5 Distributed DBS Query Processing

Overview

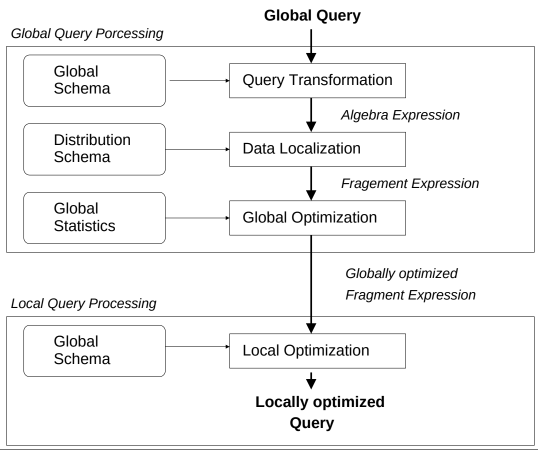
Contents

5.1 Overview

Overview

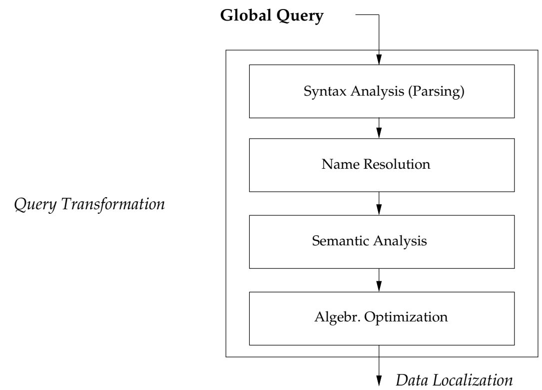
- Goal of query processing: creation of an efficient as possible query plans from a declarative query
  - Transformation to internal format (Calculus $\rightarrow$ Algebra)
  - Selection of access paths (indexes) and algorithms (e.g. Merge-Join vs. Nested-Loops-Join)
  - Cost-based selection of best possible plan
- In Distributed DBS:
  - *User view:* no difference $\rightarrow$ queries are formulated on global schema/external views
  - *Query processing:*
    - Consideration of physical distribution of data
    - Consideration of communication costs

Phases of Query Processing



- Query transformation
  - Translation of SQL to internal representation (Relational Algebra)
  - Name resolution: object names $\rightarrow$ internal names (catalog)
  - Semantic analysis: verification of global relations and attributes, view expansion, global access control
  - Normalization: transformation to canonical format
  - Algebraic optimization: improve "‘efficiency"’ of algebra expression
- Data localization:
  - Identification of nodes with fragments of used relations (from distribution schema)
- Global optimization:
  - Selection of least expensive query plan
  - Consideration of costs (execution and communication, cardinalities of intermediate results)
  - Determination of execution order and place
- Local optimization
  - Optimization of fragment query on each node
  - Using local catalog data (statistics)
  - Usage of index structures
  - Cost-based selection of locally optimal plan
- Code-generation
  - Map query plan to executable code

Query Transformation



Translation to Relational Algebra

```
select A1, ..., Am
from R1, R2, ..., Rn
where F
```

Initial relational algebra expression:

$$\pi_{A_1,\ldots,A_m}(\sigma_F(r(R_1) \times r(R_2) \times r(R_3) \times \cdots \times r(R_n)))$$

Improve algebra expression:

- *Detect joins* to replace Cartesian products
- *Resolution of subqueries* (**`not exists`**-queries to set difference)
- Consider SQL-operations not in relational algebra: (**`group by`**, **`order by`**, arithmetics, ...)

Normalization

- Transform query to unified canonical format to s implify following optimization steps
- Special importance: selection and join conditions (from **`where`**-clause)
  - *Conjunctive normal form* vs. *disjunctive normal form*
  - Conjunktive normal form (CNF) for basic predicates p_{ij}:

    $$(p_{11} \vee p_{12} \vee \cdots \vee p_{1n}) \wedge \cdots \wedge (p_{m1} \vee p_{m2} \vee \cdots \vee p_{mn})$$

  - Disjunctive normal form (DNF):

    $$(p_{11} \wedge p_{12} \wedge \cdots \wedge p_{1n}) \vee \cdots \vee (p_{m1} \wedge p_{m2} \wedge \cdots \wedge p_{mn})$$

  - Transformation according to equivalence rules for logical operations

Normalization /2

- Equivalence rules
  - $p_1 \wedge p_2 \longleftrightarrow p_2 \wedge p_1$ und $p_1 \vee p_2 \longleftrightarrow p_2 \vee p_1$
  - $p_1 \wedge (p_2 \wedge p_3) \longleftrightarrow (p_1 \wedge p_2) \wedge p_3$ und $p_1 \wedge (p_2 \vee p_3) \longleftrightarrow (p_1 \vee p_2) \vee p_3$
  - $p_1 \wedge (p_2 \vee p_3) \longleftrightarrow (p_1 \wedge p2) \vee (p_1 \wedge p_3)$ und $p_1 \vee (p_2 \wedge p_3) \longleftrightarrow (p_1 \vee p2) \wedge (p_1 \vee p_3)$
  - $\neg(p_1 \wedge p_2) \longleftrightarrow \neg p_1 \vee \neg p_2$ und $\neg(p_1 \vee p_2) \longleftrightarrow \neg p_1 \wedge \neg p_2$
  - $\neg(\neg p_1) \longleftrightarrow p_1$

