Java 私塾 Hive QL 详解

第一部分：Hadoop 计算框架的特性

什么是数据倾斜

•由于数据的不均衡原因，导致数据分布不均匀，造成数据大量的集中到一点，造成数据热点

Hadoop 框架的特性

•不怕数据大，怕数据倾斜

•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，耗时很长。原因是 map reduce 作业初始化的时间是比较长的

•sum,count,max,min 等 UDAF，不怕数据倾斜问题,hadoop 在 map 端的汇总合并优化，使数据倾斜不成问题

•count(distinct),在数据量大的情况下，效率较低，因为 count(distinct)是按 group by 字段分组，按 distinct 字段排序，一般这种分布方式是很倾斜的

第二部分：优化的常用手段

优化的常用手段

•解决数据倾斜问题

•减少 job 数

•设置合理的 map reduce 的 task 数，能有效提升性能。

•了解数据分布，自己动手解决数据倾斜问题是个不错的选择

•数据量较大的情况下，慎用 count(distinct)。

•对小文件进行合并，是行至有效的提高调度效率的方法。

•优化时把握整体，单个作业最优不如整体最优。

第三部分：Hive 的数据类型方面的优化

优化原则

•按照一定规则分区（例如根据日期）。通过分区，查询的时候指定分区，会大大减少在无用数据上的扫描，同时也非常方便数据清理。

•合理的设置 Buckets。在一些大数据 join 的情况下，map join 有时候会内存不够。如果使用 Bucket Map Join 的话，可以只把其中的一个 bucket 放到内存中，内存中原来放不下的内存表就变得可以放下。这需要使用 buckets 的键进行 join 的条件连结，并且需要如下设置

```
set hive.optimize.bucketmapjoin = true
```

第四部分：Hive 的操作方面的优化

•全排序

•怎样做笛卡尔积

•怎样决定 map 个数

•怎样决定 reducer 个数

•合并 MapReduce 操作

•Bucket 与 sampling

•Partition

•JOIN

•Group By

•合并小文件

全排序

•Hive 的排序关键字是 SORT BY，它有意区别于传统数据库的 ORDER BY 也是为了强调两者的区别–SORT BY 只能在单机范围内排序

怎样做笛卡尔积

•当 Hive 设定为严格模式（hive.mapred.mode=strict）时，不允许在 HQL 语句中出现笛卡尔积

•MapJoin 是的解决办法

•MapJoin，顾名思义，会在 Map 端完成 Join 操作。这需要将 Join 操作的一个或多个表完全读入内存

•MapJoin 的用法是在查询/子查询的 SELECT 关键字后面添加/*+ MAPJOIN(tablelist) */提示优化器转化为 MapJoin（目前 Hive 的优化器不能自动优化 MapJoin）

•其中 tablelist 可以是一个表，或以逗号连接的表的列表。tablelist 中的表将会读入内存，应该将小表写在这里

•在大表和小表做笛卡尔积时，规避笛卡尔积的方法是，给 Join 添加一个 Join key，原理很简单：将小表扩充一列 join key，并将小表的条目复制数倍，join key 各不相同；将大表扩充一列 join key 为随机数

控制 Hive 的 Map 数

•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)

•是不是 map 数越多越好

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m）,则每个小文件也会被当做一个块，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

•是不是 map 数越多越好

答案是否定的。如果一个任务有很多小文件（远远小于块大小 128m）,则每个小文件也会被当做一个块，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

•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；

•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；

•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

即，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

怎样决定 reducer 个数

•Hadoop MapReduce 程序中，reducer 个数的设定极大影响执行效率

•不指定 reducer 个数的情况下，Hive 会猜测确定一个 reducer 个数，基于以下两个设定：

参数 1：hive.exec.reducers.bytes.per.reducer（默认为 1G)

参数 2 ：hive.exec.reducers.max（默认为 999）

•计算 reducer 数的公式

•N=min(参数 2，总输入数据量/参数 1)

•依据 Hadoop 的经验，可以将参数 2 设定为 0.95*(集群中 TaskTracker 个数)

•reduce 个数并不是越多越好

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

•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，不管你有没有设置调整 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：

a) 没有 group by 的汇总

b) 用了 Order by

合并 MapReduce 操作

• Multi-group by

•Multi-group by 是 Hive 的一个非常好的特性，它使得 Hive 中利用中间结果变得非常方便

•FROM log

• insert overwrite table test1 select log.id group by log.id

• insert overwrite table test2 select log.name group by log.name

• 上述查询语句使用了 Multi-group by 特性连续 group by 了 2 次数据，使用不同的 group by key。这一特性可以减少一次 MapReduce 操作。

Bucket 与 Sampling

•Bucket 是指将数据以指定列的值为 key 进行 hash，hash 到指定数目的桶中。这样就可以支持高效采样了

•Sampling 可以在全体数据上进行采样，这样效率自然就低，它还是要去访问所有数据。而如果一个表已经对某一列制作了 bucket，就可以采样所有桶中指定序号的某个桶，这就减少了访问量。

•如下例所示就是采样了 test 中 32 个桶中的第三个桶。

•SELECT * FROM test 、、、TABLESAMPLE(BUCKET 3 OUT OF 32);

JOIN 原则

•在使用写有 Join 操作的查询语句时有一条原则：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 OOM 错误的几率

Map Join

•Join 操作在 Map 阶段完成，不再需要 Reduce，前提条件是需要的数据在 Map 的过程中可以访问到

•例如：

•INSERT OVERWRITE TABLE phone_traffic

SELECT /*+ MAPJOIN(phone_location) */ l.phone,p.location,l.traffic from phone_location p join log l on (p.phone=l.phone)

•相关的参数为：

hive.join.emit.interval = 1000 How many rows in the right-most join operand Hive should buffer before emitting the join result.

hive.mapjoin.size.key = 10000

hive.mapjoin.cache.numrows = 10000

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

•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。

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