Understanding Hive joins in explain plan output

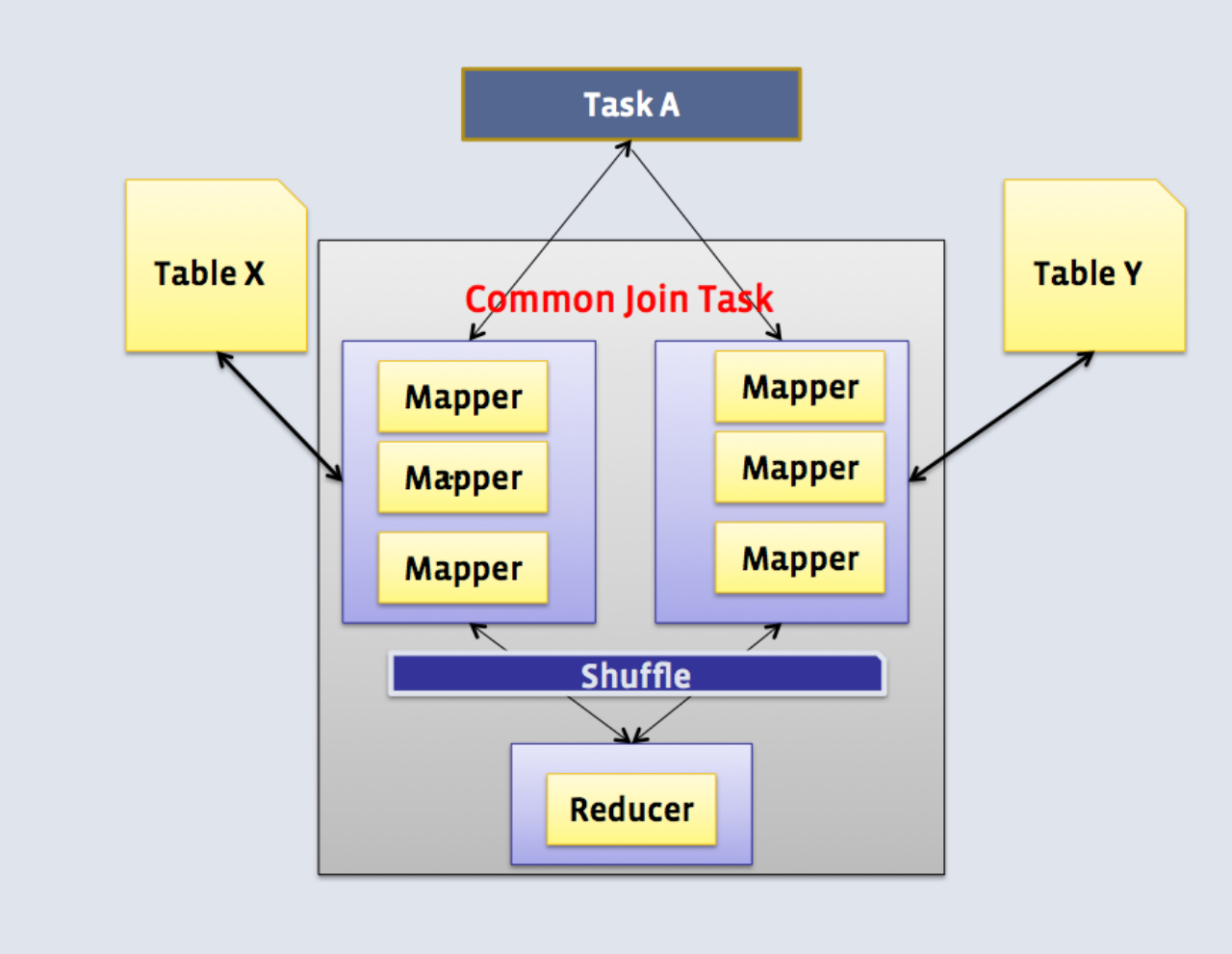
Hive is trying to embrace CBO(cost based optimizer) in latest versions, and Join is one major part of it. Understanding join best practices and use cases is one key factor of Hive performance tunning. This article will explain each kind of join and also use explain plan output to show the difference.
Note: All below tests are based on Hive 0.13.

