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 → 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 → 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 → 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

58
Query Transformation

Translation to Relational Algebra

\[ \text{select } A_1, \ldots, A_m \]
\[ \text{from } R_1, R_2, \ldots, R_n \]
\[ \text{where } F \]
\[ \pi_{A_1,\ldots,A_m}(\sigma_F(r(R_1) \times r(R_2) \times r(R_3) \times \cdots \times r(R_n))) \]

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 simplify 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 \( p_{ij} \):
    \[ (p_{11} \lor p_{12} \lor \cdots \lor p_{1n}) \land \cdots \land (p_{m1} \lor p_{m2} \lor \cdots \lor p_{mn}) \]
  - Disjunctive normal form (DNF):
    \[ (p_{11} \land p_{12} \land \cdots \land p_{1n}) \lor \cdots \lor (p_{m1} \land p_{m2} \land \cdots \land p_{mn}) \]
- Transformation according to equivalence rules for logical operations
Normalization /2

- Equivalence rules
  - $p_1 \land p_2 \iff p_2 \land p_1 \land p_1 \lor p_2 \iff p_2 \lor p_1$
  - $p_1 \land (p_2 \land p_3) \iff (p_1 \land p_2) \land p_3 \land (p_2 \lor p_3) \iff (p_2 \lor p_3) \lor p_3$
  - $p_1 \land (p_2 \lor p_3) \iff (p_1 \land p_2) \lor (p_1 \land p_3) \land p_1 \lor (p_2 \land p_3) \iff (p_1 \lor p_2) \land (p_1 \lor p_3)$
  - $\neg(p_1 \land p_2) \iff \neg p_1 \lor \neg p_2 \land \neg (p_1 \lor p_2) \iff \neg p_1 \land \neg p_2$
  - $\neg(\neg p_1) \iff p_1$

Normalization: Example

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

- Query:
  - Selection condition in CNF:
    $$P.PNr = A.PNr \land Budget > 100.000 \land (Loc = 'MD' \lor Loc = 'B')$$
  - Selection condition in DNF:
    $$(P.PNr = A.PNr \land Budget > 100.000 \land Loc = 'MD') \lor (P.PNr = A.PNr \land Budget > 100.000 \land Loc = 'B')$$
Phases of Optimization

- Logical Optimization
- Physical Optimization
- Cost-based 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
  - Identify and remove redundancies

Result: improved algebraic expression $\Rightarrow$ operator tree $\Rightarrow$ initial query plan

Algebraic Rules /1

- Operators $\sigma$ and $\bowtie$ commute, if selection attribute from one relation:
  $$\sigma_F(r_1 \bowtie r_2) \iff \sigma_F(r_1) \bowtie r_2 \quad \text{falls } \text{attr}(F) \subseteq R_1$$

- If selection condition can be split, such that $F = F_1 \land F_2$ contain predicates on attributes in only one relation, respectively:
  $$\sigma_F(r_1 \bowtie r_2) \iff \sigma_{F_1}(r_1) \bowtie \sigma_{F_2}(r_2)$$
  if $\text{attr}(F_1) \subseteq R_1$ and $\text{attr}(F_2) \subseteq R_2$

- Always: decompose to $F_1$ with attributes from $R_1$, if $F_2$ contains attributes from $R_1$ and $R_2$:
  $$\sigma_F(r_1 \bowtie r_2) \iff \sigma_{F_2}(\sigma_{F_1}(r_1) \bowtie r_2) \quad \text{if } \text{attr}(F_1) \subseteq R_1$$

Algebraic Rules /2

- Combination of conditions of $\sigma$ is identical to logical conjunction $\Rightarrow$ operations can change their order
  $$\sigma_{F_1}(\sigma_{F_2}(r_1)) \iff \sigma_{F_1 \land F_2}(r_1) \iff \sigma_{F_2}(\sigma_{F_1}(r_1))$$

(uses commutativity of logic AND)
Algebraic Rules /3

- Operator $\Join$ is commutative:
  \[ r_1 \Join r_2 \leftrightarrow r_2 \Join r_1 \]

- Operator $\Join$ is associative:
  \[ (r_1 \Join r_2) \Join r_3 \leftrightarrow r_1 \Join (r_2 \Join r_3) \]

- Domination of sequence of $\pi$ operators:
  \[ \pi_X (\pi_Y (r_1)) \leftrightarrow \pi_X (r_1) \]

- $\pi$ and $\sigma$ are commutative in some cases:
  \[ \sigma_F (\pi_Y (r_1)) \leftrightarrow \pi_X (\sigma_F (r_1)) \]
  \[ \text{if } \text{attr}(F) \subseteq X \]
  \[ \pi_{X_1} (\sigma_F (\pi_{X_1} (r_1))) \leftrightarrow \pi_{X_1} (\sigma_F (r_1)) \]
  \[ \text{if } \text{attr}(F) \supseteq X_2 \]

