**1.Fetch抓取（Hive可以避免进行MapReduce）**

Hive中对某些情况的查询可以不必使用MapReduce计算。例如：SELECT \* FROM employees;在这种情况下，Hive可以简单地读取employee对应的存储目录下的文件，然后输出查询结果到控制台。

在hive-default.xml.template文件中hive.fetch.task.conversion默认是more，老版本hive默认是minimal。

该属性修改为more以后，在全局查找、字段查找、limit查找等都不走mapreduce。

**2.开启本地模式**

有时Hive的输入数据量是非常小的。在这种情况下，为查询触发执行任务时消耗可能会比实际job的执行时间要多的多。对于大多数这种情况，Hive可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

用户可以通过设置hive.exec.mode.local.auto的值为true，来让Hive在适当的时候自动启动这个优化。

**3.Join优化**

3.1mapjoin

大表join小表

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。

map端join的参数设置：

设置自动选择mapjoin

set hive.auto.convert.join = true; -- 默认为true

大表小表的阈值设置：

set hive.mapjoin.smalltable.filesize= 25000000;

小表的输入文件大小的阈值（以字节为单位）;如果文件大小小于此阈值，它将尝试将common join转换为map join。

不等连接

select f.a,f.b from A f join B t where f.a > t.a；将连接条件写到where中mr会进行笛卡尔积，效率特别低，如果使用mapjoin将在map端进行不等值的join操作，进而提高效率

3.2大表join大表

3.2.1空key过滤，过滤掉为空的key，例如：SELECT a.\* FROM (SELECT \* FROM nullidtable WHERE id IS NOT NULL ) a JOIN ori b ON a.id = b.id;

3.2.2空key随机转换

有时虽然某个key为空对应的数据很多，但是相应的数据不是异常数据，必须要包含在join的结果中，此时我们可以表a中key为空的字段赋一个随机的值，使得数据随机均匀地分不到不同的reducer上。

为了解决这种情况，我们可以通过hive的rand函数，随记的给每一个为空的id赋上一个随机值，这样就不会造成数据倾斜。例如：

SELECT a.\* FROM nullidtable a LEFT JOIN ori b ON CASE WHEN a.id IS NULL THEN concat('hive', rand()) ELSE a.id END = b.id;

3.3Bucket-Mapjoin分桶join

大表join大表时，不足以放到内存中，如果想在map端进行join就要使用bucket-mapjoin。

实现方式：

两个join表在join key上都做hash bucket，并且把你打算复制的那个（相对）小表的bucket数设置为大表的倍数。这样数据就会按照key join，做hash bucket。小表依然复制到所有节点，Map join的时候，小表的每一组bucket加载成hashtable，与对应的一个大表bucket做局部join，这样每次只需要加载部分hashtable就可以了。

作用：

1.提高join查询效率

2.快速检测sql是否可用，可以达到快速开发目的

抽样检测

select \* from t\_bucket tablesample(bucket x out of y on xx);

如果表有别名tablesample函数紧跟着表名之后。

x表示从哪个bucket进行抽样,桶计数从1开始.y用来计算抽取数据的量，计算方式为分桶数/y.假设我们一共分了128个桶,y设置为32,则表示要抽取4个bucket,如果x为12,则抽取的数据来自于12/16/20/24….y的值可以为桶个数的公约数,可以为任意值.

条件：

1） set hive.optimize.bucketmapjoin = true;  
2） 一个表的bucket数是另一个表bucket数的整数倍  
3） bucket列 == join列  
4） 必须是应用在map join的场景中

