性能优化案例总结

1、计算数据优化

计算数据优化主要有两种思路，一种是减少处理数据量；一种是解决数据倾斜。数据倾斜一般可以分为三种：

- Mapper阶段数据倾斜
- Join阶段数据倾斜
- Reduce阶段数据倾斜

1.1 Mapper阶段数据倾斜

主要有两种方式：

- 可以修改读取数据的表的任务，最后插入数据时按照均衡的key值重新分布。也就是在最后加上distribute by ***
- 如果Mapper的任务数比较少，比如200以内，可以考虑增加Mapper的任务数，从而减少单个任务的处理数据量和执行时间。（详见第三部分的计算资源优化）

1.2 Join阶段数据倾斜

是最常见的数据倾斜，按照表的大小和join方式的不同分别有多种处理方式。

1.2.1 Mapjoin

作用：

- mapjoin可以把小表全部读入到内存，然后发送到大表所在的服务器处理，所有的逻辑都在map端完成，不需要reduce端从而避免倾斜。

适用场景：

- 大表inner join小表
- 大表left outer join小表
- 小表right outer join大表

不适用场景：

- 小表left outer join大表

- 大表right outer join小表
- 小表full outer join大表

限制：

- hive小表默认大小不能超过25M，可以调整hive.mapjoin.smalltable.filesize来。
- hive 0.7版本以后，可以通过设置set hive.auto.convert.join = true 自动优化。

具体案例

```
HQL_STR="
set hive.auto.convert.join = true
use gulfstream_test;
create table tmp_youhua_test1 as
select
      a.order_id
      ,b.fst_cat_id
from
(select
      order_id
      ,channel
from gulfstream_dwd.dwd_order_call_grab_d
where concat(year,month)='201805'
)a
left outer join
(
  select
        channel_id
        ,fst_cat_id
        ,fst_cat_name
        ,sec_cat_id
        ,sec_cat_name
  from gulfstream_dw.dim_order_channel_s
  where concat(year,month,day)='20180501'
)b
on a.channel = b.channel_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

使用mapjoin，只有map，没有reduce任务，运行时长只有2分钟左右。如果不使用mapjoin，任务运行时长需要30分钟。

```
MapReduce Total cumulative CPU time: 1 minutes 33 seconds 930 msec
Ended Job = job_1528097742108_7139411
Moving data to: hdfs://DClusterNmg4/user/hadoop/warehouse/tmp_youhua_test11
Failed with exception Unable to move source hdfs://DClusterNmg4/tmp-DClusterNmg4/hive-staging/hadoop_hive_2018-06-
21_20-07-15_197_988354099244014970-1/-ext-10001 to destination hdfs://DClusterNmg4/user/hadoop/warehouse/tmp_youhu
a_test11
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.MoveTask
MapReduce Jobs Launched:
Stage-Stage-9: Map: 572   Cumulative CPU: 7505.05 sec   HDFS Read: 2497874540 HDFS Write: 2102871877 SUCCESS
Stage-Stage-3: Map: 9   Cumulative CPU: 93.93 sec   HDFS Read: 2121538478 HDFS Write: 2102724995 SUCCESS
Total MapReduce CPU Time Spent: 0 days 2 hours 6 minutes 38 seconds 980 msec
```

1.2.2 Mapjoin+left join

作用：

- 主要是为了解决小表在left outer join左侧从而无法直接使用mapjoin的问题。通过”小表 left outer join(小表 mapjoin 大表)”这样的方式来实现优化。

适用场景：

- 小表left outer join大表
- 大表right outer join小表

具体案例：

优化前：运行时长为20分钟。

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test1;
create table tmp_youhua_test1 as
select
      a.fst_cat_id
      ,count(order_id) cnt
from
(select
        channel_id
        ,fst_cat_id
  from gulfstream_dw.dim_order_channel_s
  where concat(year,month,day)='20180501'
)a
left outer join
(
 select
       order_id
       ,channel
 from gulfstream_dwd.dwd_order_call_grab_d
 where concat(year,month)='201804'
)b
on a.channel_id = b.channel
group by fst_cat_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

优化后：运行时长为4分钟。

