脏数据项写入永久性存储器时，对应于这个数据项的条目将从检查点队列中删除。在从检查点队列中删除位于检查点队列头部的条目时，在重做日志内部，恢复处理开始的位置必须开始改变而导致检查点“提前”。检查点在重做日志中提前的越多，在出现故障时，从故障中恢复所要完成的工作也就越少。因此，节点通常会尝试将那些由位于检查点队列头部的条目所标识的脏数据项写入永久性存储器。然而，如在下文更详细描述的那样，在多个节点的缓存区中有可能存在同一数据项的多个脏数据版本，这时，协调脏数据项的写入尤为重要。

经由共享的永久性存储器的数据项转移

在多个系统同时修改数据项时，需要一种机制来协调已修改数据项关于稳定的共享永久性存储器写入。在某些系统中使用了稳定的共享永久性存储器作为介质，以便将经过修改的数据项从一个节点转移到另一个节点，由此简化这个问题。当节点中一个脏数据项需要在一个不同节点加以修改的时候，在将页面锁授予这个希望修改脏数据项的节点之前，数据项首先写入到共享的永久存储器中。当一个不同节点需要读取所修改数据项的当前版本时，相同的写入永久性存储器和读取永久性存储器的序列将会得到使用。

经由互连的数据项转移

在使用非易失存储器作为介质，由此在节点之间转移数据项的系统中，没有必要协调不同节点之间的脏数据项写入。每个节点可以使用常规机制来写出脏数据项及执行检查点。

在某些系统中，当请求节点仅仅需要所修改数据项的一个一致性快照版本时，经过修改的数据项将会发送到请求节点，而不会将该数据项写入永久性存储器。因此，通过这种一致性控制机制，尽管不同节点中的多个事务可以在提交事务之前使用行级锁来修改同一数据项，但是任何数据库的数据项都只在一个节点缓存区中是脏的。因此，在节点出现故障时，只需要从节点的检验点记录开始扫描该节点的重做日志，一直扫描到其重做日志末端，由此即可恢复数据库。此外，当多个节点发生故障时，可以扫描每个节点的重做日志并且按序应用这些重做日志，由此恢复数据库，也就是说，不需要合并来源于多个重做日志的改变。

然而，为了改善从一个具有排他锁并且有可能修改了数据项的节点向一个请求相同数据项以供排他使用或是请求现有版本以供读取的节点进行数据项转移的等待时间，较为理想的是把数据项从一个节点的主存储器直接转移到另一个节点的主存储器，而不先将数据项写入永久性存储器。当把脏数据项从一个节点转移到另一个节点时，称为过去镜像（past image）（PI）的数据项的一个拷贝既可保留在发送节点中，也可以不在发送节点中保留。

当节点准许转移脏数据项而不需要将其存入永久存储器时，必须在不同节点之间对脏数据项的写入进行协调。如果没有进行协调，则转移了脏数据项的节点可能希望通过把脏数据项写入永久存储器来使其检查点提前。然而，如果其他某些节点已经将数据项的更新版本写入了永久存储器，则将脏数据项写入永久存储器有可能会破坏数据完整性。

此外，除非把脏数据项写入磁盘，否则检查点不能提前。如果一个节点并未保持那些由该节点发送到其他节点的数据项的脏数据版本，则这个节点必须以某种方式来与另一个节点协调写入操作。

此外，对于一个可扩缩的系统来说，由系统执行的写入磁盘操作的数目不应该是系统节点数目的一个函数。相反，写入磁盘操作的数目应该只反映了对系统内部数据项的实际改变。

基于上文，较为理想的是提供那些对系统中的脏数据项写入进行协调的技术方法，在这些系统中，同一数据项的脏数据版本有可能位于一个以上的易失存储器之中。

发明内容

由此提供了用于管理一个系统中的缓存区的技术方法，该系统具有多个缓存区，这些缓存区可以包含同一数据项的不同拷贝。具体的说，提供了技术方法，用于协调在这种数据项上执行的磁盘写入操作，由此确保数据项的旧版本不会改写新版本，并且减少故障之后进行恢复所需要的处理量。并且提供了各种方法，其中使用了一个主管（master）来与多个缓存区器进行协调，以便将数据项写入永久性存储器。这种方法包括但不局限于直接写入方法、间接写入方法、基于拥有者的方法以及基于角色的方法。此外还提供了技术方法来管理那些与缓存区相关联的检查点，其中使用检查点来确定发生故障时开始处理恢复日志的位置。

附图说明

本发明是借助实例来描述的，但这并不作为限制，在附图的图形中，相同的参考数字代表相同的部件，其中：

图 1 是描述如何根据本发明一个实施例而在直接写入方法中协调磁盘写入操作的框图；

图 2 是描述如何根据本发明一个实施例而在间接写入方法中协调磁盘写入操作的框图；

图 3a 是描述根据本发明的一个实施例，当全局变脏标记为假时，如何在基于拥有者的写入方法中协调磁盘写入操作的框图；

图 3b 是描述根据本发明的一个实施例，当全局变脏标记为真时，如何在基于拥有者的写入方法中协调磁盘写入操作的框图；

图 3c 是描述根据本发明的一个实施例，当写入请求并非来自拥有者时，如何在基于拥有者的写入方法中协调磁盘写入操作的框图；

图 4a 是描述根据本发明一个实施例，当模式为本地的时候，如何在基于角色的写入方法中协调磁盘写入操作的框图；

图 4b 是描述根据本发明的一个实施例，当模式为全局的时候，如何在基于角色的写入方法中协调磁盘写入操作的框图；

图 4c 是描述根据本发明的一个实施例，当请求并非来自排他锁持有者的时候，如何在基于角色的写入方法中协调磁盘写入操作的框图；

图 4d 是描述根据本发明一个实施例，当转移是在写入操作过程中执行的时候，如何在基于角色的写入方法中协调磁盘写入操作的框图；

图 5 是一个描述检查点队列的框图；

图 6 是一个描述检查点队列的框图；

图 7 是一个对具有合并条目的检查点队列进行描述的框图；

图 8 是一个描述检查点队列的框图，其中条目分批处理到二进制文件中；以及

图 9 是一个描述可以实施本发明实施例的计算机系统的框图；

具体实施方式

以下描述一种用于协调数据项写入的方法和设备。在以下描述中，基于说明的目的而对许多细节进行了阐述，以便提供关于本发明的全面理解。然而很明显，本发明可以在不具备这些特定细节的情况下实施。在其他情况下，为了避免不必要地造成本发明不清楚，众所周知的结构和设备将以框图形式显示。

对使用一个永久性存储器作为介质用于转移的系统进行优化

在使用永久性存储器作为介质而在两个节点之间转移数据的系统中，可以修改数据库缓存区的写入子系统而将更高优先级提供给其他节点正在等待读写的写入数据项，由此减少数据项从一个节点转移到另一个节点所需要的等待时间。这个操作可以通过为那些需要写入的脏数据项提供一个单独队列（一个“侦测（ping）队列”）而得以实现，因为其他节点正在等待读取或修改这些脏数据项。当锁管理器（这个锁管理器可以是分布式锁管理器 DLM 或是全局锁

管理器 GLM）向保持节点发送一个消息来请求保持节点释放它对数据项的封锁时，脏数据项可以根据要求而被移动到侦测队列。

根据另一种方法，可以在每个数据项头部或是数据项控制块中保持一个“强制写入”计数，由此减少数据项从一个节点转移到另一个节点所需要的等待时间。只要执行写入而把数据项转移到另一个节点，强制写入计数就会递增。永久性存储器的写入子系统保持了脏数据项的一个高优先级队列，其中这些脏数据项的强制写入计数要高于某个阈值。这个队列则用于使那些数据项的写入比节点间未曾经常共享的其他脏数据项更为频繁。此外，在预期到这些数据项上的封锁需要释放的情况下，由于数据库缓存区的写入子系统急切写出了脏数据项，因此这将会改善节点之间转移封锁的等待时间。

然而，即使在以这种方式进行优化时，那些使用共享的永久性存储器作为介质而在节点之间转移数据项的系统也要遭受到与把数据项写入永久性存储器相关联的开销。以下描述的技术涉及这种系统，其中包括脏数据项在内的数据项可以在节点之间转移，而不需要首先写入永久性存储器。

正确性和可扩缩性

在那些允许在缓存区之间转移脏数据项，而不需要首先将其存入永久性存储器的系统中，出于正确性和可扩缩性的考虑，需要对各个不同节点的缓存区中的脏数据项写入进行协调。正确性需要在节点完成一个检查点（也就是记录一个起始点，从这个起始点开始，在出现故障之后，有可能应用那些来自其重做日志的变化）的时候，包含那些在检查点之前提交的改变的每个数据项的一个版本都已写入了非易失永久性存储器。此外，两个节点不能同时将一个数据项写入永久性存储器（因为它们可能会攻击相互的变化），并且节点不准使用数据项的一个较旧版本来改写一个较新版本。

可扩缩性需要将数据项写入永久性存储器的操作覆盖尽可能多的改变，即使这些改变是由不同节点造成的。基于有效性原因，数据库系统有可能希望限制那些需要扫描并有可能在节点故障之后重新使用的重做日志数量。因此数据库写入数量可以与对数据项做出的改变数量成正比，但是不和做出这些改变的节点的数目成正比。

功能概述

由此提供了不同的技术方法来协调脏数据项相对于系统永久性存储器的写入，其中这些系统允许同一数据项的脏数据版本保留在多个缓存区中。根据一种技术，这种协调是使用一个指派给该数据项的主管来完成的。根据一个实施例，用于协调数据项脏数据版本写入的主管与指派为管理封锁的实体是相同实体，其中所述封锁是对数据项访问加以控制。在这个实施例中，主管通常是锁管理系统的一个组件，例如归属于分布式锁管理系统或全局锁管理系统的一个锁管理器。

在一个实施例中，一个想要把脏数据项写入永久性存储器的节点会将一个写入永久性存储器的请求发送到分配给该数据项的主管。主管可以（1）授予请求节点执行写入的权限，或者（2）向请求节点告知另一个节点已经向永久性存储器中写入了一个至少与请求节点所保存脏数据版本一样新的版本。

在另一个实施例中，作为响应，主管也可以发送一个“执行写入”的消息，以便要求请求节点之外的一个节点将至少与请求节点中保存的脏数据版本一样新的数据项版本写入永久性存储器。当另一个节点向主管发送了一个“确认写入”的消息来指示已经执行了写入之后，主管会发送一个“写入通知”消息，以便向请求节点告知另一个节点已经把至少与请求节点所保存脏数据版本一样新的一个数据项版本写入了永久性存储器。

一旦将数据项的某个特定版本写入了永久性存储器，那么与这个特定版本相同或是比它旧的数据项的脏数据版本将会因为写入了这个特定版本而遭到覆盖。所覆盖的数据项版本不再需要（并且不应该）写入永久性存储器。在这里提到的包含所覆盖版本的节点称为“感兴趣的”节点。

除了向请求节点告知已经把数据项写入永久性存储器之外，主管还可以发送写入通知消息来向所有的感兴趣的节点告知已经将数据项写入了永久性存储器。在接收到已经将数据项写入永久性存储器的确认时，其他感兴趣的节点的写入通知消息可以立即发送，也可以延迟到某些其他事件之前再进行发送。

在另一个实施例中，每当各个节点想要把脏数据项写入永久性存储器的时候，这些节点都需要对主管进行询问，为了避免这种情况，主管可以向一个节点授权数据项的“所有权权限”。当一个节点拥有数据项所有权权限的时候，该节点可以随意将数据项写入永久性存储器，而不需要向数据项主管发送写入请求消息。所有权权限可以隐含地与排他锁的所有权一起授予，也可以从排他锁的授予中分离并且独立于排他锁的授予。

根据一个实施例，为数据项保持了一个“全局变脏”的标志。如果一个节点将数据项的脏数据版本转移到另一个节点，则将会把全局变脏的标志设定为“真”。当拥有者将一个数据项写入永久性存储器时，如果全局变脏的标志设定为“真”，则拥有者会向主管发送一个写入确认消息。然后，主管可以向感兴趣的节点发送写入通知消息。另一方面，如果将全局变脏的标志设定为“假”，则当拥有者写入数据项时，该拥有者无需向主管发送一个写入确认消息。

直接写入方法

根据直接写入的方法，把脏数据项写入永久性存储器的操作是使用一个指派给该数据项的主管来协调的。特别地，一个想要把脏数据项写入永久性存储器的节点会向指派给该数据项的主管发送一个写入请求消息。而这个主管可以（1）授予请求节点执行写入的权限，或者（2）向请求节点告知另一个节点已将一个至少与请求节点保存的脏数据版本一样新的数据版本写入了永久性存储器。

更具体地说，当从一个节点缓存区中侦测出一个脏数据项时，也就是说，当另一个节点需要相同数据项的一个当前版本，以便进行读取（S 锁）或写入（X 锁）时，发送节点缓存区中的数据项状态将会改变为 PI。而数据项仍旧保留在脏数据队列或检查点队列中。在侦测出一个干净的数据项时，该数据项既可标记为自由，也可以保留在缓存区中，从而满足一致性快照的读取。

在侦测出数据项时，数据项主管记录了数据项版本号。通常，这个版本号是一个日志序列号（LSN），一个系统提交号（SCN）或一个全局性唯一时戳，该时戳可用于与重做日志中发生了变化的数据项的版本相关联。由于数据项仍处于脏数据队列或检查点队列之中，因此检查点或缓存区的写入子系统最终需要写出 PI（或是它的某些后继者）。

根据直接写入方法，一个消息发送到主管，该主管返回数据项较新版本的一个状态已经写入或者向请求节点授予写入权限。来自其它节点的同一数据项的其他写入请求将会排队，直到写入节点用一个写入完成状态来响应锁管理器。在将一个 PI 写入永久性存储器之后，数据项主管会把 PI 版本号记录为当前在永久性存储器上的版本。

参考图 1，该图是描述一个使用了直接写入方法的系统的框图。节点 1、2 和 3 在其缓存区中分别保存了某个特定数据项的版本 V1、V2 及 V3。假设 V3>V2>V1，其中 A>B 意味着 A 是一个比 B 新的数据项版本。

主管 **100** 即为指派给数据项的主管。在图 1 描述的方案中，节点 1 和 3 向主管 **100** 发送写入请求。为了防止多个节点同时写入相同数据项，举例来说，主管 **100** 可以为每个数据项包含一个写入请求队列。为数据项接收的写入请求保存在写入请求队列中，并且这些写入请求依次由主管进行处理。在所述实例中，主管 **100** 首先对来自节点 3 的写入请求进行处理，而来自节点 1 的写入请求则保持在写入请求队列中。主管 **100** 向节点 3 发送一个准许节点 3 把 V3 写入永久性存储器的写入执行消息，由此对节点 3 的写入请求做出响应。

