

异步查询 OOM

一、相关背景：

1. 下载异步查询结果时发生 OOM。
2. 使用异步查询下载查询结果失败，数据量是 400W 行，在对 数据进行 collect 之前报错

SQL

```
estimate memory usage ${estimateAccumulatedSize} " +  
s"> allowed maximum memory usage ${maxAllowedSize}
```

这里报错是有一个 开关设置：kylin.query.memory-limit-during-collect-mb，后来改成流式处理查询结果后，在击中模型和查询下压中不再受这个开关控制，但是 spark 中还有这个开关的使用场景。

```
val jobTrace = new SparkJobTrace(jobGroup, QueryContext.current())  
val rows = df.collect()  
if (kapConfig.isQuerySparkJobTraceEnabled) jobTrace.jobFinished()  
QueryContext.current().getMetrics.setScanRows(scanRows)  
QueryContext.current().getMetrics.setScanBytes(scanBytes)  
val (scanRows, scanBytes) = QueryMetricUtils.collectScanMetrics(scanRows, scanBytes)  
QueryContext.current().getMetrics.setScanRows(scanRows)  
QueryContext.current().getMetrics.setScanBytes(scanBytes)  
val resultTypes = rowType.getFieldList.asScala  
val dt = convertResultWithMemLimit(rows) { row =>  
  if (Thread.currentThread().isInterrupted()) {  
    throw new InterruptedException  
  }  
  row.toSeq.zip(resultTypes).map {  
    case (value, relField) => SparkerTypeUtil.convertToString(value, relField)  
  }.asJava  
}.toSeq.asJava  
jobTrace.result.record("transform_result")  
QueryContext.current().record("transform_result")
```

```
private[spark] def addSizeOfArraySizeEstimatedByFistElement(obj: Any) : Unit = {  
  estimateAccumulatedSize += SizeEstimator.estimate(obj.asInstanceOf[AnyRef]) * rowCount  
  if (estimateAccumulatedSize > maxAllowedSize) {  
    throw new SparkException(s"estimate memory usage ${estimateAccumulatedSize} " +  
      s"> allowed maximum memory usage ${maxAllowedSize}")  
  }  
}  
org.apache.spark.sql.util.CollectExecutionMemoryUsage  
private def addSizeOfArraySizeEstimatedByFistElement(obj: Any): Unit  
spark-sql_2.12
```

二、异步查询行为梳理

1. 异步查询背景

1. 异步查询会提交查询，但是不会立即返回查询结果。可以通过 GET `http://host:port/kylin/api/async_query/{query_id}/status?project=learn_kylin` 返回这条 查询的查询状态，若返回显示成功，则表示查询完成。
2. 可以通过 GET `http://host:port/kylin/api/async_query/{query_id}/result_download?include_header=true&project=learn_kylin` 拿到异步查询结果。
3. 在提交异步查询时，可以指定 format (csv、xlsx、json、parquet)，默认 为 csv 格式，指定为什么样的 format，在 hdfs 中存储的结果文件就是什么样的格式。
4. 指定存储格式为 csv 时，支持 separator 导出结果的分隔符，默认为逗号。

2、报错 estimate memory usage 大于 allowed maximum memory usage

异步查询下载结果的时候，使用 `org.apache.kylin.rest.service.CSVWriter#writeDataByCsv` 根据路径直接读取结果的时候，在这 `org.apache.spark.sql.Dataset#collectFromPlan` 进行结果收集，在这里会对 结果进行一个 预估计。

```
private[spark] def addSizeOfArraySizeEstimatedByFistElement(obj: Any) : Unit = {  
    estimateAccumulatedSize += SizeEstimator.estimate(obj.asInstanceOf[AnyRef]) * rowCount  
    if (estimateAccumulatedSize > maxAllowedSize) {  
        throw new SparkException(s"estimate memory usage ${estimateAccumulatedSize} " +  
            s"> allowed maximum memory usage ${maxAllowedSize}")  
    }  
}
```

具体行为解释：



Li Xian (洗立) (不属于请求参与者) 2020年12月15日 19:36 已编辑

现状

如上所述原因，在collect阶段会生成

1. UnsafeRow
2. GenericRow
3. KE 中会把GenericRow 中的值转为String

导致实际内存消耗比源表膨胀许多；而当前maxCollectSize 值考虑了 UnsafeRow 的大小，没有针对后两者考虑内存消耗，导致即使设置了maxCollectSize 任然会观察到内存消耗超过设定值