```
HQL_STR="
set hive.auto.convert.join = true
use gulfstream_test;
drop table tmp_youhua_test2;
create table tmp_youhua_test2 as
select
      fst_cat_id
      ,cnt
from
(select
        channel_id
        ,fst_cat_id
  from gulfstream_dw.dim_order_channel_s
  where concat(year,month,day)='20180501'
)a
left outer join
(
 select
       fst_cat_id
       ,count(order_id) cnt
 from
 (select
        channel_id
        ,fst_cat_id
 from gulfstream_dw.dim_order_channel_s
 where concat(year,month,day)='20180501'
 )a
 join
 (
  select
       order_id
       ,channel
  from gulfstream_dwd.dwd_order_call_grab_d
  where concat(year,month)='201804'
 )b
on a.channel_id=b.channel
group by fst_cat_id
)c
on a.fst_cat_id = c.fst_cat_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

1.2.3 先group by再mapjoin

作用：

- 先把某个大表按照关联key做group by，来减少关联时的数据量，然后把group by的结果当作mapjoin的小表再做关联。

适用场景：

- 大表a左关联/内关联大表b，其中大表b的join key的distinct值数量比较少。

优化方式：

- 大表 a mapjoin (select key,value from 大表b group by key)

1.2.4 先过滤在关联

作用：

- 把关联后不需要的数据在关联前先过滤掉

适用场景：

- 大表a left outer join大表b，大表b的关联key有关联不上的数据导致倾斜，比如大表b的join key有很多空值情况，可以先将大表b的空值先做过滤在关联。

不适用场景：

- 上述大表b在left outer join 的左侧。

1.2.5 关联key随机化处理

作用：

- 把key值关联不上，但是需要保留的那部分数据的关联key随机化处理之后再去关联，这样就可以避免相同key数据量太大导致数据倾斜。

适用场景：

- 大表a left outer join大表b，大表a的关联key有关联不上的数据导致倾斜，比如大表a的关联key有很多空值。

具体案例：

2017.10.1期间，存在大量呼单未应答的情况，driver_id key值为0的记录数比之前多了十倍，数据量级达到几千万，导致关联时存在了数据倾斜。

优化前：

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test3;
create table tmp_youhua_test3 as
select
      a.order_id
      ,car_level
from
(select order_id
        ,driver_id
from  gulfstream_ods.g_order
where concat(year,month,day)='20180514'
)a
left outer join
(
  select
        driver_id
        ,max(driver_car_level) as car_level
  from gulfstream_dw.dim_driver_common_s
  where concat(year,month,day) = '20180514'
  and   driver_id>0
  group by driver_id
) b
on a.driver_id=b.driver_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

优化后：

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test4;
create table tmp_youhua_test4 as
select
      a.order_id
      ,car_level
from
(select order_id
        ,driver_id
from  gulfstream_ods.g_order
where concat(year,month,day)='20180514'
)a
left outer join
(
  select
        driver_id
        ,max(driver_car_level) as car_level
  from gulfstream_dw.dim_driver_common_s
  where concat(year,month,day) = '20180514'
  and   driver_id>0
  group by driver_id
) b
on (if(a.driver_id=0,cast(rand()*-100 as bigint),a.driver_id)=b.driver_id)
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

不适用场景：

- 上述大表a在left outer join的右侧，这种场景推荐使用1.2.4

1.2.6 分成两段程序处理后union all

作用：

- 把相同key记录数比较多的和相同key记录数比较少的记录分开来单独处理，然后再把结果union all 起来

适用场景：

- 大表关联大表，且其它方式无法解决数据倾斜的问题的时候

缺陷：

- 正常情况下需要额外增加一个程序来计算key对应的数据量，比如说把记录数超过10w的那些key先存到一个临时表。

1.2.7 join中的分区裁剪失效

是发生在left outer
join中，内关联不会的。本质是sql子句的执行顺序问题，内关联where和on是等效的，先筛选再关联，左关联时是先on中条件筛选，再关联，最后再where条件过滤，所以放在where里肯定会失效。

1.3 Reduce 阶段的数据倾斜

Reduce阶段的数据倾斜一般是由以下几种原因引起的：

- Group by后面的字段值存在数据倾斜
- Distinct后面的字段加上group by后面的字段（如果有group by的话，没有就是distinct后面的字段）存在数据倾斜
- 动态分区导致数据倾斜
- 窗口函数中partition by后面的字段存在数据倾斜

