

## 23 | MySQL是怎么保证数据不丢的？

2019-01-04 林晓斌



今天这篇文章，我会继续和你介绍在业务高峰期临时提升性能的方法。从文章标题“MySQL是怎么保证数据不丢的？”，你就可以看出来，今天我和你介绍的方法，跟数据的可靠性有关。

在专栏前面文章和答疑篇中，我都着重介绍了WAL机制（你可以再回顾下[第2篇](#)、[第9篇](#)、[第12篇](#)和[第15篇](#)文章中的相关内容），得到的结论是：只要redo log和binlog保证持久化到磁盘，就能确保MySQL异常重启后，数据可以恢复。

评论区有同学又继续追问，redo log的写入流程是怎么样，如何保证redo log真实地写入了磁盘。那么今天，我们就再一起看看MySQL写入binlog和redo log的流程。

### binlog的写入机制

其实，binlog的写入逻辑比较简单：事务执行过程中，先把日志写到binlog cache，事务提交的时候，再把binlog cache写到binlog文件中。

一个事务的binlog是不能被拆开的，因此不论这个事务多大，也要确保一次性写入。这就涉及到了binlog cache的保存问题。

系统给binlog cache分配了一片内存，每个线程一个，参数 binlog\_cache\_size 用于控制单个线程内binlog cache所占内存的大小。如果超过了这个参数规定的大小，就要暂存到磁盘。

事务提交的时候，执行器把binlog cache里的完整事务写入到binlog中，并清空binlog cache。状

态如图1所示。

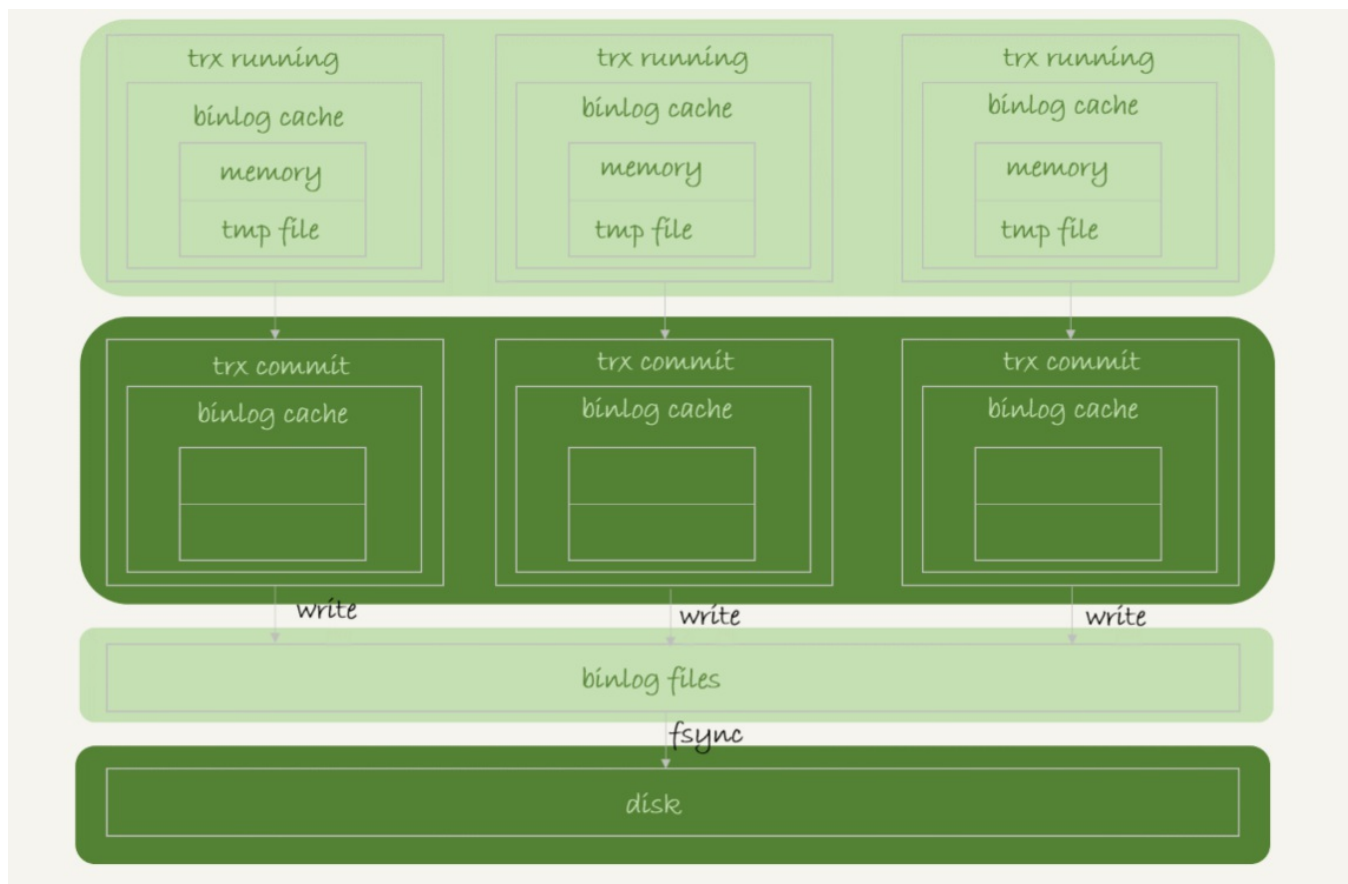


图1 binlog写盘状态

可以看到，每个线程有自己binlog cache，但是共用同一份binlog文件。

- 图中的write，指的就是指把日志写入到文件系统的page cache，并没有把数据持久化到磁盘，所以速度比较快。
- 图中的fsync，才是将数据持久化到磁盘的操作。一般情况下，我们认为fsync才占磁盘的IOPS。

write 和fsync的时机，是由参数sync\_binlog控制的：

1. sync\_binlog=0的时候，表示每次提交事务都只write，不fsync；
2. sync\_binlog=1的时候，表示每次提交事务都会执行fsync；
3. sync\_binlog=N(N>1)的时候，表示每次提交事务都write，但累积N个事务后才fsync。