Algebraic Rules /4

- Commutation of $\sigma$ and $\cup$:
  \[ \sigma_F (r_1 \cup r_2) \leftrightarrow \sigma_F (r_1) \cup \sigma_F (r_2) \]

- Commutation of $\sigma$ and with other set operation $-$ and $\cap$

- Commutation of $\pi$ and $\Join$ partially possible: join attributes must be kept and later removed (nevertheless decreases intermediate result size)

- Commutation of $\pi$ und $\cup$

- Distributivity for set operations

- Idempotent expressions, e.g. $r_1 \Join r_1 = r_1$ and $r_1 \cup r_1 = r_1$

- Operations with empty relations, e.g. $r_1 \cup \emptyset = r_1$

- Commutativity of set operations

- $\ldots$
Algebraic Optimization: Example

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

![Diagram showing the algebraic optimization example]

\[ \sigma_{\text{Capacity} > 5 \land \text{Loc} = \text{'MD' or 'B'}} \]
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: \( \sigma_{\text{Loc} = \text{MD} \land \text{Budget} \leq 100.000}(\text{PROJECT}) \) \( \Rightarrow \) \( \sigma_{\text{Loc} = \text{MD} \land \text{Budget} \leq 100.000}(\text{PROJ}_1 \cup \text{PROJ}_2 \cup \text{PROJ}_3) \)

Data Localization /2

- Requirement: further simplification of query
- Goal: eliminate queries on fragments not used in query
- Example: pushing down \( \sigma \) to fragments

\[
\sigma_{\text{Loc} = \text{MD} \land \text{Budget} \leq 100.000}(\text{PROJ}_1 \cup \text{PROJ}_2 \cup \text{PROJ}_3)[\text{lex}] \text{ because of: } \sigma_{\text{Budget} \leq 100.000}(\text{PROJ}_2) = \emptyset, \sigma_{\text{Budget} \leq 100.000}(\text{PROJ}_3) = \emptyset[\text{lex}] \\
\Rightarrow \sigma_{\text{Loc} = \text{MD} \land \text{Budget} \leq 100.000}(\text{PROJ}_1)
\]
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:
  \[ M_1 = \sigma_{MNr < M3}(\text{MEMBER}) \]
  \[ M_2 = \sigma_{M3 \leq MNr < M5}(\text{MEMBER}) \]
  \[ M_3 = \sigma_{MNr \geq M5}(\text{MEMBER}) \]
  \[ Z_1 = \sigma_{MNr < M3}(\text{ASSIGNMENT}) \]
  \[ Z_2 = \sigma_{MNr \geq M3}(\text{ASSIGNMENT}) \]

- Query: \( \text{ASSIGNMENT} \bowtie \text{MEMBER}[1ex] \Rightarrow (M_1 \cup M_2 \cup M_3) \bowtie (Z_1 \cup Z_2) \Rightarrow (M_1 \bowtie Z_1) \cup (M_2 \bowtie Z_2) \cup (M_3 \bowtie Z_2) \)

Data Localization /4

- Vertical fragmentation: reduction by pushing down projections

- Example:
  \[ \text{PROJ}_1 = \pi_{\text{PNr, PName, Loc}}(\text{PROJECT}) \]
  \[ \text{PROJ}_2 = \pi_{\text{PNr, Budget}}(\text{PROJECT}) \]

  \[ \text{PROJECT} = \text{PROJ}_1 \bowtie \text{PROJ}_2 \]

- Query: \( \pi_{\text{PName}}(\text{PROJECT})[1ex] \Rightarrow \pi_{\text{PName}}(\text{PROJ}_1 \bowtie \text{PROJ}_2) \Rightarrow \pi_{\text{PName}}(\text{PROJ}_1) \)

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

1. \( E := \sigma_F[R : q_R] \quad \rightarrow \quad [E : F \land q_R] \)
2. \( E := \pi_A[R : q_R] \quad \rightarrow \quad [E : q_R] \)
3. \( E := [R : q_R] \times [S : q_S] \quad \rightarrow \quad [E : q_R \land q_S] \)
4. \( E := [R : q_R] \setminus [S : q_S] \quad \rightarrow \quad [E : q_R] \)
5. \( E := [R : q_R] \cup [S : q_S] \quad \rightarrow \quad [E : q_R \lor q_S] \)
6. \( E := [R : q_R] \bowtie_F [S : q_S] \quad \rightarrow \quad [E : q_R \land q_S \land F] \)