1.3.1 Group by后面的字段值存在数据倾斜

设置 set hive.groupby.skewindata=true

作用：

- 在group by之前增加一步distribute，把相同key的数据打散到多个instance上处理。默认hive.groupby.skewindata为false。

适用场景：

- Group by后面的字段值存在数据倾斜
- 针对单个distinct比较有效，多个没效果

1.3.2 用打标+聚合的方式代替distinct的聚合

作用：

- 尽量减少程序中的distinct，其实去重运算是很慢的，尤其是多个count(distinct)的时候，你会发现为什么源表的数据量经过map后怎么变大了，有多少个count(distinct)就会变大几倍，导致处理很慢。所幸对于去重主体相同的情况下是可以将多个distinct改写为一个的，方法是先在子查询中通过distinct或group by去重一次，将限定条件打上标签，最后按照各类标签直接count即可。

适用场景：

- 同一个程序中有多个distinct（比如3个以上）

具体案例：

优化前：运行时长30分钟

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test7;
create TEMPORARY table tmp_youhua_test7 as
select
      driver_id
      ,count(distinct case when is_td_finish=1 then passenger_id else null end) as finish_pas_num
      ,count(distinct case when is_grab_before_cannel=1 then passenger_id else null end) as
before_cannel_pas_num
      ,count(distinct case when is_grab_after_pas_cancel=1 then passenger_id else null end) as
after_cannel_pas_num
      ,count(distinct case when is_anycar=1 then passenger_id else null end) as anycar_pas_num
      ,count(distinct case when like_wait>=1 then passenger_id else null end) as like_wait_pas_num
from gulfstream_dwd.dwd_order_make_d
where concat(year,month)='201804'
group by driver_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

优化后：运行时长10分钟

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test8;
create TEMPORARY table tmp_youhua_test8 as
select
      driver_id
      ,sum(is_td_finish) is_td_finish
      ,sum(is_grab_before_cannel) is_grab_before_cannel
      ,sum(is_grab_after_pas_cancel) is_grab_after_pas_cancel
      ,sum(is_anycar) is_anycar
      ,sum(like_wait) like_wait
from
(select
      driver_id
      ,passenger_id
      ,max(is_td_finish) is_td_finish
      ,max(is_grab_before_cannel) is_grab_before_cannel
      ,max(is_grab_after_pas_cancel) is_grab_after_pas_cancel
      ,max(is_anycar) is_anycar
      ,max(like_wait) like_wait
from gulfstream_dwd.dwd_order_make_d
where concat(year,month)='201804'
group by driver_id,passenger_id
)a
group by driver_id
;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

1.3.3 利用窗口函数

利用窗口函数可以减少1次join操作，从而提高计算性能

具体案例：

1、取最大值/最小值，例如订单统计表dm_fast_order_make_d中有每个城市的完成订单数，需要计算完单最多的那个城市。

比较繁琐的写法：

```
select
      fast_didicity
from
(select
      max(finish_order_cnt) finish_order_cnt
 from
    (select
          fast_didicity
          ,sum(finish_order_cnt) finish_order_cnt
     from  gulfstream_dm.dm_fast_order_make_d
     where concat(year,month,day)='20180520'
     group by fast_didicity
    )a
)b
join
(select
          fast_didicity
          ,sum(finish_order_cnt) finish_order_cnt
     from  gulfstream_dm.dm_fast_order_make_d
     where concat(year,month,day)='20180520'
     group by fast_didicity
)c
on  b.finish_order_cnt=c.finish_order_cnt
```

优化的写法：

```
select
      fast_didicity
from
(select
      fast_didicity
      ,row_number()over(partition by 1 order by finish_order_cnt desc) as rk
from
    (select
          fast_didicity
          ,sum(finish_order_cnt) finish_order_cnt
     from  gulfstream_dm.dm_fast_order_make_d
     where concat(year,month,day)='20180520'
     group by fast_didicity
    )a
)b
where rk=1
```