因此，在出现IO瓶颈的场景里，将sync\_binlog设置成一个比较大的值，可以提升性能。在实际的业务场景中，考虑到丢失日志量的可控性，一般不建议将这个参数设成0，比较常见的是将其设置为100~1000中的某个数值。

但是，将sync\_binlog设置为N，对应的风险是：如果主机发生异常重启，会丢失最近N个事务的binlog日志。

## redo log的写入机制

接下来，我们再说说redo log的写入机制。

在专栏的[第15篇答疑文章](#)中，我给你介绍了redo log buffer。事务在执行过程中，生成的redo log是要先写到redo log buffer的。

然后就有同学问了，redo log buffer里面的内容，是不是每次生成后都要直接持久化到磁盘呢？

答案是，不需要。

如果事务执行期间MySQL发生异常重启，那这部分日志就丢了。由于事务并没有提交，所以这时日志丢了也不会有损失。

那么，另外一个问题是，事务还没提交的时候，redo log buffer中的部分日志有没有可能被持久化到磁盘呢？

答案是，确实会有。

这个问题，要从redo log可能存在的三种状态说起。这三种状态，对应的就是图2 中的三个颜色块。

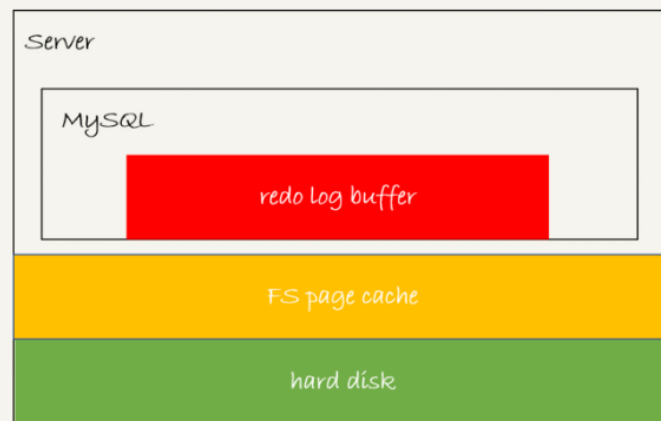


图2 MySQL redo log存储状态

这三种状态分别是：

1. 存在redo log buffer中，物理上是在MySQL进程内存中，就是图中的红色部分；
2. 写到磁盘(write)，但是没有持久化（fsync），物理上是在文件系统的page cache里面，也就

是图中的黄色部分；

3. 持久化到磁盘，对应的是**hard disk**，也就是图中的绿色部分。

日志写到**redo log buffer**是很快的，**write**到**page cache**也差不多，但是持久化到磁盘的速度就慢多了。

为了控制**redo log**的写入策略，InnoDB提供了**innodb\_flush\_log\_at\_trx\_commit**参数，它有三种可能取值：

1. 设置为0的时候，表示每次事务提交时都只是把**redo log**留在**redo log buffer**中；
2. 设置为1的时候，表示每次事务提交时都将**redo log**直接持久化到磁盘；
3. 设置为2的时候，表示每次事务提交时都只是把**redo log**写到**page cache**。

InnoDB有一个后台线程，每隔1秒，就会把**redo log buffer**中的日志，调用**write**写到文件系统的**page cache**，然后调用**fsync**持久化到磁盘。

注意，事务执行中间过程的**redo log**也是直接写在**redo log buffer**中的，这些**redo log**也会被后台线程一起持久化到磁盘。也就是说，一个没有提交的事务的**redo log**，也是可能已经持久化到磁盘的。

实际上，除了后台线程每秒一次的轮询操作外，还有两种场景会让一个没有提交的事务的**redo log**写入到磁盘中。

1. 一种是，**redo log buffer**占用的空间即将达到 **innodb\_log\_buffer\_size**一半的时候，后台线程会主动写盘。注意，由于这个事务并没有提交，所以这个写盘动作只是**write**，而没有调用**fsync**，也就是只留在了文件系统的**page cache**。
2. 另一种是，并行的事务提交的时候，顺带将这个事务的**redo log buffer**持久化到磁盘。假设一个事务A执行到一半，已经写了一些**redo log**到**buffer**中，这时候有另外一个线程的事务B提交，如果**innodb\_flush\_log\_at\_trx\_commit**设置的是1，那么按照这个参数的逻辑，事务B要把**redo log buffer**里的日志全部持久化到磁盘。这时候，就会带上事务A在**redo log buffer**里的日志一起持久化到磁盘。

这里需要说明的是，我们介绍两阶段提交的时候说过，时序上**redo log**先**prepare**，再写**binlog**，最后再把**redo log commit**。

如果把**innodb\_flush\_log\_at\_trx\_commit**设置成1，那么**redo log**在**prepare**阶段就要持久化一次，因为有一个崩溃恢复逻辑是要依赖于**prepare**的**redo log**，再加上**binlog**来恢复的。（如果你印象有点儿模糊了，可以再回顾下[第15篇文章](#)中的相关内容）。

