Query Optimization
Exercise Session 4

Bernhard Radke

November 28, 2016
MVP

\[
\begin{align*}
R_1 & \quad 0.005 \quad R_2 \\
R_4 & \quad 0.05 \quad R_3 \\
1000 & \quad 100 \\
100 & \quad 500
\end{align*}
\]
Query graph to Weighted Directed Join Graph:

- nodes = joins
- physical edges between "adjacent" joins (share one relation)
- virtual edges - everywhere else
- WDJG is a clique
MVP

\[
\begin{array}{ccc}
R_1 & 0.005 & R_2 \\
0.05 & 0.02 & 0.001 \\
R_4 & 1000 & R_3 \\
100 & 500 & \\
\end{array}
\]

\[
\begin{array}{ccc}
V_{12} & \leftrightarrow & V_{23} \\
V_{14} & \leftrightarrow & V_{34} \\
\end{array}
\]
Annotations:

- edge weight $w_{u,v} = \frac{|\Delta_u|}{|u \cap v|}$
- the cost of a node $\equiv$ the cost of the join $C_{out}$
MVP

\[
\begin{array}{c}
R_1 \\
\downarrow^{0.05} \\
R_4
\end{array}
\quad \begin{array}{c}
0.005 \\
\downarrow^{0.02} \\
0.001 \\
\downarrow^{0.05} \\
R_3
\end{array}
\quad \begin{array}{c}
R_2
\end{array}
\quad \begin{array}{c}
1000 \\
100 \\
500 \\
5000 \\
1000
\end{array}
\]
Effective spanning tree (informally)

ESP corresponds to an ”effective” execution plan (no extra joins).
Three conditions:

1. \( T \) is binary

2. For every non-leaf node \( v_i \), for every edge \( v_j \rightarrow v_i \) there is a common base relation between \( v_i \) and the subtree with the root \( v_j \)

3. For every node \( v_i = R \bowtie S \) with two incoming edges \( v_k \rightarrow v_i \) and \( v_j \rightarrow v_i \)
   
   3.1 \( R \) or \( S \) can be present at most in one of the subtrees \( v_k \) or \( v_j \)
   
   3.2 unless the subtree \( v_j \) (or \( v_k \)) contains both \( R \) and \( S \)
MVP (informally)

Construct an effective spanning tree in two steps:

**Step 1 (Choose an edge to reduce the cost of an expensive operation)**
Start with the most expensive node, find the incoming edge that can reduce the cost the most. Update the cost of the node. Add the edge to the ESP, check the conditions. Repeat until

- no more edges can reduce any cost
- no more join nodes to consider
MVP (informally)

Construct an effective spanning tree in two steps:

**Step 1** *(Choose an edge to reduce the cost of an expensive operation)*
Start with the most expensive node, find the incoming edge that can reduce the cost the most. Update the cost of the node. Add the edge to the ESP, check the conditions. Repeat until
- no more edges can reduce any cost
- no more join nodes to consider

**Step 2** *(Find edges causing minimum increase to the result of joins)*
Similar to Step 1, but start with the cheapest node.
MVP - example

We start with a graph without virtual edges.
Two cost lists:

- for the Step 1: \(Q_1 = v_{14}, v_{34}, v_{12}, v_{23}\)
- for the Step 2: \(Q_2 = \emptyset\)
MVP - example

Start with $v_{14}$,
MVP - example

Start with $v_{14}$, select the edge $v_{12} \rightarrow v_{14}$. After $v_{12}$ is executed, $|R_1 \ltimes R_2| = 500$

We replace $R_1$ by $R_1 \ltimes R_2$ in $v_{14} = R_1 \ltimes R_4$:

$v_{14} = (R_1 \ltimes R_2) \ltimes R_4$

$\text{cost}(v_{14}) = 500 \times 100 \times 0.05 + 500 = 3000$
Add edge to EST.
Add new edge $v_{14} \rightarrow v_{23}$.
Consider $v_{14}$, no incoming edge with weight $< 1$:
$Q_1 = v_{34}, v_{12}, v_{23}$.
$Q_2 = v_{14}$
Consider $v_{34}$, one incoming edge with weight $< 1$:
Recompute cost: $cost(v_{34}) = 50 \times 100 \times 0.02 + 50 = 150$

$Q_1 = v_{12}, v_{34}, v_{23}$.

