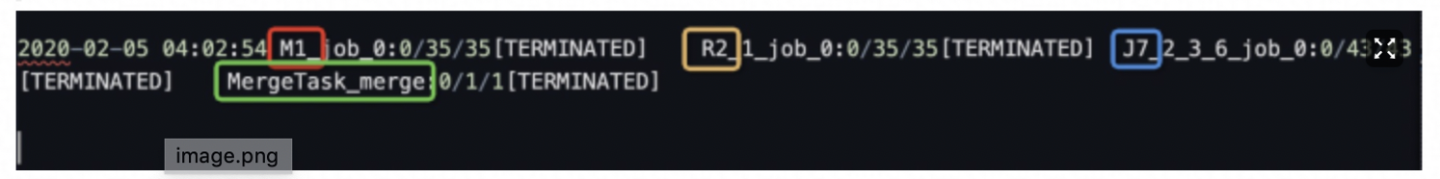
**另外地一个大重点则是去解决数据倾斜！！！**

数据倾斜，通俗地说就是某台机器（Instance）被分发到了明显大于其他机器的数据量，导致这台机器的处理量巨大，成为整个查询语句运行的“时间瓶颈”。

## **一、HQL 与 Mapreduce 任务对应关系**



ODPS 平台执行某段 SQL在运行中打印出来的日志，红框以M打头的对应Map task、黄框以R打头的为Reduce task、蓝框以J打头的为Join task、绿框以Merge打头的为文件合并任务。各任务类型与查询语句的对应关系大致为，

* Map task：select 等数据读取的操作，从磁盘中将数据读入内存；
* Reduce task：group by/order by 等聚合排序操作；
* Join task：Join 等表关联操作；
* Merge task：无对应代码，是小文件合并任务。

在 HQL 运行后可通过查看日志，观察每个 task 的运行时间或 I/O Bytes（ODPS 的伏羲任务平台也会有Long tails 直接标记出哪些 task 是长尾任务，长尾意味着运行时间长发生数据倾斜了）。**对应上面的任务类型，数据倾斜也分 3种：Map 数据倾斜、Reduce 数据倾斜、Join 数据倾斜。**

## **二、Map 数据倾斜**

Map 端读数据时，由于读入数据文件大小分布不均匀，因此导致有些 Map Instance 读取并且处理的数据特别多，而有些 Map Instance 处理的数据特别少，造成 Map 端长尾。

优化思路有以下 2 种，

**1）缩小读入数据量**

* 做好行裁剪：**务必确保好分区裁剪生效**，并通过 where 子句过滤不需要的数据；
* 做好列裁剪：能不用 select \* 就一定不要用；
* 有中间层可用就用中间层，如果没有则看是否能分段跑。比如要取3个月的数据，则可以分别写三段sql，每段取一个月的数据。

**那么分区裁剪是否生效可以怎么看呢？**

* 通过 Explain 语句或者 SQL 运行结束日志看数据读入包含了哪些分区
* 根据经验看到 Join 条件中分区裁剪条件如果在 on 子句中则生效，如果放在 where 条件中，主表的分区裁剪会生效，从表则不表。（所谓从表，即LEFT OUTER JOIN中的右表，或者RIGHT OUTER JOIN中的左表）
* 分区裁剪的条件中如果使用了函数操作，则分区裁剪很可能失效，即使是系统函数也可能失效。如下例子，

*--- 仅读取财年内日期***select** user\_id**from** user\_label**where** ds **>=** bi\_udf:bi\_get\_date(bi\_udf:bi\_fiscal\_year('${bizdate}'), **-**1) *--- 这是取阿里财年的 UDF 函数***and** ds **<** bi\_udf:bi\_get\_date(bi\_udf:bi\_fiscal\_year('${bizdate}'), 1)**group** **by** user\_id;

*--- 暴力扫描 500+分区***select** user\_id**from** eleme\_cdm.dws\_ele\_mbr\_level\_label**where** ds **>=**(

**case**

**when** **month**(to\_date('${bizdate}', 'yyyymmdd')) **>=**4 **then** concat(**year**(to\_date('${bizdate}', 'yyyymmdd')), '04', '01')

