HIVE调优规范文档

一、合理控制map/reduce数

1.1 map阶段

1、通常情况下，作业会通过input的目录产生一个或者多个map任务。

主要的决定因素有： input的文件总个数，input的文件大小，集群设置的文件块大小(目前为128M, 可在hive中通过set dfs.block.size;命令查看到，该参数不能自定义修改)

```
mapred.min.split.size: 指的是数据的最小分割单元大小；min的默认值是1B
mapred.max.split.size: 指的是数据的最大分割单元大小；max的默认值是256MB
通过调整max可以起到调整map数的作用，减小max可以增加map数，增大max可以减少map数。
需要提醒的是，直接调整mapred.map.tasks这个参数是没有效果的。
```

2、举例：

```
a) 假设input目录下有1个文件a，大小为780M，那么hadoop会将该文件a分隔成7个块（6个128M的块和1个12M的块），从而产生7个map数；

b) 假设input目录下有3个文件a,b,c，大小分别为10M，20M，130M，那么hadoop会分隔成4个块（10M，20M，128M，2M），从而产生4个map数；

注意：如果文件大于块大小（128M），那么会拆分，如果小于块大小，则把该文件当成一个块。
```

3、map数是不是越多越好?

```
其实这就涉及到小文件的问题：如果一个任务有很多小文件（远远小于块大小128M），则每个小文件也会当做一个块，用一个map任务来完成。
而一个map任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的map数是受限的。
```

4、是不是保证每个map处理接近128M的文件块，就高枕无忧了?

```
答案也是不一定。比如有一个127M的文件，正常会用一个map去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果map处理的逻辑比较复杂，用一个map任务去做，肯定也比较耗时。
```

针对上面的问题3和4，我们需要采取两种方式来解决：即减少map数和增加map数；

1.1.1 减少map数

假设一个SQL任务：

```
Select count(1) from popt_tbaccountcopy_meswhere pt = '2012-07-04'
;
```

该任务的inputdir ：

```
/group/p_sdo_data/p_sdo_data_etl/pt/popt_tbaccountcopy_mes/pt=2012
-07-04
```

共有194个文件，其中很多是远远小于128m的小文件，总大小9G，正常执行会用194个map任务。
Map总共消耗的计算资源： SLOTS_MILLIS_MAPS= 623,020

我通过以下方法来在map执行前合并小文件，减少map数：

```
set mapred.max.split.size=100000000;
set mapred.min.split.size.per.node=100000000;
set mapred.min.split.size.per.rack=100000000;
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInpu
tFormat;
```

再执行上面的语句，用了74个map任务，map消耗的计算资源：SLOTS_MILLIS_MAPS= 333,500
对于这个简单SQL任务，执行时间上可能差不多，但节省了一半的计算资源。

大概解释一下

```
100000000表示100M,
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInpu
tFormat;
这个参数表示执行前进行小文件合并,
前面三个参数确定合并文件块的大小，大于文件块大小128m的，按照128m来分隔，小于128
```

```
m,大于100m的，按照100m来分隔，把那些小于100m的（包括小文件和分隔大文件剩下
的），进行合并,最终生成了74个块。
```

1.1.2 增加map数

当input的文件都很大，任务逻辑复杂，map执行非常慢的时候，可以考虑增加Map数，来使得每个map处理的数据量减少，从而提高任务的执行效率。

假设有这样一个任务：

```
Select data_desc,
count(1),
count(distinct id),
  sum(case when …),
sum(case when ...),
sum(…)
from a group by data_desc;
```

如果表a只有一个文件，大小为120M，但包含几千万的记录，如果用1个map去完成这个任务，肯定是比较耗时的，这种情况下，我们要考虑将这一个文件合理的拆分成多个，这样就可以用多个map任务去完成。

```
set mapred.reduce.tasks=10;
create table a_1 as
select * from a
distribute by rand(123);
```

这样会将a表的记录，随机的分散到包含10个文件的a_1表中，再用a_1代替上面sql中的a表，则会用10个map任务去完成。每个map任务处理大于12M（几百万记录）的数据，效率肯定会好很多。

注：

```
看上去，貌似这两种有些矛盾，一个是要合并小文件，一个是要把大文件拆成小文件，这点正
是重点需要关注的地方，使单个map任务处理合适的数据量；
```

