hive数据倾斜原因和解决方法 - CSDN博客

在做Shuffle阶段的优化过程中，遇到了数据倾斜的问题，造成了对一些情况下优化效果不明显。主要是因为在Job完成后的所得到的Counters是整个Job的总和，优化是基于这些Counters得出的平均值，而由于数据倾斜的原因造成map处理数据量的差异过大，使得这些平均值能代表的价值降低。Hive的执行是分阶段的，map处理数据量的差异取决于上一个stage的reduce输出，所以如何将数据均匀的分配到各个reduce中，就是解决数据倾斜的根本所在。规避错误来更好的运行比解决错误更高效。在查看了一些资料后，总结如下。

1数据倾斜的原因

1.1操作：

关键词	情形	后果
Join	其中一个表较小，但是key集中	分发到某一个或几个Reduce上的数据远高于平均值
大表与大表，但是分桶的判断字段0值或空值过多	这些空值都由一个reduce处理，灾常慢	
group by	group by 维度过小，某值的数量过多	处理某值的reduce灾常耗时
Count Distinct	某特殊值过多	处理此特殊值的reduce耗时

1.2原因：

1)、key分布不均匀

2)、业务数据本身的特性

3)、建表时考虑不周

4)、某些SQL语句本身就有数据倾斜

1.3表现：

任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。

单一reduce的记录数与平均记录数差异过大，通常可能达到3倍甚至更多。 最长时长远大于平均时长。

2数据倾斜的解决方案

2.1参数调节：

hive.map.aggr = true

Map 端部分聚合，相当于Combiner

hive.groupby.skewindata=true

有数据倾斜的时候进行负载均衡，当选项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

2.2 SQL语句调节：

如何Join：

关于驱动表的选取，选用join key分布最均匀的表作为驱动表

做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小的效果。

大小表Join：

使用map join让小的维度表（1000条以下的记录条数） 先进内存。在map端完成reduce.

大表Join大表：

把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

count distinct大量相同特殊值

count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union。

group by维度过小：

采用sum() group by的方式来替换count(distinct)完成计算。

特殊情况特殊处理：

在业务逻辑优化效果的不大情况下，有些时候是可以将倾斜的数据单独拿出来处理。最后union回去。

3典型的业务场景

3.1空值产生的数据倾斜

场景：如日志中，常会有信息丢失的问题，比如日志中的 user_id，如果取其中的 user_id 和 用户表中的user_id 关联，会碰到数据倾斜的问题。

解决方法1： user_id为空的不参与关联（红色字体为修改后）

```
select * from log a
 join users b
 on a.user_id is not null
 and a.user_id = b.user_id
union all
select * from log a
 where a.user_id is null;
```

解决方法2 ：赋与空值分新的key值

```
select *
 from log a
 left outer join users b
 on case when a.user_id is null then concat('hive',rand() ) else a.user_id end = b.user_id;
```

结论：方法2比方法1效率更好，不但io少了，而且作业数也少了。解决方法1中 log读取两次，jobs是2。解决方法2 job数是1 。这个优化适合无效 id (比如 -99 , '', null 等) 产生的倾斜问题。把空值的 key 变成一个字符串加上随机数，就能把倾斜的数据分到不同的reduce上 ,解决数据倾斜问题。

3.2不同数据类型关联产生数据倾斜

场景：用户表中user_id字段为int，log表中user_id字段既有string类型也有int类型。当按照user_id进行两个表的Join操作时，默认的Hash操作会按int型的id来进行分配，这样会导致所有string类型id的记录都分配到一个Reducer中。

解决方法：把数字类型转换成字符串类型

```
select * from users a
 left outer join logs b
 on a.usr_id = cast(b.user_id as string)
```

3.3小表不小不大，怎么用 map join 解决倾斜问题

使用 map join 解决小表(记录数少)关联大表的数据倾斜问题，这个方法使用的频率非常高，但如果小表很大，大到map join会出现bug或异常，这时就需要特别的处理。以下例子:

```
select * from log a
 left outer join users b
 on a.user_id = b.user_id;
```

users 表有 600w+ 的记录，把 users 分发到所有的 map 上也是个不小的开销，而且 map join 不支持这么大的小表。如果用普通的

join，又会碰到数据倾斜的问题。

解决方法：

```
select /*+mapjoin(x)*/* from log a
 left outer join (
 select /*+mapjoin(c)*/d.*
 from ( select distinct user_id from log ) c
 join users d
 on c.user_id = d.user_id
 ) x
 on a.user_id = b.user_id;
```

假如，log里user_id有上百万个，这就又回到原来map join问题。所幸，每日的会员uv不会太多，有交易的会员不会太多，有点击的会员不会太多，有佣金的会员不会太多等等。所以这个方法能解决很多场景下的数据倾斜问题。

4总结

使map的输出数据更均匀的分布到reduce中去，是我们的最终目标。由于Hash算法的局限性，按key Hash会或多或少的造成数据倾斜。大量经验表明数据倾斜的原因是人为的建表疏忽或业务逻辑可以规避的。在此给出较为通用的步骤：

1、采样log表，哪些user_id比较倾斜，得到一个结果表tmp1。由于对计算框架来说，所有的数据过来，他都是不知道数据分布情况的，所以采样是并不可少的。

2、数据的分布符合社会学统计规则，贫富不均。倾斜的key不会太多，就像一个社会的富人不多，奇特的人不多一样。所以tmp1记录数会很少。把tmp1和users做map join生成tmp2,把tmp2读到distribute file cache。这是一个map过程。

3、map读入users和log，假如记录来自log,则检查user_id是否在tmp2里，如果是，输出到本地文件a,否则生成的key,value对，假如记录来自member,生成的key,value对，进入reduce阶段。

4、最终把a文件，把Stage3 reduce阶段输出的文件合并起写到hdfs。

如果确认业务需要这样倾斜的逻辑，考虑以下的优化方案：

1、对于join，在判断小表不大于1G的情况下，使用map join

2、对于group by或distinct，设定 hive.groupby.skewindata=true

3、尽量使用上述的SQL语句调节进行优化