[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/46 (2006.01)
G06F 17/30 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200910143961.7

[43] 公开日 2009 年 11 月 11 日

[11] 公开号 CN 101576830A

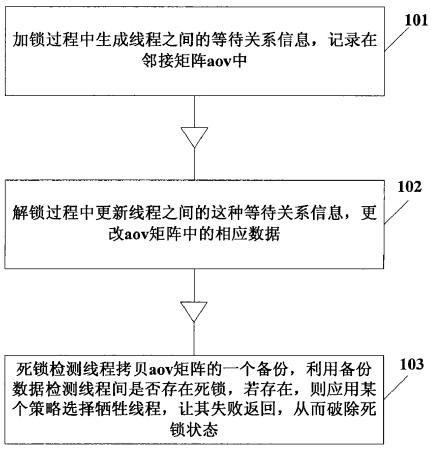
[22] 申请日 2009.6.4
[21] 申请号 200910143961.7
[71] 申请人 中兴通讯股份有限公司
地址 518057 广东省深圳市南山区高新技术产业园科技南路中兴通讯大厦法务部
[72] 发明人 陈河堆 常二鹏 卢勤元
[74] 专利代理机构 信息产业部电子专利中心
代理人 吴永亮
权利要求书 3 页 说明书 14 页 附图 6 页

[54] 发明名称

数据库事务锁机制的死锁检测方法及装置

[57] 摘要

本发明公开了一种数据库事务锁机制的死锁检测方法及装置，预定义一个用于存放线程间等待关系信息的邻接矩阵，所述方法包括：加锁线程将加锁过程中生成的线程间等待关系信息记录在所述邻接矩阵中；解锁线程在解锁过程中根据需要更新所述邻接矩阵中的相应等待关系信息；死锁检测线程利用所述邻接矩阵及图论原理对所述线程进行检测计算，从而判断是否存在死锁；所述装置包括：信息存储模块、死锁检测模块、每个线程中包含的信息记录模块和信息更新模块；本发明所提出的技术方案具有极高的死锁检测速度，而且能够充分利用加解锁过程获得的有用信息来帮助随后的检测死锁，节约了计算资源。



1、一种数据库事务锁机制的死锁检测方法，其特征在于，预定义一个用于存放线程间等待关系信息的邻接矩阵，则所述方法包括:

步骤 A: 加锁线程将加锁过程中生成的线程间等待关系信息记录在所述邻接矩阵中;

步骤 B:解锁线程在解锁过程中根据需要更新所述邻接矩阵中的相应等待关系信息;

步骤 C:死锁检测线程利用所述邻接矩阵及图论原理对所述线程进行检测计算，从而判断是否存在死锁。

2、根据权利要求 1 所述的方法，其特征在于，所述步骤 A 具体包括:

在对某锁对象进行加锁的过程中，当加锁线程要申请的锁类型与该锁对象已授予其他线程的锁类型不相容时，将加锁线程等待其他线程的等待关系信息记录在所述邻接矩阵中;

加锁线程挂起等待相关线程进行解锁。

3、根据权利要求 2 所述的方法，其特征在于，所述步骤 B 具体包括:

步骤 B1:在相关线程进行解锁过程中，进行解锁的线程称为解锁线程; 解锁线程对于所述邻接矩阵中记录的要申请解锁线程正在释放的这把锁的每个线程，判断它所要申请的锁类型是否与该解锁线程拥有的锁类型相容，若不相容，执行步骤 B2; 若相容，则执行步骤 B3;

步骤 B2: 保持等待关系信息不变;

步骤 B3: 判断是否与至少一个其他线程拥有的锁类型不相容，如果是，则进行锁等待关系迁移，更新所述邻接矩阵中的相应等待关系信息，否则仅清除原来的线程间等待关系信息。

4、根据权利要求 1 到 3 中任意一项所述的方法，其特征在于，所述步骤 C 具体包括:

步骤 C1：死锁检测线程拷贝所述邻接矩阵的一个备份;

步骤 C2：利用备份的邻接矩阵中的线程间等待关系信息及图论原理中的拓扑排序方法对所述线程进行检测计算，检测线程间是否存在死锁回路，如果存在，则根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回，从而破除死锁状态。

5、根据权利要求 4 所述的方法，其特征在于，所述步骤 C2 中的具体包括:

步骤 C21：计算所述邻接矩阵的入度数组和出度数组，并判断入度数组是否等于出度数组，如果是，执行步骤 C22，否则执行步骤 C23;

步骤 C22：入度数组是否是 0 向量，如果是，则计算结束；否则根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回;

步骤 C23：清除所有出度不为且 0 入度为 0 的元素，并转到步骤 C21。

6、一种数据库事务锁机制的死锁检测装置，其特征在于，所述装置包括:信息存储模块、死锁检测模块、每个线程中包含的信息记录模块和信息更新模块，其中，

所述信息记录模块，用于将加锁过程中生成的线程间等待关系信息记录到所述信息存储模块中;

所述信息更新模块，用于在解锁过程中根据需要更新所述邻接矩阵中的相应等待关系信息;

所述信息存储模块，用于以预定义邻接矩阵的形式保存线程间等待关系信息;

所述死锁检测模块，用于根据所述信息存储模块存放的线程间等待关系信息及图论原理对线程进行检测计算，从而判断是否存在死锁。

7、根据权利要求 6 所述的装置，其特征在于，所述信息记录模块具体用于，在加锁过程中，当加锁线程要申请的锁类型与该锁对象已授予其他线程的锁类型不相容时，所述信息记录模块将加锁线程等待其他线程的等待关系信息记录在所述信息存储模块中，然后挂起等待相关线程进行解锁。

8、根据权利要求 6 所述的装置，其特征在于，所述信息更新模块具体用于，在相关线程进行解锁过程中，判断所述信息存储模块中记录的要申请这把锁的每个线程，它所要申请的锁类型是否与该解锁线程拥有的锁类型相容，若不相容，则保持等待关系信息不变，若相容，则进一步判断是否与至少一个其他线程拥有的锁类型不相容，如果是，则进行锁等待关系迁移，更新所述邻接矩阵中的相应等待关系信息，否则仅清除原来的线程间等待关系信息。

9、根据权利要求 6 所述的装置，其特征在于，所述死锁检测模块具体用于，拷贝所述信息存储模块中的矩阵数据作为一个备份，利用备份的矩阵数据及图论原理对所述线程进行检测计算，检测线程间是否存在死锁回路，当确认存在死锁回路时，根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回。