Dev Design

1. spark 添加conf `spark.sql.driver.maxMemoryUsageDuringCollect` 控制collect 阶段的内存消耗
2. spark 添加类 `CollectExecutionMemoryUsage` 估算collect时生成的 UnsafeRow 和 GenericRow 的大小；其中使用 array 的第一个 element 的大小乘以array长度作为array的大小。`CollectExecutionMemoryUsage` 检查估算的内存消耗是否超过 `maxMemoryUsageDuringCollect`，如果是则抛异常
3. 如果在spark collect 时候内存消耗没有抛异常，KE 从 `CollectExecutionMemoryUsage` 中获取估算的内存消耗大小，然后再估算把结果 装换为String需要消耗的内存，检查是否超过 `maxMemoryUsageDuringCollect`，超过则抛异常
4. KE 中添加conf `kylin.query.memory-limit-during-collect-mb` 配置collect阶段 KE + spark 中的内存消耗，KE根据 `kylin.query.memory-limit-during-collect-mb` 配置 `spark.sql.driver.maxMemoryUsageDuringCollect`，并转换成String的时候的 内存消耗

☺

详

经:

开:

报:

Te

Sc

Sp

原:

修:

Cu

标:

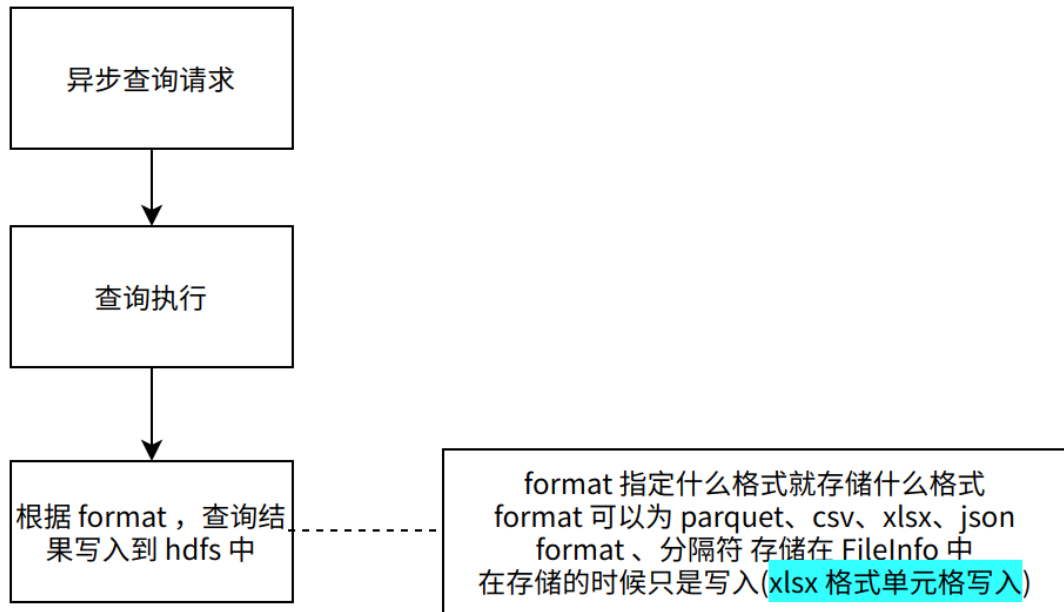
当 `estimateAccumulatedSize > maxAllowedSize` 的时候，就会报上述错误。
`maxAllowedSize` 为 `kylin.query.memory-limit-during-collect-mb` 设置的大小。

