Hive 常见优化

一、数据倾斜

1、什么是数据倾斜？Hadoop 框架的特性决定最怕数据倾斜

•由于数据分布不均匀，造成数据大量的集中到一点，造成数据热点。

节点间数据分布不均衡，会造成 map 端每个 map 任务的工作量不同，即 map 端数据倾斜。Map-reduce，把相同 key 提交给同一个 reduce，如果 key 不均衡就会造成不同的 reduce 的工作量不同。
以京东首页活动为例，曝光率大的是大活动，曝光率小的是小活动：
假如 reduce1 处理的是小活动，reduce2 处理大活动，reduce2 干的活比其他 reduce 多很多，会出现其他 reduce 执行完毕了，reduce2 还在缓慢执行。

症状：map 阶段快，reduce 阶段非常慢；
　　某些 map 很快，某些 map 很慢；
　　某些 reduce 很快，某些 reduce 奇慢。

如下情况：
A、数据在节点上分布不均匀
B、join 时 on 关键词中个别值量很大（如 null 值）
C、count(distinct),在数据量大的情况下，容易数据倾斜，因为 count(distinct)是按 group by 字段分组，按 distinct 字段排序。

其中 A 无法避免。B 见后边的 Join 章节。C 语法上有时无法避免

如何解决数据倾斜？实际上是没办法避免的，这里的解决只是个别情况起效：
　　有数据倾斜的时候进行负载均衡
　　set hive.groupby.skewindata = false;
　　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。

二、 Join 、MapJoin、Group by

Join

按照 join 的 key 进行分发，而在 join **左边**的表的部分数据会首先读入内存，如果左边表的 key 相对分散(或少，分散的意思是相同 key 的数据量小)，join 任务执行会比较快；而如果左边的表 key 比较集中（key 的大小量级分化），而这张表的数据量很大，那么数据倾斜就会比较严重。

Map 阶段同一 Key 数据分发给同一个 reduce。

Join 原则：

小表 Join 大表，原因是在 Join 操作的 Reduce 阶段（不是 Map 阶段），位于 Join 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的几率。

多个表关联时，最好分拆成小段，避免大 sql（无法控制中间 Job）。

多表 Join on 条件相同时合并为一个 Map-Reduce，做 OUTER JOIN 的时候也是一样，查看执行计划 explain

比如查询,3 个表关联：

```
select pt.page_id,count(t.url) PV
 from rpt_page_type pt
 join
 (select url_page_id,url from trackinfo where ds='2013-10-11') t
 on pt.page_id=t.url_page_id
 join
 (select page_id from rpt_page_kpi_new where ds='2013-10-11') r
 on t.url_page_id=r.page_id
 group by pt.page_id;
```

比较 2 个表关联：

```
select pt.page_id,count(t.url) PV
 from rpt_page_type pt
 join
 (select url_page_id,url from trackinfo where ds='2013-10-11') t
 on pt.page_id=t.url_page_id
group by pt.page_id;
```

利用这个特性，可以把相同 join on 条件的放在一个 job 处理。

如果 Join 的条件不相同，如：

```
INSERT OVERWRITE TABLE page_pv
select pt.page_id,count(t.url) PV
 from rpt_page_type pt
 join
 (select url_page_id,url,province_id from trackinfo where ds='2013-10-11') t
 on pt.page_id=t.url_page_id
 join
 (select page_id,province_id from rpt_page_kpi_new where ds='2013-10-11') r
 on t.province_id=r.province_id
 group by pt.page_id;
```

大表 Join 大表

访户未登录时，日志中 userid 是空，在用 user_id 进行 hash 分桶的时候，会将日志中 userid 为空的数据分到一起，导致了过大空 key 造成倾斜。

解决办法：

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

案例：

End_user 5000 万，纬度表

Trackinfo 每日 2 亿，按日增量表

原写法：

```
select u.id,t.url,t.track_time
 from end_user u
 join
 (select end_user_id,url,track_time from trackinfo where ds='2013-12-01') t
 on u.id=t.end_user_id limit 2;
```

调整为：

```
select u.id,t.url,t.track_time
 from end_user u
 join
(select case when end_user_id='null' or end_user_id is null
                then cast (concat('00000000_',floor(rand()*1000000)) as bigint)
                else end_user_id end end_user_id ,
          url,track_time
    from trackinfo where ds='2013-12-01') t
 on u.id=t.end_user_id limit 2;
```

此例子只是为了说明原理。

当前这个场景不需要这么麻烦,如下即可：

```
select u.id,t.url,t.track_time
 from end_user u
```

```
 join
(select end_user_id,
            url,track_time
    from trackinfo where ds='2013-12-01' and end_user_id is not null) t
 on u.id=t.end_user_id limit 2;
```

例子 2：

```
Select count(distinct end_user_id)
From trackinfo where ds='2013-12-01' ;
```

建议改为：

```
Select count(distinct end_user_id)+1
From trackinfo where ds='2013-12-01'    and end_user_id is not null;
```

MapJoin

Join 操作在 Map 阶段完成，如果需要的数据在 Map 的过程中可以访问到则不再需要 Reduce。

小表关联一个超大表时，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。

如：

```
INSERT OVERWRITE TABLE page_pv
select /*+ MAPJOIN(pt) */
        pt.page_id,count(t.url) PV
 from rpt_page_type pt
 join
 (select url_page_id,url from trackinfo where ds='2013-10-11') t
 on pt.page_id=t.url_page_id;
```

如果是小表，能否自动选择 Mapjoin?

set hive.auto.convert.join=true; **默认为** false

该参数为 true 时，Hive 自动对左边的表统计量，如果是小表就加入内存，即对小表使用 Map join

大表小表的阀值：

```
hive> set hive.mapjoin.smalltable.filesize;

hive.mapjoin.smalltable.filesize=25000000
```

默认值是 25mb

hive.mapjoin.cache.numrows
•说明： mapjoin 存在内存里的数据量
•默认值：25000

hive.mapjoin.followby.gby.localtask.max.memory.usage
•说明：map join 做 group by 操作时，可以使用多大的内存来存储数据，如果数据太大，则不会保存在内存里
•默认值：0.55

hive.mapjoin.localtask.max.memory.usage
•说明：本地任务可以使用内存的百分比
•默认值： 0.90

Group By

Map 端部分聚合：

并不是所有的聚合操作都需要在 Reduce 端完成，很多聚合操作都可以先在 Map 端进行部分聚合，最后在 Reduce 端得出最终结果。

基于 Hash

参数包括：

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

默认情况下，Map 阶段同一 Key 数据分发给一个 reduce，当一个 key 数据过大时就倾斜了

有数据倾斜的时候进行负载均衡

hive.groupby.skewindata = false;

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。

Count(distinct) 容易倾斜，当该字段存在大量值为 NULL 或空的记录

解决思路

count distinct 时，将值为空的数据在 where 里过滤调，在最后结果中加 1

如：

count(distinct end_user_id) as user_num

修正为

count(distinct end_user_id)+1 user_num

where end_user_id is not null and query <> ''

笛卡尔积

尽量避免笛卡尔积，join 的时候不加 on 条件，或者无效的 on 条件，Hive 只能使用 1 个 reducer 来完成笛卡尔积

On 中支持 udf 函数