在同意节点 3 的写入请求的同时，主管 **100** 不会向任何其他节点授予写入永久性存储器的权限。因此，来自节点 1 的写入请求会在写入请求队列中保持挂起。

在节点 3 已将 V3 写入永久性存储器之后，节点 3 会向主管 **100** 发送一个写入确认消息，该消息指示写入永久性存储器的操作已经完成，并且节点 3 已经释放了写入永久性存储器的权限。由于 V3 比 V1 和 V2 更新，因此 V1 和 V2 将会因为 V3 的写入而被覆盖。

然后，主管 **100** 继续处理队列中的下一个写入请求。在本实例中，主管 **100** 对来自节点 1 的写入请求进行处理。节点 1 的写入请求是一个要求写入 V1 的请求。由于 V1 已经由 V3 的写入所覆盖，因此，主管 **100** 向节点 1 发送一个指示 V1 已被覆盖的写入通知消息。响应于这个写入通知消息，节点 1 从它的检查点队列中删除关于 V1 的条目，而不会将 V1 写入永久性存储器。由于节点 1 现在

知道 V1 已被覆盖，因此节点 1 不需要在存储器中保持 V1 的一个拷贝。

根据一个实施例，节点 2 包含了由于写入 V3 而被覆盖的 V2，在节点 2 向主管 **100** 发送一个关于 V2 的写入请求之前，不会向该节点发送一个写入通知消息。

间接写入方法

在使用直接写入方法的情况下，每个节点为节点检查点队列中的每个条目发送一个写入请求消息。在某些情况下，节点会接收一个响应于这个写入请求的写入执行消息。当接收到一个写入执行消息时，请求节点必须执行一个写入操作。在其他情况下，请求节点会接收一个响应于写入请求的写入通知。当接收到一个写入通知消息时，请求节点不需要执行写入操作。

间接写入方法尝试提高与写入通知消息一起应答的写入请求所占有的百分比。为此目的，相对于要求执行写入操作的节点而言，主管 **100** 是有选择的。特别地，主管 **100** 可以通过向一个节点发送一个写入执行消息而对来自另一个节点的写入请求消息做出响应。可以基于多种因素来选择写入执行消息所发至的节点，其中包括缓存区中保存的数据项版本的新近状态。根据一个实施例，主管 **100** 总是将写入执行消息发送到包含数据项当前版本的节点，而不会考虑发送写入请求的节点。

更具体的说，根据一个实施例，主管将写入请求转发到具有过去镜像中的最高版本的节点，或者优选将其转发到排他锁（X）的持有者（该持有者具有数据项的当前版本）。将写入请求转发到最高 PI，而不是排他锁持有者，这就允许对当前数据项不断进行修改。

在将一个数据项写入永久性存储器的时候，该数据项是不能修改的；因此，为了写入一个有可能会受到进一步修改的当前数据项，有必要将其封锁来防止修改，或者对其进行“克隆”，从而对一个不同的拷贝进行修改。封锁通常不合乎需要；如果克隆是可行的，则较为优选的是把写入请求指引到具有当前数据项（也就是 X 锁或 S 锁）的节点。

通过将数据的当前版本写入永久性存储器，这使得永久性存储器能够进行写入，以便覆盖尽可能多的变化。在完成了永久性存储器写入的时候，具有写入完成状态以及所写入数据项版本号的一个消息将会发送到主管。主管记录永久性存储器上的版本号，并将写入通知消息发送到具有数据项 PI 版本的所有节点，其中该数据项现在将因为永久性存储器的写入而受到覆盖。当一个节点接收到一个写入通知时，如果其脏数据队列或检查点队列上的所有数据项在检验点记录之前已经写入了永久性存储器，或是由于写入了其他节点中的相同数据项而从主管那里接收到写入通知，则该节点可以使其检验点记录恰当提前并且释放 PI 数据项。在写入一个数据项的时候，主管在逻辑上保持一个写入请求队列，但是只需要记录所接收的最高写入请求的版本号。

例如，在图 2 描述的方案中，节点 3 并未将一个关于 V3 的写入请求消息发送到主管 **100**。然而，响应于来自节点 1 并要求写入数据项版本 V1 的写入请求消息，主管 **100** 选择节点 3 作为写入数据项的节点。节点 3 通过写入数据项版本 V3 并将一个写入确认消息发送到主管 **100** 来做出响应。然后，主管 **100** 把一个写入通知消息发送到节点 1。

由于选择节点 3 来将 V3 写入永久性存储器，因此 V1 和 V2 都会被覆盖。与之相反，如果（根据直接写入方法）主管 **100** 已经授

权节点 1 写入 V1，则 V2 和 V3 不会受到覆盖。在将 V2、V3 写入永久性存储器的时候，必须执行分离的写入操作。

间接写入方法还向那些未曾发送写入请求消息的节点以及发送了写入请求消息的节点优先发送写入通知消息，由此尝试减少必需发送到主管 **100** 的写入请求消息的数目。举例来说，在图 2 描述的方案中，在使用间接写入方法的情况下，即使节点 2 并未发送一个关于 V2 的写入请求，主管 **100** 还是会将一个写入通知消息发送到节点 2。根据一个实施例，主管 **100** 会把写入通知消息发送到所有的感兴趣的节点。

当一个感兴趣的节点收到一个写入通知时，该节点从它的检查点队列中删除关于该数据项相应版本的条目。在使用间接写入方法的情况下，在为检查点队列中的很多条目发送写入请求之前，很多条目都可以借助这种方式而被删除。因此，节点发送的写入请求消息数量明显少于其位于检查点队列中的条目数量。

基于拥有者的写入

在间接写入方法和直接写入方法中，即使数据项仅仅在一个节点缓存区中变脏，写入请求消息也会发送到数据项主管。在许多数据库系统中，可以通过在节点之间对内部永久性存储器进行分区（例如为每个节点分离数据项的自由列表）或者借助于把事务路由到节点的应用等级而在节点之间划分数据库工作集的一个显著部分。在这种系统中，数据项只在一个节点缓存区中频繁变脏。基于拥有者的写入方法免除了在这些情况下发送写入请求的需要。

基于拥有者的写入方法导致了由当前指定为数据项“拥有者”的节点所进行的数据项的所有写入。与直接写入或间接写入的方法相反，当数据项拥有者希望得到将要写入的数据项的一个版本时，该拥有者允许将这个数据项写入永久性存储器，而不会将写入请求

消息发送到数据项主管。此外还可以使用不同因素来选择充当数据项拥有者的节点。根据一个实施例，数据项拥有者是基于以下规则来选择的：

（1）如果向节点授予了关于数据项的排他锁，则认为该节点即为数据项持有者；

（2）如果不存在排他锁拥有者，也就是说，存在多个共享锁（S）的持有者，则具有数据项的最近排他锁的节点将会选为数据项拥有者；以及

（3）如果数据项未曾由任何节点变脏，则该数据项的拥有者是不存在的。

在一个作为数据项拥有者的节点中，即使数据项不会在该节点变脏，该数据项也会链接到节点的脏数据队列或是检查点队列。

在数据项拥有者把数据项写入永久性存储器之后，该拥有者确定数据项是否“全局变脏”。如果由拥有者之外的任何节点所进行的任何修改都没有被该节点存入永久性存储器，则这个数据项是全局变脏的。如果数据项是全局变脏的，则拥有者会把一个写入确认消息发送到主管。然后主管可以向感兴趣的节点发送写入通知。如果数据项不是全局变脏的，则拥有者不需要向主管发送一个写入确认消息。

可以使用不同的技术方法来使数据项拥有者能够确定是否数据项是全局变脏的。根据一个实施例，一个全局变脏的标志与数据项相关联。当节点把数据项的一个脏数据版本发送到永久性存储器而不是写入永久性存储器的时候，发送节点会把这个数据项的全局变脏标志设定为“真”。为了确定数据项是否是全局变脏的，所述拥有者只需要调查那些与这个数据项相关联的全局变脏标志。如果

写入永久性存储器的数据项的版本是（1）数据项的当前版本或者（2）最新的 PI 版本，则在将数据项写入永久性存储器之后，拥有者会把全局变脏标志设定为“假”。

可以使用多种方式来保存一个数据项的全局变脏标志。例如，当数据项是数据库系统中的一个数据项时，全局变脏标志可以存入（1）保存数据项的数据块的数据块头部，（2）数据项的数据项控制块，（3）当授予数据项的新拥有者等等进行封锁时，在一个本地锁管理器中的锁结构。

参考图 3a，该图描述了一种方案，在这个方案中，数据项拥有者（节点 3）希望将一个数据项写入永久性存储器，其中全局变脏标志设定为“假”。在图 3a 中可以看出，在这些情况下，节点 3 不需要从主管 **100** 那里索要权限。另外，节点 3 无需向主管 **100** 告知已经执行了写入永久性存储器的操作。

参考图 3b，该图描述了一种方案，在这个方案中，数据项拥有者（节点 3）想要把一个数据项写入永久性存储器，其中全局变脏标志设定为“真”。在所述方案中，节点 1 和 2 具有数据项的脏数据版本 V1 和 V2，这两个版本比节点 3 中保存的版本 V3 旧。与图 3a 所示方案相似，在这种方案中，节点 3 无需请求权限以便将 V3 写入永久性存储器。然而，由于全局变脏标志是“真”，因此在将 V3 写入永久性存储器之后，节点 3 会把一个写入确认消息发送到主管 **100**。然后，主管 **100** 把写入通知消息发送到节点 1 和 2。在将 V3 写入永久性存储器之后，节点 3 把全局变脏标志设定为“假”。

参考图 3c，该图描述了一种方案，其中一个并非数据项拥有者的实体（节点 1）想要将数据项写入永久性存储器。在这个方案中，节点 1 把一个写入请求消息发送到主管 **100**。然后，主管 **100** 把一个写入执行消息发送到数据项拥有者（节点 3）。节点 3 把 V3 写入永久性存储器，并将全局变脏标志设置为“假”，此外该节点还向

主管100发送一个写入确认消息。然后，主管100向感兴趣的节点（节点1和2）发送写入通知消息。

基于角色的方法

基于拥有者的方法免除了在将数据项写入永久性存储器之前由数据项拥有者从数据项主管那里获取权限的需要。然而，为了消除两个节点同时尝试将数据项写入永久性存储器的可能性，在数据项当前拥有者将数据项写入永久性存储器的时候，数据项所有权是不准改变的。因此，在那些把排他锁持有者当作拥有者的系统中，在数据项的当前拥有者将数据项写入永久性存储器的时候，排他锁是不能传递到另一个节点的。结果，在将数据项写入永久性存储器之前，为将权限修改到一个希望修改数据项的并发节点而进行的转移将会延迟。这种延迟降低了系统的整体性能。另外，对于数据项拥有者而言，即使拥有者没有弄脏数据项，但是拥有者仍需在其脏队列中链接该数据项，这也是不合乎需要的。

基于角色的方法从（2）将数据项写入永久性存储器而不发送写入请求的权限中分离了（1）数据项中的排他锁的所有权。由于一个数据项中的排他锁的所有权与把数据写入永久性存储器而不发送写入请求的权限相分离，因此，即使在进行写入永久性存储器操作的时候，数据项排他锁的所有权也可以在节点之间转移。

根据基于角色的方法，为每个锁都指派了一个锁角色。如果数据项仅仅有可能会在一个节点缓存区中变脏，则这个锁角色是“本地的”。因此在整个系统中，当首次将数据项的一个封锁授予一个节点时，这个封锁是与本地角色一起授予的。由具有本地角色的一个锁进行封锁的数据项既可以由保持封锁不受主管干涉的节点写入永久性存储器，也可以由该节点从永久性存储器中读取。

当一个数据项因为一个来自不同节点的封锁请求而被从节点缓存区中侦测出时，如果数据项在保持节点缓存区中是脏的，则关于该封锁的角色将会转换成“全局”。否则，与数据项一起转移的锁将会仍由本地角色封锁。因此，只有在多节点系统中存在至少一个关于数据项的 PI 时，一个数据项才需要受到全局角色封锁。

当一个 PI 数据项或全局角色中的一个当前数据项需要写入永久性存储器时，其保持节点会把一个具有所要写入数据项的版本号的写入请求消息发送到主管。主管可以将写入请求转发到具有当前数据项的节点（X 锁的持有者）或是版本号大于或等于所要写入 PI 的版本号的任何 PI。在完成写入的时候，主管会把写入通知发送到所有节点，其中所述节点具有由写入永久性存储器的数据项版本所覆盖的 PI。

由于在全局角色中具有排他锁的节点还需要将它的写入永久性存储器操作与主管进行协调，因此，即使由排他锁封锁的数据项正被写入，也可以将所述排他锁转移到另一个节点。出于同样理由，除非节点上的数据项变脏，否则节点不会把数据项链接到它的检查点队列或是脏队列。在脏数据项受到本地角色封锁而被写入的同时，在侦测出一个脏数据项时，锁角色将会切换成全局，正在进行的写入也会传递到主管。

参考图 4a，该图描述了一种方案，其中本地模式锁的持有者（节点 3）希望将数据项的一个版本写入永久性存储器。由于节点 3 拥有的锁处于本地模式，因此节点 3 会把数据项写入永久性存储器，而不会从主管 **100** 那里索取许可。并且节点 3 还不需要向主管 **100** 告知已经将该数据项写入了永久性存储器。

参考图 4b，该图描述了一种方案，其中全局模式锁的持有者（节点 3）希望将数据项的一个版本 V3 写入永久性存储器。由于封锁模式是全局性的，因此另一个节点有可能正在写入数据项。

因此，节点 3 将一个写入请求消息发送到主管 **100**。响应于这个写入请求消息，主管 **100** 选择一个节点来写出数据项。优选地，主管 **100** 选择具有至少与版本 V3 一样新的数据项版本的一个节点。在当前实例中，V3 即为数据项的当前版本。因此，主管 **100** 向节点 3 回送一个写入执行消息。

响应于写入执行消息，节点 3 将 V3 写入永久性存储器，并且将一个写入确认信息回送到主管 **100**。然后，主管 **100** 将一个写入通知消息发送到感兴趣的节点（节点 1 和 2）。

如果写入永久性存储器的数据项版本即为当前版本，则将数据项写入永久性存储器的节点还把封锁从全局模式转换成本地模式。这个转换可以在将当前版本写入永久性存储器的时候执行。根据节点在数据项上保持一个排他锁这一事实，将当前版本写入永久性存储器的节点能够确定节点正在写入当前版本。在当前实例中，V3 即为当前版本，因此，在将 V3 写入永久性存储器之后，节点 3 会把模式从全局转换成本地。