3、异步查询 OOM 问题

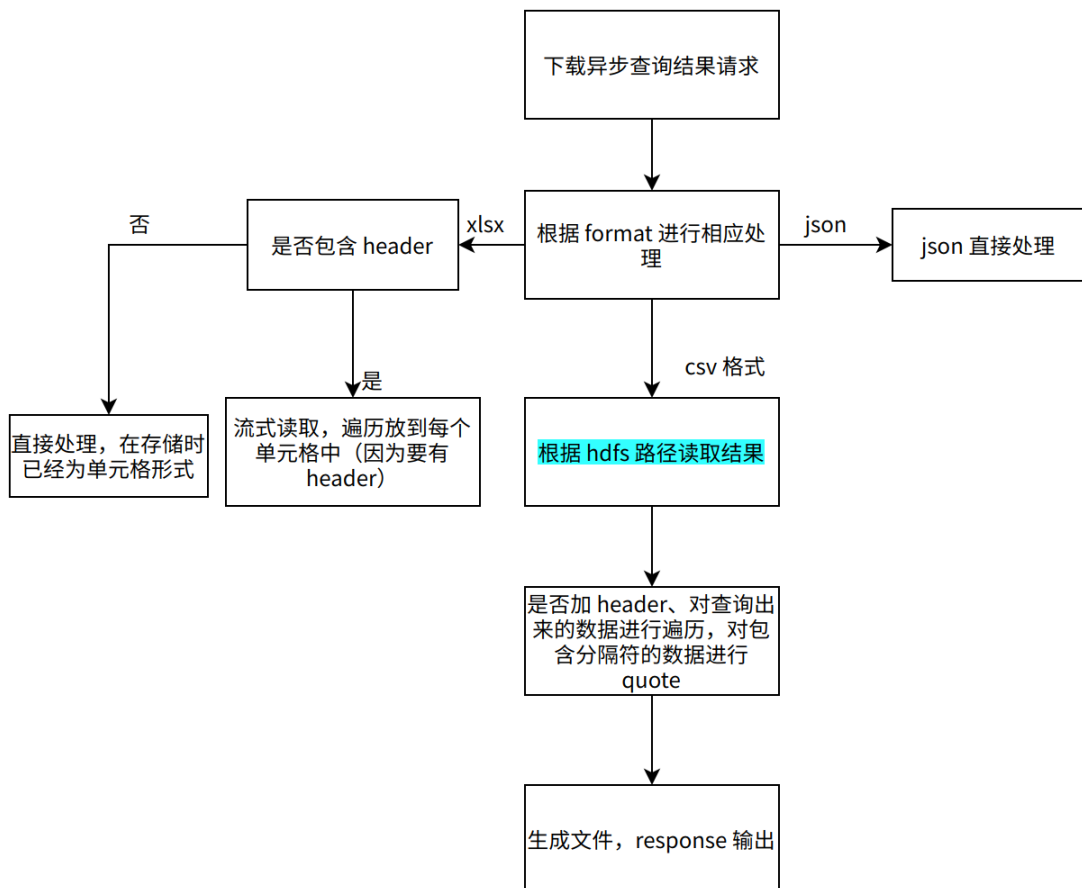
分析出来的主要原因是在进行异步查询结果下载的时候，**format 为 xlsx 格式**，从 hdfs 拿到查询结果之后，对 parquet 格式转换时，对结果进行 遍历 set 到 单元格 中，这里耗费的时间比较长。

4、现如今行为梳理

异步查询请求：



下载异步查询结果：



在提交异步查询的时候，默认 format 为 csv 格式；可以指定 format 为 csv、parquet、xlsx、json。

1. 在提交异步查询将结果存储到 hdfs 上的时候
 - a. csv 格式，采用 spark 原生的 csv 方法写入，重写 df 的 columns
 - b. xlsx 格式，通过 poi 接口生成 excel 文件，上传到 hdfs 上，这里用到了 **df.collect()**
 - c. Json 格式
 - d. parquet，原生 spark 写入 parquet 的方式
2. 异步查询完成后，下载异步查询结果。

在提交异步查询的时候，会根据指定的 format 下载，**暂不支持 parquet 下载**。这个信息来源于查询请求，这个信息会存储到 FileInfo 中并保存下来，下载的时候会读取这个 format，并根据这个格式去下载。

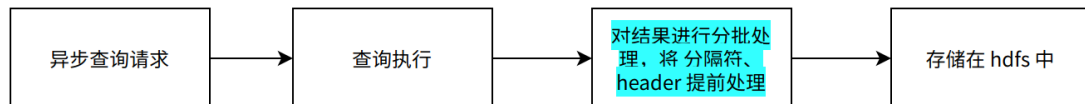
- a. csv 格式下载，直接使用 spark 原生的 read csv 文件的方式，这里会调用 **df.collect()**，不管是否有 header，都需要需要对分隔符处理（字符乱码，多分隔符）
- b. xlsx 格式 格式下载，不带有 header 直接处理；带有 header，会进行格式转换，转换会比直接处理要慢。

