An Overview of Query Optimization in Relational Systems

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com
http:/research.microsoft.com/~surajitc

Multi-Block Queries

- Multi-block structure arises due to
  - views with aggregates
  - table expressions
  - nested sub-queries
- Techniques for Optimization
  - Merge into a single block
  - Share information across blocks

Example of A Nested Subquery

```
Select Emp.Name
From Emp
Where Emp. Age < 30
        And Emp.Dept# IN
                (Select Dept.Dept#
                 From Dept
                Where Dept.Loc = "Denver"
                AND Emp.Emp# = Dept.Mgr)
```


Merging Nested Subquery

```
Select Emp.Name
From Emp
Where Emp.age < 30 And Emp.Dept# IN
(Select Dept.Dept#
From Dept
Where Dept.Loc = "Denver"
And Emp.Emp# = Dept.Mgr)
```

- Think of "IN" as a Join between Emp and Dept
  ON {Emp.Dept# = Dept.Dept# ,
  Emp.Emp# = Dept.Mgr}

Equivalent Single Block Query

- Select Emp.Name
- From Emp, Dept
- Where Emp.Age < 30
- Emp.Dept# = Dept.Dept#
- And Emp.Emp# = Dept.Mgr
- And Dept.Loc = "Denver"

Nested Subquery with Aggregates

```
Select D.Name
From Dept D
Where D.parking < =
        (Select count (E.Emp#)
        From Emp E
        Where E.Dept# = D. Dept #)
```

Merging Nested Subqueries

- Results in a left outerjoin between the parent and the child block (preserves tuples of the parent)
- Outerjoin reduces to a join for sum(), average(), max(), min()
- Transformed Query:

```
Select D.Name                    Select D.name
From Dept D                      From Dept D LOJ Emp E
Where D.parking <                ON (E.Dept# = D.Dept#)
Select count(E.Emp#)             Group By D.#
From Emp E                       Having D.parking
Where E.Dept# = D. Dept #        < count(E.Emp#)
```


Optimization Across Blocks

- Collapsing into a single block query is not always feasible or beneficial
- We can still optimize by "sideways information passing" across blocks
- Idea similar to semi-join

Semi-Join

- Proposed for optimizing distributed queries
  - Operator introduction
  - Sideways Information Passing (SIP)
- Semi-Join (R,S', P): Join that preserves all attributes of R
  - Apply local selection on S
  - Transmit projection of result (S') to R
  - R' = Semi-Join (R,S',P)
  - Join (R,Sel(S), P) = Join (R, Semi-Join (R, Sel(S), P))

Exploiting Semi-Join

- Outer provides inner with a list of potentially required bindings ("sideways information passing")
- Helps restrict inner's computation
- "Once only" invocation of inner for each binding

Example: A Query with a View

```
Create View DepAvgSal as
(Select E.did, Avg(E.Sal) as avgsal
From Emp E
Group By E.did)

Select E.eid, E.sal
From Emp E, Dept D, DepAvgsal V
Where E.did = D.did And D.did = V.did
And E.age < 30 and D.budget > 100k
And E.sal > V.avgsal
```


Example: Use of SIP

```
Select E.eid, E.sal
From Emp E, Dept D, DepAvgsal V
Where E.did = D.did
And E.did = V.did
And E.age < 30 and D.budget > 100k
And E.sal > V.avgsal
```

- DepAvgsal needs to be evaluated only for cases where V.did IN

```
Select E.did
From Emp E, Dept D
Where E.did = D.did
And E.age < 30 and D.budget > 100k
```

Example: Result of SIP

Supporting Views

A) Create view ED as (Select E.eid, E.did, E.sal
From Emp E, Dept D
Where E.did = D.did
And E.age < 30 and D.budget > 100k)

B) Create View LAvgSal as (
Select E.did, Avg(E.Sal) as avgsal
From Emp E, ED
Where E.did = ED.did
Group By E.did)

Transformed Query

Select ED.eid, ED.sal
From ED, Lavgsal
Where E.did = ED.did and ED.sal > Lavgsal.avgsal

Predicate Propagation Across Blocks

- Q = Join (A, B)
  - Selection on Q can translate to selections on view A
- Q = Intersect (A, B)
  - Selection on A can translate into a selection on B
- Yet another use of SIP

Comments on Multi-Block Transformations

- Strong Synergy
  - Nested Sub-query => Single Block transformations result in J/OJ expressions
  - SIP (semi-join) techniques result in use of "extra" views
  - Merging views directly related to commuting Group By and Join
- Caveats:
  - SQL semantics make applicability conditions tricky
  - Transformations must be cost based

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - *Statistical Model*
  - Tree-Finder
- Tuning Optimizers
- Active Areas of Research
- Conclusion

Cost Estimation

- Cost of an operator is a function of statistical properties of input streams
- For every operator: Register functions that
  - For given statistical parameters of the input data streams, determine:
    - Cost of the operator node
    - Statistical parameters of the output data stream
- Statistical Parameters: Number of tuples, Number of distinct values
  - For base tables, this information is computed by "run statistics"

Cost Estimates for Scan

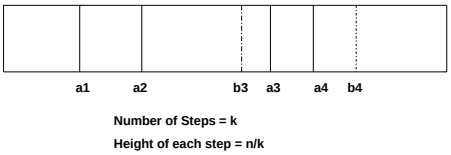
- What to measure?
  - Throughput
  - IO cost + w * CPU cost
  - IO cost = Page Fetches
- Examples of Scan cost
  - S: # of Pages(R)
  - CI: # of Pages(R') + # of Index Pages
  - NCI: # of Tuples(R') + # of Index Pages
- Interesting Issue
  - Effect of database buffers?