1.2 reduce阶段

```
Reduce的个数对整个作业的运行性能有很大影响。
如果Reduce设置过大，那么将会产生很多小文件，对NameNode会产生一定的影响，而且整个作业的运行时间未必会少；如果Reduce设置过小，那么单个Reduce处理的数据将会加大，很可能会引起OOM异常。

如果设置了mapred.reduce.tasks/mapreduce.job.reduces参数，Hive会直接使用它的值作为Reduce的个数；
如果mapred.reduce.tasks/mapreduce.job.reduces的值没有设置（也就是-1），那么Hive会根据输入文件的大小估算出Reduce的个数。
根据输入文件估算Reduce的个数可能未必很准确，因为Reduce的输入是Map的输出，而Map的输出可能会比输入要小，所以最准确的数根据Map的输出估算Reduce的个数。
```

1.2.1 hive如何确定reduce数

reduce个数的设定极大影响任务执行效率，不指定reduce个数的情况下，Hive会猜测确定一个reduce个数，基于以下两个设定：

```
hive.exec.reducers.bytes.per.reducer（每个reduce任务处理的数据量，默认为1000^3=1G）
hive.exec.reducers.max（每个任务最大的reduce数，默认为999）

在只配了hive.exec.reducers.bytes.per.reducer以及hive.exec.reducers.max的情况下，实际的reduce个数会根据实际的数据总量/每个reduce处理的数据量来决定

计算reducer数的公式很简单N=min（参数2，总输入数据量/参数1）
```

即，如果reduce的输入（map的输出）总大小不超过1G，那么只会有一个reduce任务；

```
如：select pt,count(1) from popt_tbaccountcopy_mes where pt = '2012-07-04' group by pt;
/group/p_sdo_data/p_sdo_data_etl/pt/popt_tbaccountcopy_mes/pt=2012-07-04 总大小为9G多，
因此这句有10个reduce
```

1.2.2 调整reduce个数方法一

调整hive.exec.reducers.bytes.per.reducer参数的值；

```
set hive.exec.reducers.bytes.per.reducer=500000000;  (500M)
select pt, count(1) from popt_tbaccountcopy_mes where pt = '2012-07-04' group by pt;
这次有20个reduce
```

1.2.3 调整reduce个数方法二

调整mapred.reduce.tasks参数的值；

```
实际运行的reduce个数，默认是-1，可以认为指定，但是如果认为在此指定了，那么就不会通过实际的总数据量/hive.exec.reducers.bytes.per.reducer来决定reduce的个数了

set mapred.reduce.tasks=15;
select pt,count(1) from popt_tbaccountcopy_mes where pt = '2012-07-04' group by pt;
这次有15个reduce
```

1.2.4 reduce数并不是越多越好

```
同map一样，启动和初始化reduce也会消耗时间和资源；
另外，有多少个reduce，就会有个多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；
```

1.2.5 什么情况下只有一个reduce

很多时候你会发现任务中不管数据量多大，不管你有没有调整reduce个数的参数，任务中一直都只有一个reduce任务；其实只有一个reduce任务的情况，除了数据量小于hive.exec.reducers.bytes.per.reducer

参数值的情况外，还有以下原因：

```
1 （1）没有group by的汇总，
2 比如把select pt,count(1) from popt_tbaccountcopy_mes where pt = ‘2012-07-04’ group by pt;  写成select count(1) from popt_tbaccountcopy_mes where pt = ‘2012-07-04’;
3 这点非常常见，希望大家尽量改写。
4
5 （2）用了Order by
6
7 （3）有笛卡尔积。
```

注意：

```
1 在设置reduce个数时也需要考虑这两个原则：使大数据量利用合适的reduce数；是单个reduce任务处理合适的数据量；
```

二、优化sql

2.1 列裁剪

Hive在读数据的时候，可以只读取查询中所需要用到的列，而忽略其他列。例如，若有以下查询：

```
1 SELECT a,b FROM q WHERE e<10;
```

在实施此项查询中，Q表有5列（a，b，c，d，e），Hive只读取查询逻辑中真实需要的3列a、b、e， 而忽略列c，d；这样做节省了读取开销，中间表存储开销和数据整合开销。

```
1 裁剪对应的参数项为：hive.optimize.cp=true（默认值为true）
```

2.2 分区裁剪

可以在查询的过程中减少不必要的分区。例如，若有以下查询：

```
SELECT * FROM (SELECT a1, COUNT(1) FROM T GROUP BY a1) subq WHERE subq.prtn=100; #（多余分区）
SELECT * FROM T1 JOIN (SELECT * FROM T2) subq ON (T1.a1=subq.a2) WHERE subq.prtn=100;
```

