# Hive on Spark: Reduce-Side Join

## Background

Reduce-side join in MapReduce is one of the two basic types of data joins (the other is a map-join). In a nutshell, the basis for Reduce-side join is to make the Mapper’s output key to be the join column, and the output value to be the rest of the data row. This leverages the shuffle functionality of MapReduce framework, which shuffles rows with the same key to the same reducer. The joining then happens in the reducer.

Quick example:

Orders (ProductId, Quantity, Month)

{1, 500, Jan}

{1, 400, Feb}

{2, 300, Jan}

Products (ProductId, Name)

{1, Hamburgers}

{2, Hot Dogs}

select \* from orders join products on orders.productId == products.productId.

In this query, “productId” is the join column, and will be outputted by mappers as the output key. MapReduce shuffles all rows with productId = 1 to the first reducer, and rows with productId = 2 to the second one. The reducers will create the following output:

Reducer(ProductId=1)

* Input

Orders: {1, 500, Jan}

Products: {1, Hamburgers}

Orders: {1, 400, Feb}

* Output

{1, 400, Feb, Hamburgers}, {1, 500, Jan, Hamburgers}

Reducer(ProductId=2)

* Input

Orders: {2, 300, Jan}

Products: {2, Hot Dogs}

* Output

{2, 300, Jan, Hot Dogs}

## Hive Reduce Side Join

In Hive implementation, the output key of the Mapper is a composite key: {Join Column, Table Alias}.

The Join Column is as described in the earlier section (value of join column). Hive configures the MapReduce job so that only this portion of the output key determines which reducer to go to. So far, it follows the general algorithm described above.

The Table Alias is a byte describing the origin table. The reason it’s in the composite key (but ignored by partitioner in determining reducer) is to take advantage of another feature of MapReduce, which sorts the input to each reducer by keys.

In the above example, say table alias of Orders is 0, table alias of Products is 1. The reducer taking productId = 1 will first get data of productId = 1 from the Orders table, followed by all data with productId = 1 from the Products table. This allows the reducer join logic to be greatly simplified.

## Hive on Spark Design

Hive on Spark involves generation of a SparkPlan, which chains together MapTran, ReduceTran, and UnionTran, which are sent to Spark cluster and invoked in sequence on the input RDD’s. The generation of a SparkPlan is done by reading a SparkWork, which wraps a tree of MapWork, ReduceWork, and UnionWork output given from Hive query compilation.

The proposal is in a given SparkWork to find Reduce Join by identifying the following pattern. The parents in this case are two MapWorks.

ReduceWork

(JoinOperator)

MapWork1

MapWork2

Then, we will create the following SparkTran by inserting a UnionTran before the ReduceTran. The UnionTran will invoke the union transformation on the two output RDD’s of the parent trans (in this case MapTrans).

UnionTran

MapTran2

MapTran1

ReduceTran

(JoinOperator)

## Spark Shuffling

Currently, SparkTran automatically injects SparkShufflers between MapTran and ReduceTran, which emulate the MapReduce shuffle functionality via invoking either of groupBy or sortBy RDD transformation. As mentioned in the [Hive-on-Spark design](https://cwiki.apache.org/confluence/display/Hive/Hive+on+Spark), there is no equivalent as of this writing of both group+sort as MapReduce shuffle provides, which will partition an RDD according to a partition scheme and then sort the keys within each partition.

This is a requirement on Hive side, as we require the key to be partitioned by JoinColumn portion of the composite key, and the sort by the entire composite key {JoinColumn, TableAlias}, as described in section “Hive Reduce Side Join”.

This work to make a group+sort operator is tracked in [SPARK-2978](https://issues.apache.org/jira/browse/SPARK-2978).

We will use this group+sort operator with a custom partitioner, which will partition Hive composite key based only on the first {JoinColumn} portion. This operator will then give keys to Spark’s ReduceTran in sorted manner.