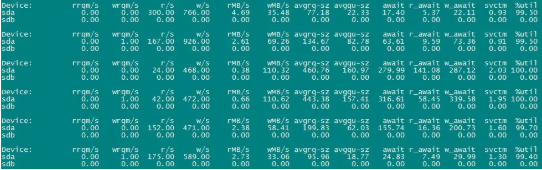
# 磁盘IO高

## 问题描述

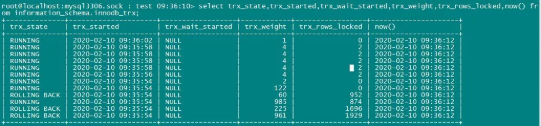
线上数据库突发严重告警，业务方反馈写入数据一直堵住，很多锁超时回滚了。

1. 系统负载，主要是磁盘I/O的负载数据

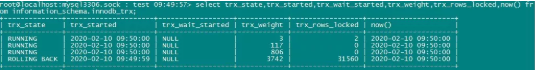


该服务器的磁盘是由6块2T SSD硬盘组成的RAID-5阵列。从上面的截图来看，I/O %util已经基本跑满了，iowait也非常高，很明显磁盘I/O压力太大了。那就再查查什么原因导致的这么高压力。

1. 活跃事务列表



可以看到，有几个活跃的事务代价很高，锁定了很多行。其中有两个因为太久超时被回滚了。



再看一次活跃事务列表，发现有个事务锁定的行更多了，说明活跃业务SQL的效率不太好，需要进行优化。这个算是原因之一，先记下。

3、查看InnoDB状态

执行 SHOW ENGINE INNODB STATUS\G 查看InnoDB状态，这里只展示了几个比较关键的地方：

4、查看MySQL的线程状态\*

+---------+------+--------------+---------------------

| Command | Time | State | Info |

+---------+------+--------------+---------------------

| Query | 1 | update | insert xxx

| Query | 0 | updating | update xxx

| Query | 0 | updating | update xxx

| Query | 0 | updating | update xxx

| Query | 0 | updating | update xxx

+---------+------+--------------+---------------------

可以看到几个事务都处于updating状态。意思是正在扫描数据并准备更新，肉眼可见这些事务状态时，一般是因为系统负载比较高，所以事务执行起来慢；或者该事务正等待行锁释放。

## 问题分析

# 线上IO问题

## 问题描述

某业务CDB实例，每天在特地时间段内（00:07:00 - 00:08:00左右）机器对应IO监控出现写入尖刺，且主从实例都有类似现象，从机器监控可以看到，问题确实存在。

不仅master，进行同步的slave上有相同的现象，业务方希望找到导致该IO尖刺问题稳定出现的原因。

## 问题分析

首先确定问题来源，上图所示监控为机器级别，机器IO写入负载是否来源于mysqld进程？如果来源于mysqld进程，是来自于mysqld进程的哪一部分写入操作引起？

为了获取IO来源，在slave机上部署mysqld实例监控，以及iotop采集监控，获取对应时间段更详细的相关信息，抓取对应时间段进行IO写入的进程（线程），同时观察对应时间段mysql实例状态。

这段时间内的较大IO写入线程号为：（截取部分记录）

时间 线程号 进程名 读取速度 写入速度

00:07:34 145378 be/4 mysql201 139.10 K/s 263111.57 K/s

00:07:35 145378 be/4 mysql201 124.11 K/s 249703.84 K/s

00:07:36 145378 be/4 mysql201 120.23 K/s 289920.70 K/s

00:07:39 145378 be/4 mysql201 5168.09 K/s 875194.69 K/s

通过slave上iotop采集到的统计信息，可以看出较大写入来源为145378这一线程，确实来自于mysql进程，该时间段内没有抓到其他大量写入的记录，同时该实例slave mysql为单机独占，可以基本确定写入来源为mysql中145378这个线程，那么这个线程是哪一个线程呢？

145378这个线程号正是mysqld slave的线程，而且为单线程回放的sql线程。

Thread 85 (Thread 0x7f68c4c4c700 (LWP 145378)):

#0 0x00007fa2badd3945 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib64/libpthread.so.0

#4 exec\_relay\_log\_event (rli=0x1771c43c8, thd=0x7f68b0000990)

#5 handle\_slave\_sql (arg=0x1771c3010)

通过进一步搜索监控记录，可以发现，其实sql线程引入大量写入IO不仅存在于这个时间段，在其他时间段也有较高写入的记录（超过100MB/s），在00:08:00左右持续时间相对较长。这个现象反应出该问题可能贯穿于整个执行过程，只是其他时间段没有这么明显。

