数据治理-计算资源治理

前言

由于数据治理层面可以分多个层面且内容繁多（包括模型合规、数据质量、数据安全、计算/存储资源、数据价值等治理内容），因此需要单独拆分为6个模块单独去阐述其中内容。

作为数仓开发经常会收到大量集群资源满载、任务产出延时等消息/邮件，甚至下游数分及其他同学也会询问任务运行慢的情况，在这里很多数仓同学遇到这类问题第一想到的都是加资源解决，但事实真不一定是缺少资源，而是需要优化当前问题任务，所以本期从语兴团队做计算资源治理视角出发带大家清楚计算资源到底该如何进行,同时也希望这篇文章能够帮助在做治理/面试的同学(需要Maxcompute调优资料同学可以扫描下方二维码进群获得)。

1. 问题出现

在做计算治理之前（2022.12）我们团队盘点了下当前计算资源存在的几个问题

1. **30+高消耗任务：**由于数仓前中期业务扩张，要覆盖大量场景应用，存在大量问题代码运行时数据倾斜，在消耗大量集群计算资源下，产出时间也久；
2. **200w+的小文件：**当前任务存在未合并小文件、任务Reduce数量过多、上游数据源接入（尤其是API数据接入）会造成过多小文件出现，小文件过多会开启更多数据读取，执行会浪费大量的资源，严重影响性能；
3. **任务调度安排不合理：**多数任务集中在凌晨2-5点执行且该区间CPU满载，导致该时间段资源消耗成了重灾区，所有核心/非核心任务都在争抢资源，部分核心任务不能按时产出一直在等待阶段；
4. **线上无效DQC（数据质量监控）&监控配置资源过小：**存在部分历史任务没下线表及DQC场景，每日都在空跑无意义DQC浪费资源，同时DQC资源过少导致DQC需要运行过长时间，这里考虑到新同学不了解DQC，跳转链接：https://mp.weixin.qq.com/s/Li8sxDhuRPVIatX5tuUAHA
5. **重复开发任务/无用任务：**早期协助下游做了较多烟囱数据模型，因为种种原因，部分任务不再被使用，烟囱模型分散加工导致资源复用率降低
6. **任务缺少调优参数&部分任务仍然使用MapReduce/Spark2计算引擎：**任务缺少调优参数导致资源不能适配及动态调整，甚至线上仍有早期配置MapReduce/Spark2计算引擎导致运行效率较低

二、思考与行动

**治理前的思考：**

在治理之前我想到一个问题-切入点该从哪里开始最合适，经过与团队多次脑暴对当前治理优先级/改动成本大小/难度做了一个排序因此我们先选择从简单的参数调优&任务引擎切换开始->小文件治理->DQC治理->高消耗任务治理->调度安排->下线无用模型及沉淀指标到其他数据资产,同时在初期我们完成各类元数据接入搭建治理看板以及团队治理产出统计数据模型，并通过数据治理平台解决具体细节问题

数据治理平台图

图形用户界面, 应用程序, 网站

描述已自动生成

治理行动：

1. **大部分任务切换至Spark3计算引擎&补充任务调优参数：**

补充Spark调优参数（参数内容详见标题5），任务统一使用Spark3引擎加速，并充分利用Spark3的AQE特性及Z-Order排序算法特性

AQE解释：Spark 社区在 DAG Scheduler 中，新增了一个 API 在支持提交单个 Map 阶段，以及在运行时修改 shuffle 分区数等等，而这些就是 AQE，在 Spark 运行时，每当一个 Shuffle、Map 阶段进行完毕，AQE 就会统计这个阶段的信息，并且基于规则进行动态调整并修正还未执行的任务逻辑计算与物理计划（在条件运行的情况下），使得 Spark 程序在接下来的运行过程中得到优化。

Z-Order解释：Z-Order 是一种可以将多维数据压缩到一维的技术，在时空索引以及图像方面使用较广，比如我们常用order by a,b,c 会面临索引覆盖的问题，Z-Order by a,b,c 效果对每个字段是对等的

1. **小文件治理：**

在这里我们使用内部数据治理平台-数据治理360对存在小文件较多表提供内容展示（本质采集HDFS对应路径下文件数的日志去显示）

当前小文件处理：

（1）对于分区较多使用Spark3进行动态分区刷新，（Spark3具备小文件自动合并功能，如未使用Spark3可配置Spark3/Hive小文件合并参数刷新，参数在标题5）

代码如下

set hive.exec.dynamic.partition.mode=nonstrict;

insert overwrite table xxx.xxx partition (ds)

select column

,ds

from xxx.xxx

（2）对于分区较少或未分区的表采用重建表，补数据方法回刷

小文件预防：

（1）使用Spark3引擎，自动合并小文件

（2）减少Reduce的数量(可以使用参数进行控制)

（3）用Distribute By Rand控制分区中数据量

（4）添加合并小文件参数

（5）将数据源抽取后的表做一个任务（本质也是回刷分区合并小文件任务）去处理小文件保障从数据源开始小文件不向下游流去

1. **DQC治理：**

无效DQC下线：难点在于需要查找所有DQC对应的线上任务，查看该DQC任务是否与线上任务一一匹配，从而找到无效DQC任务下线，内容繁杂耗时较多

DQC资源：由于之前DQC配置资源为集群默认参数，效率极低导致所有DQC运行时长均超过10min，从而使得整体任务链路运行时长过久，调整Driver内存为2048M，Executor个数为2，Executor内存为4096M

