网易终面：100G内存下，MySQL查询200G大表会OOM么？

码猿技术专栏 2025年04月23日 13:40 浙江

我的主机内存只有100G，现在要全表扫描一个200G大表，会不会把DB主机的内存用光？

逻辑备份时，可不就是做整库扫描吗？若这样就会把内存吃光，逻辑备份不是早就挂了？

所以大表全表扫描，看起来应该没问题。这是为啥呢？

全表扫描对server层的影响

假设，我们现在要对一个200G的InnoDB表db1. t，执行一个全表扫描。当然，你要把扫描结果保存在客户端，会使用类似这样的命令：

```
mysql -h$host -P$port -u$user -p$pwd -e
  "select * from db1.t" > $target_file
```

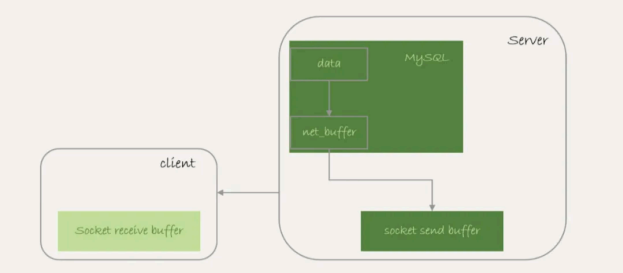
InnoDB数据保存在主键索引上，所以全表扫描实际上是直接扫描表t的主键索引。这条查询语句由于没有其他判断条件，所以查到的每一行都可以直接放到结果集，然后返回给客户端。

那么，这个“结果集”存在哪里呢？

服务端无需保存一个完整结果集。取数据和发数据的流程是这样的：

- 获取一行，写到**net_buffer**。这块内存的大小是由参数**net_buffer_length**定义，默认16k
- 重复获取行，直到**net_buffer**写满，调用网络接口发出去
- 若发送成功，就清空**net_buffer**，然后继续取下一行，并写入**net_buffer**
- 若发送函数返回**EAGAIN**或**WSAEWOULDBLOCK**，就表示本地网络栈（socket send buffer）写满了，进入等待。直到网络栈重新可写，再继续发送

查询结果发送流程：



可见：

- 一个查询在发送过程中，占用的MySQL内部的内存最大就是**net_buffer_length**这么大，不会达到200G
- socket send buffer 也不可能达到200G（默认定义/proc/sys/net/core/wmem_default），若socket send buffer被写满，就会暂停读数据的流程

所以MySQL其实是“边读边发”。这意味着，若客户端接收得慢，会导致MySQL服务端由于结果发不出去，这个事务的执行时间变长。

比如下面这个状态，就是当客户端不读**socket receive buffer**内容时，在服务端show processlist看到的结果。

服务端发送阻塞：

```
mysql> show processlist;
+----+------+-----------------+------+---------+------+-------------------+-----------------------+
| Id | User | Host            | db   | Command | Time | State             | Info                  |
+----+------+-----------------+------+---------+------+-------------------+-----------------------+
|  4 | root | localhost:61696 | test | Query   |    0 | starting          | show processlist      |
|  5 | root | localhost:61772 | test | Query   |    9 | Sending to client | select * from t       |
+----+------+-----------------+------+---------+------+-------------------+-----------------------+
```

若看到State一直是“Sending to client”，说明服务器端的网络栈写满了。

若客户端使用–quick参数，会使用mysql_use_result方法：读一行处理一行。假设某业务的逻辑较复杂，每读一行数据以后要处理的逻辑若很慢，就会导致客户端要过很久才取下一行数据，可能就会出现上图结果。

