33查这么多数据，会不会把数据库内存打爆

**全表扫描对server层的影响**

假设对一个200G的InnoDB表db1.t，执行一次全表扫描，当然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的数据是保存在主键索引上的，所以全表扫描实际上是直接扫描表t的主键索引，这条查询由于没有其他的条件，所以查到的每一行都可以直接放到结果集里面，然后返回给客户端。

结果集存在哪里？

实际上，服务端不需要保存一个完整的结果集，取数据和发数据的流程是这样的

1. 获取一行，写到net\_buffer中，这块内存的大小是由参数net\_buffer\_lenth定义的默认是16k
2. 重复获取行，直到net\_buffer写满，调用网络接口发出去
3. 如果发送成功，就清空ne\_buffer，然后继续取下一行，并写入net\_buffer
4. 如果发送函数返回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。

过程如图

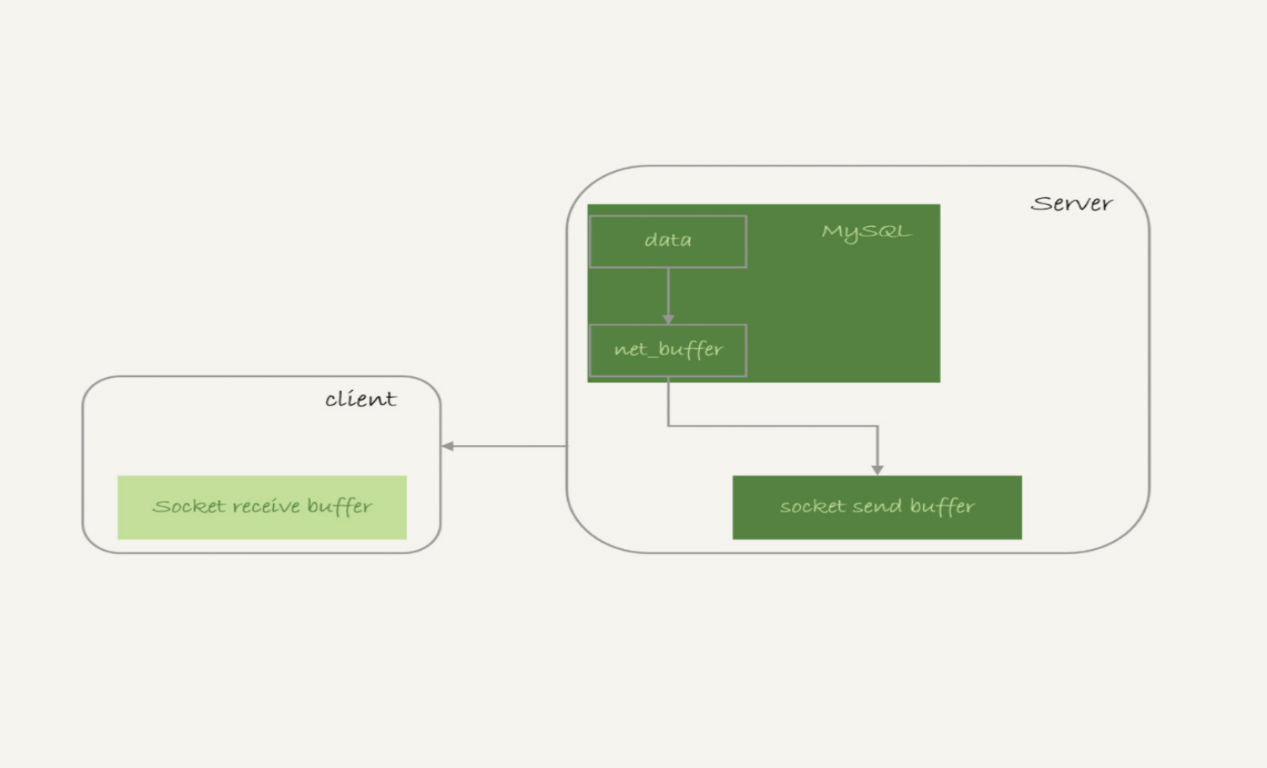


图 1 查询结果发送流程

1. 一个查询在发送过程中，占用MySQL内部的内存最大就是net\_buffer\_length这么大，并不会达到200G
2. Socket send buffer也不可能达到200G,(默认定义)/proc/sys/net/core/wmem\_default)，如果socket send buffer 被写满，就户暂停读数据的流程。