参考图 4c，该图描述了一种方案，其中，未曾保持数据项当前版本的节点（节点 1）请求将数据项写入永久性存储器。图 4c 显示的事件顺序与图 4b 的那些相同，只不过写入请求消息来源于节点 1 而不是节点 3。

如图 4b 所示，与基于拥有者的方法相比，在基于角色的方法中，数据项排他锁的拥有者仍须寻求授权而在封锁处于全局模式时写入来源于主管 **100** 的数据项。然而，与基于拥有者的方法不同，一个数据项（及其排他锁）可以从一个节点转移到另一个节点，而不需要等待写入永久性存储器的操作结束。

举例来说，图 4d 描述了与图 4c 相同的方案，只不过一个节点（节点 4）已经请求了该数据项的排他所有权。即使节点 3 响应于

写入执行消息而正将 V3 写入永久性存储器，节点 3 也能将数据项转移到节点 4。在具有排他性写入封锁的情况下，节点 4 可以继续修改数据项来创建版本 V4。然而，由于模式是全局的，因此节点 4 无法将 V4 写入永久性存储器。

在图 4c 中，一旦从节点 3 接收到写入确认信息，那么主管 **100** 会向节点 4 发送一个转换到本地的消息。响应于转换到本地的消息的接收，节点 4 将模式从全局转换成本地。当把模式转回本地之后，在没有主管 **100** 的任何许可的情况下，节点 4 可以将数据项写入永久性存储器并从永久性存储器中读取数据项。

在一个替换实施例中，响应于写入确认消息，主管 **100** 并不发送一个转换到本地的消息。在没有转换到本地的消息的情况下，在节点 4，排他锁模式保持为全局。由于模式是全局，因此，如果节点 4 希望将 V4 写入永久性存储器，则节点 4 会将一个写入请求发送到主管 **100**。响应于写入请求消息，主管 **100** 可以将一个转换到本地的消息发送到节点 4。在将模式转换成本地之后，节点 4 在没有得到进一步许可的情况下写入 V4。

延迟的写入通知

在以上给出的情况中已经提及：将写入通知消息发送到感兴趣的节点是可以立即执行的，该发送也可服从于某些或全部感兴趣的节点。根据一个实施例，在执行写入永久性存储器操作的时候，一个写入通知消息仅仅立即发送到那些已经请求写入一个 PI 的节点，其中所述 PI 由已执行的写入所覆盖。举例来说，在图 1 中，主管 **100** 立即将一个写入通知消息发送到节点 1，而不是节点 2。

永久性存储器上的数据项版本号稍后可以使用各种技术中的任何一种而从主管那里发送到其他感兴趣的节点。举例来说，位于永久性存储器的数据项的版本号可以作为（1）新的封锁请求的封

锁许可消息，或者（2）当数据项当前版本需要发送到另一个节点时的侦测（ping）消息的一部分而被传达。因此，当另一个感兴趣的节点需要写入或替换其 PI 时，它们只与本地锁管理器进行通信即可丢弃它们的 PI。

分批处理的消息

用于减少主管与所关注节点之间所传递消息数目的另一种技术包括将往返于主管的写入请求消息以及写入通知消息分批处理成较少的更大消息，从而减少消息数量。举例来说，如果节点 1 希望将其检查点队列提前三个条目，则节点 1 可以将一个单独的写入请求消息发送到主管 **100**，该消息标识的是必须写入永久性存储器的所有三个数据（及其相应的版本）。同样，如果节点 1 关注到已经完成的三个写入永久性存储器的操作，则主管 **100** 可以将一个单独的写入确认消息发送到节点 1，该消息识别的是已经写入永久性存储器的三个数据项（及其相应的版本）。

检查点队列：管理相同数据项的多个过去镜像

在以上给出的方案中，假设每个节点的缓存区都具有每个数据项的最多一个 PI。实际上，在将数据项某个版本写入永久性存储器之前，数据项可以经由多个节点而循环若干次。较为正确的是在每次把脏数据项用侦测队列发送到另一个节点的时候都创建一个 PI，并且在节点缓存区的脏数据队列或检查点队列的不同位置具有关于若干 PI 的条目。

举例来说，图 5 描述了一种方案，其中节点的检查点队列 **500** 具有某个特定数据项（数据项 5）的三个条目。特别地，检查点队列 **500** 具有一个头部 **502**、一个尾部 **504** 以及与数据项 5 的版本 V1、V6 和 V8 相对应的三个条目 **506**、**508** 和 **510**。同样，图 6 描述了

一种方案，其中另一个节点的检查点队列 **600** 具有数据项 5 的两个条目。特别地，条目 **606** 和 **608** 对应于数据项 5 的版本 V3 和 V7。

出于说明目的，假定检查点队列 **500** 是节点 A（未示出）的检查点队列，并且检查点队列 **600** 是节点 B（未示出）的检查点队列。

数据项主管是与最新 PI 版本号一起更新的，其中该 PI 是将脏数据项转移到另一个节点之后才创建的。因此，当节点 A 创建数据项 5 的 V1 并将数据项 5 转移到另一个节点时，数据项 5 的主管将会更新，以便指示节点 A 具有 V1。当节点 A 随后创建数据项 5 的 V6 并将数据项 5 转移到另一个节点时，数据项 5 的主管将会更新，以便指示节点 A 具有 V6。同样，当节点 A 随后创建数据项 5 的 V8 并将数据项 5 转移到另一个节点时，数据项 5 的主管将会更新，以便指示节点 A 具有 V8。

然而，在将 PI 或是一个更近的版本写入永久性存储器之前，这个 PI 将会占用缓存区中的存储器，并且这个 PI 是不能替换的。因此，当一个脏数据项移出缓存区时，如果已经存在一个 PI，则新创建的 PI 可以与先前的 PI 合并（或是将其替换）。但是，与所合并 PI 相关联的检查点条目必须与合并中包含的最早版本的条目保持在脏数据队列或检查点队列中的相同位置，这是因为一个检查点不能够考虑完整直到在创建第一 PI 时对数据项进行的改变将会反映在数据项的永久性存储器版本上。此外，在合并中的最新版本由写入磁盘的操作覆盖之前，合并条目是不能从检查点队列中删除的。

例如，图 7 描述了检查点队列 **500**，其中与数据项 5 的版本 V1、V6 和 V8 相对应的条目 **506**、**508** 以及 **510** 合并为单个条目 **702**。由于条目 **506** 是合并中包含的最早条目，因此单个条目 **702** 位于条目 **506** 占据的位置。

部分覆盖的合并条目

在对一个数据项的 PI 进行合并时，当把数据项的一个版本写入一个不同节点的永久性存储器时，该版本有可能覆盖合并 PI 中反映的某些而非全部变化。举例来说，如果节点 B 把数据项 5 的 V7 写入永久性存储器，则只有与合并条目 **702** 的 V1 和 V6 相关联的变化将会受到覆盖。而与 V8 相关联的变化则不会受到覆盖。

当永久性存储器版本完全覆盖了所合并 PI 中包含的变化时，涉及 PI 的条目可以丢弃，检查点可以超过 PI 中进行的最早变化而得以提前。举例来说，如果已经将数据项 5 的 V9 写入了永久性存储器，则可以丢弃合并条目 **702**。

另一方面，当永久性存储器写入仅仅覆盖了所合并 PI 的某些变化时，所合并 PI 的条目不能被丢弃。举例来说，即使将 V7 写入永久性存储器这个操作能够允许从检查点队列 **500** 中删除那些没有经过合并的条目 **506** 和 **508**，但是它并不允许从检查点队列 **500** 中删除合并条目 **702**。

尽管不能丢弃与受到部分覆盖的所合并 PI 相对应的条目，但是可以在脏数据队列或检查点队列中将这个条目移动到对应于某个版本的条目的位置，其中所述版本位于写入到永久性存储器的版本之后。举例来说，在将数据项 5 的 V7 写入永久性存储器之后，在检查点队列 **500** 中，条目 **702** 可以移动到与数据项 5（也就是条目 **510**）的 V8 相对应的条目的位置。这允许检查点在第一条目还没有受到写入磁盘操作覆盖之前得以继续进行，而不会受到与所合并 PI 的条目的阻拦。

避免创建部分覆盖的合并条目

在某些系统中，脏队列或检查点队列是作为一个链接列表来执行的。在 CPU 使用方面，对链接列表进行扫描并将一个合并条目插入队列中的正确位置，这种操作有可能是非常昂贵的。可以执行一个存储器内部索引来简化这个操作，但在把数据项链接到检查点队列的时候，这将会造成额外的开销。

根据一个实施例，通过避免创建部分覆盖的已移动条目，可以免除对那些受到部分覆盖的合并条目进行移动所涉及的开销。具体地说，当一个合并操作有可能创建一个将会部分覆盖的合并条目时，该合并操作不会执行。

根据一个实施例，在（1）将数据项一个版本写入永久性存储器，以及（2）在节点之间转移数据项时，主管会把当前正在写入永久性存储器的数据项的版本号（“正被写入”的版本号）传递到该数据项正在移至的节点（“接收”节点）。接收节点由此知道不对等同或是早于正被写入版本的任何数据项版本以及晚于正被写入版本的任何数据项版本进行合并。

再次参考图 5 和 6，假定节点 A 对写入数据项 5 的 V6 进行处理。在完成写入操作之前，节点 A 将数据项 5 发送到节点 B，并且节点 B 修改数据项 5 的接收版本来创建数据项 5 的 V7。主管向节点 B 告知：数据项 5 的 V6 是在主管向节点 B 发送一个侦测队列时写入永久性存储器的。因此，节点 B 不会对数据项 5 的 V7 以及数据项 5 的 V3 进行合并，因为由此产生的合并数据项仅仅会因为 V6 的写入而受到部分覆盖。由于写入 V6 完全覆盖了 V3，因此在结束了 V6 的写入之后，节点 B 可以丢弃 V3 并从队列 **600** 中删除条目 **606**。

因此，在进行写入永久性存储器的操作中，和那些至少与正被写入版本一样旧的版本相关联的 PI 及条目可以相互合并，并且，与那些比正被写入版本更新的版本相关联的 PI 和条目可以相互合并。然而，对于那些至少与正被写入版本一样旧的版本以及那些比正被写入版本更新的版本而言，与这两种版本分别相关的 PI 不应该进行合并。

在一个最近版本持有者总是执行写入永久性存储器的操作的系统中，通过使用这种技术，确保了不会有合并 PI 受到写入永久性存储器操作的部分覆盖。具体地说，当一个节点被侦测而发送一个经历过写入永久性存储器操作的数据项时，该节点不会将数据项的新版本与旧版本相互合并。如果数据项没有经历过写入永久性存储器的操作，则接收到的数据项将是最新版本，并且此后不会有其他节点要求将数据项的更早版本写入永久性存储器。

一种避免写入对局部变化造成覆盖的替换方法是启发式的确定何时创建新的检查点队列条目，而不是合并现有的检查点队列条目。举例来说，假设存在对应于数据项 3 的版本 V7 的检查点队列条目。有可能会有必要确定是否为数据项 3 的新版本创建一个新条目或是将新版本与现有版本相合并。举例来说，对于是否合并所进行的判断可以启发性的基于对现有条目进行的首次变化相对于（1）重做日志中存在的最新变化以及（2）对脏队列或检查点队列开头数据项进行的最早变化到底有多长时间。这种启发式方法估计了与现有条目相关联的 PI 不久将被写入（或是因为写入而受到覆盖）的可能性，并且能使节点扩展检查点，使之超过 PI 中的首次变化。

举例来说，如果重做日志中的最新变化对应于远远晚于 V7 的一个时间，并且处于检查点队列开头的数据项与接近 V7 的一个时间相关联，则存在一个较高概率而使一个与现有条目相关联的 PI 不久即被写入（或是由一次写入所覆盖），因此应该为新版本产生

一个单独条目。另一方面，如果重做日志中的最新变化对应于一个接近 V7 的时间，并且处于检查点队列开头的数据项对应于一个远远早于 V7 的时间，则存在一个较低的可能性而使一个与现有条目相关联的 PI 不久即被写入（或是由一次写入所覆盖）。因此，新版本应该合并到现有条目中。

单个节点故障的检查点队列

如上所述，在重做日志内部，位于检查点队列开头的条目确定了故障之后必须开始恢复处理的位置。对于一个精确恢复来说，较为安全的是从对应于检查点队列开头的条目的位置开始处理重做日志，而不考虑此次故障涉及了群集内部多少节点。

根据一个实施例，提供了一种检查点机制来为每个节点追踪两个检查点：一个多重故障检查点和一个单独故障检查点。多重故障检查点指示了包括该节点的多节点故障之后开始处理节点恢复的位置。单独故障检查点指示了节点的单个节点故障之后开始处理节点重做日志的位置。

如在下文将要描述的那样，在不允许从多重故障检查点队列中删除条目的情况下，可以从单个故障检查点队列中删除条目。因此，单独故障检查点通常要比多重故障检查点提前很多。由于单独故障检查点被提前很多，因此保持单独故障检查点只会产生从单个节点故障中进行恢复而必须执行的较少工作。

相对于提前检查点而言，当节点将一个脏数据项转移到另一个节点时，该节点的多重故障检查点不会改变。由于数据项是脏的，因此在多重故障检查点队列中存在一个关于该数据项的条目。在转移了脏数据之后，该条目保持在多重故障检查点的队列中。

与之相反，在把脏数据项转移到另一个节点时，与脏数据项相关联的条目是从单独故障检查点队列中删除的。因为只要转移节点出现故障，那么对脏数据项所做的改变就不会丢失，所以从单独故障检查点队列中删除所转移脏数据项的条目是非常安全的。仅仅响应于转移节点的故障，转移节点所进行的改变是在发送到接收节点的数据项版本中反映出来的。在这些情况下，确保将改变存入永久性存储器的责任是与数据项一起转移的。因此，即使接收节点并没有对数据项进行任何进一步的修改，接收节点也必须任选其一（1）确保将转移节点所做的改变（或是关于这些变化的重做）写入了永久性存储器，或者（2）将脏数据项（以及责任）转移到另一个节点。

将脏数据项转移到另一个节点，这使得转移节点能够从它的单个节点故障的检查点队列删除关于所转移数据项的条目。因此，想要将其单个节点故障的检查点队列提前的节点只须将与其单个节点故障的检查点队列开头的条目相对应的脏数据项转移到另一个节点。即使是在接收脏数据项的节点从不请求数据项的情况下，也可以为此目的而执行脏数据项的转移。