Normalization: Example

- Query:

```
select * from Project P, Assignment A
where P.PNr = A.PNr and
      Budget > 100.000 and
      (Loc = 'MD' or Loc = 'B')
```

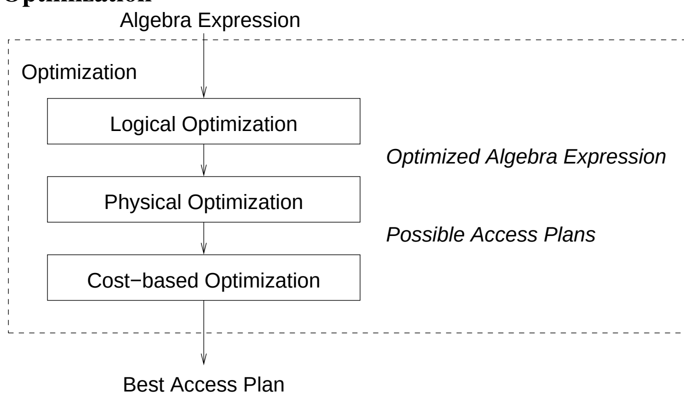
- Selection condition in CNF:

  P.PNr = A.PNr $\wedge$ Budget $>$ 100.000 $\wedge$ (Loc = 'MD' $\vee$ Loc = 'B')

- Selection condition in DNF:

  (P.PNr = A.PNr $\wedge$ Budget $>$ 100.000 $\wedge$ Loc = 'MD') $\vee$
  (P.PNr = A.PNr $\wedge$ Budget $>$ 100.000 $\wedge$ Loc = 'B')

Phases of Optimization



Algebraic Optimization

- Term replacement based on semantic equivalences
- Directed replacement rules to *improve* processing of expression
- Heuristic approach:
  - Move operation to get smaller intermediate results
  - Indentify and remove redundancies
- Result: improved algebraic express $\Rightarrow$ operator tree $\Rightarrow$ initial query plan

Algebraic Rules /1

- Operators σ and $\bowtie$ commute, if selection attribute from one relation:

$$\sigma_F(r_1 \bowtie r_2) \longleftrightarrow \sigma_F(r_1) \bowtie r_2 \qquad \text{falls } attr(F) \subseteq R_1$$

- If selection condition can be split, such that $F = F_1 \wedge F_2$ contain predicates on attributes in only one relation, respectively:

$$\sigma_F(r_1 \bowtie r_2) \longleftrightarrow \sigma_{F_1}(r_1) \bowtie \sigma_{F_2}(r_2)$$

$$\text{if } attr(F_1) \subseteq R_1 \text{ and } attr(F_2) \subseteq R_2$$

- Always: decompose to F_1 with attributes from R_1, if F_2 contains attributes from R_1 and R_2:

$$\sigma_F(r_1 \bowtie r_2) \longleftrightarrow \sigma_{F_2}(\sigma_{F_1}(r_1) \bowtie r_2) \qquad \text{if } attr(F_1) \subseteq R_1$$

Algebraic Rules /2

- Combination of conditions of σ is identical to logical conjunction $\Rightarrow$ operations can change their order

$$\sigma_{F_1}(\sigma_{F_2}(r_1)) \longleftrightarrow \sigma_{F_1 \wedge F_2}(r_1) \longleftrightarrow \sigma_{F_2}(\sigma_{F_1}(r_1))$$

(uses commutativity of logic AND)

Algebraic Rules /3

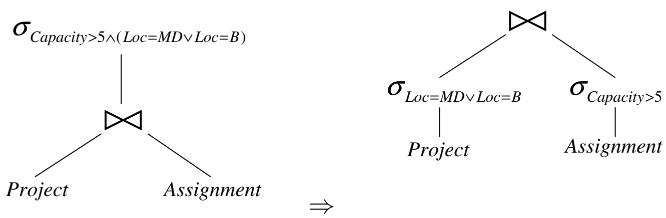
- Operator $\bowtie$ is commutative:
$$r_1 \bowtie r_2 \longleftrightarrow r_2 \bowtie r_1$$
- Operator $\bowtie$ is associative:
$$(r_1 \bowtie r_2) \bowtie r_3 \longleftrightarrow r_1 \bowtie (r_2 \bowtie r_3)$$
- Domination of sequence of π operators:
$$\pi_X(\pi_Y(r_1)) \longleftrightarrow \pi_X(r_1)$$
- π and σ are commutative in some cases:
$$\sigma_F(\pi_X(r_1)) \longleftrightarrow \pi_X(\sigma_F(r_1))$$
$$\text{if } \ attr(F) \subseteq X$$
$$\pi_{X_1}(\sigma_F(\pi_{X_1 X_2}(r_1))) \longleftrightarrow \pi_{X_1}(\sigma_F(r_1))$$
$$\text{if } \ attr(F) \supseteq X_2$$