**MySQL是“边发边读”**这个概念很重要，意味着，如果客户端接收得慢，会导致MySQL服务端由于结果发不出去了，这个是事务的执行时间变长。

下面这个状态，就是故意让客户端不去读socket receive buffer中的内容，然后在服务端show processlist看到的结果

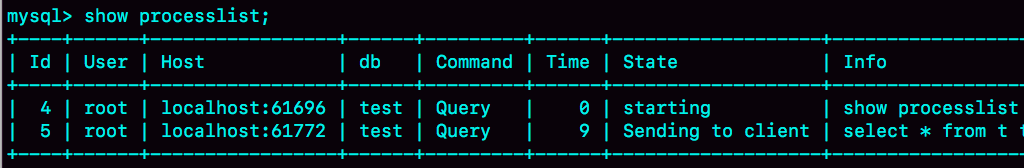


图 2 服务端发送阻塞

如果看到State的值一直处于“Sending to client”，就表示服务端的网络栈写满了。

上一篇中提到，如果客户端使用-quick参数，会使用mysql\_use\_result方法，这个方法是读一行处理一行，可以想象一下，假设有一个业务的逻辑比较复杂，每读一行数据以后要处理的逻辑很慢，就会导致客户端要过很久才会去去下一行数据，可能会出现如图2的情况

因此，**对于正常的线上业务来说，如果一个查询的返回结果不会很多的话，都建议使用mysql\_store\_result这个接口，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，在30篇，评论区，执行一个大查询导致客户端内存20G,这种情况就需要改用mysql\_use\_result接口了。

另一方面，如果维护的MySQL里看到很多线程都处于“Send to client”这个状态，就以为着要优化查询结果，并评估这么多的返回结果是否合理。

而如果要快速减少处于这个状态的线程的话，将net\_buffer\_length参数设置一个更大的值是一个可选方案。

与“Sent to client”长想很类似的一个状态是“Sending data”,这是一个经常被误会的问题，在自己维护的实例上看到很多查询语句的状态是“Sending data”,但查看网络没什么问题，为什么Sending data要这么久

实际上，一个查询语句的状态变化是这样的（注意：这里略去了其他无关的状态）

·MySQL 查询语句进入执行阶段后，首先把状态设置成“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

