如何快速定位MySQL性能问题

一　引言
二　常见性能问题
三　以“数据工厂”流水线作业理解SQL处理过程
四　充分挖掘“各工序”的计数
五　掌握“各工序”的“天花板”指标

系统管理中心数据库组　徐春阳

中国民生银行 CHINA MINSHENG BANK 科技开发部

一 引言

- 面对应用运维/研发人员反馈数据库有性能问题后，第一反应是什么？有何思路?

- 当发现数据库存在异常现象？异常现象是原因还是结果？例如cpu利用率比较高的时候。

- 透过直观的外部现象，对数据库内部的各种指标挖掘找根本原因。

二 常见的性能问题

1、与数据库建立连接巨慢，甚至建立失败。

2、偶尔出现锁超时，总体响应变慢。

3、数据库响应超慢，比正常情况下慢一个数量级。

4、数据库的访问量升高一点点，但服务器某类资源接近用光。（增加的一点点压力成了最后一根稻草）

5、数据库响应变慢，比平时慢一倍。但批量任务，执行时间就要延长一倍（这类问题往往是最复杂的问题）

。。。。。。。。。。。。。。。。。。。。

三 以“数据加工厂”流水线作业来理解SQL处理过程

1、**工厂流水线:有些环节可以并行，有些环节只能串行**。表面上看，数据库每秒钟可以执行非常多的sql,感觉全部环节都是在并行工作，但依然存在大量串行环节，如:

a.进入innodb存储引擎工作队列（工作队列满时，用户线程就要在工作队列之外排队等待）。

b.事务是一组一组提交的，提交有先后，必然要串行。同时如果开启半同步，则在binlog同步到从库之前需等待。这是网络来回通信，等待时间跟网络响应时间成正比。

c.很多全局的mutex. 全局互斥锁，但这类锁，通常单个线程拥有锁的时间非常短。

2、有些环节步骤简单，单次耗时少，但该环节有时可能需要执行非常多次。例如在表上读取一行，耗时极少，但范围查找，就需要将这个范围内（甚至整个表，具体情况根据sql的执行计划决定）的所有记录一条一条找出，则耗时较长。

3、尽量避免“工具”/零部件短缺的情况。在加工过程中，需要使用一些“工具”或者零部件，如果没有提前就绪，则需要临时重新制造，从而影响流水线效率。如各种类型的cache以及buffer充当类似的功能. 当某个被访问的表不存在于table cache中时，需要重新构建。 当某个要访问的数据页不在buffer pool中时， 则需要从磁盘中加载。

四 充分挖掘“各工序”的计数

- 通过“各工序”的计数，来了解“数据工厂”各环节的任务压力。

- 综合各“各工序”的计数，来确定“工厂”效率“降低”的源头。假如流水线前面环节卡住了，后面环节的工序反而闲下来。

看懂“各工序”计数的前提，充分理解各工序的含义。

4.1 thread相关计数

```
mysql> show global  status like 'thread%';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Threads_cached    | 170   |
| Threads_connected | 5     |
| Threads_created   | 178   |
| Threads_running   | 2     |
+-------------------+-------+
4 rows in set (0.00 sec)
```


- 如果thread_created在不停地增加，则表明存在短链接访问数据库，且当链接建立时，mysql主线程会创建线程。1、考虑增大thread_cache_size。 2、考虑是否将短链接改造成长连接的方式。

```
bool
Per_thread_connection_handler::add_connection(Channel_info*
channel_info)
{
  if (!check_idle_thread_and_enqueue_connection(channel_info))
    DBUG_RETURN(false);
  /*
    There are no idle threads avaliable to take up the new
    connection. Create a new thread to handle the connection
  */
  channel_info->set_prior_thr_create_utime();
  error= mysql_thread_create(key_thread_one_connection, &id,
                             &connection_attrib,
                             handle_connection,
                             (void*) channel_info);
  Global_THD_manager::get_instance()->inc_thread_created();
  DBUG_PRINT("info",("Thread created"));
  DBUG_RETURN(false);
}
```