Algebraic Rules /4

- Commutation of σ and $\cup$:
$$\sigma_F(r_1 \cup r_2) \longleftrightarrow \sigma_F(r_1) \cup \sigma_F(r_2)$$
- Commutation of σ and with other set operation $-$ and $\cap$
- Commutation of π and $\bowtie$ partially possible: join attributes must be kept and later removed (nevertheless decreases intermediate result size)
- Commutation of π und $\cup$
- Distributivity for set operations
- Idempotent expressions, e.g. $r_1 \bowtie r_1 = r_1$ and $r_1 \cup r_1 = r1$
- Operations with empty relations, e.g. $r_1 \cup \emptyset = r_1$
- Commutativity of set operations
- ...

Algebraic Optimization: Example

```
select * from Procekt P, Assignment A
where P.PNr = A.PNr and
       Capacity > 5 and
       (Loc = 'MD' or Loc = 'B')
```

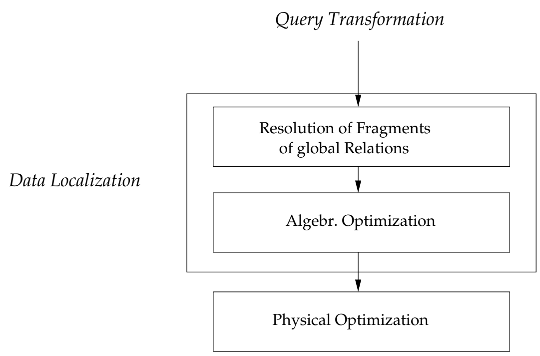


5.2 Data Localization

Data Localization

- Task: create fragment queries based on data distribution
  - Replace global relation with fragments
  - Insert reconstruction expression using fragments of global relation

Data Localization Phase



Data Localization: Example I

- Schema:

  $\text{PROJ}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJEKT})$
  $\text{PROJ}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECT})$
  $\text{PROJ}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECT})$

  $\text{PROJECT} = \text{PROJ}_1 \cup \text{PROJ}_2 \cup \text{PROJ}_3$

- Query: $\sigma_{\text{Loc='MD'} \wedge \text{Budget} \leq 100.000}(\text{PROJECT})$ [1ex] $\Longrightarrow \sigma_{\text{Loc='MD'} \wedge \text{Budget} \leq 100.000}(\text{PROJ}_1 \cup \text{PROJ}_2 \cup \text{PROJ}_3)$

Data Localization /2

- Requirement: further simplification of query
- Goal: eliminate queries on fragments not used in query
- Example: pushing down σ to fragments

  $\sigma_{\text{Loc='MD'} \wedge \text{Budget} \leq 100.000}(\text{PROJ}_1 \cup \text{PROJ}_2 \cup \text{PROJ}_3)$[1ex] because of: $\sigma_{\text{Budget} \leq 100.000}(\text{PROJ}_2) = \emptyset$, $\sigma_{\text{Budget} \leq 100.000}(\text{PROJ}_3) = \emptyset$[1ex]
  $\Longrightarrow \sigma_{\text{Loc='MD'}}(\sigma_{\text{Budget} \leq 100.000}(\text{PROJ}_1))$

Data Localization /3

- For horizontal fragmentation
  - Also possible simplification of join processing
  - Push down join if fragmentation on join attribute

Data Localization: Example II

- Schema:

  $\mathrm{M}_1 = \sigma_{\mathrm{MNr}<\text{'M3'}}(\textsc{Member})$
  $\mathrm{M}_2 = \sigma_{\text{'M3'}\leq\mathrm{MNr}<\text{'M5'}}(\textsc{Member})$
  $\mathrm{M}_3 = \sigma_{\mathrm{MNr}\geq\text{'M5'}}(\textsc{Member})$

  $\mathrm{Z}_1 = \sigma_{\mathrm{MNr}<\text{'M3'}}(\textsc{Assignment})$
  $\mathrm{Z}_2 = \sigma_{\mathrm{MNr}\geq\text{'M3'}}(\textsc{Assignment})$

- Query: $\textsc{Assignment} \bowtie \textsc{Member}$[1ex] $\Longrightarrow$ $(\mathrm{M}_1\cup \mathrm{M}_2\cup \mathrm{M}_3) \bowtie (\mathrm{Z}_1\cup \mathrm{Z}_2)$
  $\Longrightarrow$ $(\mathrm{M}_1 \bowtie \mathrm{Z}_1) \cup (\mathrm{M}_2 \bowtie \mathrm{Z}_2) \cup (\mathrm{M}_3 \bowtie \mathrm{Z}_2)$