数据库事务锁机制的死锁检测方法及装置

技术领域

本发明涉及数据库技术领域，尤其涉及一种数据库事务锁机制的死锁检测方法及装置。

背景技术

事务（Transaction）是数据库管理系统（DBMS）提供的基本也是最重要的功能之一，而事务功能的实现要依赖2个核心基础技术：锁机制和REDO/UNDO日志功能。锁机制是事务实现的核心技术，它关系到事务功能能否实现，系统的性能和吞吐量，以及系统的稳定性。

各种数据库管理系统所采用的锁机制的基本理论是一样的，但是实现方法各有不同。目前主流的数据库管理系统，如Oracle，Sybase，MS SQL Server，MySQL（InnoDB）等，都实现自己的一整套锁机制，各有其优缺点。

死锁检测是锁机制的一个关键技术。尽管在某些特定场景中可以使用一些方法避免和降低死锁发生，但是死锁有时还是难以避免，因此死锁检测成了锁机制中不可或缺的一部分。死锁检测需要消耗不少计算资源，其效率的好坏对系统整体性能有不小的影响。

例如，MS SQL Server 2008的死锁检测方法如下：当锁监视器对特定线程启动死锁搜索时，会标识线程正在等待的资源。然后，锁监视器查找特定资源的所有者，并递归地继续执行对那些线程的死锁搜索，直到找到一个循环。用这种方式标识的循环形成一个死锁。这属于事后检测。

再例如，InnoDB 的死锁检测也有类似的递归算法，即，事务 T1（请求者线程）要对某个对象 obj（表或者记录）加锁，当需要等待时，首先创建一个锁对象 wait_lock，然后事先测试是否发生死锁。它调用一个自递归函数 lock_deadlock_recursive（T1，wait_lock），表示判断在 wait_lock 之前的加锁者（事务）是否等待 T1。具体死锁检测算法如下：1）如果加在对象 obj 上的前一个锁不存在，则没有死锁，退出。2）判断是否 wait_lock 需要等待前一个锁 lock，且 lock 的拥有者 T2 等于 T1，若是，则表示已经发生死锁，退出。3）得到事务 T2 正在等待的锁 wait_lock2，递归调用 lock_deadlock_recursive（T1，wait_lock2）。可见，InnoDB 也是采用递归调用的方式来测试是否出现事务之间的等待环。

MS SQL Server 和 InnoDB 都是采用递归方式来检测死锁，特别是 InnoDB，采用双向链表来组织保护对象已授予的锁对象列表，以及事务拥有的锁对象列表，死锁检测算法遍历这些链表来检测等待环的存在。这种方式检测效率较低，而且一次检测最多只能检测出一个等待环，同时检测过程对正在进行的线程（事务）有不小的妨碍。

综上所述，目前大部分数据库管理系统，不管是事先检测还是事后检测，主要都是利用资源图（已分配资源图和请求资源图）来检测死锁。这种方法的缺点是丢失了加解锁过程中产生的有助于死锁检测的信息，使得死锁检测时需要重新计算这些信息，造成死锁检测繁琐及计算资源的浪费。

发明内容

鉴于上述的分析，本发明旨在提供一种数据库事务锁机制的死锁检测方

法及装置，用以解决现有技术中存在的死锁检测繁琐及计算资源浪费的问题。

本发明的目的主要是通过以下技术方案实现的:

本发明提供了一种数据库事务锁机制的死锁检测方法，预定义一个用于存放线程间等待关系信息的邻接矩阵，则所述方法包括:

步骤 A: 加锁线程将加锁过程中生成的线程间等待关系信息记录在所述邻接矩阵中;

步骤 B:解锁线程在解锁过程中根据需要更新所述邻接矩阵中的相应等待关系信息;

步骤 C:死锁检测线程利用所述邻接矩阵及图论原理对所述线程进行检测计算，从而判断是否存在死锁。

进一步地，所述步骤 A 具体包括:

在对某锁对象进行加锁的过程中，当加锁线程要申请的锁类型与该锁对象已授予其他线程的锁类型不相容时，将加锁线程等待其他线程的等待关系信息记录在所述邻接矩阵中;

加锁线程挂起等待相关线程进行解锁。

进一步地，所述步骤 B 具体包括:

步骤 B1:在相关线程进行解锁过程中，进行解锁的线程称为解锁线程; 解锁线程对于所述邻接矩阵中记录的要申请解锁线程正在释放的这把锁的每个线程，判断它所要申请的锁类型是否与该解锁线程拥有的锁类型相容，若不相容，执行步骤 B2; 若相容，则执行步骤 B3;

步骤 B2: 保持等待关系信息不变;

步骤 B3: 判断是否与至少一个其他线程拥有的锁类型不相容，如果是，则

进行锁等待关系迁移，更新所述邻接矩阵中的相应等待关系信息，否则仅清除原来的线程间等待关系信息。

进一步地，所述步骤 C 具体包括:

步骤 C1：死锁检测线程拷贝所述邻接矩阵的一个备份;

步骤 C2：利用备份的邻接矩阵中的线程间等待关系信息及图论原理中的拓扑排序方法对所述线程进行检测计算，检测线程间是否存在死锁回路，如果存在，则根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回，从而破除死锁状态。

进一步地，所述步骤 C2 中的具体包括:

步骤 C21：计算所述邻接矩阵的入度数组和出度数组，并判断入度数组是否等于出度数组，如果是，执行步骤 C22，否则执行步骤 C23;

步骤 C22：入度数组是否是 0 向量，如果是，则计算结束；否则根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回;

步骤 C23：清除所有出度不为且 0 入度为 0 的元素，并转到步骤 C21。

本发明实施例还提供了一种数据库事务锁机制的死锁检测装置，所述装置包括：信息存储模块、死锁检测模块、每个线程中包含的信息记录模块和信息更新模块，其中，

所述信息记录模块，用于将加锁过程中生成的线程间等待关系信息记录到所述信息存储模块中;

所述信息更新模块，用于在解锁过程中根据需要更新所述邻接矩阵中的相应等待关系信息;

所述信息存储模块，用于以预定义邻接矩阵的形式保存线程间等待关系信

息;

