[MYSQL] 出现大量的Waiting for table flush导致业务表查询不了

原创 大大刺猬 大大刺猬 2025年06月27日 18:10 上海

导读

昨晚发现有大量的sql执行失败，使用`show processlist`发现存在大量的`Waiting for table flush`状态的连接.如下图(下图为复现环境)

```
(root@127.0.0.1) [performance_schema]> show processlist;
+----+------+----------------+--------------------+---------+------+-------------------------+-------------------------------
-----------------------------------------------------------------+
| Id | User | Host           | db                 | Command | Time | State                   | Info
                                                                 |
+----+------+----------------+--------------------+---------+------+-------------------------+-------------------------------
-----------------------------------------------------------------+
| 16 | root | 127.0.0.1:39136 | db1                | Query   |  750 | Sending data            | select count(*) from t2025062
as a,t20250627 as b where a.c2 in (select c1 from t20250627 order by  |
| 17 | root | 127.0.0.1:39138 | db1                | Query   |  735 | Waiting for table flush | select * from t20250627 limit
                                                                 |
| 18 | root | 127.0.0.1:39140 | db1                | Query   |  724 | Waiting for table flush | select * from t20250627 limit
                                                                 |
| 19 | root | 127.0.0.1:39142 | performance_schema | Query   |    0 | starting                | show processlist
                                                                 |
+----+------+----------------+--------------------+---------+------+-------------------------+-------------------------------
-----------------------------------------------------------------+
4 rows in set (0.00 sec)

(root@127.0.0.1) [performance_schema]>
```


就这么几个连接，排除自己，就只剩一个select的了，总不能是select导致的其它表`Waiting for table flush`吧，我们再看下mdl相关信息:

```
-- 如果没有开启mdl，可以使用如下sql开启
-- update performance_schema.setup_instruments set ENABLED='YES' where name
='wait/lock/metadata/sql/mdl';

-- 查询mdl锁
select * from performance_schema.metadata_locks;
```

```
(root@127.0.0.1) [performance_schema]> select * from performance_schema.metadata_locks;
+-------------+-------------------+----------------+----------------------+-------------+--------------+-------------+------
---+----------------+---------------+
| OBJECT_TYPE | OBJECT_SCHEMA     | OBJECT_NAME    | OBJECT_INSTANCE_BEGIN | LOCK_TYPE   | LOCK_DURATION | LOCK_STATUS | SOUR
CE | OWNER_THREAD_ID | OWNER_EVENT_ID |
+-------------+-------------------+----------------+----------------------+-------------+--------------+-------------+------
---+----------------+---------------+
| TABLE       | db1               | t20250627      |       140274303048624 | SHARED_READ | TRANSACTION   | GRANTED     |
   |              41 |             11 |
| TABLE       | db1               | t20250627      |       140274437224704 | SHARED_READ | TRANSACTION   | GRANTED     |
   |              42 |             12 |
| TABLE       | db1               | t20250627      |       140274235897232 | SHARED_READ | TRANSACTION   | GRANTED     |
   |              43 |             11 |
| TABLE       | performance_schema | metadata_locks |       140274371794624 | SHARED_READ | TRANSACTION   | GRANTED     |
   |              44 |            105 |
+-------------+-------------------+----------------+----------------------+-------------+--------------+-------------+------
---+----------------+---------------+
```


都是SHARED_READ啊

初步分析

首先我们来看下什么情况会产生`Waiting for table flush`，查询官网 发现: **FLUSH TABLES**, FLUSH TABLES tbl_name, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE 均会导致产生这个状态.

第一个命令(FLUSH TABLES)是不是看着很眼熟？就是我们使用mysqldump备份时会执行的，而且也会因为存在大事务，导致其状态变为`Waiting for table flush`，于是我们查看下备份时间点，恰好能对得上. **说明就是我们备份导致的业务查询失败，问题就解决了!**

且慢，我们备份已经跑完了啊，而且也失败了. 那为啥还是会出现大量的`Waiting for table flush`，而且即使是新连接进来查询该表也会出现这个状态.