Data Localization /4

- Vertical fragmentation: reduction by pushing down projections
- Example:

  $\textsc{Proj}_1 = \pi_{\mathrm{PNr,PName,Loc}}(\textsc{Project})$
  $\textsc{Proj}_2 = \pi_{\mathrm{PNr,Budget}}(\textsc{Project})$

  $\textsc{Project} = \textsc{Proj}_1 \bowtie \textsc{Proj}_2$

- Query: $\pi_{\mathrm{PName}}(\textsc{Project})$[1ex] $\Longrightarrow \pi_{\mathrm{PName}}(\textsc{Proj}_1 \bowtie \textsc{Proj}_2) \Longrightarrow \pi_{\mathrm{PName}}(\textsc{Proj}_1)$

Qualified Relations

- Descriptive information to support algebraic optimization
- Annotation of fragments and intermediate results with content condition (combination of predicates that are satisfied here)
- Estimation of size of relation
- If $r' = Q(r)$, then r' *inherits* condition from r, plus additional predicates from Q
- Qualification condition q_R: $[R : q_R]$
- Extended relational algebra: $\sigma_F[R : q_R]$

Extended Relational Algebra

$$
\begin{array}{lll}
(1) & E := \sigma_F[R : q_R] & \rightarrow [E : F \wedge q_R] \\
(2) & E := \pi_A[R : q_R] & \rightarrow [E : q_R] \\
(3) & E := [R : q_R] \times [S : q_S] & \rightarrow [E : q_R \wedge q_S] \\
(4) & E := [R : q_R] - [S : q_S] & \rightarrow [E : q_R] \\
(5) & E := [R : q_R] \cup [S : q_S] & \rightarrow [E : q_R \vee q_S] \\
(6) & E := [R : q_R] \bowtie_F [S : q_S] & \rightarrow [E : q_R \wedge q_S \wedge F]
\end{array}
$$

Extended Relational Algebra /2

- Usage of rules for *description* – no processing
- Example: $\sigma_{100.000 \leq \text{Budget} \leq 200.000}(\text{PROJECT})$

$$
\begin{array}{rcl}
E_1 & = & \sigma_{100.000 \leq \text{Budget} \leq 200.000}[\text{PROJ}_1 : \text{Budget} \leq 150.000] \\
 & \leadsto & [E_1 : (100.000 \leq \text{Budget} \leq 200.000) \wedge (\text{Budget} \leq 150.000)] \\
 & \leadsto & [E_1 : 100.000 \leq \text{Budget} \leq 150.000] \\
E_2 & = & \sigma_{1000 \leq \text{Budget} \leq 200.000}[\text{PROJ}_2 : 150.000 < \text{Budget} \leq 200.000] \\
 & \leadsto & [E_2 : (100.000 \leq \text{Budget} \leq 200.000) \wedge \\
 & & (150.000 < \text{Budget} \leq 200.000)] \\
 & \leadsto & [E_2 : 150.000 < \text{Budget} \leq 200.000] \\
E_3 & = & \sigma_{100.000 \leq \text{Budget} \leq 200.000}[\text{PROJ}_3 : \text{Budget} > 200.000] \\
 & \leadsto & [E_3 : (100.000 \leq \text{Budget} \leq 200.000) \wedge (\text{Budget} > 200.000)] \\
 & \leadsto & E_3 = \emptyset
\end{array}
$$

5.3 Join Processing

Join Processing

- Join operations:
  - Common task in relational DBS, very expensive ($\leq O(n^2)$)
  - In distributed DBS: join of nodes stored on different nodes
- Simple strategy: process join on one node
  - *Ship whole*: transfer the full relation beforehand
  - *Fetch as needed*: request tuples for join one at a time

"‘Fetch as needed "’ vs. "‘Ship whole"’ /1

R

A	B
3	7
1	1
4	6
7	7
4	5
6	2
5	7

S

B	C	D
9	8	8
1	5	1
9	4	2
4	3	3
4	2	6
5	7	8

$R \bowtie S$

A	B	C	D
1	1	5	1
4	5	7	8

Strategy	#Messages	#Values
SW at R-node	2	18
SW at S-node	2	14
SW at 3. node	4	32
FAN at S-node	$6 * 2 = 12$	$6 + 2 * 2 = 10$
FAN at R-node	$7 * 2 = 14$	$7 + 2 * 3 = 13$

"‘Fetch as needed "’ vs. "‘Ship whole"’ /2

- Comparison:
  - "‘Fetch as needed"’ with higher number of messages, useful for small left hand-side relation (e.g. restricted by previous selection)
  - "‘Ship whole"’ with higher data volume, useful for smaller right hand-side (transferred) relation
- Specific algorithms for both:
  - Nested-Loop Join
  - Sort-Merge Join
  - *Semi-Join*
  - *Bit Vector-Join*

