实证分析

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 来源 |  | 环境 | 简述 | 错误 |
| Spark-11576 | Mixup column with Self join | Spark 2.0 | 当一个表同时参与多次join操作但不重新命名时（子查询和外查询）,spark会将列名弄混导致结果错误 | Wrong logical plan |
| Spark-17685 | WholeStageCodegenExec throws IndexOutOfBoundsException | Spark2.0 | 1.sort-merge-Join，  2.inner join的条件为两个  3．执行count语句  报错数组越界 | java.lang.IndexOutOfBoundsException |
|  | NOT IN subquery with null column value may return wrong results | Spark2.0 | Not in 语句 包含null 值时会返回错误的结果 | Wrong results |
|  | SortMergeJoin with skewed data may cause JVM crashed | Spark2.0 | 当小表Join大表采用sortMergeJoin时，大表若有倾斜数据可能会引起内存溢出，jvm崩溃 | java.lang.OutOfMemoryError: GC overhead limit exceeded |
| Paper：Workload Characterization and Optimization of TPC-H Queries on Apache Spark | Failure occurs in TPC-H Query 5 | Spark1.5 | The failure occurs by by calling native snappy compression library when using a single JVM with many worker threads | No specific explanation in paper |
| Spark submit 2016：  Spark SQL 2.0 Experiences Using TPC-DS | Some failures occurs in TPC-DS test for spark2.0 | Spark2.0 | 1.bad physical plan: Q14,Q16,Q23,Q95 never finish in 5+ hour  2.YARN container JVM exits with code 134: Q84 | 1.bad physical plan  2.not clear |

具体分析

1. eg

|  |
| --- |
| **val** df1 = *Seq*((1, 2), (3, 1)).toDF(**"col1"**, **"col2"**) **val** df2 = *Seq*((1, 2), (3, 1)).toDF(**"col1"**, **"col3"**) **val** df3 = df1.join(df2, df1(**"col1"**) === df2(**"col1"**)).select(df1(**"col1"**), **$"col3"**) **val** result0 = df3.join(df1, df3(**"col3"**) === df1(**"col1"**)) result0.show() |

结果为 null 正确答案为（3 1 1 2）

原因：由于spark sql的lazy操作，这时虽然显示的用df3存储了中间结果，但最终只有在.show()调用时，整个计划才会被执行，然而df1前后两次参与join，逻辑计划在列名绑定时出现了错误。

|  |
| --- |
| *Join Inner, (col3#16 = col1#5) //col1#5 should be col1#49*  *:- Project [col1#5, col3#16]*  *: +- Join Inner, (col1#5 = col1#15)*  *: :- Project [\_1#2 AS col1#5, \_2#3 AS col2#6]*  *: : +- LocalRelation [\_1#2, \_2#3]*  *: +- Project [\_1#12 AS col1#15, \_2#13 AS col3#16]*  *: +- LocalRelation [\_1#12, \_2#13]*  *+- Project [\_1#2 AS col1#49, \_2#3 AS col2#50]*  *+- LocalRelation [\_1#2, \_2#3]* |

以上可以看到，逻辑计划中的列名被弄混淆了。

1. eg

|  |
| --- |
| **val** tab1 = *Seq*((1,1,**"str"**),(2,2,**"str"**)).toDF(**"int"**,**"int2"**,**"str"**) **val** tab2 = *Seq*((1,1,**"str"**),(2,3,**"str"**)).toDF(**"int"**,**"int2"**,**"str"**) tab1.createOrReplaceTempView(**"tab1"**) tab2.createOrReplaceTempView(**"tab2"**) **val** st =  **"""  |SELECT  | count(\*)  |FROM  | (  | SELECT t1.int, t2.int2  | FROM (SELECT \* FROM tab1 LIMIT 1310720) t1  | INNER JOIN (SELECT \* FROM tab2 LIMIT 1310721) t2  | ON (t1.int=t2.int AND t1.int2=t2.int2)  | ) t  """**.stripMargin **val** result = spark.sql(st) *println*(result.queryExecution) result.show() |

Limit 1310720是为了使计划执行sortMergeJoin操作，小于否则会执行broadcast join

解释 <https://github.com/apache/spark/pull/15259>

1. eg

|  |
| --- |
| *Seq*[java.lang.Integer](2).toDF(**"a1"**).createOrReplaceTempView(**"t1"**) *Seq*[java.lang.Integer](1,**null**).toDF(**"a2"**).createOrReplaceTempView(**"t2"**) spark.sql(**"select \* from t1"**).show() spark.sql(**"select \* from t2"**).show() spark.sql(**"select \* from t1 where a1 not in (select a2 from t2)"**).show |

返回结果为空，正确答案为2

原因为spark 处理not in 子句时对null处理不对

类似spark-19017

1. eg

|  |
| --- |
| **val** sqltext3 = **"select pagerank,sourceipaddr,url,adrevenue "** +  **"from rankings "** + //rankings is small table  **"INNER JOIN uservisits "** + // uservisits is big and skewed table  **"ON url=destinationURL "** +  **"order by adrevenue desc limit 100"** |

可能会引起oom错误

可能原因：虽然从物理计划上来看，join的顺序没有区别，但是在spark实现时是有区别的，sort-merge-join实现：先将两表都进行排序，Spark将前一个表作为驱动表（streamedIter），后一个表作为缓冲表（bufferedIter），即我们流式遍历驱动表中的每一条记录，在缓冲表表中寻找相应匹配的记录，并将记录放入匹配表（bufferedMatches）中。所以当我们把大表作为缓冲表时，找到的匹配记录会很多，如果此时大表存在严重数据倾斜，匹配表占用内存也会相应变多，我们在查询相关key时会发生OutOfMemory错误。

解决方法有：1.增大executor memory的值 2.将rankings和uservisits join 顺序对调