****Hive优化****

**要点：优化时，把hive sql当做map reduce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。**

长期观察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。(比如，10w+级别的计算，用160个reduce，那是相当的浪费，1个足够)。

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。自己动手，丰衣足食。

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。

## **优化案例：**

**问题1：如日志中，常会有信息丢失的问题，比如全网日志中的user\_id，如果取其中的user\_id和bmw\_users关联，就会碰到数据倾斜的问题。**

方法：解决数据倾斜问题

解决方法1. User\_id为空的不参与关联，例如：

Select \*

From log a

Join  bmw\_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 ：

Select \*

from log a

left outer join bmw\_users b

**on case when a.user\_id is null then concat(‘dp\_hive’,rand() ) else a.user\_id end = b.user\_id;**

**总结：**2比1效率更好，不但io少了，而且作业数也少了。1方法log读取两次，jobs是2。2方法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。因为空值不参与关联，即使分到不同的reduce上，也不影响最终的结果。附上hadoop通用关联的实现方法（关联通过二次排序实现的，关联的列为parition key,关联的列c1和表的tag组成排序的group key,根据parition key分配reduce。同一reduce内根据group key排序）。

**问题2：不同数据类型id的关联会产生数据倾斜问题。**

一张表s8的日志，每个商品一条记录，要和商品表关联。但关联却碰到倾斜的问题。s8的日志中有字符串商品id,也有数字的商品id,类型是string的，但商品中的数字id是bigint的。猜测问题的原因是把s8的商品id转成数字id做hash来分配reduce，所以字符串id的s8日志，都到一个reduce上了，解决的方法验证了这个猜测。

**方法：把数字类型转换成字符串类型**

Select \* from s8\_log a

Left outer join r\_auction\_auctions b

**On a.auction\_id = cast(b.auction\_id as string);**

**问题3：利用hive 对UNION ALL的优化的特性**

**hive对union all优化只局限于非嵌套查询。**

比如以下的例子：

select \* from

(select \* from t1

 Group by c1,c2,c3

Union all

Select \* from t2

Group by c1,c2,c3) t3

   Group by c1,c2,c3;

从业务逻辑上说，子查询内的group by 怎么都看显得多余（功能上的多余,除非有count(distinct)），如果不是因为hive bug或者性能上的考量(曾经出现如果不子查询group by ，数据得不到正确的结果的hive bug)。所以这个hive按经验转换成

select \* from

(select \* from t1

Union all

Select \* from t2

) t3

   Group by c1,c2,c3;

经过测试，并未出现union all的hive bug,数据是一致的。mr的作业数有3减少到1。

t1相当于一个目录，t2相当于一个目录，那么对map reduce程序来说，t1,t2可以做为map reduce 作业的mutli inputs。那么，这可以通过一个map reduce 来解决这个问题。Hadoop的计算框架，不怕数据多，就怕作业数多。

但如果换成是其他计算平台如oracle，那就不一定了，因为把大的输入拆成两个输入，分别排序汇总后merge(假如两个子排序是并行的话)，是有可能性能更优的（比如希尔排序比冒泡排序的性能更优）。

**问题4：比如推广效果表要和商品表关联，效果表中的auction id列既有商品id,也有数字id,和商品表关联得到商品的信息**。那么以下的hive sql性能会比较好

Select \* from effect a

Join (select auction\_id as auction\_id from auctions

Union all

Select auction\_string\_id as auction\_id from auctions

) b

On a.auction\_id = b.auction\_id。

比分别过滤数字id,字符串id然后分别和商品表关联性能要好。

这样写的好处,1个MR作业,商品表只读取一次，推广效果表只读取一次。把这个sql换成MR代码的话，map的时候，把a表的记录打上标签a,商品表记录每读取一条，打上标签b，变成两个<key ,value>对，<b,数字id>，<b,字符串id>。所以商品表的hdfs读只会是一次。

**问题5：先join生成临时表，在union all还是写嵌套查询，这是个问题**。比如以下例子：

Select \*

From (select \*

     From t1

     Uion all

     select \*

     From t4

     Union all

     Select \*

     From t2

     Join t3

     On t2.id = t3.id

     ) x

Group by c1,c2;

这个会有4个jobs。假如先join生成临时表的话t5,然后union all，会变成2个jobs。

Insert overwrite table t5

Select \*

     From t2

     Join t3

     On t2.id = t3.id

;

Select \* from (t1 union all t4 union all t5) ;

**hive在union all优化上可以做得更智能（把子查询当做临时表），这样可以减少开发人员的负担。出现这个问题的原因应该是union all目前的优化只局限于非嵌套查询。如果写MR程序这一点也不是问题，就是multi inputs。**

**问题6：使用map join解决数据倾斜的常景下小表关联大表的问题，但如果小表很大，怎么解决**。这个使用的频率非常高，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 members b

On a.memberid = b.memberid.