“Sending data”并不一定是指“正在发送数据”，而可能是处于执行过程中的任一阶段，比如可以构造一个锁等待的场景，就能看到Sending data状态

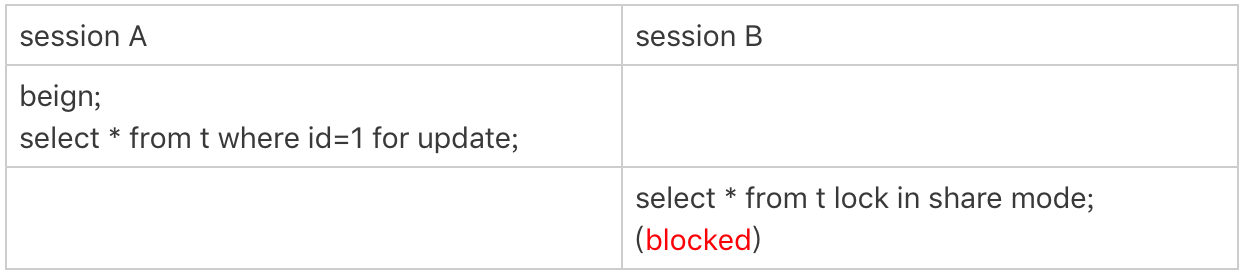


图 3 读全表被锁

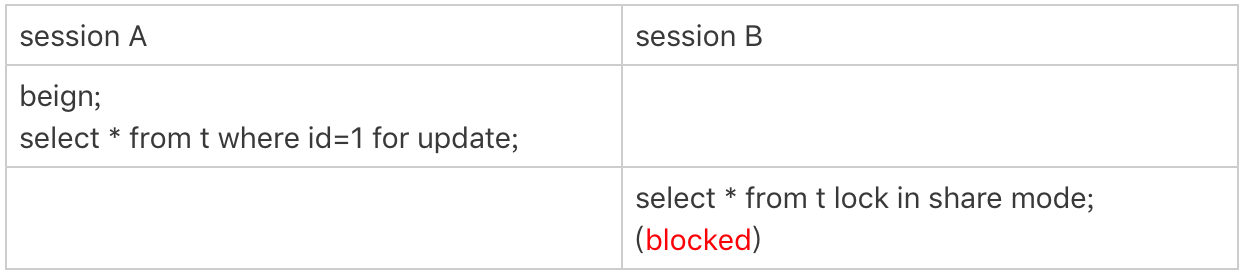


图 Sending data 状态

SessionB明显在等锁，状态显示为Sending data,也就是说，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”;而如果显示成“Sending data”它的意思是“正在执行”。

查询的结果是分段发送给客户端的，因为全表扫描，查询返回大量的数据，并不会打爆内存。

在server层的处理逻辑清楚了，在InnoDB引擎里面又是怎么处理的呢？扫描全表会不会对引擎系统造成影响？

**全表扫描对InnoDB的影响**

在2和15篇中，InnoDB内存的一个作用是，保存更新的结果，在配合redo log，就避免了随机写盘。

内存的数据页是在Buffer Pool(BP)中管理的，在WAL里Buffer Pool起到了加速更新的作用，而实际上，Buffer Pool还有一个更重要的作用，就是加速查询。

由于WAL机制，当事务提交的时候，磁盘上的数据页时旧的，如果马上有一个查询要来读这个数据页，是不是要马上把redo log应用到数据页？

答案是不需要，因为内存数据页的结果是最新的，直接读内存就可以了，此时查询根本不需要读磁盘，直接从内存拿结果，速度是很快的，所以说，Buffer Pool还有加速查询的作用。

而Buffer Pool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，一个稳定服务的线上系统，要保证响应时间要符合要求的话，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，显示的就是当前命中率，如图5



图 5 show engine innodb status显示内存命中率

如果所有查询的数据页都能够直接从内存得到，是最好的，对应的命中率就是100%，但，在实际生产上是很难的

InnoDB Buffer Pool的大小是由参数innodb\_buffer\_pool\_size确定的，一般建议设置成可用物理内存的68-80%。

大约在十年前，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有128G甚至更高的内存，但是单机的数据量却达到了上T级别。

所以，innodb\_buffer\_pool\_size小于磁盘的数据量是很常见的，如果一个Buffer Pool满了，而要从磁盘读入一个新的数据页，肯定是要淘汰一个旧的数据页。

