07 | 行锁功过：怎么减少行锁对性能的影响？

2018-11-28 林晓斌

在上一篇文章中，我跟你介绍了MySQL的全局锁和表级锁，今天我们就来讲讲MySQL的行锁。

MySQL的行锁是在引擎层由各个引擎自己实现的。但并不是所有的引擎都支持行锁，比如MyISAM引擎就不支持行锁。不支持行锁意味着并发控制只能使用表锁，对于这种引擎的表，同一张表上任何时刻只能有一个更新在执行，这就会影响到业务并发度。InnoDB是支持行锁的，这也是MyISAM被InnoDB替代的重要原因之一。

我们今天就主要来聊聊InnoDB的行锁，以及如何通过减少锁冲突来提升业务并发度。

顾名思义，行锁就是针对数据表中行记录的锁。这很好理解，比如事务A更新了一行，而这时候事务B也要更新同一行，则必须等事务A的操作完成后才能进行更新。

当然，数据库中还有一些没那么一目了然的概念和设计，这些概念如果理解和使用不当，容易导致程序出现非预期行为，比如两阶段锁。

从两阶段锁说起

我先给你举个例子。在下面的操作序列中，事务B的update语句执行时会是什么现象呢？假设字段id是表t的主键。

这个问题的结论取决于事务A在执行完两条update语句后，持有哪些锁，以及在什么时候释放。你可以验证一下：实际上事务B的update语句会被阻塞，直到事务A执行commit之后，事务B才能继续执行。

知道了这个答案，你一定知道了事务A持有的两个记录的行锁，都是在commit的时候才释放的。

也就是说，**在InnoDB事务中，行锁是在需要的时候才加上的，但并不是不需要了就立刻释放，而是要等到事务结束时才释放。这个就是两阶段锁协议。**

知道了这个设定，对我们使用事务有什么帮助呢？那就是，如果你的事务中需要锁多个行，要把最可能造成锁冲突、最可能影响并发度的锁尽量往后放。我给你举个例子。

假设你负责实现一个电影票在线交易业务，顾客A要在影院B购买电影票。我们简化一点，这个业务需要涉及到以下操作：

1. 从顾客A账户余额中扣除电影票价；
2. 给影院B的账户余额增加这张电影票价；
3. 记录一条交易日志。

也就是说，要完成这个交易，我们需要update两条记录，并insert一条记录。当然，为了保证交

易的原子性，我们要把这三个操作放在一个事务中。那么，你会怎样安排这三个语句在事务中的顺序呢？

试想如果同时有另外一个顾客C要在影院B买票，那么这两个事务冲突的部分就是语句2了。因为它们要更新同一个影院账户的余额，需要修改同一行数据。

根据两阶段锁协议，不论你怎样安排语句顺序，所有的操作需要的行锁都是在事务提交的时候才释放的。所以，如果你把语句2安排在最后，比如按照3、1、2这样的顺序，那么影院账户余额这一行的锁时间就最少。这就最大程度地减少了事务之间的锁等待，提升了并发度。

好了，现在由于你的正确设计，影院余额这一行的行锁在一个事务中不会停留很长时间。但是，这并没有完全解决你的困扰。

如果这个影院做活动，可以低价预售一年内所有的电影票，而且这个活动只做一天。于是在活动时间开始的时候，你的MySQL就挂了。你登上服务器一看，CPU消耗接近100%，但整个数据库每秒就执行不到100个事务。这是什么原因呢？

这里，我就要说到死锁和死锁检测了。

死锁和死锁检测

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。这里我用数据库中的行锁举个例子。

这时候，事务A在等待事务B释放id=2的行锁，而事务B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，就是进入了死锁状态。当出现死锁以后，有两种策略：

- 一种策略是，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。
- 另一种策略是，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innodb_deadlock_detect设置为on，表示开启这个逻辑。

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，当出现死锁以后，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。对于在线服务来说，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，比如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的默认值本身就是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，但是它也是有额外负担的。

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，如此循环，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一行的场景呢？

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，这是一个时间复杂度是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100万这个量级的。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CPU利用率很高，但是每秒却执行不了几个事务。

