链滴

MySQL | 死锁和死锁检测

作者：douniwan

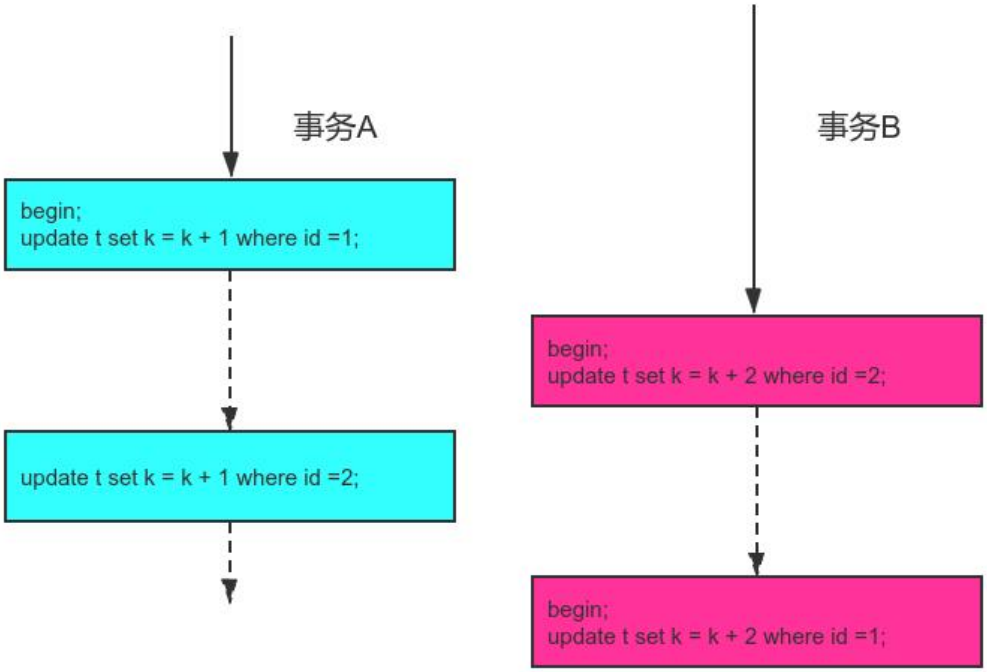
原文链接：https://ld246.com/article/1582361885687

来源网站：链滴

死锁的定义

当并发系统中不同线程出现循环资源依赖，涉及的线程都在等待别的线程释放资源时，就会导致这几个线程都进入无限等待的状态，称为死锁。

这里用innodb的行锁举个例子,如下图：



对上图的两个事务，由两阶段锁协议可知：这时候，事务 A 在等待事务 B 释放 id=2 的行锁，而事务 在等待事务 A 释放 id=1 的行锁。 事务 A 和事务 B 在互相等待对方的资源释放，就是进入了死锁状。

如何检测

当出现死锁以后，有两种策略：

- 直接进入等待，直到超时。这个超时时间可以通过参数 innodb_lock_wait_timeout 来设置
- 发起死锁检测，发现死锁后，主动回滚死锁链条中的某一个事务，让其他事务得以继续执行。将参数 nnodb_deadlock_detect 设置为 on，表示开启这个逻辑

如何选择策略

在 InnoDB 中，innodb_lock_wait_timeout 的默认值是 50s，意味着如果采用第一个策略，当出现锁以后，第一个被锁住的线程要过 50s 才会超时退出，然后其他线程才有可能继续执行。对于在线服来说，这个等待时间往往是无法接受的。但是，我们又不可能直接把这个时间设置成一个很小的值，如 1s。这样当出现死锁的时候，确实很快就可以解开，但如果不是死锁，而是简单的锁等待呢？所以超时时间设置太短的话，会出现很多误伤。

所以，**正常情况下我们还是要采用第二种策略**，即：主动死锁检测，而且innodb_deadlock_detect 默认值本身就是 on。主动死锁检测在发生死锁的时候，是能够快速发现并进行处理的，但是它也是额外负担。

你可以想象一下这个过程：每当一个事务被锁的时候，就要看看它所依赖的线程有没有被别人锁住，此循环，最后判断是否出现了循环等待，也就是死锁。那如果是我们上面说到的所有事务都要更新同行的场景呢？每个新来的被堵住的线程，都要判断会不会由于自己的加入导致了死锁，这是一个时间杂度是 O(n) 的操作。假设有 1000 个并发线程要同时更新同一行，那么死锁检测操作就是 100 万这量级的。虽然最终检测的结果是没有死锁，但是这期间要消耗大量的 CPU资源。因此，你就会看到 CU 利用率很高，但是每秒却执行不了几个事务。

如何热点行更新导致的性能问题？

● 一种头痛医头的方法，就是如果你能确保这个业务一定不会出现死锁，可以临时把死锁检测关掉。是这种操作本身带有一定的风险，因为业务设计的时候一般不会把死锁当做一个严重错误，毕竟出现锁了，就回滚，然后通过业务重试一般就没问题了，这是业务无损的。而关掉死锁检测意味着可能会现大量的超时，这是业务有损的。

● 另一个思路是控制并发度。根据上面的分析，你会发现如果并发能够控制住，比如同一行同时最多有 10 个线程在更新，那么死锁检测的成本很低，就不会出现这个问题。一个直接的想法就是，在客端做并发控制。但是，你会很快发现这个方法不太可行，因为客户端很多。我见过一个应用，有 600 客户端，这样即使每个客户端控制到只有 5 个并发线程，汇总到数据库服务端以后，峰值并发数也可要达到 300

因此，这个并发控制要做在**数据库服务端**。如果你有中间件，可以考虑在中间件实现；如果你的团队能修改 MySQL 源码的人，也可以做在 MySQL 里面。基本思路就是，对于相同行的更新，在进入引之前排队。这样在 InnoDB 内部就不会有大量的死锁检测工作。

本文转载于：极客时间·MySQL实战45讲