每秒一次后台轮询刷盘，再加上崩溃恢复这个逻辑，InnoDB就认为redo log在commit的时候就不需要fsync了，只会write到文件系统的page cache中就够了。

通常我们说MySQL的“双1”配置，指的就是sync\_binlog和innodb\_flush\_log\_at\_trx\_commit都设置成1。也就是说，一个事务完整提交前，需要等待两次刷盘，一次是redo log（prepare阶段），一次是binlog。

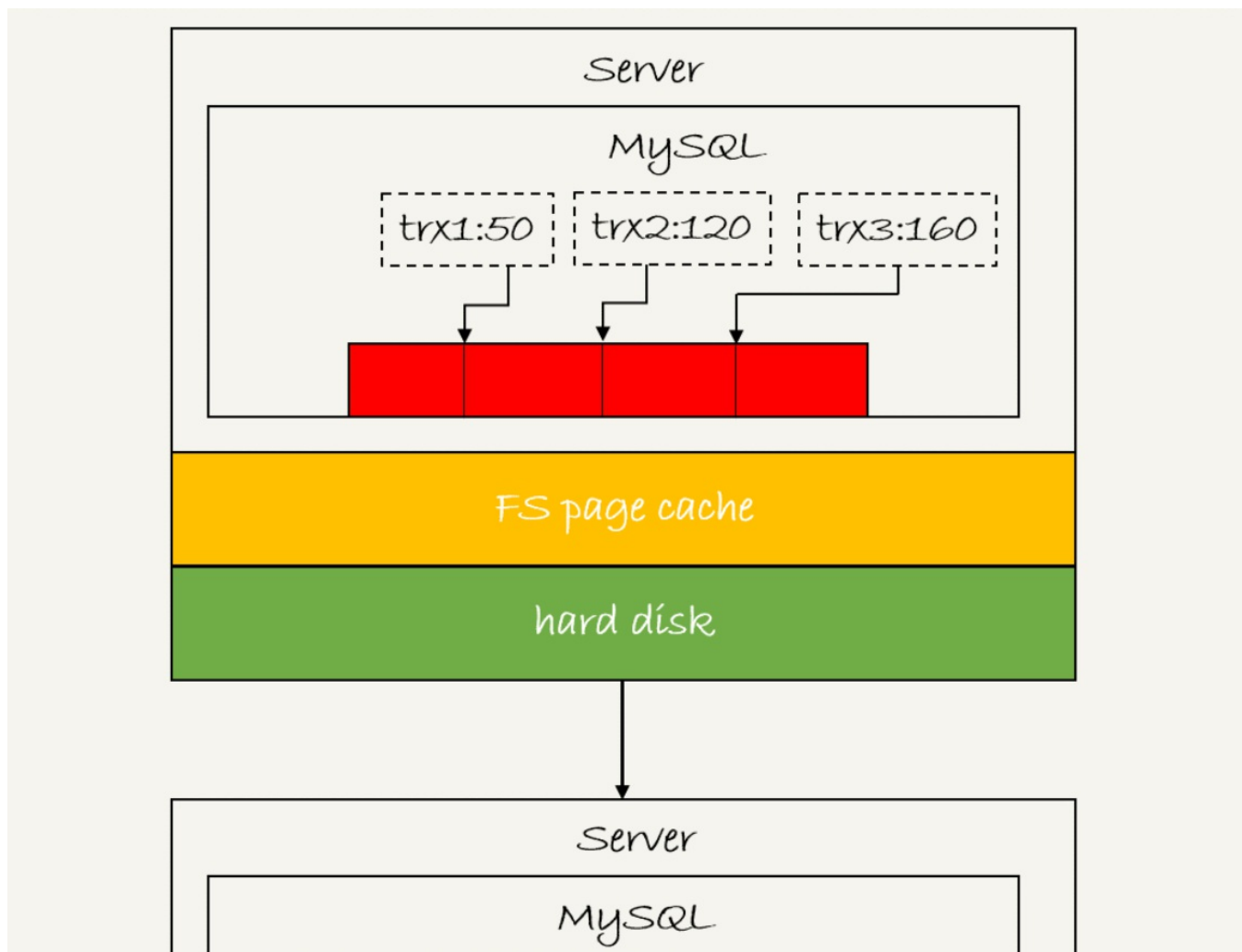
这时候，你可能有一个疑问，这意味着我从MySQL看到的TPS是每秒两万的话，每秒就会写四万次磁盘。但是，我用工具测试出来，磁盘能力也就两万左右，怎么能实现两万的TPS？

解释这个问题，就要用到组提交（group commit）机制了。

这里，我需要先和你介绍日志逻辑序列号（log sequence number, LSN）的概念。LSN是单调递增的，用来对应redo log的一个个写入点。每次写入长度为length的redo log，LSN的值就会加上length。

LSN也会写到InnoDB的数据页中，来确保数据页不会被多次执行重复的redo log。关于LSN和redo log、checkpoint的关系，我会在后面的文章中详细展开。

如图3所示，是三个并发事务(trx1, trx2, trx3)在prepare阶段，都写完redo log buffer，持久化到磁盘的过程，对应的LSN分别是50、120 和160。