1. Shuffle Join(Common Join).

How:

The shuffle join is the default option and it includes a map stage and a reduce stage.

- Mapper: reads the tables and output the join key-value pairs into an intermediate file.
- Shuffle: these pairs are sorts and merged.
- Reducer: gets the sorted data and does the join.



Use case:

It works for any table size.
Especially when other join types cannot be used, for example, full outer join.

Cons:

Most resource intensive since shuffle is an expensive operation.

Example:

```
hive> explain select a.* from passwords a, passwords2 b where a.col0=b.col1;
OK
STAGE DEPENDENCIES:
  Stage-5 is a root stage , consists of Stage-1
  Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-5
    Conditional Operator

  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: b
            Statistics: Num rows: 9961472 Data size: 477102080 Basic stats:
COMPLETE Column stats: NONE
            Reduce Output Operator
              key expressions: col1 (type: string)
              sort order: +
              Map-reduce partition columns: col1 (type: string)
              Statistics: Num rows: 9961472 Data size: 477102080 Basic stats:
COMPLETE Column stats: NONE
              value expressions: col1 (type: string)
          TableScan
            alias: a
            Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
            Reduce Output Operator
              key expressions: col0 (type: string)
              sort order: +
              Map-reduce partition columns: col0 (type: string)
              Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
              value expressions: col0 (type: string), col1 (type: string), col2
(type: string), col3 (type: string), col4 (type: string), col5 (type: string),
col6 (type: string)
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1} {VALUE._col2} {VALUE._col3}
{VALUE._col4} {VALUE._col5} {VALUE._col6}
            1 {VALUE._col1}
          outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6,
_col10
          Statistics: Num rows: 10960295 Data size: 524940416 Basic stats:
COMPLETE Column stats: NONE
          Filter Operator
            predicate: (_col0 = _col10) (type: boolean)
            Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
            Select Operator
              expressions: _col0 (type: string), _col1 (type: string), _col2
(type: string), _col3 (type: string), _col4 (type: string), _col5 (type:
```

```
string), _col6 (type: string)
              outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6
              Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
              File Output Operator
                compressed: false
                Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
                table:
                    input format: org.apache.hadoop.mapred.TextInputFormat
                    output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                    serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

  Stage: Stage-0
    Fetch Operator
      limit: -1

Time taken: 1.707 seconds, Fetched: 58 row(s)
```

Tips:

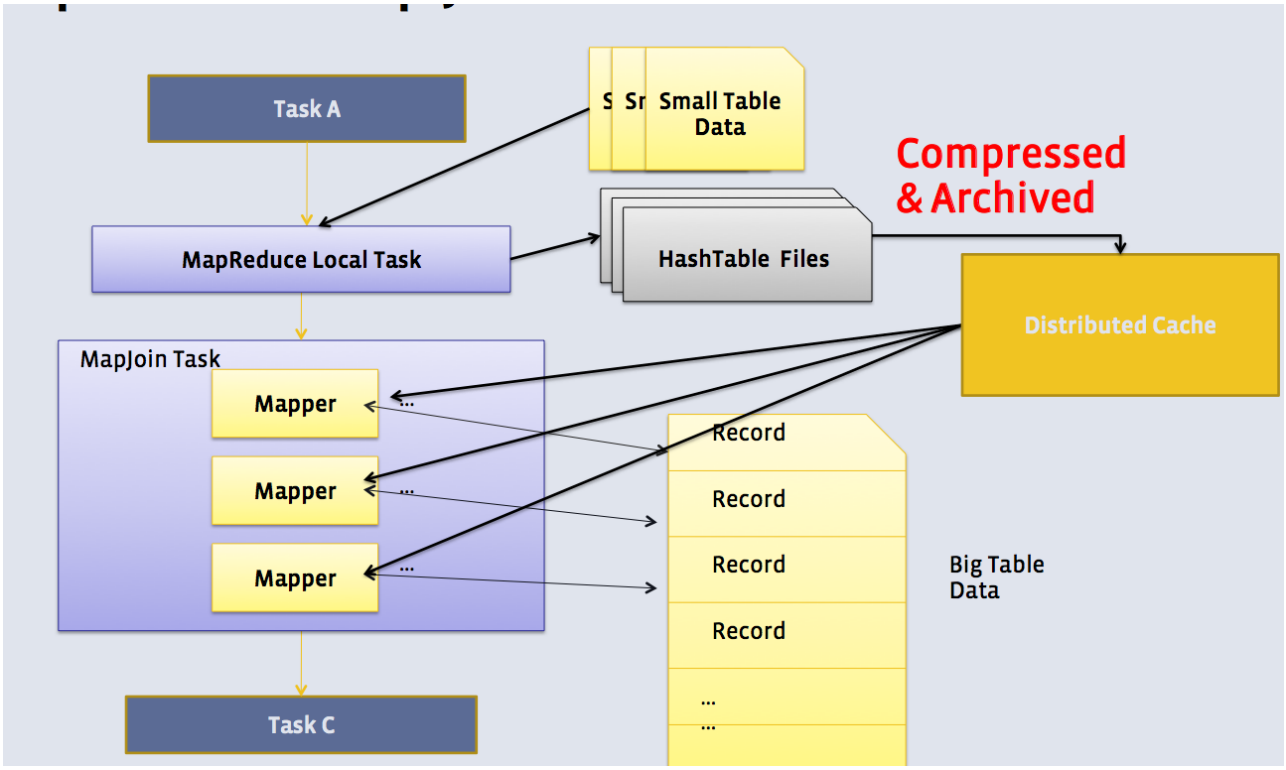
The largest table should be put on the rightmost since it should be the stream table.
However you can use hint "STREAMTABLE" to change the stream table in each map-reduce stage.

```
select /*+ STREAMTABLE(a) */ a.* from passwords a, passwords2 b, passwords3 c
where a.col0=b.col0 and b.col0=c.col0;
```

