MySQL索引与Index Condition Pushdown

作者 张洋 | 发布于 2013-12-05

MySQL MariaDB 索引 IndexConditionPushdown

大约在两年前，我写了一篇关于MySQL索引的文章。最近有同学在文章的评论中对文章的内容提出质疑，质疑主要集中在联合索引的使用方式上。在那篇文章中，我说明联合索引是将各个索引字段做字符串连接后作为key，使用时将整体做前缀匹配。

而这名同学在这个页面找到了如下一句话：index condition pushdown is usually useful with multi-column indexes: the first component(s) is what index access is done for, the subsequent have columns that we read and check conditions on。从而认为联合索引的使用方式与文中不符。

实际上，这个页面所讲述的是在MariaDB 5.3.3（MySQL是在5.6）开始引入的一种叫做Index Condition Pushdown（以下简称ICP）的查询优化方式。由于本身不是一个层面的东西，前文中说的是Index Access，而这里是Query Optimization，所以并不构成对前文正确性的影响。在写前文时，MySQL还没有ICP，所以文中没有涉及相关内容，但考虑到新版本的MariaDB或MySQL中ICP的启用确实影响了一些查询行为的外在表现。所以决定写这篇文章详细讲述一下ICP的原理以及对索引使用方式的优化。

实验

先从一个简单的实验开始直观认识ICP的作用。

安装数据库

首先需要安装一个支持ICP的MariaDB或MySQL数据库。我使用的是MariaDB 5.5.34，如果是使用MySQL则需要5.6版本以上。

Mac环境下可以通过brew安装：

```
1.  brew install mairadb
```

其它环境下的安装请参考MariaDB官网关于下载安装的文档。

导入示例数据

与前文一样，我们使用Employees Sample Database，作为示例数据库。完整示例数据库的下载地址为：https://launchpad.net/test-db/employees-db-1/1.0.6/+download/employees_db-full-1.0.6.tar.bz2。

将下载的压缩包解压后，会看到一系列的文件，其中employees.sql就是导入数据的命令文件。执行

```
1.  mysql -h[host] -u[user] -p < employees.sql
```

就可以完成建库、建表和load数据等一系列操作。此时数据库中会多一个叫做employees的数据库。库中的表如下：

```
1.  MariaDB [employees]> SHOW TABLES;
2.  +----------------------+
3.  | Tables_in_employees |
4.  +----------------------+
5.  | departments          |
6.  | dept_emp             |
7.  | dept_manager         |
8.  | employees            |
9.  | salaries             |
10. | titles               |
11. +----------------------+
12. 6 rows in set (0.00 sec)
```

我们将使用employees表做实验。

建立联合索引

employees表包含雇员的基本信息，表结构如下：

```
MariaDB [employees]> DESC employees.employees;
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| emp_no     | int(11)       | NO   | PRI | NULL    |       |
| birth_date | date          | NO   |     | NULL    |       |
| first_name | varchar(14)   | NO   |     | NULL    |       |
| last_name  | varchar(16)   | NO   |     | NULL    |       |
| gender     | enum('M','F') | NO   |     | NULL    |       |
| hire_date  | date          | NO   |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

这个表默认只有一个主索引，因为ICP只能作用于二级索引，所以我们建立一个二级索引：

```
ALTER TABLE employees.employees ADD INDEX first_name_last_name (first_name, last_name);
```

这样就建立了一个first_name和last_name的联合索引。

查询

为了明确看到查询性能，我们启用profiling并关闭query cache：

```
SET profiling = 1;
SET query_cache_type = 0;
SET GLOBAL query_cache_size = 0;
```

然后我们看下面这个查询：

```
MariaDB [employees]> SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man';
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
| 254642 | 1959-01-17 | Mary       | Botman    | M      | 1989-11-24 |
| 471495 | 1960-09-24 | Mary       | Dymetman  | M      | 1988-06-09 |
| 211941 | 1962-08-11 | Mary       | Hofman    | M      | 1993-12-30 |
| 217707 | 1962-09-05 | Mary       | Lichtman  | F      | 1987-11-20 |
| 486361 | 1957-10-15 | Mary       | Oberman   | M      | 1988-09-06 |
| 457469 | 1959-07-15 | Mary       | Weedman   | M      | 1996-11-21 |
+--------+------------+------------+-----------+--------+------------+
```

根据MySQL索引的前缀匹配原则，两者对索引的使用是一致的，即只有first_name采用索引，last_name由于使用了模糊前缀，没法使用索引进行匹配。我将查询联系执行三次，结果如下：