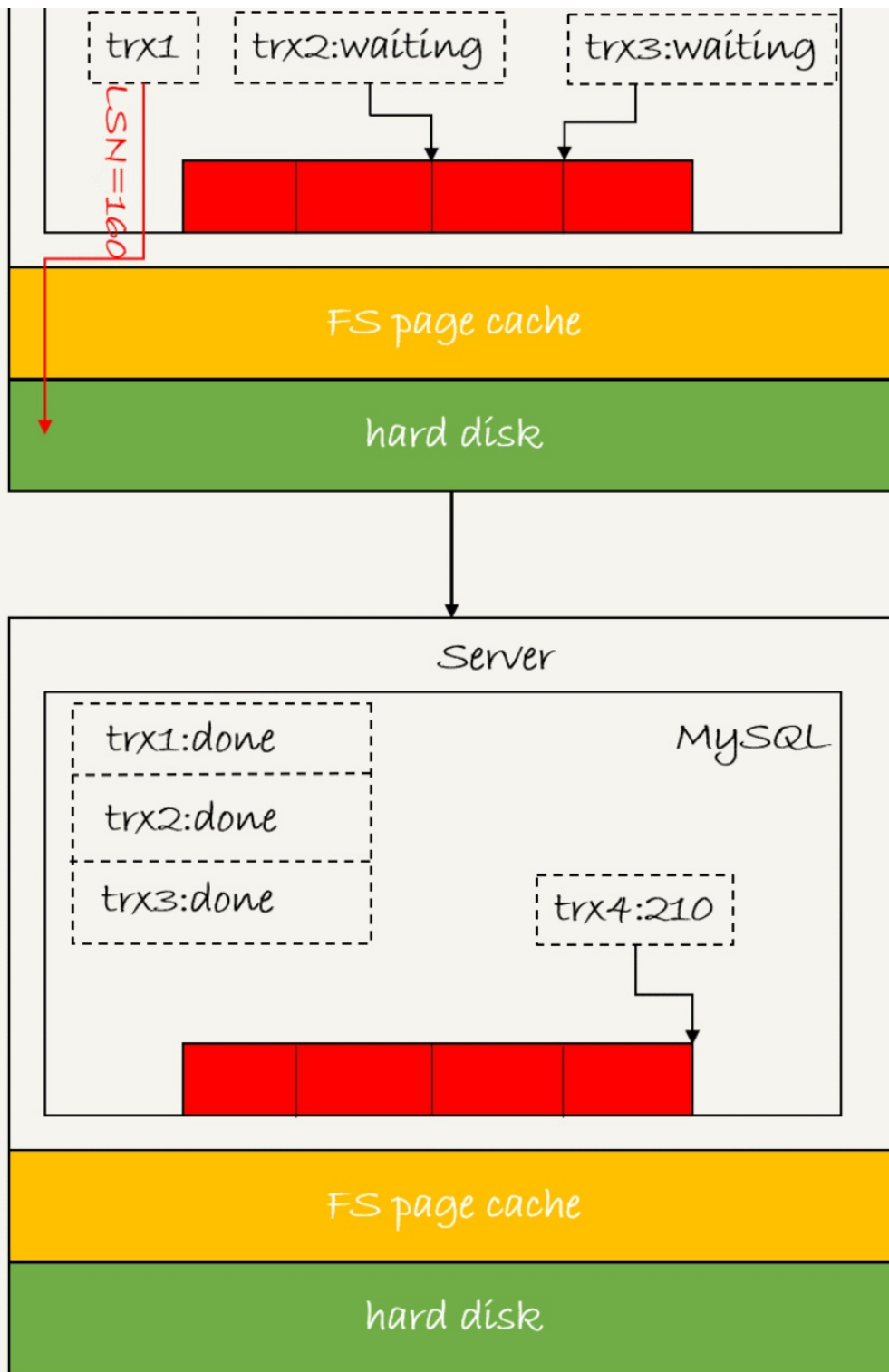


图3 redo log 组提交

从图中可以看到，

1. `trx1`是第一个到达的，会被选为这组的 `leader`;
2. 等`trx1`要开始写盘的时候，这个组里面已经有了三个事务，这时候`LSN`也变成了160;
3. `trx1`去写盘的时候，带的就是`LSN=160`，因此等`trx1`返回时，所有`LSN`小于等于160的redo log，都已经被持久化到磁盘;

