【mysql】关于ICP、MRR、BKA等特性

Index Condition Pushdown (ICP)是mysql使用索引从表中检索行数据的一种优化方式，从mysql5.6开始支持，mysql5.6之前，存储引擎会通过遍历索引定位基表中的行，然后返回给Server层，再去为这些数据行进行WHERE后的条件的过滤。mysql 5.6之后支持ICP后，如果WHERE条件可以使用索引，MySQL 会把这部分过滤操作放到存储引擎层，存储引擎通过索引过滤，把满足的行从表中读取出。ICP能减少引擎层访问基表的次数和 Server层访问存储引擎的次数。

- ICP的目标是减少从基表中读取操作的数量，从而降低IO操作
- 对于InnoDB表，ICP只适用于辅助索引
- 当使用ICP优化时，执行计划的Extra列显示Using indexcondition提示
- 数据库配置 optimizer_switch="index_condition_pushdown=on”;

使用场景举例

辅助索引INDEX (a, b, c)

```
SELECT * FROM peopleWHERE a='12345' AND b LIKE '%xx%'AND c LIKE '%yy%';
```

若不使用ICP：则是通过二级索引中a的值去基表取出所有a='12345'的数据，然后server层再对b LIKE '%xx%'AND c LIKE '%yy%' 进行过滤

若使用ICP：则b LIKE '%xx%'AND c LIKE '%yy%'的过滤操作在二级索引中完成，然后再去基表取相关数据

ICP特点

- mysql 5.6中只支持 MyISAM、InnoDB、NDB cluster
- mysql 5.6中不支持分区表的ICP，从MySQL 5.7.3开始支持分区表的

ICP

- ICP的优化策略可用于range、ref、eq_ref、ref_or_null 类型的访问数据方法
- 不支持主建索引的ICP（对于Innodb的聚集索引，完整的记录已经被读取到Innodb Buffer，此时使用ICP并不能降低IO操作）
- 当 SQL 使用覆盖索引时但只检索部分数据时，ICP 无法使用
- ICP的加速效果取决于在存储引擎内通过ICP筛选掉的数据的比例

二、Multi-Range Read (MRR)

MRR 的全称是 Multi-Range Read Optimization，是优化器将随机 IO 转化为顺序 IO 以降低查询过程中 IO 开销的一种手段，这对IO-bound类型的SQL语句性能带来极大的提升，适用于range ref eq_ref类型的查询

MRR优化的几个好处

使数据访问有随机变为顺序，查询辅助索引是，首先把查询结果按照主键进行排序，按照主键的顺序进行书签查找

减少缓冲池中页被替换的次数

批量处理对键值的操作

在没有使用MRR特性时

第一步 先根据where条件中的辅助索引获取辅助索引与主键的集合，结果集为rest

```
select key_column, pk_column from tb where key_column=x order by key_column
```

第二步 通过第一步获取的主键来获取对应的值

```
for each pk_column value in rest do:
select non_key_column from tb where pk_column=val
```

使用MRR特性时

第一步 先根据where条件中的辅助索引获取辅助索引与主键的集合，结果集为rest

```
select key_column, pk_column from tb where key_column = x order by key_column
```

第二步 将结果集rest放在buffer里面(read_rnd_buffer_size 大小直到buffer满了)，然后对结果集rest按照pk_column排序，得到结果集是rest_sort

第三步 利用已经排序过的结果集，访问表中的数据，此时是顺序IO.

```
select non_key_column fromtb where pk_column in (rest_sort)
```

在不使用 MRR 时，优化器需要根据二级索引返回的记录来进行“回表”，这个过程一般会有较多的随机IO, 使用MRR时，SQL语句的执行过程是这样的：

- 优化器将二级索引查询到的记录放到一块缓冲区中
- 如果二级索引扫描到文件的末尾或者缓冲区已满，则使用快速排序对缓冲区中的内容按照主键进行排序
- 用户线程调用MRR接口取cluster index，然后根据cluster index 取行数据
- 当根据缓冲区中的 cluster index取完数据，则继续调用过程 2) 3)，直至扫描结束

通过上述过程，优化器将二级索引随机的 IO 进行排序，转化为主键的有序排列，从而实现了随机 IO 到顺序 IO 的转化，提升性能

此外MRR还可以将某些范围查询，拆分为键值对，来进行批量的数据查询，如下：

SELECT * FROM t WHERE *key_part1* >= 1000 AND *key_part1* < 2000AND *key_part2* = 10000;

表t上有二级索引(key_part1, key_part2)，索引根据key_part1,key_part2的顺序排序。

若不使用MRR：此时查询的类型为Range，sql优化器会先将key_part1大于1000小于2000的数据取出，即使key_part2不等于10000，带取出之后再进行过滤，会导致很多无用的数据被取出

若使用MRR：如果索引中key_part2不为10000的元组越多，最终MRR的效果越好。优化器会将查询条件拆分为（1000,1000），（1001,1000），...（1999,1000）最终会根据这些条件进行过滤

相关参数

当mrr=on,mrr_cost_based=on，则表示cost base的方式还选择启用MRR优化,当发现优化后的代价过高时就会不使用该项优化

当mrr=on,mrr_cost_based=off，则表示总是开启MRR优化

```
SET  @@optimizer_switch='mrr=on,mrr_cost_based=on';
```

