# Hive on Spark: Map-Side Join

## MapReduce Background:

MapJoin is an optimization of a regular Reduce-side join of Hive. In an n-way join, it does the following:

1. Determine if there are n-1 small tables that can fit into memory.
2. Load small table(s) into memory, constructing a map of join-key to value.
3. Scan through big-table and for each record, lookup the join-key in the hash table to generate the corresponding record.

In Hive on Map-Reduce, this is done by the following two classes:

1. HashTableSink, which dumps the small table into a file. Done in local job
2. HashTableLoader to load the file and populate a local hash-map. Done distributed (in a mapper that is reading records from big table).

## MapReduce Mode Query Optimizations

In query planning phase, physical-optimizers convert original reduce-side join into map-side join using the following algorithm:

1. Pick small table’s map operator-tree. Convert ReduceSink into HashTableSink. Convert from MapRedWork to local job. (HashTableSink doesn’t run distributed).
2. Merge the big table’s map operator-tree with the join operator-tree of the reducer. Put these into the map stage of a second MapReduceWork. The join operator becomes a MapJoin operator, which loads the file to memory hashmap via HashTableLoader.

It’s best to see by example. Note on Hive on MapReduce syntax, note each MapRedWork represents either one MapReduce job, or one local job.

* Query: SELECT \* from src1 JOIN src2 ON (src1.key = src2.key);

1. Original Plan (Reduce Side Join):

MapRedWork

* MapWork
  + Src1-OperatorTree: TableScan -> Filter -> ReduceSink
  + Src2-OperatorTree: TableScan -> Filter -> ReduceSink
* ReduceWork
  + JoinOperator -> Filter -> FileSink (output)

1. Optimized Plan: Note that the first MapRedWork has become a local work with HashTableSink. The second MapReduceWork is a map-only job, which reads the large table and feeds to a MapJoinOperator, which loads small table file into an in-memory hashmap using HashTableLoader.

MapRedWork1

* LocalWork
  + Src1-OperatorTree: TableScan -> Filter -> HashTableSink

MapRedWork2 (child)

* MapWork
  + Src2-OperatorTree: TableScan -> Filter -> MapJoinOperator (HashTableLoader) -> Filter -> FileSink (Output)

In another example, we see some other optimizations for merging nested join statement into multiple map joins:

* Query2: SELECT src1 JOIN src2 ON (src1.key = src2.key) JOIN src3 ON (src1.key + src2.key = src3.key);

1. Original plan

MapRedWork1

* MapWork
  + Src1-OperatorTree: TableScan -> Filter -> ReduceSink
  + Src2-OperatorTree: TableScan -> Filter -> ReduceSink
* ReduceWork
  + JoinOperator -> Filter -> FileSink (intermed-output)

MapRedWork2

* MapWork
  + MR1Output- OperatorTree: TableScan -> ReduceSink
  + Src3- OperatorTree: TableScan -> Filter -> ReduceSink
* ReduceWork
  + JoinOperator -> Filter -> FileSink (final-output)

1. Optimized plan. Note that two map-side joins would have been originally generated from the two reduce-side joins. An additional step combines them into one.

MapRedWork1

* MapLocalWork
  + Src1-OperatorTree: TableScan -> Filter --\
  + Src2-OperatorTree: TableScan -> Filter ---> HashTableSink

MapRedWork2

* MapWork
  + Src3- OperatorTree: TableScan -> Filter -> MapJoinOperator -> ReduceSink

The physical optimizers that are at play are: CommonJoinResolver, and MapJoinResolver. A combination of these two make the transformation above.

There are many other logic of these optimizers, such as make conditional tasks of various map-join options of which one is chosen at runtime, that are not going to be considered for now for scoping purpose.

## Tez Background:

TezCompiler runs similar physical optimizations for map-join, but there are key differences.

The first step of reading the small table(s) don’t involve dumping to file, so instead of a local task with a HashTableSink, this part remains as a distributed mapper task with a reduce-sink for this operation.

Hive-on-Tez uses a Tez feature called a broadcast edge to feed this edge into the second mapper (for the big-table), which has a Tez-specific HashLoader to read from the edge and populate the hashmap.

For example:

* Query1: SELECT src1 JOIN src2 ON (src1.key = src2.key);

Optimized Plan: Note that ReduceSink is kept, and the first work remains a distributed MapReduce work.

This optimizer logic is in the class ConvertJoinMapJoin.

TezWork

* MapWork1
  + Src1-OperatorTree: TableScan -> Filter -> ReduceSink
* MapWork2 (child)
  + Src2-OperatorTree: TableScan -> Filter -> MapJoinOperator (TezHashTableLoader) -> Filter -> FileSink (Output)

TezWork, which is similar to SparkWork, is different than MapReduceWork in following ways:

* Spark/TezWork represents entire chain of work, can have multiple MapWork, ReduceWork all in a chain.
* MapReduceWork has one MapWork linked to one ReduceWork, or alternatively one LocalWork. Thus optimizers need to create a new MapReduceWork in mapjoin, this is not the case in Spark/TezWork.
* In Spark/TezWork, each table read in a given join is a separate MapWork
* In MapReduceWork, each table read in a given join is part of the same MapWork, they just are different MapOperators in that MapWork.