Query Processing Components

- Query language that is used
  - SQL “intergalactic dataspeak”
- Query execution methodology
  - The steps that one goes through in executing high-level (declarative) user queries.
- Query optimization
  - How do we determine the “best” execution plan?
- We assume a homogenous D-DBMS

Example

```
SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND RESP = "Manager"
```

Cost of Alternatives

- Assume
  - \(\text{size}(\text{EMP}) = 400\)
  - \(\text{size}(\text{ASG}) = 1000\)
  - 20 managers / uniform distribution
- Tuple access cost = 1 unit;
- Tuple transfer cost = 10 units
- Index on EMP.ENO
- Index on ASG.RESP
Cost of Alternatives

**Strategy 1**
- Produce ASG': (10+10) * tuple access cost = 200
- Transfer ASG' to the sites of EMP: (10+10) * tuple transfer cost = 200
- Produce EMP': (10+10) * tuple access cost = 200
- Final union and projection: 40 * tuple access cost = 20

Total Cost = 460

**Strategy 2**
- Transfer EMP to site 5: 400 * tuple transfer cost = 4,000
- Transfer ASG to site 5: 1000 * tuple transfer cost = 10,000
- Produce ASG': 1000 * tuple access cost = 1,000
- Join EMP and ASG': 400 * 20 * tuple access cost = 8,000

Total Cost = 23,000

Query Optimization Objectives

- Minimize a cost function
  - I/O cost + CPU cost + communication cost
- These might have different weights in different distributed environments
- Wide area networks
  - Communication cost may dominate or vary much
  - Bandwidth
  - Speed
  - High protocol overhead
- Local area networks
  - Communication cost not that dominant
  - Total cost function should be considered
- Can also maximize throughput

Complexity of Relational Operations

- Assume
  - Relations of cardinality \( n \)
  - Sequential scan
- Operation
<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>( O(p) )</td>
</tr>
<tr>
<td>Project (without elimination)</td>
<td>( O(p + \log n) )</td>
</tr>
<tr>
<td>Group (with elimination)</td>
<td>( O(p + \log n) )</td>
</tr>
<tr>
<td>Join</td>
<td>( O(p + \log n) )</td>
</tr>
<tr>
<td>Semi-join</td>
<td></td>
</tr>
<tr>
<td>Decision</td>
<td></td>
</tr>
<tr>
<td>Set Operators</td>
<td></td>
</tr>
<tr>
<td>Cartesian Product</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Query Optimization Issues – Types Of Optimizers

- Exhaustive search
  - Cost-based
  - Optimal
  - Combinatorial complexity in the number of relations
- Heuristics
  - Not optimal
  - Regroup common sub-expressions
  - Perform selection, projection first
  - Replace a join by a series of semi-joins
  - Reorder operations to reduce intermediate relation size
  - Optimize individual operations

Query Optimization Issues – Optimization Granularity

- Single query at a time
  - Cannot use common intermediate results
- Multiple queries at a time
  - Efficient if many similar queries
  - Decision space is much larger
Query Optimization Issues – Optimization Timing

- **Static**
  - Compilation ➔ optimize prior to the execution
  - Difficult to estimate the size of the intermediate results ➔ error propagation
  - Can amortize over many executions ➔ R*
- **Dynamic**
  - Run time optimization
  - Exact information on the intermediate relation sizes ➔ error propagation
  - Have to reoptimize for multiple executions ➔ Distributed INGRES
- **Hybrid**
  - Compile using a static algorithm
  - If the error in estimate sizes > threshold, reoptimize at run time ➔ Mermaid

Query Optimization Issues – Statistics

- **Relation**
  - Cardinality
  - Size of a tuple
  - Fraction of tuples participating in a join with another relation
- **Attribute**
  - Cardinality of domain
  - Actual number of distinct values
  - Common assumptions
  - Independence between different attribute values
  - Uniform distribution of attribute values within their domain

Query Optimization Issues – Decision Sites

- **Centralized**
  - Single site determines the “best” schedule
  - Simple
  - Need knowledge about the entire distributed database
- **Distributed**
  - Cooperation among sites to determine the schedule
  - Need only local information
  - Cost of cooperation
- **Hybrid**
  - One site determines the global schedule
  - Each site optimizes the local subqueries