4. 这时候trx2和trx3就可以直接返回了。

所以，一次组提交里面，组员越多，节约磁盘IOPS的效果越好。但如果只有单线程压测，那就只能老老实实地一个事务对应一次持久化操作了。

在并发更新场景下，第一个事务写完redo log buffer以后，接下来这个fsync越晚调用，组员可能越多，节约IOPS的效果就越好。

为了让一次fsync带的组员更多，MySQL有一个很有趣的优化：拖时间。在介绍两阶段提交的时候，我曾经给你画了一个图，现在我把它截过来。



图4 两阶段提交

图中，我把“写binlog”当成一个动作。但实际上，写binlog是分成两步的：

1. 先把binlog从binlog cache中写到磁盘上的binlog文件；
2. 调用fsync持久化。

MySQL为了让组提交的效果更好，把redo log做fsync的时间拖到了步骤1之后。也就是说，上面的图变成了这样：



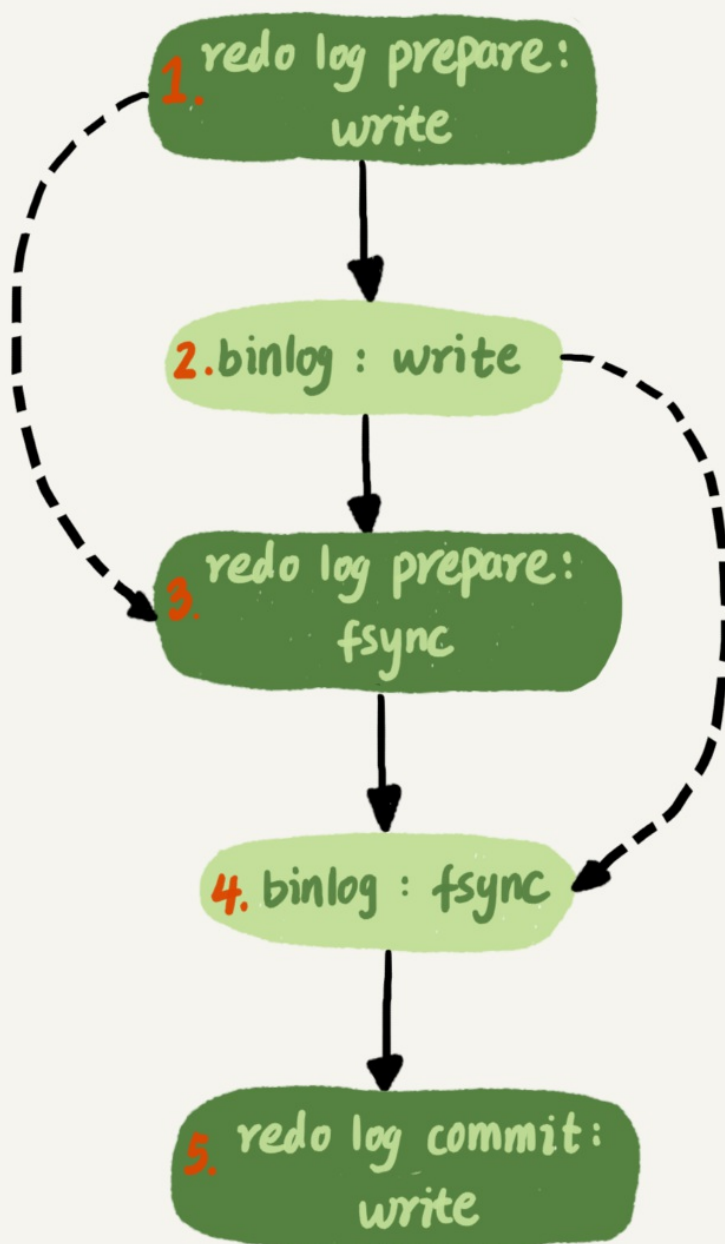


图5 两阶段提交细化

这么一来，binlog也可以组提交了。在执行图5中第4步把binlog fsync到磁盘时，如果有多个事务的binlog已经写完了，也是一起持久化的，这样也可以减少IOPS的消耗。



不过通常情况下第3步执行得会很快，所以binlog的write和fsync间的间隔时间短，导致能集合到一起持久化的binlog比较少，因此binlog的组提交的效果通常不如redo log的效果那么好。

如果你想提升binlog组提交的效果，可以通过设置 binlog\_group\_commit\_sync\_delay 和 binlog\_group\_commit\_sync\_no\_delay\_count来实现。

1. binlog\_group\_commit\_sync\_delay参数，表示延迟多少微秒后才调用fsync;
2. binlog\_group\_commit\_sync\_no\_delay\_count参数，表示累积多少次以后才调用fsync。

这两个条件是或的关系，也就是说只要有一个满足条件就会调用fsync。

所以，当binlog\_group\_commit\_sync\_delay设置为0的时候，binlog\_group\_commit\_sync\_no\_delay\_count也无效了。

之前有同学在评论区问到，WAL机制是减少磁盘写，可是每次提交事务都要写redo log和binlog，这磁盘读写次数也没变少呀？

现在你就能理解了，WAL机制主要得益于两个方面：

1. redo log 和 binlog都是顺序写，磁盘的顺序写比随机写速度要快；
2. 组提交机制，可以大幅度降低磁盘的IOPS消耗。

分析到这里，我们再来回答这个问题：如果你的MySQL现在出现了性能瓶颈，而且瓶颈在IO上，可以通过哪些方法来提升性能呢？

针对这个问题，可以考虑以下三种方法：

1. 设置 binlog\_group\_commit\_sync\_delay 和 binlog\_group\_commit\_sync\_no\_delay\_count参数，减少binlog的写盘次数。这个方法是基于“额外的故意等待”来实现的，因此可能会增加语句的响应时间，但没有丢失数据的风险。
2. 将sync\_binlog 设置为大于1的值（比较常见是100~1000）。这样做的风险是，主机掉电时会丢binlog日志。
3. 将innodb\_flush\_log\_at\_trx\_commit设置为2。这样做的风险是，主机掉电的时候会丢数据。

我不建议你吧innodb\_flush\_log\_at\_trx\_commit 设置成0。因为把这个参数设置成0，表示redo log只保存在内存中，这样的话MySQL本身异常重启也会丢数据，风险太大。而redo log写到文件系统的page cache的速度也是很快的，所以将这个参数设置成2跟设置成0其实性能差不多，但这样做MySQL异常重启时就不会丢数据了，相比之下风险会更小。

## 小结

在专栏的[第2篇](#)和[第15篇](#)文章中，我和你分析了，如果redo log和binlog是完整的，MySQL是如何保证crash-safe的。今天这篇文章，我着重和你介绍的是MySQL是“怎么保证redo log和binlog是完整的”。