这两个检查点可以使用各种方法来加以实现，并且本发明并不局限于任何特定实施。举例来说，单独故障的检查点队列以及多重故障的检查点队列可以作为两个完全独立的队列来加以保持。作为选择，可以保持条目的单独“组合”队列，以便适合单独故障的检查点队列和多重故障的检查点队列。在使用一个组合队列的时候，在组合队列内部，可以使用一个指针来识别哪个条目位于单独故障的检查点队列开头。在从多重故障的检查点队列中删除条目的时候，这些条目是从组合队列中删除的。在从单个故障的检查点队列中删除条目的时候，它们是据此标记的，但是这些条目并没有从组合队列中删除。

基于二进制文件的分批处理

根据基于二进制文件的分批处理方法，在一个节点中保持了两个独立的检查点队列：一个全局变脏的检查点队列和一个本地变脏的检查点队列。节点的本地变脏的检查点队列包含了对应于那些仅仅在该节点中变脏的数据项的条目。节点的全局变脏检查点队列包含了对应于那些在其它节点中也已变脏的数据项的条目。

根据一个实施例，在全局变脏的检查点队列中的条目合并为“二进制文件”。每个二进制文件都与一定的时间范围相关联，并且包含了关于数据项版本的条目，其中这些数据项首先在这个时间范围内变脏。因此，如果合并条目对应于数据项在时间 T7、T9 和 T12 变脏时产生的数据项版本，则合并条目将会落入与包含 T7 的时间范围相对应的二进制文件，这是因为 T7 是由该条目覆盖的“首次变脏时间”。

举例来说，图 8 描述了节点 X 的全局变脏的检查点队列 **800**，该队列分成了二进制文件 **812**、**814** 以及 **816**。二进制文件 **812** 与时间范围 T15 到 T25 相关联并且包含了与那些在 T15 与 T25 之间具有首次变脏时间的全局变脏数据项相对应的条目。二进制文件 **814** 与时间范围 T16 到 T35 相关联并且包含了与那些在 T16 与 T35 之间具有首次变脏时间的全局变脏的数据项相对应的条目。二进制文件 **816** 与时间范围 T36 到 T45 相关联并且包含了与那些在 T36 与 T45 之间具有首次变脏时间的全局性脏数据项相对应的条目。

根据一个实施例，每个二进制文件都指派了一个版本号。举例来说，二进制文件的版本号可以是二进制文件中任何一个条目的首次变脏时间值。举例来说，二进制文件 **812** 包含条目 **805**、**806** 以及 **807**，这三个条目分别与数据项 1 的 V1、数据项 5 的 V1 以及数据项 8 的 V3 相关联。假设数据项 1 的 V1、数据项 5 的 V1 以及数据项 8 的 V3 分别是在时间 T17、T19 以及 T23 首次变脏的。在这

种情况中，T23 是二进制文件 **812** 中任何一个 PI 的最高的首次变脏时间。因此将会把版本号 T23 指派给二进制文件 **812**。

根据一个实施例，通过使永久性存储器的写入子系统基于逐个二进制文件而不是基于逐个条目来向主管发布写入请求，由此减少了写入请求消息的数目。举例来说，为使检查点队列 **800** 提前，节点 X 向主管发送一个单独的写入请求消息，以便写入那些与二进制文件 **812** 中的所有条目相对应的数据项。写入请求消息只须借助版本号 T23（而不是二进制文件内部的特殊条目）即可识别二进制文件 **812**。响应于写入请求，主管将写入执行消息发送到全部数据项的当前锁持有者，其中对于具有一个 PI 的这些数据项来说，它们的首次变脏时间小于或等于写入请求中指定的版本号。在当前实例中，主管将写入执行消息发送到所有数据项的当前锁持有者，对于具有一个 PI 的这些数据项来说，它们的首次变脏时间小于或等于 T23。

当每个节点完成了将最早在 T23 或者 T23 之前发生变化的所有脏数据项写入磁盘时，该节点向主管发送一个写入确认信息。当主管从写入执行消息所发至的所有节点接收到写入确认信息时，主管会向所有节点发送写入通知消息，以便向其告知所请求的写入已经完成。作为响应，每个节点可以清空对应的二进制文件。举例来说，当节点 X 被告知首次变脏时间在 T23 或 T23 之前的全部数据项已经写入磁盘时，该节点 X 可以清空二进制文件 **812**。清空二进制文件 **812** 可以通过（1）丢弃所有那些没有覆盖掉 T23 之后所作改变的条目，以及（2）将覆盖 T23 之后所作改变的二进制文件 **812** 内部的那些条目移动到其他二进制文件。举例来说，如果条目 **806** 是一个覆盖了 T19 与 T40 时所作改变的合并条目，则在清空二进制文件 **812** 的时候，条目 **806** 移动到了二进制文件 **814**。

根据一个实施例，主管追踪（1）PI的首次变脏时间，以及（2）与PI的最后改变（“最后变脏时间”）相关联的版本号。举例来说，对于合并条目 **702** 而言，主管知道合并条目对应于版本 V8（合并条目中的最新版本）以及版本V1（合并条目中的最早版本）。在这种实施例中，当节点从主管那里接收到一个具有二进制文件版本号的写入通知时，它会（1）丢弃最后变脏时间小于或等于二进制文件版本号的二进制文件中的所有条目，以及（2）将最后变脏时间大于二进制文件版本号的二进制文件中的所有条目移动到队列中的下一个二进制文件，由此清空二进制文件。在这个方案中，当存在一次部分覆盖了PI中所包含变化的写入时，由于与最终得到的合并PI相对应的条目很容易移动到它的恰当二进制文件在旧的PI二进制文件中，当因为脏数据项转移到另一个节点而创建一个新 PI时，新PI的条目总是可以替换掉若有的较旧PI的条目。

基于二进制文件的分批处理通常更适于那些使用全局主管而不是分布式锁管理器的多节点系统。发送到当前锁持有者的消息易于进行批量处理，因为它们是同时产生的。实质上，主管还为全局变脏的数据项追踪永久性存储器的版本号，而不是追踪永久性存储器上的数据项版本号以及那些处于写入过程中的数据项的版本号，这非常类似于检验点记录对那些涉及节点中所有脏数据项的变化进行追踪。

恢复

对在多节点系统中执行的写入磁盘操作进行追踪，这一点是非常重要的。举例来说，对于判定哪个条目可以从检查点队列中删除以及确定是否数据项的过去镜像可以写入磁盘和/或从缓存区中释放（“刷新”）来说，这种信息是非常重要的。具体地说，如果已经将数据项的新近版本写入磁盘，则决不应该将数据项的先前一个版

本写入磁盘。此外，在将数据项的更近版本写入磁盘的时候，可以从缓存区刷新数据项的 PI 版本。

在某种情况下，是否成功执行了写入磁盘操作有可能是不清楚的。举例来说，如果将数据项写入磁盘的节点在写入操作过程中出现了故障，则是否在故障发生成功完成了写入操作的前后有可能是不清楚的。同样，如果某个数据项的主管驻留的节点发生故障，则这个故障可能会导致与数据项有关的信息损失。这种信息可能包含了用于指示写入到磁盘的数据项的最近版本的信息。

在发生了一种不清楚是否成功执行了写入磁盘操作的情况时，可以通过扫描磁盘上的数据项来确定其版本，由此可能解决这个问题。然而，作为恢复操作一部分的扫描磁盘将会耗费大量时间和资源，并且有可能过度延迟数据的有效性。

根据本发明的一个方面，可以通过以下手段来免除扫描磁盘上的数据项的需要：（1）如果不清楚是否已将数据项的某个特定版本写入磁盘并且恢复信息（例如重做日志）指示已经将这个特定版本写入了磁盘，则引起恢复处理去假设特定数据项已经成功写入了磁盘，以及（2）将较早缓存的所有数据项版本都标记为“怀疑”。在恢复操作之后，系统可以借助相反的假设来继续进行。具体地说，系统是基于数据项特定版本没有写入磁盘这个假设而继续进行的。然而，在将数据项的任何怀疑版本写入磁盘之前，该系统会读取驻留在磁盘上的数据项版本。如果数据项的磁盘版本更近，则不会执行写磁盘操作，并且主管将被告知磁盘上的是哪个版本。可选地，主管然后将写入通知消息发送到保持了由磁盘上的版本所覆盖的版本的全部节点。另一方面，数据项被恢复。

同样，当一个节点请求数据项的当前版本时，不能向该请求节点提供数据项的怀疑版本，这是因为磁盘中可能包含了数据项的更近版本。取而代之的是，从磁盘中读取数据项的磁盘版本。如果从

磁盘中读取的数据项版本是最近版本，则该版本将会提供给请求节点。如果数据项的磁盘版本不是最近版本，则将会根据出现故障的节点的重做日志所保持的恢复信息来创建最近的版本。

不保留过去镜像而对检查点进行管理

在以上给出的许多情况中，假设每个节点都被配置成保留一个PI，直到该PI受到写入磁盘操作所覆盖。然而，根据本发明的一个实施例，并没有保留这种PI。

具体地说，每个节点保持一个全局变脏的检查点队列以及一个本地变脏的检查点队列。与本地变脏的检查点队列中的条目相关联的脏数据项将会保留，直到它由写入磁盘操作所覆盖。然而，与全局变脏的检查点队列相关联的PI无需以这种方式加以保留。

在这个实施例中，如上所述，执行写入磁盘操作的权限受到数据项上保持的封锁模式的束缚。具体地说，如果（1）节点持有数据项的排他锁，或者（2）不存在持有数据项的排他锁的节点，并且一个节点是保持排他锁的最新节点，那么该节点有权为一个数据项执行写入磁盘的操作。

由于一个节点将会具有本地变脏的所有数据项的排他锁，因此该节点能将那些与本地变脏的队列相关联的数据项写入磁盘，而不需要主管介入。该节点还可以具有一个数据项的排他锁或者已拥有最近的排他锁，其中这个数据项与全局变脏的队列中的条目相关联，并且该节点由此可以将这个数据项写入磁盘，而不需要主管介入。

由于从缓存区中侦测出脏数据项时，节点并未保持一个PI，因此需要专门的恢复处理。具体地说，在数据项转移过程中或是因为节点故障而使数据项的当前版本丢失时，该系统把来自所有节点并

经过合并的重做日志中的变化应用于永久性存储器上的数据项，以便重新产生数据项的当前版本。在每个重做日志内部，开始恢复处理的位置是通过一个与该节点相关联的检查点而得到确定的。除非数据项版本中包含了永久性存储器的检查点之前在节点中进行的变化，否则不能认为节点中的检查点完结。因此，在将一个脏数据项被侦测出发送到另一个节点，而不是在任何检查点队列中都保持数据项过去镜像的时候，数据项本身可以丢弃，并且数据项头部或控制块链接到全局变脏的队列。

全局变脏的队列是借助与条目相关联的首次变脏时间来进行排序的，并且它与本地变脏的队列相似，只不过没有为每个条目保持相关的真实数据项（也就是说，节点缓存区中并没有数据项内容）。节点中的检查点低于出于本地变脏队列开头的条目的首次变脏时间以及处于全局变脏队列开头的条目的首次变脏时间。

当一个节点想要将其检查点提前时，该节点可以在没有主管介入的情况下写入本地变脏队列中的数据项（因为绝不存在两个节点同时写入同一数据项的可能性）或是将一个写入请求发送到主管，以便在拥有者的节点写出数据项，其中该数据项对应于全局变脏队列中数据项头部的一个更近的版本。

根据一个替换实施例，在每个节点中保存了两个检验点记录（每一个都对应于每个队列）。第一检验点记录指示的是时间 TX，其中在节点缓存区中，在 TX 之前对当前变脏的数据项所做的全部改变都记录在永久性存储器的数据项版本上。第二检验点记录包括数据项列表以及在这个节点中进行的首次变化的版本号，在这个节点中，这个版本号曾经变脏，但是此后已被侦测出，并且并未写入永久性存储器。一旦侦测出脏数据项，那么缓存区将会无法追踪脏数据项，但是仍旧在主管之中保持将封锁开启（也就是说，在出现一个写入通知之前，锁定是不会关闭的）。

在出现节点故障时，扫描出现故障的节点上的重做日志的起始位置是通过确定（1）第一检查点记录所确定的日志中的位置（称之为本地检查点记录）以及（2）第二检查点记录中由数据项列表进行的最早变化所确定的日志中的位置（它可以认为是全局检查点记录的特定节点部分）中的较少的一方来进行计算。

在恢复过程中，对于全局检查点记录到节点的本地检查点记录之间的日志部分的可能恢复而言（假设全局检查点记录是在本地检验点记录之后），只有那些对应于全局检查点记录中的数据项的日志记录需要被考虑。一旦到达了本地检验点记录，那么在到达日志末尾之前，需要为可能的恢复而对所有日志记录加以考虑。

这个方案优于现有方法，因为它把第二检验点记录中的数据项列表限制到了仅仅那些先前在这个节点已经变脏的数据项（与整个系统中的所有脏数据项相反）。第二，可以独立于其他节点来写入每个节点的全局检查点记录（也就是说，不需要协调全局主管或是GLM检查点）。最终，在恢复过程中需要扫描的各个节点的重做日志部分总是比较短的，因为不需要从整个系统中最早的未写入变化开始扫描每个节点的重做日志。

此外，在具有全局缓存区的情况下，现有永久性存储器写入协议假设对一个同步的全局时钟进行访问，其中将来自时钟的数值用作日志序列码（LSN）。这里给出的技术方法无需访问一个同步的全局时钟。此外，现有技术需要一个保持封锁一致性的全局主管（GLM）以及群集中的脏数据项的恢复序列号。此外，现有技术无法轻易扩展到那些主管分布在几个节点的系统（DLM）。

硬件综述

图9是描述可以执行本发明一个实施例的计算机系统**900**的数据项框图。计算机系统**900**包括一条总线**902**或是用于传递信息的

其他通信机制，并且包括一个与总线902耦合并用于处理信息的处理器904。计算机系统900还包含一个主存储器906，例如随机访问存储器（RAM）或是其它动态存储设备，这个存储设备与总线902耦合，用于保存信息以及处理器904所要执行的指令。在运行处理器904所执行指令的过程中，主存储器906还可用于保存临时变量或是其它中间信息。计算机系统900还包括一个只读存储器（ROM）908或其它静态存储设备，它与总线902耦合，用于保存静态信息和涉及处理器904的指令。此外还提供了诸如磁盘或光盘这种存储设备910，它与总线902耦合，用于保存信息和指令。

计算机系统900可以经由总线902而与阴极射线管（CRT）这类显示器912相连，从而将信息显示给计算机用户。包括字母数字及其他按键的输入设备914与总线902相连，以便将信息和命令选择传达到处理器904。另一种用户输入设备是光标控制916，例如鼠标、轨迹球或光标方向键，用于将方向信息和命令选择传递给处理器904以及控制显示器912上的光标移动。这种输入设备通常在第一轴（例如x）和第二轴（例如y）这两个轴上具有两个自由度，由此设备能够确定一个平面上的位置。

