MySQL REPLACE死锁问题深入剖析

姜承尧

IT圈最会讲故事的破产码农

上周五发布的MySQL ACE或许都说不清的死锁问题一例，很多同学跃跃欲试想要尝试年薪100万的MySQL难题，可惜的是后台很多同学没人答对此题。同样的，若只将此题理解成REPLACE会拆成DELETE+INSERT也是不正确的，因为REPLACE在锁的处理上并非完全的等同于DELETE + INSERT。虽然同样可以模拟出死锁，然而两者输出的结果是不一样的。此外，这个死锁只需要并发度为2就能出线死锁，又进一步提升了其逼格。

本题难点在于下面这几部分：

- INSERT并发执行：包括RC和RR事务隔离级别的理解
- INSERT唯一索引产生锁等待的条件：REPLACE产生何种锁等待
- INSERT INTENTION LOCK产生的原因与前提

就上述这几个知识点已足以作为P8面试题，更何况是这个死锁难题。在这里姜老师将逐步分析上面的三个知识点，通过这三个知识点看看小伙伴是否能有解答出上题的能力。anyway，要自己动手分析和实践哦。

方圆几里 薛之谦 - 意外

INSERT并发执行

INSERT是否可以并发执行取决于事务隔离级别和外键约束。通常来说，外

键使用的比较少，这部分暂不做考虑。那么RC事务隔离级别由于基本没有（注意只是基本，而不是完全没有）了Next-key Lock，因此很多同学会认为INSERT操作是可以并行执行的。

但是要让INSERT可达到完全的并行执行，还需要有一个前提条件：

> 插入的表没有唯一索引，或者插入的数据唯一约束的没有冲突（表上没有外键约束）

INSERT唯一索引产生锁等待的条件

若有唯一索引，并且产生了锁等待，则InnoDB会有两种完全不同的锁机制，这部分在文档中其实有过说明（然而并不准确）：

1. If a duplicate-key error occurs, a shared lock on the duplicate index record is set.

2. REPLACE is done like an INSERT if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.

这部分文档其实说的并非准确（主要原因是锁其实非常复杂，并不好表述），还是由姜老师来说明下。

对于INSERT操作来说，**若发生唯一约束冲突，则需要对冲突的唯一索引加上S Next-key Lock。从这里会发现，即使是RC事务隔离级别，也同样会存在Next-Key Lock锁，从而阻塞并发。**

然而，文档没有说明的是，**对于检测到冲突的唯一索引，等待线程在获得S Lock之后，还需要对下一个记录进行加锁**，在源码中由函数row_ins_scan_sec_index_for_duplicate进行判断。

通过下面的这个例子可以很好的进行解释（RC事务隔离级别）：

```
CREATE TABLE c (

   a INT AUTO_INCREMENT PRIMARY KEY,

   b INT,

   c INT,

   UNIQUE KEY (b)

)

INSERT INTO c values (NULL,1,2);
```

接着按下面的步骤执行：

session1 session2

| begin

| delete from c where b = 1; begin;

| insert into c values (NULL,1) ;

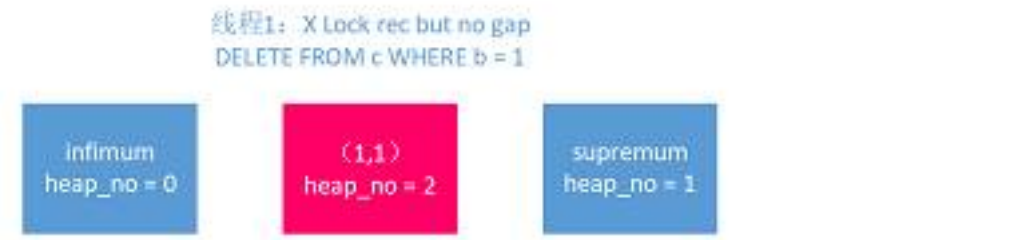
| commit;

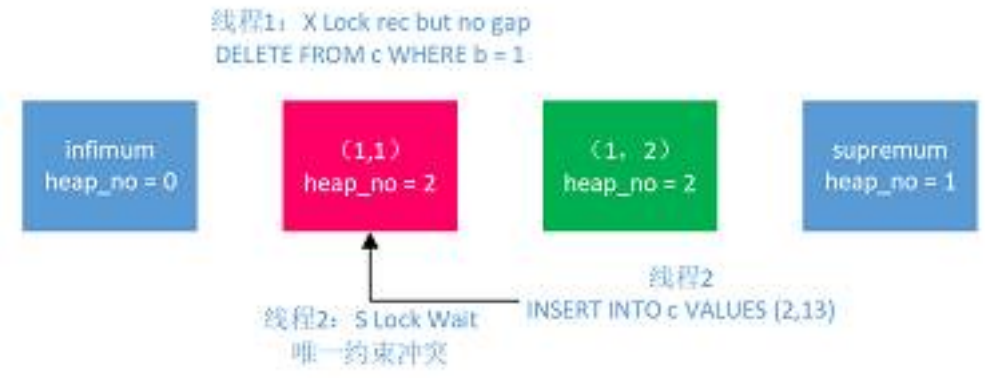
V commit;