这种情况我们通常会猜测: 后面的查询都堵在某个队列里面，只要最原始的sql不执行完，后面的就永远只能堵着. 那么`kill`掉第一个查询，后面的sql就能跑完. 我们来验证下:

```
+----+------+----------------+--------------------+---------+------+-------------------------+-----------------------------
-------------------------------------------------------------------+
| 16 | root | 127.0.0.1:39136 | db1                | Query   | 1284 | Sending data            | select count(*) from t20250627
 as a,t20250627 as b where a.c2 in (select c1 from t20250627 order by  |
| 17 | root | 127.0.0.1:39138 | db1                | Query   | 1269 | Waiting for table flush | select * from t20250627 limit
1                                                                    |
| 18 | root | 127.0.0.1:39140 | db1                | Query   | 1258 | Waiting for table flush | select * from t20250627 limit
1                                                                    |
| 19 | root | 127.0.0.1:39142 | performance_schema | Query   |    0 | starting                | show processlist
                                                                     |
+----+------+----------------+--------------------+---------+------+-------------------------+-----------------------------
-------------------------------------------------------------------+
4 rows in set (0.00 sec)

(root@127.0.0.1) [performance_schema]> kill 16;
Query OK, 0 rows affected (0.00 sec)

(root@127.0.0.1) [performance_schema]> show processlist;
+----+------+----------------+--------------------+---------+------+-------------+--------------------------------------
-------------------------------------------------------------+
| Id | User | Host           | db                 | Command | Time | State       | Info
                                                             |
+----+------+----------------+--------------------+---------+------+-------------+--------------------------------------
-------------------------------------------------------------+
| 17 | root | 127.0.0.1:39138 | db1                | Sleep   | 1285 |             | NULL
                                                             |
| 18 | root | 127.0.0.1:39140 | db1                | Sleep   | 1274 |             | NULL
                                                             |
| 19 | root | 127.0.0.1:39142 | performance_schema | Query   |    0 | starting    | show processlist
                                                             |
| 22 | root | 127.0.0.1:39156 | db1                | Query   |    8 | Sending data | select count(*) from t20250627 as a,t2025
0627 as b where a.c2 in (select c1 from t20250627 order by  |
+----+------+----------------+--------------------+---------+------+-------------+--------------------------------------
-------------------------------------------------------------+
4 rows in set (0.00 sec)

(root@127.0.0.1) [performance_schema]>
```


当我们Kill掉"罪魁祸首"之后，后面的sql确实执行成功了．那真的有这么个队列吗？总感觉有点不太对

复现

在我们深入分析前，我们先复现下问题吧．刚才都分析清楚了,是有大事务的时候,执行`flush tables`就会导致后续该表的查询堵着．

```
-- 数据准备
dropprocedureifexists pro_insert_nrows;
delimiter //
createprocedure pro_insert_nrows( IN rows1 int)
begin
declare n int;
set n=1;
set autocommit=off;
while n <= rows1
do
insertinto t20250627 values(n,n,md5(n));
set n=n+1;
endwhile;
commit;
end//
delimiter ;

createtable t20250627(c1 int,c2 int,c3 varchar(200));
createtable t20250627_2(c1 int,c2 int,c3 varchar(200));
createtable t20250627_3(c1 int,c2 int,c3 varchar(200));
call pro_insert_nrows(200000);
insertinto t20250627 select * from t20250627;
insertinto t20250627 select * from t20250627;
insertinto t20250627 select * from t20250627;
```

session 1: 模拟大事务

```
selectcount(*) from t20250627 as a,t20250627 as b where a.c2 in (select c1 fr
om t20250627 orderby c3);
```

session 2: 模拟备份

```
flushtables
```

session 3: 模拟正常业务查询

```
select * from t20250627  limit1
```

> 用sleep模拟大事务应该也是可以的.

