Automated Data Configuration for NoSQL Systems

Kevin Lee Akshay Mittal Sachin Ravi

1 High-level Design

1. Will receive SQL query workload
2. Finding Interesting Table-subsets with selection conditions imposed (based purely on table-subsets occurring in queries) [1]
3. Do merging to get more options (to get table-subsets that are not explicitly in any query)
4. For each of these table-subsets, we calculate score across all queries. Supose Q is the set of queries we are considering and $m \in M$, the set of all table-subsets we are considering, is one possible table-subset,

$$queryCost(m, Q) = \text{cost to answer query using table-subset } m$$
$$updateCost(m, Q) = \beta \cdot \text{number of queries that update any table in } m$$
$$storageCost(m) = \alpha \cdot \text{size of storing } m$$
$$cost(m, Q) = queryCost(m, Q) + updateCost(m, Q) + storageCost(m)$$

   Add each cost to priority queue

5. Get k lowest costs out of priority queue and keep the table-subsets corresponding to these k lowest costs (if having picked smaller subset, ingore larger subset if it is considered later)

1.1 Finding Interesting Table-Subsets

$$Cost(T) = \text{cost of all queries where subset } T \text{ occurs}$$

$$WeightedCost(T) = \sum_i cost(Q_i) \cdot \frac{\text{sum of sizes of tables in T}}{\text{sum of sizes of all tables referenced in } Q_i}$$

Lemma 1. *For $T_1 \subseteq T_2$, $Cost(T_1) \geq Cost(T_2)$.*

Lemma 2. $WeightedCost(T) \geq C \Rightarrow Cost(T) \geq C \Leftrightarrow Cost(T) \not\geq C \Rightarrow WeightedCost(T) \not\geq C$

[1] Might be interesting to try without selection conditions imposed and no merging

Algorithm 1 Algorithm for finding potential table-subsets in query workload

INPUT: C (baseline), $maxSize$ (max size for a table-subset)

$S_1 \leftarrow \{T \mid T$ is a table-subset of size 1 with $Cost(T) \geq C\}$
$i \leftarrow 1$

while $i < maxSize$ and $|S_i| > 0$ **do**
$i = i + 1$
$S_i = \{\}$
$G \leftarrow \{T \mid T$ is a table subset of size i and $\exists s \in S_{i-1}$such that $s \subset T\}$
  for $g \in G$ **do**
    if $Cost(g) \geq C$ **then** $S_i = S_i \cup \{g\}$
    end if
  end for
end while
$S \leftarrow S_1 \cup S_2 \cup \ldots \cup S_{maxSize}$
$R \leftarrow \{T \mid T \in S$ and $Weight(T) \geq C\}$
return R

When picking table-subsets, we want to pick subsets that are in expensive queries (because materializing these table-subsets will decrease a great cost). We pass in a threshold value of the minimum total cost we would like to satisfy when considerng a certain table-subset. We start with single table-subsets and continually work our way up to higher table-subsets, while ensuring that we still satisfy the threshold.

1.2 Merging Table-Subsets

The problem with only using the previous procedure is that we only consider table-subsets that exist in some query in the workload. Oftentimes, there are table-subsets that don't occur in any query but are useful to materialize anyway.

[Insert Example]

Algorithm 2 Pairwise Merge

INPUT: T_1, table-subset, T_2 table-subset

if $T_1 \cap T_2 = \emptyset$ **then** return $\emptyset$
end if

$newT \leftarrow$ union of T_1 and T_2's projection columns (only the ones on the insterseciton of T_1 and T_2's intersecting table-subsets), intersection of T_1 and T_2's selection conditions, and intersection of T_1 and T_2's table-subsets

return $newT$

Algorithm 3 Algorithm for producing merged table-subsets

INPUT: T, set of table-subsets

$R \leftarrow T$

while $|R| > 1$ **do**
    $M' = \emptyset$
    for $R_1, R_2 \in R$ **do**
        $newR \leftarrow$ Pairwise Merge of R_1 and R_2
        $M' \leftarrow M' \cup newR$
    end for

    if $M' = \emptyset$ **then**
        return $R - T$
    end if

    for all $m \in M'$ **do**
        Remove both parents of m from R
    end for
    $R \leftarrow R \cup M'$

end while
return $R - T$

1.3 Pruning

1.4 Cost Model

The basic thing we need to write is $cost(q, T)$ where q is a normalized query and $T = \{T_1, \ldots, T_k\}$ is a subset of tables with possible selection conditions imposed (the possible denormalized option).
$cost(q, T)$