Distributed Query Processing
(Chapters 7, 8, 9)

DISTRIBUTED QUERY PROCESSING PROBLEM

Besides the choice of ordering relational algebra operations, we must also select the best sites to process data.

Example: SELECT *
FROM E, G
WHERE E.ENO=G.ENO AND
RESP='Manager'

G or ASG (ENO, PNO, RESP, DUR)
E or EMP(ENO, ENAME, TITLE)

Bad algebra: \[ \sigma_{\text{RESP}='\text{Manager'} \land E.\text{ENO}=G.\text{ENO}} (E \times G) \]

Good algebra: \[ E \bowtie_{\text{ENO}} (\sigma_{\text{RESP}='\text{Manager'}} (G)) \]
\[ EMP \bowtie_{ENO} (\sigma_{\text{RESP}=\text{Manager}^+}(ASG)) \]

**Fragmentation Schema**

- \( EMP_1 = \sigma_{ENOS^E3}(EMP) \)
- \( EMP_2 = \sigma_{ENO^E3}(EMP) \)
- \( ASG_1 = \sigma_{ENOS^E3}(EMP_1) \)
- \( ASG_2 = \sigma_{ENO^E3}(EMP_2) \)

**Allocation Schema**

- Site 1: ASG₁
- Site 2: ASG₂
- Site 3: EMP₁
- Site 4: EMP₂

---

**Query Plans**

- **Strategy A**
  - Site 1: ASG₁
  - Site 2: ASG₂
  - Site 3: EMP₁
  - Site 4: EMP₂

- **Strategy B**
  - Site 1: ASG₁
  - Site 2: ASG₂
  - Site 3: EMP₁
  - Site 4: EMP₂

---

**Regular Join**

\[ \text{result} = R \bowtie_A S \]

**Semijoin:**

\[ R \bowtie_A S \iff (R_{SJ_A} S) \bowtie_A S' \]

- \( R' = R_{SJ_A} S' \)
- \( S' = \Pi_A(S) \)

Typically \(|S'| + |R'| \ll |R|\)

More processing overhead to do a join using semi-join
**PRINCIPLES:**

- The most selective operations that reduce cardinalities should be performed first.
- Operations should be ordered by increasing complexity so that expensive operations can be avoided or delayed.

Assume that tuples of each relation must be sorted.

\( n \) denotes the relation cardinality.

**OBJECTIVES OF QUERY OPTIMIZATION**

1. **Good measurement parameters:**
   1. **Total cost:** The sum of all time incurred in processing the operations at various sites and in intersite communication.
   2. **Response Time:** The time elapsed for executing the query.

2. **Cost = f(CPU\_cost, IO\_cost, Communication\_Cost)**

- Minimize CPU\_cost: use less expensive strategies
- Minimize IO\_cost: use fast access methods and good buffer management strategies
- Minimize communication\_cost: choose the execution sites intelligently.

**Very slow communication networks:**

- Minimize communication costs generally at the expense of local processing.
Distributed Query Processing Steps

Query Decomposition:
- Normalization
- Semantically analyze the normalized query to eliminate incorrect queries.
- Simplify the correct query by removing redundant predicates.
- Restructure the algebraic query into a “better” algebraic specification.

This step is the same as standalone DBMS.

Distributed Query Processing Steps

Data Localization:
- Determine which fragments are involved and transform the global query into fragment queries.

Global Query Optimization:
- Find the “best” ordering of fragment queries and specifies the communication operations.

Local Query Optimization:
- Each site determines the access methods for the local fragment queries using the local schema.
Query Decomposition and Data Localization

Query Decomposition

- Normalization
- Semantic Analysis
- Simplification
- Rewriting

Data Localization: Reduction techniques for different types of fragmentation.

- Horizontal
- Vertical
- Derived
- Hybrid

Normalization

- The input query may be arbitrary complex.
- Normalization transforms the query into a normalized form to facilitate further processing.

Two possible normal forms:

Conjunctive Normal Form:

\[(p_{11} \lor p_{12} \lor \ldots \lor p_{1n}) \land (p_{m1} \lor p_{m2} \lor \ldots \lor p_{mn})\]

Disjunctive Normal Form:

\[(p_{11} \land p_{12} \land \ldots \land p_{1n}) \lor (p_{m1} \land p_{m2} \land \ldots \land p_{mn})\]

where \(p_{ij}\) is a simple predicate.
Simple Predicates

Given a relation \( R(A_1, A_2, \ldots, A_n) \) where \( A_i \) has domain \( D_i \), a simple predicate \( p_j \) defined on \( R \) has the form
\[
p_j: A_i \theta \text{ Value}
\]
where \( \theta \in \{=,<,\neq,\leq,\geq\} \) and \( \text{Value} \in D_i \)

Example:

<table>
<thead>
<tr>
<th>JNO</th>
<th>JNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Instrumental</td>
<td>150,000</td>
<td>Montreal</td>
</tr>
<tr>
<td>J2</td>
<td>Database Dev</td>
<td>135,000</td>
<td>New York</td>
</tr>
<tr>
<td>J3</td>
<td>CAD/CAM</td>
<td>250,000</td>
<td>New York</td>
</tr>
<tr>
<td>J4</td>
<td>Maintenance</td>
<td>350,000</td>
<td>Orlando</td>
</tr>
</tbody>
</table>

Simple predicates:
\[
p_1: \text{JNAME}= \text{'Maintenance'}
p_2: \text{BUDGET}= \text{'200,000'}
\]

Examples:

Query:
\[
\text{SELECT ENAME FROM E, G WHERE E.ENO=G.ENO AND G.JNO=’J1’ AND (DUR=12 OR DUR=24)}
\]

Qualification in conjunctive normal form:
\[
(E.ENO = G.ENO) \land (G.JNO =’J1’) \land (DUR = 12 \lor DUR = 24)
\]

Qualification in disjunctive normal form:
\[
(E.ENO = G.ENO \land G.JNO =’J1’\land DUR = 12) \lor (E.ENO = G.ENO \land G.JNO =’J1’\land DUR = 24)
\]

Disjunctive normal form can lead to replicated join and select predicates.
Semantic Analysis

Semantic analysis enables rejection of incorrect queries.

• Type checking detects type incorrect problems.

```
SELECT E#
FROM E
WHERE ENAME>200;
```

Undefined attribute!

Incompatible operation!

• Query graph: Determine the semantic correctness of a conjunctive multivariable (multi-table) query without negation.

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

Systems should reject queries with the unconnected join graphs.

Figure 8.2: Disconnected Query Graph
Elimination of Redundancy

Redundant predicates are likely to arise when a query is the result of system transformations applied to the user query.

Such redundancy and thus redundant work may be eliminated by simplifying the qualification using well-known idempotency rules.

Example:

\[
(\neg p_1 \land (p_1 \lor p_2) \land \neg p_2) \lor p_3 \\
= (\neg p_1 \land p_1 \land \neg p_2) \lor (\neg p_1 \land p_2 \land \neg p_2) \lor p_3 \\
= (F \land \neg p_2) \lor (\neg p_1 \land F) \lor p_3 \\
= F \lor F \lor p_3 = p_3
\]

Rewriting

Two Steps:

1. Transforming the query into an algebraic relational query tree.
2. Restructuring the algebraic tree to improve performance