Query Acceleration in Presto using Sampled Tables - Proposal

Gurmeet Singh, Uber

Goals

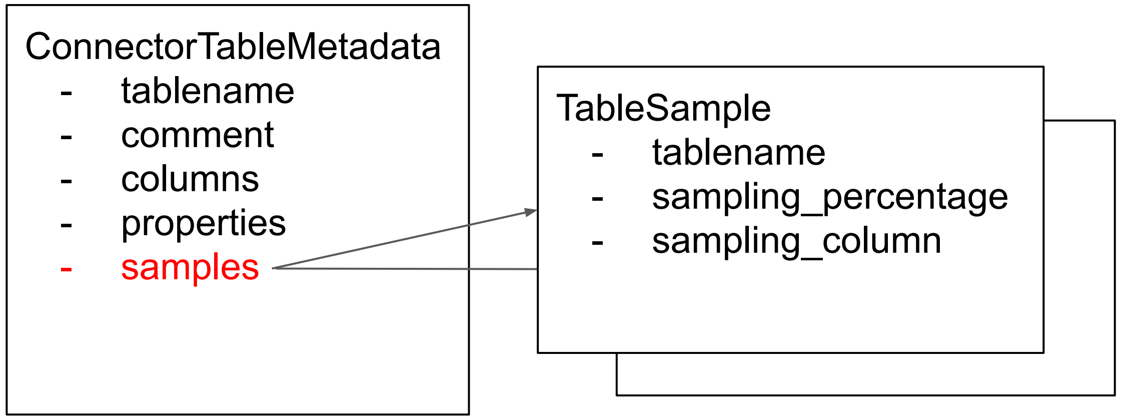
- Accelerate queries by using sampled datasets
- Drive adoption by having the engine automatically discover and use the sample datasets instead of the user having to know about them.
- Make aggregations like count, sum, approx_distinct return results of the same magnitude as the original query
- Explicit opt-in behavior so that the caller knows about it.

Non-Goals

- Creation of sampled datasets. This is just about how to make them known and drive adoption.
- Support for all kind of queries. Fail in unsupported cases
- User can’t see the optimized query (except through explain plan)

A Connector independent way to declare sample tables

- ConnectorTableMetadata can contain information about sample tables.



- A table can have one or more samples
- Sampling_column means that sampling is done based on a column e.g. keep in sample if crc32(xxhash64(to_utf8(column))) % 100 < X where X is the sampling percentage

Declaring sample tables in HMS

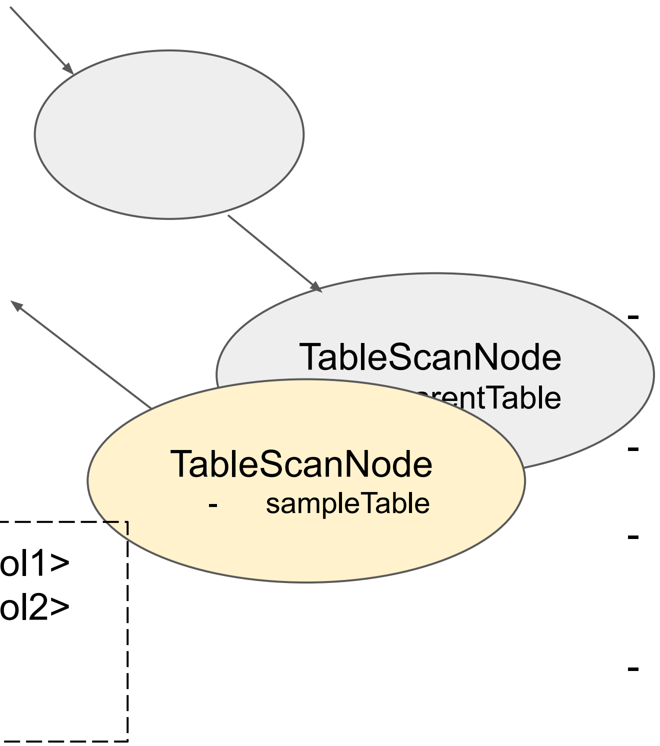
In our prototype, for the Hive Connector

- The parent table declares its samples by a property ‘sampled_tables’. This is a comma separated list of table names that are its samples
- The sample table declares its sampling percentage and sampling column (if any) by properties ‘sampling_pct’ and ‘sampling_column’ respectively defined on the sample table.
- The *HiveMetadata::getTableMetadata()* when loading the parent table metadata also populates the samples in the ConnectorTableMetadata based on above

Table replacement and Auto-scaling Aggregations

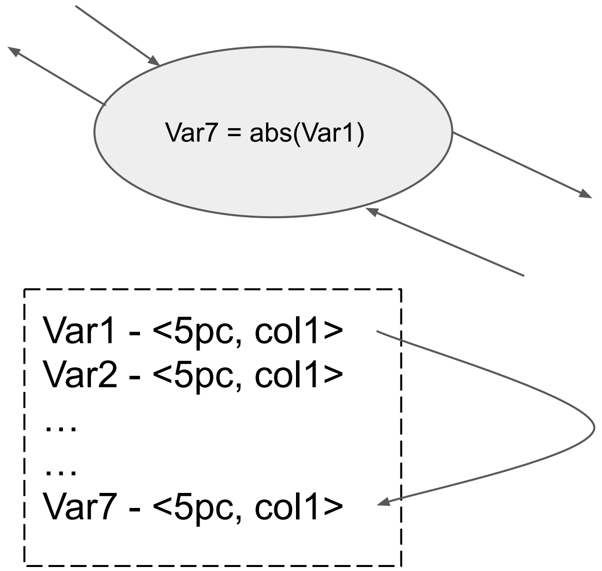
- Just replacing the parent tables by the sample tables can make the query results wrong when aggregations (sum, count, approx_distinct) are involved
- Aggregations need to be scaled as well
- Whole thing need to work within the optimization framework i.e. *tablereplaced_and_scaled_plan_root = optimize(plan_root)* in order to minimize changes to the presto code.
- The optimization is gated by a session flag (so that user can explicitly opt in)