本发明涉及使用计算机系统900来执行这里所描述的技术方法。根据本发明的一个实施例，处理器904执行主存储器906中包含的一个或多个指令的一个或多个序列，计算机系统900对此做出响应，由此执行这些技术方法。这些指令可以从诸如存储设备910等等的另一种计算机可读介质读入主存储器906。通过执行主存储器906中包含的指令序列，处理器904执行这里描述的处理步骤。在替换实施例中，硬布线电路可用于取代软件指令或是与之组合，由此实现本发明。因此，本发明的实施例并不局限于硬件电路和软件的任何一种特定组合。

这里使用的术语“计算机可读介质”是指任何一种参与向处理器 **904** 提供指令以供执行的介质。这种介质可以采取很多形式，其中包括但不局限于：非易失介质、易失介质和传输介质。举例来说，非易失介质包括光盘或磁盘，例如存储设备 **910**。易失介质包括动态存储器，例如主存储器 **906**。传输介质包括同轴电缆、铜线和光纤，其中包括了构成总线 **902** 的线路。传输介质还可以采取声波或光波的形式，例如无线电波和红外数据通信中产生的信号。

举例来说，计算机可读介质的通用形式包括：软盘、软磁盘、硬盘、磁带或任何其它磁介质、CD-ROM 或任何其它光学介质、穿孔卡、纸带纸条或具有孔洞图案的任何其它物理介质、RAM、PROM 和 EPROM、FLASH-EPROM、其它任何存储芯片或盒式磁带机、如下所述的载波或是计算机可以读取的其它任何介质。

不同形式的计算机可读介质可以涉及向处理器 **904** 传递用于实施的一个或多个指令的一个或多个序列。举例来说，最初可以在远程计算机磁盘上携带指令。远程计算机可以将指令加载到它的动态存储器中，并且使用调制解调器经由电话线来发送指令。计算机系统 **900** 本地的调制解调器可以在电话线上接收数据并使用红外发射机来将数据转换成红外信号。红外检测器可以接收红外信号中携带的数据，而恰当的电路则可将数据安插到总线 **902** 上。总线 **902** 将数据传送到主存储器 **906**，处理器 **904** 从主存储器 **906** 中检索并执行指令。在由处理器 **904** 执行之前或之后，主存储器 **906** 接收的指令可以随意保存在存储设备 **910** 中。

计算机系统 **900** 还包括一个与总线 **902** 相连的通信接口 **918**。通信接口 **918** 提供了一个与网络链路 **920** 耦合的双向数据通信，其中网络链路 **920** 与本地网络 **922** 相连。举例来说，通信接口 **918** 可以是一个向对应类型的电话线路提供数据通信连接的综合业务数字网（ISDN）网卡或调制解调器。作为另一个实例，通信接口 **918**

可以是一个LAN网卡，它向兼容的LAN提供数据通信连接。此外还可以实施无线链路。在任何一种这类实施中，通信接口**918**都会收发电、电磁或光信号，这些信号传送的是那些代表不同类型信息的数字数据流。

网络链路**920**通常经由一个或多个网络来向其它数据设备提供数据通信。举例来说，网络链路**920**可以经由本地网络**922**而将一个连接提供给计算机主机**924**或是互联网服务供应商（ISP）**926**运作的数据设备。ISP **926** 进而又经由现在通常称为“互联网”的全球分组数据通信网络**928**来提供数据通信业务。本地网络**922**和互联网**928**都使用了传送数字数据流的电、电磁或光信号。经由不同网络的信号以及网络链路**920**上经由通信接口**918**的信号传送的是那些往返于计算机系统**900**的数字数据，而这些信号即为传送信息的载波的示范性形式。

计算机系统**900**可以经由一个或多个网络、网络链路**920**以及通信接口**918**来发送消息和接收数据，其中包括了程序代码。在互联网实例中，服务器**930**可以经由互联网**928**、ISP**926**、本地网络**922**以及通信接口**918**来发送一个用于应用程序的被请求码。

接收到的代码可以在接收时由处理器 **904** 执行和/或存入存储设备**910**或其它非易失存储器以供稍后执行。这样，计算机系统**900**可以得到载波形式的应用码。在前述说明中，本发明是参考其特定实施例而被描述的。然而很明显，可以对本发明进行各种修改和变化，而不脱离本发明较宽的实质和范围。因此，说明书以及附图仅仅被看作是说明性的，它们并不具有限制意义。

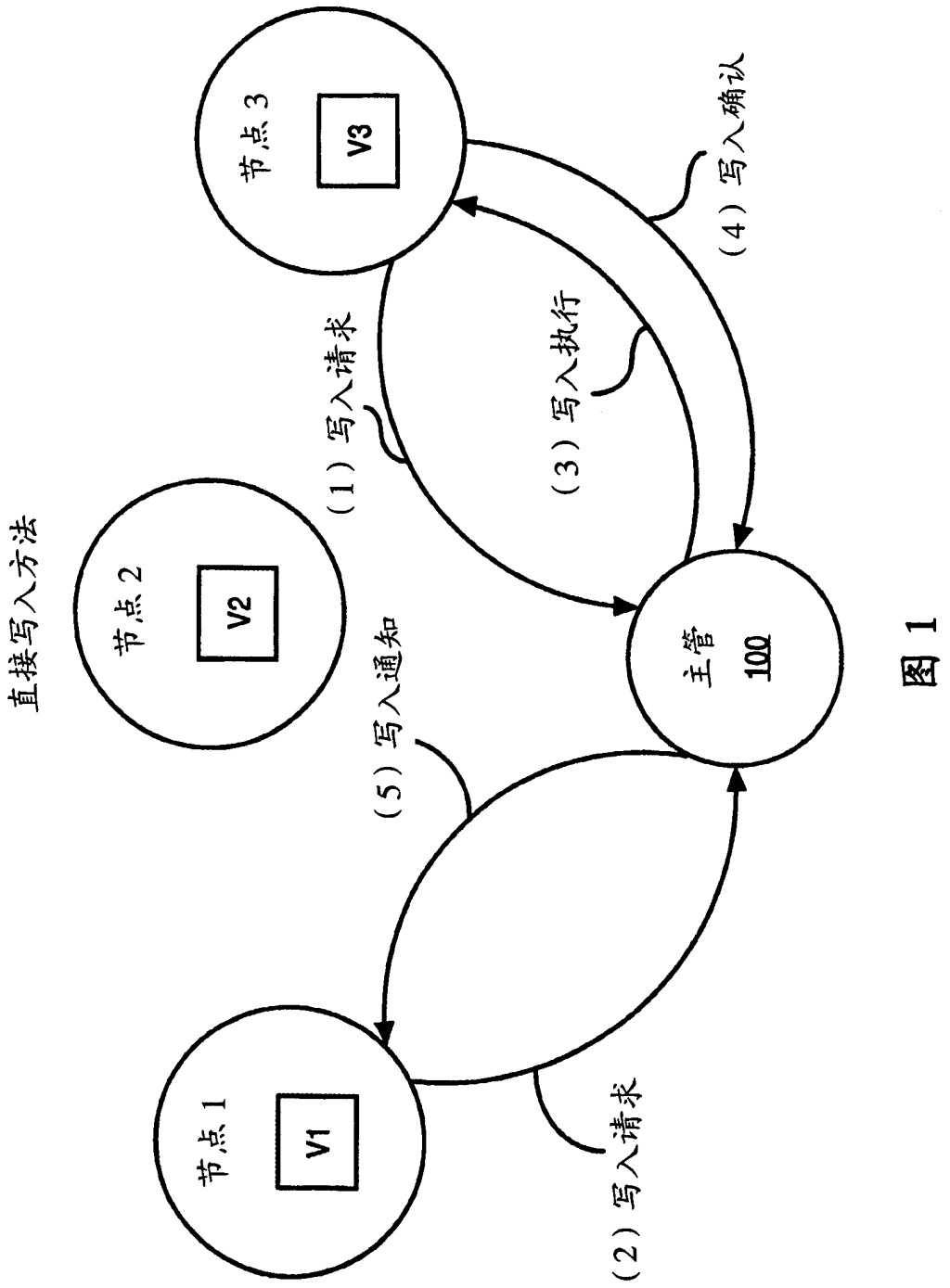

直接写入方法
节点 1
V1
节点 2
V2
节点 3
V3
主管
100
(1) 写入请求
(2) 写入请求
(3) 写入执行
(4) 写入确认
(5) 写入通知

图 1

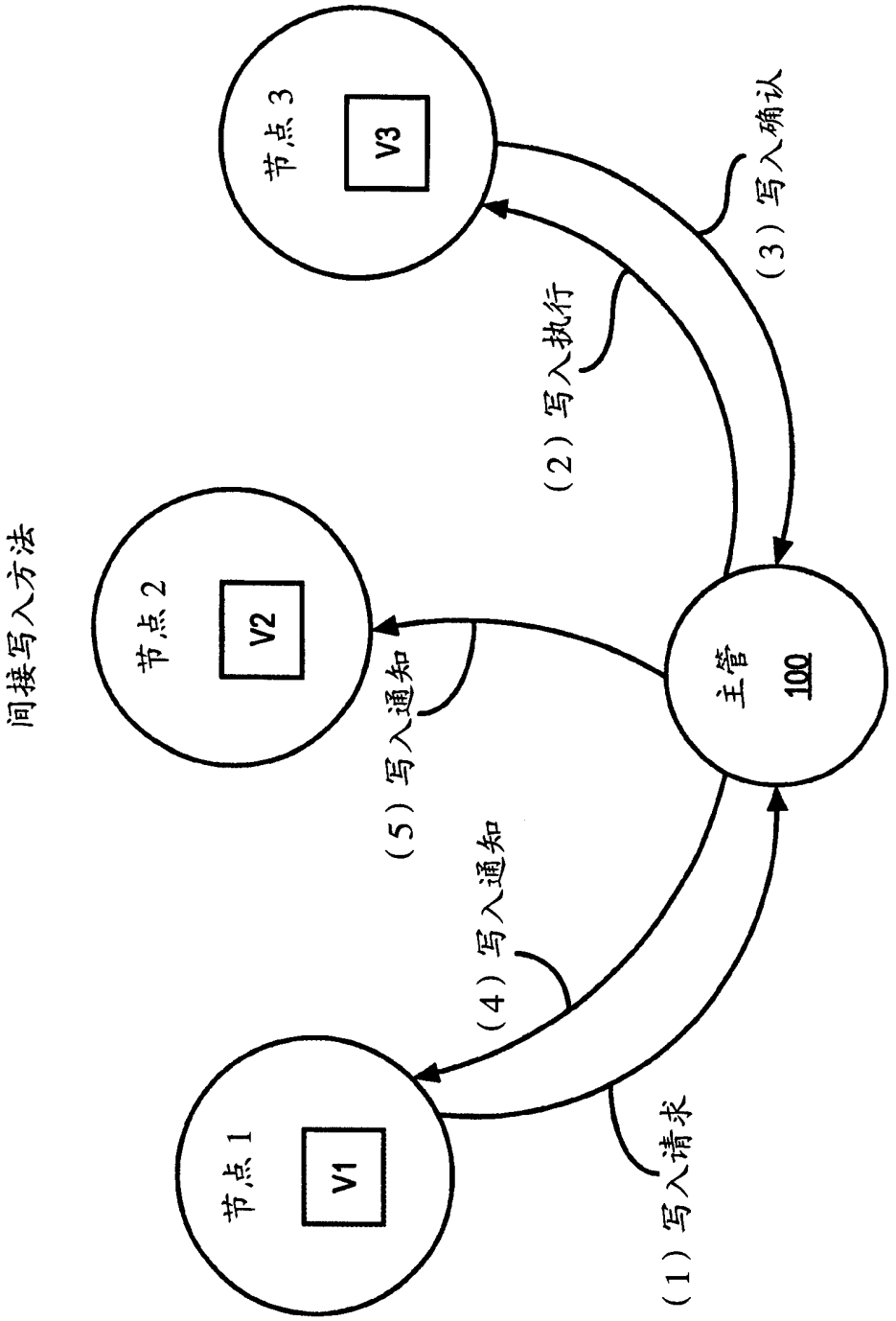

间接写入方法
节点 1
V1
节点 2
V2
节点 3
V3
主管
100
(1) 写入请求
(2) 写入执行
(3) 写入确认
(4) 写入通知
(5) 写入通知

