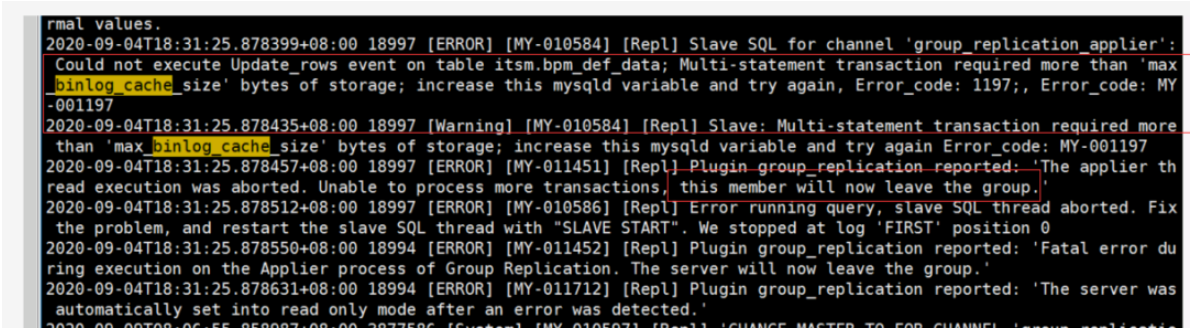
第二十六节：修改max_binlog_cache_size参数导致MGR集群异常---实战篇

一、问题来源

因为前期设置max_binlog_cache_size为8m，后面在线进行了修改本参数，但是结果导致整个3节点的MGR集群除了primary节点其他两个second节点均掉线。大概的日志如下：



二、使用binlog cache的大概流程

- 开启读写事务。
- 执行'DML'语句，在'DML'语句第一次执行的时候会分配内存空间给binlog cache缓冲区。
- 执行'DML'语句期间生成的Event不断写入到binlog cache缓冲区。
- 如果binlog cache缓冲区已经写满了，则将binlog cache缓冲区的数据写入到binlog cache临时文件，同时清空binlog cache缓冲区，这个临时文件名以ML开头。
- 事务提交，binlog cache缓冲区和binlog cache临时文件数据全部写入到binary log中进行固化，释放binlog cache缓冲区和binlog cache临时文件。但是注意此时binlog cache缓冲区的内存空间留用供下次事务使用，但是binlog cache临时文件被截断为0，保留文件描述符。其实也就是IO_CACHE结构保留，并且保留IO_CACHE中分配的内存空间和临时文件文件描述符。
- 断开连接，这个过程会释放IO_CACHE同时释放其持有的binlog cache缓冲区内存以及持有的binlog cache临时文件。

三、max_binlog_cache_size参数的作用

max_binlog_cache_size：修改需要使用set global进行修改，定义了binlog cache临时文件的最大容量。如果某个事务的Event总量大于了（max_binlog_cache_size+binlog_cache_size）的大小那么将会报错，如下：

```
ERROR 1197 (HY000): Multi-statement transaction required more than
'max_binlog_cache_size' bytes of storage; increase this mysqld variable
and try again
```

在函数_my_b_write可以看到如下代码：

```
if (pos_in_file+info->buffer_length > info->end_of_file) //判断binlog cache临时文件的位置加上本次需要写盘的数据大于info->end_of_file的大小则抛错
  {
    errno=EFBIG;
    set_my_errno(EFBIG);
    return info->error = -1;
  }
```

其中info->end_of_file的大小正是来自于我们的参数max_binlog_cache_size。

四、分析问题

从second节点的报错来看，是applier线程应用的事务超过了max_binlog_cache_size设置的大小，但是已经修改了其大小，并且主库并没有报这个错误。
我们知道MGR applier线程从启动MGR的那一刻开始就不会停止，类似的master-slave的sql线程也是一样，我们修改参数是通过set global修改的参数，但是实际上在对于MGR的applier线程并不会生效。
但是对于主库来讲，我们修改参数后只要重启应用重新连接那么参数就生效了，这个时候实际上primary session的max_binlog_cache_size和second applier的max_binlog_cache_size并不一致，一旦有主库做一个稍大的事务，如果这个事务的binlog大于以前设置的值，主库虽然能成功，但是备节点就会由于applier线程的max_binlog_cache_size过小而导致备节点脱离整个集群。
对于这一点我们可以通过debug MySQL的sql线程进行验证。

五、验证

这里我们使用master-slave来进行验证，我们对sql线程进行debug。如下，

- 当前配置

```
mysql> show variables like '%binlog_cache%';
+-----------------------+----------+
| Variable_name         | Value    |
+-----------------------+----------+
| binlog_cache_size     | 4096     |
| max_binlog_cache_size | 39997440 |
+-----------------------+----------+
2 rows in set (0.01 sec)

mysql>
```

就绪

sql线程

```
|     66 |        26534 | sql/slave_io                                | FOREGROUND |
|     67 |        26535 | sql/slave_sql                               | FOREGROUND |
+--------+--------------+---------------------------------------------+------------+
52 rows in set (0.28 sec)
```

修改参数

```
mysql> set global max_binlog_cache_size=50000000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show global variables like '%max_binlog_cache_size%';
+-----------------------+----------+
| Variable_name         | Value    |
+-----------------------+----------+
| max_binlog_cache_size | 49999872 |
+-----------------------+----------+
1 row in set (0.00 sec)
```

主库执行一个事务，从库执行
我们可以查看sql线程binlog cache的IO CACHE的信息如下：

```
(gdb) p cache_data->m_cache->m_file->m_max_cache_size
$12 = 39997440
(gdb) info threads
  Id   Target Id         Frame
* 68   Thread 0x7fff61881700 (LWP 26535) "mysqld" MYSQL_BIN_LOG::write_transaction (this=
    at /opt/mysql/mysql-8.0.21/sql/binlog.cc:1479
  67   Thread 0x7fff618c9700 (LWP 26534) "mysqld" 0x00007ffff5e2dcff in ppoll () from /
```

可以看到这个值还是老值。

- 重启后sql线程后，主库再做一个事务观察

```
#1  0x0000 7ffff3e388dd in clone () from /lib64/libc.so.6
(gdb) p cache_data->m_cache->m_file->m_max_cache_size
$13 = 49999872
(gdb)
```

很明显我们刚才修改的值重启sql线程后才生效。