查询语句若将"subq.prtn=100"条件放入子查询中更为高效，可减少读入的分区数。Hive自动执行这种裁剪优化。

```
分区参数为：hive.optimize.pruner=true（默认值为true）
```

2.3 where条件优化

优化前（关系数据库不用考虑会自动优化）：

```
select m.cid,u.id from order m join customer u on( m.cid =u.id )where m.dt='20180808';
```

优化后（where条件在map端执行而不是在reduce端执行）：

```
select m.cid,u.id from  (select * from order where dt='20180818') m join customer u on( m.cid =u.id);
```

2.4 union优化

尽量不要使用union （union 去掉重复的记录），而是使用 union all， 然后在用group by 去重

2.5 count(distinct)优化

在数据比较倾斜的时候COUNT(DISTINCT)会比较慢。这时可以尝试用GROUP BY改写代码

数据量小的时候无所谓，数据量大的情况下，由于COUNT DISTINCT操作需要用一个Reduce Task来完成，这一个Reduce需要处理的数据量太大，就会导致整个Job很难完成

```
select count(1) from (select id from tablename group by id) tmp;
```

2.6 mapjoin操作

如果不指定MapJoin或者不符合MapJoin的条件，那么Hive解析器会将Join操作转换成Common Join，即：在Reduce阶段完成join，容易发生数据倾斜，可以用MapJoin把小表全部加载到内存在map端进行join，避免reducer处理。

```
开启MapJoin参数设置:
1）设置自动选择MapJoin
set hive.auto.convert.join = true;默认为true
2）大表小表的阀值设置（默认25M一下认为是小表）：
set hive.mapjoin.smalltable.filesize=25000000;
```

注：新版hive默认是开启的

2.7 优化in/exists语句

虽然hive1.2之后也支持in/exists操作，但还是推荐使用一个高效的替代方案：left semi join

```
比如说:
select a.id, a.name from a where a.id in (select b.id from b);
select a.id, a.name from a where exists (select id from b where a.
id = b.id);
应该转换成:
select a.id, a.name from a left semi join b (on a.id = b.id);
```

2.8 排序选择

- **cluster by**: 对同一字段分桶并排序，不能和sort by连用；

- **distribute by + sort by**: 分桶，保证同一字段值只存在一个结果文件当中，结合sort by 保证每个reduceTask结果有序；
- **sort by**: 单机排序，单个reduce结果有序
- **order by**：全局排序，缺陷是只能使用一个reduce

三、小文件相关

3.1 小文件如何产生的

```
（1）动态分区插入数据，产生大量的小文件，从而导致map数量剧增；

（2）reduce数量越多，小文件也越多（reduce的个数和输出文件是对应的）；

（3）数据源本身就包含大量的小文件。
```

3.2 小文件过多的影响

```
（1）从Hive的角度看，小文件会开很多map，一个map开一个JVM去执行，所以这些任务的初始化，启动，执行会浪费大量的资源，严重影响性能。

（2）在HDFS中，每个小文件对象约占150byte，如果小文件过多会占用大量内存。这样NameNode内存容量严重制约了集群的扩展。
```

3.3 小文件解决方案

（1）在Map执行前合并小文件，减少Map数：CombineHiveInputFormat具有对小文件进行合并的功能（系统默 认的格式）。HiveInputFormat没有对小文件合并功能。

（2）少用动态分区，用时记得按distribute by分区

（3）**merge -- 参数调节输出合并小文件**

```
--在map端输出进行合并，默认为true
```

```
SET hive.merge.mapfiles = true;
--在map-reduce任务结束时合并小文件,默认false
SET hive.merge.mapredfiles = true;
--合并文件的大小,默认256M
SET hive.merge.size.per.task = 268435456;
--当输出文件的平均大小小于该值时，启动一个独立的map-reduce任务进行文件merge,
默认16M
SET hive.merge.smallfiles.avgsize = 16777216;
```

（4）开启JVM重用

Hadoop通常是使用派生JVM来执行map和reduce任务的。这时JVM的启动过程可能会造成相当大的开销，尤其是执行的job包含成百上千的task任务的情况。JVM重用可以使得JVM示例在同一个job中时候

```
set mapreduce.job.jvm.numtasks=10
```