Members有600w+的记录，把members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 memberid from log ) c

Join members d

On c.memberid = d.memberid

)x

On a.memberid = b.memberid。

先根据log取所有的memberid，然后mapjoin 关联members取今天有日志的members的信息，然后在和log做mapjoin。

假如，log里memberid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。

**问题7：HIVE下通用的数据倾斜解决方法,double被关联的相对较小的表，这个方法在mr的程序里常用。**还是刚才的那个问题：

Select  \* from log a

Left outer join (select  /\*+mapjoin(e)\*/

memberid, number

             From members d

             Join num e

             ) b

On a.memberid=  b.memberid

And mod(a.pvtime,30)+1=b.number。

Num表只有一列number，有30行，是1,30的自然数序列。就是把member表膨胀成30份，然后把log数据根据memberid和pvtime分到不同的reduce里去，这样可以保证每个reduce分配到的数据可以相对均匀。就目前测试来看，使用mapjoin的方案性能稍好。后面的方案适合在map join无法解决问题的情况下。

**如下的优化方案可以做成通用的hive优化方法**

**1. 采样log表，哪些memberid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。Stage1**

**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members做map join生成tmp2,把tmp2读到distribute file cache。这是一个map过程。Stage2**

**3.    map读入members和log，假如记录来自log,则检查memberid是否在tmp2里，如果是，输出到本地文件a,否则生成<memberid,value>的key,value对，假如记录来自member,生成<memberid,value>的key,value对，进入reduce阶段。Stage3.**

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

**这个方法在hadoop里应该是能实现的。Stage2是一个map过程，可以和stage3的map过程可以合并成一个map过程。**

**这个方案目标就是：倾斜的数据用mapjoin,不倾斜的数据用普通的join，最终合并得到完整的结果。用hive sql写的话，sql会变得很多段，而且log表会有多次读。倾斜的key始终是很少的，这个在绝大部分的业务背景下适用。那是否可以作为hive针对数据倾斜join时候的通用算法呢？**

**问题8：多粒度(平级的)uv的计算优化**，比如要计算店铺的uv。还有要计算页面的uv,pvip.

**方案1:**

Select shopid,count(distinct uid)

From log group by shopid;

Select pageid, count(distinct uid),

From log group by pageid;

由于存在数据倾斜问题，这个结果的运行时间是非常长的。

**方案二：**

From log

Insert overwrite table t1 (type=’1’)

Select shopid

Group by shopid ,acookie

Insert overwrite table t1 (type=’2’)

Group by pageid,acookie;

店铺uv:

Select shopid,sum(1)

From t1

Where type =’1’

Group by shopid ;

页面uv:

Select pageid,sum(1)

From t1

Where type =’1’

Group by pageid ;

**这里使用了multi insert的方法，有效减少了hdfs读，但multi insert会增加hdfs写，多一次额外的map阶段的hdfs写。使用这个方法，可以顺利的产出结果。**

**方案三：**

Insert into t1

Select type,type\_name,’’ as uid

From (

Select  ‘page’ as type,

        Pageid as type\_name,

        Uid

From log

Union all

Select  ‘shop’ as type,

       Shopid as type\_name,

       Uid

From log ) y

Group by type,type\_name,uid;

Insert into t2

Select type,type\_name,sum(1)

From t1

Group by type,type\_name;

From t2

Insert into t3

Select type,type\_name,uv

Where type=’page’

Select type,type\_name,uv

Where type=’shop’ ;

最终得到两个结果表t3,页面uv表，t4,店铺结果表。从io上来说，log一次读。但比方案2少次hdfs写（multi insert有时会增加额外的map阶段hdfs写）。作业数减少1个到3，有reduce的作业数由4减少到2，第三步是一个小表的map过程，分下表，计算资源消耗少。但方案2每个都是大规模的去重汇总计算。

**这个优化的主要思路是，map reduce作业初始化话的时间是比较长，既然起来了，让他多干点活**，顺便把页面按uid去重的活也干了，省下log的一次读和作业的初始化时间，省下网络shuffle的io，但增加了本地磁盘读写。效率提升较多。