希望这三篇文章串起来的内容，能够让你对crash-safe这个概念有更清晰的理解。

之前的第15篇答疑文章发布之后，有同学继续留言问到了一些跟日志相关的问题，这里为了方便你回顾、学习，我再集中回答一次这些问题。

**问题1：**执行一个update语句以后，我再去执行hexdump命令直接查看ibd文件内容，为什么没有看到数据有改变呢？

回答：这可能是因为WAL机制的原因。update语句执行完成后，InnoDB只保证写完了redo log、内存，可能还没来得及将数据写到磁盘。

**问题2：**为什么binlog cache是每个线程自己维护的，而redo log buffer是全局共用的？

回答：MySQL这么设计的主要原因是，binlog是不能“被打断的”。一个事务的binlog必须连续写，因此要整个事务完成后，再一起写到文件里。

而redo log并没有这个要求，中间有生成的日志可以写到redo log buffer中。redo log buffer中的内容还能“搭便车”，其他事务提交的时候可以被一起写到磁盘中。

**问题3：**事务执行期间，还没到提交阶段，如果发生crash的话，redo log肯定丢了，这会不会导致主备不一致呢？

回答：不会。因为这时候binlog也还在binlog cache里，没发给备库。crash以后redo log和binlog都没有了，从业务角度看这个事务也没有提交，所以数据是一致的。

**问题4：**如果binlog写完盘以后发生crash，这时候还没给客户端答复就重启了。等客户端再重连进来，发现事务已经提交成功了，这是不是bug？

回答：不是。

你可以设想一下更极端的情况，整个事务都提交成功了，redo log commit完成了，备库也收到binlog并执行了。但是主库和客户端网络断开了，导致事务成功的包返回不回去，这时候客户端也会收到“网络断开”的异常。这种也只能算是事务成功的，不能认为是bug。

实际上数据库的crash-safe保证的是：

1. 如果客户端收到事务成功的消息，事务就一定持久化了；
2. 如果客户端收到事务失败（比如主键冲突、回滚等）的消息，事务就一定失败了；

3. 如果客户端收到“执行异常”的消息，应用需要重连后通过查询当前状态来继续后续的逻辑。  
此时数据库只需要保证内部（数据和日志之间，主库和备库之间）一致就可以了。

最后，又到了课后问题时间。

今天我留给你的思考题是：你的生产库设置的是“双1”吗？如果平时是的话，你有在什么场景下改成过“非双1”吗？你的这个操作又是基于什么决定的？

另外，我们都知道这些设置可能有损，如果发生了异常，你的止损方案是什么？

你可以把你的理解或者经验写在留言区，我会在下一篇文章的末尾选取有趣的评论和你一起分享和分析。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

我在上篇文章最后，想要你分享的是线上“救火”的经验。

@Long 同学，在留言中提到了几个很好的场景。

- 其中第3个问题，“如果一个数据库是被客户端的压力打满导致无法响应的，重启数据库是没用的。”，说明他很好地思考了。  
这个问题是因为重启之后，业务请求还会再发。而且由于是重启，**buffer pool**被清空，可能会导致语句执行得更慢。
- 他提到的第4个问题也很典型。有时候一个表上会出现多个单字段索引（而且往往这是因为运维工程师对索引原理不够清晰做的设计），这样就可能出现优化器选择索引合并算法的现象。但实际上，索引合并算法的效率并不好。而通过将其中的一个索引改成联合索引的方法，是一个很好的应对方案。

还有其他几个同学提到的问题场景，也很好，很值得你一看。

@Max 同学提到一个很好的例子：客户端程序的连接器，连接完成后会做一些诸如**show columns**的操作，在短连接模式下这个影响就非常大了。

这个提醒我们，在**review**项目的时候，不止要**review**我们自己业务的代码，也要**review**连接器的行为。一般做法就是在测试环境，把**general\_log**打开，用业务行为触发连接，然后通过**general log**分析连接器的行为。

@Manjusaka 同学的留言中，第二点提得非常好：如果你的数据库请求模式直接对应于客户请求，这往往是一个危险的设计。因为客户行为不可控，可能突然因为你们公司的一个运营推广，压力暴增，这样很容易把数据库打挂。

在设计模型里面设计一层，专门负责管理请求和数据库服务资源，对于比较重要和大流量的业务，是一个好的设计方向。

@Vincent 同学提了一个好问题，用文中提到的DDL方案，会导致binlog里面少了这个DDL语句，后续影响备份恢复的功能。由于需要另一个知识点（主备同步协议），我放在后面的文章中说明。



# MySQL 实战 45 讲

从原理到实战，丁奇带你搞懂 MySQL

林晓斌

网名丁奇  
前阿里资深技术专家



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言



锅子

2

老师好，有一个疑问：当设置`sync_binlog=0`时，每次commit都只写进page cache，并不会fsync。但是做实验时binlog文件中还是会有记录，这是什么原因呢？是不是后台线程每秒一次的轮询也会将binlog cache持久化到磁盘？还是有其他的参数控制呢？