$Q_2 = v_{14}$
Remove edges, add to EST.

$Q_1 = v_{12}, v_{34}, v_{23}$.

$Q_2 = v_{14}$
MVP - example

\[ v_{12}, v_{23}, v_{14}, v_{34} \]

\[ v_{12} \text{ no incoming edge with weight } < 1 \]
\[ Q_1 = v_{34}, v_{23}. \]
\[ Q_2 = v_{12}, v_{14}. \]
\[ v_{34}, v_{23}: \text{no incoming edges with the weights } > 1 \]

\[ Q_1 = \emptyset. \]

\[ Q_2 = v_{23}, v_{34}, v_{12}, v_{14} \]

End of Step 1.
Step 2: try to increase the cost of the EST as little as possible. $v_{23}$: one incoming edge, does not violate the EST conditions. Add it and stop.
MVP - example

\[ v_{12} \rightarrow 500 \rightarrow v_{14} \quad v_{14} \leftarrow 3000 \leftarrow v_{12} \]

\[ v_{23} \rightarrow 50 \rightarrow v_{23} \]

\[ v_{34} \rightarrow 10 \rightarrow v_{34} \]
MVP - example
See also

Overview Dynamic Programming Strategy

- generate optimal join trees bottom up
- start from optimal join trees of size one (relations)
- build larger join trees by (re-)using those of smaller sizes
DP: Generating Linear Trees

DPsizeLinear(\(R\))

Input: a set of relations \(R = \{R_1, \ldots, R_n\}\) to be joined
Output: an optimal left-deep (right-deep, zig-zag) join tree

\(B = \) an empty DP table \(2^R \rightarrow \) join tree

for each \(R_i \in R\)

\(B[\{R_i\}] = R_i\)

for each \(1 < s \leq n\) ascending \{

for each \(S \subset R, R_i \in R : |S| = s - 1 \land R_i \not\in S\) \{

if \(\neg\)cross products \(\land \neg\)S connected to \(R_i\) continue

\(p_1 = B[S], p_2 = B[\{R_i\}]\)

if \(p_1 = \epsilon\) continue

\(P = \) CreateJoinTree\((p_1, p_2)\);

if \(B[S \cup \{R_i\}] = \epsilon \lor C(B[S \cup \{R_i\}]) > C(P)\)

\(B[S \cup \{R_i\}] = P\)

\}

\}

return \(B[\{R_1, \ldots, R_n\}]\)
iterate over subsets of the set of relations, the size is increasing

- $S_1, S_2$: $S_1 \cap S_2 = \emptyset$, $S_1$ is connected to $S_2$
DPsize - example

\[
\begin{array}{ccc}
R_1 & \text{0.1} & R_2 \\
\text{10} & \text{0.5} & \text{10} \\
R_3 & \text{0.6} & R_4 \\
\text{10} & \text{0.2} & \text{10}
\end{array}
\]
Bushy vs. Linear trees

- Linear: add one more relation every time, i.e. add $R$ to optimal $T_1$ to get optimal $T = T_1 \Join R$
- Bushy: consider all pairs of optimal $T_1$ and $T_2$ to find optimal $T = T_1 \Join T_2$
Iterate over subsets in the integer order
Before a join tree for $S$ is generated, all the relevant subsets of $S$ must be available
Enumerate \( \{ R_1, R_2, R_3, R_4 \} \) in Integer order.
The ability to build the DP table is crucial for passing the exam!
Homework: Task 1 (15 points)

- Give an example query graph with join selectivities for which the greedy operator ordering (GOO) algorithm does not give the optimal (with regards to $C_{out}$) join tree. Specify the optimal join tree.
- For that example perform the IKKBZ-based heuristics
Homework: Task 2 (15 points)

- Using the program from the last exercise as basis, construct the query graph for each connected component.
Info

- Exercises due: 9 AM, Dezember 05, 2016