**这个方案适合平级的不需要逐级向上汇总的多粒度uv计算，粒度越多，节省资源越多，比较通用。**

**问题9：多粒度，逐层向上汇总的uv结算。**比如4个维度，a,b,c,d，分别计算a,b,c,d,uv；

a,b,c,uv;a,b,uv;a;uv,total uv4个结果表。这可以用问题8的方案二，这里由于uv场景的特殊性，多粒度，逐层向上汇总，就可以使用一次排序，所有uv计算受益的计算方法。

**案例：**目前mm\_log日志一天有25亿+的pv数，要从mm日志中计算uv，与ipuv,一共计算

三个粒度的结果表

（memberid,siteid,adzoneid,province,uv,ipuv）  R\_TABLE\_4

（memberid,siteid,adzoneid,uv,ipuv） R\_TABLE\_3

 (memberid,siteid,uv,ipuv) R\_TABLE\_2

**第一步：按memberid,siteid,adzoneid,province,使用group去重**,产生临时表，对cookie,ip

打上标签放一起，一起去重，临时表叫T\_4;

Select memberid,siteid,adzoneid,province,type,user

From(

Select memberid,siteid,adzoneid,province,‘a’ type ,cookie as user from mm\_log where ds=20101205

Union all

Select memberid,siteid,adzoneid,province,‘i’ type ,ip as user from mm\_log where ds=20101205

) x group by memberid,siteid,adzoneid,province,type,user ;

**第二步：排名**,产生表T\_4\_NUM.Hadoop最强大和核心能力就是parition 和 sort.按type，acookie分组，

Type，acookie，memberid,siteid,adzoneid,province排名。

Select \* ,

row\_number(type,user,memberid,siteid,adzoneid ) as adzone\_num ,  
row\_number(type,user,memberid,siteid ) as site\_num,

row\_number(type,user,memberid ) as member\_num,

row\_number(type,user ) as total\_num

from (select  \* from T\_4 distribute by type,user sort by type,user, memberid,siteid,adzoneid ) x;

这样就可以得到不同层次粒度上user的排名，相同的user id在不同的粒度层次上，排名等于1的记录只有1条。取排名等于1的做sum，效果相当于Group by user去重后做sum操作。

第三步：不同粒度uv统计，先从最细粒度的开始统计，产生结果表R\_TABLE\_4,这时，结果集只有10w的级别。

如统计memberid,siteid,adzoneid,provinceid粒度的uv使用的方法就是

Select memberid,siteid,adzoneid, provinceid,

sum(case when  type =’a’ then cast(1) as bigint end ) as province\_uv ,

sum(case when  type =’i’ then cast(1) as bigint end ) as province\_ip ,

sum(case when adzone\_num =1 and type =’a’ then cast(1) as bigint end ) as adzone\_uv ,

sum(case when adzone\_num =1 and type =’i’ then cast(1) as bigint end ) as adzone\_ip ,

sum(case when site\_num =1 and type =’a’ then cast(1) as bigint end ) as site\_uv ,

sum(case when site\_num =1 and type =’i’ then cast(1) as bigint end ) as site\_ip ,

sum(case when member\_num =1 and type =’a’ then cast(1) as bigint end ) as member\_uv ,

sum(case when member\_num =1 and type =’i’ then cast(1) as bigint end ) as member\_ip ,

sum(case when total\_num =1 and type =’a’ then cast(1) as bigint end ) as total\_uv ,

sum(case when total\_num =1 and type =’i’ then cast(1) as bigint end ) as total\_ip ,

from T\_4\_NUM

group by memberid,siteid,adzoneid, provinceid ;

广告位粒度的uv的话，从R\_TABLE\_4统计，这是源表做10w级别的统计

Select memberid,siteid,adzoneid,sum(adzone\_uv),sum(adzone\_ip)

From R\_TABLE\_4

Group by memberid,siteid,adzoneid；

memberid,siteid的uv计算 ，

memberid的uv计算,

total uv 的计算也都从R\_TABLE\_4汇总。

## 一．join优化

Join查找操作的基本原则：应该将条目少的表/子查询放在 Join 操作符的左边。原因是在 Join 操作的 Reduce 阶段，位于 Join 操作符左边的表的内容会被加载进内存，将条目少的表放在左边，可以有效减少发生内存溢出错误的几率。

Join查找操作中如果存在多个join，且所有参与join的表中其参与join的key都相同，则会将所有的join合并到一个mapred程序中。

案例：

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)  在一个mapre程序中执行join

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)   在两个mapred程序中执行join

Map join的关键在于join操作中的某个表的数据量很小，案例：

SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

  FROM a join b on a.key = b.key

Mapjoin 的限制是无法执行a FULL/RIGHT OUTER JOIN b，和map join相关的hive参数：**hive.join.emit.interval  hive.mapjoin.size.key  hive.mapjoin.cache.numrows**