注意：要向分桶表中填充成员，需要将 hive.enforce.bucketing 属性设置为 true。

分区表和分桶表的区别

分区表将一个文件按照分区字段放到不同的分区目录，提升查询性能，分区字段是额外的字段跟表字段无关

分桶表将一个文件按照分桶字段拆分多个文件，在特殊场景提升多表查询的性能，分桶字段是表的某个字段来进行分桶

3.4SMB JOIN sort merge bucket join

基于bucket-mapjoin的**有序bucket**，可实现在map端完成join操作，可以有效地减少或避免shuffle的数据量。

原理：首先进行排序，继而合并，然后放到所对应的bucket中去，bucket是hive中和分区表类似的技术，就是按照key进行hash，相同的hash值都放到相同的buck中去。在进行两个表联合的时候。我们首先进行分桶，在join会大幅度的对性能进行优化。也就是说，在进行联合的时候，是table1中的一小部分和table1中的一小部分进行联合，table联合都是等值连接，相同的key都放到了同一个bucket中去了，那么在联合的时候就会大幅度的减小无关项的扫描。

条件：

|  |  |
| --- | --- |
| **bucket mapjoin** | **SMB join** |
| set hive.optimize.bucketmapjoin = true; | --写入数据强制分桶  set hive.enforce.bucketing=true;  --写入数据强制排序  set hive.enforce.sorting=true;  --开启bucketmapjoin  set hive.optimize.bucketmapjoin = true;  --开启SMB Join  set hive.auto.convert.sortmerge.join=true;  set hive.optimize. bucketmapjoin.sortedmerge=true;  set hive.auto.convert.sortmerge.join.noconditionaltask=true; |
| 一个表的bucket数是另一个表bucket数的整数倍 | 小表的bucket数**=**大表bucket数 |
| bucket列 == join列 | Bucket 列 == Join 列 == **sort 列** |
| 必须是应用在map join的场景中 | 必须是应用在bucket mapjoin 的场景中 |

**4.SQL优化**

4.1列裁剪：读数据的时候只读查询所需的列，例如：select a,b from z where c < 10;

42分区裁剪：查询的时候加上分区的条件

4.3group by

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。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。

开启Map端聚合参数设置

--（1）是否在Map端进行聚合，默认为True

set hive.map.aggr = true;

--（2）在Map端进行聚合操作的条目数目

set hive.groupby.mapaggr.checkinterval = 100000;

--（3）有数据倾斜的时候进行负载均衡（默认是false）

set 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。

44count(distinct)

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，一般COUNT DISTINCT使用先GROUP BY再COUNT的方式替换

45笛卡尔积

尽量避免笛卡尔积，即避免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。例如select \* from a join b where a.id = b.id;应使用select \* from a join b on a.id = b.id; mysql对join做了

优化，会自动识别将where转换为on

#### 4.6Hive索引

Hive支持索引，但是Hive的索引与关系型数据库中的索引并不相同，比如，Hive不支持主键或者外键。

Hive索引可以建立在表中的某些列上，以提升一些操作的效率，例如减少MapReduce任务中需要读取的数据块的数量。

在可以预见到分区数据非常庞大的情况下，分桶和索引常常是优于分区的。而分桶由于SMB Join对关联键要求严格，所以并不是总能生效。

4.6.1Hive原始索引

Hive的索引目的是提高Hive表指定列的查询速度。

没有索引时，类似'WHERE tab1.col1 = 10' 的查询，Hive会加载整张表或分区，然后处理所有的rows，但是如果在字段col1上面存在索引时，那么只会加载和处理文件的一部分。

。在每次建立、更新数据后，Hive索引不会自动更新，需要手动进行更新（重建索引以构建索引表），会触发一个mr job

Hive索引使用过程繁杂，而且性能一般，在Hive3.0中已被删除，在工作环境中不推荐优先使用，在分区数量过多或查询字段不是分区字段时，索引可以作为补充方案同时使用。推荐使用ORC文件格式的索引类型进行查询。

4.6.2Row group index

一个ORC文件包含一个或多个stripes(groups of row data)，每个stripe中包含了每个column的min/max值的索引数据，当查询中有<,>,=的操作时，会根据min/max值，跳过扫描不包含的stripes。

而其中为每个stripe建立的包含min/max值的索引，就称为Row Group Index行组索引，也叫min-max Index大小对比索引，或者Storage Index。