2. Map Join(Broardcast Join)

How:

If one or more tables are small enough to fit in memory, the mapper scans the large table and do the joins. No shuffle and reduce stage.



Use case:

Small table(dimension table) joins big table(fact table). It is very fast since it saves shuffle and reduce stage.

Cons:

It requires at least one table is small enough.
Right/Full outer join don't work.

Example:

Here passwords3 table is very small table while passwords table is huge.

```
hive> explain select a.* from passwords a,passwords3 b where a.col0=b.col0;
OK
STAGE DEPENDENCIES:
  Stage-4 is a root stage
  Stage-3 depends on stages: Stage-4
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-4
    Map Reduce Local Work
      Alias -> Map Local Tables:
        b
          Fetch Operator
            limit: -1
      Alias -> Map Local Operator Tree:
        b
          TableScan
            alias: b
            Statistics: Num rows: 1 Data size: 31 Basic stats: COMPLETE Column
stats: NONE
            HashTable Sink Operator
              condition expressions:
                0 {col0} {col1} {col2} {col3} {col4} {col5} {col6}
                1 {col0}
              keys:
                0 col0 (type: string)
                1 col0 (type: string)

  Stage: Stage-3
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: a
            Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
            Map Join Operator
              condition map:
                   Inner Join 0 to 1
              condition expressions:
                0 {col0} {col1} {col2} {col3} {col4} {col5} {col6}
                1 {col0}
              keys:
                0 col0 (type: string)
                1 col0 (type: string)
              outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6, _col9
              Statistics: Num rows: 10960295 Data size: 524940416 Basic stats:
```

```
COMPLETE Column stats: NONE
              Filter Operator
                predicate: (_col0 = _col9) (type: boolean)
                Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
                Select Operator
                  expressions: _col0 (type: string), _col1 (type: string), _col2
(type: string), _col3 (type: string), _col4 (type: string), _col5 (type:
string), _col6 (type: string)
                  outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6
                  Statistics: Num rows: 5480147 Data size: 262470184 Basic
stats: COMPLETE Column stats: NONE
                  File Output Operator
                    compressed: false
                    Statistics: Num rows: 5480147 Data size: 262470184 Basic
stats: COMPLETE Column stats: NONE
                    table:
                        input format: org.apache.hadoop.mapred.TextInputFormat
                        output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                        serde:
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
      Local Work:
        Map Reduce Local Work

  Stage: Stage-0
    Fetch Operator
      limit: -1

Time taken: 0.1 seconds, Fetched: 63 row(s)
```