Nested-Loop Join

Nested loop over all tuples $t_1 \in r$ and all $t_2 \in s$ for operation $r \bowtie s$

$r \bowtie_\varphi s$:
```
for each t_r ∈ r do
begin
    for each t_s ∈ s do
    begin
        if φ(t_r, t_s) then put(t_r · t_s) endif
    end
end
```

Sort Merge-Join

$X := R \cap S$; if not yet sorted, first sort r and s on join attributes X

1. $t_r(X) < t_s(X)$, read next $t_r \in r$
2. $t_r(X) > t_s(X)$, read next $t_s \in s$
3. $t_r(X) = t_s(X)$, join t_r with t_s and all subsequent tuples to t_s equal regarding X with t_s
4. Repeat for the first $t'_s \in s$ with $t'_s(X) \neq t_s(X)$ starting with original t_s and following t'_r of t_r until $t_r(X) = t'_r(X)$

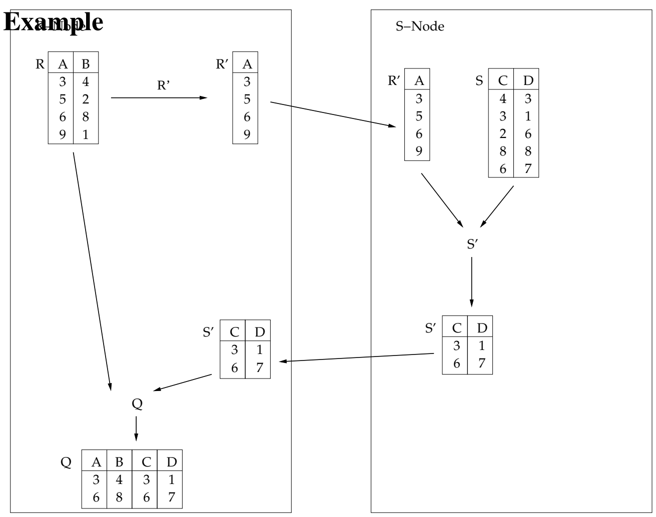
Sort Merge-Join: Costs

- Worst case: all tuples with identical X-values: $O(n_r * n_s)$
- X keys of R or S: $O(n_r \log n_r + n_s \log n_s)$
- If relations are already sorted (e.g. index on join attributes, often the case): $O(n_r + n_s)$

Semi-Join

- Idea: request join partner tuples in one step to minimize message overhead (combines advantages of SW and FAN)
- Based on: $r \bowtie s = r \bowtie (s \ltimes r) = r \bowtie (s \bowtie \pi_A(r))$ (A is set of join attributes)
- Procedure:
  1. Node N_r: computation of $\pi_A(r)$ and transfer to N_s
  2. Node N_s: computation of $s' = s \bowtie \pi_A(r) = s \ltimes r$ and transfer to N_r
  3. Node N_r: computation of $r \bowtie s' = r \bowtie s$

Semi-Join: Example



Bit Vector-Join

- Bit Vector or Hash Filter-Join
- Idea: minimize request size (semi-join) by mapping join attribute values to bit vector $B[1 \dots n]$
- Mapping:
  - Hash function h maps values to buckets $1 \dots n$
  - If value exists in bucket according bit is set to 1

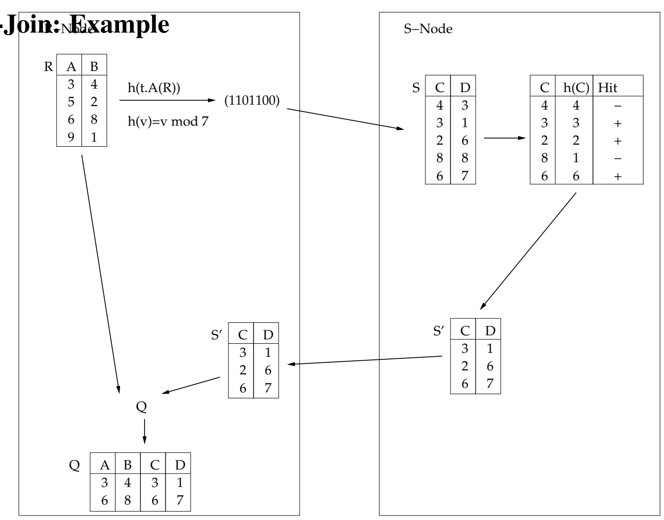
Bit Vector-Join /2

- Procedure:
  1. Node N_r: for each value v in $\pi_A(r)$ set according bit in $B[h(v)]$ and transfer bit vector B to N_s
  2. Node N_s: compute $s' = \{t \in s \mid B[h(t.A)]$ is set $\}$ and transfer to N_r
  3. Node N_r: compute $r \bowtie s' = r \bowtie s$