```
+----------+------------+-------------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                         |
+----------+------------+-------------------------------------------------------------------------------+
|       38 | 0.00084400 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
|       39 | 0.00071800 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
|       40 | 0.00089600 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
+----------+------------+-------------------------------------------------------------------------------+
```

然后我们关闭ICP：

```
SET optimizer_switch='index_condition_pushdown=off';
```

在运行三次相同的查询，结果如下：

```
+----------+------------+-------------------------------------------------------------------------------+
| Query_ID | Duration   | Query                                                                         |
+----------+------------+-------------------------------------------------------------------------------+
|       42 | 0.00264400 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
|       43 | 0.01418900 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
|       44 | 0.00234200 | SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man' |
+----------+------------+-------------------------------------------------------------------------------+
```

有意思的事情发生了，关闭ICP后，同样的查询，耗时是之前的三倍以上。下面我们用explain看看两者有什么区别：

```
MariaDB [employees]> EXPLAIN SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man';
+------+-------------+-----------+------+---------------------+---------------------+---------+-------+------+-----------------
------+
| id   | select_type | table     | type | possible_keys       | key                 | key_len | ref   | rows | Extra
|
```

```
+------+-------------+-----------+------+----------------------+----------------------+---------+-------+------+-----------------------
------+
|    1 | SIMPLE      | employees | ref  | first_name_last_name | first_name_last_name | 44      | const |  224 | Using index
condition |
+------+-------------+-----------+------+----------------------+----------------------+---------+-------+------+-----------------------
------+
1 row in set (0.00 sec)
```