Tips:

1. Auto convert shuffle/common join to map join.

3 parameters are related:

```
set hive.auto.convert.join=true;
set hive.auto.convert.join.noconditionaltask=true;
set hive.auto.convert.join.noconditionaltask.size=10000000;
```

Starting from Hive 0.11, hive.auto.convert.join=true by default.
You can disable this feature by setting hive.auto.convert.join=false.
When hive.auto.convert.join.noconditionaltask=true, if estimated size of small table(s) is smaller than hive.auto.convert.join.noconditionaltask.size(default 10MB), then common join can convert to map join automatically.
From above SQL plan output, we know estimated "Table b's Data Size=31" according to statistics.
If "**set hive.auto.convert.join.noconditionaltask.size = 32;**", the explain output shows map join operator:

```
Map Join Operator
```

If "**set hive.auto.convert.join.noconditionaltask.size = 31;**", then the join becomes common join operator:

```
Join Operator
```

2. Hint "MAPJOIN" can be used to force to use map join.

Before using the hint, firstly make sure below parameter is set to false(Default is true in Hive 0.13).

```
set hive.ignore.mapjoin.hint=false;
```

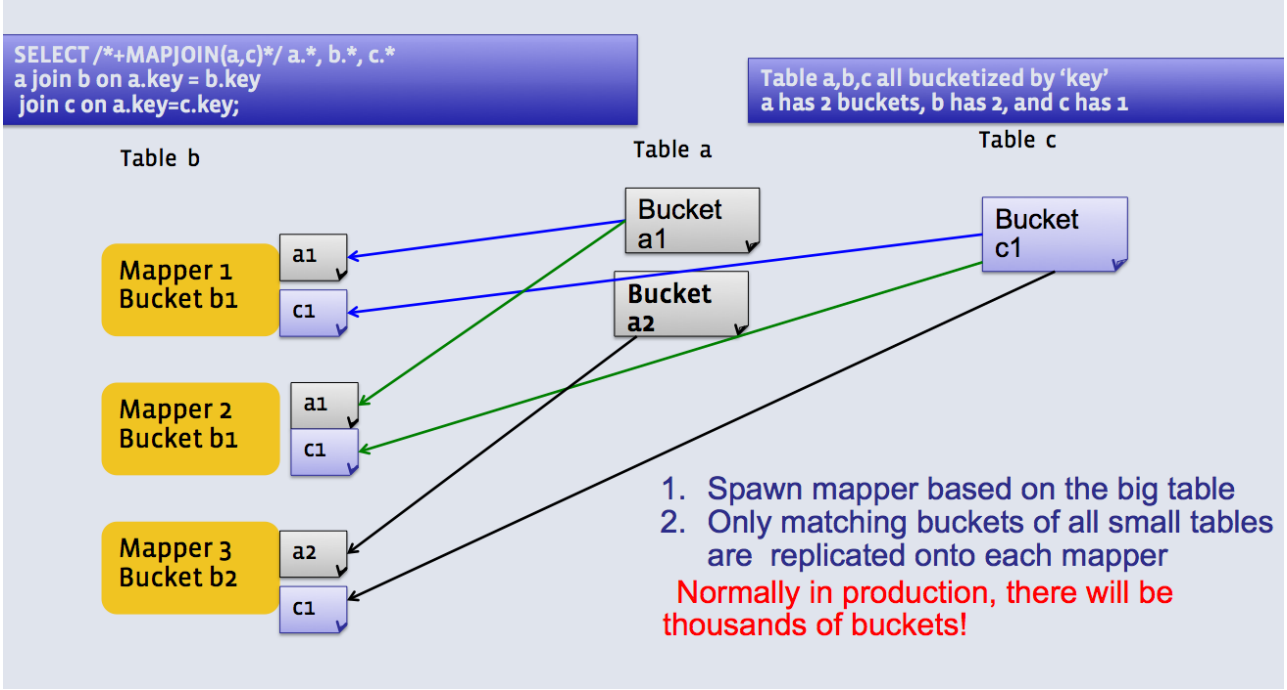
Then:

```
select /*+ MAPJOIN(a) */ a.* from passwords a, passwords2 b where
a.col0=b.col0 ;
```

3. Bucket Map Join

How:

Join is done in Mapper only. The mapper processing bucket 1 for table A will only fetch bucket 1 of table B.



