Estimating Progress of Execution for SQL Queries

Santosh Kumar C

March 28, 2006

Introduction and Motivation

- What is a progress estimator?
- How is it useful?
  - End user
  - DBA
  - Query level scheduling
  - Query dependent server timeouts (k*est-time as opposed to const time outs)
- What is the challenge?

Definitions

- execution plan
- blocking operator
- pipeline
- driver node

Recognizing the pipelines in Query plan

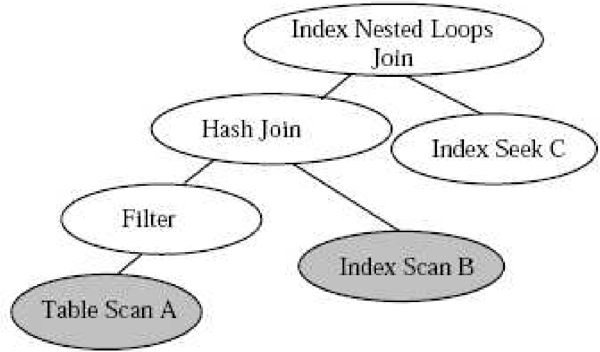
Execute the following steps in bottom-up manner on query plan

- 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 a 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



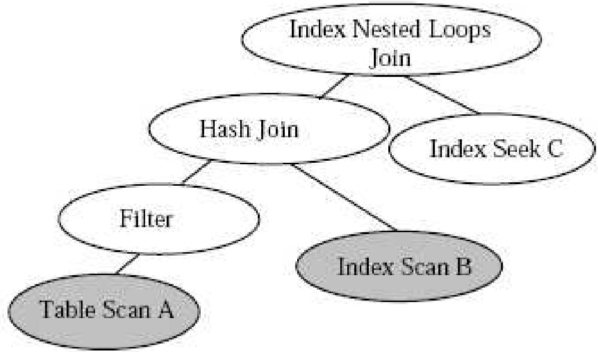
Note: A is the build relation, B is the probe relation

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

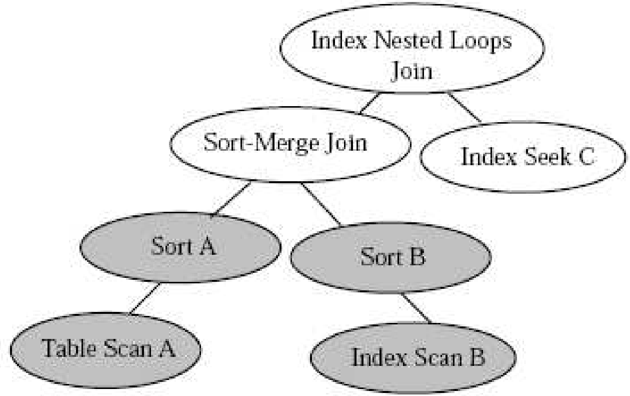
- Every pipeline has a set of driver nodes, i.e., operators that are the sources of tuples operated upon by remaining nodes in the pipeline.
- More precisely, we define the driver nodes of a pipeline as the set of all leaf nodes of the pipeline, except those that are in the inner subtree of a Nested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

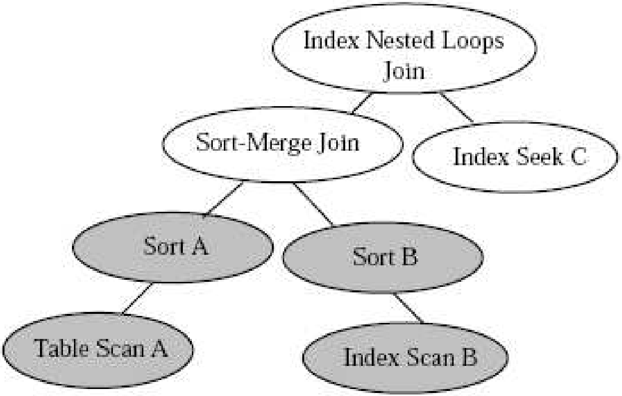
It is possible for a pipeline to contain more than one driver node.

