

Parquet stats for evaluating aggregates

Taras Bobrovysky, March 20th, 2017

We want to optimize aggregate evaluation for columns that have statistics on Parquet tables. The idea is that instead of scanning and materializing every row in a row group, we utilize Parquet statistics and emit only a single row (or a few rows) per row group without scanning the entire row group. The following examples illustrate several interesting cases that impact the design.

Example A: Single statistic per column.

```
SELECT MIN(int_col), MAX(bigint_col) from parquet_table
```

- Emit a single tuple for each row group
- Populate the scan slots with the appropriate min/max statistic
- No plan changes needed

Example B: Multiple statistics per column.

```
SELECT MIN(int_col), MAX(int_col) FROM parquet_table
```

- Scan tuple only has one slot, but we need two statistics
- Could expand tuple, or emit two rows with appropriate setting of NULLs
- Expanding the tuple requires plan changes

Example C: Handling columns without statistics.

```
SELECT MIN(int_col), MAX(int_col), MAX(timestamp_col) FROM parquet_table
```

- No statistics for timestamp_col
- Can we still use the min/max stats for int_col?
- Generally: A Parquet file may have stats for a subset of the desired columns

Two approaches to handle the above cases were identified. We will use the following example query to demonstrate them:

Example D

```
SELECT MIN(int_col), MAX(int_col), MAX(bigint_col), MIN(timestamp_col), MAX(timestamp_col)  
FROM parquet_table
```

In this example query, the timestamp_col has no stats.

Approach 1: Expand Vertically

Conceptually separate tuples into 3 mutually exclusive categories.

1. Tuples that have been populated from column data.
2. Tuples that take data from the min statistic.
3. Tuples that take data from the max statistic.

The tuple layout is identical to what it is today. For the *Example D* query, the row has 3 slots: {int_col, bigint_col, timestamp_col}.

The scan produces $N + 2$ rows for each row group, where N is the number of rows in the row group. The stats are used to prepare a template tuple. Only timestamp_col is scanned.

The first category has N rows, and they look like this:

[int_col=null, bigint_col=null, timestamp_col=<value read from the file>]

There is 1 row from the second category, which looks like this:

[int_col=<min statistic>, bigint_col=null, timestamp_col=null]

There is 1 row from the third category which looks like this:

[int_col=<max statistic>, bigint_col=<max statistic>, timestamp_col=null]

All of the changes are hidden completely inside the scan node and the aggregation node does not need to know about this. The downside to this approach is that the number of rows received by the aggregation node per row group is different than the number of rows in the row group. This is a problem for a query like the following:

Example E

```
SELECT MIN(int_col), MAX(int_col), SUM(COALESCE(bigint_col, 1)) FROM parquet_table
```

If we optimize this query using stats, then the SUM() will return incorrect results because we changed the number of rows produced by the scan. Setting bigint_col to NULL will not work. It is difficult to even determine during planning that the Parquet stats optimization should not be applied for this query.

Approach 2: Expand Horizontally

Increase the number of slots in the row in order to be able to fit both min, max and possibly count statistics. This is a more invasive solution and requires making tuple and plan changes in the FE.

For the *Example D* query, The query has 4 slots: {int_col (min stat), int_col (max stat), bigint_col (max stat), timestamp_col}. There is only 1 slot for timestamp_col because the data that it contains is not taken from a Parquet stat (we know that we cannot use stats for TIMESTAMP).

The scan produces N rows for each row group, where N is the number of rows in the row group. The stats are used to prepare a template tuple. Only timestamp_col is scanned.

So for each tuple, some slots are populated with values from the column data, and some slots are populated with data taken from Parquet stat.

Conclusion

Approach 2 is preferable to approach 1 because it is more general and can be applied to more queries. There is also some additional complexity with approach 1 because we have to be able to determine in the FE when it is appropriate to apply the Parquet stats optimization and when it is not.