2019-01-04

作者回复

你看到的“binlog的记录”，也是从page cache读的哦。

Page cache是操作系统文件系统上的

好问题

2019-01-04



倪大人

4

老师求解`sync_binlog`和`binlog_group_commit_sync_no_delay_count`这两个参数区别

如果

`sync_binlog = N`

`binlog_group_commit_sync_no_delay_count = M`

`binlog_group_commit_sync_delay = 很大值`

这种情况`fsync`什么时候发生呀，`min(N,M)`吗？

感觉`sync_binlog`搭配`binlog_group_commit_sync_delay`也可以实现组提交？

如果

`sync_binlog = 0`

`binlog_group_commit_sync_no_delay_count = 10`

这种情况下是累计10个事务`fsync`一次？

2019-01-04

作者回复

好问题，我写这篇文章的时候也为了这个问题去翻了代码，是这样的：

达到N次以后，可以刷盘了，然后再进入(`sync_delay`和`no_delay_count`)这个逻辑；

`Sync_delay`如果很大，就达到`no_delay_count`才刷；

只要`sync_binlog=0`,也会有前面的等待逻辑，但是等完后还是不调`fsync`！

2019-01-06



WilliamX

3

为什么 `binlog cache` 是每个线程自己维护的，而 `redo log buffer` 是全局共用的？

这个问题，感觉还有一点，`binlog`存储是以`statement`或者`row`格式存储的，而`redo log`是以`page`页格式存储的。`page`格式，天生就是共有的，而`row`格式，只跟当前事务相关

2019-01-04

作者回复

嗯，这个解释也很好。👍

2019-01-04



一大只

2

你是怎么验证的？等于0的时候虽然有走这个逻辑，但是最后调用`fsync`之前判断是0，就啥也没做就走了

回复老师：

老师，我说的`sync_binlog=0`或`=1`效果一样，就是看语句实际执行的效果，参数`binlog_group_commit_sync_delay`我设置成了500000微秒，在`=1`或`=0`时，对表进行`Insert`，然后都会有0.5秒的等待，也就是执行时间都是0.51 sec，关闭`binlog_group_commit_sync_delay`，`insert`执行会飞快，所以我认为`=1`或`=0`都是受组提交参数的影响的。

2019-01-05

作者回复

👍

非常好

然后再补上我回答的这个逻辑，就完备了

2019-01-05



alias cd=rm -rf

👍 1

事务A是当前事务，这时候事务B提交了。事务B的redo log持久化时候，会顺道把A产生的redo log也持久化，这时候A的redo log状态是prepare状态么？

2019-01-28

作者回复

不是。

说明一下哈，所谓的 redo log prepare，是“当前事务提交”的一个阶段，也就是说，在事务A提交的时候，我们才会走到事务A的redo log prepare这个阶段。

事务A在提交前，有一部分redo log被事务B提前持久化，但是事务A还没有进入提交阶段，是无所谓“redo log prepare”的。

好问题

2019-01-28



某、人

👍 1

有调到非双1的时候,在大促时非核心库和从库延迟较多的情况。

设置的是sync\_binlog=0和innodb\_flush\_log\_at\_trx\_commit=2

针对0和2,在mysql crash时不会出现异常,在主机挂了时，会有几种风险：

1.如果事务的binlog和redo log都还未fsync,则该事务数据丢失

2.如果事务binlog fsync成功,redo log未fsync,则该事务数据丢失。

虽然binlog落盘成功,但是binlog没有恢复redo log的能力,所以redo log不能恢复。

不过后续可以解析binlog来恢复这部分数据

3.如果事务binlog fsync未成功,redo log成功。

由于redo log恢复数据是在引擎层,所以重新启动数据库,redo log能恢复数据,但是不能恢复server层的binlog,则binlog丢失。

如果该事务还未从FS page cache里发送给从库,那么主从就会出现不一致的情况

4.如果binlog和redo log都成功fsync,那么皆大欢喜。

老师我有几个问题：

1.因为binlog不能被打断,那么binlog做fsync是单线程吧？

如果是的话,那么binlog的write到fsync的时间,就应该是redo log fsync+上一个事务的binlog fsync时间。

但是测试到的现象,一个超大事务做fsync时,对其它事务的提交影响也不大。

如果是多线程做fsync,怎么保证的一个事务binlog在磁盘上的连续性？

2. 5.7的并行复制是基于binlog组成员并行的,为什么很多文章说是表级别的并行复制？

2019-01-06

## 作者回复

1. Write的时候只要写进去了，**fsync**其实很快的。连续性是**write**的时候做的（写的时候保证了连续）

2. 你的理解应该是对的。不是表级

2019-01-06



永恒记忆

👍 1

主从模式下，内网从库如果设置双1，刚还原的数据发现根本追不上主库，所以从库设置了0，老师后面章节会讲关于**mysql**包括主从监控这块的内容吗。

2019-01-04

## 作者回复

会讲到

2019-01-04



往事随风，顺其自然

👍 1

**redolog** 里面有已经提交事物日志，还有未提交事物日志都持久化到磁盘，此时异常重启，**binlog** 里面不是多余记录的未提交事物，干嘛不设计不添加未提交事物不更好

2019-01-04



miu

👍 0

老师，关于**BINLOG\_GROUP\_COMMIT\_SYNC\_DELAY**，**BINLOG\_GROUP\_COMMIT\_SYNC\_NO\_DELAY\_COUNT**，**SYNC\_BINLOG**三个参数，我的理解是：

若**SYNC\_BINLOG>1**时，且设置了**BINLOG\_GROUP\_COMMIT\_SYNC\_DELAY**和**BINLOG\_GROUP\_COMMIT\_SYNC\_NO\_DELAY\_COUNT**两个参数。

例如

**sync\_binlog=2**，

**BINLOG\_GROUP\_COMMIT\_SYNC\_DELAY=1000000**，

**BINLOG\_GROUP\_COMMIT\_SYNC\_NO\_DELAY\_COUNT=3**，

那么在执行完第1个事务后，在第2个事务提交时，会根据后续的事务提交来判断**fsync**等待的时间，

若后续在1秒内没有累积3个事务的提交，则会等待1秒后再做**fsync**，从SQL语句来看，执行第一个语句很快，第二个语句需要等待1秒才成功。这时延时等待的时间是**BINLOG\_GROUP\_COMMIT\_SYNC\_DELAY**所设置的值。