Distributed Query Processing Methodology

1. **Input:** Calculus query on global relations
2. **Distributed Query Processing**
   - Overview
   - Query decomposition and localization
   - Distributed query optimization

Outline

1. **Distributed Query Processing**
   - Overview
   - Query decomposition and localization
   - Distributed query optimization

Step 1 – Query Decomposition

- **Input:** Calculus query on global relations
- **Normalization**
  - manipulate query quantifiers and qualification
- **Analysis**
  - detect and reject “incorrect” queries
  - possible for only a subset of relational calculus
- **Simplification**
  - eliminate redundant predicates
- **Restructuring**
  - calculus query ➔ algebraic query
  - more than one translation is possible
  - use transformation rules
Normalization

- Lexical and syntactic analysis
  - check validity (similar to compilers)
  - check for attributes and relations
  - type checking on the qualification
- Put into normal form
  - Conjunctive normal form
    \[(p_1 \land p_2 \land \ldots \land p_n) \land \ldots \land (q_m \land p_{n+1} \land \ldots \land p_{2m})\]
  - Disjunctive normal form
    \[(p_1 \lor p_2 \lor \ldots \lor p_{n}) \lor \ldots \lor (q_m \lor p_{n+1} \lor \ldots \lor p_{2m})\]
- OR’s mapped into union
- AND’s mapped into join or selection

Analysis

- Refute incorrect queries
- Type incorrect
  - If any of its attribute or relation names are not defined in the global schema
  - If operations are applied to attributes of the wrong type
- Semantically incorrect
  - Components do not contribute in any way to the generation of the result
  - Only a subset of relational calculus queries can be tested for correctness
  - Those that do not contain disjunction and negation
- To detect
  - connection graph (query graph)
  - join graph

Analysis – Example

```sql
SELECT ENAME, RESP
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = PROJ.PNO
AND PNAME = "CAD/CAM"
AND DUR >= 36
AND TITLE = "Programmer"
```

Simplification

- Why simplify?
  - Remember the example
- How? Use transformation rules
  - Elimination of redundancy
    - idempotency rules
      \[p_1 \land \neg (p_1) \Rightarrow \text{false}\]
      \[p_1 \land (q_1 \lor p_2) \Rightarrow p_1\]
      \[p_1 \lor \text{false} \Rightarrow p_1\]
      \[
      \ldots
      \]
  - Application of transitivity
  - Use of integrity rules

Simplification – Example

```sql
SELECT TITLE
FROM EMP
WHERE EMP.ENAME = "J. Doe" OR NOT (EMP.TITLE = "Programmer") OR EMP.TITLE = "Elect. Eng." AND NOT (EMP.TITLE = "Elect. Eng."))
```
### Simplification – Example

```sql
SELECT TITLE
FROM EMP
WHERE EMP.ENAME = "J. Doe"
```

### Restructuring – Transformation Rules

- Commutativity of binary operations
  - \( R \times S = S \times R \)
  - \( R \sqcup S = S \sqcup R \)

- Associativity of binary operations
  - \( (R \times S) \times T = R \times (S \times T) \)
  - \( (R \sqcup S) \sqcup T = R \sqcup (S \sqcup T) \)

- Idempotence of unary operations
  - \( \Pi_i (\Pi_i (R)) = \Pi_i (R) \)
  - \( \sigma_{A_i} (\Pi_i (R)) = \sigma_{A_i} (\Pi_i (R)) \)
  - \( \sigma_{A_i} (\sigma_{B_j} (R)) = \sigma_{A_i} (\sigma_{B_j} (R)) \)
  - \( \pi_{A_i} (\pi_{B_j} (R)) = \pi_{A_i} (\pi_{B_j} (R)) \)
  - \( \sigma_{A_i} (\sigma_{B_j} (R)) = \sigma_{A_i} (\sigma_{B_j} (R)) \)
  - \( \pi_{A_i} (\pi_{B_j} (R)) = \pi_{A_i} (\pi_{B_j} (R)) \)
  - \( \sigma_{A_i} (\sigma_{B_j} (R)) = \sigma_{A_i} (\sigma_{B_j} (R)) \)
  - \( \pi_{A_i} (\pi_{B_j} (R)) = \pi_{A_i} (\pi_{B_j} (R)) \)