再次分析

这次我们使用gdb查看堆栈信息, 先看下堵塞连接的线程id:

```
select * from performance_schema.threads whereNAME='thread/sql/one_connectio
n'
```

```
*************************** 2. row ***************************
          THREAD_ID: 43
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 18
   PROCESSLIST_USER: root
   PROCESSLIST_HOST: 127.0.0.1
     PROCESSLIST_DB: db1
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 8
  PROCESSLIST_STATE: Waiting for table flush
   PROCESSLIST_INFO: select * from t20250627 limit 1
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: SSL/TLS
       THREAD_OS_ID: 26830
*************************** 3. row ***************************
```


然后我们使用gdb来查看下该线程的堆栈信息

```
gdb -p `pidof mysqld`
```

使用`info thread`查看线程信息, 然后使用`thread n`切换到我们刚才查询到的堵塞的线程, 然后使用`bt`查看堆栈信息

```
  from /lib64/libpthread.so.0
  6    Thread 0x7f9475430700 (LWP 25510) "mysqld" 0x0000000000c75b5b in Item_sum_count::add (this=
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/item_sum.cc:1690
  5    Thread 0x7f94753ee700 (LWP 26715) "mysqld" 0x00007f95632ccc3d in poll () from /lib64/libc.s
  4    Thread 0x7f94753ac700 (LWP 26720) "mysqld" 0x00007f95632ccc3d in poll () from /lib64/libc.s
  3    Thread 0x7f947536a700 (LWP 26830) "mysqld" 0x00007f9564823de2 in pthread_cond_timedwait@@GL
  from /lib64/libpthread.so.0
  2    Thread 0x7f9475328700 (LWP 27943) "mysqld" 0x00007f9564823a35 in pthread_cond_wait@@GLIBC_2
  from /lib64/libpthread.so.0
* 1    Thread 0x7f9564c42780 (LWP 25474) "mysqld" 0x00007f95632ccc3d in poll () from /lib64/libc.s
(gdb) thread 3
[Switching to thread 3 (Thread 0x7f947536a700 (LWP 26830))]
#0  0x00007f9564823de2 in pthread_cond_timedwait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
(gdb) bt
#0  0x00007f9564823de2 in pthread_cond_timedwait@@GLIBC_2.3.2 () from /lib64/libpthread.so.0
#1  0x0000000000cb8da0 in native_cond_timedwait (abstime=0x7f9475367fc0, mutex=0x7f9424015348, cor
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/include/thr_cond.h:136
#2  my_cond_timedwait (abstime=0x7f9475367fc0, mp=0x7f9424015348, cond=0x7f9424015378)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/include/thr_cond.h:189
#3  inline_mysql_cond_timedwait (src_line=1868,
    src_file=0x15f6bc0 "/var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/mdl.cc", abstime
    mutex=0x7f9424015348, that=0x7f9424015378)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/include/mysql/psi/mysql_thread.h:1236
#4  MDL_wait::timed_wait (this=0x7f9424015348, owner=0x7f94240152b0, abs_timeout=0x7f9475367fc0, s
    wait_state_name=<optimized out>) at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/
#5  0x0000000000e19d02 in TABLE_SHARE::wait_for_old_version (this=0x7f944809d1d0, thd=0x7f9424015
    abstime=0x7f9475367fc0, deadlock_weight=<optimized out>)
```


然后我们稍微整理下堆栈信息,得到如下:

```
pthread_cond_timedwait
native_cond_timedwait
my_cond_timedwait
inline_mysql_cond_timedwait
MDL_wait::timed_wait
TABLE_SHARE::wait_for_old_version
tdc_wait_for_old_version
open_table
open_and_process_table
open_tables
open_tables_for_query
execute_sqlcom_select
mysql_execute_command
mysql_parse
dispatch_command
do_command
handle_connection
```

也就是 解析SQL，打开表时，wait_for_old_version了. 那么进入这个wait_for_old_version的依据是什么呢？我们查询sql/sql_base.cc发现如下逻辑:

```
if (!(flags & MYSQL_OPEN_IGNORE_FLUSH))
  {
if (share->has_old_version())
    {
.......
      wait_result= tdc_wait_for_old_version(thd, table_list->db,
                                            table_list->table_name,
                                            ot_ctx->get_timeout(),
                                            deadlock_weight);
.....
    }
  }
```

也就是判断到has_old_version了. 我们在sql/table.h里面找到了has_old_version的逻辑:

```
structTABLE_SHARE{
/**
    TABLE_SHARE version, if changed the TABLE_SHARE must be reopened.
NOTE: The TABLE_SHARE will not be reopened during LOCK TABLES in
    close_thread_tables!!!
  */
  ulong   version;

inlineboolhas_old_version()const
{
return version != refresh_version;
  }

}
```

