LanguageManual JoinOptimization

Join Optimization

For a general discussion of Hive joins including syntax, examples, and restrictions, see the Joins wiki doc.

Improvements to the Hive Optimizer

Version

The join optimizations described here were added in Hive version 0.11.0. See HIVE-3784 and related JIRAs.

This document describes optimizations of Hive's query execution planning to improve the efficiency of joins and reduce the need for user hints.

Hive automatically recognizes various use cases and optimizes for them. Hive 0.11 improves the optimizer for these cases:

- Joins where one side fits in memory. In the new optimization:
  - that side is loaded into memory as a hash table
  - only the larger table needs to be scanned
  - fact tables have a smaller footprint in memory
- Star-schema joins
- Hints are no longer needed for many cases.
- Map joins are automatically picked up by the optimizer.

Star Join Optimization

A simple schema for decision support systems or data warehouses is the star schema, where events are collected in large *fact tables*, while smaller supporting tables (*dimensions*) are used to describe the data.

The TPC DS is an example of such a schema. It models a typical retail warehouse where the events are sales and typical dimensions are date of sale, time of sale, or demographic of the purchasing party. Typical queries aggregate and filter fact tables along properties in the dimension tables.

Star Schema Example

```
Select count(*) cnt
From store_sales ss
     join household_demographics hd on (ss.ss_hdemo_sk = hd.hd_demo_sk)
     join time_dim t on (ss.ss_sold_time_sk = t.t_time_sk)
     join store s on (s.s_store_sk = ss.ss_store_sk)
Where
     t.t_hour = 8
     t.t_minute >= 30
     hd.hd_dep_count = 2
order by cnt;
```

Prior Support for MAPJOIN

Hive supports MAPJOINs, which are well suited for this scenario – at least for dimensions small enough to fit in memory. Before release 0.11, a MAPJOIN could be invoked either through an optimizer hint:

```
select /*+ MAPJOIN(time_dim) */ count(*) from
store_sales join time_dim on (ss_sold_time_sk = t_time_sk)
```

or via auto join conversion:

```
set hive.auto.convert.join=true;
select count(*) from
store_sales join time_dim on (ss_sold_time_sk = t_time_sk)
```

The default value for hive.auto.convert.join was false in Hive 0.10.0. Hive 0.11.0 changed the default to true (HIVE-3297). Note that hive-default.xml.template incorrectly gives the default as false in Hive 0.11.0 through 0.13.1.

MAPJOINs are processed by loading the smaller table into an in-memory hash map and matching keys with the larger table as they are streamed through. The prior implementation has this division of labor:

- 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
- No reduce task

Limitations of Prior Implementation

The MAPJOIN implementation prior to Hive 0.11 has these limitations:

- The mapjoin operator can only handle one key at a time; that is, it can perform a multi-table join, but only if all the tables are joined on the same key. (Typical star schema joins do not fall into this category.)
- Hints are cumbersome for users to apply correctly and auto conversion doesn't have enough logic to consistently predict if a MAPJOIN will fit into memory or not.
- A chain of MAPJOINs is not coalesced into a single map-only job, unless the query is written as a cascading sequence of `mapjoin(table, subquery(mapjoin(table, subquery....)`. Auto conversion never produces a single map-only job.
- The hashtable for the mapjoin operator has to be generated for each run of the query, which involves downloading all the data to the Hive client machine as well as uploading the generated hashtable files.

Enhancements for Star Joins

The optimizer enhancements in Hive 0.11 focus on efficient processing of the joins needed in star schema configurations. The initial work was limited to star schema joins where all dimension tables after filtering and projecting fit into memory at the same time. Scenarios where only some of the dimension tables fit into memory are now implemented as well (HIVE-3996).

The join optimizations can be grouped into three parts:

- Execute chains of mapjoins in the operator tree in a single map-only job, when maphints are used.
- Extend optimization to the auto-conversion case (generating an appropriate backup plan when optimizing).
- Generate in-memory hashtable completely on the task side. (Future work.)

The following sections describe each of these optimizer enhancements.

Optimize Chains of Map Joins

The following query will produce two separate map-only jobs when executed:

```
select /*+ MAPJOIN(time_dim, date_dim) */ count(*) from
store_sales
join time_dim on (ss_sold_time_sk = t_time_sk)
join date_dim on (ss_sold_date_sk = d_date_sk)
where t_hour = 8 and d_year = 2002
```

It is likely, though, that for small dimension tables the parts of both tables needed would fit into memory at the same time. This reduces the time needed to execute this query dramatically, as the fact table is only read once instead of reading it twice and writing it to HDFS to communicate between the jobs.

Current and Future Optimizations

1. Merge M*-MR patterns into a single MR.

2. Merge MJ->MJ into a single MJ when possible.
3. Merge MJ* patterns into a single Map stage as a chain of MJ operators. (Not yet implemented.)

If `hive.auto.convert.join` is set to true the optimizer not only converts joins to mapjoins but also merges MJ* patterns as much as possible.

Optimize Auto Join Conversion

When auto join is enabled, there is no longer a need to provide the map-join hints in the query. The auto join option can be enabled with two configuration parameters:

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

The default for `hive.auto.convert.join.noconditionaltask` is true which means auto conversion is enabled. (Originally the default was false – see HIVE-3784 – but it was changed to true by HIVE-4146 before Hive 0.11.0 was released.)