**when** **month**(to\_date('${bizdate}', 'yyyymmdd')) **<**4 **then** concat(**year**(to\_date('${bizdate}', 'yyyymmdd'))**-**1, '04', '01')

**end**)**group** **by** user\_id;

**2）合理使用参数控制上游小文件的合并**

**set** odps.**sql**.mapper.merge.**limit**.**size=**64; *--- 设定小文件合并的最大阈值，单位：M***set** odps.**sql**.mapper.split.**size=**256; *--- 设定一个 Map 的最大数据输入量，单位：M*

需要注意的是后者参数 set odps.sql.mapper.split.size=256; 需谨慎设置，设置过小会消耗过多机器资源，且可能出现 Map Instance 个数超过系统设置的情况。当需要的Map Instance个数太多，超过99999个Instance个数的限制。

physical plan generation failed: java.lang.RuntimeException: com.aliyun.odps.lot.cbo.FailFastException: instance count exceeds limit 99999.

## **三、Join 数据倾斜**

Join 执行阶段会将 Join Key 相同的数据分发到同一个执行 Instance 上处理 。  
如果某个Key 上的数据量比较大，则是发生数据倾斜，会导致该 Instance 执行时间较长。比如，电商大促场景下，某些大型店铺的 PV 会远远超过一般店铺，当用 PV表关联没店铺维度表时，会按照店铺 ID 纪念性分发，导致某些大卖家所在的 Instance 处理的数据量远远超过其他 Instance。，而整个任务会因为这个长尾 Instance 迟迟无法结束。

对应不同场景优化访问不同，

**1）当大小表关联且小表是从表时，使用 map join**

map join 可将小表放入内存中，避免长尾的分发。所谓从表，即LEFT OUTER JOIN中的右表，或者RIGHT OUTER JOIN中的左表。

**select** */\* mapjoin(b) \*/* a.c2, b.c3**from** (**select** c1, c2 **from** t1) a**left** **outer** **join** (**select** c1, c3 **from** t2 ) b **on** a.c1 **=** b.c1; *--- b表为小表*

**2）Join 的 2个表都是大表，且由于空值导致长尾，可将空值处理成随机值**

**select** col\_a, col\_b**from** table\_a **left** **join** table\_b **on** coalesce(table\_a.**key**, rand()**\***9999) **=** table\_b.**key**

**3）Join 的 2个表都是大表，且由于热点值导致长尾，可以先将热点Key取出，对于主表数据用热点Key切分成热点数据和非热点数据两部分分别处理，最后合并。比如下述示例**

1. 取出热点Key：将PV大于50000的商品ID取出到临时表

**insert** overwrite **table** topk\_item PARTITION (ds **=** '${bizdate}'）**select** item\_id**from**(

**select** item\_id, **count**(1) **as** cnt

**from** dwd\_tb\_log\_pv\_di

**where** ds **=** '${bizdate}'

**and** url\_type **=** 'ipv'

**and** item\_id **is** **not** **null**

**group** **by** item\_id) a**where** cnt **>=** 50000;

2. 取出非热点数据：将主表（sdwd\_tb\_log\_pv\_di）和热点key表（topk\_item）外关联后通过条件b1.item\_id is null，取出关联不到的数据即非热点商品的日志数据，此时需要用MAP JOIN。再用非热点数据关联商品维表，因为已经排除了热点数据，不会存在长尾。

**select** ...**from**(

**select** **\***

**from** dim\_tb\_itm

**where** ds **=** '${bizdate}') a**right** **outer** **join**(

**select** */\* mapjoin(b1) \*/*

b2.**\***

**from**(

**select** item\_id

**from** topk\_item

**where** ds **=** '${bizdate}'

) b1

**right** **outer** **join**(

**select** **\***

**from** dwd\_tb\_log\_pv\_di

**where** ds **=** '${bizdate}'

**and** url\_type **=** 'ipv'

) b2 **on** b1.item\_id **=** coalesce(b2.item\_id, concat("tbcdm",rand())

