Outline

- Introduction & architectural issues
- Data distribution
- Distributed query processing
  - Query Processing Methodology
  - Localization
- Distributed query optimization
- Distributed transactions & concurrency control
- Distributed reliability
- Data replication
- Parallel database systems
- Database integration & querying
- Advanced topics

Query Processing

high level user query

query processor

low level data manipulation commands
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?

Selecting Alternatives

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

Strategy 1

\[ \Pi_{ENAME}(\sigma_{\text{RESP} = \text{“Manager”}} \land \text{EMP.ENO = ASG.ENO})(\text{EMP} \cdot \text{ASG})] \]

Strategy 2

\[ \Pi_{ENAME}(\text{EMP} \bowtie_{\text{ENO}} (\sigma_{\text{RESP} = \text{“Manager”}})(\text{ASG})) \]

Strategy 2 avoids Cartesian product, so is “better”
What is the Problem?

<table>
<thead>
<tr>
<th>Site 1</th>
<th>Site 2</th>
<th>Site 3</th>
<th>Site 4</th>
<th>Site 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP(=\sigma_{\text{ENO}\leq &quot;E3&quot;}(\text{EMP}))</td>
<td>EMP(=\sigma_{\text{ENO}&gt;'E3'}(\text{EMP}))</td>
<td>EMP(=\sigma_{\text{ENO}&gt;'E3'}(\text{EMP}))</td>
<td>EMP(=\sigma_{\text{ENO}&gt;'E3'}(\text{EMP}))</td>
<td>Result</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Site 1</th>
<th>Site 2</th>
<th>Site 3</th>
<th>Site 4</th>
<th>Site 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASG(=\sigma_{\text{RESP}='Manager'}(\text{ASG}))</td>
<td>ASG(=\sigma_{\text{RESP}='Manager'}(\text{ASG}))</td>
<td>ASG(=\sigma_{\text{RESP}='Manager'}(\text{ASG}))</td>
<td>Result</td>
<td></td>
</tr>
</tbody>
</table>

Cost of Alternatives

- **Assume:**
  - \(\text{size}(\text{EMP}) = 400, \text{size}(\text{ASG}) = 1000\)
  - tuple access cost = 1 unit; tuple transfer cost = 10 units

- **Strategy 1**
  1. produce \(\text{ASG}'\): \((10+10)\times\text{tuple access cost}\) \(= 20\)
  2. transfer \(\text{ASG}'\) to the sites of \(\text{EMP}\): \((10+10)\times\text{tuple transfer cost}\) \(= 200\)
  3. produce \(\text{EMP}'\): \((10+10)\times\text{tuple access cost}\times 2\) \(= 40\)
  4. transfer \(\text{EMP}'\) to result site: \((10+10)\times\text{tuple transfer cost}\) \(= 200\)
  - Total cost \(= 460\)

- **Strategy 2**
  1. transfer EMP to site 5: \(400\times\text{tuple transfer cost}\) \(= 4,000\)
  2. transfer ASG to site 5: \(1000\times\text{tuple transfer cost}\) \(= 10,000\)
  3. produce \(\text{ASG}'\): \(1000\times\text{tuple access cost}\) \(= 1,000\)
  4. join EMP and ASG\(':400\times20\times\text{tuple access cost}\) \(= 8,000\)
  - 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 will dominate
  - low bandwidth
  - low speed
  - high protocol overhead
- most algorithms ignore all other cost components

Local area networks

- communication cost not that dominant
- total cost function should be considered

Can also maximize throughput

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 semijoins
  - 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
  - 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
Query Optimization Issues – Network Topology

- **Wide area networks** (WAN) – point-to-point
  - Characteristics
    - Low bandwidth
    - Low speed
    - High protocol overhead
  - Communication cost will dominate; ignore all other cost factors
  - Global schedule to minimize communication cost
  - Local schedules according to centralized query optimization

- **Local area networks** (LAN)
  - Communication cost not that dominant
  - Total cost function should be considered
  - Broadcasting can be exploited (joins)
  - Special algorithms exist for star networks

Distributed Query Processing Methodology

CONTROL SITE

LOCAL SITES

Calculus Query on Distributed Relations

Algebraic Query on Distributed Relations

Data Localization

Global Optimization

Optimized Fragment Query with Communication Operations

Optimized Local Queries

GLOBAL SCHEMA

FRAGMENT SCHEMA

STATS ON FRAGMENTS

LOCAL SCHEMAS
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 $\Rightarrow$ algebraic query
  - More than one translation is possible
  - Use transformation rules

Restructuring

- Convert relational calculus to relational algebra
- Make use of query trees
- Example
  
  Find the names of employees other than J. Doe who worked on the CAD/CAM project for either 1 or 2 years.

  ```sql
  SELECT ENAME
  FROM EMP, ASG, PROJ
  WHERE EMP.ENO = ASG.ENO
  AND ASG.PNO = PROJ.PNO
  AND ENAME ≠ "J. Doe"
  AND PNAME = "CAD/CAM"
  AND DUR = 12 or DUR = 24)
  ```
Restructuring – Transformation Rules

- Commutativity of binary operations
  - \( R \times S \Leftrightarrow S \times R \)
  - \( R \bowtie S \Leftrightarrow S \bowtie R \)
  - \( R \cup S \Leftrightarrow S \cup R \)

- Associativity of binary operations
  - \((R \times S) \times T \Leftrightarrow R \times (S \times T)\)
  - \((R \bowtie S) \bowtie T \Leftrightarrow R \bowtie (S \bowtie T)\)

