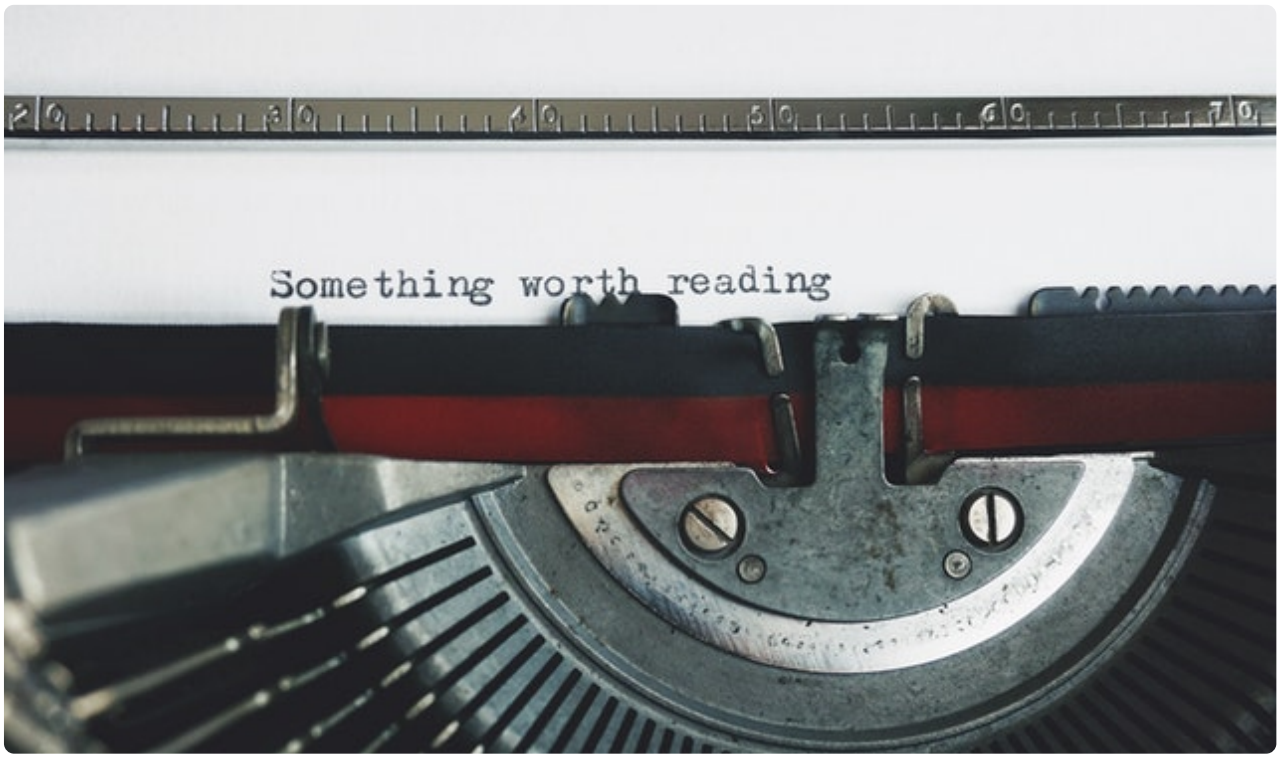
25 加速 order by 查询，可以从哪些方面做优化呢？

更新时间：2020-05-11 09:27:46



> 耐心是一切聪明才智的基础。——柏拉图

ORDER BY 是用于对查询结果集排序（忽略正序和逆序，原理是一样的）的关键字，它同样也是非常高频的出现在我们的日常工作、学习中。我们在使用 ORDER BY 的“经历”中，大概率会遇到查询缓慢的性能问题。那么，你知道为什么 ORDER BY 会慢吗 ？如果想要提高性能，我们又可以怎样去做呢 ？这一节里，我们就来详细的解读下关于 ORDER BY 查询的问题。

1. ORDER BY 的执行过程

就好像我们在写代码一样，出现问题了，先要去看异常日志，再去看代码的执行逻辑，知道了它是怎么执行的之后，再去解决问题。所以，我们暂时不要去想着优化 ORDER BY 查询，先把它的执行过程搞清楚，再去分析“慢”的根本原因，最后才去做合理的优化。

1.1 全字段排序

为了更好的对执行过程（查询语句）进行说明，我们首先去创建一张数据表 worker，建表语句如下：

```
CREATE TABLE `worker` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT COMMENT 'id',
  `type` char(64) NOT NULL DEFAULT '' COMMENT '员工类型',
  `name` char(64) NOT NULL DEFAULT '' COMMENT '姓名',
  `salary` bigint(20) unsigned DEFAULT '0' COMMENT '薪水',
  `province` char(64) NOT NULL DEFAULT '' COMMENT '省份',
  `city` char(64) NOT NULL DEFAULT '' COMMENT '城市',
  PRIMARY KEY (`id`),
  KEY `city_idx` (`city`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='员工表';
```