所述死锁检测模块，用于根据所述信息存储模块存放的线程间等待关系信息及图论原理对线程进行检测计算，从而判断是否存在死锁。

进一步地，所述信息记录模块具体用于，

在加锁过程中，当加锁线程要申请的锁类型与该锁对象已授予其他线程的锁类型不相容时，所述信息记录模块将加锁线程等待其他线程的等待关系信息记录在所述信息存储模块中，然后挂起等待相关线程进行解锁。

进一步地，所述信息更新模块具体用于，在相关线程进行解锁过程中，判断所述信息存储模块中记录的要申请这把锁的每个线程，它所要申请的锁类型是否与该解锁线程拥有的锁类型相容，若不相容，则保持等待关系信息不变，若相容，则进一步判断是否与至少一个其他线程拥有的锁类型不相容，如果是，则进行锁等待关系迁移，更新所述邻接矩阵中的相应等待关系信息，否则仅清除原来的线程间等待关系信息。

进一步地，所述死锁检测模块具体用于，拷贝所述信息存储模块中的矩阵数据作为一个备份，利用备份的矩阵数据及图论原理对所述线程进行检测计算，检测线程间是否存在死锁回路，当确认存在死锁回路时，根据预定策略在所述死锁回路中选择一个线程作为牺牲线程，让其失败返回。

本发明有益效果如下:

本发明所提出的技术方案具有极高的死锁检测速度，而且能够充分利用加解锁过程获得的有用信息来帮助随后的检测死锁，节约了计算资源。

本发明的其他特征和优点将在随后的说明书中阐述，并且，部分的从说明书中变得显而易见，或者通过实施本发明而了解。本发明的目的和其他优点可

通过在所写的说明书、权利要求书、以及附图中所特别指出的结构来实现和获得。

附图说明

图 1 为本发明实施例所述死锁检测方法的主体流程示意图;

图 2 为本发明实施例所述方法中，死锁检测信息更新的流程示意图;

图 3 为本发明实施例所述方法中，死锁检测算法的流程示意图;

图 4 为本发明实施例所述方法中，用有向图描述的发生死锁现象的场景示意图;

图 5 为本发明实施例所述方法中，应用图 3 所述算法检测到死锁前邻接矩阵的状态图;

图 6 为本发明实施例所述方法中，应用图 3 所述算法检测到死锁后邻接矩阵的状态图;

图 7 本发明实施例所述装置的结构示意图。

具体实施方式

本发明的主要内容为，线程在加锁和解锁过程中产生和维护死锁检测信息，然后利用死锁检测信息及 aov 网络原理对线程进行拓扑排序，从而判断是否存在死锁。

下面结合附图来具体描述本发明的优先实施例，其中，附图构成本申请一部分，并与本发明的实施例一起用于阐释本发明的原理。

首先结合附图 1 到附图 4 对本发明实施例所述方法进行详细说明。

如图 1 所示，图 1 为本发明实施例所述死锁检测方法的主体流程示意图，

主要包括 2 个阶段，即，死锁检测信息的收集和利用死锁检测信息检测死锁状态。所述死锁检测信息是指线程间的等待关系信息，所述等待关系信息全部存储在在一个邻接矩阵中，这个矩阵叫做 aov，它记录线程之间的等待关系信息。这里，加锁线程负责生成线程间等待关系信息，解锁线程负责在解锁过程中根据需要更新线程间的等待关系信息，死锁检测线程负责根据加解锁过程收集到的等待关系信息，利用 aov 网络原理，对线程进行拓扑排序，借以判断是否存在死锁，以及存在的循环等待环。

具体可以包括以下步骤:

步骤 101: 在加锁过程中，加锁线程如果需要等待其他线程，则将这个线程间等待关系信息记录在邻接矩阵 aov 中;

步骤 102: 在解锁过程中，解锁线程根据需要更新线程之间的这种等待关系信息，更改 aov 邻接矩阵中的相应数据;

步骤 103: 死锁检测线程先拷贝 aov 邻接矩阵（aov 邻接矩阵构成一个有向图 G）的一个备份，接着应用 aov 网络拓扑排序的方法对备份数据进行运算，查找其中的循环等待环（死锁回路）: 去掉所有没有前驱的结点以及所有它发出的弧，如果所有结点都可以去掉，则说明没有发生死锁，结束检测；否则说明存在死锁回路，根据预定策略（可以根据用户需要进行制定，比如最小代价策略，即选择工作量最小的线程作为牺牲线程），选取每个环中的一个结点作为死锁检测的牺牲线程。这里，死锁检测可以由一个后台独立线程（死锁检测线程）定期执行（事后检测），也可以在加锁过程中即时检测（事先检测）。

需要说明的是，我们不是机械地照搬 aov 网络理论，aov 网络主要是用来评估工程进度图的可行性，它用顶点表示活动，用弧表示活动间的优先关系。而

我们主要是借用 aov 网络的特点来作为死锁检测的有力工具：1）aov 网络中的弧表示活动之间存在的某种制约关系，2）aov 网络中不能出现回路。本发明用顶点表示线程本身，用弧表示线程之间的等待关系，当 aov 网络中不存在回路时表明不存在死锁，否则至少存在一个死锁。需要指出的是，并不是所有线程之间都存在等待关系，所有直接或者间接存在等待关系的线程构成一个狭义的小 aov 网络，本发明实施例所述的 aov 邻接矩阵中可能存在若干个这样的小 aov 网络。

下面进一步对本发明实施例所述方法进行详细说明。

死锁检测信息生成（即步骤 101）过程具体包括:

在对某个锁对象（例如，lm）加锁的过程中，如果当前线程 i 要申请的锁类型与 lm 已授予其他线程（任取其中一个线程 j）的锁类型不相容，则当前线程 i 就需要挂起等待。在挂起等待之前，在 aov 的 i 行 j 列记录这个等待关系信息(表示线程 i 等待线程 j)，而在唤醒并且获得申请的锁之后清空。

线程 i 申请锁对象 lm 的某个类型的锁，当该类型锁与 lm 已授予线程 j 的锁类型不相容时，表明线程 i 至少需要等待线程 j 释放相应类型的锁才能成功获得申请的锁类型。这样，线程 i 和线程 j 之间存在着某种偏序关系（等待关系），在有向图中用结点 i 到结点 j 的一条弧来表示，在 aov 邻接矩阵中则是在 aov[i][j] 单元中记录线程 i 正在等待的通知事件 event 的指针。