在建立ORC格式表时，指定表参数’orc.create.index’=’true’之后，便会建立Row Group Index，需要注意的是，为了使Row Group Index有效利用，向表中加载数据时，必须对需要使用索引的字段进行排序，否则，min/max会失去意义。另外，这种索引主要用于数值型字段的查询过滤优化上。

设置hive.optimize.index.filter为true，并重启hive

创建表/插入数据：

|  |
| --- |
| CREATE TABLE test\_orc stored AS ORC  TBLPROPERTIES  (  'orc.compress'='SNAPPY',  *-- 开启行组索引*  'orc.create.index'='true'  )  AS  SELECT *CAST*(siteid AS INT) AS id,  pcid  FROM test\_text  *-- 插入的数据保持排序*  DISTRIBUTE BY id sort BY id; |

查询：

|  |
| --- |
| set hive.optimize.index.filter=true;  SELECT *COUNT*(1) FROM test\_orc WHERE id >= 1382 AND id <= 1399; |

4.6.3Bloom Filter Index

在建表时候，通过表参数”orc.bloom.filter.columns”=”pcid”来指定为那些字段建立BloomFilter索引，这样，在生成数据的时候，会在每个stripe中，为该字段建立BloomFilter的数据结构，当查询条件中包含对该字段的**=号过滤**时候，先从BloomFilter中获取以下是否包含该值，如果不包含，则跳过该stripe。

创建：

|  |
| --- |
| CREATE TABLE lxw1234\_orc2 stored AS ORC  TBLPROPERTIES  (  'orc.compress'='SNAPPY',  'orc.create.index'='true',  *-- pcid字段开启BloomFilter索引*  "orc.bloom.filter.columns"="pcid"  )  AS  SELECT *CAST*(siteid AS INT) AS id,  pcid  FROM lxw1234\_text  DISTRIBUTE BY id sort BY id; |

查询：

|  |
| --- |
| SET hive.optimize.index.filter=true;  SELECT *COUNT*(1) FROM lxw1234\_orc1 WHERE id >= 0 AND id <= 1000  AND pcid IN ('0005E26F0DCCDB56F9041C','A'); |

只有在数据量较大时，使用索引才能带来性能优势。

**5.Hive并行操作**

5.1Hive编译查询限制

Hive默认同时只能编译一段HiveQL，并上锁。

将hive.driver.parallel.compilation设置为true，各个会话可以同时编译查询，提高团队工作效率。否则如果在UDF中执行了一段HiveQL，或者多个用户同时使用的话, 就会锁住。

修改hive.driver.parallel.compilation.global.limit的值，0或负值为无限制，可根据团队人员和硬件进行修改，以保证同时编译查询。

5.2Hive小文件合并

此部分设置，要根据硬件内存来进行调整，个人电脑配置较低，不建议修改。

hive.merge.mapfiles

是否开启合并Map端小文件，在Map-only的任务结束时合并小文件，true是打开。

hive.merge.mapredfiles

是否开启合并Reduce端小文件，在map-reduce作业结束时合并小文件。true是打开。hive.merge.size.per.task

合并后MR输出文件的大小，默认为256M。

hive.merge.smallfiles.avgsize

当输出文件的平均大小小于此设置值时，启动一个独立的map-reduce任务进行文件merge，默认值为16M。

5.3矢量化查询

hive的默认查询执行引擎一次处理一行，而矢量化查询执行是一种hive特性，目的是按照每批1024行读取数据，并且一次性对整个记录整合（而不是对单条记录）应用操作，注意：要使用矢量化查询执行，就必须以ORC格式存储数据。

set hive.vectorized.execution.enabled=true;

5.4读取零拷贝

ORC可以使用新的HDFS缓存API和ZeroCopy读取器来避免在扫描文件时将额外的数据复制到内存中。

set hive.exec.orc.zerocopy=true;

**6.关联优化器**

6.1shulffle共享

在Hive的一些复杂关联查询中，可能同时还包含有group by等能够触发shuffle的操作，有些时候shuffle操作是可以共享的，通过关联优化器选项，可以尽量减少复杂查询中的shuffle，从而提升性能。

set hive.optimize.correlation=true;