从建表语句可以得知，MySQL 会为 worker 表创建两个索引 B+ 树：聚簇（主键）索引和普通（city、id）索引。下面，我们可以对 worker 表进行如下的排序查询：

```
-- 注意到 WHERE 条件可以使用到 city_idx 索引
SELECT type, name, salary FROM worker WHERE city = '宿州市' ORDER BY name LIMIT 100;
```

MySQL 对于排序过程会给每个线程分配一块内存，称之为 sort_buffer，注意，这里还并未区分排序类型。接下来，我就去对上面的查询语句进行全字段排序的过程进行讲解说明：

- MySQL 初始化 sort_buffer，并根据查询语句确定需要放入 `type`，`name`，`salary` 三个字段
- 从 city_idx 索引中找到第一个满足 `city = '宿州市'` 条件的主键
- 根据找到的主键回到聚簇索引中取出完整的数据记录，并把 `type`，`name`，`salary` 三个字段的值放入 sort_buffer 中
- 从 city_idx 索引中取出下一个记录的主键
- 重复步骤3和4，直到 `city = '宿州市'` 的条件不满足
- 对 sort_buffer 中的数据按照字段 `name` 做快速排序
- 按照排序结果取前100条返回给客户端

需要注意，“按照 name 排序”的动作可能在内存中完成，也可能会借助临时文件使用外部排序完成，这完全取决于排序所需的内存和 MySQL 参数 sort_buffer_size（也就是开辟 sort_buffer 内存的大小）。

1.2 rowid 排序

对于“全字段排序”算法来说，存在一个不可忽视的问题：如果查询需要返回很多字段，那么，sort_buffer 中需要放入的数据也会很多。一旦超过了 sort_buffer_size 定义的阈值，就需要使用临时文件去做外排序，性能将会严重降低。所以，如果单行数据很大，“全字段排序”的方法并不好。

MySQL 当然也考虑到了这个问题，它定义了 max_length_for_sort_data 参数去控制用于排序的行数据的长度。这个参数所表达的意思是：如果单行的数据长度超过了预定义的值，MySQL 就会认为单行太大，需要换一种排序算法，即“rowid 排序”。我们可以去看一看这个参数：

```
mysql> SHOW VARIABLES LIKE 'max_length_for_sort_data';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| max_length_for_sort_data | 1024  |
+--------------------------+-------+
1 row in set (0.01 sec)
```

那么，假设查询的三个字段 `type`，`name`，`salary` 总长度超过了 max_length_for_sort_data 阈值，MySQL 将会采用“rowid 排序”算法。执行流程总结如下：

- MySQL 初始化 sort_buffer，但是只会放入 name 和 id 两个字段（这也是称为 rowid 排序的原因）
- 从 city_idx 索引中找到第一个满足 city='宿州市' 条件的主键
- 根据找到的主键回到聚簇索引中取出完整的数据记录，并把 name, id 两个字段的值放入 sort_buffer 中
- 从 city_idx 索引中取出下一个记录的主键
- 重复步骤 3 和 4，直到 city = '宿州市' 的条件不满足
- 对 sort_buffer 中的数据按照字段 name 做快速排序
- 遍历排序结果，取前 100 条，并根据 id 再回到聚簇索引中取出 `type`, `name`, `salary` 三个字段返回给客户端

所以，可以知道，rowid 排序相比于全字段排序，除了基本的排序过程之外，还需要根据 id 再回到原表中将完整的数据列值取出。

1.3 全字段与 rowid 排序的对比总结

其实，从以上对这两种排序算法的介绍可以知道：MySQL 依赖于内存空间是否足够去决定选择哪一种排序方式。当然，我们也可以知道，“全字段排序” 的方式肯定是更优的。下面，我来对这两种排序方式进行对比总结：

- “全字段排序” 是 MySQL 的首选，也体现了 MySQL 的设计思想：内存足够，就更多的利用内存，尽量减少磁盘访问（这同样也对我们的程序设计有启发意义）
- 如果 MySQL 认为内存不够用，影响排序效率，则会采用 “rowid 排序”，但是会多出一次回表过程