2、需要同时提取明细数据和汇总数据，可以采用窗口函数。

1.3.4 用max函数替代row_number排序取第一

作用：

- 避免窗口函数中partition by导致数据倾斜，同时可以减少一步reduce任务

适用场景：

- 程序中通过row_number函数按照日期或者某个值排序，取最大的那条记录的某个值。

具体案例：

获取一个拼车行程中，第一单开始计费的订单所在的城市作为行程的城市。

```
HQL_STR="
use gulfstream_test;
drop table tmp_youhua_test9;
create TEMPORARY table tmp_youhua_test9 as
select
      travel_id
      ,split(str_city_id,',')[1] as city_id
      ,begin_charge_time
      ,finish_time
      ,normal_distance
from
(select
      travel_id
      ,min(concat(begin_charge_time,',',city_id)) as str_city_id
      ,min(begin_charge_time) as begin_charge_time
      ,max(finish_time)       as finish_time
      ,sum(normal_distance) normal_distance
from gulfstream_dwd.dwd_order_make_d
where concat(year,month,day) = '20170520'
and product_category in (3,7,99,20)
and gesake_order_type=1
and is_td_finish=1
group by travel_id
)a ;"
echo "$HQL_STR"
hive -e "$HQL_STR"
```

1.4 增加中间表减少任务数据处理量

作用：

- 把多个程序中类似的处理逻辑提炼到同一个任务中处理，生产中间表供后续任务共用

适用场景：

- 后续有2个及以上任务可以从这个中间表计算；
- 中间表可以有效地减少后续任务的数据处理量；

1.5 增加分区减少后续任务数据处理量

作用：

- 通过把原表的分区粒度设计的更细，以减少后续任务的处理数据量

适用场景：

- 原表数据量比较大，比如几亿以上；
- 后续使用数据的时候，很多任务条件中都包含某一个字段；

这个字段可枚举，数值不宜太多，多了表数据不能保留太久。使用这个字段作为条件能有效减少记录数，如果这个字段记录数集中在某一个值上，且后续都是用这个值过滤，那就不太适合；

2 计算步骤优化

计算步骤优化主要是为了减少单个sql中的task的数量。

2.1 多表join尽量保持join key一致

作用：

- 同一个sql中相同关联key的表join时会放在一个join任务中完成，数据类型也必须保持一致。

具体案例：

```
create table a(id bigint,name string);
create table b(id bigint,name string);
create table c(id bigint,name string);
```

关联的key相同，字段类型也一致，只有一个join任务：

```
select
      a.name as a_name
      ,b.name as b_name
      ,c.name as c_name
from gulfstream_test.a
join gulfstream_test.b
on a.id=b.id
join gulfstream_test.c
on b.id=c.id;
```

```
Starting Job = job_1525428476097_4494643, Tracking URL = http://bigdata-nmg-hdprm01.nmg01:8088/proxy/application_1525428476097_4494643/
Kill Command = /usr/local/hadoop-2.7.2-1107/bin/hadoop job  -kill job_1525428476097_4494643
Hadoop job information for Stage-1: number of mappers: 3; number of reducers: 1
2018-05-21 20:01:57,511 Stage-1 map = 0%,  reduce = 0%
2018-05-21 20:02:08,128 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 8.68 sec
2018-05-21 20:02:19,468 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 12.31 sec
MapReduce Total cumulative CPU time: 12 seconds 310 msec
Ended Job = job_1525428476097_4494643
MapReduce Jobs Launched:
Stage-Stage-1: Map: 3  Reduce: 1   Cumulative CPU: 12.31 sec   HDFS Read: 24223 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 12 seconds 310 msec
```

关联的key的字段类型不一致，有2个join任务：

```
select
      a.name as a_name
      ,b.name as b_name
      ,c.name as c_name
from gulfstream_test.a
join gulfstream_test.b
on cast(a.id as string)=cast(b.id as string)
join gulfstream_test.c
on b.id=c.id;
```

```
Starting Job = job_1525428476097_4495190, Tracking URL = http://bigdata-nmg-hdprm01.nmg01:8088/proxy/application_1525428476097_4495190/
Kill Command = /usr/local/hadoop-2.7.2-1107/bin/hadoop job  -kill job_1525428476097_4495190
Hadoop job information for Stage-2: number of mappers: 2; number of reducers: 1
2018-05-21 20:04:53,954 Stage-2 map = 0%,  reduce = 0%
2018-05-21 20:05:08,357 Stage-2 map = 50%,  reduce = 0%, Cumulative CPU 2.78 sec
2018-05-21 20:05:09,392 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 5.77 sec
2018-05-21 20:05:21,765 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 9.52 sec
MapReduce Total cumulative CPU time: 9 seconds 520 msec
Ended Job = job_1525428476097_4495190
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 8.47 sec   HDFS Read: 14833 HDFS Write: 135 SUCCESS
Stage-Stage-2: Map: 2  Reduce: 1   Cumulative CPU: 9.52 sec   HDFS Read: 12926 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 17 seconds 990 msec
OK
```