参数read_rnd_buffer_size 用来控制键值缓冲区的大小。二级索引扫描到文件的末尾或者缓冲区已满，则使用快速排序对缓冲区中的内容按照主键进行排序

三、Batched Key Access (BKA) 和 Block Nested-Loop(BNL)

Batched Key Access (BKA) 提高表join性能的算法。当被join的表能够使用索引时，就先排好顺序，然后再去检索被join的表，听起来和MRR类似，实际上MRR也可以想象成二级索引和 primary key的join

如果被Join的表上没有索引，则使用老版本的BNL策略(BLOCK Nested-loop)

BKA原理

对于多表join语句，当MySQL使用索引访问第二个join表的时候，使用一个join buffer来收集第一个操作对象生成的相关列值。BKA构建好key后，

批量传给引擎层做索引查找。key是通过MRR接口提交给引擎的（mrr目的是较为顺序）MRR使得查询更有效率。

大致的过程如下:

- BKA使用join buffer保存由join的第一个操作产生的符合条件的数据
- 然后BKA算法构建key来访问被连接的表，并批量使用MRR接口提交keys到数据库存储引擎去查找查找。
- 提交keys之后，MRR使用最佳的方式来获取行并反馈给BKA

BNL和BKA都是批量的提交一部分行给被join的表，从而减少访问的次数，那么它们有什么区别呢?

- BNL比BKA出现的早，BKA直到5.6才出现，而NBL至少在5.1里面就存在。
- BNL主要用于当被join的表上无索引
- BKA主要是指在被join表上有索引可以利用，那么就在行提交给被join的表之前，对这些行按照索引字段进行排序，因此减少了随机IO，排序这才是两者最大的区别，但是如果被join的表没用索引呢？那就使用NBL

BKA和BNL标识

Using join buffer (Batched Key Access)和Using join buffer (Block Nested Loop)

相关参数

BAK使用了MRR，要想使用BAK必须打开MRR功能，而MRR基于mrr_cost_based的成本估算并不能保证总是使用MRR，官方推荐设置mrr_cost_based=off来总是开启MRR功能。打开BAK功能(BAK默认OFF)：

```
1 SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

BKA使用**join buffer size**来确定buffer的大小，buffer越大，访问被join的表/内部表就越顺序。

BNL默认是开启的，设置BNL相关参数：

```
SET optimizer_switch='block_nested_loop'
```

支持inner join, outer join, semi-join operations,including nested outer joins

BKA主要适用于join的表上有索引可利用，无索引只能使用BNL

四、总结

ICP（Index Condition Pushdown）

Index Condition Pushdown是用索引去表里取数据的一种优化，减少了引擎层访问基表的次数和Server层访问存储引擎的次数，在引擎层就能够过滤掉大量的数据，减少io次数，提高查询语句性能

MRR（Multi-Range Read）

是基于辅助/第二索引的查询，减少随机IO，并且将**随机IO转化为顺序IO**，提高查询效率。

- **不使用MRR之前**(MySQL5.6之前)，先根据where条件中的辅助索引获取辅助索引与主键的集合，再通过主键来获取对应的值。辅助索引获取的主键来访问表中的数据会导致随机的IO(辅助索引的存储顺序并非与主键的顺序一致)，随机主键不在同一个page里时会导致多次IO和随机读。

- **使用MRR优化**(MySQL5.6之后)，先根据where条件中的辅助索引获取辅助索引与主键的集合，再将结果集放在buffer(read_rnd_buffer_size 直到buffer满了)，然后对结果集按照pk_column排序，得到有序的结果集rest_sort。最后利用已经排序过的结果集，访问表中的数据，此时是顺序IO。即MySQL 将根据辅助索引获取的结果集根据主键进行排序，将无序化为有序，可以用主键

顺序访问基表，将随机读转化为顺序读，多页数据记录可一次性读入或根据此次的主键范围分次读入，减少IO操作，提高查询效率。

***Nested Loop Join**算法*

将驱动表/外部表的结果集作为循环基础数据，然后循环该结果集，每次获取一条数据作为下一个表的过滤条件查询数据，然后合并结果，获取结果集返回给客户端。Nested-Loop一次只将一行传入内层循环, 所以外层循环(的结果集)有多少行, 内存循环便要执行多少次，效率非常差。

***Block Nested-Loop Join**算法*

将外层循环的行/结果集存入join buffer, 内层循环的每一行与整个buffer中的记录做比较，从而减少内层循环的次数。主要用于当被join的表上无索引。

***Batched Key Access**算法*

*当被join的表能够使用索引时，就先好顺序，然后再去检索被join的表。对这些行按照索引字段进行**排序**，因此**减少了随机IO**。如果被Join的表上没有索引，则使用老版本的BNL策略(BLOCK Nested-loop)。*

参考：

http://dev.mysql.com/doc/refman/5.7/en/select-optimization.html

http://www.kancloud.cn/taobaomysql/monthly/117959

http://www.kancloud.cn/taobaomysql/monthly/67181

http://www.cnblogs.com/zhoujinyi/p/4746483.html

如果您觉得本文对您的学习有所帮助，可通过支付宝（左） 或者 微信（右） 来打赏博主，增加博主的写作动力