Atitit mysql 性能优化

目录

[1. 性能分析工具 1](#_Toc5050)

[1.1. performance\_schema库 （推荐） 1](#_Toc13271)

[1.2. 慢查询日志 1](#_Toc27974)

[1.3. druid日志 可执行sql打印 1](#_Toc17045)

[2. 优化配置 2](#_Toc4359)

[2.1. 事务落盘模式 每秒 2](#_Toc260)

[2.2. Rc隔离级别 2](#_Toc3010)

[2.3. 调大cache 和 2](#_Toc21848)

[2.4. 多线程模式 max conn与 write io theard参数配置 2](#_Toc3418)

[3. 行锁变表锁问题 分库解决 分区 2](#_Toc28792)

[1. 死锁和死锁检测 3](#_Toc2398)

# 性能分析工具

## performance\_schema库 （推荐）

## 慢查询日志

## druid日志 可执行sql打印

# 优化配置

## 事务落盘模式 每秒

## Rc隔离级别

## 调大cache 和

## 多线程模式 max conn与 write io theard参数配置

# 行锁变表锁问题 分库解决 分区

[mysql](https://m.yisu.com/mysql/" \o "mysql" \t "https://m.yisu.com/zixun/_blank):  
[Mysql](https://m.yisu.com/mysql/" \o "Mysql" \t "https://m.yisu.com/zixun/_blank)以表级锁为主，对资源锁定的粒度很大，如果一个session对一个表加锁时间过长，会让其他session无法更新此表中的数据。  
虽然InnoDB引擎的表可以用行级锁，但这个行级锁的机制依赖于表的索引，如果表没有索引，或者sql语句没有使用索引，那么仍然使用表级锁。

oracle:  
oracle使用行级锁，对资源锁定的粒度要小很多，只是锁定sql需要的资源，并且加锁是在数据库中的数据行上，不依赖与索引。所以oracle对并发性的支持要好很多。

# 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线等待的状态，称为死锁。

当出现死锁以后，有两种策略：

1. 一种策略是，直接进入等待，知道超时。这个超时时间可以通过参数 innodb\_lock\_wait\_timeout 来设置。

1. 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InnoDD中，innodb\_lock\_wait\_timeout的默认值是50s，意味着当出现死锁以后，第一个被锁住的线程要过50s才会超时退出，然后其他线程采有可能继续执行。对于在线服务来说，这个等待时间往往是不能接受的，但是如果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。

每个信赖的被堵住的线程都需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，数据库实例的CPU利用率很高，但是每秒却执行不了几个事务。

那么，该如何解决这种由于热点行更新导致的性能问题呢？

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，因为业务涉及的时候一般不会把死锁当成一个严重错误，毕竟出现了死锁就回滚，然后通过业务充实一般就没有问题了，这个业务无损的。而关掉死锁检测意味着可能出现大量的超时，这个业务有损的。

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就很低。一个直接的想法就是在客户端做并发控制。但是，你很快就会发现这个方法不太可行，因为客户端很多。因此，这个并发控制要做在数据库服务端。可以考虑在中间件实现；也可以通过修改MySQL源码，对于相同行的更新，在进入引擎之前排队，这样InnoDB内部就不会有大量的死锁检测工作了。

第三种处理方式，优化设计，将一行改成逻辑上的多行来减少锁冲突。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个记录的值的综合。这样每次要给影院账户加金额的时候，随机选其中一条来加。这样每次冲突的概率变成了原来的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，那么这个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。

作者：FastCoder  
链接：https://www.jianshu.com/p/6312c3995737  
来源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。