若执行完第1个事务后，并行执行3个事务（1秒内完成），则后续3个事务会同时做**fsync**，这时延时等待的时间是**BINLOG\_GROUP\_COMMIT\_SYNC\_NO\_DELAY\_COUNT**设置的数量的事务提交的间隔时间。

也就是**sync\_binlog+BINLOG\_GROUP\_COMMIT\_SYNC\_NO\_DELAY\_COUNT-1** 个事务做一次**fsync**。

我测试的版本是MySQL官方5.7.24，请老师点评。

2019-02-01



## 作者回复

这两个逻辑不建议放到一起算

就是按照这样：

1. 有设置 `BINLOG_GROUP_COMMIT_SYNC_NO_DELAY_COUNT` 这个值，（假设 `SYNC_DELAY` 很大），提交的时候就等这么多次才能过；
2. 到了提交阶段，又要按照 `sync_binlog` 来判断是否刷盘。

新春快乐~

2019-02-04



alias cd=rm -rf

👍 0

老师不好意思，我接着刚才的问题问哈

并发事务的 `redolog` 持久化，会把当前事务的 `redolog` 持久化，当前事务的 `redolog` 持久化后 `prepare` 状态么？`redolog` 已经被持久化到磁盘了，那么当前事务提交时候，`redolog` 变为 `prepare` 状态，这时候是从 `redologbuffer` 加载还是从磁盘加载？

2019-01-28

## 作者回复

每个事务在提交过程的 `prepare` 阶段，会把 `redolog` 持久化；“当前事务的 `redolog` 持久化后 `prepare` 状态么”这个描述还是不清楚，你用事务A、事务B这样来描述吧！

`redolog` 已经被持久化到磁盘了，那么当前事务提交时候，

（其实这里只是“部分”被持久化，因为这个事务自己在执行的过程中，还会产生新的日志），只需要继续持久化剩下的 `redo log`

2019-01-28



alias cd=rm -rf

👍 0

您好，我看文章后有俩点疑问，前提条件如果 `mysql` 设置双1

- 1、这时候磁盘中的 `redolog` 的状态是什么状态呢？是 `prepare` 么？
- 2、如果一个事务在进行中的时候 `redolog` 已经被持久化，在事务提交时候，这条 `redolog` 还在 `redolog-buffer` 中么？

2019-01-27

## 作者回复

1. “这时候磁盘中的 `redolog` 的状态是什么状态呢？是 `prepare` 么？”这个“这时候”是什么意思？
2. 还在，不过随时可以被覆盖

2019-01-28



嘻嘻

👍 0

1. 如果客户端收到事务成功的消息，事务就一定持久化了；`commit` 是在什么阶段返回的？如果写完 `page cache` 就返回也没有持久化吧？

2019-01-25

| 作者回复

第一个问题没看懂。

“如果写完page cache就返回也没有持久化吧”，是的，

“客户端收到事务成功的消息，事务就一定持久化了”是建立在双1基础上的。

2019-01-26



Geek\_527020

👍 0

您好，老师，我有一个以后，组提交，把为提交事务的redo log写入磁盘，如果有查询，岂不是查到未提交事务的更新内容了？

2019-01-25

| 作者回复

不会啊，有MVCC的，08篇再看下

2019-01-25



J!

👍 0

共同写一个binlog文件，这个过程应该需要锁来维持提交的时序吧，写文件的时候是不是可能会变成瓶颈点？

2019-01-23

| 作者回复

不会的，大家分头写，然后一起持久化到磁盘

2019-01-23



Komine

👍 0

为什么binlog是不能“被打断的”的呢？主要出于什么考虑？

2019-01-22

| 作者回复

好问题

我觉得一个比较重要的原因是，一个线程只能同时有一个事务在执行。

由于这个设定，所以每当执行一个begin/start transaction的时候，就会默认提交上一个事务；这样如果一个事务的binlog被拆开的时候，在备库执行就会被当做多个事务分段自行，这样破坏了原子性，是有问题的。

2019-01-22



就是个渣渣

👍 0

林老师，你好！超过了binlog\_cache\_size，暂存到磁盘，那如果超过了max\_binlog\_cache\_size就直接报错了呢，这两个参数的关联是什么呢？

2019-01-19

| 作者回复

`max_binlog_cache_size`只是用来限制设置`binlog_cache_size`的时候的上限  
并不参与执行语句的逻辑的

2019-01-19



似水流年

👍 0

我网上查`pagecache`是在内存里的，这与您讲的一样吗？

2019-01-15

作者回复

就是文件系统的`page cache`，是属于操作系统的内存的一部分

2019-01-15



猪哥哥

👍 0

老师好，能说下`innodb_log_buffer_size`参数的作用吗

2019-01-10



roaming

👍 0

看了几遍，终于看明白了

2019-01-10

作者回复

👍

2019-01-10



猪哥哥

👍 0

老师 我想问下文件系统的`page cache`还是不是内存，是不是文件系统向内核申请的一块内存？

2019-01-10

作者回复

你理解的是对的

2019-01-10