6.2数据倾斜Skew join

6.2.1表连接数据倾斜skew join

**set hive.optimize.skewjoin=true;**

默认关闭。

如果大表和大表进行join操作，则可采用skew join(倾斜关联)来开启对倾斜数据的优化。

**Skew join原理：**

1. 对于skew join.key，在执行job时，将它们存入临时的HDFS目录，其它数据正常执行
2. 对倾斜数据开启map join操作（多个map并行处理），对非倾斜值采取普通join操作
3. 将倾斜数据集和非倾斜数据集进行合并union操作。

开启skew join以后，究竟多大的数据才会被认为是倾斜了的数据呢？

**set hive.skewjoin.key=****100000;**

默认值100000。

如果join的key对应的记录条数超过这个值，就认为这个key产生了数据倾斜，则会对其进行分拆优化。

6.2.2编译时优化

上面的配置项其实应该理解为hive.optimize.skewjoin.runtime，也就是sql运行时来对偏斜信息进行优化；除此之外还有另外一个配置：

**set hive.optimize.skewjoin.****compiletime=true;**

默认关闭。

此参数的用处和上面的hive.optimize.skewjoin一致，但在编译sql时就已经将执行计划优化完毕。但要注意的是，只有在表的元数据中存储的有数据倾斜信息时，才能生效。因此建议runtime和compiletime都设置为true。

可以通过建表语句来指定数据倾斜元数据：

CREATE TABLE list\_bucket\_single (key STRING, value STRING)

*-- 倾斜的字段和需要拆分的key值*

SKEWED BY (key) ON (1,5,6)

*-- 为倾斜值创建子目录单独存放*

[STORED AS DIRECTORIES];

CREATE TABLE list\_bucket\_single (col1 STRING, clo2 STRING,clo3 STRING)

*-- 倾斜的字段和需要拆分的key值*

SKEWED BY (col1,clo2) ON (1,1),(2,2),(3,3)

*-- 为倾斜值创建子目录单独存放*

[STORED AS DIRECTORIES];

6.2.3union优化

应用了表连接倾斜优化以后，会在执行计划中插入一个新的union操作，此时建议开启对union的优化配置：

**set hive.optimize.union.remove=true;**

默认关闭。

此项配置减少对Union all子查询中间结果的二次读写，可以避免union输出的额外扫描过程，当我们开启了skewjoin时尤其有用，建议同时开启。

set hive.optimize.skewjoin=true;

set hive.optimize.skewjoin.compiletime=true;

set hive.optimize.union.remove=true;

6.3分组统计数据倾斜

6.3.1map阶段聚合

**hive.map.aggr=true;**

开启map端combiner。此配置可以在group by语句中提高HiveQL聚合的执行性能。这个设置可以将顶层的聚合操作放在Map阶段执行，从而减轻数据传输和Reduce阶段的执行时间，提升总体性能。默认开启，无需显示声明。

6.3.2两个mr job

**hive.groupby.skewindata=true;**

默认关闭。

这个配置项是用于决定group by操作是否支持倾斜数据的负载均衡处理。当数据出现倾斜时，如果该变量设置为true，那么Hive会自动进行负载均衡。

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。

注意：在多个列上进行的去重操作与hive环境变量hive.groupby.skewindata存在冲突。

当hive.groupby.skewindata=true时，hive不支持多列上的去重操作，并报错：

|  |
| --- |
| Error in semantic analysis: DISTINCT on different columns notsupported with skew in data. |

比如：

|  |
| --- |
| (1) SELECT count(DISTINCT uid) FROM log  (2) SELECT ip, count(DISTINCT uid) FROM log GROUP BY ip  (3) SELECT ip, count(DISTINCT uid, uname) FROMlog GROUP BY ip  (4) SELECT ip, count(DISTINCT uid), count(DISTINCT uname) FROMlog GROUP BY ip |

1、2、3能够正常执行，但是4会报错。