Cost Estimates for Join

- Nested Loop Join
  - Cost-of(N1) + Size-of(N1) * Scan-cost(N2)
  - Scan-cost(N2) depends on indexes used
- Sort-Merge Join
  - Sort(N1) + Sort(N2) + Scan(Temp1) + Scan(Temp2)

Histograms

- Statistical Descriptor for a stream



Various Histogram Structures

- Equi-depth:
  - All buckets have same number of values
  - Adjacent values co-located in buckets
- MHIST
  - Groups contiguous sets of frequencies
  - Minimizes variance of the frequency approximation
  - Breakpoints where spreads are maximal
- A General Framework [PIHS96]
  - Assign a metric to each value
  - How to partition the metric space?
  - What information is kept for each bucket?
  - What assumptions are made of values within a bucket?

Histograms for Output Streams

- Filter
  - Filter acts as a mask
  - Interpolate count in a partial bucket using uniformity assumption
  - Filter with host variables hard to handle
  - Filter Expressions:
    - F(P1 AND P2) = F(P1)* F(P2)
    - F(NOT P1) = 1- F(P1)
- Join
  - "Normalize" two histograms
  - "Join" two histograms
- Shortcoming: Cannot capture correlation

Histograms on Base Tables

- Advantage
  - Optimization aided by available statistics
- Disadvantage
  - Expensive to collect and maintain
  - Trend: "Auto-maintain" statistical descriptors
- Cost of Building: Use sampling?
  - Needs "block" sampling for efficiency
  - Not effective for number of distinct values
  - How sensitive is optimization to accuracy of statistics?

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - Statistical Model
  - *Tree-Finder: System R, Volcano, Starburst*
- Tuning Optimizers
- Active Areas of Research
- Conclusion

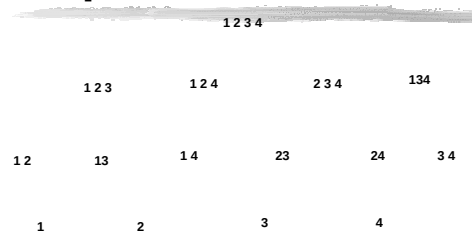
System R "Tree-Finder"

- Need to order joins (linearly)
- Naïve strategy:
  - Generate all n! permutations of joins
- Prohibitively expensive for a large number of joins
  - Overlapping subproblems
  - Ideal for dynamic programming

Use of Dynamic Programming

- Goal: Find the optimal plan for Join(R_1,..R_n, R_{n+1})
  - For each S in {R_1,..R_n, R_{n+1}} do
  - Find Optimal plan for Join(Join(R_1,..R_n), S)
  - Endfor
  - Pick the plan with the least cost
- Principle of Optimality:
  - Optimal plan for a larger expression is derived from optimal plan of one of its sub-expressions

Example



Effect of DP on Complexity

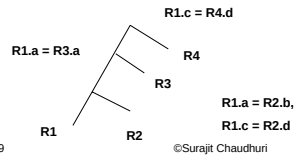
- Enumeration cost drops from O(n!) to O(n2^n)
- May need to store O(2^n) partial plans
- Significantly more efficient than the naïve scheme

Key System-R Tree-Finder Features

- Avoid Cartesian product
  - Defer all Cartesian products as late as possible to avoid "blow-up"
    - Don't consider (R1 X R2) Join R3 if (R1 Join R3) Join R2 is feasible
- Recognize "interesting orders" as violation of principle of optimality:
  - Cost-of(SM (R1,R2)) > Cost-of (NL(R1,R2))
  - But, Cost-of (SM(SM(R1,R2)), R3) may be much less expensive than other alternatives

Handling Interesting Orders in Tree-Finder

- Identify all columns that may exploit sorted order (by examining join predicates)
- Collapse into equivalent groups
- One optimal partial plan for each interesting order
- Example:



Key Ideas from System R

- Cost model based on
  - Access methods
  - Size and cardinality of relations
- Enumeration exploits
  - Dynamic programming
  - One optimal plan for each equivalent expression
  - Violation of principle of optimality handled using interesting order

Limitations of System R

- Limited Transformations
  - Join ordering and choice of access methods only
  - Limited to single block queries
- Weak Cost Model

Outline

- Preliminaries
- Query Optimization Framework
- Building Blocks
  - Equivalence Transformations
  - Statistical Model
  - *Tree-Finder: System R, Volcano, Starburst*
- Tuning Optimizers
- Active Areas of Research
- Conclusion