特别说明

使用 Hive 可以高效而又快速地编写复杂的 MapReduce 查询逻辑。但是某些情况下，因为不熟悉数据特性，或没有遵循 Hive 的优化约定，Hive 计算任务会变得非常低效，甚至无法得到结果。一个"好"的 Hive 程序仍然需要对 Hive 运行机制有深入的了解。

有一些大家比较熟悉的优化约定包括：Join 中需要将大表写在靠右的位置；尽量使用 UDF 而不是 transfrom……诸如此类。下面讨论 5 个性能和逻辑相关的问题，帮助你写出更好的 Hive 程序。

全排序

Hive 的排序关键字是 SORT BY，它有意区别于传统数据库的 ORDER BY 也是为了强调两者的区别–SORT BY 只能在单机范围内排序。考虑以下表定义：

```
CREATE TABLE if not exists t_order(

id int, -- 订单编号

sale_id int, -- 销售 ID

customer_id int, -- 客户 ID

product _id int, -- 产品 ID

amount int -- 数量

) PARTITIONED BY (ds STRING);
```

在表中查询所有销售记录，并按照销售 ID 和数量排序：

```
set mapred.reduce.tasks=2;

Select sale_id, amount from t_order

Sort by sale_id, amount;
```

这一查询可能得到非期望的排序。指定的 2 个 reducer 分发到的数据可能是（各自排序）：

Reducer1：

```
Sale_id | amount

0 | 100

1 | 30

1 | 50

2 | 20
```

Reducer2：

```
Sale_id | amount

0 | 110

0 | 120
```

```
3 | 50

4 | 20
```

因为上述查询没有 reduce key，hive 会生成随机数作为 reduce key。这样的话输入记录也随机地被分发到不同 reducer 机器上去了。为了保证 reducer 之间没有重复的 sale_id 记录，可以使用 DISTRIBUTE BY 关键字指定分发 key 为 sale_id。改造后的 HQL 如下：

```
set mapred.reduce.tasks=2;

Select sale_id, amount from t_order
Distribute by sale_id
Sort by sale_id, amount;
```

这样能够保证查询的销售记录集合中，销售 ID 对应的数量是正确排序的，但是销售 ID 不能正确排序，原因是 hive 使用 hadoop 默认的 HashPartitioner 分发数据。

这就涉及到一个全排序的问题。解决的办法无外乎两种：

1.) 不分发数据，使用单个 reducer：

```
set mapred.reduce.tasks=1;
```

这一方法的缺陷在于 reduce 端成为了性能瓶颈，而且在数据量大的情况下一般都无法得到结果。但是实践中这仍然是最常用的方法，原因是通常排序的查询是为了得到排名靠前的若干结果，因此可以用 limit 子句大大减少数据量。使用 limit n 后，传输到 reduce 端（单机）的数据记录数就减少到 n* （map 个数）。

2.) 修改 Partitioner，这种方法可以做到全排序。这里可以使用 Hadoop 自带的 TotalOrderPartitioner（来自于 Yahoo!的 TeraSort 项目），这是一个为了支持跨 reducer 分发有序数据开发的 Partitioner，它需要一个 SequenceFile 格式的文件指定分发的数据区间。如果我们已经生成了这一文件（存储在/tmp/range_key_list，分成 100 个 reducer），可以将上述查询改写为

```
set mapred.reduce.tasks=100;

set hive.mapred.partitioner=org.apache.hadoop.mapred.lib.TotalOrderPartitioner;

set total.order.partitioner.path=/tmp/ range_key_list;

Select sale_id, amount from t_order
Cluster by sale_id
Sort by amount;
```

有很多种方法生成这一区间文件（例如 hadoop 自带的 o.a.h.mapreduce.lib.partition.InputSampler 工具）。这里介绍用 Hive 生成的方法，例如有一个按 id 有序的 t_sale 表：

```
CREATE TABLE if not exists t_sale (
```

```
id int,
name string,
loc string
);
```

则生成按 sale_id 分发的区间文件的方法是：

```
create external table range_keys(sale_id int)
row format serde
'org.apache.hadoop.hive.serde2.binarysortable.BinarySortableSerDe'
stored as
inputformat
'org.apache.hadoop.mapred.TextInputFormat'
outputformat
'org.apache.hadoop.hive.ql.io.HiveNullValueSequenceFileOutputFormat'
location '/tmp/range_key_list';

insert overwrite table range_keys
select distinct sale_id
from source t_sale sampletable(BUCKET 100 OUT OF 100 ON rand()) s
sort by sale_id;
```