由于join操作是在where操作之前执行，所以当你在执行join时，where条件并不能起到减少join数据的作用；案例：

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

  WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'

最好修改为：

SELECT a.val, b.val FROM a LEFT OUTER JOIN b

  ON (a.key=b.key AND b.ds='2009-07-07' AND a.ds='2009-07-07')

在join操作的每一个mapred程序中，hive都会把出现在join语句中相对靠后的表的数据stream化，相对靠前的变的数据缓存在内存中。当然，也可以手动指定stream化的表：SELECT /\*+ STREAMTABLE(a) \*/ a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

## 二、group by 优化

Map端聚合，首先在map端进行初步聚合，最后在reduce端得出最终结果，相关参数：

**hive.map.aggr = true**是否在 Map 端进行聚合，默认为 True

**hive.groupby.mapaggr.checkinterval = 100000**在 Map 端进行聚合操作的条目数目

数据倾斜聚合优化，设置参数**hive.groupby.skewindata = true，**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。

## 三、合并小文件

文件数目过多，会给 HDFS 带来压力，并且会影响处理效率，可以通过合并 Map 和 Reduce 的结果文件来消除这样的影响：

**hive.merge.mapfiles = true**是否和并 Map 输出文件，默认为 True

**hive.merge.mapredfiles = false**是否合并 Reduce 输出文件，默认为 False

**hive.merge.size.per.task = 256\*1000\*1000**合并文件的大小

## 四、Hive实现(not) in

通过left outer join进行查询,（假设B表中包含另外的一个字段 key1

select a.key from a left outer join b on a.key=b.key where b.key1 is null

通过left semi join 实现 in

SELECT a.key, a.val FROM a LEFT SEMI JOIN b on (a.key = b.key)

Left semi join 的限制：join条件中右边的表只能出现在join条件中。

## 五、排序优化

Order by 实现全局排序，一个reduce实现，效率低

Sort by 实现部分有序，单个reduce输出的结果是有序的，效率高，通常和DISTRIBUTE BY关键字一起使用（DISTRIBUTE BY关键字 可以指定map 到 reduce端的分发key）

CLUSTER BY col1 等价于DISTRIBUTE BY col1 SORT BY col1

## 六、使用分区

Hive中的每个分区都对应hdfs上的一个目录，分区列也不是表中的一个实际的字段，而是一个或者多个伪列，在表的数据文件中实际上并不保存分区列的信息与数据。Partition关键字中排在前面的为主分区（只有一个），后面的为副分区

静态分区：静态分区在加载数据和使用时都需要在sql语句中指定

          案例：(stat\_date='20120625',province='hunan')

动态分区：使用动态分区需要设置hive.exec.dynamic.partition参数值为true，默认值为false，在默认情况下，hive会假设主分区时静态分区，副分区使用动态分区；如果想都使用动态分区，需要设置set hive.exec.dynamic.partition.mode=nostrick，默认为strick

          案例：(stat\_date='20120625',province)

## 七、Distinct使用

Hive支持在group by时对同一列进行多次distinct操作，却不支持在同一个语句中对多个列进行distinct操作。

## 八、Hql使用自定义的mapred脚本

注意事项：在使用自定义的mapred脚本时，关键字MAP REDUCE 是语句SELECT TRANSFORM ( ... )的语法转换，并不意味着使用MAP关键字时会强制产生一个新的map过程，使用REDUCE关键字时会产生一个red过程。

自定义的mapred脚本可以是hql语句完成更为复杂的功能，但是性能比hql语句差了一些，应该尽量避免使用，如有可能，使用UDTF函数来替换自定义的mapred脚本

## 九、UDTF

UDTF将单一输入行转化为多个输出行，并且在使用UDTF时，select语句中不能包含其他的列，UDTF不支持嵌套，也不支持group by 、sort by等语句。如果想避免上述限制，需要使用lateral view语法，案例：

select a.timestamp, get\_json\_object(a.appevents, '$.eventid'), get\_json\_object(a.appenvets, '$.eventname') from log a;

select a.timestamp, b.\*

from log a lateral view json\_tuple(a.appevent, 'eventid', 'eventname') b as f1, f2;

其中，get\_json\_object为UDF函数，json\_tuple为UDTF函数。

UDTF函数在某些应用场景下可以大大提高hql语句的性能，如需要多次解析json或者xml数据的应用场景。