Hive on Spark EXPLAIN statement

In Hive, command EXPLAIN can be used to show the execution plan of a query. The language manual has lots of good information. For Hive on Spark, this command itself is not changed. It behaves the same as before. It still shows the dependency graph, and plans for each stage. However, if the query engine (hive.execution.engine) is set to “spark”, it shows the execution plan with the Spark query engine, instead of the default (“mr”) MapReduce query engine.

Dependency Graph

Dependency graph shows the dependency relationship among stages. For Hive on Spark, there are Spark stages instead of Map Reduce stages. There is no difference for other stages, for example, Move stage, Stats-Aggr stage, etc.. For most queries, there is just one Spark stage since many map and reduce works can be done in one Spark work. Therefore, for a same query, with Hive on Spark, there may be less number of stages. For some queries, there are multiple Spark stages, for example, queries with map join, skew join, etc..

One thing should be pointed out that here a stage means a Hive stage. It is very different from the stage concept in Spark. A Hive stage could correspond to multiple stages in Spark. In Spark, a stage usually means a group of tasks that can be processed in one executor. In Hive, a stage contains a list of operations that can be processed in one job.

Spark Stage Plan

The plans for each stage are shown by command EXPLAIN, besides dependency graph. For Hive on Spark, the Spark stage is new. It replaces the Map Reduce stage for Hive on MapReduce. The Spark stage shows the Spark work graph, which is a DAG (directed acyclic graph). It contains:

- DAG name, the name of the Spark work DAG;
- Edges, that shows the dependency relationship among works in this DAG;
- Vertices, that shows the operator tree of each work.

For each individual operator tree, there is no change for Hive on Spark. The difference is dependency graph. For MapReduce, you can’t have a reducer without a mapper. For Spark, that’s not a problem. Therefore, Hive on Spark can optimize the plan and get rid of those mappers not needed.

The edge information is new for Hive on Spark. There is no such information for MapReduce. Different edge type indicates different shuffle requirement. For example,

PARTITION-LEVEL-SORT means that rows should be sorted on partition level during shuffling.

Sample Query Plans

The following is some sample Hive on Spark query plans. These are just samples. There are more optimization settings and plans which are not covered here.

- Common Join

Here is a sample Hive on Spark plan for a join query:

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
            Spark
            Edges:
            Reducer 2 <- Map 1 (PARTITION-LEVEL SORT, 1), Map 4 (PARTITION-LEVEL SORT, 1)
            Reducer 3 <- Reducer 2 (GROUP, 1)
            DagName: user_20150212135050_e044347f-39d0-46f7-a2da-12b9e727bdea:10
            Vertices:
            Map 1
            Map Operator Tree:
            TableScan
           .........
            Map 4
            Map Operator Tree:
            TableScan
           .........
            Reducer 2
            Reduce Operator Tree:
            Join Operator
           .........
            Reduce Output Operator
           .........
            Reducer 3
            Reduce Operator Tree:
            Group By Operator
           .........

  Stage: Stage-0
            Fetch Operator
            limit: -1
            Processor Tree:
            ListSink
```

It is a common join, and there is just one Spark stage. The Fetch stage is the same as in MapReduce. In the Spark stage, there are two map works (Map 1 and Map 4). Reducer 2 depends on these two map works, and Reducer 3 depends on Reducer 2.

In MapReduce, a reducer can not depend on another reducer. So it has more stages:

```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 depends on stages: Stage-2

STAGE PLANS:
  Stage: Stage-1
            Map Reduce
            Map Operator Tree:
            TableScan
           .........
            TableScan
           .........
            Reduce Operator Tree:
            Join Operator
           .........
            Group By Operator
           .........

  Stage: Stage-2
            Map Reduce
            Map Operator Tree:
            TableScan
            Reduce Output Operator
           .........

  Stage: Stage-0
            Fetch Operator
            limit: -1
            Processor Tree:
            ListSink
```

There are two Map Reduce stages. So for Hive on Spark, there is one job, while there are two jobs for MapReduce.

- Map Join

If we set hive.auto.convert.join to true, the plan is:

```
STAGE DEPENDENCIES:
  Stage-2 is a root stage
  Stage-1 depends on stages: Stage-2
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-2
```

```
        Spark
        DagName: user_20150212132222_446d5ea7-acc5-46e0-b3f4-0752821d2d81:5
        Vertices:
        Map 3
        Map Operator Tree:
        TableScan
        .....
        Local Work:
        Map Reduce Local Work

Stage: Stage-1
        Spark
        Edges:
        Reduce 2 <- Map 1 (GROUP, 1)
        DagName: user_20150212132222_446d5ea7-acc5-46e0-b3f4-0752821d2d81:4
        Vertices:
        Map 1
        Map Operator Tree:
        TableScan
        .....
                Map Join Operator
        .....
        Local Work:
        Map Reduce Local Work
        Reducer 2
        Reduce Operator Tree:
        Group By Operator
        .....