2. ORDER BY 与索引的关系

索引不仅能够提高普通查询的性能，它当然也能够影响到排序，但是，`ORDER BY` 如何结合索引有时并不好理解。所以，接下来，我们将以实例的形式分析各种关于 `ORDER BY` 的查询语句是否会使用到索引。以此来启发我们对数据表、索引、查询方式的设计。

2.1 ORDER BY 使用索引的情况分析

在某些情况下，MySQL 可能会使用索引来满足一个 ORDER BY 子句，避免额外的排序性能消耗。但是，这建立在你对业务需求足够理解的基础上，即能够预判会有怎样的查询，再去建立合适的索引。例如，对于之前的排序查询：

```
SELECT type, name, salary FROM worker WHERE city = '宿州市' ORDER BY name LIMIT 100;
```

由于当前的 worker 表只有 city 字段存在索引（忽略聚簇索引），所以，MySQL 不可避免的会对表扫描并排序。但是，如果此时我们给 worker 表添加一个额外的索引：

```
mysql> ALTER TABLE `worker` ADD INDEX city_name_idx(`city`, `name`);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

此时，由于 `city`, `name` 这两个字段有一个联合索引存在，且 WHERE 子句之后的 city 条件是一个常量，所以，在找到 `city = '宿州市'` 对应的记录之后，就可以按顺序从索引中返回前 100 条记录（需要再回到聚簇索引中取得完整的记录），而不需要再去排序了。其实，针对当前的查询，我们还可以再做一次优化。给 worker 表添加如下索引：

```
mysql> ALTER TABLE `worker` ADD INDEX city_nam_type_salary_idx(`city`, `name`, `type`, `salary`);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

你一定也看出来了，这其实就是利用到了覆盖索引的特性。由于 city_nam_type_salary_idx 索引上的信息能够满足查询要求，就不需要再回到聚簇索引上取数据。

下面，我们再来看两个排序查询 SQL 语句：

```
SELECT type, name, salary FROM worker ORDER BY city, name LIMIT 100;
SELECT id, type, name, salary FROM worker ORDER BY city, name LIMIT 100;
```

虽然第二条 SQL 会比第一条多查询出一个 id 字段，但是，在 MySQL 看来，它们是一模一样的，都会使用到 city_nam_type_salary_idx 索引（如果创建了这个索引）。这是因为：在 MySQL 中，不论是什么类型的索引，都会包含主键。

最后，还需要去考虑 ORDER BY 对多字段的排序规则问题。默认情况下，如果不显示的指定 ORDER BY 的规则，则 MySQL 使用的是 ASC，即升序。对于多字段排序的情况下，需要各自分别定义升序或降序。所以，如果想要使用索引，ORDER BY 所有字段的排序规则必须是一致的。例如：

```
SELECT id, type, name, salary FROM worker ORDER BY city, name LIMIT 100;
SELECT id, type, name, salary FROM worker ORDER BY city ASC, name ASC LIMIT 100;
SELECT id, type, name, salary FROM worker ORDER BY city DESC, name DESC LIMIT 100;
```

2.2 ORDER BY 不使用索引的情况分析

这里所说的 ORDER BY 不使用索引其实指的是需要对查询结果进行排序。首先，你肯定能想到的“反面案例”就是多字段排序规则不一致的查询。例如：

```
SELECT id, type, name, salary FROM worker ORDER BY city ASC, name DESC LIMIT 100;
SELECT id, type, name, salary FROM worker ORDER BY city DESC, name ASC LIMIT 100;
```

另外，当 ORDER BY 字段使用函数时，优化器解析 ORDER BY 时也会放弃索引。例如：

```
SELECT type, name, salary FROM worker WHERE city = '宿州市' ORDER BY LOWER(name) LIMIT 100;
```

当 ORDER BY 与 GROUP BY 一起使用时，如果它们有不同的表达式，即使有对应的索引，也不能使用。例如：

```
SELECT type, name FROM worker WHERE city = '宿州市' GROUP BY type, name ORDER BY name LIMIT 100;
```

最后一种情况是，对于指定了排序索引长度的索引，索引不能够完全解析排序的顺序，仍然需要使用排序算法来对结果进行排序。例如，我们在 name 字段上建立了索引 name(10)，而实际上 name 的长度是 64。那么，对于 ORDER BY name 的情况也是不能够使用索引来直接完成查询的。

3. 如何去做 ORDER BY 查询的优化

MySQL 通过系统变量和系统表提供了对查询语句的执行跟踪，这类似于我们的应用日志。通过执行跟踪信息，我们可以对影响 ORDER BY 执行的参数阈值进行调整，以达到优化查询的目的。