```
MariaDB [employees]> EXPLAIN SELECT * FROM employees WHERE first_name='Mary' AND last_name LIKE '%man';
+------+-------------+-----------+------+----------------------+----------------------+---------+-------+------+-------------+
| id   | select_type | table     | type | possible_keys        | key                  | key_len | ref   | rows | Extra       |
+------+-------------+-----------+------+----------------------+----------------------+---------+-------+------+-------------+
|    1 | SIMPLE      | employees | ref  | first_name_last_name | first_name_last_name | 44      | const |  224 | Using where |
+------+-------------+-----------+------+----------------------+----------------------+---------+-------+------+-------------+
1 row in set (0.00 sec)
```

前者是开启ICP，后者是关闭ICP。可以看到区别在于Extra，开启ICP时，用的是Using index condition；关闭ICP时，是Using where。

其中Using index condition就是ICP提高查询性能的关键。下一节说明ICP提高查询性能的原理。

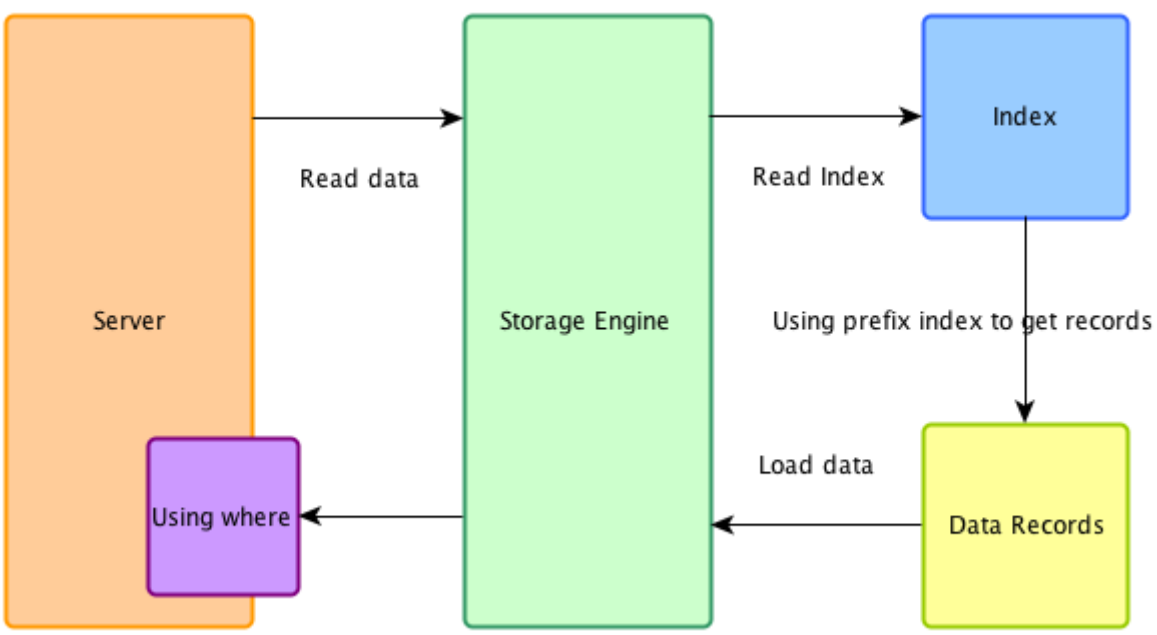
原理

ICP的原理简单说来就是将可以利用索引筛选的where条件在存储引擎一侧进行筛选，而不是将所有index access的结果取出放在server端进行where筛选。

以上面的查询为例，在没有ICP时，首先通过索引前缀从存储引擎中读出224条first_name为Mary的记录，然后在server段用where筛选last_name的like条件；而启用ICP后，由于last_name的like筛选可以通过索引字段进行，那么存储引擎内部通过索引与where条件的对比来筛选掉不符合where条件的记录，这个过程不需要读出整条记录，同时只返回给server筛选后的6条记录，因此提高了查询性能。

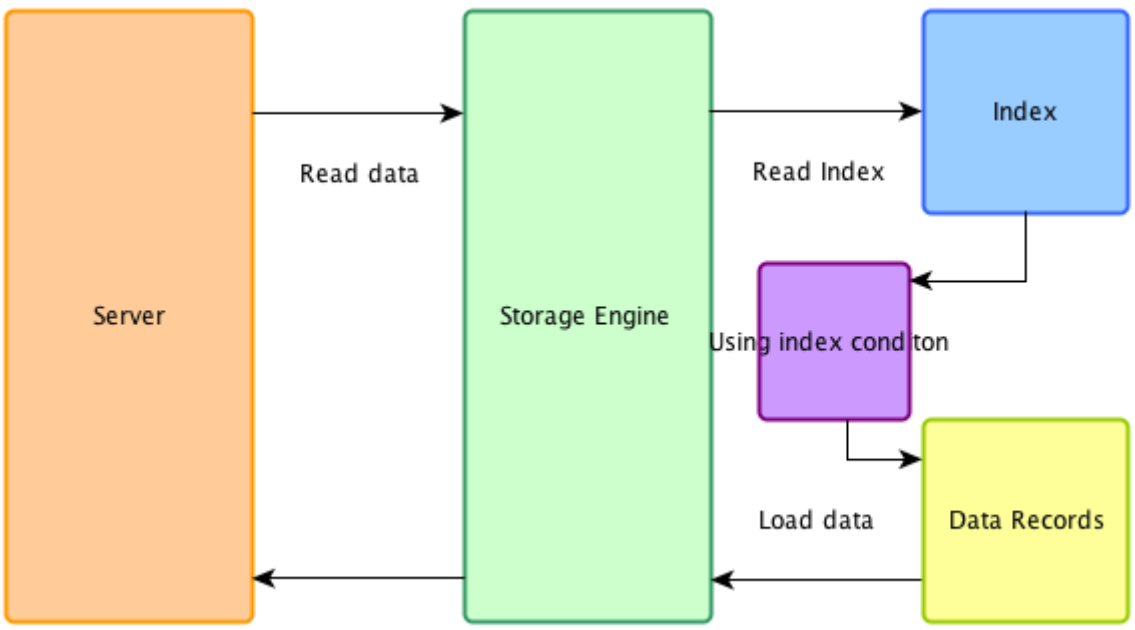
下面通过图两种查询的原理详细解释。

关闭ICP



在不支持ICP的系统下，索引仅仅作为data access使用。

开启ICP



在ICP优化开启时，在存储引擎端首先用索引过滤可以过滤的where条件，然后再用索引做data access，被index condition过滤掉的数据不必读取，也不会返回server端。

注意事项

有几个关于ICP的事情要注意：

- ICP只能用于二级索引，不能用于主索引。
- 也不是全部where条件都可以用ICP筛选，如果某where条件的字段不在索引中，当然还是要读取整条记录做筛选，在这种情况下，仍然要到server端做where筛选。
- 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。

参考

[1] https://mariadb.com/kb/en/index-condition-pushdown/

[2] http://dev.mysql.com/doc/refman/5.6/en/index-condition-pushdown-optimization.html