Stage: Stage-0
        Fetch Operator
        limit: -1
        Processor Tree:
        ListSink
```

Now, it uses map join. There are two Spark stages. The first Spark stage (Stage-2) has one map operator. The second Spark stage (Stage-1) contains one map operator and one reducer operator.

The following shows the reducer work depends on the map work. There is a GROUP BY operator in the map work 1. The reducer work has just one reducer.

```
Reducer 2 <- Map 1 (GROUP, 1)
```

For map join, Hive on Spark has at least two stages by design. The first stage loads the small table and processes it, then writes the output to some files in HDFS. The rest stages load the files and do map join.

- Bucket Map Join

For bucket map join, the query plan is the same as map join. However, if you use command “EXPLAIN EXTENDED”, it will show something like:

BucketMapJoin: true

and

Bucket Mapjoin Context:

If the tables are bucketed, and hive.optimize.bucketmapjoin is set to true, the extended plan is:

```
ABSTRACT SYNTAX TREE:

TOK_QUERY
   .....

STAGE DEPENDENCIES:
  Stage-2 is a root stage
  Stage-1 depends on stages: Stage-2
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-2
            Spark
            DagName: user_20150213093535_06b94571-5d6a-42c1-a4ab-5f9863edf812:13
            Vertices:
            Map 3
            Map Operator Tree:
            TableScan
            .....
            Local Work:
            Map Reduce Local Work
            Bucket Mapjoin Context:
            .....

  Stage: Stage-1
            Spark
            Edges:
            Reducer 2 <- Map 1 (SORT, 1)
            DagName: user_20150213093535_06b94571-5d6a-42c1-a4ab-5f9863edf812:12
            Vertices:
            Map 1
            Map Operator Tree:
            TableScan
            .....
                        BucketMapJoin: true
            .....
            Local Work:
            Map Reduce Local Work
            Bucket Mapjoin Context:
            .....
            Reducer 2
            Needs Tagging: false
            Reduce Operator Tree:
            Select Operator
```

.....

Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink

- Sorted Merge Bucket Map Join

If hive.auto.convert.sortmerge.join is set to true, optimizer will check if a query can be converted to sorted merge bucket (SMB) join, if so, the plan looks like:

STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-0 depends on stages: Stage-1

STAGE PLANS:
Stage: Stage-1
Spark
DagName: user_20150213102222_9bf6b872-a690-48bc-b222-adf102f209e6:4
Vertices:
Map 1
Map Operator Tree:
TableScan
alias: a
Statistics: Num rows: 2 Data size: 208 Basic stats: COMPLETE Column stats: NONE
Filter Operator
.....
Sorted Merge Bucket Map Join Operator
.....

Stage: Stage-0
Fetch Operator
limit: -1
Processor Tree:
ListSink

- Skew Join

If a table is skewed, we set can set hive.optimize.skewjoin to true, and hive.skewjoin.key to the row count for a skewed join key, a join on the skewed key will be converted to skew join. The plan looks like:

STAGE DEPENDENCIES:
Stage-1 is a root stage
Stage-3 depends on stages: Stage-1 , consists of Stage-4
Stage-4
Stage-2 depends on stages: Stage-4
Stage-0 depends on stages: Stage-2

```
STAGE PLANS:
  Stage: Stage-1
        Spark
        Edges:
        Reducer 2 <- Map 1 (PARTITION-LEVEL SORT, 1), Map 3 (PARTITION-LEVEL SORT, 1)
        DagName: user_20150213104545_a99fe625-7d9b-483d-a148-e6c78f179746:11
        Vertices:
        Map 1
        Map Operator Tree:
        TableScan
        ..................
        Map 3
        Map Operator Tree:
        TableScan
        ..................
        Reducer 2
        Reduce Operator Tree:
        Join Operator
        condition map:
                Inner Join 0 to 1
        handleSkewJoin: true
        ..................

  Stage: Stage-3
        Conditional Operator

  Stage: Stage-4
        Spark
        DagName: user_20150213104545_a99fe625-7d9b-483d-a148-e6c78f179746:13
        Vertices:
        Map 5
        Map Operator Tree:
        TableScan
        Spark HashTable Sink Operator
                keys:
                0 reducesinkkey0 (type: string)
                1 reducesinkkey0 (type: string)
        Local Work:
        Map Reduce Local Work

  Stage: Stage-2
        Spark
        DagName: user_20150213104545_a99fe625-7d9b-483d-a148-e6c78f179746:12
        Vertices:
        Map 4
        Map Operator Tree:
        TableScan
        Map Join Operator
                ..................
        Local Work:
        Map Reduce Local Work

  Stage: Stage-0
        Fetch Operator
        limit: -1
        Processor Tree:
```

ListSink