**where** b1.item\_id **is** **null**) l **on** a.item\_id **=** coalesce(l.item\_id,concat("tbcdm", rand());

3. 取出热点数据：将主表（sdwd\_tb\_log\_pv\_di）和热点Key表（topk\_item）内关联，此时需要用MAP JOIN，取到热点商品的日志数据。同时，需要将商品维表（dim\_tb\_itm）和热点Key表（topk\_item）内关联，取到热点商品的维表数据，然后将第一部分数据外关联第二部分数据，因为第二部分只有热点商品的维表，数据量比较小，可以用MAP JOIN避免长尾。

**select** */\* mapjoin(a) \*/*

...**from**(

**select** */\* mapjoin(b1) \*/*

b2.**\***

**from**(

**select** item\_id

**from** topk\_item

**where** ds **=** '${bizdate}'

)b1

**join**(

**select** **\***

**from** dwd\_tb\_log\_pv\_di

**where** ds **=** '${bizdate}'

**and** url\_type **=** 'ipv'

**and** item\_id **is** **not** **null**

) b2 **on** (b1.item\_id **=** b2.item\_id)) l**left** **outer** **join**(

**select** */\* mapjoin(a1) \*/*

a2.**\***

**from**(

**select** item\_id

**from** topk\_item

**where** ds **=** '${bizdate}'

) a1

**join**(

**select** **\***

**from** dim\_tb\_itm

**where** ds **=** '${bizdate}'

) a2 **on** (a1.item\_id **=** a2.item\_id)) a **on** a.item\_id **=** l.item\_id;

4. 将步骤2和步骤3的数据通过union all合并后即得到完整的日志数据，并且关联了商品的信息。

## **4）设置 odps.sql.skewjoin 参数解决长尾**

**set** odps.**sql**.skewjoin**=true**; *--- 开启功能***set** odps.**sql**.skewinfo**=**skewed\_src:(skewed\_key) [("skewed\_value")]; *--- 倾斜值较多，或会动态变化则不适合这样设置*

## **四、Reduce 数据倾斜**

Reduce 端负责的是将Map 端梳理后的有序 Key-value 键值对进行聚合，即进行count、sum、Avg 等聚合操作，得到最终聚合的结果。

什么样的场景会发生数据倾斜？聚合计算依赖的 key 分布不均匀时就会发生数据倾斜。比如，按店铺汇总订单量时，某一商户的订单量占到60%，则就可能发生长尾。

优化方法有，

**1）用两次 group by 代替 count distinct**

第一次 group by 用来去重数据记录达到缩小数据量的目的，第二次 group by 进行 count 聚合。

**select** ds, **count**(**distinct** userid), **count**(order\_id)**from**(

**select** ds, userid, order\_id

**from** table\_a *--- 子订单表，每订单含多个订单项* **group** **by** ds, userid, order\_id)**group** **by** ds;

**2）合理使用参数，开启二次分发**

**set** odps.**sql**.groupby.skewindata**=true**; *--- 长尾Instance，会二次分发规避长尾*

**3）不同指标的 count distinct 放到多段 SQL 中执行，执行后再 UNION 或 JOIN 合并**

多个 Distinct 同时出现在 SQL 代码中时（如对 uid、order\_id、shop\_id等均需去重技术时），数据会被分发多次，导致节点效率低。

## **五、以上优化执行后仍不能解决的 SQL 优化**

如果通过缩小数据量和上述 3种数据倾斜优化仍不能达到足够的查询优化效果。那么还有 2个不得已而为之的技巧：

**1）增加执行机器资源，有几个简单原则供借鉴：**

1. 增加机器资源时，优先 instance 个数：在没有出现数据倾斜的情况下，如果通过设置Cpu参数(含Memory参数)和设置Instance个数两种方式都能调优的话，最好是先设置Instance个数。因为如果Cpu/Memory参数设置不合理，执行任务的机器满足不了参数的要求，要重新找机器的，这样反而会影响效率。
2. 执行日志中出现Dump，最好是Instance个数/Memory都增大一下：如何选择合适的参数个数？用二分法寻找最合适instance 个数，如果一个instance处理的数据量降到了1亿以下，或者instance的执行时间小于15-20Min，那么就说明当前的资源设置已经比较恰当了。
3. 默认的Reduce instance一般是Map instance 的三分之一，一般Join instance个数一般是Reduce instance的个数之和

需要提醒的是过分依赖增加资源，会造成单个任务消耗过多资源，影响其他任务的正常运行。