InnoDB内存管理用的是最近最少使用（Least Rencently Userd,LRU）算法，这个算法的核心就是淘汰最久为使用的数据

如图是一个LRU算法的基本模型

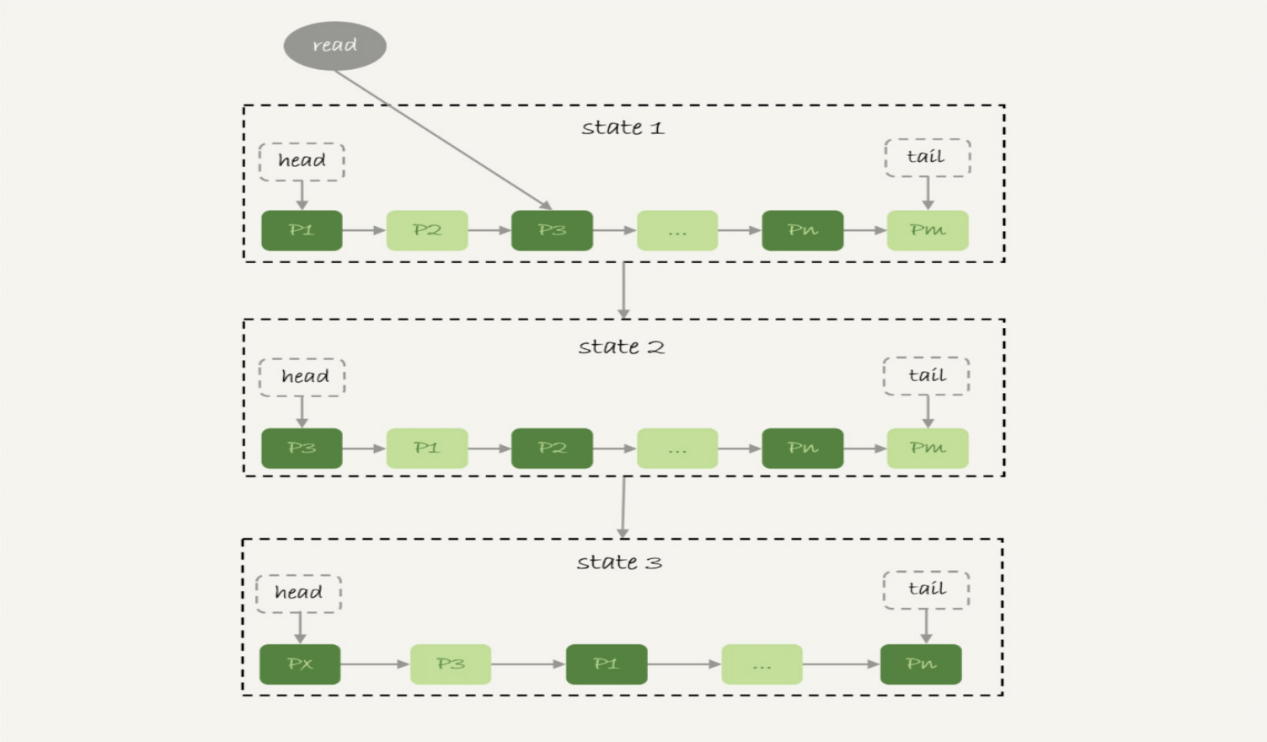


图 6 基本LRU算法

InnoDB管理Buffer Poo的LRU算法，是用链表来实现的

1. 在图6的状态1里，链表头部是P1,表示P1最近刚刚被访问过的数据页；假设内存里只能放这么多数据页
2. 此时有一个请求访问P3,因此变成状态2，P3被移到最前面
3. 状态3表示，这次访问数据页不存在于链表中，所以需要在Buffer Pool中新申请一个数据页Px,加到链表头部。但是由于内存已经满了，不能申请新的内存，于是会清空链表末尾Pm这个数据页的内存，存入Px的内容，然后放到链表头部
4. 从效果上看，就是最久没有被访问的数据页Pm，被淘汰了。

这个算法看上去没什么问题，如果考虑到全表扫描，会不会有问题呢？

假设按照这个算法，要扫描一个200G的表，而这个表时一个历史数据表，平时没什么业务访问，按照这个算法，就会把当前的Buffer Pool里面的数据全部淘汰掉，存入扫描过程中访问到的数据也的内容，也就是说Buffer Pool里面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，Buffer Pool的内存命中率急剧下降，磁盘压力增加，SQL响应变慢。