The size configuration enables the user to control what size table can fit in memory. This value represents the sum of the sizes of tables that can be converted to hashmaps that fit in memory. Currently, n-1 tables of the join have to fit in memory for the map-join optimization to take effect. There is no check to see if the table is a compressed one or not and what the potential size of the table can be. The effect of this assumption on the results is discussed in the next section.

For example, the previous query just becomes:

```
select count(*) from
store_sales
join time_dim on (ss_sold_time_sk = t_time_sk)
join date_dim on (ss_sold_date_sk = d_date_sk)
where t_hour = 8 and d_year = 2002
```

If time_dim and date_dim fit in the size configuration provided, the respective joins are converted to map-joins. If the sum of the sizes of the tables can fit in the configured size, then the two map-joins are combined resulting in a single map-join. This reduces the number of MR-jobs required and significantly boosts the speed of execution of this query. This example can be easily extended for multi-way joins as well and will work as expected.

Outer joins offer more challenges. Since a map-join operator can only stream one table, the streamed table needs to be the one from which all of the rows are required. For the left outer join, this is the table on the left side of the join; for the right outer join, the table on the right side, etc. This means that even though an inner join can be converted to a map-join, an outer join cannot be converted. An outer join can only be converted if the table(s) apart from the one that needs to be streamed can be fit in the size configuration. A full outer join cannot be converted to a map-join at all since both tables need to be streamed.

Auto join conversion also affects the sort-merge-bucket joins.

Version 0.13.0 and later

Hive 0.13.0 introduced `hive.auto.convert.join.use.nonstaged` with a default of false (HIVE-6144).

For conditional joins, if the input stream from a small alias can be directly applied to the join operator without filtering or projection, then it does not need to be pre-staged in the distributed cache via a MapReduce local task. Setting `hive.auto.convert.join.use.nonstaged` to true avoids pre-staging in those cases.

Current Optimization

1. Group as many MJ operators as possible into one MJ.

As Hive goes through the conversion to map-joins for join operators based on the configuration flags, an effort is made at the end of these conversions to group as many together as possible. Going through in a sequence, if the sum of the sizes of the tables participating in the individual map-join operators is within the limit configured by the `noConditionalTask.size` flag, these MJ operators are combined together. This ensures more speedup with regard to these queries.

Auto Conversion to SMB Map Join

Sort-Merge-Bucket (SMB) joins can be converted to SMB map joins as well. SMB joins are used wherever the tables are sorted and bucketed. The join boils down to just merging the already sorted tables, allowing this operation to be faster than an ordinary map-join. However, if the tables are partitioned, there could be a slow down as each mapper would need to get a very small chunk of a partition which has a single key.

The following configuration settings enable the conversion of an SMB to a map-join SMB:

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

There is an option to set the big table selection policy using the following configuration:

```
set hive.auto.convert.sortmerge.join.bigtable.selection.policy
    = org.apache.hadoop.hive.ql.optimizer.TableSizeBasedBigTableSelectorForAutoSMJ;
```

By default, the selection policy is average partition size. The big table selection policy helps determine which table to choose for only streaming, as compared to hashing and streaming.

The available selection policies are:

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

The names describe their uses. This is especially useful for the fact-fact join (query 82 in the TPC DS benchmark).

SMB Join across Tables with Different Keys

If the tables have differing number of keys, for example Table A has 2 SORT columns and Table B has 1 SORT column, then you might get an index out of bounds exception.

The following query results in an index out of bounds exception because emp_person let us say for example has 1 sort column while emp_pay_history has 2 sort columns.

Error Hive 0.11

```
SELECT p.*, py.*
FROM emp_person p INNER JOIN emp_pay_history py
ON   p.empid = py.empid
```

This works fine.

Working query Hive 0.11

```
SELECT p.*, py.*
FROM emp_pay_history py INNER JOIN emp_person p
ON   p.empid = py.empid
```

Generate Hash Tables on the Task Side

Future work will make it possible to generate in-memory hashtables completely on the task side.

Pros and Cons of Client-Side Hash Tables

Generating the hashtable (or multiple hashtables for multitable joins) on the client machine has drawbacks. (The *client machine* is the host that is used to run the Hive client and submit jobs.)

- **Data locality:** The client machine typically is not a data node. All the data accessed is remote and has to be read via the network.
- **Specs:** For the same reason, it is not clear what the specifications of the machine running this processing will be. It might have limitations in memory, hard drive, or CPU that the task nodes do not have.
- **HDFS upload:** The data has to be brought back to the cluster and replicated via the distributed cache to be used by task nodes.

Pre-processing the hashtables on the client machine also has some benefits:

- What is stored in the distributed cache is likely to be smaller than the original table (filter and projection).
- In contrast, loading hashtables directly on the task nodes using the distributed cache means larger objects in the cache, potentially reducing opportunities for using MAPJOIN.

Task-Side Generation of Hash Tables

When the hashtables are generated completely on the task side, all task nodes have to access the original data source to generate the hashtable. Since in the normal case this will happen in parallel it will not affect latency, but Hive has a concept of storage handlers and having many tasks access the same external data source (HBase, database, etc.) might overwhelm or slow down the source.

Further Options for Optimization

1. Increase the replication factor on dimension tables.

2. Use the distributed cache to hold dimension tables.