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

The GetNext Model of Work (GNM)

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 i^{th} operator

N_i is the number of GetNext() calls made on 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 :

$Op_1 \longrightarrow Op_2 \longrightarrow Op_3 \longrightarrow \ldots \longrightarrow Op_m$

$dne = \frac{K_1}{N_1}$

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
  - $\frac{gnm}{m} \leq dne \leq m.gnm$

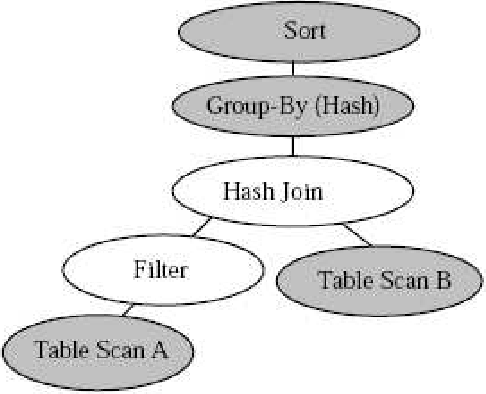
Estimator for arbitrary query plans

For a execution plan with s pipelines:
$gnm = \frac{\sum_{P1} K + \ldots \sum_{Ps} K}{\sum_{P1} N + \ldots \sum_{Ps} N}$ We always know, K values accurately

- If Pi is completed, then $\sum_{Pi} N = \sum_{Pi} K$
- If Pi is currently executing, then $\sum_{Pi} N = \frac{\sum_{Pi} K}{dne}$
- If Pi 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

Exploiting Execution Feedback for Refining Estimates (contd..)

- Associate two additional values LB_i, UB_i (upper and lower bounds of cardinalities of i^th node)
- The invariant: $LB_i \leq currentestimateofN_i \leq UB_i$
- Whenever N_i is found outside the bounds, adjust it to appropriate bound.
- These bounds are solely dependent on algebraic properties of operators

Exploiting Execution Feedback for Refining Estimates (contd..)

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

Exploiting Execution Feedback for Refining Estimates (contd..)

- 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

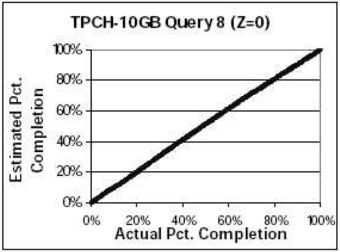
- The datastructure corresponding to a node augmented with counters for K_i , N_i , LB_i , UB_i
- Identify the pipelines of plan P, given by optimizer
- Identify the driver nodes
- Initialize the K_i to zero and N_i top optimizer estimates
- Update K_i at each node (in GetNext())
- Compute the progress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

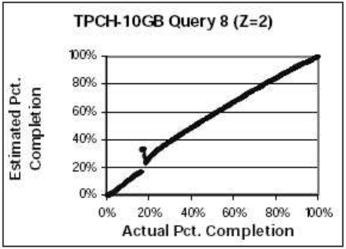
Experiments

Estimation Errors TPC-H Benchmark Queries

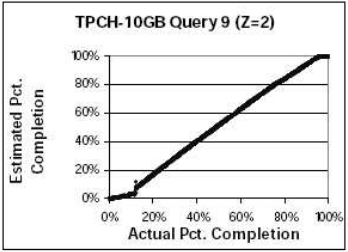
	Estimation Error(Z=0)		Estimation Error (Z=2)	
Query	Mean	Max	Mean	Max
Q1	0.9%	2.8%	0.2%	0.5%
Q3	1.1%	2.0%	3.4%	4.7%
Q4	0.5%	1.0%	0.6%	1.4%
Q5	7.3%	9.0%	3.7%	5.4%
Q6	1.2%	2.9%	2.8%	4.6%
Q7	2.3%	4.0%	3.8%	7.6%
Q8	0.8%	1.7%	5.2%	16.2%
Q9	2.7%	4.9%	2.9%	8.3%
Q10	0.4%	1.4%	1.6%	4.4%
Q12	1.0%	1.7%	0.9%	3.8%
Q14	0.5%	1.8%	1.5%	3.2%
Q15	0.6%	1.3%	1.6%	4.4%
Q17	1.7%	2.6%	0.7%	2.0%
Q18	5.9%	16.8%	14.2%	25.5%
Q19	0.5%	1.5%	1.8%	2.7%
Q20	3.0%	9.8%	3.7%	5.9%
Q21	0.9%	2.5%	15.7%	38.8%

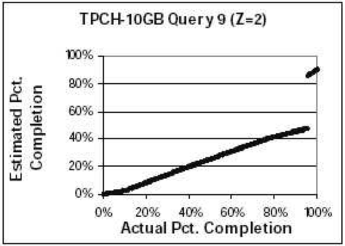
Plot of actual vs. estimated percentage completed (TPC-H Q8)





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

Estimating Progress of Execution of SQL Queries, Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy, SIGMOD 2004.