当线程 j 释放已获得的锁类型时，线程 i 和线程 j 之间的等待关系可能需要调整。

如图 2 所示，图 2 为本发明实施例所述方法中死锁检测信息更新(步骤 102)的流程示意图，具体可以包括:

步骤 201：线程 j 在对某个锁对象 lm 解锁的过程中，检查 aov 邻接矩阵中第 j 列中记录的等待这把锁的第一个线程 t;

步骤 202：判断线程 t 所要申请的锁类型与线程 j 拥有的锁类型是否相容，若是，执行步骤 203，否则转到步骤 205;

步骤 203：判断是否存在拥有该锁的其他线程 k，它拥有的锁类型与线程 t 所要申请的锁类型不相容？若是，则顺序执行步骤 204，否则转到步骤 205;

步骤 204：进行锁等待关系迁移，即令 aov[t] [k] = aov[t][j];

步骤 205：清除 aov[t][j]为空;

步骤 206：判断 aov 邻接矩阵第 j 列中是否还有申请这把锁的下一个线程，若是，转到步骤 202，否者结束解锁过程。

这里，由于线程 j 释放了锁对象 lm 的某个类型的锁，这将导致原来被线程 j 阻塞的其他线程有机会获得所申请的锁类型。对于被线程 j 阻塞的其他线程 t，此时存在 2 中可能，第一种可能是线程 j 拥有的锁类型与线程 t 正要申请的锁类型不相容，那么这 2 个线程之间的等待关系保持不变；第二种可能是线程 j 拥有的锁类型与线程 t 正要申请的锁类型相容，则线程 t 有机会获得所申请的锁类型，线程之间的等待关系可能需要改变，这取决于锁对象 lm 已授予别的线程的锁类型是否与线程 t 正要申请的锁类型相容。假设有某个线程 k 拥有的锁类型与线程 t 正要申请的锁类型不相容，那么线程 t 还是无法获得所申请的锁类型，在这种情况下，线程之间的等待关系发生了变化：线程 t 现在不再等待线程 j，而是等待线程 k。我们要求把这种等待关系的变化反映到邻接矩阵 aov 中，这个过程称为线程间等待关系迁移。线程解锁过程的任务就是要维护线程间等待关系迁移的信息。

以上线程加锁过程和解锁过程共同完成了死锁检测信息的收集过程，它们为死锁检测线程准备了全局的所有线程之间的等待关系信息。

如图 3 所示，图 3 为死锁检测算法的流程示意图，其描述了利用图论 aov 网络原理检测死锁的算法。这里要说明的是，虽然 aov 邻接矩阵中的数据是动态变化着的，但是我们使用 aov 邻接矩阵的备份数据来检测已经发生的死锁现象还是非常准确的。这是因为对于已经进入死锁状态的线程而言，它们都处于阻塞状态，在没有外部介入的情况下，死锁状态不可能自动解除，所以，虽然检测死锁过程中使用的是静态数据，但是还是可以准确无误地检测到业已发生的死锁。不过，在检测死锁期间我们不排除有新的死锁现象产生，它们需要等到下一次死锁检测过程才能检测得到。

具体包括以下步骤:

步骤301：拷贝aov的一个备份以便计算，因为没有什么歧义，所以这个备份还称为aov;

步骤302：计算aov的入度数组，结果存放在in_degree;

步骤303：计算aov的出度数组，结果存放在out_degree;

步骤304：in_degree是否等于out_degree？如果是，执行步骤305；否则执行步骤307;

步骤305：in_degree是否是0向量？如果是，表明没有出现回路，计算结束（唯一函数出口），否则转到步骤306;

步骤306：根据预定策略，从in_degree中选取一个非0的下标k，线程k就是要回滚事务所在的线程号。从aov中找到非0元素aov[k][j]，它是一个等待事件对

象指针，根据它找到所在的锁lm，找到线程k加锁信息表tli，将tli的status字段置成－1，然后唤醒线程k。将aov[k][j]清0，跳转步骤302;

步骤307: 对于所有出度不为0且入度为0的下标i，将aov[i][0..MAX_THREADS]清0。跳转步骤2）。这里MAX_THREADS表示最大线程数。

如图 4 所示，图 4 为用有向图描述的发生死锁现象的场景示意图。

图 4 的上半图描述了该系统在某一个时刻线程之间的等待关系，顶点表示线程，若顶点 t_i（i=1、2...10，i≠j）存在一条到达顶点 t_j（j=1、2...10，j≠i）的弧，则表示线程 t_i 等待 t_j，从图中可以看到存在 2 个死锁回路，应用 aov 网络拓扑排序方法可以检测到这 2 个回路。

如图 4 的下半图所示，死锁检测算法找到所有只有出度、没有入度的顶点，去掉这些顶点和它们发起的弧，不断重复该过程，直到没有任何这样的顶点。最后就剩下下半图所示的 2 个回路（死锁），根据预定策略，分别选定线程 t5 和线程 t8 作为破除死锁状态的牺牲线程。这里有必要提醒的是，任何顶点都最多只有一条由它发起的弧，这是因为任何线程申请锁失败之后就会被阻塞，没有机会再申请其他锁而发生第二次被阻塞的情形。

如图 5 和图 6 所示，图 5 和图 6 为应用图 3 所述算法检测死锁的 aov 邻接矩阵状态变化图，其中图 5 为检测到死锁前 aov 邻接矩阵的状态图，图 6 为检测到死锁后 aov 邻接矩阵的状态图。若顶点 t_i 存在一条到达顶点 t_j 的弧，则 aov 的第 i 行第 j 列就记录一个等待事件对象指针，在图 5 和图 6 中用实心圆点表示，没有实心圆点的单元都是空数据。为了简化表述，假设总共只有 10 个线程，线程号用阿拉伯数字表示，线程 t_i 的线程号就是 i。aov 邻接矩阵的第 i 行记录着线程 t_i 等待其他线程的信息，如前所述，显然，一行最多只有一个非空单元。aov

邻接矩阵的第 j 列记录着其他线程等待线程 t_i 的信息，一列可以有多个非空单元，表示多个线程在等待该线程。应用图 3 的算法，只需经过简单的矩阵运算：计算入度向量和出度向量，以及向量之间的逻辑比较，就可以对 aov 邻接矩阵进行化简，从而得到图 6 所示的矩阵状态，空心圆点所在行对应的线程就是选定的牺牲线程。由于矩阵运算就是简单的加法运算和逻辑比较，所以性能极高。