所以，InnoDB不直接使用这个LRU算法，实际上，InnoDB对LRU算法改进

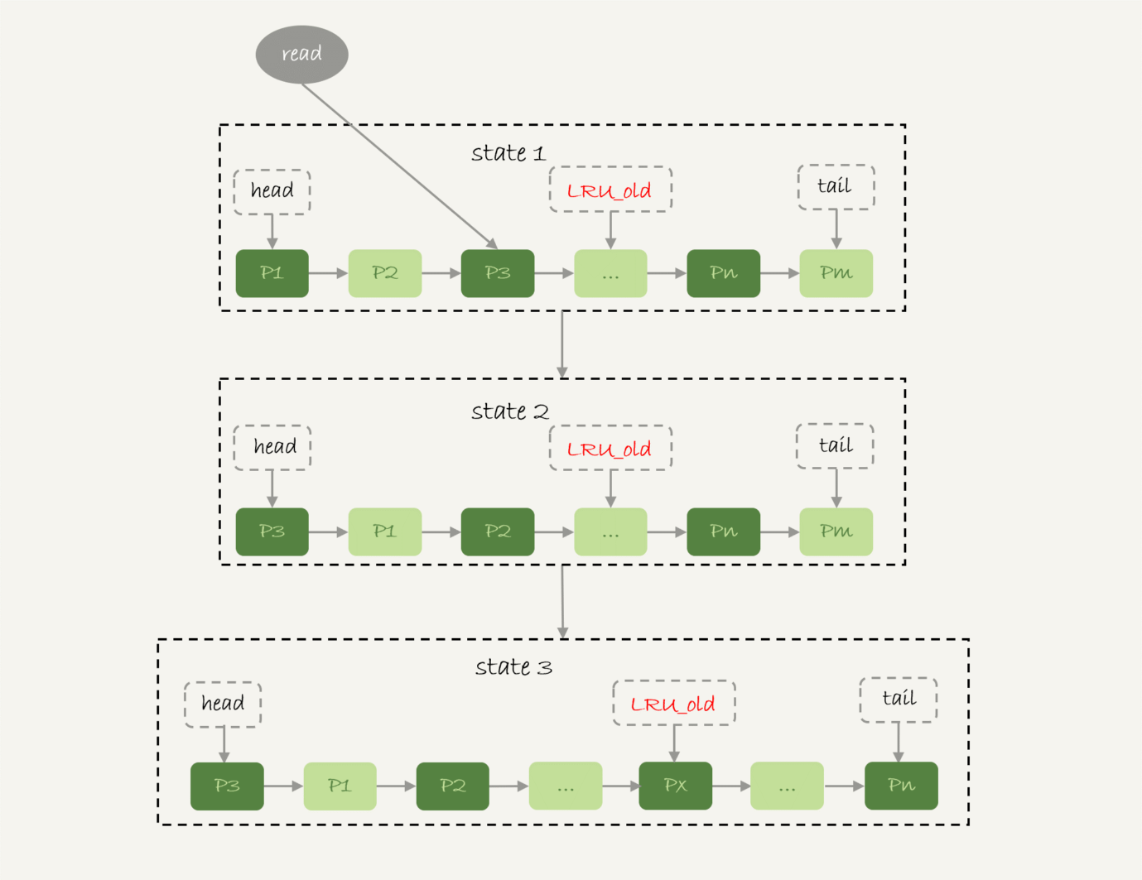


图 7 改进的LRU算法

在InnoDB实现上，按照5:3的比例把整个LUR链表分成了young区域和old区域，图中LRU\_old执行的就是old区域的第一个位置，是整个链表5/8处，也就是说，靠近链表头部的5/8是yound区域，靠近链表尾部的3/8是old区域。

改进后的LRU算法执行流程变成了下面这样

1. 图7中状态1，要访问数据页P3，由于P3在young区域，因此和优化前的LRU算法一样，将其移到链表头部，变成状态2.
2. 之后要访问一个新的不存在于当前链表的数据页，这时候依然是淘汰掉数据页Pm,但是新插入的数据页Px,是放在LRU\_old处。
3. 处于old区域的数据页，每次被访问的时候都要做下面这个判断；

·若这个数据页在LRU链表中存在的时间超过了1秒，就把他移到链表头部

·如果这个数据页在LRU链表中存在的时间短于1秒，位置保持不变。1秒这个时间，是由参数innodb\_old\_blocks\_time控制的，其默认值是1000，单位是毫秒

这个策略，就是为了处理类似全表扫描的操作量身定制的，还是以刚刚扫描200G的历史数据表为例，看看改进后的LRU算法的操作逻辑

1. 扫描过程中，需要新插入的数据页，都被放到old区域
2. 一个数据页里面有多条记录，这个数据页会被多次访问到，但是由于顺序扫描，这个数页第一次被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
3. 在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（也就是young区域），很快会被淘汰出去。

这个策略最大收益，就是扫描这个大表的过程中，虽然也用到了Buffer Pool，但是对于young区域完全没有影响，从而保证了Buffer Pool形影正常业务的查询命中率。

思考题

如果由于客户端压力太大，迟迟不能接受结果，会导致MySQL无法发送结果而影响语句执行，这还不是最糟的情况。

可以设想由于客户端的问题，对数据库影响更严重的例子吗？或者经历过这样的场景？怎么优化？