也就是这个TABLE_SHARE的版本如果低于refresh_version的话，就会去wait_for_old_version, 然后堵着. 那么这个refresh_version又是啥呢？我们发现其定义为: sql/mysqld.cc

```
ulong refresh_version;  /* Increments on each reload */
```

看起来是每次reload就会++，也就是我们的flush table可能触发了这个refresh_version++, 导致和table_share的版本. 我们来瞅瞅.

```
gdb -p `pidof mysqld` --batch --ex "print refresh_version" | grep '\$1'
```

```
16:24:24 [root@ddcw21 ~]#gdb -p `pidof mysqld` --batch --ex "print refresh_version" | grep '\$1'
$1 = 7
16:24:27 [root@ddcw21 ~]#mysql -h127.0.0.1 -P3418 -p123456 -e 'flush tables'
mysql: [Warning] Using a password on the command line interface can be insecure.
^C^C -- query aborted
ERROR 1317 (70100) at line 1: Query execution was interrupted
16:24:40 [root@ddcw21 ~]#gdb -p `pidof mysqld` --batch --ex "print refresh_version" | grep '\$1'
$1 = 8
16:24:44 [root@ddcw21 ~]#
```


我们发现即使我们的flush tables失败了, 这个refresh_version还是会加1 对应源码如下:

```
boolclose_cached_tables(THD *thd, TABLE_LIST *tables,
bool wait_for_refresh, ulong timeout)
{
...
if (!tables)
  {
/*
      Force close of all open tables.

      Note that code in TABLE_SHARE::wait_for_old_version() assumes that
      incrementing of refresh_version and removal of unused tables and
      shares from TDC happens atomically under protection of LOCK_open,
      or putting it another way that TDC does not contain old shares
      which don't have any tables used.
    */
refresh_version++;

    table_cache_manager.free_all_unused_tables();
/* Free table shares which were not freed implicitly by loop above. */
while (oldest_unused_share->next)
      (void) my_hash_delete(&table_def_cache, (uchar*) oldest_unused_share);

  }
...
}
```

逻辑比较简单, 就是先refresh_version++; 然后遍历表并关闭. 我们来gdb看下堆栈呢 (break close_cached_tables)

```
(gdb) break close_cached_tables
Breakpoint 1 at 0xd18122: file /var/lib/pb2/sb_1-12949965-1697025378.23/mysql
-5.7.44/sql/sql_base.cc, line 1208.
(gdb) c
Continuing.
[Switching to Thread 0x7f94753ac700 (LWP 26720)]

Breakpoint 1, close_cached_tables (thd=0x7f94300008c0, tables=0x0, wait_for_r
efresh=true, timeout=31536000)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_base.cc:
1208
1208     /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_base.c
c: No such file or directory.
(gdb) bt
#0  close_cached_tables (thd=0x7f94300008c0, tables=0x0, wait_for_refresh=tru
e, timeout=31536000)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_base.cc:
1208
#1  0x0000000000d9986a in reload_acl_and_cache (thd=0x7f94300008c0, options=
4, tables=0x0, write_to_binlog=0x7f94753ab270)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_reload.c
c:293
#2  0x0000000000d69712 in mysql_execute_command (thd=0x7f94300008c0, first_le
vel=true)
```

```
     at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_parse.c
c:4150
#3  0x0000000000d6e1ad in mysql_parse (thd=0x7f94300008c0, parser_state=<opti
mized out>)
     at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_parse.c
c:5584
#4  0x0000000000d6f9e8 in dispatch_command (thd=0x7f94300008c0, com_data=0x7f
94753abda0, command=COM_QUERY)
     at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_parse.c
c:1492
#5  0x0000000000d70594 in do_command (thd=0x7f94300008c0)
     at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/sql_parse.c
c:1031
#6  0x0000000000e43b2c in handle_connection (arg=<optimized out>)
     at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/sql/conn_handle
r/connection_handler_per_thread.cc:313
#7  0x000000000123a884 in pfs_spawn_thread (arg=0x8cf69c0)
    at /var/lib/pb2/sb_1-12949965-1697025378.23/mysql-5.7.44/storage/perfsche
ma/pfs.cc:2197
#8  0x00007f956481fea5 in start_thread () from /lib64/libpthread.so.0
#9  0x00007f95632d796d in clone () from /lib64/libc.so.6
```