下面结合附图 7 对本发明实施例所述装置进行详细说明。

如图 7 所示，图 7 为本发明实施例所述装置的结构示意图，具体可以包括：信息存储模块、死锁检测模块、每个线程中的信息记录模块和信息更新模块，下面对各个模块分别进行详细说明。

信息存储模块，在加锁过程中，加锁线程如果需要等待其他线程，就会生成线程间等待关系信息，这些线程间的等待关系信息就存放到信息存储模块中；所述信息存储模块在本发明实施例中采用的是 aov 邻接矩阵的形式。

信息记录模块，可以包含于于线程中，当线程进行加锁时该线程称为加锁线程；在加锁过程中，当加锁线程要申请的锁类型与该锁对象已授予其他线程的锁类型不相容时，所述信息记录模块将加锁线程等待其他线程的等待关系信息记录在所述信息存储模块中，然后挂起等待相关线程进行解锁。比如，在对某个锁对象（例如，lm）加锁的过程中，如果加锁线程 i 要申请的锁类型与 lm 已授予其他线程（任取其中一个线程 j）的锁类型不相容，则加锁线程 i 的信息记录模块在 aov 邻接矩阵的 i 行 j 列记录这个等待关系信息（aov[i][j]，表示线程 i 等待线程 j），而在唤醒并且获得申请的锁之后清空。

信息更新模块，可以包含于于线程中，当线程进行解锁时该线程称为解锁线程；在相关线程进行解锁过程中，判断所述信息存储模块中记录的要申请这

把锁的每个线程，所要申请的锁类型是否与该解锁线程拥有的锁类型相容，若不相容，则保持等待关系信息不变，若相容，则进一步判断是否与至少一个其他线程拥有的锁类型不相容，如果是，则进行锁等待关系迁移，更新所述邻接矩阵中的相应等待关系信息，否则仅仅清除原来的线程间等待关系信息。比如，当解锁程 j 在对某个锁对象 lm 解锁的过程中，对于 aov 邻接矩阵第 j 列中每一个非空位置对应的线程 t，解锁程 j 的信息更新模块判断是否线程 t 正在等待的锁是 lm 且线程 t 所要申请的锁类型与本线程还拥有的锁类型相容，若不相容，则保持 aov[t][j]不变，若相容，则查找下一个已拥有的锁类型与线程 t 要申请的锁类型不相容的线程 k，令 aov[t][k] = aov[t][j]，清除 aov[t][j]为空；否则，仅仅清空 aov[t][j]为空。

需要注意的是，每个线程本身既可以包括信息记录模块，用于在该线程作为加锁线程时记录其与其他线程间的等待关系信息到所述信息存储模块中；还可以包括信息更新模块，用于在该线程作为解锁模块时根据需要更新所述信息存储模块中的相应等待关系信息。本发明实施例中仅以两个线程作为示例，但本领域技术人员应该知道，数据库管理系统中具有多个线程，自然就会产生不止一个的线程间等待关系信息。

死锁检测模块，本发明实施例中死锁检测模块由一个后台独立线程（死锁检测线程）定期执行（事后检测）或者在加锁过程中即时检测（事先检测）。死锁检测线程先拷贝 aov 邻接矩阵（aov 邻接矩阵构成一个有向图 G）的一个备份，接着应用图 3 所示的检测算法对备份数据进行运算，查找其中的循环等待环（死锁回路）：去掉所有没有前驱的结点以及所有它发出的弧，如果所有结点都可以去掉，则说明没有发生死锁，结束检测；否则说明存在死锁回路，根据预定策

略选取死锁回路中的一个结点作为死锁检测的牺牲线程。

对于本发明实施例所述装置的具体实现过程，由于本发明实施例所述方法中已进行了详细说明，故此处不再赘述。

综上所述，本发明实施例提供了一种数据库事务锁机制的死锁检测方法及装置，具体以下优点：

利用邻接矩阵来记录线程之间的等待关系，只需对矩阵进行简单的运算就可以检测出线程之间的所有循环等待环，对每个环，根据某个策略选定一个牺牲线程，让其失败返回，就可以破除死锁，具有极高的检测效率。

可以充分利用加、解锁过程已获得的信息来协助检测死锁，避免真正执行死锁检测过程中重复计算这些信息，从而提高了系统计算资源的利用率。而目前很多锁机制把加、解锁过程和死锁检测过程完全分开，丢失了一些有用的信息，造成后期执行死锁检测过程的重复计算，浪费了计算资源。

利用 aov 邻接矩阵备份数据来检测死锁，可以使得死锁检测线程极小妨碍正常线程的加、解锁过程，几乎不会阻塞正常线程的执行。而目前很多锁机制因为检测死锁过程中需要访问资源图和锁实体中的数据，所以死锁检测过程不可避免地会瞬时阻塞其他正常线程的加、解锁过程。

死锁检测阶段使用备份数据进行运算，不会干扰正常线程的加解锁过程，从而提高了系统的整体效率。

以上所述，仅为本发明较佳的具体实施方式，但本发明的保护范围并不局限于此，任何熟悉本技术领域的技术人员在本发明揭露的技术范围内，可轻易想到的变化或替换，都应涵盖在本发明的保护范围之内。因此，本发明的保护范围应该以权利要求书的保护范围为准。