通过mysql实例的监控可，可以看到mysql主要进行的操作为insert，slave mysql主要在进行单线程回放，执行这些insert操作，操作数量会有一些增长，每秒操作数不稳定，峰值可能达到4000左右。

从对于slave实例的新增监控可以得到结论，这段时间的主要写入来源确实为mysqld进程，且为mysqld的sql线程，那么问题转换为：

在该业务模型下，进行单线程回放的mysql slave sql thread， 为什么会在一些时段产生较大的写入IO？产生这个现象的时候，mysql在做什么事情？

为了更好的分析问题，同时不影响线上业务，在另外一台实验机器上单独搭建mysql 5.7的slave，连到源实例的master作为slave，问题现象也可以复现，为了获取更多信息，打开mysql 5.7的performance\_schema，在实验机器的slave机监控上，依然能看到该问题存在。

现在我们需要分析一下，SQL线程回放，可能产生哪些IO写（注意其他线程的IO不会记录到SQL线程头上，例如page cleaner flush），一条SQL语句回放过程中，可能经历的路径上有哪些操作会引发IO操作。

一个slave mysql读取relay log进行日志回放，首先想到回放执行语句，可能由此引发下列写入IO：

mysql server binlog日志记录，即回放过程中语句写入的本地binlog。

innodb redo undo日志记录。

用户态page flush（free page吃紧，需要自行flush获取可用free page）。

relay log purge，删除文件。默认每当slave回放完一个完整relay log，会将对应文件进行删除。

那么，为了找到真正引发写入的来源，需要对于上述来源逐步进行分析、排除，对可能原因进行验证。

首先在5.7 slave上配置log\_slave\_updates=OFF，使得回放过程不记录binlog日志，问题现象依旧，排除binlog写入。

分钟级监控显示，master实例在对应时间段insert数量有一定幅度增加：

为了保证free page够用，调大了buffer pool，确保free page足够使用，另外关闭relay log purge功能，使得SQL线程不会触发删除relay log文件，问题现象依旧，排除清理relay以及用户态page flush。

通过sys schema统计值发现，对于文件写入，count write统计值较大的记录主要来源于redo log（例如ib\_logfile0写入7439952次，且总量为57.47GB），但是innodb自身对于log写入的单位时间统计值显示却不大。

/data1/mysql\_root/log/20120/ib\_logfile0 5 4.00 KiB 819 bytes 7439952 57.47 GiB 8.10 KiB 57.47 GiB 100.00

/data1/mysql\_root/log/20120/ib\_logfile1 2 64.50 KiB 32.25 KiB 3025254 23.39 GiB 8.11 KiB 23.39 GiB 100.00

为了进一步排除干扰，修改mysql实现关闭redo log写入，替换mysql 5.7实验版本，统计值显示确实redo文件几乎没有写入增长，问题依旧，排除redo写入造成，再看ibdata相关记录增长，也非常有限，也可以排除undo文件写入。

到目前为止，通常能想到的用户态写入（例如sql线程回放执行一个事务），都可以排除掉，还有什么原因可以sql线程造成大量写入呢？需要重新整理一下思路。

再回到业务本身，看一下业务的库表结构模型和数据规模，表结构大体如下：

FRFrom int(10) unsigned NOT NULL DEFAULT ‘0’,

FRtBody mediumblob NOT NULL,

FPSQL varchar(20480) NOT NULL,

FBody mediumblob NOT NULL,

FResTime bigint(20) NOT NULL DEFAULT ‘0’,

FCime bigint(20) NOT NULL DEFAULT ‘0’,

FInt1 int(10) unsigned NOT NULL DEFAULT ‘0’,

FInt2 int(10) unsigned NOT NULL DEFAULT ‘0’,

FInt3 int(10) unsigned NOT NULL DEFAULT ‘0’,

FInt4 int(10) unsigned NOT NULL DEFAULT ‘0’,

FInt5 int(10) unsigned NOT NULL DEFAULT ‘0’,

FChar1 varchar(256) NOT NULL DEFAULT ‘’,

FChar2 varchar(256) NOT NULL DEFAULT ‘’,

FChar3 varchar(512) NOT NULL DEFAULT ‘’,

FChar4 varchar(1024) NOT NULL DEFAULT ‘’,

FChar5 varchar(1024) NOT NULL DEFAULT ‘’,

FExt blob,

FGrp2EvtNo bigint(20) DEFAULT NULL,

PRIMARY KEY (FEventNo)