Use case:

When all tables are:

- Large.
- Bucketed using the join columns.
- The number of buckets in one table is a multiple of the number of buckets in the other table.
- Not sorted.

Cons:

Tables need to be bucketed in the same way how the SQL joins, so it cannot be used for other types of SQLs.

Tips:

1. The tables need to be created bucketed on the same join columns and also data need to be

bucketed when inserting.

One way is to set "hive.enforce.bucketing=true" before inserting data.

For example:

```
create table b1(col0 string,col1 string,col2 string,col3 string,col4 string,col5
string,col6 string)
clustered by (col0) into 32 buckets;
create table b2(col0 string,col1 string,col2 string,col3 string,col4 string,col5
string,col6 string)
clustered by (col0) into 8 buckets;

set hive.enforce.bucketing = true;
From passwords insert OVERWRITE  table b1 select * limit 10000;
From passwords insert OVERWRITE  table b2 select * limit 10000;
```

2. hive.optimize.bucketmapjoin must to be set to true.

```
set hive.optimize.bucketmapjoin=true;
select /*+ MAPJOIN(b2) */ b1.* from b1,b2 where b1.col0=b2.col0 ;
```

4. Sort Merge Bucket(SMB) Map Join

How:

Join is done in Mapper only. The corresponding buckets are joined with each other at the mapper.

Use case:

When all tables are:

- Large.
- Bucketed using the join columns.
- Sorted using the join columns.
- All tables have the same number of buckets.

Cons:

Tables need to be bucketed in the same way how the SQL joins, so it cannot be used for other types of SQLs.

Partition tables might slow down.

Example:

```
hive> explain select  c1.* from c1,c2 where c1.col0=c2.col0;
OK
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: c1
            Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
            Sorted Merge Bucket Map Join Operator
              condition map:
                   Inner Join 0 to 1
              condition expressions:
                0 {col0} {col1} {col2} {col3} {col4} {col5} {col6}
                1 {col0}
              keys:
                0 col0 (type: string)
                1 col0 (type: string)
              outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6, _col9
              Filter Operator
                predicate: (_col0 = _col9) (type: boolean)
                Select Operator
                  expressions: _col0 (type: string), _col1 (type: string), _col2
(type: string), _col3 (type: string), _col4 (type: string), _col5 (type:
string), _col6 (type: string)
                  outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6
                  File Output Operator
                    compressed: false
                    table:
                        input format: org.apache.hadoop.mapred.TextInputFormat
                        output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                        serde:
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

  Stage: Stage-0
    Fetch Operator
      limit: -1

Time taken: 0.134 seconds, Fetched: 37 row(s)
```

Tips:

1. The tables need to be created bucketed and sorted on the same join columns and also data need to be bucketed when inserting.

One way is to set "hive.enforce.bucketing=true" before inserting data.

For example:

```
create table c1(col0 string,col1 string,col2 string,col3 string,col4 string,col5
string,col6 string)
```

```
clustered by (col0) sorted by (col0) into 32 buckets;
create table c2(col0 string,col1 string,col2 string,col3 string,col4 string,col5
string,col6 string)
clustered by (col0) sorted by (col0) into 32 buckets;
set hive.enforce.bucketing = true;
From passwords insert OVERWRITE  table c1 select *  order by col0;
From passwords insert OVERWRITE  table c2 select *  order by col0;
```

2. Below parameters need to set to convert SMB join to SMB map join.

```
set hive.auto.convert.sortmerge.join=true;
set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set hive.auto.convert.sortmerge.join.noconditionaltask=true;
```

3. Big table selection policy parameter

"hive.auto.convert.sortmerge.join.bigtable.selection.policy" determines which table is for only streaming.