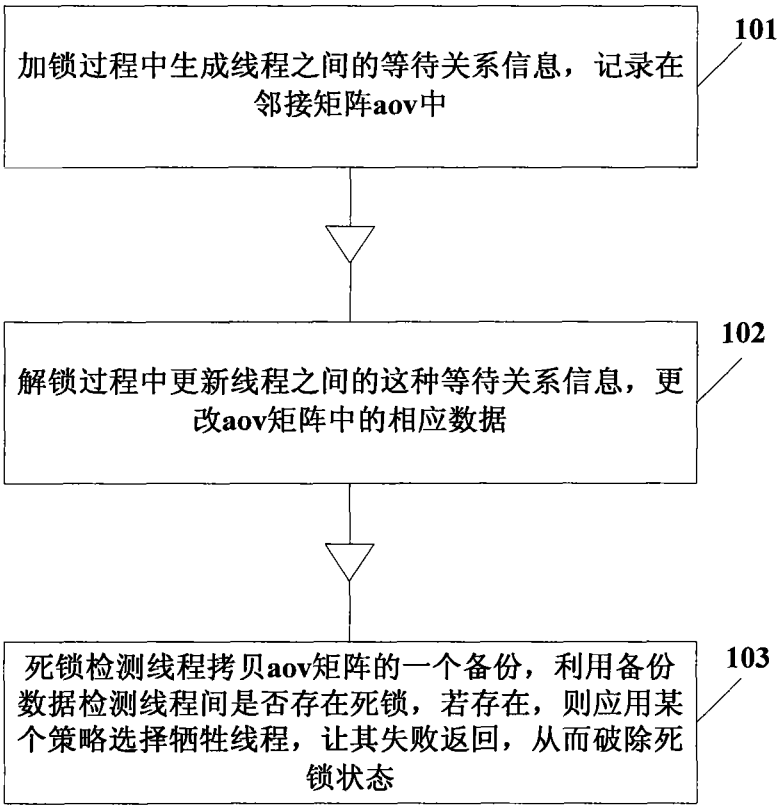

加锁过程中生成线程之间的等待关系信息，记录在邻接矩阵aov中
101
解锁过程中更新线程之间的这种等待关系信息，更改aov矩阵中的相应数据
102
死锁检测线程拷贝aov矩阵的一个备份，利用备份数据检测线程间是否存在死锁，若存在，则应用某个策略选择牺牲线程，让其失败返回，从而破除死锁状态
103

图 1

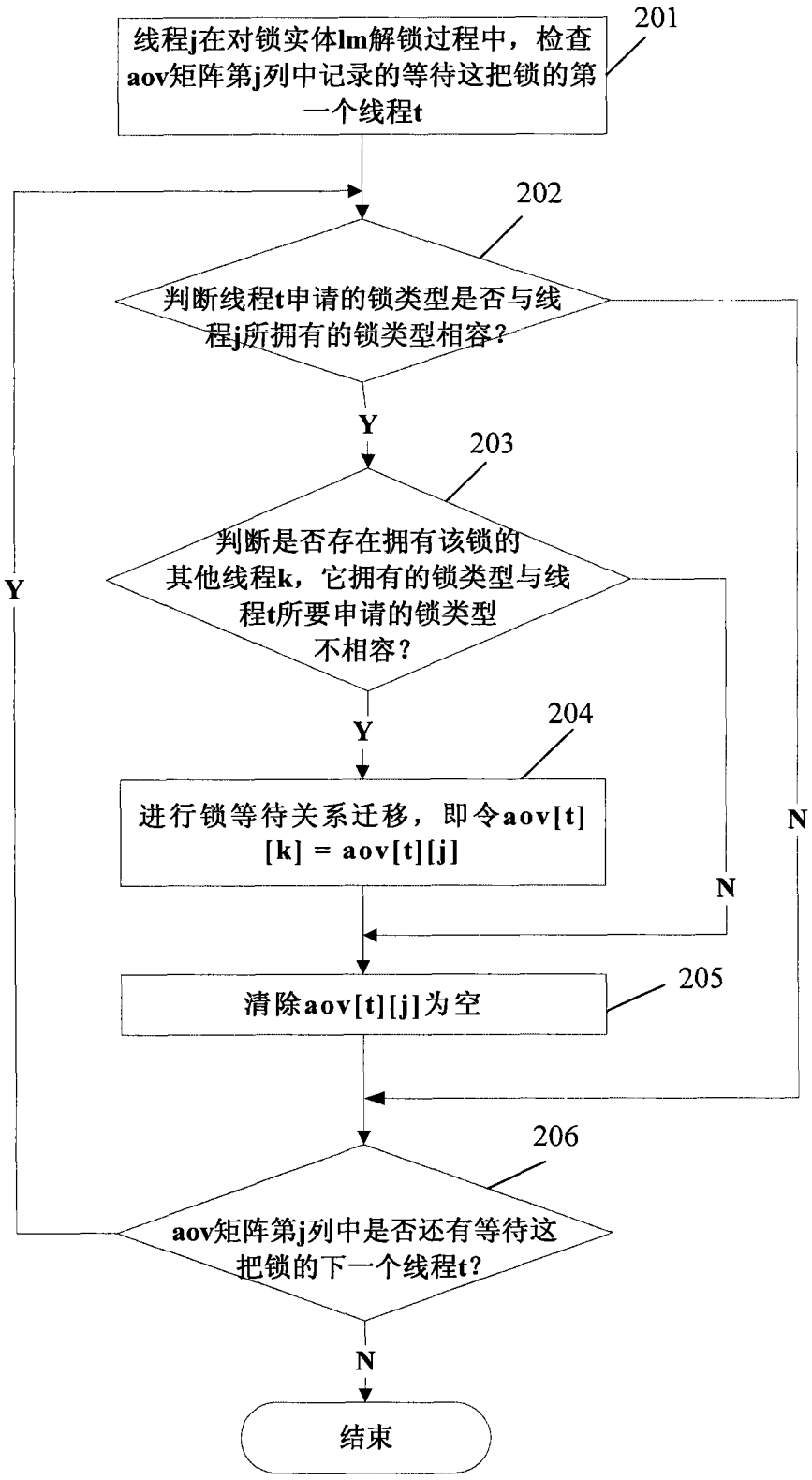

线程j在对锁实体lm解锁过程中，检查aov矩阵第j列中记录的等待这把锁的第一个线程t
201
判断线程t申请的锁类型是否与线程j所拥有的锁类型相容?
202
Y
判断是否存在拥有该锁的其他线程k，它拥有的锁类型与线程t所要申请的锁类型不相容?
203
Y
进行锁等待关系迁移，即令aov[t][k] = aov[t][j]
204
N
N
清除aov[t][j]为空
205
aov矩阵第j列中是否还有等待这把锁的下一个线程t?
206
Y
N
结束