根据上面的分析，我们来讨论一下，怎么解决由这种热点行更新导致的性能问题呢？问题的症结在于，死锁检测要耗费大量的CPU资源。

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。但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，毕竟出现死锁了，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出现大量的超时，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只有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，在客户端做并发控制。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个客户端，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，峰值并发数也可能要达到3000。

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；如果你的团队有能修改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，在进入引擎之前排队。这样在InnoDB内部就不会有大量的死锁检测工作了。

可能你会问，**如果团队里暂时没有数据库方面的专家，不能实现这样的方案，能不能从设计上优化这个问题呢？**

你可以考虑通过将一行改成逻辑上的多行来减少锁冲突。还是以影院账户为例，可以考虑放在多条记录上，比如10个记录，影院的账户总额等于这10个记录的值的总和。这样每次要给影院账户加金额的时候，随机选其中一条记录来加。这样每次冲突概率变成原来的1/10，可以减少锁等待个数，也就减少了死锁检测的CPU消耗。

这个方案看上去是无损的，但其实这类方案需要根据业务逻辑做详细设计。如果账户余额可能会减少，比如退票逻辑，那么这时候就需要考虑当一部分行记录变成0的时候，代码要有特殊处理。

小结

今天，我和你介绍了MySQL的行锁，涉及了两阶段锁协议、死锁和死锁检测这两大部分内容。

其中，我以两阶段协议为起点，和你一起讨论了在开发的时候如何安排正确的事务语句。这里的原则/我给你的建议是：如果你的事务中需要锁多个行，要把最可能造成锁冲突、最可能影响并发度的锁的申请时机尽量往后放。

但是，调整语句顺序并不能完全避免死锁。所以我们引入了死锁和死锁检测的概念，以及提供了三个方案，来减少死锁对数据库的影响。减少死锁的主要方向，就是控制访问相同资源的并发事务量。

最后，我给你留下一个问题吧。如果你要删除一个表里面的前10000行数据，有以下三种方法可以做到：

- 第一种，直接执行delete from T limit 10000;
- 第二种，在一个连接中循环执行20次 delete from T limit 500;
- 第三种，在20个连接中同时执行delete from T limit 500。

你会选择哪一种方法呢？为什么呢？

你可以把你的思考和观点写在留言区里，我会在下一篇文章的末尾和你讨论这个问题。感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

上期问题时间

上期我给你留的问题是：当备库用-single-transaction做逻辑备份的时候，如果从主库的binlog传来一个DDL语句会怎么样？

假设这个DDL是针对表t1的， 这里我把备份过程中几个关键的语句列出来：

```
Q1:SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Q2:START TRANSACTION  WITH CONSISTENT SNAPSHOT;
/* other tables */
Q3:SAVEPOINT sp;
/* 时刻 1 */
Q4:show create table `t1`;
/* 时刻 2 */
Q5:SELECT * FROM `t1`;
/* 时刻 3 */
Q6:ROLLBACK TO SAVEPOINT sp;
/* 时刻 4 */
/* other tables */
```

在备份开始的时候，为了确保RR（可重复读）隔离级别，再设置一次RR隔离级别(Q1);

启动事务，这里用 WITH CONSISTENT SNAPSHOT确保这个语句执行完就可以得到一个一致性视图（Q2);

设置一个保存点，这个很重要（Q3）；

show create 是为了拿到表结构(Q4)，然后正式导数据 （Q5），回滚到SAVEPOINT sp，在这里的作用是释放 t1的MDL锁 （Q6）。当然这部分属于“超纲”，上文正文里面都没提到。

DDL从主库传过来的时间按照效果不同，我打了四个时刻。题目设定为小表，我们假定到达后，如果开始执行，则很快能够执行完成。

参考答案如下：

1. 如果在Q4语句执行之前到达，现象：没有影响，备份拿到的是DDL后的表结构。
2. 如果在“时刻 2”到达，则表结构被改过，Q5执行的时候，报 Table definition has changed, please retry transaction，现象：mysqldump终止；
3. 如果在“时刻2”和“时刻3”之间到达，mysqldump占着t1的MDL读锁，binlog被阻塞，现象：主从延迟，直到Q6执行完成。
4. 从“时刻4”开始，mysqldump释放了MDL读锁，现象：没有影响，备份拿到的是DDL前的表结构。