It has 3 values:

```
org.apache.hadoop.hive.ql.optimizer.AvgPartitionSizeBasedBigTableSelectorForAuto
SMJ (default)
org.apache.hadoop.hive.ql.optimizer.LeftmostBigTableSelectorForAutoSMJ
org.apache.hadoop.hive.ql.optimizer.TableSizeBasedBigTableSelectorForAutoSMJ
```

4. Hint "MAPJOIN" can determine which table is small and should be loaded into memory.

5. Small tables are read on demand which means not holding small tables in memory.

6. Outer join is supported.

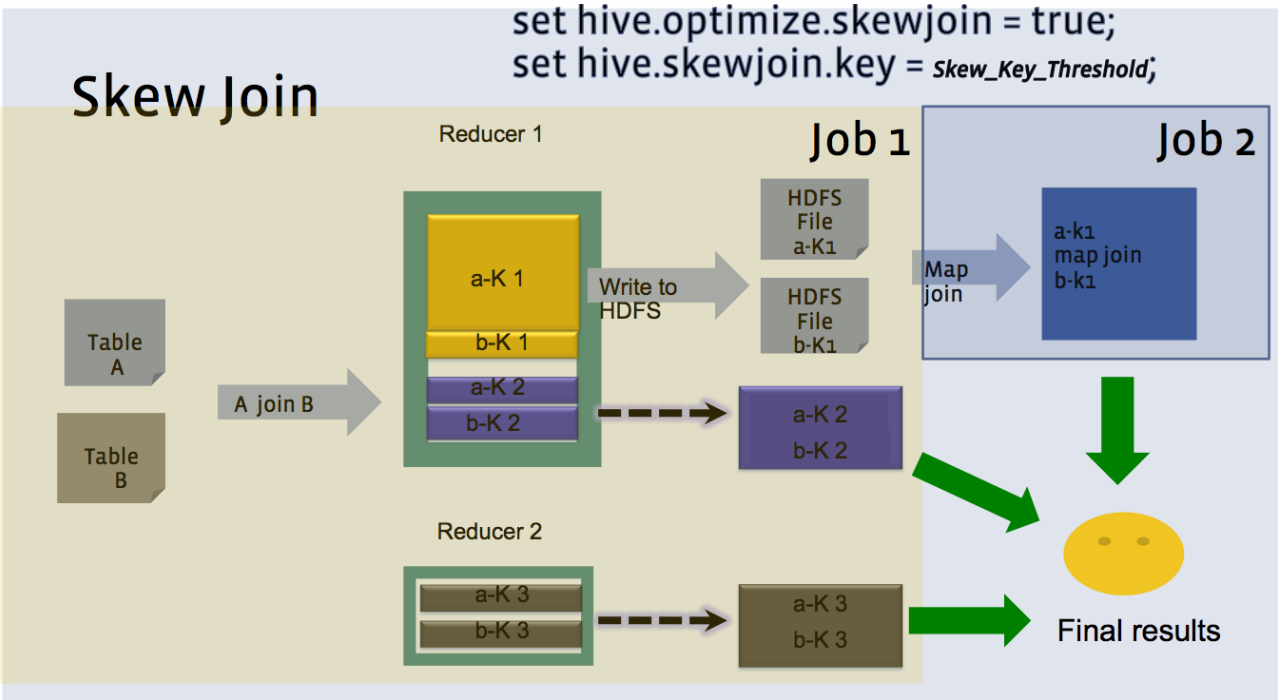
5. Skew Join

How:

If table A join B, and A has skew data "1" in joining column.
First read B and store the rows with key 1 in an in-memory hash table. Now run a set of mappers to read A and do the following:

- If it has key 1, then use the hashed version of B to compute the result.
- For all other keys, send it to a reducer which does the join. This reducer will get rows of B also from a mapper.

This way, we end up reading only B twice. The skewed keys in A are only read and processed by the Mapper, and not sent to the reducer. The rest of the keys in A go through only a single Map/Reduce.
The assumption is that B has few rows with keys which are skewed in A. So these rows can be loaded into the memory.



Use case:

One table has huge skew values on the joining column.

Cons:

One table is read twice.

Users should be aware of the skew key.