Algorithm - TableScanNode



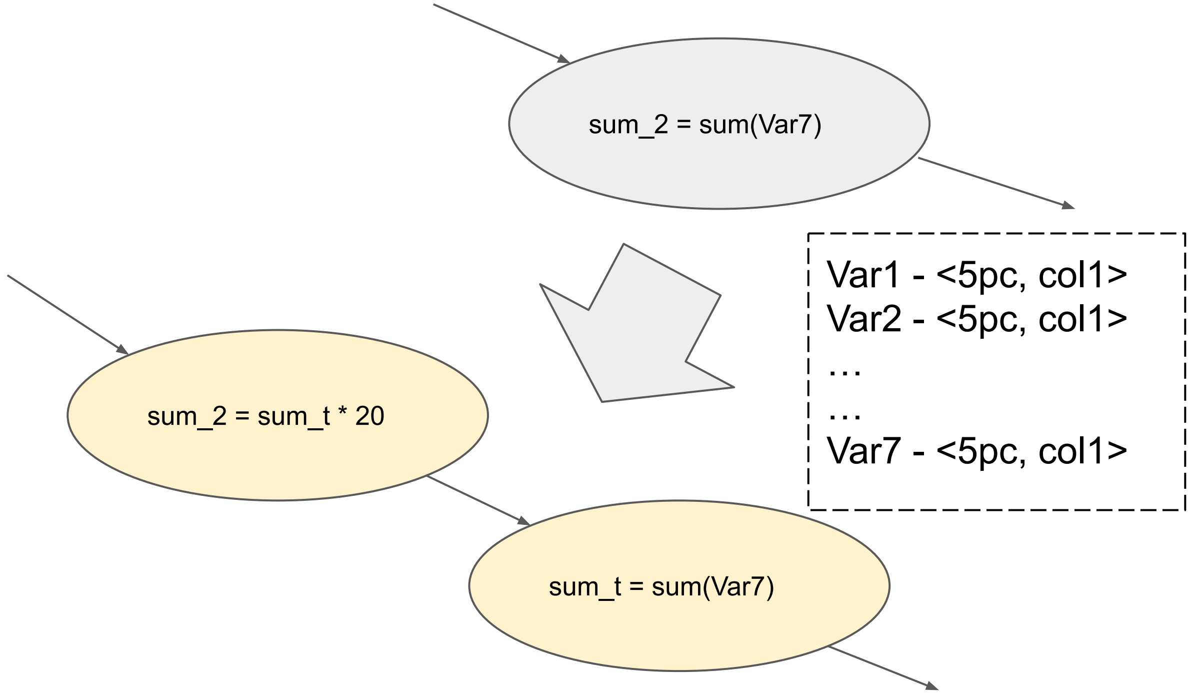
- On reaching a table scan node, if the table has a sample, then replace the node with one for the sample table.
- If replacing, put output variables in a map
- The map is between the variable name and sampling percentage, sampling column (of the sample table)
- The map is passed up the tree

Algorithm - Project Node



- When a project node is assigning a new variable based on one that exists in the map, then add the derived variable to the map too with the same pct, column id

Algorithm - Aggregation Node



- An aggregation node that is aggregating any of the variables in the map, adds a project node on top of it in order to scale the aggregated variable (based on the sampling pct in the map)
- Only for sum, count, approx_distinct kind of aggregations
- The name of the scaled variable remains the same as the original variable so as to not impact the ancestors in the tree

Fail close

- The scaling can encounters scenarios when the right behavior is not clear
- Choose to fail the query with an unsupported exception so that the caller (outside Presto) can retry the query after removing the session parameter
- This makes the engine behavior very clear

Salient features of auto scaling

- All changes in planning (optimizing) stage. Users can see the transformed plan using “explain …”.
- The algorithm works only by node local actions and does not require cross node interactions.
- Join nodes can throw exception if both sides of the join are being scaled (since joining two sampled sets can be very sparse)
- Filter nodes can prune map if the filtering is happening based on sampling column e.g. if a table is sampled on trip_id and query is ‘select …. from table where trip_id = xxx’ then there is no point in scaling aggregations in the query

Salient features of auto scaling

- count(*) can throw exception since it is an implicit aggregation rather than based on certain vars.
- Some columns can have low cardinality and hence it is not appropriate to scale their aggregations. The tablescan node can avoid putting those column based variables in the map.

Current State

- Prototyped with basic aggregation scaling
  - Need to define rules for join nodes, filter nodes
  - Need to handle low cardinality columns
- Need to understand rules for complex nodes like Window etc
- Feedback/comments/suggestions appreciated.