1. **\*高消耗任务调优：**

这里存在2个难点：优化效果不可控、高消耗任务调整到何种程度算合适，针对这个这个难点我们取所有核心数据资产任务均值，保障单个任务消耗小于平均消耗，同时我们针对当前高消耗任务列举出如下可优化的方式：

1. 关联表过多，需拆分
2. 关联时一对多，数据膨胀
3. 资源配置过多，运行时资源严重浪费，需要将配置调小（包括Driver内存、Executor个数、Executor内存）
4. 代码结尾添加Distribute By Rand（），用来控制Map输出结果的分发
5. 查询中列和行未裁剪、分区未限定、Where条件未限定
6. SQL中Distinct切换为Group by（Distinct会被hive翻译成一个全局唯一Reduce任务来做去重操作，Group by则会被hive翻译成分组聚合运算，会有多个Reduce任务并行处理，每个Reduce对收到的一部分数据组，进行每组聚合（去重））
7. 关联后计算切换为子查询计算好后再关联
8. 使用Map Join（Map Join会把小表全部读入内存中，在Map阶段直接拿另外一个表的数据和内存中表数据做匹配，由于在Map是进行了Join操作，省去了Reduce运行的效率也会高很多）可用参数代替
9. **任务调度合理优化：**

对于调度优化一开始会无从下手，统计凌晨2-5点区间下大概600+任务难梳理，同时存在任务依赖，修改起来可能会对下游整体有大的影响，因此我们选择循序渐进先梳理再改善

（1）找到所有表的输出输入点即启始ODS与末尾ADS

（2）划分其中核心表/非核心表，及对应任务开始时间与结束时间

（3）按照梳理内容把非核心的任务穿插在当前集群资源非高峰时期（2点前与5点后），同时把核心任务调度提前，保障CDM层任务及时产出

（4）对实践后内容再度调优，达到资源最大利用率

1. **烟囱任务下沉&无用任务下线：**

烟囱表过多，需下沉指标到DWS中提升复用性，对于无用任务也需要及时下线（这里需要拿到元数据血缘最好到报表层级的数据血缘，防止任务下线后导致可视化内容问题产生），减少开发资源消耗，

1. 治理效果

1.Hive与Spark2任务升级Spark3.1，总计升级任务137个,升级任务后总体任务执行效率提升43%，cpu资源消耗降低41%，内存资源消耗降低46%

2. 治理小文件数大于10000+以上的数仓表总计30+张，小文件总数由216w下降至67w

3. 下线无效DQC任务总计50+，修改DQC配置资源降低运行时长，由原来10min优化至3min内

4.完成线上30+个任务优化及20+个任务下线及20+表指标下沉，优化后节省任务耗时146分钟，减少CPU损耗800w+，损耗下降53%，降低内存消耗2600w+，损耗下降48%（相当于节省了8个200+字段5亿数据量任务消耗）

5.调度重新分配后2-5点资源使用率由90+%降低至50+%

效果趋势图：

CPU

图表, 折线图, 直方图

描述已自动生成

Memory

图表, 直方图

描述已自动生成

1. 整体计算资源治理思考

计算资源治理核心在于降本增效，用有限资源去运行更多任务，通过一系列治理操作也让数仓同学积累技术经验同时规范化自身开发标准，让治理反推进组内技术进步，计算资源治理是一件长久之事，并不能因为资源紧张才去治理，而要将计算治理常态化，可通过周/月资源扫描内容及时推送给每个同学，并为之打分，让每个任务都有源可循，有方法可优化。

1. 参数内容

参数并不是设置越多任务性能越好，根据数据量、消耗、运行时间进行调整达到合理效果

Hive：

（1）set hive.auto.convert.join = true; （是否自动转化成Map Join）

（2）set hive.map.aggr=true; （用于控制负载均衡，顶层的聚合操作放在Map阶段执行，从而减轻清洗阶段数据传输和Reduce阶段的执行时间，提升总体性能，该设置会消耗更多的内存）

（3）set hive.groupby.skewindata=true; （用于控制负载均衡，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）

（4）set hive.merge.mapfiles=true; （用于hive引擎合并小文件使用）

（5）set mapreduce.map.memory.mb=4096; （设置Map内存大小，解决Memory占用过大/小）

（6）set mapreduce.reduce.memory.mb=4096;（设置Reduce内存大小，解决Memory占用过大/小）

（7）set hive.exec.dynamic.partition.mode=nonstrict;（动态分区开启）

Spark：

（1）set spark.sql.legacy.parquet.datetimeRebaseModeInRead=LEGACY;（用于Spark3中字段类型不匹配（例如datetime无法转换成Date），消除SQL中时间歧义，将Spark .sql. LEGACY . timeparserpolicy设置为LEGACY来恢复Spark 3.0之前的状态来转化）

（2）set spark.sql.adaptive.enabled=true;（是否开启调整Partition功能，如果开启，spark.sql.shuffle.partitions设置的Partition可能会被合并到一个Reducer里运行。平台默认开启，同时强烈建议开启。理由：更好利用单个Executor的性能，还能缓解小文件问题）

（3）set spark.sql.hive.convertInsertingPartitionedTable=false;（解决数据无法同步Impala问题，使用Spark3引擎必填）

（4）set spark.sql.finalStage.adaptive.advisoryPartitionSizeInBytes=2048M;（Spark小文件合并）