Extended Relational Algebra /2

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

\[ E_1 \quad = \quad \sigma_{100.000 \leq \text{Budget} \leq 200.000} (\text{PROJECT}_1 : \text{Budget} \leq 150.000) \]
\[ \leadsto \quad [E_1 : (100.000 \leq \text{Budget} \leq 200.000) \land (\text{Budget} \leq 150.000)] \]
\[ \leadsto \quad [E_1 : 100.000 \leq \text{Budget} \leq 150.000] \]

\[ E_2 \quad = \quad \sigma_{100.000 \leq \text{Budget} \leq 200.000} (\text{PROJECT}_2 : 150.000 < \text{Budget} \leq 200.000) \]
\[ \leadsto \quad [E_2 : (100.000 \leq \text{Budget} \leq 200.000) \land (150.000 < \text{Budget} \leq 200.000)] \]
\[ \leadsto \quad [E_2 : 150.000 < \text{Budget} \leq 200.000] \]

\[ E_3 \quad = \quad \sigma_{100.000 \leq \text{Budget} \leq 200.000} (\text{PROJECT}_3 : \text{Budget} > 200.000) \]
\[ \leadsto \quad [E_3 : (100.000 \leq \text{Budget} \leq 200.000) \land (\text{Budget} > 200.000)] \]
\[ \leadsto \quad E_3 = \emptyset \]
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

```
<table>
<thead>
<tr>
<th>Strategy</th>
<th>#Messages</th>
<th>#Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW at R-node</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>SW at S-node</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>SW at 3. node</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>FAN at S-node</td>
<td>6 * 2 = 12</td>
<td>6 + 2 * 2 = 10</td>
</tr>
<tr>
<td>FAN at R-node</td>
<td>7 * 2 = 14</td>
<td>7 + 2 * 3 = 13</td>
</tr>
</tbody>
</table>
```

"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 \( r \bowtie s \)

for each \( t_r \in r \) do
begin
for each \( t_s \in s \) do
begin
if \( \varphi(t_r, t_s) \) then put \((t_r \cdot t_s)\) 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 \( t'_r(X) \neq t_s(X) \) 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: \( r \bowtie s = r \bowtie (s \bowtie r) = r \bowtie (s \bowtie \pi_A(r)) \) \((A\) 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 \( s' = s \bowtie \pi_A(r) = s \bowtie r \) and transfer to \( N_r \)
  3. Node \( N_r \): computation of \( r \bowtie s' = r \bowtie s \)

**Semi-Join: Example**

---

69
Bit Vector-Join

- Bit Vector or Hash Filter-Join
- Idea: minimize request size (semi-join) by mapping join attribute values to bit vector \( B[1 \ldots n] \)
- Mapping:
  - Hash function \( h \) maps values to buckets \( 1 \ldots n \)
  - 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 \( s' = \{ t \in s \mid B[h(t.A)] \text{ is set} \} \) 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 \( \rightarrow \) 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 \(\rightarrow\) join order as nested structure/loops, all inner nodes (operations) have at least one input relation
- Bushy trees \(\rightarrow\) 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
    \[
    T_{\text{total}} = T_{\text{CPU}} \times \#\text{insts} + T_{\text{I/O}} \times \#\text{I/Os} + \\
    T_{\text{MSG}} \times \#\text{msgs} + T_{\text{TR}} \times \#\text{bytes}
    \]
  - Communication time:
    \[
    CT(\#\text{bytes}) = T_{\text{MSG}} + T_{\text{TR}} \times \#\text{bytes}
    \]
  - Coefficients characteristic for Distributed DBS:
    - WAN: communication time \((T_{\text{MSG}}, T_{\text{TR}})\) dominates
    - LAN: also local costs \((T_{\text{CPU}}, T_{\text{I/O}})\) relevant

Cost Functions /2

- **Response time**
  - Timespan from initiation of query until availability of full results
    \[
    T_{\text{total}} = T_{\text{CPU}} \times \text{seq}_\#\text{insts} + T_{\text{I/O}} \times \text{seq}_\#\text{I/Os} + \\
    T_{\text{MSG}} \times \text{seq}_\#\text{msgs} + T_{\text{TR}} \times \text{seq}_\#\text{bytes}
    \]
  - With \text{seq}_\#x is maximum number \(x\) that must be performed sequentially

**Total Time vs. Response Time**

\[
T_{\text{total}} = 2T_{\text{MSG}} + T_{\text{TR}}(x + y) \\
T_{\text{response}} = \max\{T_{\text{MSG}} + T_{\text{TR}} \times x, T_{\text{MSG}} + T_{\text{TR}} \times y\}
\]
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: $\text{length}(A_i)$ (in Byte)
  - Number of distinct values of $A_i$ for each fragment $R_j$: $\text{val}(A_i, R_j)$
  - Min and max attribute values: $\text{min}(A_i)$ and $\text{max}(A_i)$
  - Cardinality of value domain of $A_i$: $\text{card}(\text{dom}(A_i))$
  - Number of tuples in each fragment: $\text{card}(R_j)$