评论区留言点赞板：

> @Aurora 给了最接近的答案；
> @echo__陈 问了一个好问题；
> @壹笙□漂泊 做了很好的总结。

精选留言

木木北月生

55

老师，关于死锁检测innodb_deadlock_detect我想请教一下，是每条事务执行前都会进行检测吗？如果是这样，即使简单的更新单个表的语句，当每秒的并发量达到上千的话，岂不是也会消耗大量资源用于死锁检测吗？

2018-12-03

作者回复

是个好问题

如果他要加锁访问的行上有锁，他才要检测。

这里面我担心你有两个误解，说明下：

1. 一致性读不会加锁，就不需要做死锁检测；

2. 并不是每次死锁检测都都要扫所有事务。比如某个时刻，事务等待状态是这样的：

B在等A，

D在等C，

现在来了一个E，发现E需要等D，那么E就判断跟D、C是否会形成死锁，这个检测不用管B和A

2018-12-03

bluefantasy3

54

请教老师一个问题：

innodb行级锁是通过锁索引记录实现的。如果update的列没建索引，即使只update一条记录也

会锁定整张表吗？比如update t set t.name='abc' where t.name='cde'; name字段无索引。为何innodb不优化一下，只锁定name='cde'的列？

2018-11-28

作者回复

第一个问题是好问题，我加到答疑文章中。简单的回答：是的。但是你可以再往前考虑一下，如果是 你的update 语句后面加个limit 1, 会怎么锁？

Innodb支持行锁，没有支持“列锁” 哈

2018-11-28

荒漠甘泉

41

老师，本节课讲的不支持行锁的引擎，只能使用表锁，而表锁同一张表在同一时刻只能有一个更新。但是上节课讲的表级锁中的MDL锁，dml语句会产生MDL读锁，而MDL读锁不是互斥的，也就是说一张表可以同时有多个dml语句操作。感觉这两种说法有点矛盾，请老师解惑！

2018-11-28

作者回复

不矛盾，MDL锁和表锁是两个不同的结构。

比如：

你要在myisam 表上更新一行，那么会加MDL读锁和表的写锁；

然后同时另外一个线程要更新这个表上另外一行，也要加MDL读锁和表写锁。

第二个线程的*MDL读锁是能成功加上*的，但是被表写锁堵住了。从语句现象上看，就是第二个线程要等第一个线程执行完成。

2018-11-28

蓝天

17

老师：上一节讲的dml时会产生读MDL锁（表锁），也就是update会持有读MDL。读和读不互斥。但是对于行锁来说。两个update同时更新一条数据是互斥的。这个是因为多种锁同时存在时，以粒度最小的锁为准的原因么？

2019-01-09

作者回复

不是“以粒度最小为准”