Bit Vector-Join /3

- Comparison:
  - Decreased size of request message compared to semi-join
  - Hash-mapping not injective $\rightarrow$ only potential join partners in bit vector $\rightsquigarrow$ sufficiently great n and suitable hash function h required

Bit Vector-Join: Example

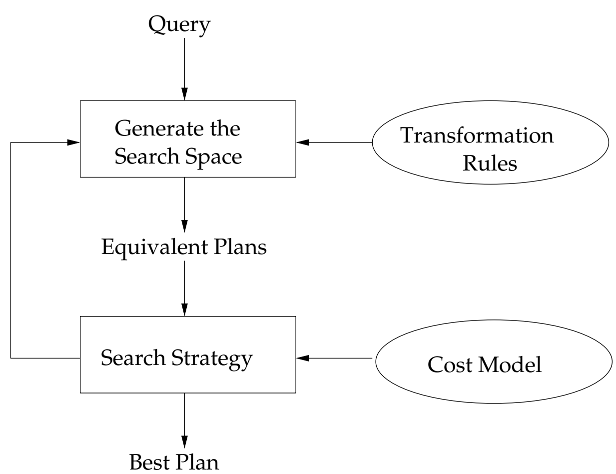


5.4 Global Optimization

Global Optimization

- Task: selection of most cost-efficient plan from set of possible query plans
- Prerequisite: knowledge about
  - Fragmentation
  - Fragment and relation sizes
  - Value ranges and distributions
  - Cost of operations/algorithms
- In Distributed DBS often details for nodes not known:
  - Existing indexes, storage organization, ...
  - Decision about usage is task of local optimization

Cost-based optimization: Overview

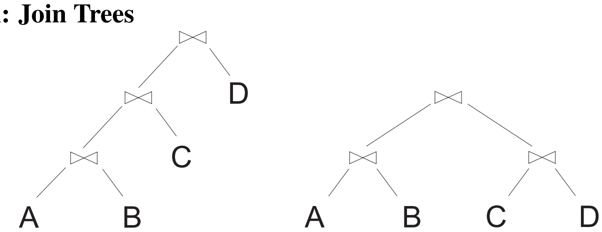


Optimization: Search Space

- Search space: set of all equivalent query plans
- Generated by transformation rules:
  - Algebraic rules with no preferred direction, e.g. join commutativity and associativity (join trees)
  - Assignment of operation implementation/algorithm, e.g. distributed join processing
  - Assignment of operations to nodes

- Constraining the search space
  - Heuristics (like algebraic optimization)
  - Usage of "‘preferred"’ query plans (e.g. pre-defined join trees)

Optimization: Join Trees



- *Left deep trees* or *right deep trees* $\rightsquigarrow$ join order as nested structure/loops, all inner nodes (operations) have at least one input relation
- *Bushy trees* $\rightsquigarrow$ better potential for parallel processing, but higher optimization efforts required (greater number of possible alternatives)

Optimization: Search Strategy

- Traversing the search space and selection of best plan based on cost model:
  - *Which plans* are considered: full or partial traversal
  - In *which order* are the alternatives evaluated
- Variants:
  - **Deterministic**: systematic generation of plans as bottom up construction, simple plans for access to base relations are combined to full plans, grants best plan, computationally complex (e.g. dynamic programming)
  - **Random-based**: create initial query plan (e.g. with greedy strategy or heuristics) and improve these by randomly creating "‘neighbors"’, e.g. exchanging operation algorithm or processing location or join order, less expensive (e.g. genetic algorithms) but does not grant best plan

Cost Model

- Allows comparison/evaluation of query plans
- Components
  - Cost function
    - Estimation of costs for operation processing
  - Database statistics
    - Data about relation sizes, value ranges and distribution
  - Formulas
    - Estimation of sizes of intermediate results (input for operations)

Cost Functions

- **Total time**
  - Sum of all time components for all nodes / transfers

$$\begin{aligned} T_{\text{total}} = & T_{\text{CPU}} * \textit{\#insts} + T_{\text{I/O}} * \textit{\#I/Os} + \\ & T_{\text{MSG}} * \textit{\#msgs} + T_{\text{TR}} * \textit{\#bytes} \end{aligned}$$

  - Communication time:

$$CT(\textit{\#bytes}) = T_{\text{MSG}} + T_{\text{TR}} * \textit{\#bytes}$$

  - Coefficients characteristic for Distributed DBS:
  - WAN: communication time (T_{MSG}, T_{TR}) dominates
  - LAN: also local costs (T_{CPU}, $T_{\text{I/O}}$) relevant