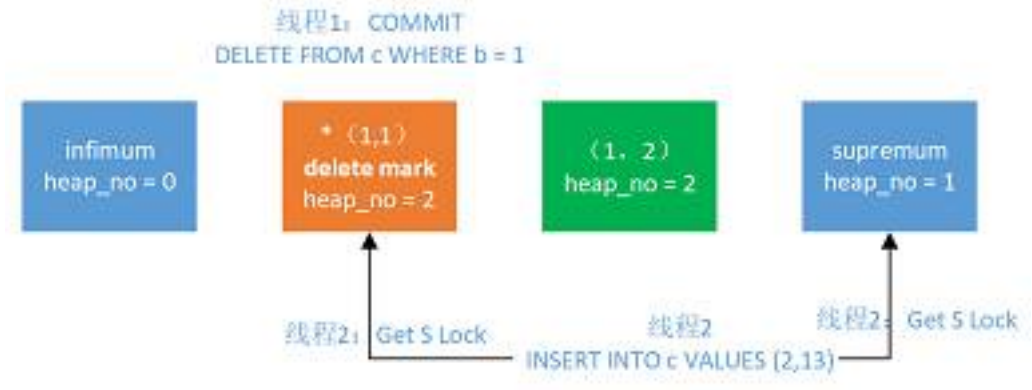
由于唯一索引b=1这条记录上有锁，因此线程2在插入时需要等待，这时需要获取b=1的S锁（S Next-Key Lock），当线程1提交时，线程2顺利获得S锁，但是同样会对下一条记录也加上S锁。通过命令SHOW ENGINE INNODB STATUS看到的结果是：

> TABLE LOCK table `mysqlslap`.`d` trx id 78416369 lock mode IX
>
> RECORD LOCKS space id 500 page no 4 n bits 72 index b of table `mysqlslap`.`d` trx id 78416369 **lock mode S**
>
> Record lock, **heap no 1** PHYSICAL RECORD: n_fields 1; compact format; info bits 0 0: len 8; hex 73757072656d756d; asc supremum;;
>
> RECORD LOCKS space id 500 page no 4 n bits 72 index b of table `mysqlslap`.`d` trx id 78416369 **lock mode S locks gap before rec**
>
> Record lock, **heap no 2** PHYSICAL RECORD: n_fields 2; compact format; info bits 0
>
> 0: len 4; hex 80000001; asc ;;
>
> 1: len 4; hex 80000002; asc ;;

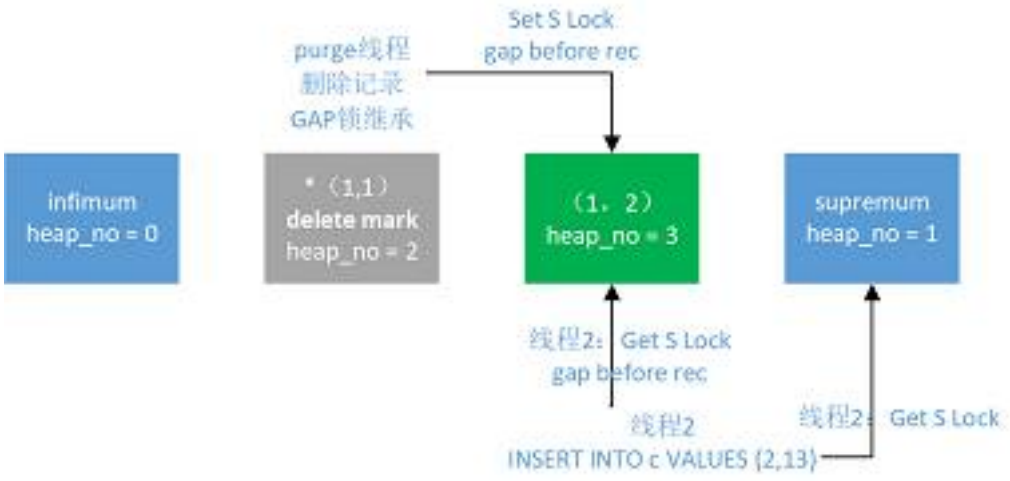
看到了S锁，但有同学可能困惑的是，这里只有heap_no=1的记录上加上了锁，而没有之前删除的那条记录。这是因为删除的记录，由于已经提交，被purge线程删除，并将锁继承给了待插入的记录。所以整个流程应为：


线程1：X Lock rec but no gap
DELETE FROM c WHERE b = 1
infimum
heap_no = 0
（1,1）
heap_no = 2
supremum
heap_no = 1


线程1：X Lock rec but no gap
DELETE FROM c WHERE b = 1
infimum
heap_no = 0
（1,1）
heap_no = 2
（1，2）
heap_no = 2
supremum
heap_no = 1
线程2
INSERT INTO c VALUES (2,13)
线程2：S Lock Wait
唯一约束冲突