- Idempotence of unary operations
  - \( \Pi_A (\Pi_A(R)) \Leftrightarrow \Pi_A(R) \)
  - \( \sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \Leftrightarrow \sigma_{p_1(A_1)}(\sigma_{p_2(A_2)}(R)) \)
  where \( R[A] \) and \( A' \subseteq A \), \( A'' \subseteq A \) and \( A' \subseteq A'' \)

- Commuting selection with projection

Previous Example – Equivalent Query

\[ \Pi_{ENAME} \]
\[ \sigma_{DUR=12 \land DUR=24} \]
\[ \sigma_{PNAME="CAD/CAM"} \]
\[ \sigma_{ENAME\neq "J. Doe"} \]
\[ \bowtie PNO \]
\[ \bowtie ENO \]
\[ PROJ \]
\[ ASG \]
\[ EMP \]

\[ \Pi_{ENAME} \]
\[ \sigma_{PNAME= "CAD/CAM" \land (DUR=12 \land DUR=24) \land ENAME \neq "J. Doe"} \]
\[ \bowtie PNO,ENO \]
\[ PROJ \]
\[ ASG \]
\[ EMP \]
Restructuring

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 EMP$_1$, EMP$_2$, EMP$_3$ as follows:
  - EMP$_1$ = $\sigma_{\text{ENO} \leq \text{"E3"}}$(EMP)
  - EMP$_2$ = $\sigma_{\text{"E3"} < \text{ENO} \leq \text{"E6"}}$(EMP)
  - EMP$_3$ = $\sigma_{\text{ENO} \geq \text{"E6"}}$(EMP)
- ASG fragmented into ASG$_1$ and ASG$_2$ as follows:
  - ASG$_1$ = $\sigma_{\text{ENO} \leq \text{"E3"}}$(ASG)
  - ASG$_2$ = $\sigma_{\text{ENO} > \text{"E3"}}$(ASG)

Replace EMP by (EMP$_1$ $\cup$ EMP$_2$ $\cup$ EMP$_3$) and ASG by (ASG$_1$ $\cup$ ASG$_2$) in any query.

Provides Parallelism
Eliminates Unnecessary Work

\[ \text{EMP}_1 \cup \text{ASG}_1 \quad \bowtie \quad \text{ENO} \quad \bowtie \quad \text{ENO} \quad \bowtie \quad \text{ENO} \]

Reduction for PHF

- Reduction with selection
  - Relation \( R \) and \( F_R=\{R_1, R_2, \ldots, R_w\} \) where \( R_j=\sigma_{p_j}(R) \)
  - Example
    
    ```
    \begin{verbatim}
    SELECT *
    FROM EMP
    WHERE ENO=\"E5\"
    \end{verbatim}
    ```

\[ \text{EMP}_1 \cup \text{EMP}_2 \cup \text{EMP}_3 \quad \bowtie \quad \text{ENO}_\text{ENO}=\text{ENO} \]

\[ \text{EMP}_2 \]

Page 12
**Reduction for PHF**

- Reduction with join
  - Possible if fragmentation is done on join attribute
  - Distribute join over union
    \[(R_1 \cup R_2) \bowtie S \Leftrightarrow (R_1 \bowtie S) \cup (R_2 \bowtie S)\]
  - Given \(R_i = \sigma_{p_i}(R)\) and \(R_j = \sigma_{p_j}(R)\)
    \[R_i \bowtie R_j = \emptyset \text{ if } \forall x \in R_i, \forall y \in R_j: \neg (p_i(x) \land p_j(y))\]

---

**Reduction for PHF**

- Assume EMP is fragmented as before and
  - \(ASG_1: \sigma_{ENO \leq "E3"}(ASG)\)
  - \(ASG_2: \sigma_{ENO > "E3"}(ASG)\)
- Consider the query
  
  ```
  SELECT *
  FROM EMP, ASG
  WHERE EMP.ENO = ASG.ENO
  ```

  - Distribute join over unions
  - Apply the reduction rule
Reduction for VF

- Find useless (not empty) intermediate relations
  - Relation $R$ defined over attributes $A = \{A_1, ..., A_n\}$ vertically fragmented as $R_i = \Pi_{A_i}(R)$ where $A_i \subseteq A$:
  - $\Pi_{D,K}(R_i)$ is useless if the set of projection attributes $D$ is not in $A_i'$
  - Example: $\text{EMP}_1 = \Pi_{\text{ENO, ENAME}}(\text{EMP})$, $\text{EMP}_2 = \Pi_{\text{ENO, TITLE}}(\text{EMP})$

$$\text{SELECT ENAME FROM EMP}$$

\[ \text{EMP}_1 \bowtie \text{ENO} \text{EMP}_2 \]

\[ \text{EMP}_1 \]

Reduction for DHF

- Rule:
  - Distribute joins over unions
  - Apply the join reduction for horizontal fragmentation
- Example
  - $\text{ASG}_1$: $\text{ASG} \bowtie \text{ENO} \text{EMP}_1$
  - $\text{ASG}_2$: $\text{ASG} \bowtie \text{ENO} \text{EMP}_2$
  - $\text{EMP}_1$: $\sigma_{\text{TITLE}=\text{"Programmer"}}(\text{EMP})$
  - $\text{EMP}_2$: $\sigma_{\text{TITLE}=\text{"Programmer"}}(\text{EMP})$
- Query
  $$\text{SELECT } * \text{ FROM EMP, ASG WHERE ASG.ENO = EMP.ENO AND EMP.TITLE = "Mech. Eng."}$$
Reduction for DHF

Generic query

Selections first

Joins over unions

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