三、DEV DESIGN

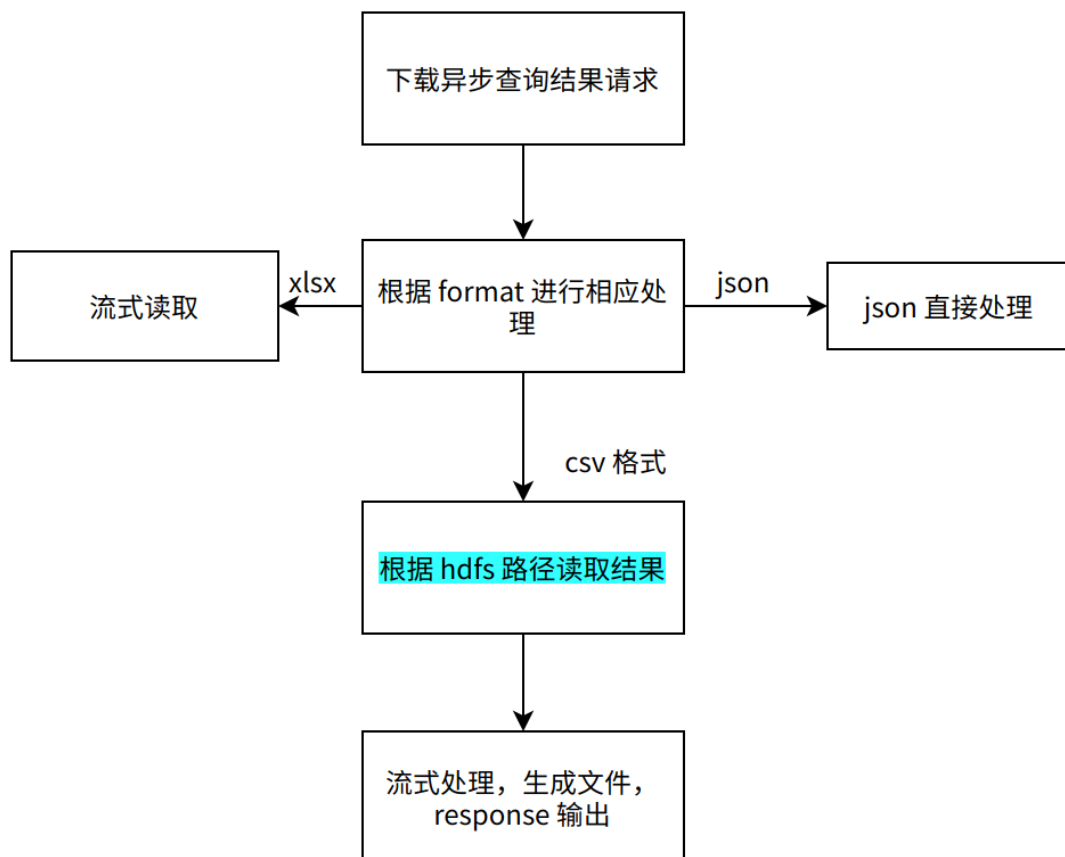
1. 目前存在的问题
 - a. 对于 parquet 格式和 json 格式的问题较少，直接进行读取处理即可
 - b. 对于 csv 格式文件，存储结果会先一次性读取出来再遍历，数据量大时，下载会导致 ke 内存飙升。
 - c. 对于 xlsx 格式的文件，需要遍历将数据 set 到每个单元格中，耗费时间较长。
2. 将在下载结果中读取全部数据 **df.collect()** 的步骤删掉，在提交异步查询请求将结果存储在 hdfs 之前，就生成好对应的文件格式。
 - a. 指定 csv 格式时，分批结果并将对应的 header、分隔符(只有 csv 支持指定分隔符)的处理放在这里
 - b. 指定 xlsx 格式时，分批处理结果并将结果遍历放到单元格中
3. 具体的行为变更
 - a. 在提交异步查询的 api 中，增加 **include_header** 字段，默认为 **false**。此字段指定在生成的 hdfs 文件中是否包含表头
 - b. 保持向前兼容的能力，原先下载异步查询结果的老代码保留。
 - c. 异步查询结果下载时，直接将 HDFS 上存储好的格式文件流式下载

d. 在下载异步查询结果的 api 中，将 include_header 字段删除。根据 在提交异步查询时的 include_header 字段 判断下载结果文件中是否要含有表头。（这里需要向前兼容，手册说明升级后的下载异步查询结果 api 请求不需要此字段）

4. 流程变化



异步查询结果下载时，直接流式下载即可



四、测试建议

1. 向前兼容能力
2. 数据中带有特殊字符
3. Csv 格式指定分隔符能力

Dorris Zhang 张睿轩 4936