因此，对于正常的线上业务来说，若一个查询的返回结果不多，推荐使用**mysql_store_result**接口，直接把查询结果保存到本地内存。

当然前提是查询返回结果不多。如果太多，因为执行了一个大查询导致客户端占用内存近20G，这种情况下就需要改用**mysql_use_result**接口。

若你在自己负责维护的MySQL里看到很多个线程都处于“Sending to client”，表明你要让业务开发同学优化查询结果，并评估这么多的返回结果是否合理。

若要快速减少处于这个状态的线程的话，可以将**net_buffer_length**设置更大。

有时，实例上看到很多查询语句状态是“Sending data”，但查看网络也没什么问题，为什么Sending data要这么久？

一个查询语句的状态变化是这样的：

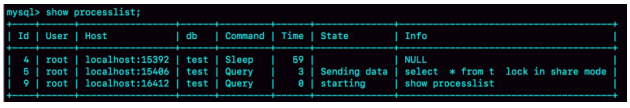
- MySQL查询语句进入执行阶段后，先把状态设置成 **Sending data**
- 然后，发送执行结果的列相关的信息（meta data) 给客户端
- 再继续执行语句的流程
- 执行完成后，把状态设置成空字符串

即“Sending data”并不一定是指“正在发送数据”，而可能是处于执行器过程中的任意阶段。比如，你可以构造一个锁等待场景，就能看到Sending data状态。

读全表被锁：

session 1	session2
begin select * from t where id=1 for update	启动事务
	select * from t lock in share mode (blocked)

Sending data状态



可见session2是在等锁，状态显示为Sending data。

- 仅当一个线程处于“等待客户端接收结果”的状态，才会显示"Sending to client"
- 若显示成“Sending data”，它的意思只是“正在执行”

所以，查询的结果是分段发给客户端，因此扫描全表，查询返回大量数据，并不会把内存打爆。

以上是server层的处理逻辑，在InnoDB引擎里又是怎么处理？

全表扫描对InnoDB的影响

InnoDB内存的一个作用，是保存更新的结果，再配合redo log，避免随机写盘。

内存的数据页是在Buffer Pool (简称为BP)管理，在WAL里BP起加速更新的作用。

BP还能加速查询。

▶▶

由于WAL，当事务提交时，磁盘上的数据页是旧的，若这时马上有个查询来读该数据页，是不是要马上把redo log应用到数据页？

不需要。因为此时，内存数据页的结果是最新的，直接读内存页即可。这时查询无需读磁盘，直接从内存取结果，速度很快。所以，Buffer Pool能加速查询。

▶▶

而BP对查询的加速效果，依赖于一个重要的指标，即：内存命中率。

可以在show engine innodb status结果中，查看一个系统当前的BP命中率。一般情况下，一个稳定服务的线上系统，要保证响应时间符合要求的话，内存命中率要在99%以上。

执行show engine innodb status ，可以看到“Buffer pool hit rate”字样，显示的就是当前的命中率。比如下图命中率，就是100%。

```
Buffer pool hit rate 1000 / 1000
```

若所有查询需要的数据页都能够直接从内存得到，那是最好的，对应命中率100%。

InnoDB Buffer Pool的大小是由参数 **innodb_buffer_pool_size**确定，一般建议设置成可用物理内存的60%~80%。

在大约十年前，单机的数据量是上百个G，而物理内存是几个G；现在虽然很多服务器都能有128G甚至更高的内存，但是单机的数据量却达到了T级别。

所以，**innodb_buffer_pool_size**小于磁盘数据量很常见。若一个 Buffer Pool满了，而又要从磁盘读入一个数据页，那肯定是要淘汰一个旧数据页的。

InnoDB内存管理

使用的最近最少使用 (Least Recently Used, LRU)算法，淘汰最久未使用数据。

基本LRU算法

InnoDB管理BP的LRU算法，是用链表实现的：

- state1，链表头部是P1，表示P1是最近刚被访问过的数据页
- 此时，一个读请求访问P3，因此变成状态2，P3被移到最前
- 状态3表示，这次访问的数据页不存在于链表，所以需要在BP中新申请一个数据页Px，加到链表头。但由于内存已满，不能申请新内存。于是清空链表末尾Pm数据页内存，存入Px的内容，放到链表头部