2.2 多路插入

作用：一次完成同一份数据的多种不同处理。

具体案例：

订单表里面有订单id、司机id、乘客id，需要统计司机的订单数和乘客的订单数。

不使用多路插入，需要读取原表两次。

```
use gulfstream_test;
create  table tmp1 as
select driver_id
       ,count(1) as cnt
from   gulfstream_dwd.dwd_order_make_d
where  concat(year,month,day)='20180520'
and    is_td_finish=1
group by driver_id;

create table tmp2 as
select passenger_id
       ,count(1) as cnt
from gulfstream_dwd.dwd_order_make_d
group by passenger_id;
```

多路插入，只读取一次原表。

```
use gulfstream_test;
from
    (
        select driver_id,passenger_id,order_id
        from gulfstream_dwd.dwd_order_make_d
        where  concat(year,month,day)='20180520'
        and    is_td_finish=1
    ) a
    insert overwrite table tmp1
    select driver_id,count(1) as cnt
    group by driver_id
    insert overwrite table tmp2
    select passenger_id,count(1) as cnt
    group by passenger_id;
```

2.3 避免暴力扫描

可以采用历史全量+当天增量的方式进行避免。

3 计算资源优化

计算资源主要分为cpu和内存两种，按照map-reduce不同阶段又可以分为mapper和reduce两种类型。目前调整计算资源主要是调整实例个数，内存调整的比较少。

影响mapper的instance个数的因素主要有2个：

- 读入数据的文件总个数
- 读入数据的单个文件大小

Reduce又可以分为join和reduce两种，join是多表join时的reduce个数，reduce是计算的时候的reduce个数。

3.1 增加Mapper的instance个数

目标： 减少mapper阶段执行时间。

适用场景：

- 单个mapper的instance处理的数据量很大，比如超过1亿条；
- mapper总的instance个数比较少，比如小于100，且mapper时间相对比较长，比如超过10分钟
- mapper总的instance个数比较少，比如小于100，且mapper阶段存在一定的数据倾斜.

不适用场景：

- mapper整个任务已经很快，比如3分钟以内
- Mapper单个instance已经不算慢，比如1分钟以内
- Mapper总的instance数已经很大，比如5000以上

可以参考设置如下参数：

set mapred.max.split.size=64000000;（默认256m） 一个split最大的大小

set mapred.min.split.size.per.node=64000000;//（默认256m）一个节点上一个split的至少的大小

set mapred.min.split.size.per.rack=64000000;//（默认256m）一个交换机下一个split至少的大小

3.2 减少Mapper的instance个数

目标：减少mapper阶段执行时间，或者在对执行时间不产生较大影响的前提下减少消耗的
instance数，因为instance的创建和死亡本身也需要时间。

适用场景：

- Mapper阶段执行时间很短，比如1分钟以内
- Mapper的instance数非常大，比如5000以上，且单个instance执行时间比较短，比如30s以内

可以参考设置如下参数：

set mapred.max.split.size=512000000;（默认 256000000） 一个split最大的大小

set mapred.min.split.size.per.node=512000000;//（默认 256000000）一个节点上一个split的至少的大小

set mapred.min.split.size.per.rack=512000000;//（默认 256000000）一个交换机下一个split至少的大小

3.3 增加reduce的instance个数

目的：减少任务的执行时间

适用场景：

- reduce的instance数比较少，比如200以内，且时间比较长，比如10分钟以上
- Join前数据量不是很大，join后数据量很大，导致默认的join的instance数太少

可以参考设置如下参数：

- 参数1：hive.exec.reducers.bytes.per.reducer（每个reduce任务处理的数据量，默认为256m）
- 参数2：hive.exec.reducers.max（每个任务最大的reduce数，默认为999）
- 计算reducer数的公式=min(参数2，总输入数据量/参数1) ，如果reduce的输入（map的输出）总大小不超过256m,
  那么只会有一个reduce任务；

增加reduce的instance个数，可以调整hive.exec.reducers.bytes.per.reducer参数的值；
例如：set hive.exec.reducers.bytes.per.reducer=64000000; （64M）