图 2

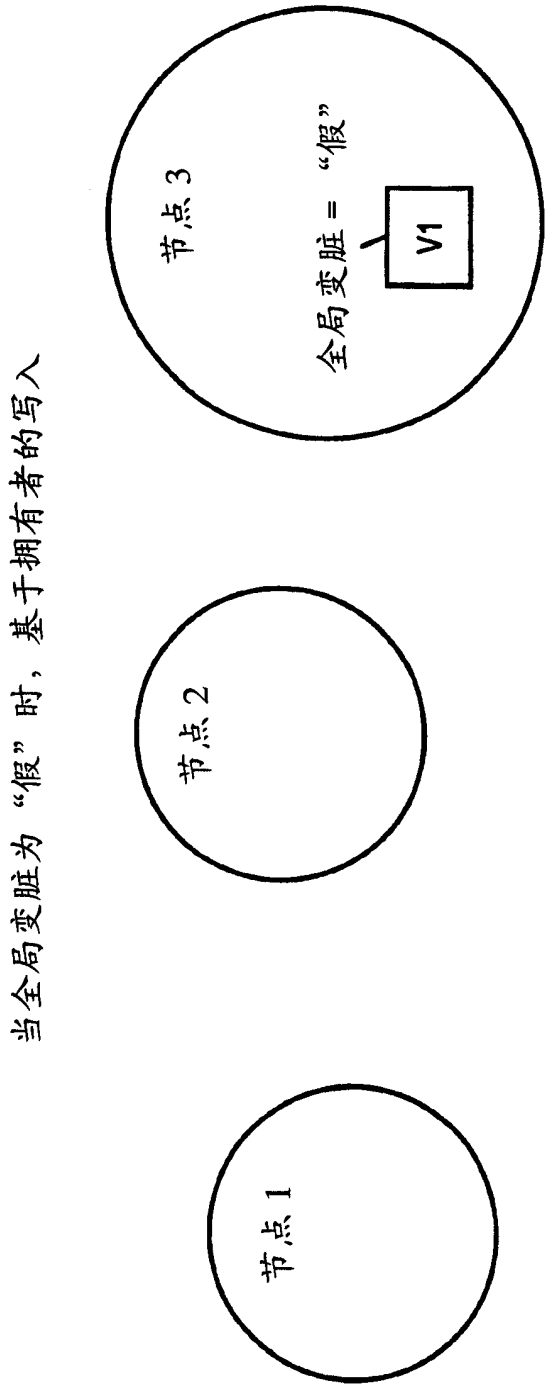


图 3a

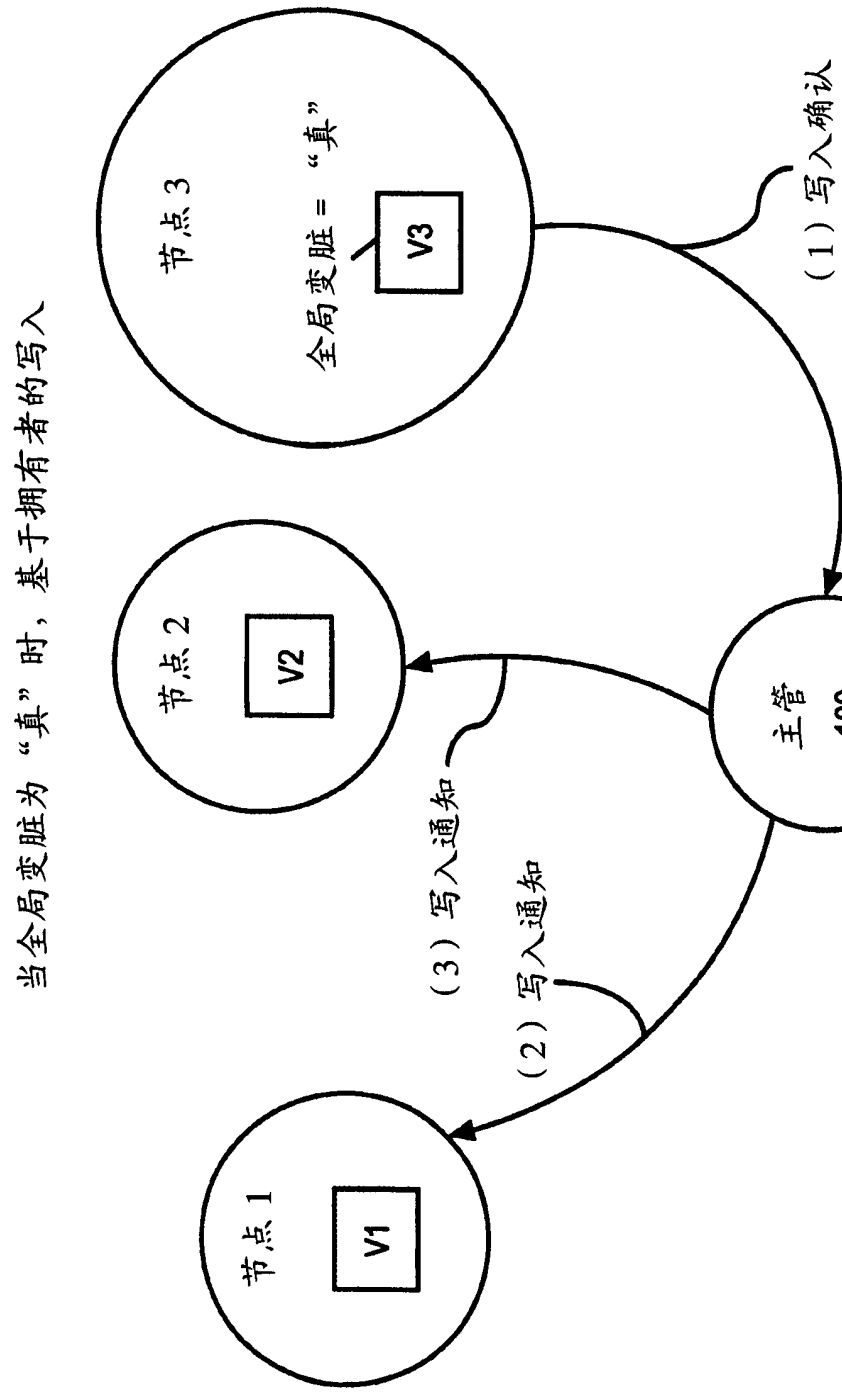

当全局变脏为“真”时，基于拥有者的写入
节点 3
全局变脏 = “真”
V3
节点 2
V2
节点 1
V1
主管
100
(1) 写入确认
(2) 写入通知
(3) 写入通知

图 3b

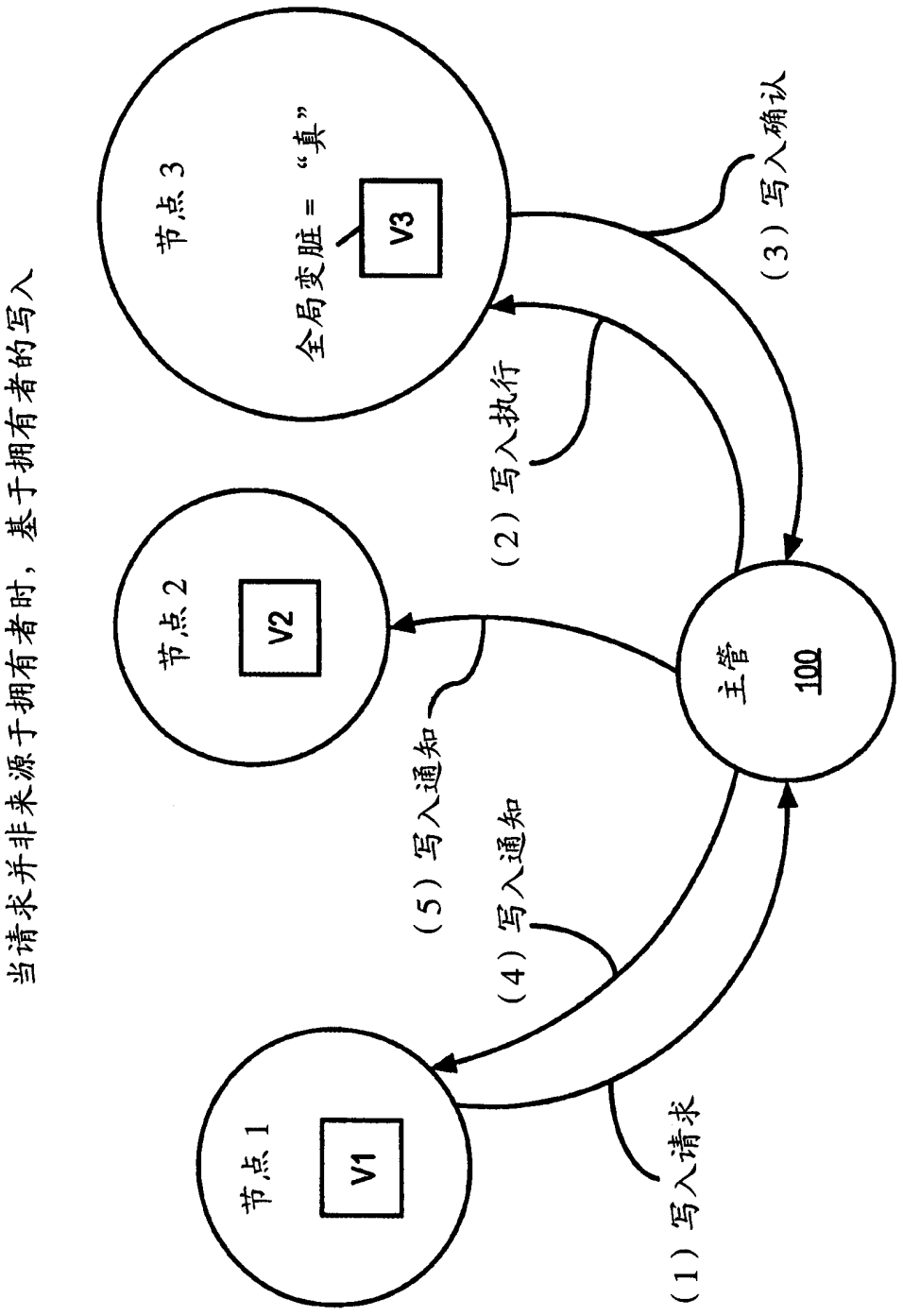

当请求并非来源于拥有者时，基于拥有者的写入
节点 1
V1
节点 2
V2
节点 3
全局变脏 = “真”
V3
主管
100
（1）写入请求
（2）写入执行
（3）写入确认
（4）写入通知
（5）写入通知

图 3c

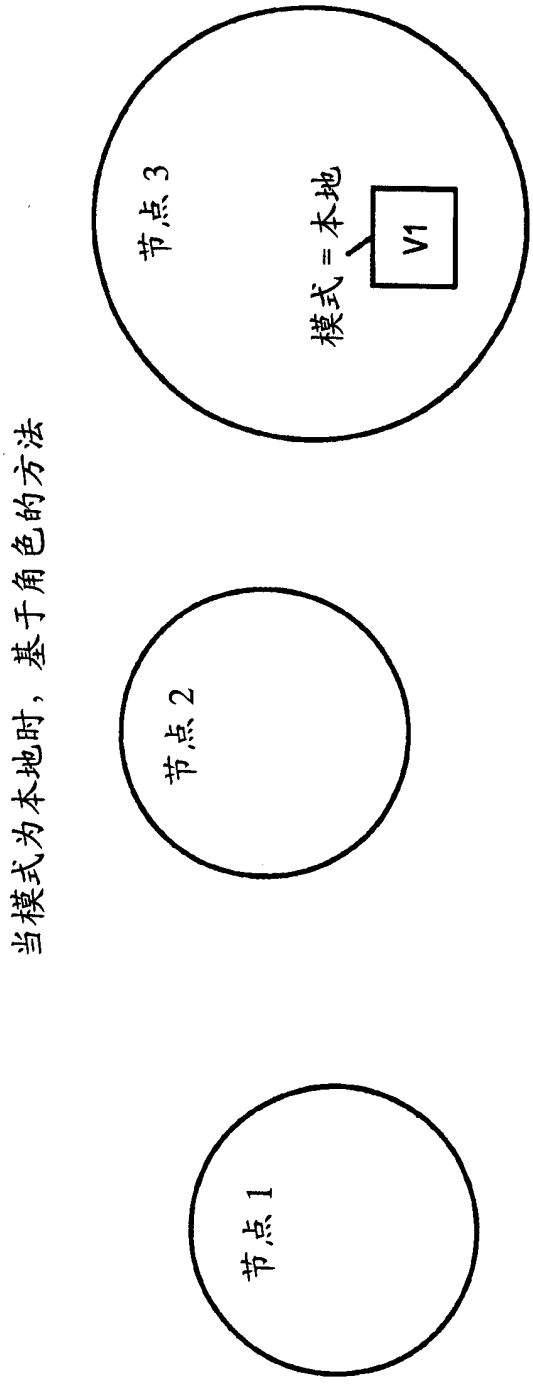


图 4a

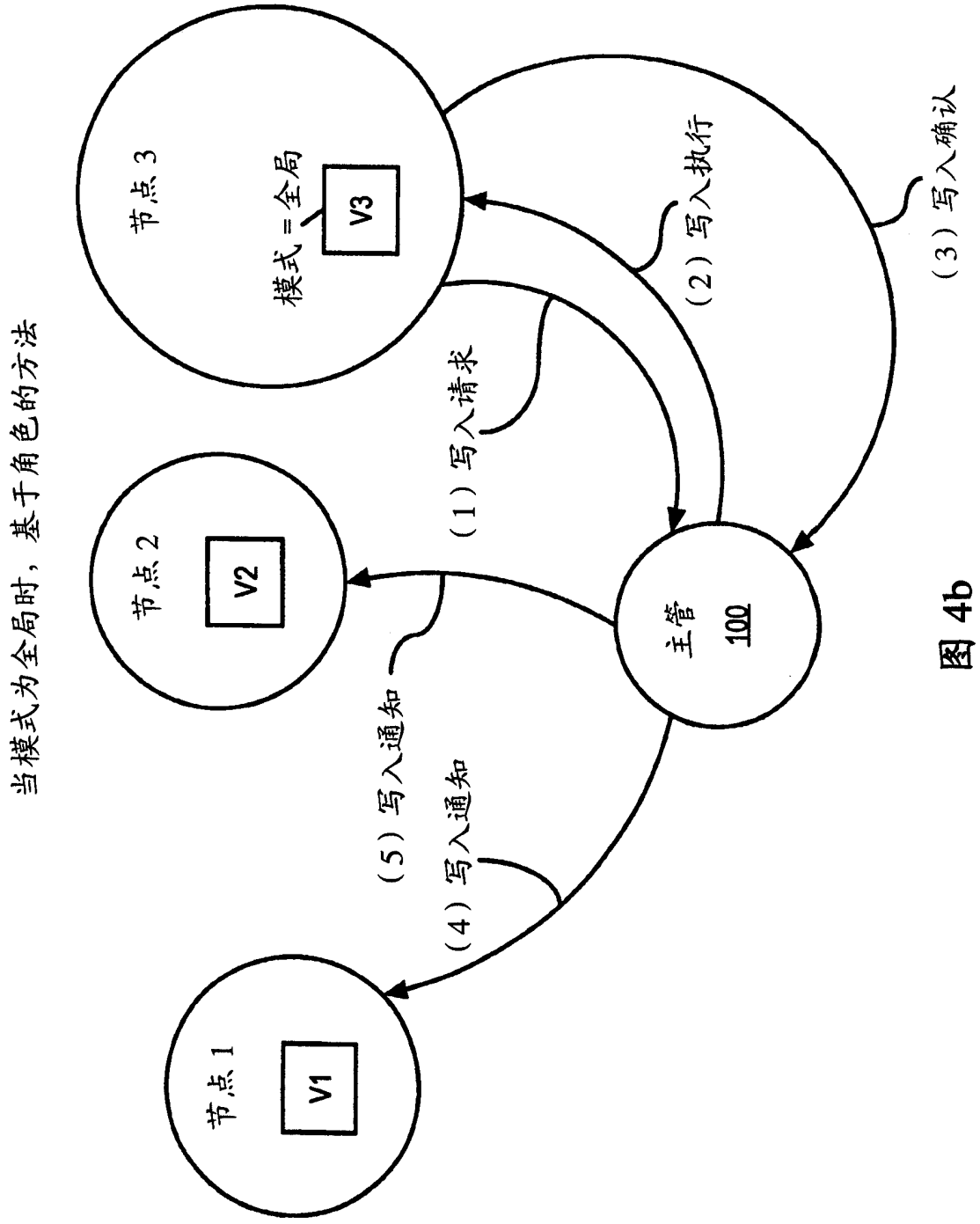

当模式为全局时，基于角色的方法
节点 1
V1
节点 2
V2
节点 3
模式 = 全局
V3
主管
100
(1) 写入请求
(2) 写入执行
(3) 写入确认
(4) 写入通知
(5) 写入通知