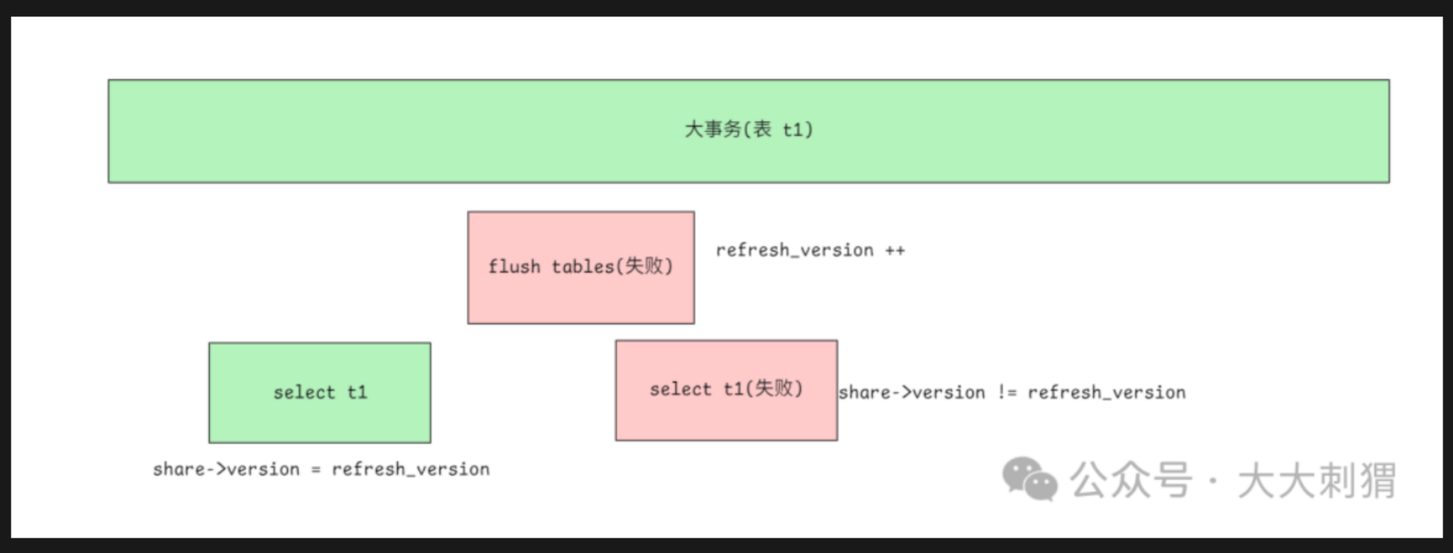
到这里基本上就能确定是flush时,refresh_version++, 然后由于大事务导致超时(其实无所谓了,只要执行了就会导致), 其它会话open_table的时候发现share->version != refresh_version 于是就去wait_for_old_version...

> 我们还发现flush tables会刷binlog, 以前都没注意过.

总结

也就是备份(mysqldump)给refresh_version++了, 然后 后面的表就只能去open_table, 但是有个大事务'挡着'的, 就只能等着(直到超时)

差不多就是如下图:



本次问题算是之前 大事务导致备份失败的 后续问题分析吧, 但当时那个场景是没得业务查询的, 所以未发现这个隐藏的坑.

建议:

1. 优化业务逻辑, 减少大事务/长时间执行的sql

2.备份时间要选在无大事务时间点(但业务变更可能会撞上来,简直防不胜防啊)

> 测试发现`flush tables with read lock`也会导致refresh_version++

参考:
https://dev.mysql.com/doc/refman/8.0/en/general-thread-states.html
https://dev.mysql.com/doc/refman/8.0/en/performance-schema-metadata-

[MYSQL] 出现大量的Waiting for table flush导致业务表查询不了
https://mp.weixin.qq.com/s/s4cFhr-AjTvY1biLmLl6cg

```
                                locks-table.html
```