ALCHEMY: An Algebra for Optimizing Private Data Federation Queries

Authors

February 10, 2019

We need to formalize how ALCHEMY derives its cardinality bounds so that the optimizer can order and configure the operators such that we minimize the overhead of secure multi-party computation. Consider query Q that computes on the attributes λ_Q. We consider only single-column statistics for simplicity, but these statistics will generalize to composite keys, or sets of attributes, using standard techniques. The optimizer makes all of its decisions at the granularity of the operator. For each operator Θ in Q, it produces the output:

$$R_\Theta = \begin{cases} T_\Theta & \text{an array of tuples} \\ \mathcal{M}_\Theta & \text{metadata for } T_\Theta \text{ w.r.t } Q \end{cases} \tag{1}$$

This output relation's metadata, $\mathcal{M}_\Theta$, contains statistics with which the query planner derives the intermediate result sizes for each step in a proposed query tree. These statistics come from the federation's common data model and client-supplied bounds. The operator has a set of attributes, $\lambda_\Theta \subseteq \lambda_Q$, on which it or its ancestors computes. Hence, $\mathcal{M}_\Theta = \{|T_\Theta|, \mathcal{M}_{\Theta,i} : \forall i \in \lambda_\Theta\}$ or the length of Θ's obliviously evaluated output paired with metadata for each attribute in the query. For each attribute, $a_i \in \lambda_\Theta$, the engine maintains the following values:

$$\mathcal{M}_{\Theta,i} = \begin{cases} C_i & \text{Distinct cardinality of } a_i \\ D_i & \text{Domain of } a_i \\ M_i & \text{Maximum multiplicity of } a_i \\ [min_i, max_i] & \text{Range of } a_i \end{cases} \tag{2}$$

These statistics are upper bounds on these values. In practice, and in practice the input data may much lower values for these figures. For example, if we derive the distinct cardinality, C_i, of a column from its source relation and filter it, then its new C_i will often be substantially reduced. If a given figure is unknown, the planner initializes it to infinity. For each operator we calculate its maximum possible output cardinality and its attribute-level metadata using the makeup of its inputs, denoted as Θ_{-1}. If Θ is a binary operator, then we union the metadata of its children for its input. For example, if we are preparing $R\,\Theta\,S$, then $\mathcal{M}_{\Theta_{-1}} = \mathcal{M}_R \cup \mathcal{M}_S$.

Operator	$\overline{\mathcal{F}_\Theta}(\mathcal{M}_{\Theta_{-1},i})$
Filter ($\sigma_{a_i=x}$)	$C_\sigma = 1$ $D_\sigma = min_\sigma = max_\sigma = x$ $M_\sigma = M_{-1}$ $\lvert T_\sigma \rvert = M_\sigma$
Join ($R \bowtie_{id} S$)	$D_\bowtie = D_{-1,R.id} \cap D_{-1,S.id}$ $min_\bowtie = max(min_{-1,R.id}, min_{-1,S.id})$ $max_\bowtie = min(max_{-1,R.id}, max_{-1,S.id})$ $C_\bowtie = min(\lvert D_\bowtie \rvert, max_\bowtie - min_\bowtie,$ $min(C_{-1,R.id}, C_{-1,S.id}))$ $M_\bowtie = M_{-1,R.id} * M_{-1,S.id}$ $\lvert T_\bowtie \rvert = C_\bowtie * M_\bowtie$
Sort	$\mathcal{M}_{sort} = \mathcal{M}_{sort_{-1}}$
Projection (Π_{expr})	$\forall V \in \{D, M, min, max\} : V_\Pi = expr(V_{-1})$ $C_\Pi = min(\lvert D_\Pi \rvert, max_\Pi - min_\Pi, C_{-1})$ $\lvert T_\Pi \rvert = \lvert T_{\Pi_{-1}} \rvert$
Simple Aggregation	$C_{agg} = M_{agg} = \lvert T_{agg} \rvert = 1$ min, max, and domain are reset to ∞
Aggregation with Group-by	$\mathcal{M}_{agg} = \mathcal{M}_{agg_{-1}}$ $M_{agg} = 1$ $\lvert T_{agg} \rvert = C_{-1}$

Table 1: Maximum cardinality functions for maintaining query metadata.