4.2 table cache相关计数

```
mysql> show global  status like '%table%' ;
+----------------------------------+-------------+
| Variable_name                    | Value       |
+----------------------------------+-------------+
| Open_table_definitions           | 12291       |
| Open_tables                      | 233099      |
| Opened_table_definitions         | 35171       |
| Opened_tables                    | 430610      |
| Table_open_cache_hits            | 1783135010  |
| Table_open_cache_misses          | 430610      |
| Table_open_cache_overflows       | 0           |
+----------------------------------+-------------+
```

- table_open_cache table open cache缓存的是在sql执行过程中必须使用的一个对象（可以理解为数据加工/访问过程中必须使用的一个工具），如果没有，则需要重构，将大大降低处理速度 。如果Table_open_cache_misses不停增长，则需要考虑参数table_open_cache是否设置得足够大。

```
bool open_table(THD *thd, TABLE_LIST *table_list,
Open_table_context *ot_ctx)
{
  TABLE *table; TABLE_SHARE *share;
  ..........................................................
retry_share:
  {
    Table_cache *tc= table_cache_manager.get_cache(thd);
    tc->lock();
    /*
      Try to get unused TABLE object or at least pointer to
      TABLE_SHARE from the table cache.
*/
table= tc->get_table(thd, hash_value, key, key_length, &share);

... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
```

```
if (table)
{
  /* We have found an unused TABLE object. */
  thd->status_var.table_open_cache_hits++;
      goto table_found;
}
    else if (share)
{
  /*
    We weren't able to get an unused TABLE object. Still we have
    found TABLE_SHARE for it. So let us try to create new TABLE
    for it. We start by incrementing share's reference count and
    checking its version.
  */
        goto share_found;
}
    else
{
  /*
    We have not found neither TABLE nor TABLE_SHARE object in
    table cache (this means that there are no TABLE objects for
    it in it).
    Let us try to get TABLE_SHARE from table definition cache or
    from disk and then to create TABLE object for it.
  */
  tc->unlock();
}
```

.....

获取表的shared对象

```
if (!(share= get_table_share_with_discover(thd, table_list, key,
                                           key_length, OPEN_VIEW,
                                           &error,
                                           hash_value)))
```

share_found:

.....

通过table_shared对象，生成TABLE对象.

```
/* make a new table */
```

首先通过malloc分配内存，这个是比较消耗系统资源的，硬内存分配。

```
if (!(table= (TABLE*) my_malloc(key_memory_TABLE,
                                sizeof(*table), MYF(MY_WME))))
  goto err_lock;

 error= open_table_from_share(thd, share, alias, 。。。。。。。。。。)
 ... ... ... ... ... ... ... ... ... ..
```

thd->status_var.table_open_cache_misses++;

Table_found:

发现现成的可以使用的TABLE结构,然后进行一些初始化的设置。

.....

}

4.4 handler 相关计数

```
Handler_read_first          | 509
Handler_read_key            | 2048617015
Handler_read_last           | 0
Handler_read_next           | 666954748
Handler_read_prev           | 0
Handler_read_rnd            | 63963
Handler_read_rnd_next       | 36253181
```

以上Handler相关指标:精确到行级别的粒度反映当前数据库的压力。指标是累积值，需要作差才能反映当前压力状态。

4.4.1 Handler_read_key与Handler_read_next分析

Handler_read_key表示基于key值的查找次数。如select * from xcytest_ind where a in (1,2,3)，如果sql执行时走a列上的索引，则Handler_read_key会增加3。因为在执行时，是通过指定key的方式，在索引树上进行三次查找。 同时，如果a列的索引是唯一性索引，则Handler_read_next不会增加。

唯一性索引非唯一性索引的区别在于，在唯一性索引上查找，对于指定key值查找，命中就返回。 在非唯一性索引上查找，找到之后还需要查找下一条（行）记录是否满足，所以Handler_read_next会增加，直到下一条记录不满足查找条件，才完成本次查找。 满足条件的记录越多， Handler_read_next则增加越多.