而是如果有多种锁，必须得“全部不互斥”才能并行，只要有一个互斥，就得等。

好问题

2019-01-09

三木禾

10

老师，如果开启事务，然后进行死锁检测，如果发现有其它线程因为这个线程的加入，导致其它线程的死锁，这个流程能帮着分析一下么

2018-12-14

作者回复

好问题

理论上说，之前没死锁，现在A加进来，出现了死锁，那么死锁的环里面肯定包含A，

因此只要从A出发去扫就好了

2018-12-14

某、人

8

老师，针对我的第一个问题。我就是想问怎么能拿到比较全得死锁信息以及把这些信息保存到文件里。

第二个问题，如果reset以后，是不是就失去了长连接的意义了呢？相当于再次进行连接。

2018-11-28

作者回复

1. 就是持续监控，发现新的就存起来

2. 不会，reset_connection只是复位状态，恢复到连接和权限验证之后的状态，没有重连

2018-11-29

武者

5

老师 你好

有以下情况 帮忙分析下会锁表不

update a，b set a.name = b.name where a.uid=b.uid and b.group=1；

update c，b set c.age=b.age where c.uid=b.uid and b.group = 1；

如果两个语句同时执行期间 是不是有个执行不了 要等b解锁。还是说没有更新b的字段b不会锁，两个可并行执行

请老师指导下

2018-11-29

作者回复

这个你得同时贴表结构。

还有，会不会锁，不是验证一下就可以吗，两个都用begin + 语句，

两阶段锁协议会帮助你

2018-11-29

泉

52

我选第二种。
第一种，需要锁资源多，事务较大，持有锁时间最长。
第三种，多个事务会对同一行产生锁竞争，消耗cpu资源。
请指正。

2018-11-28

Tony Du 48

方案一，事务相对较长，则占用锁的时间较长，会导致其他客户端等待资源时间较长。
方案二，串行化执行，将相对长的事务分成多次相对短的事务，则每次事务占用锁的时间相对较短，其他客户端在等待相应资源的时间也较短。这样的操作，同时也意味着将资源分片使用（每次执行使用不同片段的资源），可以提高并发性。
方案三，人为自己制造锁竞争，加剧并发量。
方案二相对比较好，具体还要结合实际业务场景。

另，对于innodb的行锁，我觉得可以增加一讲，如何加锁（依赖于具体的隔离级别，是否有索引，是否是唯一索引，SQL的执行计划），特别是在RR隔离级别下的GAP锁，对于innodb，RR级别是可以防止幻读的。

2018-11-28

作者回复

分析得很好。

嗯嗯索引和锁的内容很多，也是需要慢慢安排

突然上概念怕大家看得不开心

2018-11-28

WL 29

继续把该讲内容总结为几个问题, 大家复习的时候可以先尝试回答这些问题检查自己的掌握程度:

1.
两阶段锁的概念是什么? 对事务使用有什么帮助?
2.
死锁的概念是什么? 举例说明出现死锁的情况.
3.
死锁的处理策略有哪两种?
4.
等待超时处理死锁的机制什么?有什么局限?
5.
死锁检测处理死锁的机制是什么? 有什么局限?
6.
有哪些思路可以解决热点更新导致的并发问题?

2018-12-01

作者回复

继续手动

2018-12-01

岁月安然

25

总结：

两阶段锁：在 InnoDB 事务中，锁是在需要的时候才加上的，但并是需要就立刻释放， 而是要等到事务结束时才释放。

建议：如果你的事务中需要锁多个行，要把最可能造成锁冲突、最可能影响并发度的锁尽量往后放。

死锁：当并发系统中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。

解决方案：

1、通过参数 innodb_lock_wait_timeout 根据实际业务场景来设置超时时间，InnoDB引擎默认值是50s。

2、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。将参数 innodb_deadlock_detect 设置为 on，表示开启这个逻辑（默认是开启状态）。

如何解决热点行更新导致的性能问题？

1、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闭掉。一般不建议采用

2、控制并发度，对应相同行的更新，在进入引擎之前排队。这样在InnoDB内部就不会有大量的死锁检测工作了。

3、将热更新的行数据拆分成逻辑上的多行来减少锁冲突，但是业务复杂度可能会大大提高。

innodb行级锁是通过锁索引记录实现的，如果更新的列没建索引是会锁住整个表的。

2018-11-29

作者回复

这个总结

2018-11-29

suynan

16

系列课程看到现在，我能说，这是我看过的最好的mysql课程吗。网上的文章要么漏洞百出小学生水平，要么浅尝辄止并赠送一堆废话，要么千篇一律copy加转载。

说实话，真的被误导过，可怕的不是误导我一人，而是千千万万的菜鸟开发者。

文章很用心，感谢作者，超值！

2019-03-06

作者回复

感谢鼓励

2019-03-10

锅子

13

老师好，关于上一期的问题我有2疑问：

1.Q2，WITH CONSISTENT SNAPSHOT语句执行完可以确保得到一个一致性视图，为什么还会备份到Q2时间点之后更改的表结构啊？如果这样那是不是意味着如果有一个数据库一直有数据在写入的话，备份会一直都无法完成。

