Introduction and Motivation

- What is a progress estimator?
- How is it useful?
  - End user
  - DBA
  - Query level scheduling
  - Query dependent server timeouts ($k \times \text{est-time}$ as opposed to const timeouts)
- What is the challenge?
Definitions

- execution plan
- blocking operator
- pipeline
- driver node
Recognizing the pipelines in Query plan

Execute the following steps in bottom-up manner on query plan

- A leaf node (TableScan, IndexScan, IndexSeek) starts the pipeline.
- A FilterNode is part of the pipeline that its child operator belongs to.
- For a Hash Join, the join operator is included in the pipeline of the probe child, and the build child is the root of another pipeline.
- For a Merge-Join, the pipelines containing its children and the Merge Join operator itself are unioned to create a single pipeline.
Recognizing the pipelines in Query plan (contd.)

- For a Nested Loops or Index Nested Loops Join operator, the outer child, the join operator and its entire inner subtree are part of the same pipeline as the outer child node.
- Both Sort and Group-By (hash-based) operators, which are blocking, start a new pipeline of their own.
Example 1

Note: A is the build relation, B is the probe relation

P1 = \{ Table Scan A, Filter \}
P2 = \{ Index Scan B, Hash Join, Index Nested Loops, Index Seek C \}

Highlighted nodes are driver nodes.
Example 1

Note: A is the build relation, B is the probe relation
P1 = \{Table Scan A, Filter\}
P2 = \{Index Scan B, Hash Join, Index Nested Loops, Index Seek C\}
Highlighted nodes are driver nodes.
Driver nodes

- Every pipeline has a set of driver nodes, i.e., operators that are the sources of tuples operated upon by remaining nodes in the pipeline.
- More precisely, we define the driver nodes of a pipeline as the set of all leaf nodes of the pipeline, except those that are in the inner subtree of a Nested Loops/ Index Nested Loops join.

In the prev. example, the shaded nodes are driver nodes
- TableScan A is the driver node for the pipeline P1,
- Index Scan B is the driver node for pipeline P2.

It is possible for a pipeline to contain more than one driver node.
Example 2

P1 = Table Scan A
P2 = Index Scan B
P3 = Sort A, Sort B, Merge Join, Index Nested Loops, Index Seek C

Highlighted nodes are driver nodes.
Example 2

P1 = Table Scan A
P2 = Index Scan B
P3 = Sort A, Sort B, Merge Join, Index Nested Loops, Index Seek C
Highlighted nodes are driver nodes.
Desirable Properties of Progress Estimator

- Accuracy
- Fine granularity
- Low overhead
- Leveraging the feedback (from execution)
- Monotonicity
Operators in a query execution plan are typically implemented using a demand driven iterator model. Each operator exports a standard interface:

- Open()
- Close()
- GetNext()

Work done by query = total number of GetNext() calls
Progress estimation based on GNM

\[ gnm = \frac{\sum_i K_i}{\sum_i N_i} \]

- \( K_i \) is the number of GetNext() calls made on the \( i^{th} \) operator.
- \( N_i \) is the number of GetNext() calls made on the \( i^{th} \) operator by the end of query execution.
The Driver Node Estimator (DNE): Single pipeline queries

For simplicity, assume the pipeline is chain of m operators:

\[ Op_1 \rightarrow Op_2 \rightarrow Op_3 \rightarrow \ldots \rightarrow Op_m \]

\[ dne = \frac{K_1}{N_1} \]

**Driver node hypothesis**

\[ \frac{K_1}{N_1} \approx \frac{\sum_i K_i}{\sum_i N_i} \]

- Reasons
Monotonically decreasing pipelines

- What are monotonically decreasing pipelines?
  - $K_i \geq K_{i+1}$ and $N_i \geq N_{i+1}$

- Guarantee of $dne$ for monotonically decreasing pipelines
  - $\frac{gnm}{m} \leq dne \leq m.gnm$
Estimator for arbitrary query plans

For a execution plan with $s$ pipelines:

$$g_{nm} = \frac{\sum_{P_1} K + \ldots + \sum_{P_s} K}{\sum_{P_1} N + \ldots + \sum_{P_s} N}$$