3.1 跟踪 ORDER BY 查询的执行过程

对于我们的 SQL 执行过程，除了基本的日志之外，MySQL 还提供了“跟踪”的功能。这个功能默认是关闭的，我们需要的时候可以打开。如下所示：

```
-- 打开 optimizer_trace，且只对当前的会话有效
mysql> SET optimizer_trace = 'enabled=on';
Query OK, 0 rows affected (0.01 sec)
```

系统库 performance_schema 中的 session_status 表中记录了 InnoDB 读取的行记录数。这是一个非常有用的数据，我们可以在执行一条 SQL 语句的前后获取到这个值，然后计算它们的差值，而这个差值就标识了 InnoDB 扫描的记录数。首先，在执行查询之前，先把“当前值”记录下来。执行如下语句：

```
mysql> SELECT VARIABLE_VALUE INTO @a FROM performance_schema.session_status WHERE variable_name = 'Innodb_rows_read';
Query OK, 1 row affected (0.00 sec)
```

执行之后，你可以使用 SELECT @a 查看下这个值，我这里不再演示。记录下“当前值”之后，我们可以去执行查询语句了。例如（可以考虑 fake 多一些数据到测试表中，也是为了更好的观测结果）：

```
SELECT type, name, salary FROM worker WHERE city = '宿州市' ORDER BY name LIMIT 100;
```

此时，我们就可以看一看 MySQL 的跟踪结果信息。需要注意，由于跟踪信息的内容比较多，所以，我只截取了部分结果信息。如下所示（包含注释信息）：

```
mysql> SELECT * FROM `information_schema`.`OPTIMIZER_TRACE`\G
......
-- 排序信息总结（如果使用到了索引，就不会有排序信息）
"filesort_summary": {
  -- 记录行数
  "rows": 121350,
  -- 参与排序的行数
  "examined_rows": 121350,
  -- 使用到的临时文件个数
  "number_of_tmp_files": 24,
  -- MySQL 为排序开辟的内存空间
  "sort_buffer_size": 73936,
  -- 使用 rowid 排序算法
  "sort_mode": "<sort_key, rowid>"
}
......
```

根据跟踪信息可以知道：查询语句在排序过程中使用了 24 个（number_of_tmp_files）临时文件。当内存不够时（排序数据超过了 sort_buffer_size 设定的阈值），MySQL 会使用临时文件做外部排序，而外部排序一般使用归并算法来完成。这也是效率低下的主要原因。

最后，我们再去获取 InnoDB 读取的行记录数，并把这个值保存在变量 b 中。同时，计算两次读取的差值，也就获取了对于之前的查询 InnoDB 扫描的记录数。如下所示：

```
mysql> SELECT VARIABLE_VALUE INTO @b FROM performance_schema.session_status WHERE variable_name = 'Innodb_rows_read';
Query OK, 1 row affected (0.00 sec)

-- 获取 InnoDB 扫描的记录数，注意，结果完全看你当前表中的记录数
mysql> SELECT @b-@a;
+-------+
| @b-@a |
+-------+
| 121350|
+-------+
1 row in set (0.00 sec)
```

3.2 优化 ORDER BY 查询的建议

优化 ORDER BY 查询的原因一定是它执行的比较慢，那么，为什么会慢，我们就可以通过以上讲解的跟踪过程来确定。确定了瓶颈出现的原因，我们就可以去想办法优化了。下面，我来给出关于优化 ORDER BY 查询的一些建议：

- 调低 max_length_for_sort_data 参数的值，控制 MySQL 排序选择算法的触发点
- 调大 sort_buffer_size 参数的值，理想情况下，这个值越大越好，使得排序过程只发生在内存中
- 根据业务需求（确定常用查询）创建合适的索引，最好的情况是可以使用覆盖索引
- 排序的字段不应该过多，复杂的查询会占据大量的时间，且通常很难做优化

4. 总结

ORDER BY 查询确实是有很多“技巧”的，这不仅仅是需要对它的执行原理非常了解，还需要熟悉你正在做的业务需求。另外，虽然调整 MySQL 的一些参数可以起到优化查询的目的，但是，也同样需要考虑这会对其他查询或客户端造成的影响。

5. 问题

你遇到过 ORDER BY 查询缓慢的问题吗 ？你是怎样做优化的呢 ？

调整 sort_buffer_size 参数的值可以优化 ORDER BY 查询，但是，应该调整到多少呢 ？

除了我这一节里提到的优化建议，你还能说出哪些 ？

6. 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：Sorting Rows

MySQL 官方文档：Server System Variables

MySQL 官方文档：ORDER BY Optimization

}

24 大数据量插入遇到瓶颈，我该怎样做性能优化呢？

26 遇到慢查询问题，可以这样思考与解决