最终就是最久没有被访问的数据页Pm被淘汰。

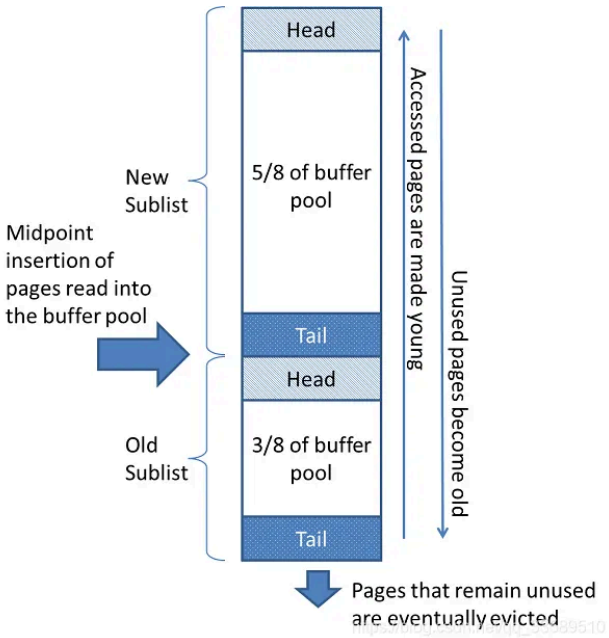
若此时要做一个全表扫描，会咋样？若要扫描一个200G的表，而这个表是一个历史数据表，平时没有业务访问它。

那么，按此算法扫描，就会把当前BP里的数据全部淘汰，存入扫描过程中访问到的数据页的内容。也就是说BP里主要放的是这个历史数据表的数据。

对于一个正在做业务服务的库，这可不行呀。你会看到，BP内存命中率急剧下降，磁盘压力增加，SQL语句响应变慢。

所以，InnoDB不能直接使用原始的LRU。InnoDB对其进行了优化。

改进的LRU算法



InnoDB按5:3比例把链表分成New区和Old区。图中LRU_old指向的就是old区域的第一个位置，是整个链表的5/8处。即靠近链表头部的5/8是New区域，靠近链表尾部的3/8是old区域。

改进后的LRU算法执行流程：

- 状态1，要访问P3，由于P3在New区，和优化前LRU一样，将其移到链表头部 =》状态2
- 之后要访问一个新的不存在于当前链表的数据页，这时依然是淘汰掉数据页Pm，但新插入的数据页Px，是放在LRU_old处
- 处于old区的数据页，每次被访问的时候都要做如下判断：
  - 若该数据页在LRU链表中存在的时间超过1s，就把它移动到链表头部
  - 若该数据页在LRU链表中存在的时间短于1s，位置保持不变。1s是由参数innodb_old_blocks_time控制，默认值1000，单位ms。

该策略，就是为了处理类似全表扫描的操作量身定制。还是扫描200G历史数据表：

- 扫描过程中，需要新插入的数据页，都被放到old区域
- 一个数据页里面有多条记录，这个数据页会被多次访问到，但由于是顺序扫描，这个数据页第一次被访问和最后一次被访问的时间间隔不会超过1秒，因此还是会被保留在old区域
- 再继续扫描后续的数据，之前的这个数据页之后也不会再被访问到，于是始终没有机会移到链表头部（New区），很快就会被淘汰出去。

可以看到，这个策略最大的收益，就是在扫描这个大表的过程中，虽然也用到了BP，但对young区完全没有影响，从而保证了Buffer Pool响应正常业务的查询命中率。

小结

MySQL采用的是边算边发的逻辑，因此对于数据量很大的查询结果来说，不会在server端保存完整的结果集。所以，如果客户端读结果不及时，会堵住MySQL的查询过程，但是不会把内存打爆。

而对于InnoDB引擎内部，由于有淘汰策略，大查询也不会导致内存暴涨。并且，由于InnoDB对LRU算法做了改进，冷数据的全表扫描，对Buffer Pool的影响也能做到可控。

全表扫描还是比较耗费IO资源的，所以业务高峰期还是不能直接在线上主库执行全表扫描的。

文章目录可以扫码进入查看