图 4b

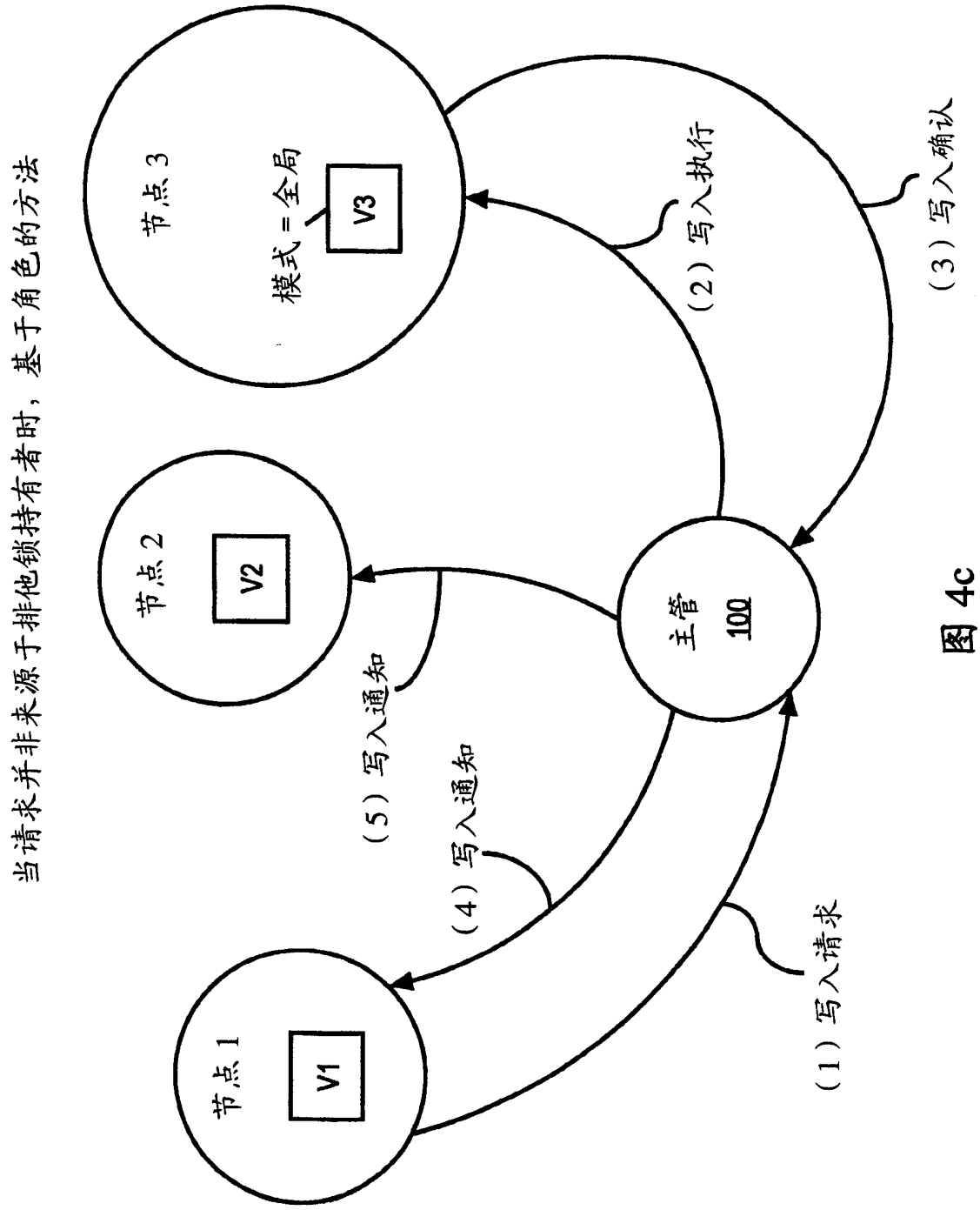

当请求并非来源于排他锁持有者时，基于角色的方法
节点 1
V1
节点 2
V2
节点 3
模式 = 全局
V3
主管
100
(1) 写入请求
(2) 写入执行
(3) 写入确认
(4) 写入通知
(5) 写入通知

图 4c

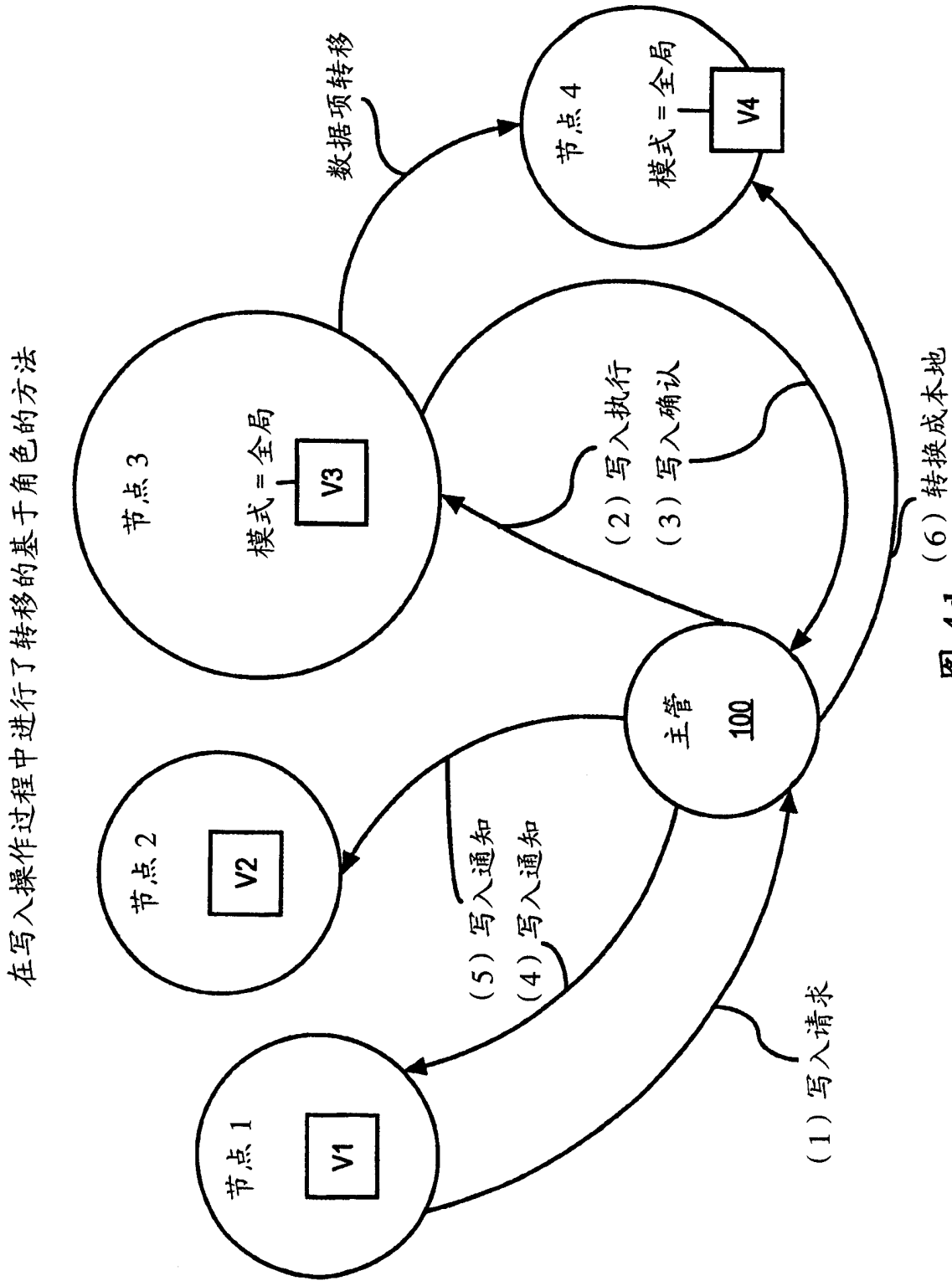

在写入操作过程中进行了转移的基于角色的方法
节点 1
V1
节点 2
V2
节点 3
模式 = 全局
V3
节点 4
模式 = 全局
V4
数据项转移
主管
100
(1) 写入请求
(2) 写入执行
(3) 写入确认
(4) 写入通知
(5) 写入通知
(6) 转换成本地

图 4d

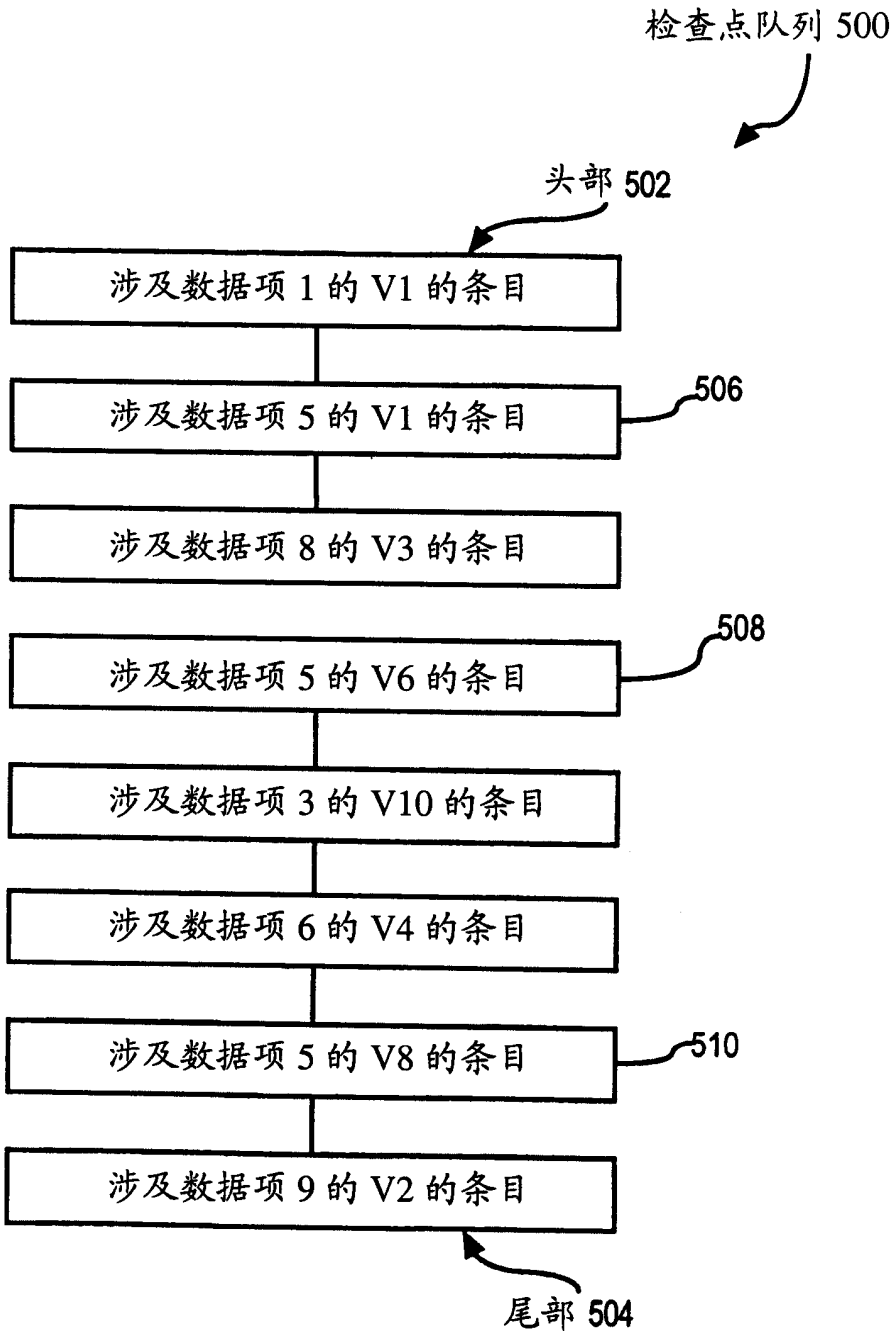

检查点队列 500
头部 502
涉及数据项 1 的 V1 的条目
涉及数据项 5 的 V1 的条目
506
涉及数据项 8 的 V3 的条目
涉及数据项 5 的 V6 的条目
508
涉及数据项 3 的 V10 的条目
涉及数据项 6 的 V4 的条目
涉及数据项 5 的 V8 的条目
510
涉及数据项 9 的 V2 的条目
尾部 504