To plan the execution of the operator Θ that computes on attribute i, Alchemy derives $\mathcal{M}_{\Theta,i}$ using a function, $\overline{\mathcal{F}_\Theta}(\mathcal{M}_{\Theta_{-1},i})$, that is specific to the operator at hand to update its metadata. For all other attributes, $a_i \in \lambda_\Theta$, we *trim* their values to match the output cardinality of Θ by invoking a method, $trim(\mathcal{M}_{\Theta_{-1}}, i, |T_\Theta|)$. The trim function sets the attribute's distinct cardinality, C_i, to the minimum of the output cardinality, $|T_\Theta|$, and its previous value, $\mathcal{M}_{\Theta_{-1},C_i}$. Likewise, we set the maximum multiplicity of a_i as $\mathcal{M}_{\Theta,M_i} = min(|T_\Theta|, \mathcal{M}_{\Theta_{-1},M_i})$. We mirror the domain and range of $\mathcal{M}_{\Theta,i}$ from its predecessor. To populate the metadata of R_Θ, for each a_i in λ_Θ we calculate $\mathcal{M}_{\Theta,i}$. In other words:

$$\forall i \in \lambda_\Theta : \mathcal{M}_{\Theta,i} = \begin{cases} \overline{\mathcal{F}_\Theta}(\mathcal{M}_{\Theta_{-1},1}) & \text{if } \Theta \text{ computes on } a_i \\ trim(\mathcal{M}_{\Theta_{-1}}, i, |T_\Theta|) & \text{otherwise} \end{cases} \tag{3}$$

Table 1 describes how the system calculates the upper bound cardinalities and associated statistics, $\overline{\mathcal{F}_\Theta}$, for each query operator. Unless otherwise noted, we show attribute i's previous metadata using the shorthand $\mathcal{M}_{\Theta_{-1},i} = \{C_{-1}, D_{-1}, M_{-1}, min_{-1}, max_{-1}\}$.

For filters with equality predicates, the number of distinct values, C_σ, in the output a_i is one because it contains only to the values that match the selection criteria. This step also collapses the domain and range of a_i to the value for which we filtered. The maximum multiplicity, M_σ, is equal to that of the operator's inputs. The output size of the filter is equal in length to the maximum multiplicity of its inputs.

We handle joins by first analyzing how the input relations intersect based on their metadata using standard techniques. If the domain and range overlap, then we tighten these statistics to reflect their common values and ranges. If they do not overlap, then the domain and range are empty sets and the join's output will be have a length of zero. The join's distinct output cardinality for a_i is the minimum of 1) its output domain, 2) its output range, and 3) the maximum potential overlap between the distinct cardinalities of the input relations. The maximum multiplicity of each match in the join's output is equal to the product of this figure for its inputs to accommodate a full cross-product over the largest set of matches on each side of the join. Naturally, the output cardinality of this operator is its distinct value count multiplied by the maximum number of duplicates admitted.

Our algebra shows how to calculate metadata for joins and filters with equality predicates. If we have information about the domain and range of the input attributes, then we may extend this work to handle additional comparisons such as greater than or less than for theta joins. This logic is the same as that used for conventional optimizers for statistics estimation, thus we omit it here.

Since sorting the data does not alter its contents, a sort operator retains the metadata of its input relations. Projection over an expression follows a similar logic albeit with the domain and range of the values computed on being revised to reflect their transformation. If we consider the example in Figure **??**, the projection `decade = birth_year /10` will transform this attribute's range from $\{1900, \ldots, NOW()\}$ to $\{190, \ldots, NOW()/10\}$ and its domain collapses accordingly. Similarly, the maximum multiplicity of this column is multiplied by 10 to reflect how more values are now indistinguishable from one another. Since this operator only transforms the values and not the inclusion of a tuple, its output cardinality is the same size as its input.

Simple aggregation, e.g., `SELECT COUNT(*) FROM ...`, naturally has an output cardinality of one. Its distinct value count and maximum multiplicity follow suit. The system defaults to initializing its domain and range to infinity. In future work, we may tailor these statistics using the aggregate function and the source column's domain and range.

We now consider $\overline{\mathcal{F}_\Theta}$ for complex aggregates, or ones with group-by clauses. This function begins by copying the metadata from the input relation. It then adjusts the maximum multiplicity of its output to one because each distinct group-by partition may produce at most one output tuple. The output cardinality of this operator is equal to the maximum number of distinct group-by values, C_{-1}.

This algebra enables the query optimizer to calculate the minimum intermediate result size for each operator in a proposed query plan without blindly resorting to the worst-case scenario. By maintaining these statistics to obliviously model the tuples as they flow through the query tree, we will be able to profitably reorder operators without revealing new information.
TODO: add example query. Should this go before or after the algebra walk-

through? Before. Use it as motivation.