如果Handler_read_next增加较多，则表明可能存在范围查询或者模糊查询，或者等值查询时表匹配的行较多。

4.4.2 Handler_read_first与Handler_read_rnd_

这两个指标跟全表扫描有关：

Handler_read_first：当进行全表扫描时，查找第一条记录之前，该值会增加1。通过该值，可以判断全表扫描的次数。

Handler_read_rnd_next：该值直接跟表的记录数对应，每读取一行，该值加1。

Handler_read_rnd_next值的增长速度，直接反映全表扫描给服务器带来的压力。

```
/***********************************************************************//**
Positions a cursor on the first record in an index and reads the
corresponding row to buf.
@return 0, HA_ERR_END_OF_FILE, or error code */

int
ha_innobase::index_first(
/*=====================*/
        uchar*  buf)    /*!< in/out: buffer for the row */
{
        DBUG_ENTER("index_first");

        ha_statistic_increment(&SSV::ha_read_first_count);

        int     error = index_read(buf, NULL, 0, HA_READ_AFTER_KEY);

        /* MySQL does not seem to allow this to return HA_ERR_KEY_NOT_FOUND */

        if (error == HA_ERR_KEY_NOT_FOUND) {
                error = HA_ERR_END_OF_FILE;
        }

        DBUG_RETURN(error);
}
```

```
/**************************************************************************//**
Reads the next row in a table scan (also used to read the FIRST row
in a table scan).
@return 0, HA_ERR_END_OF_FILE, or error number */
int
```

ha_innobase::rnd_next(

```
/*==================*/
        uchar*  buf)    /*!< in/out: returns the row in this buffer,
                        in MySQL format */
{
        int     error;

        DBUG_ENTER("rnd_next");

        ha_statistic_increment(&SSV::ha_read_rnd_next_count);

        if (m_start_of_scan) {
                error = index_first(buf);

                if (error == HA_ERR_KEY_NOT_FOUND) {
                        error = HA_ERR_END_OF_FILE;
                }

                m_start_of_scan = false;
        } else {
                error = general_fetch(buf, ROW_SEL_NEXT, 0);
        }

        DBUG_RETURN(error);
}
```

4.5.1 Buffer Pool 活跃度指标

```
mysql> show global  status like '%buffer%';
+---------------------------------------+--------------+
| Variable_name                         | Value        |
+---------------------------------------+--------------+
| Innodb_buffer_pool_read_requests      | 16572324902  |
| Innodb_buffer_pool_pages_dirty        | 405454       |
| Innodb_buffer_pool_pages_flushed      | 32871155     |
| Innodb_buffer_pool_reads              | 2620654      |
| Innodb_buffer_pool_wait_free          | 0            |
| Innodb_buffer_pool_write_requests     | 639009344    |
+---------------------------------------+--------------+
```

以上指标反映buffer的活跃程度，包括对buffer中的数据页的读，写，页面刷新，以及物理读等次数。

4.5.2 Buffer Pool 活跃度指标解释

- 仅仅通过buffer pool读写次数，无法判断性能是否良好.

例如如果存在全表扫描，buffer pool读的次数很高，但性能反而降低。

如果存在表锁等待等，进而buffer pool 的读（写）频率可能会较低，buffer pool的活跃度降低。侧面推导：如果数据库性能低，同时buffer pool的活跃度低，则可以排除全表扫描，问题可能出现在其他部分。

- buffer pool的活跃数据，仅代表buffer pool的活跃程度，只能作为性能问题定位的参考依据。

4.6 SEMI-SYNC 相关指标

```
mysql> show global  status like '%semi%wait%';
+-------------------------------------------+-------+
| Variable_name                             | Value |
+-------------------------------------------+-------+
| Rpl_semi_sync_master_net_avg_wait_time    | 0     |
| Rpl_semi_sync_master_net_wait_time        | 0     |
| Rpl_semi_sync_master_net_waits            | 50    |
| Rpl_semi_sync_master_tx_avg_wait_time     | 807   |
| Rpl_semi_sync_master_tx_wait_time         | 20186 |
| Rpl_semi_sync_master_tx_waits             | 25    |
| Rpl_semi_sync_master_wait_pos_backtraverse| 0     |
| Rpl_semi_sync_master_wait_sessions        | 0     |
+-------------------------------------------+-------+
```

五 “各工序”天花板

- 如果开启半同步，数据库所支持的TPS无法突破半同步机制的所产生的瓶颈。

- 物理随机IO的瓶颈。当数据库的缓存命中率低，随之带来就是同步的物理读，如果超出磁盘IO能力，性能将雪崩。

- 热点行更新的瓶颈。由于锁保护机制，更新热点行的速度很容易成为瓶颈。

- 各种串行的处理方式。如select for update等语句。

Thanks!