图 5

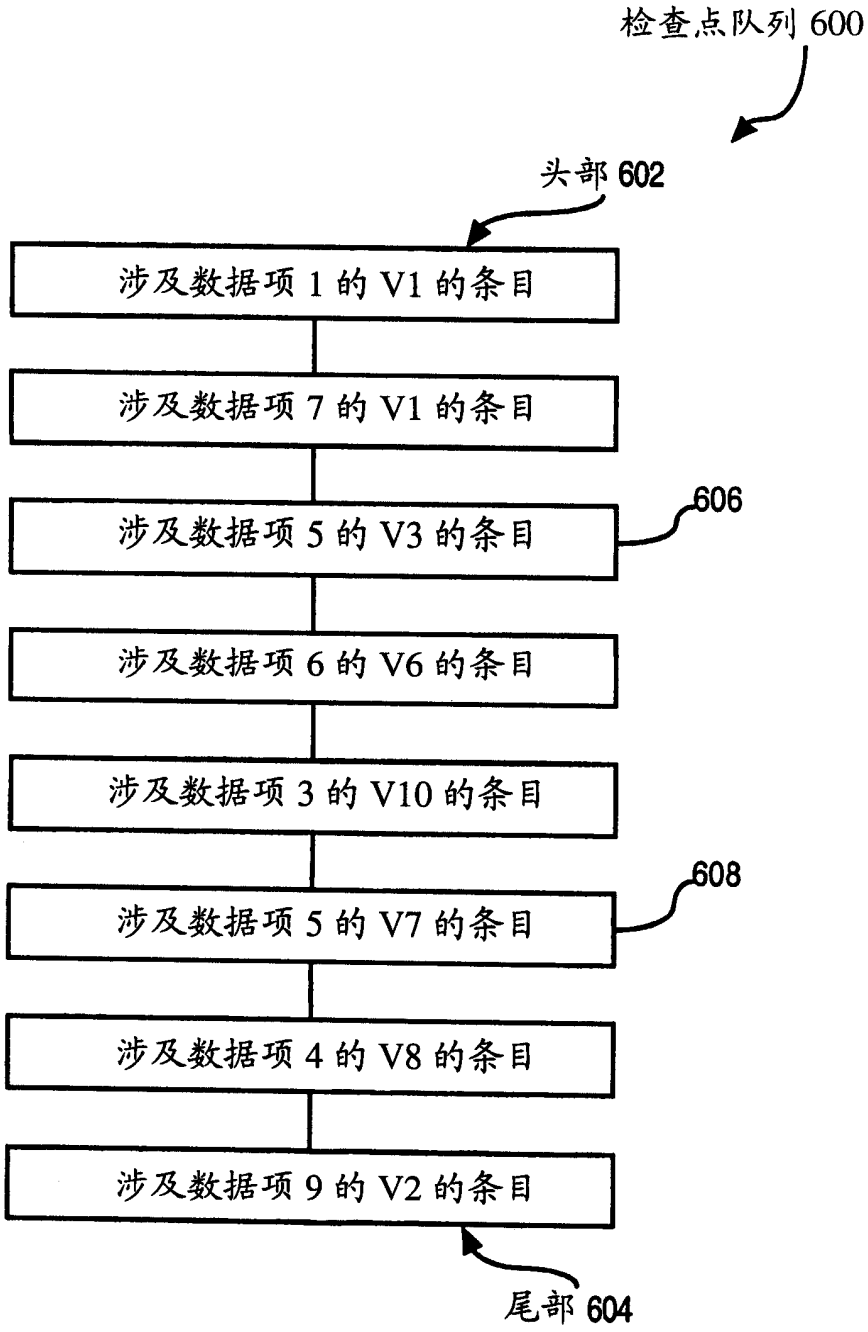

检查点队列 600
头部 602
涉及数据项 1 的 V1 的条目
涉及数据项 7 的 V1 的条目
涉及数据项 5 的 V3 的条目
606
涉及数据项 6 的 V6 的条目
涉及数据项 3 的 V10 的条目
608
涉及数据项 5 的 V7 的条目
涉及数据项 4 的 V8 的条目
涉及数据项 9 的 V2 的条目
尾部 604

图 6

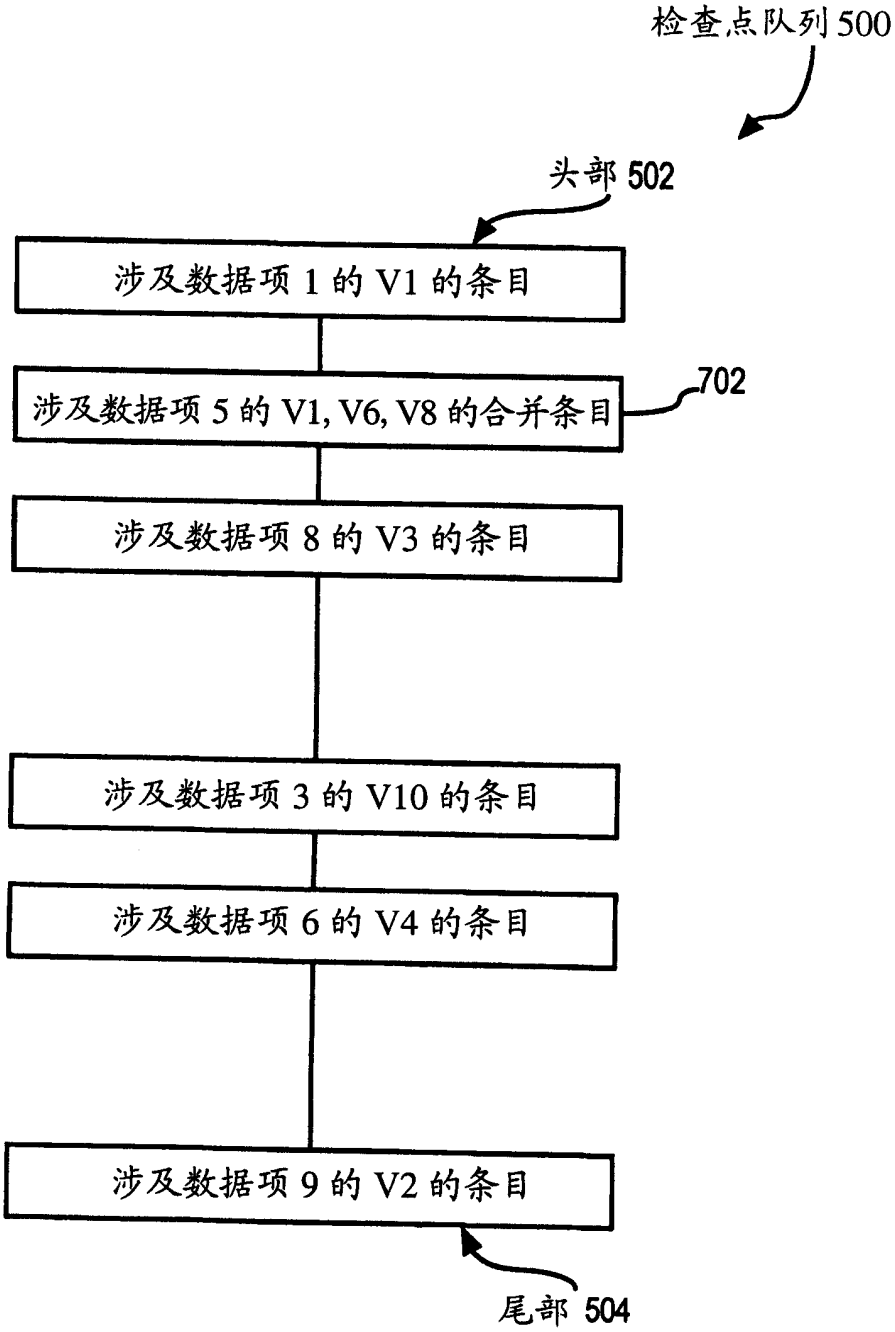

检查点队列 500
头部 502
涉及数据项 1 的 V1 的条目
涉及数据项 5 的 V1, V6, V8 的合并条目
702
涉及数据项 8 的 V3 的条目
涉及数据项 3 的 V10 的条目
涉及数据项 6 的 V4 的条目
涉及数据项 9 的 V2 的条目
尾部 504

图 7

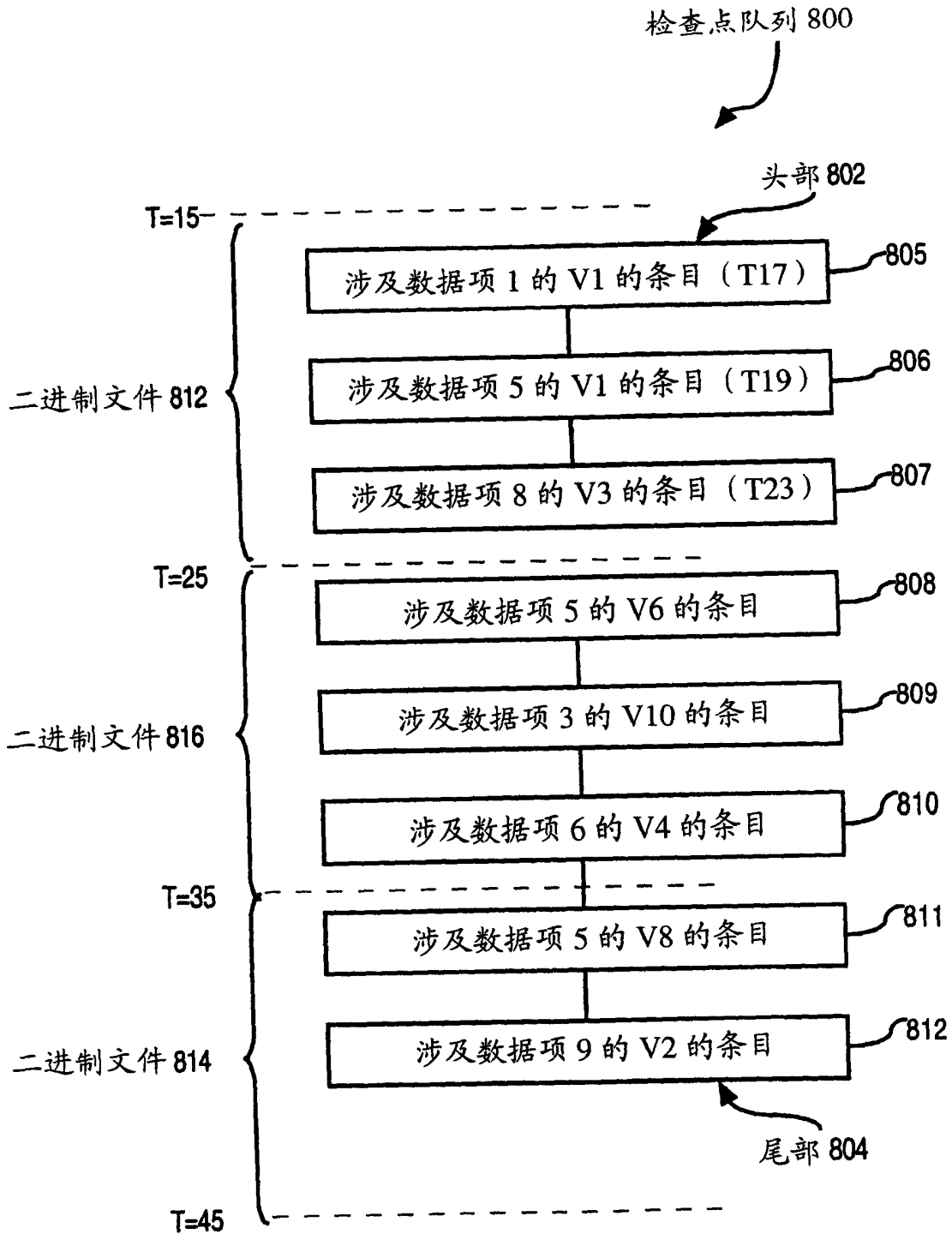

检查点队列 800
头部 802
T=15
涉及数据项 1 的 V1 的条目（T17）
805
涉及数据项 5 的 V1 的条目（T19）
806
二进制文件 812
涉及数据项 8 的 V3 的条目（T23）
807
T=25
涉及数据项 5 的 V6 的条目
808
涉及数据项 3 的 V10 的条目
809
二进制文件 816
涉及数据项 6 的 V4 的条目
810
T=35
涉及数据项 5 的 V8 的条目
811
涉及数据项 9 的 V2 的条目
812
二进制文件 814
尾部 804
T=45

图 8

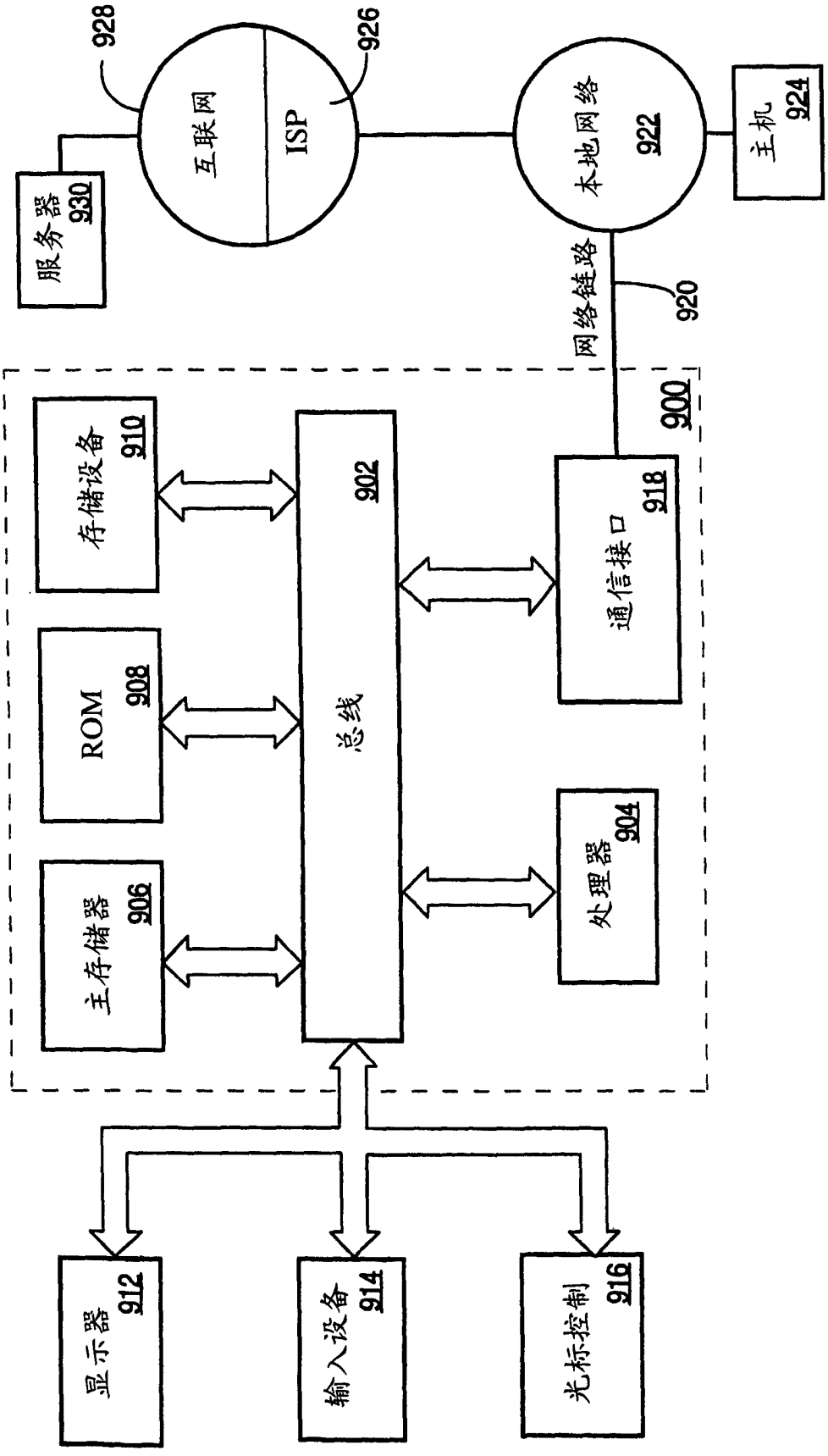

928
互联网
ISP
926
服务器
930
本地网络
922
主机
924
网络链路
920
900
存储设备
910
902
通信接口
918
ROM
908
总线
主存储器
906
处理器
904
显示器
912
输入设备
914
光标控制
916

图 9