2.Q3设置了保存点，之后读到主库的DDL语句，那Q6又回滚到了Q3设置的保存点，那是不是

就主从不一致了啊？

2018-11-28

某、人 10

老师我有几个问题:

1.如何在死锁发生时,就把发生的sql语句抓出来？

2.在使用连接池的情况下,连接会复用.比如一个业务使用连接set sql_select_limit=1,释放掉以后.其他业务复用该连接时,这个参数也生效.请问怎么避免这种情况,或者怎么禁止业务set session？

3.很好奇双11的成交额,是通过redis累加的嘛？

4.不会改源码能成为专家嘛？

2018-11-28

作者回复

1. show engine innodb status 里面有信息，不过不是很全...

2. 5.7的reset_connection接口可以考虑一下

3. 用redis的话，为了避免超卖需要增加了很多机制来保证。修改都在数据库里执行就方便点。前提是要解决热点问题

4. 我认识几位处理问题和分析问题经验非常丰富的专家，不用懂源码，但是原理还是要很清楚的

2018-11-28

杰之7 9

昨天和今天学习了锁相关的知识，对锁有了一定的认识。锁分为全局锁，表锁，行锁三类。全局锁在有事物支持的情况下，使用Mysqldump的single - transaction的方法进行备份时的更新。昨天晚上也问了老师一个关于表锁的疑问，通过老师的解答，理解了write比read的权限高。在今天的学习中，学习了行锁，顾名思义，行锁就是在进行行字段操作时，其他操作不能同时对行进行操作。根据这一问题，提出了两阶段锁协议，并发的锁尽量往后排，这样可以提升并发度。在锁中，还有两种情况，一是死锁，就是事物相互等待，直到等待结束可以通过设置innodb_lock_wait_timed的等待时间来控制。另一种情况是死锁检测，正常情况下是开的，可以对锁进行检测，但老师举了一个1000个并发请求，这样相互等待就是1000的平方量级，导致CPU消耗100%的性能，秒执行不到100行的例子。针对这种问题，老师给予了三个解决方法，把锁关了，不一定是最好，，第二是控制并发度，但有可能有几百个客户端并发并发进行，第三是在逻辑上的多行，当然也需要在程序上对特殊情况下的处理。学习到今天，数据库的核心有难度，但我相信自己可以搞定它，这就是我坚持它的理由。

2018-12-07

作者回复

手动点赞

2018-12-07

Aurora 9

针对第一层楼主提到的问题，我记得是，如果update 没有走索引，innodb内部是全表根据主键索引逐行扫描 逐行加锁，释放锁。

2018-11-28

作者回复

逐行加锁，

事务提交的时候统一释放。– 记得两阶段锁哈

2018-11-28

bing

8

在开发时一般都是按照顺序加锁来避免死锁。比如都是按照先拿t1,再拿t2.

2018-11-28

作者回复

是个好的实践经验

2018-11-30

肉山

8

不考虑数据表的访问并发量，单纯从这个三个方案来对比的话。
第一个方案，一次占用的锁时间较长，可能会导致其他客户端一直在等待资源。
第二个方案，分成多次占用锁，串行执行，不占有锁的间隙其他客户端可以工作，类似于现在多任务操作系统的时间分片调度，大家分片使用资源，不直接影响使用。
第三个方案，自己制造了锁竞争，加剧并发。
至于选哪一种方案要结合实际场景，综合考虑各个因素吧，比如表的大小，并发量，业务对此表的依赖程度等。

2018-11-28

～嗡嗡

6

个人理解，选择第二种
1.直接delete 10000可能使得执行事务时间过长
2.效率慢点每次循环都是新的短事务，并且不会锁同一条记录
3.效率虽高，但容易锁住同一条记录，发生死锁的可能性比较高

2018-12-26

作者回复

对的

2018-12-27

shawn

6

话说留言系统打不出大于小于号是为了防止xss攻击吧，推荐下，能不能用作转义的方式解决呢，留言里小于now()整个没有之后语句不通了
技术向的服务自己的技术得过关吧(手动滑稽)

2018-11-29