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 $R$ $\Rightarrow SF = 0.1$
- Size of an intermediate relation:
  $$\text{size}(R) = \text{card}(R) \times \text{length}(R)$$

Cardinality of Selections

- Cardinality
  $$\text{card}(\sigma_F(R)) = SF_S(F) \times \text{card}(R)$$
- $SF$ depends on selection condition with predicates $p(A_i)$ and constants $v$
  \[
  SF_S(A = v) = \frac{1}{\text{val}(A, R)} \\
  SF_S(A > v) = \frac{\text{max}(A) - v}{\text{max}(A) - \text{min}(A)} \\
  SF_S(A < v) = \frac{v - \text{min}(A)}{\text{max}(A) - \text{min}(A)}
  \]
Cardinality of Selections /2

\[
SF_S(p(A_i) \land p(A_j)) = SF_S(p(A_i)) \cdot SF_S(p(A_j))
\]
\[
SF_S(p(A_i) \lor p(A_j)) = SF_S(p(A_i)) + SF_S(p(A_j)) - (SF_S(p(A_i)) \cdot SF_S(p(A_j)))
\]
\[
SF_S(A \in \{v_1, \ldots, v_n\}) = SF_S(A = v) \cdot \text{card}(\{v_1, \ldots, v_n\})
\]

Cardinality of Projections

- Without duplicate elimination
  \[
  \text{card}(\pi_A(R)) = \text{card}(R)
  \]
- With duplicate elimination (for non-key attributes \(A\))
  \[
  \text{card}(\pi_A(R)) = \text{val}(A, R)
  \]
- With duplicate elimination (a key is subset of attributes in \(A\))
  \[
  \text{card}(\pi_A(R)) = \text{card}(R)
  \]

Cardinality of Joins

- Cartesian products
  \[
  \text{card}(R \times S) = \text{card}(R) \cdot \text{card}(S)
  \]
- Join
  - Upper bound: cardinality of Cartesian product
  - Better estimation for foreign key relationships \(S.B \rightarrow R.A\):
    \[
    \text{card}(R \bowtie_{A=B} S) = \text{card}(S)
    \]
  - Selectivity factor \(SF_J\) from database statistics
    \[
    \text{card}(R \bowtie S) = SF_J \cdot \text{card}(R) \cdot \text{card}(S)
    \]

Cardinality of Semi-joins

- Operation \(R \bowtie_A S\)
- Selectivity factor for attribute \(A\) from relation \(S\): \(SF_{SJ}(S.A)\)
  \[
  SF_{SJ}(R \bowtie_A S) = \frac{\text{val}(A, S)}{\text{card(dom}(A))}
  \]
- Cardinality
  \[
  \text{card}(R \bowtie_A S) = SF_{SJ}(S.A) \cdot \text{card}(R)
  \]
Cardinality of Set Operations

- Union $R \cup S$
  - Lower bound: $\max\{\text{card}(R), \text{card}(S)\}$
  - Upper bound: $\text{card}(R) + \text{card}(S)$

- Set difference $R - S$
  - Lower bound: 0
  - Upper bound: $\text{card}(R)$

Example

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

Example: Query Plans

- Variant 1:
  $\sigma_{\text{Budget} > 150.000}(\text{PROJECT}_1 \cup \text{PROJECT}_2 \cup \text{PROJECT}_3)$

- Variant 2:
  $\sigma_{\text{Budget} > 150.000}(\text{PROJECT}_1) \cup \sigma_{\text{Budget} > 150.000}(\text{PROJECT}_2) \cup \sigma_{\text{Budget} > 150.000}(\text{PROJECT}_3)$
Join Order in DDBS

- Huge influence on overall performance
- General rule: avoid Cartesian products where possible

Join order for 2 relations $R \Join S$

- $R \Join S$ if size($R$) < size($S$)
- $S \Join R$ if size($R$) > size($S$)

Join order for 3 relations $R \Join_A S \Join_B T$

- 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 $R \Join_A S$

- $R \Join_A S$
- $R \Join_A (S \Join_A R)$
- $(R \Join_A S) \Join_A (S \Join_A R)$

Comparison $R \Join_A S$ vs. $(R \Join_A S) \Join_A S)$ for size($R$) < size($S$)

Costs for $R \Join_A S$: transfer of $R$ to $N_2 \to T_{TR} \times$ 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 $R' := R \bowtie_A S$
  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

$$T_{TR} \times \text{size}(\pi_A(S)) + T_{TR} \times \text{size}(R \bowtie_A S)$$

• Accordingly: semi-join is better strategy if

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

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