表数量很多，超过1W张表，且单表数据量不大

表结构没有其他索引，只有主键

包含blob字段

从relay log分析插入语句，blob每次都有较大的数据量进行插入

表数量较多，插入操作，blob字段，并发插入且插入操作分散到各个表，这几点看起来有些关联。在关掉了包括binlog，relay purge，redo log等多个写入之后，再抓一下sql线程回放的堆栈，看一下写入调用的来源：

#0 os\_aio\_func (type=..., mode=mode@entry=24,

name=0x7ee4493f97b8 "./DB\_xxx/xxx\_36.ibd", file=..., buf=buf@entry=0x7ee453ed4000, offset=offset@entry=122683392, n=n@entry=1048576, read\_only=read\_only@entry=false, m1=m1@entry=0x0, m2=m2@entry=0x0)

#1 0x0000000001187922 in fil\_write\_zeros (node=0x7ee4493f5338, node=0x7ee4493f5338, read\_only\_mode=<optimized out>,

len=4194304, start=121634816, page\_size=<optimized out>)

#2 fil\_space\_extend (space=space@entry=0x7ee448d8dbd8, size=<optimized out>)

#3 0x00000000007701d1 in fsp\_try\_extend\_data\_file (space=space@entry=0x7ee448d8dbd8, header=header@entry=0x7ef4a4ff0026 "",

#4 0x000000000118f0df in fsp\_reserve\_free\_extents (n\_reserved=n\_reserved@entry=0x7f20868f4fa0, space\_id=9054, n\_ext=3,

alloc\_type=alloc\_type@entry=FSP\_NORMAL, mtr=mtr@entry=0x7f20868f5890, n\_pages=n\_pages@entry=2)

#5 0x00000000010e6394 in btr\_cur\_pessimistic\_insert (flags=flags@entry=0, cursor=cursor@entry=0x7f20868f5150,

offsets=offsets@entry=0x7f20868f50b0, heap=heap@entry=0x7f20868f50a0, entry=entry@entry=0x7ee453143488,

rec=rec@entry=0x7f20868f5570, big\_rec=big\_rec@entry=0x7f20868f5090, n\_ext=n\_ext@entry=0, thr=thr@entry=0x7ee4544623f0,

mtr=mtr@entry=0x7f20868f5890)

回顾一下基于主键索引的插入操作，对于b+树，如果插入的record的较大（例如很大的blob），可能会触发分裂操作。类似

对于innodb而言，插入的过程中，首先尝试乐观插入索引，如果空间大小不够，再尝试悲观插入，悲观插入首先保证表空间大小足够（ibd文件 innodb\_file\_per\_table=ON，每个表对应一个文件），这里用户场景，正是每个表一个ibd文件，那么16000张表总共有16000个ibd文件。如果空间不够，尝试对于ibd文件进行扩展，扩展逻辑如下：

每次扩展4个extent（每个extent包含 16kb\*64 = 1MB大小供64个data page），即每次扩展形成4MB，业务场景包含16000多个表，且每次插入数据量相对固定，表结构相同，插入目标表随机分散，所以很多表大小，和后续操作，非常均衡可以看作是齐头并进。

从slave实例监控来看，insert在对应时间段超过1000个每秒，最多可以达到4000个每秒。假设相同数据量模型，其中500个表同时扩展大小，这一秒内，可能同时产生500\*4MB = 2GB左右的写入。为了印证这个写入来源，继续在innodb层添加日志跟踪，对于idb文件扩展加入以下逻辑进行日志跟踪：

每100MB扩展数据量，查看累计时间，如果累计时间在1秒以内，打印一条warning日志，且sleep 0.5s。

查看对应时间段新增日志（+8为北京时间），

对应时间扩展很频繁，除去sleep时间，大约0.1-0.2秒能够产生100MB的扩展写入，IO尖刺数据量基本吻合，与iotop抓取到的大io写入也基本吻合。至此我们基本可以得出问题结论。

## 问题结论

业务模型比较特殊，多个包含BLOB字段的小表（超过16000），业务模型为insert，写入目标表分散，在某些时间段会并发insert（平均2000个左右每秒，峰值4000个每秒），由于BLOB字段占用空间较大，各个表使用分散，导致频繁同时触发分裂，进而导致底层data page扩展，使得底层多个ibd文件同时并发extend，初始化data page产生某些时间段较大IO，反应到机器监控上为某些时间段IO尖刺

## 解决方法

业务层面，对于类似场景，考虑合并表数量，减少并发扩展带来的写入压力，可以一定程度缓解IO尖刺。

MySQL层面，考虑更加智能的数据文件扩展算法，适配上述场景。减少扩展的并发性。

MySQL层面，考虑用户指定初始化表空间大小，提前预分配和初始化，避免动态扩展。