Example:

```
hive> explain select a.* from passwords a, passwords2 b where a.col0=b.col1;
OK
STAGE DEPENDENCIES:
  Stage-7 is a root stage , consists of Stage-1
  Stage-1
  Stage-4 depends on stages: Stage-1 , consists of Stage-8
  Stage-8
  Stage-3 depends on stages: Stage-8
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-7
    Conditional Operator

  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: b
            Statistics: Num rows: 9961472 Data size: 477102080 Basic stats:
COMPLETE Column stats: NONE
            Reduce Output Operator
              key expressions: col1 (type: string)
              sort order: +
              Map-reduce partition columns: col1 (type: string)
              Statistics: Num rows: 9961472 Data size: 477102080 Basic stats:
COMPLETE Column stats: NONE
```

```
              value expressions: col1 (type: string)
          TableScan
            alias: a
            Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
            Reduce Output Operator
              key expressions: col0 (type: string)
              sort order: +
              Map-reduce partition columns: col0 (type: string)
              Statistics: Num rows: 9963904 Data size: 477218560 Basic stats:
COMPLETE Column stats: NONE
              value expressions: col0 (type: string), col1 (type: string), col2
(type: string), col3 (type: string), col4 (type: string), col5 (type: string),
col6 (type: string)
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1} {VALUE._col2} {VALUE._col3}
{VALUE._col4} {VALUE._col5} {VALUE._col6}
            1 {VALUE._col1}
          handleSkewJoin: true
          outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6,
_col10
          Statistics: Num rows: 10960295 Data size: 524940416 Basic stats:
COMPLETE Column stats: NONE
          Filter Operator
            predicate: (_col0 = _col10) (type: boolean)
            Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
            Select Operator
              expressions: _col0 (type: string), _col1 (type: string), _col2
(type: string), _col3 (type: string), _col4 (type: string), _col5 (type:
string), _col6 (type: string)
              outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5, _col6
              Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
              File Output Operator
                compressed: false
                Statistics: Num rows: 5480147 Data size: 262470184 Basic stats:
COMPLETE Column stats: NONE
                table:
                    input format: org.apache.hadoop.mapred.TextInputFormat
                    output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                    serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

  Stage: Stage-4
    Conditional Operator

  Stage: Stage-8
    Map Reduce Local Work
      Alias -> Map Local Tables:
        1
          Fetch Operator
            limit: -1
      Alias -> Map Local Operator Tree:
        1
          TableScan
            HashTable Sink Operator
              condition expressions:
                0 {0_VALUE_0} {0_VALUE_1} {0_VALUE_2} {0_VALUE_3} {0_VALUE_4}
{0_VALUE_5} {0_VALUE_6}
```

```
                1 {1_VALUE_0}
              keys:
                0 joinkey0 (type: string)
                1 joinkey0 (type: string)

  Stage: Stage-3
    Map Reduce
      Map Operator Tree:
          TableScan
            Map Join Operator
              condition map:
                   Inner Join 0 to 1
              condition expressions:
                0 {0_VALUE_0} {0_VALUE_1} {0_VALUE_2} {0_VALUE_3} {0_VALUE_4}
{0_VALUE_5} {0_VALUE_6}
                1 {1_VALUE_0}
              keys:
                0 joinkey0 (type: string)
                1 joinkey0 (type: string)
              outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6, _col10
              Filter Operator
                predicate: (_col0 = _col10) (type: boolean)
                Select Operator
                  expressions: _col0 (type: string), _col1 (type: string), _col2
(type: string), _col3 (type: string), _col4 (type: string), _col5 (type:
string), _col6 (type: string)
                  outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6
                  File Output Operator
                    compressed: false
                    table:
                        input format: org.apache.hadoop.mapred.TextInputFormat
                        output format:
org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
                        serde:
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
      Local Work:
        Map Reduce Local Work

  Stage: Stage-0
    Fetch Operator
      limit: -1

Time taken: 0.331 seconds, Fetched: 110 row(s)
```

As above shows, there are 2 join operators, one is common join and the other one is map join.
And it shows "handleSkewJoin: true".

Tips:

1. Below parameter needs to be set to enable skew join.

```
set hive.optimize.skewjoin=true;
```

2. Below parameter determine if we get a skew key in join.
If we see more than the specified number of rows with the same key in join operator, we think the key as a skew join key.

```
set hive.skewjoin.key=100000;
```