生成的文件（/tmp/range_key_list 目录下）可以让 TotalOrderPartitioner 按 sale_id 有序地分发 reduce 处理的数据。区间文件需要考虑的主要问题是数据分发的均衡性，这有赖于对数据深入的理解。

怎样做笛卡尔积？

当 Hive 设定为严格模式（hive.mapred.mode=strict）时，不允许在 HQL 语句中出现笛卡尔积，这实际说明了 Hive 对笛卡尔积支持较弱。因为找不到 Join key，Hive 只能使用 1 个 reducer 来完成笛卡尔积。

当然也可以用上面说的 limit 的办法来减少某个表参与 join 的数据量，但对于需要笛卡尔积语义的需求来说，经常是一个大表和一个小表的 Join 操作，结果仍然很大（以至于无法用单机处理），这时 MapJoin 才是最好的解决办法。

MapJoin，顾名思义，会在 Map 端完成 Join 操作。这需要将 Join 操作的一个或多个表完全读入内存。

MapJoin 的用法是在查询/子查询的 SELECT 关键字后面添加/*+ MAPJOIN(tablelist) */提示优化器转化为 MapJoin（目前 Hive 的优化器不能自动优化 MapJoin）。其中 tablelist 可以是一个表，或以逗号连接的表的列表。tablelist 中的表将会读入内存，应该将小表写在这里。

PS：有用户说 MapJoin 在子查询中可能出现未知 BUG。在大表和小表做笛卡尔积时，规避笛卡尔积的方法是，给 Join 添加一个 Join key，原理很简单：将小表扩充一列 join key，并将小表的条目复制数倍，join key 各不相同；将大表扩充一列 join key 为随机数。

怎样写 exist in 子句？

Hive 不支持 where 子句中的子查询，SQL 常用的 exist in 子句需要改写。这一改写相对简单。考虑以下 SQL 查询语句：

```
SELECT a.key, a.value

FROM a

WHERE a.key in

(SELECT b.key

FROM B);
```

可以改写为

```
SELECT a.key, a.value

FROM a LEFT OUTER JOIN b ON (a.key = b.key)

WHERE b.key  NULL;
```

一个更高效的实现是利用 left semi join 改写为：

```
SELECT a.key, a.val

FROM a LEFT SEMI JOIN b on (a.key = b.key);
```

left semi join 是 0.5.0 以上版本的特性。

Hive 怎样决定 reducer 个数？

Hadoop MapReduce 程序中，reducer 个数的设定极大影响执行效率，这使得 Hive 怎样决定 reducer 个数成为一个关键问题。遗憾的是 Hive 的估计机制很弱，不指定 reducer 个数的情况下，Hive 会猜测确定一个 reducer 个数，基于以下两个设定：

1. hive.exec.reducers.bytes.per.reducer（默认为 1000^3）

2. hive.exec.reducers.max（默认为 999）

计算 reducer 数的公式很简单：

N=min(参数 2，总输入数据量/参数 1)

通常情况下，有必要手动指定 reducer 个数。考虑到 map 阶段的输出数据量通常会比输入有大幅减少，因此即使不设定 reducer 个数，重设参数 2 还是必要的。依据 Hadoop 的经验，

可以将参数 2 设定为 0.95*(集群中 TaskTracker 个数)。

合并 MapReduce 操作

Multi-group by

Multi-group by 是 Hive 的一个非常好的特性，它使得 Hive 中利用中间结果变得非常方便。例如，

```
FROM (SELECT a.status, b.school, b.gender
FROM status_updates a JOIN profiles b
ON (a.userid = b.userid and
a.ds='2009-03-20' )
) subq1
INSERT OVERWRITE TABLE gender_summary
PARTITION(ds='2009-03-20')
SELECT subq1.gender, COUNT(1) GROUP BY subq1.gender
INSERT OVERWRITE TABLE school_summary
PARTITION(ds='2009-03-20')
SELECT subq1.school, COUNT(1) GROUP BY subq1.school
```

上述查询语句使用了 Multi-group by 特性连续 group by 了 2 次数据，使用不同的 group by key。这一特性可以减少一次 MapReduce 操作。

Multi-distinct

Multi-distinct 是淘宝开发的另一个 multi-xxx 特性，使用 Multi-distinct 可以在同一查询/子查询中使用多个 distinct，这同样减少了多次 MapReduce 操作。