- Commuting selection with binary operations

- Commuting projection with binary operations

### Example

Recall the previous example:

Find the names of employees other than J. Doe who worked on the CAD/CAM project for either one or two years.

```sql
SELECT ENAME
FROM EMP
WHERE EMP.ENAME <> "J. Doe"
AND (PROJ.ENO = ASG.ENO OR ASG.ENO = EMP.ENO)
AND ASG.PNO = PROJ.PNO
AND ENAME = "J. Doe"
AND (DUR = 12 OR DUR = 24)
```
Restructuring

Example

Assume
- EMP is fragmented into EMP₁, EMP₂, EMP₃
  EMP₁ as follows:
  - EMP₁ = σENO ≤ "E₃"(EMP)
  - EMP₁ = σENO ≤ "E₃"(EMP)
  - EMP₁ = σENO ≤ "E₃"(EMP)
- ASG fragmented into ASG₁ and ASG₂
  ASG₁ = σENO ≤ "E₃"(ASG)
  ASG₂ = σENO ≤ "E₃"(ASG)
Replace EMP by (EMP₁ ∪ EMP₂ ∪ EMP₃) and ASG by (ASG₁ ∪ ASG₂) in any query

Step 2 – Data Localization

Input: Algebraic query on distributed relations
- Determine which fragments are involved
- Localization program
  - substitute for each global query its materialization program
  - optimize

Example

Assume
- EMP is fragmented into EMP₁, EMP₂, EMP₃
  EMP₁ as follows:
  - EMP₁ = σENO ≤ "E₃"(EMP)
  - EMP₁ = σENO ≤ "E₃"(EMP)
  - EMP₁ = σENO ≤ "E₃"(EMP)
- ASG fragmented into ASG₁ and ASG₂
  ASG₁ = σENO ≤ "E₃"(ASG)
  ASG₂ = σENO ≤ "E₃"(ASG)
Replace EMP by (EMP₁ ∪ EMP₂ ∪ EMP₃) and ASG by (ASG₁ ∪ ASG₂) in any query

Provides Parallellism

Eliminates Unnecessary Work
Reduction for PHF

- Reduction with selection
  - Relation \( R \) and \( F = \{R_p, R_q, \ldots, R_j\} \) where \( R \models \phi(x) \)
  - \( \sigma_p(R) = \emptyset \) if \( \forall x \in R: \neg \phi_p(x) \land \phi_j(x) \)
  - Example

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \\
& \quad \text{WHERE} \\
\text{EMP} & \quad \text{ENO} = \text{"E5"}
\end{align*}
\]

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union

\[
(R \cup R_2) \bowtie S \iff (R \bowtie S) \cup (R_2 \bowtie S)
\]

Assume EMP is fragmented as before and

- Given \( R \models \phi(x) \) and \( R_j = \sigma_p(R) \)

Example

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \\
& \quad \text{EMP, ASG} \\
& \quad \text{WHERE} \\
\text{EMP.ENO} & = \text{ASG.ENO} \\
& \quad \text{AND} \\
\text{EMP.TITLE} & = \text{"Mech. Eng."}
\end{align*}
\]

Reduction for DHF

- Rule:
  - Distribute joins over unions
  - Apply the join reduction for horizontal fragmentation
  - Example

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \\
& \quad \text{EMP, ASG} \\
& \quad \text{WHERE} \\
\text{EMP.ENO} & = \text{ASG.ENO} \land \text{EMP.TITLE} = \text{"Mech. Eng."}
\end{align*}
\]
Reduction for DHF

- Joins over unions
- Elimination of the empty intermediate relations (left sub-tree)

Reduction for Hybrid Fragmentation

- Combine the rules already specified:
  - Remove empty relations generated by contradicting selections on horizontal fragments;
  - Remove useless relations generated by projections on vertical fragments;
  - Distribute joins over unions in order to isolate and remove useless joins.