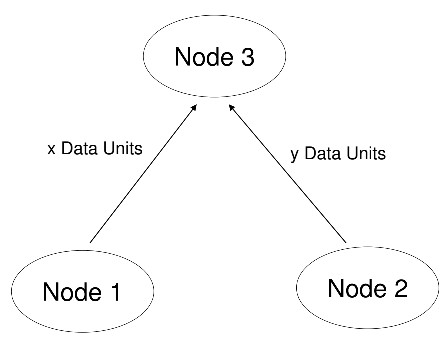
Cost Functions /2

- **Response time**
  - Timespan from initiation of query until availability of full results

$$\begin{aligned} T_{\text{total}} = & T_{\text{CPU}} * \textit{seq\_\#insts} + T_{\text{I/O}} * \textit{seq\_\#I/Os} + \\ & T_{\text{MSG}} * \textit{seq\_\#msgs} + T_{\text{TR}} * \textit{seq\_\#bytes} \end{aligned}$$

  - With *seq_#x* is maximum number *x* that must be performed sequentially

Total Time vs. Response Time



$$\begin{aligned} T_{\text{total}} &= 2T_{\text{MSG}} + T_{\text{TR}}(x + y) \\ T_{\text{response}} &= \max\{T_{\text{MSG}} + T_{\text{TR}} * x, T_{\text{MSG}} + T_{\text{TR}} * y\} \end{aligned}$$

Database statistics

- Main factor for costs: size of intermediate results
- Estimation of sizes based on *statistics*
- For relation R with attributes $A_1, \ldots, A_n$ and fragments $R_1, \ldots, R_f$
  - Attribute size: $\mathrm{length}(A_i)$ (in Byte)
  - Number of distinct values of A_i for each fragment R_j: $\mathrm{val}(A_i, R_j)$
  - Min and max attribute values: $\min(A_i)$ and $\max(A_i)$
  - Cardinality of value domain of A_i: $\mathrm{card}(\mathrm{dom}(A_i))$
  - Number of tuples in each fragment: $\mathrm{card}(R_j)$

Cardinality of Intermediate Results

- Estimation often based on following simplifications
  - Independence of different attributes
  - Equal distribution of attribute values
- Selectivity factor *SF*:
  - Ratio of result tuples vs. input relation tuples
  - Example: $\sigma_F(R)$ returns 10% of tuples from $R \rightsquigarrow$ *SF*$= 0.1$
- Size of an intermediate relation:

$$\mathrm{size}(R) = \mathrm{card}(R) * \mathrm{length}(R)$$

Cardinality of Selections

- Cardinality

$$\mathrm{card}(\sigma_F(R)) = \mathit{SF}_S(F) * \mathrm{card}(R)$$

- SF depends on selection condition with predicates $p(A_i)$ and constants v

$$\begin{aligned}
\mathit{SF}_S(A = v) &= \frac{1}{\mathrm{val}(A, R)} \\
\mathit{SF}_S(A > v) &= \frac{\max(A) - v}{\max(A) - \min(A)} \\
\mathit{SF}_S(A < v) &= \frac{v - \min(A)}{\max(A) - \min(A)}
\end{aligned}$$

Cardinality of Selections /2

$$\begin{aligned}
SF_S(p(A_i) \wedge p(A_j)) &= SF_S(p(A_i)) * SF_S(p(A_j)) \\
SF_S(p(A_i) \vee p(A_j)) &= SF_S(p(A_i)) + SF_S(p(A_j)) - \\
&\quad (SF_S(p(A_i)) * SF_S(p(A_j))) \\
SF_S(A \in \{v_1, \ldots, v_n\}) &= SF_S(A = v) * \mathrm{card}(\{v_1, \ldots, v_n\})
\end{aligned}$$

Cardinality of Projections

- Without duplicate elimination

$$\mathrm{card}(\pi_A(R)) = \mathrm{card}(R)$$

- With duplicate elimination (for non-key attributes A)

$$\mathrm{card}(\pi_A(R)) = \mathrm{val}(A, R)$$

- With duplicate elimination (a key is subset of attributes in A)

$$\mathrm{card}(\pi_A(R)) = \mathrm{card}(R)$$

Cardinality of Joins

- Cartesian products

$$\mathrm{card}(R \times S) = \mathrm{card}(R) * \mathrm{card}(S)$$

- Join
  - Upper bound: cardinality of Cartesian product
  - Better estimation for foreign key relationships $S.B \rightarrow R.A$:

$$\mathrm{card}(R \bowtie_{A=B} S) = \mathrm{card}(S)$$

  - Selectivity factor SF_J from database statistics

$$\mathrm{card}(R \bowtie S) = SF_J * \mathrm{card}(R) * \mathrm{card}(S)$$

Cardinality of Semi-joins

- Operation $R \ltimes_A S$
- Selectivity factor for attribute A from relation S: $SF_{SJ}(S.A)$

$$SF_{SJ}(R \ltimes_A S) = \frac{\mathrm{val}(A, S)}{\mathrm{card}(\mathrm{dom}(A))}$$

- Cardinality:

$$\mathrm{card}(R \ltimes_A S) = SF_{SJ}(S.A) * \mathrm{card}(R)$$

Cardinality of Set Operations

- Union $R \cup S$
  - Lower bound: $\max\{\mathrm{card}(R), \mathrm{card}(S)\}$
  - Upper bound: $\mathrm{card}(R) + \mathrm{card}(S)$
- Set difference $R - S$
  - Lower bound: 0
  - Upper bound: $\mathrm{card}(R)$