图 2

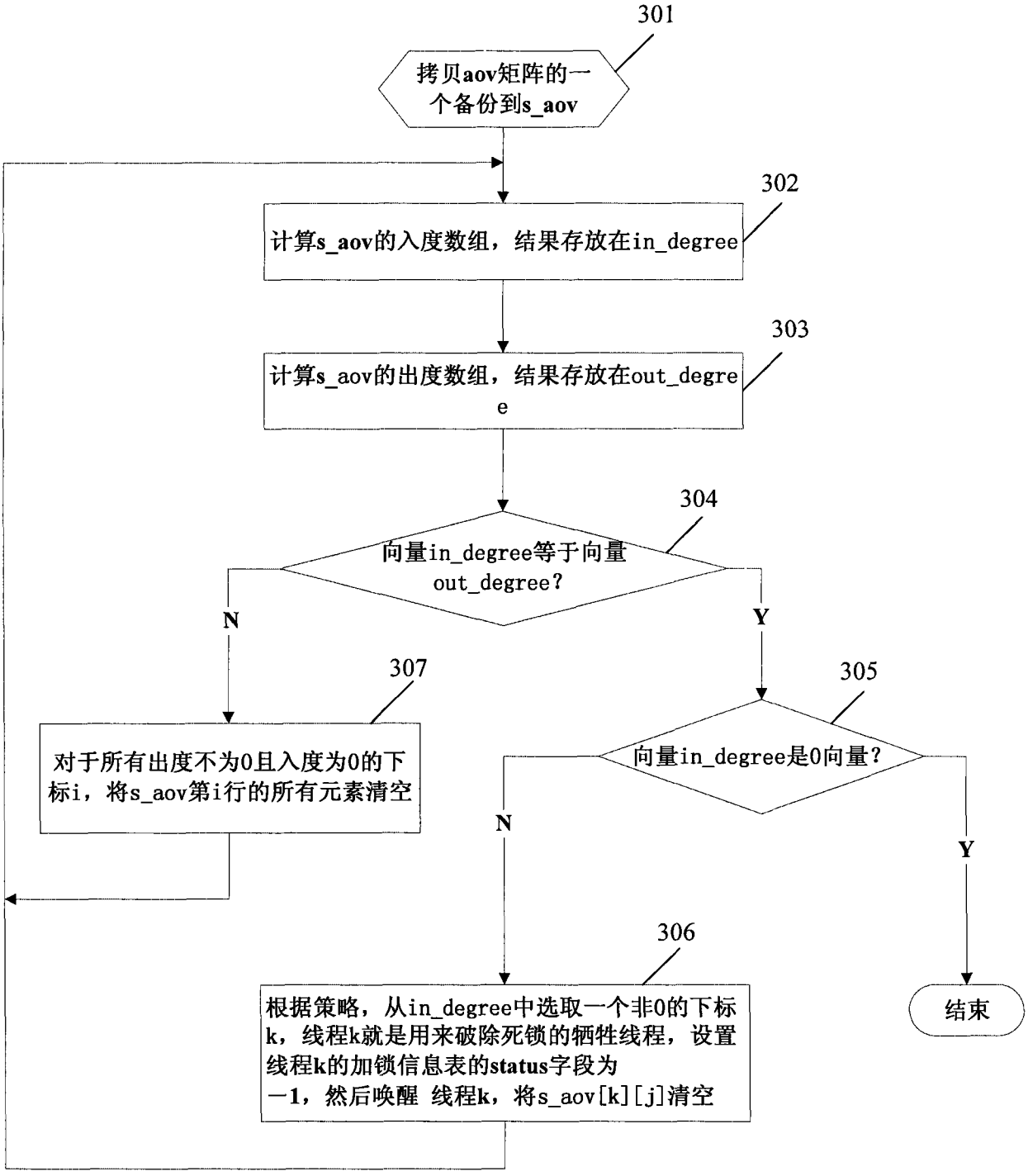

301
拷贝aov矩阵的一个备份到s_aov
302
计算s_aov的入度数组，结果存放在in_degree
303
计算s_aov的出度数组，结果存放在out_degree
304
向量in_degree等于向量out_degree?
N
Y
307
对于所有出度不为0且入度为0的下标i，将s_aov第i行的所有元素清空
305
向量in_degree是0向量?
N
Y
306
根据策略，从in_degree中选取一个非0的下标k，线程k就是用来破除死锁的牺牲线程，设置线程k的加锁信息表的status字段为-1，然后唤醒 线程k，将s_aov[k][j]清空
结束