Example

Consider the following hybrid fragmentation:

- EMP = \( \sigma_{ENO \leq "E4"}(\Pi_{ENO,ENAME}(EMP)) \)
- EMP = \( \sigma_{ENO > "E4"}(\Pi_{ENO,ENAME}(EMP)) \)
- EMP = \( \sigma_{ENO,TITLE}(EMP) \)

and the query

- SELECT ENAME FROM EMP WHERE ENO = "E5"

Step 3 – Global Query Optimization

- Find the best (not necessarily optimal) global schedule
  - Minimize a cost function
  - Distributed join processing
  - Bushy vs. linear trees
  - Which relation to ship where?
  - Ship-whole vs ship-as-needed
  - Decide on the use of semijoins
  - Semijoins save on communication at the expense of more local processing.
  - Join methods
    - nested loop vs ordered joins (merge join or hash join)

Query Optimization Process

- Search space characterized by alternative execution
- Focus on join trees
- For \( N \) relations, there are \( O(N!) \) equivalent join trees that can be obtained by applying commutativity and associativity rules

Search Space
Information about data

- Relations & fragments
- Size and cardinality
- Per attribute:
  - cardinality,
  - max and min values,
  - distribution,
  - size,
  - effective distinct values

Estimation of the size of intermediate results

- Selectivity factor, SF ... similar to centralised systems
  - \( \text{card}(R) = \text{card}(R) \times \text{SF}(c) \)
- \( \text{card}(R \times S) = \text{card}(R) \times \text{card}(S) \times \text{SF of join condition} \)
- Statistiques issued from previous executions

Joins!

- The evaluation cost is « dominated » by data transfers
- Where do we evaluate joins?
  - Exemple : \( E \bowtie S1 \times G \bowtie S2 \times J \bowtie S3 \)
  - \( E \rightarrow S2; \text{join} \); \( \text{Res} \rightarrow S3 \)
  - \( G \rightarrow S1; \text{join} \); \( \text{Res} \rightarrow S1 \)
  - \( J \rightarrow S2; \text{join} \); \( \text{Res} \rightarrow S1 \)
  - \( E, J \rightarrow S2; \text{joins} \)

Semi-join / join

- Important technique
  - Given \( R(X), S(Y) \) condition \( c \) over attributes \( Z \)
  - \( R \cap^c S = \Pi_x (R \times^c S) \)
  - \( R \times^c S = (R \cap^c S) \times^c (S \cap^c R) \)
- Idea:
  - use semi-joins to evaluate joins
  - Transfer less data

Technique using Semi-joins

- Example: \( R \bowtie S1 \times S2 \)
  - \( E \bowtie (S) \rightarrow S1 \)
  - At \( S1: R \cap S: \text{Res} \rightarrow S2 \)
  - Finish at \( S2: \text{Res} \cap S \)
  - Transferred data:
    - \( \text{size}(\Pi_x (S)) + \text{size}(R \cap S) \)
- Transferred data without semi-joins:
  - \( \text{size}(R) \)

Discussion...

- \( \text{size}(\Pi_x (S)) + \text{size}(R \cap S) \) vs \( \text{size}(R) \)
- Depends on the selectivity factor
- More treatment:
  - One relation is used twice
  - No index on intermediate results
  - In some cases ... less transfers
Semi-joins & bit vectors

- $R@S1 \subset S@S2$, condition over attributes $Z$
- Use a hash function
  - $F : \text{dom}(Z) \rightarrow [1,N]$
- Evaluate $@S2: F(\Pi Z(S))$
  - This is a sub-set of $[1,N]$
- Represent as a bit vector and send it to $S1$
- Evaluate $@S1$: $R1$ and send back to $S2$
  - $R1 = \{ r \in R / F(r.z) \in F(\Pi Z(S)) \}$
  - $R \circ S$ is a sub-set of $R1$ which is a sub-set of $R$
- Evaluate $@ S2$ final result...

Semi-joins & bit vectors: Discussion

- Pros: less transfers
- Cons:
  - More I/O:2 access to $S$
  - Hash function: collisions?
- Variants:
  - Compression of the vector
  - Use several vectors
  - Send vectors in the «two-ways»
  - ...

Conclusion & perspectives

- Extension of centralized techniques
- Many algorithms!!!
- Complexity is increased by
  - Autonomy of the participants
  - Increased number of participants
  - Large scale
- What about energy consumption issues?