Example

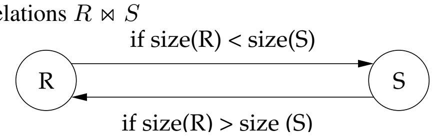
- Fragmentation: $\textsc{Project} = \textsc{Project}_1 \cup \textsc{Project}_2 \cup \textsc{Project}_3$
- Query:
$$\sigma_{\text{Budget}>150.000}(\textsc{Project})$$
- Statistics:
  - $\mathrm{card}(\textsc{Project}_1) = 3.500, \mathrm{card}(\textsc{Project}_2) = 4.000, \mathrm{card}(\textsc{Project}_3) = 2.500$
  - $\mathrm{length}(\textsc{Project}) = 30$
  - $\min(\text{Budget}) = 50.000, \max(\text{Budget}) = 300.000$
  - $T_{\text{MSG}} = 0.3s$
  - $T_{\text{TR}} = 1/1000s$

Example: Query Plans

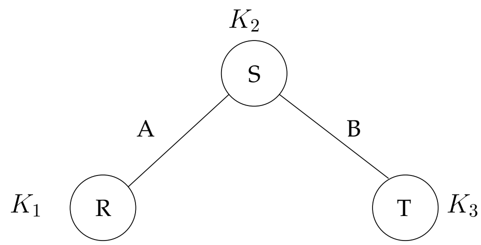
- Variant 1:
$$\sigma_{\text{Budget}>150.000}(\textsc{Project}_1 \cup \textsc{Project}_2 \cup \textsc{Project}_3)$$
- Variant 2:
$$\begin{gathered}\sigma_{\text{Budget}>150.000}(\textsc{Project}_1)\cup\\ \sigma_{\text{Budget}>150.000}(\textsc{Project}_2)\cup\\ \sigma_{\text{Budget}>150.000}(\textsc{Project}_3)\end{gathered}$$

Join Order in DDBS

- Huge influence on overall performance
- General rule: avoid Cartesian products where possible
- Join order for 2 relations $R \bowtie S$



- Join order for 3 relations $R \bowtie_A S \bowtie_B T$



Join Order in DDBS /2

- (cont.) Possible strategies:
  1. $R \rightarrow N_2$; N_2 computes $R' := R \bowtie S$; $R' \rightarrow N_3$; N_3 computes $R' \bowtie T$
  2. $S \rightarrow N_1$; N_1 computes $R' := R \bowtie S$; $R' \rightarrow N_3$; N_3 computes $R' \bowtie T$
  3. $S \rightarrow N_3$; N_3 computes $S' := S \bowtie T$; $S' \rightarrow N_1$; N_1 computes $S' \bowtie R$
  4. $T \rightarrow N_2$; N_2 computes $T' := T \bowtie S$; $T' \rightarrow N_1$; N_1 computes $T' \bowtie R$
  5. $R \rightarrow N_2$; $T \rightarrow N_2$; N_2 computes $R \bowtie S \bowtie T$
- Decision based on size of relations and intermediate results
- Possible utilization of parallelism in variant 5

Utilization of Semi-Joins

- Consideration of semi-join-based strategies
- Relations R at node N_1 and S at node N_2
- Possible strategies $R \bowtie_A S$
  1. $(R \ltimes_A S) \bowtie_A S$
  2. $R \bowtie_A (S \ltimes_A R)$
  3. $(R \ltimes_A S) \bowtie_A (S \ltimes_A R)$
- Comparison $R \bowtie_A S$ vs. $(R \ltimes_A S) \bowtie_A S)$ for $\text{size}(R) < \text{size}(S)$
- Costs for $R \bowtie_A S$: transfer of R to $N_2 \rightsquigarrow T_{TR} * \text{size}(R)$

Utilization of Semi-Joins /2

- Processing of semi-join variant

  1. $\pi_A(S) \rightarrow N_2$
  2. At node N_2: computation of $R' := R \ltimes_A S$
  3. $R' \rightarrow N_1$
  4. At node N_1: computation of $R' \bowtie_A S$

- Costs: costs for step 1 + costs for step 2

$$T_{TR} * \text{size}(\pi_A(S)) + T_{TR} * \text{size}(R \ltimes_A S)$$

- Accordingly: semi-join is better strategy if

$$\text{size}(\pi_A(S)) + \text{size}(R \ltimes_A S) < \text{size}(R)$$

Summary: Global Optimization in DDBS

- Extension of centralized optimization regarding distribution aspects
  - Location of processing
  - Semi Join vs. Join
  - Fragmentation
  - Total time vs. response time
  - Consideration of additional cost factors like transfer time and number of message messages
- Current system implementations very different regarding which aspects are considered or not