图 3

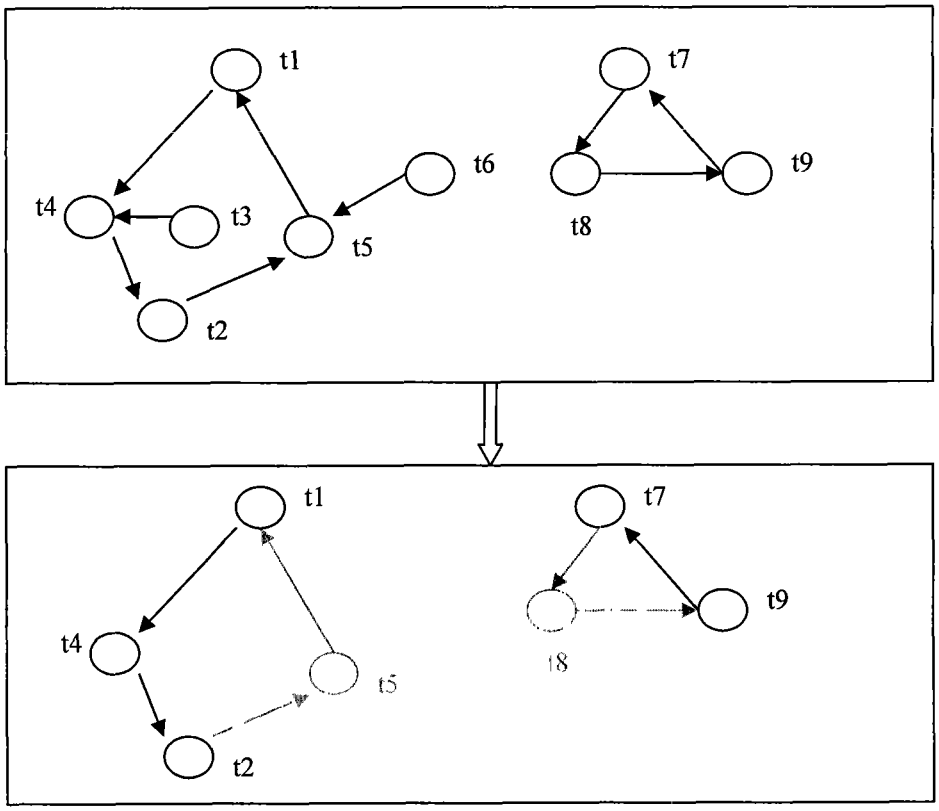


图 4

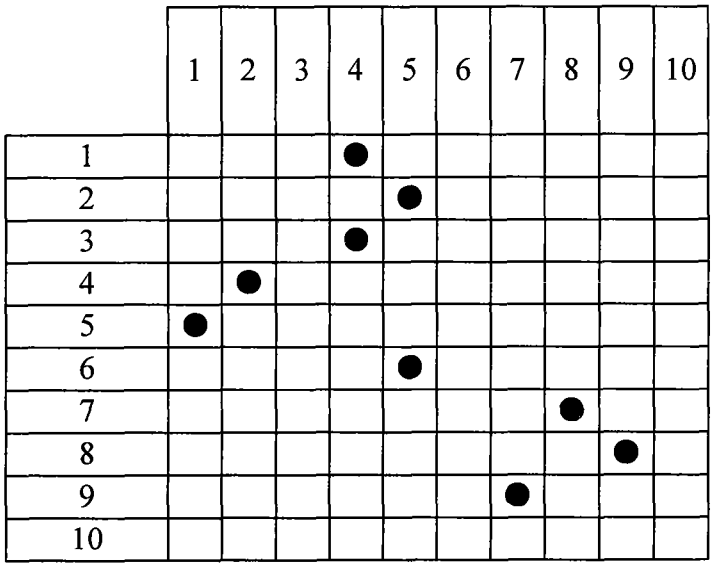


图 5

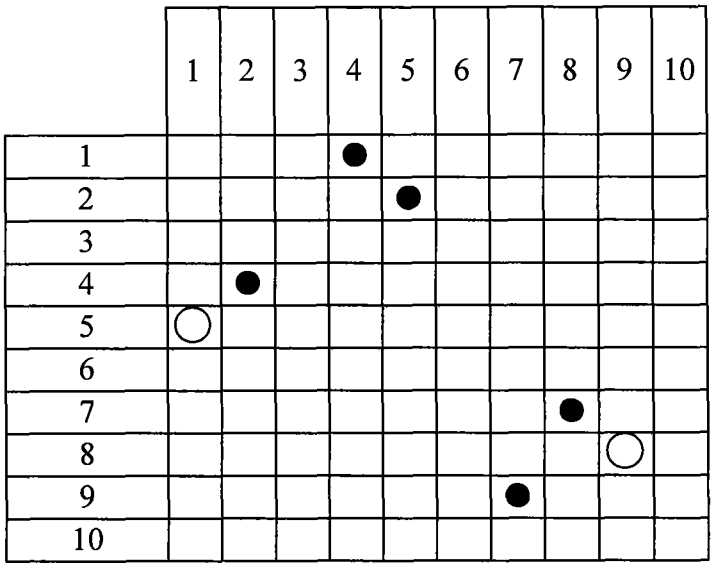


图 6

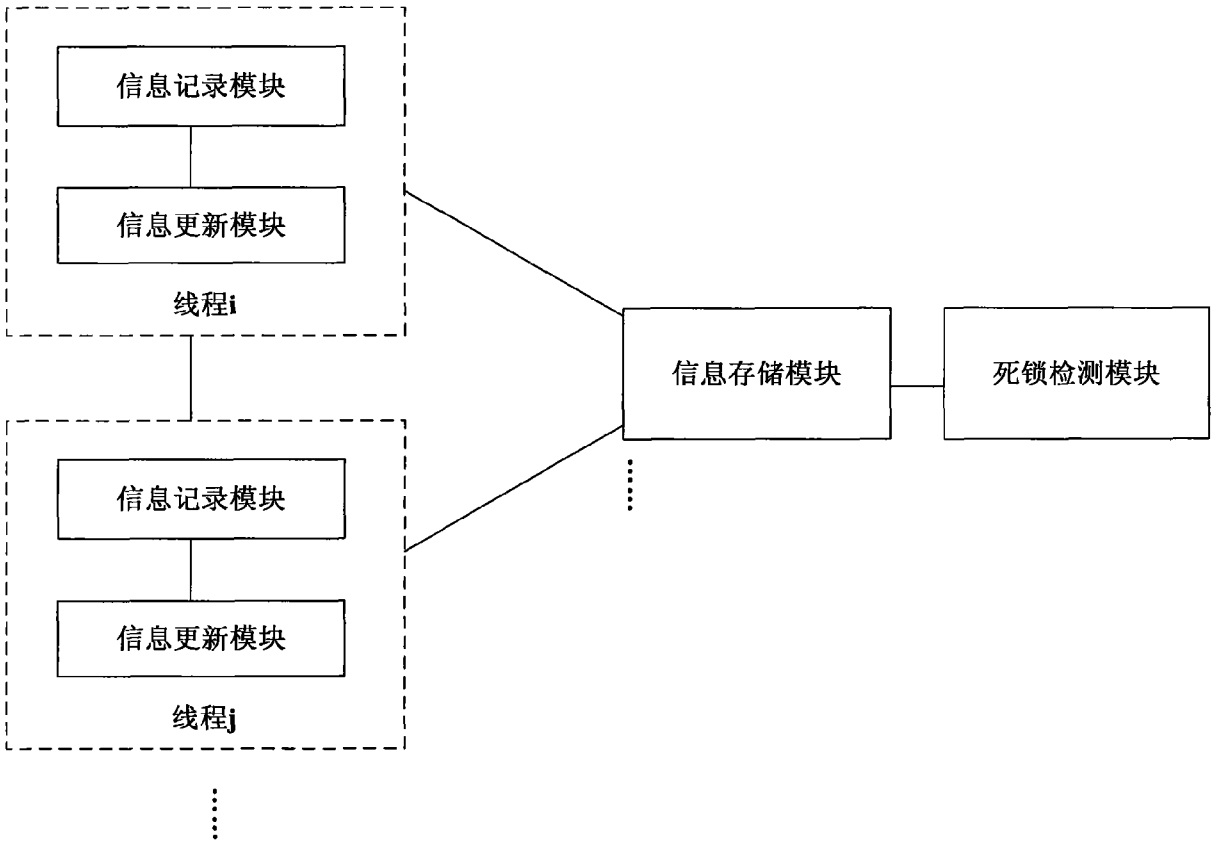


图 7