Hive on Spark: Bucket Map Join

Map reduce background

Bucket Map Join is a join type and it applies for the condition when total table/partition size is too big for map join. It's implementation depends on the prior map join implementation (mapjon hint), therefore it only works when the map join hint is enabled. However, in the recent Hive versions (since Hive-0.11), the Bucket Map Join is turned off by default and all joins are handled by regular map join.

Conditions for Bucket Map Join:

1. turn on the optimization flag: *set hive.optimize.bucketmapjoin = true;*
2. enable mapjoin hint: *set hive.ignore.mapjoin.hint=false;*
3. all join tables are bucketed, and each small table's bucket number can be divided by big table's bucket number
4. bucket columns == join columns

Example:

```
SELECT /*+ MAPJOIN(b) */ a.key, a.value
FROM a join b on a.key = b.key
```

If the table being joined are bucketed, and the buckets are multiple of each other, the buckets can be joined with each other. If table A has 8 buckets and table B has 4 buckets, the above join can be done on the mapper only. Instead of fetching B completely for each mapper of A, only the required buckets are fetched.

Implementation:

There are 2 phase of Optimization in hive, logical optimization and physical optimization, both are rule based optimizations.

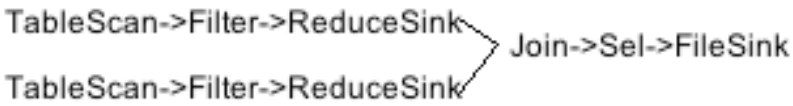
- Logical Optimization

  In the logical optimization, mapjoin optimization was followed by bucketmapjoin optimization, bucketmapjoin optimization takes the MapJoin operator tree as input and convert it into BucketMapJoin, so a hinted query would be first transformed into a mapjoin and then a

bucketmapjoin.

1. Operator tree is changed from JOIN operator to MAPJOIN operator after map join optimization

Operator tree before doing map join optimization:



Operator tree after doing map join optimization:

2. Bucketmapjoin optimization does not change operator tree, but setup bigtable/smalltable bucket mapping.

Implementation is done in *BucketMapJoinOptimizer & BucketMappingProc* classes

- Physical Optimization

  In the physical optimization, the hive.auto.convert.join was checked and MapJoinResolver was used and just convert a reduce join into a MapJoin. No further BucketMapJoin Optimization rules in this phase.

Details:

In the task generation phase, if a MAPJOIN operator is detected, the *MapJoinResolver* takes action by replacing the MapJoinOperator to HashTableSink operator and generating local work.

```
MapRedWork1
   LocalWork src1-OperatorTree: TableScan -> Filter -> HashTableSink
MapRedWork2
   MapWork src2-OperatorTree: TableScan -> Filter -> MapJoinOperator
(HashTableLoader) -> Filter -> FileSink (output)
```

MapJoin operator is processed by loading the smaller table into an in-memory hash map and matchings keys with the larger table as they are streamed through. The following works are involved:

- Local work:
  - read records via standard table scan (including filters and projections) from source on local machine
  - build hashtable in memory
  - write hashtable to local disk
  - upload hashtable to dfs
  - add hashtable to distributed cache
- Map task
  - read hashtable from local disk (distributed cache) into memory
  - match records' keys against hashtable
  - combine matches and write to output

The difference between bucket map join and regular map join is that: multiple hashtables are created from the small table buckets. When the larger table are streamed through, only the corresponding hashtable is loaded into memory to do a map join with the larger table bucket.

Some Map Reduce background on SMB Join

SMB join has been controlled by different settings.

Conditions for Bucket Map Join:

1. set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;
   set hive.optimize.bucketmapjoin = true;
   set hive.optimize.bucketmapjoin.sortedmerge = true
   set hive.ignore.mapjoin.hint = false;

Both hive.optimize.bucketmapjoin and hive.optimize.bucketmapjoin.sortedmerge depends on map join hint. If map join hint is disabled, the optimization for two settings will not be triggered.

2. all join tables are bucketed and sorted, and each small table’s bucket number can be divided by big table’s bucket number
3. bucket columns == join columns==sort columns

Or use the following settings

set hive.auto.convert.sortmerge.join = true; -a join was converted to a sort-merge join
set hive.auto.convert.sortmerge.join.to.mapjoin = true: -when hive.auto.convert.sortmerge.join is set to true, and a join was converted to a sort-merge join. This parameter decides whether each table should be tried as a big table, and effectively a map-join should be tried.

The above two parameters does not depend on map join hint. Therefore, SMB join work for both map join hint enabled and disabled.

When the SMB join optimization is enabled, the optimization is done in the logical optimization phase by converting join operator to SMB join operator. During this conversion, it first converts join to bucket map join by setting up the bucket mapping info and then further convert bucket map join to SMB join.

From above analysis, it shows possibility that SMB join and BucketMap join design/impl share the same logic during the time of generating operator tree and doing logical optimization. It would be nice if bucketmap join has a setting like hive.auto.convert.bucketmapjoin which does the bucket map join automatically without depending on map join hint. In this case, the BucketMapJoin and SMB join would share the same logic except that if the bucket join column is sorted, then use merge sort join to handle, otherwise use map join to handle.

Tez background

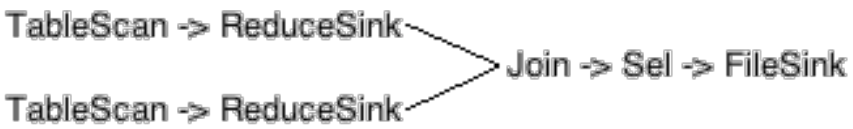
Hive on Tez uses a different config parameter to enable bucket map join optimization. In addition, the bucket map join on Tez does not depend on the map join hint and a join can be converted to bucket map join automatically if the following criteria are met:

1. set hive.convert.join.bucket.mapjoin.tez = true;
2. all join tables are bucketed, and each small table's bucket number can be divided by big table's bucket number
3. bucket columns == join columns

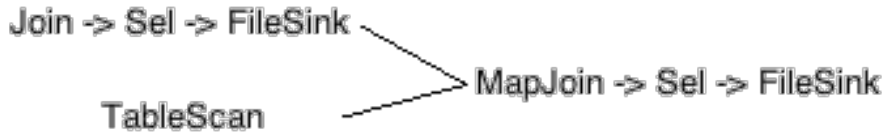
In the optimize operator plan step, the ConvertJoinMapJoin processor is triggered for the join operator. It checks if a join can be converted to a

BucketMap join, if so, a MapJoin operator is generated to replace the Join operator in the original operator tree and also the bucket mapping information are attached to the MapJoin operator. In the meanwhile, the reducesink operator of the big table are removed.

Operator tree before doing operator plan optimization



Operator tree after doing operator plan optimization



In the task tree generation phase, two Map works are generated which is similar to the map join case. The first map work processes the small table and the second map work processes the big table. Instead of using a broadcast edge to connect the two map works in regular map join, tez uses a custom edge to connect the first map work to the second map work if the map join is a bucket map join.