四、数据倾斜

4.1 数据倾斜的具体表现

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。

4.2 数据倾斜的原因

某个reduce的数据输入量远远大于其他reduce数据的输入量

```
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
```

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
join	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，非常慢
group by	group by 维度过小，某值的数量过多	处理某值的reduce非常耗时
count distinct	某特殊值过多	处理此特殊值reduce耗时

4.3 常见的数据倾斜处理方案

4.3.1 参数调节

1. 设置hive.map.aggr=true

```
开启map端部分聚合功能，相当于Combiner，就是将key相同的归到一起，减少数据量，这样就可以相对地减少进入reduce的数据量，在一定程度上可以提高性能，当然，如果数据的减少量微乎其微，那对性能的影响几乎没啥变化。
```

2. 设置hive.groupby.skewindata=true
   如果发生了数据倾斜就可以通过它来进行负载均衡。

```
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。

点评：它使计算变成了两个mapreduce，先在第一个中在shuffle过程partition时随机给 key打标记，使每个key随机均匀分布到各个reduce上计算，但是这样只能完成部分计算，因为相同key没有分配到相同reduce上。
所以需要第二次的mapreduce，这次就回归正常shuffle，但是数据分布不均匀的问题在第一次mapreduce已经有了很大的改善，因此基本解决数据倾斜。因为大量计算已经在第一次mr中随机分布到各个节点完成。
```

4.3.2 sql语句优化

1. **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
```

2. **大小表Join**

   使用map join让小的维度表先进内存。在map端完成reduce

```
不使用普通join的原因：数据进入reduce，肯定要先进行分区，而分区就是根据key的hash值来进行的，既然数据的key本身就是不均匀的了（即某些key很集中，或者干脆就是有很多相同的key，比如key为无效值null），那这样分区的结果就是这些集中的key会被分到同一个分区中，而这些key的数量本就大，所以会产生数据倾斜。

使用map join的话，直接在map端就完成表的join操作（表的join得到的结果（一张新表）就是我们这次的目标了），进入map端的数据都是经过split（分片）得到的，没有根据key分区这一操作，所以数据都是相对均匀地分布在每个map task中的，所以就不会产生数据倾。
```

3. **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
```

4. **count(distinct xxx)大量相同特殊值**

```
方法1：
可以先找出这个特殊值是什么，过滤掉它，然后通过count(distinct)计算出剩余的个数，最后再加1（这1个就是那个被过滤掉的特殊值）

方法2：
我们可以先通过group by分组，然后再在分组得到的结果的基础之上进行count
```

4.4 典型业务场景

4.4.1 不同数据类型关联产生数据倾斜

情形：比如用户表中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。

后果：处理此特殊值的reduce耗时；只有一个reduce任务。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。

解决方式：**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)
```

4.4.2 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

```
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。

```
1 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。
```

五、总结

优化时，把hive sql当做mapreduce程序来读，会有意想不到的惊喜。理解hadoop的核心能力，是hive优化的根本。

- **hadoop处理数据的过程，有几个显著的特征**:

1. 不怕数据多，就怕数据倾斜。
2. 对jobs数比较多的作业运行效率相对比较低，比如即使有几百行的表，如果多次关联多次汇总，产生十几个jobs，没半小时是跑不完的。map reduce作业初始化的时间是比较长的。
3. 对sum，count来说，不存在数据倾斜问题。
4. 对count(distinct),效率较低，数据量一多，准出问题，如果是多count(distinct)效率更低。

- **优化可以从几个方面着手：**

1. 好的模型设计事半功倍。
2. 解决数据倾斜问题。
3. 减少job数。
4. 设置合理的map reduce的task数，能有效提升性能。
5. 自己动手写sql解决数据倾斜问题是个不错的选择。set hive.groupby.skewindata=true;这是通用的算法优化，但算法优化总是漠视业务，习惯性提供通用的解决方法。 Etl开发人员更了解业务，更了解数据，所以通过业务逻辑解决倾斜的方法往往更精确，更有效。
6. 对count(distinct)采取漠视的方法，尤其数据大的时候很容易产生倾斜问题，不抱侥幸心理。
7. 对小文件进行合并，是行至有效的提高调度效率的方法，假如我们的作业设置合理的文件数，对云梯的整体调度效率也会产生积极的影响。
8. 优化时把握整体，单个作业最优不如整体最优??