线程1：COMMIT
DELETE FROM c WHERE b = 1
infimum
heap_no = 0
* （1,1）
delete mark
heap_no = 2
（1，2）
heap_no = 2
supremum
heap_no = 1
线程2：Get S Lock
线程2
INSERT INTO c VALUES (2,13)
线程2：Get S Lock


infimum
heap_no = 0
* （1,1）
delete mark
heap_no = 2
（1，2）
heap_no = 3
supremum
heap_no = 1
线程2：Get S Lock
gap before rec
线程2：Get S Lock
线程2
INSERT INTO c VALUES (2,13)


purge线程
删除记录
GAP锁继承
Set S Lock
gap before rec
infimum
heap_no = 0
* （1,1）
delete mark
heap_no = 2
（1，2）
heap_no = 3
supremum
heap_no = 1
线程2：Get S Lock
gap before rec
线程2：Get S Lock
线程2
INSERT INTO c VALUES (2,13)


infimum
heap_no = 0
（1，2）
heap_no = 3
supremum
heap_no = 1
线程2：Get S Lock

REPLACE操作和INSERT不同的点在于，这就留给小伙伴们自己测试了：

- REPLACE遇到唯一索引约束冲突会加X锁（Next Key Lock）

- REPLACE操作由于删除的记录没有提交，因此无法被purge线程删除

INSERT INTENTION LOCK

INSERT INTENTION LOCK，翻译为插入意向锁，其实准确来说应该是INSERT INTENTION GAP LOCK。这个锁类型在老版本的InnoDB中并不存在，后来是为了优化插入性能而设计的。官方文档中的说明如下：

> Prior to inserting the row, a type of gap lock called an **insert intention gap lock** is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap.

个人觉得文档的说明依然有些歧义，这把锁并非每次插入需要加的。另外，在RC事务隔离级别下，由于插入大部分是不需要等待的，所以这把锁大部分时候也是不存在的。只有当发生锁等待时，即插入的这条记录下一条记录next_rec有锁，并且带有GAP属性，则这时需要对next_rec再加一个插入意向锁。由于插入意向锁和S/X Lock不兼容，因此需要等待。
在RC事务隔离级别下，虽然大多数插入操作是并发的，不会发生锁等待。然而，由于唯一约束的存在，这时就需要加上插入意向锁。而这也是上次死锁案例问题产生的原因。
最后，我们来体会下，上次的死锁输出：

> *** (1) TRANSACTION:
>
> TRANSACTION 78601472, ACTIVE 0 sec **updating or deleting**
>
> .....

*** (1) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 498 page no 4 n bits 72 index b of table `mysqlslap`.`cc` trx id 78601472 **lock_mode X locks gap before rec insert intention waiting**

Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 32

 0: len 4; hex 80000001; asc   ;;

 1: len 4; hex 80000013; asc   ;;

*** (2) TRANSACTION:

TRANSACTION 78601476, ACTIVE 0 sec **inserting**

.....

replace into cc values (NULL,1)

*** (2) HOLDS THE LOCK(S):

RECORD LOCKS space id 498 page no 4 n bits 72 index b of table `mysqlslap`.`cc` trx id 78601476 **lock_mode X locks gap before rec**

Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format; info bits 32

 0: len 4; hex 80000001; asc   ;;

 1: len 4; hex 80000013; asc   ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 498 page no 4 n bits 72 index b of table `mysqlslap`.`cc` trx id 78601476 lock_mode X waiting

Record lock, heap no 4 PHYSICAL RECORD: n_fields 2; compact format;

```
info bits 32

 0: len 4; hex 80000001; asc     ;;

 1: len 4; hex 80000013; asc     ;;
```

其实如果唯一索引引起的问题，都很好总结，你会发现**死锁锁住的都是同一条记录**，并且已经持有锁的线程在等待同一条记录的锁，比如上面的（1，18）这条记录。

好累，终于讲完，希望同学们都看懂了。第一次写篇文章写到想要放弃，涉及的知识点非常多。同学们回去慢慢消化，也祝大家都能早日成为公司中死锁分析的小王子。

猜你喜欢

- 中国数据库排行榜 · 2017年5月 TiDB进前10！！！

- WTF？MySQL DBA技术难度低为什么工资比Oracle高？
- 如何花肯德基的钱，吃五星级酒店自助餐？
- MySQL 8.0.1 SKIP LOCKED，热行（Hot Rows）并发新武器？
- IT人的34岁之疡？
- 破产码农看房价
- 终于来了，MySQL 5.7与PostgreSQL 9.6的百万QPS大比拼
- 一触即发，2017年，数据库世界的诸神之战
- MySQL Group Replication性能测试，星辰大海还是前路茫茫？
- 为什么我不再看好MariaDB
- 原谅我这么幼稚，所以才会喜欢你这麼久 #MySQL#
- 从你的全世界路过
- 男人找个支持你理想的老婆，少奋斗25年