We always know, $K$ values accurately

- If $P_i$ is completed, then $\sum_{P_i} N = \sum_{P_i} K$
- If $P_i$ is currently executing, then $\sum_{P_i} N = \sum_{P_i} K / dne$
- If $P_i$ has not started, then we take optimizer’s estimates for $N$ values
Exploiting Execution Feedback for Refining Estimates

Motivating example

Note: A is the build relation, B is the probe relation
Associate two additional values $LB_i$, $UB_i$ (upper and lower bounds of cardinalities of $i^{th}$ node)

The invariant: $LB_i \leq \text{current estimate of } N_i \leq UB_i$

Whenever $N_i$ is found outside the bounds, adjust it to appropriate bound.

These bounds are solely dependent on algebraic properties of operators.
To refine lower, upper bounds of $N_i$, the following info. is used:

- input, output cardinalities of the operator (i.e. $K_i$ of the operator as well as its input operators)
- Algebraic properties of the operator
- The current state of the operator
For refining lower bounds, $K_i$ is itself a correct lower bound.

Current state of group-by (hash) operator: the number of distinct values observed so far would give correct lower bound.

For upper bound of NL join (foreign-key join),
$$UB_i = (UB_{i-1} - K_{i-1}) + K_i$$

For Sort, $UB_i = UB_{i-1}$

For hash join (A join B), let $s$ is the number of tuples in largest bucket, then every row in probe relation can produce at most $s$ tuples. This info, we can use to adjust the upper bound.
Implementation

- The datastructure corresponding to a node augmented with counters for $K_i$, $N_i$, $LB_i$, $UB_i$
- Identify the pipelines of plan $P$, given by optimizer
- Identify the driver nodes
- Initialize the $K_i$ to zero and $N_i$ top optimizer estimates
- Update $K_i$ at each node (in GetNext() )
- Compute the progress of query periodically and log (possibly in a file)
- Client may be modified to read the progress from log and display.
### Experiments

<table>
<thead>
<tr>
<th>Query</th>
<th>Estimation Error ($Z=0$)</th>
<th>Estimation Error ($Z=2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Max</td>
</tr>
<tr>
<td>Q1</td>
<td>0.9%</td>
<td>2.8%</td>
</tr>
<tr>
<td>Q3</td>
<td>1.1%</td>
<td>2.0%</td>
</tr>
<tr>
<td>Q4</td>
<td>0.5%</td>
<td>1.0%</td>
</tr>
<tr>
<td>Q5</td>
<td>7.3%</td>
<td>9.0%</td>
</tr>
<tr>
<td>Q6</td>
<td>1.2%</td>
<td>2.9%</td>
</tr>
<tr>
<td>Q7</td>
<td>2.3%</td>
<td>4.0%</td>
</tr>
<tr>
<td>Q8</td>
<td>0.8%</td>
<td>1.7%</td>
</tr>
<tr>
<td>Q9</td>
<td>2.7%</td>
<td>4.9%</td>
</tr>
<tr>
<td>Q10</td>
<td>0.4%</td>
<td>1.4%</td>
</tr>
<tr>
<td>Q12</td>
<td>1.0%</td>
<td>1.7%</td>
</tr>
<tr>
<td>Q14</td>
<td>0.5%</td>
<td>1.8%</td>
</tr>
<tr>
<td>Q15</td>
<td>0.6%</td>
<td>1.3%</td>
</tr>
<tr>
<td>Q17</td>
<td>1.7%</td>
<td>2.6%</td>
</tr>
<tr>
<td>Q18</td>
<td>5.9%</td>
<td>16.8%</td>
</tr>
<tr>
<td>Q19</td>
<td>0.5%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Q20</td>
<td>3.0%</td>
<td>9.8%</td>
</tr>
<tr>
<td>Q21</td>
<td>0.9%</td>
<td>2.5%</td>
</tr>
</tbody>
</table>
Plot of actual vs. estimated percentage completed (TPC-H Q8)
Validation of Driver Node Hypothesis
Conclusion and Future Work

- Estimating the time remaining (only % of completion is achieved by this paper).
- Providing more granular info (per every operator) to users.
References