Distributed Databases

by Farnoush Banaei-Kashani

Excerpt from “Principles of Distributed Database Systems”
by M. Tamer Özsu and Patrick Valduriez

Topics

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Semantic Data Control
- Distributed Query Processing
- Distributed Transaction Management
- Parallel Database Systems
- Distributed Object DBMS
- Database Interoperability
- Current Issues
Outline

- Problem Definition
- Issues to Consider
- Methodology
  - Step 1: Query Decomposition
  - Step 2: Data Localization
  - Step 3: Global Optimization
  - Step 4: Local Optimization

Query Processing

```
high level user query
query processor

low level data manipulation commands
```

April 19, 2006
CSCI585 - Distributed Databases

3

April 19, 2006
CSCI585 - Distributed Databases

4
Problem?

```sql
SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND DUR > 37
```

**Strategy 1**

\[ \Pi_{\text{ENAME}}(\sigma_{\text{DUR}>37} \land \text{EMP.ENO} = \text{ASG.ENO}) (\text{EMP} \times \text{ASG}) \]

**Strategy 2**

\[ \Pi_{\text{ENAME}}(\text{EMP} \bowtie_{\text{ENO}} \sigma_{\text{DUR}>37} (\text{ASG})) \]

Strategy 2 avoids Cartesian product, so is “better”

Problem in DDBS?

<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_1 = \sigma_{\text{ENO}=\text{EN}_1}(\text{ASG})</td>
<td>ASG_2 = \sigma_{\text{ENO}=\text{EN}_2}(\text{ASG})</td>
<td>EMP_3 = \sigma_{\text{ENO}=\text{EN}_3}(\text{EMP})</td>
<td>EMP_4 = \sigma_{\text{ENO}=\text{EN}_4}(\text{EMP})</td>
<td>Result</td>
</tr>
</tbody>
</table>

**Strategy 1**

Site 5
result = EMP_5 \cup \setminus EMP_5

**Strategy 2**

Site 5
result = EMP_5 \cup \setminus EMP_5 \setminus \sigma_{\text{DUR}>37}(\text{ASG}_1 \cup \text{ASG}_2)
Problem in DDBS?

- Assume:
  - \textit{size}(EMP) = 400, \textit{size}(ASG) = 1000, \textit{selectivity} = 20\% 
  - tuple access cost = 1 unit; tuple transfer cost = 10 units

- **Strategy 1**
  1. produce ASG': (10+10) \times \text{tuple access cost} = 20
  2. transfer ASG' to the sites of EMP: (10+10) \times \text{tuple transfer cost} = 200
  3. produce EMP': (10+10) \times \text{tuple access cost} = 40
  4. transfer 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 ASG': 1000 \times \text{tuple access cost} = 1,000
  4. join EMP and ASG': 400 \times 20 \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
Complexity of Relational Operations

- Assume
  - relations of cardinality \( n \)
  - sequential scan

<table>
<thead>
<tr>
<th>Operation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select (without duplicate elimination)</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Project Group</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Join</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Semi-join</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Division</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Set Operators</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Cartesian Product</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

Outline

- Problem Definition
- Issues to Consider
- Methodology
  - Step 1: Query Decomposition
  - Step 2: Data Localization
  - Step 3: Global Optimization
  - Step 4: Local Optimization
Query Processing 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 Processing 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 Processing 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 Processing 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 Processing 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 Processing 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
Outline

- Problem Definition
- Issues to Consider
- Methodology
  - Step 1: Query Decomposition
  - Step 2: Data Localization
  - Step 3: Global Optimization
  - Step 4: Local Optimization

Distributed Query Processing Methodology

1. Calculus Query on Distributed Relations
2. Algebraic Query on Distributed Relations
3. Fragment Query
4. Global Optimization
5. Optimized Fragment Query with Communication Operations
6. Local Optimization
7. Optimized Local Queries
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

---

Our Running Example

<table>
<thead>
<tr>
<th>EMP</th>
<th>ENO</th>
<th>ENAME</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>J. Doe</td>
<td>Elect. Eng.</td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>M. Smith</td>
<td>Syst. Anal.</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>J. Miller</td>
<td>Programmer</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>B. Casey</td>
<td>Syst. Anal.</td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>J. Jones</td>
<td>Syst. Anal.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASG</th>
<th>ENO</th>
<th>PNO</th>
<th>RESP</th>
<th>DUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>P1</td>
<td>Manager</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>P1</td>
<td>Analyst</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>P2</td>
<td>Analyst</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>P3</td>
<td>Consultant</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>P1</td>
<td>Engineer</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>P2</td>
<td>Programmer</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>P3</td>
<td>Manager</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>P3</td>
<td>Manager</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJ</th>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Database Develop</td>
<td>135000</td>
<td>New York</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAY</th>
<th>TITTLE</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Elect. Eng.</td>
<td>40000</td>
</tr>
<tr>
<td>E2</td>
<td>Syst. Anal.</td>
<td>34000</td>
</tr>
<tr>
<td>E3</td>
<td>Mech. Eng.</td>
<td>27000</td>
</tr>
<tr>
<td>E4</td>
<td>Programmer</td>
<td>24000</td>
</tr>
</tbody>
</table>
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.

```
SELECT ENAME
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.FNO = PROJ.FNO
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
  - $\Pi_{A}(\Pi_{A}(R)) \Leftrightarrow \Pi_{A}(R)$
  - $\sigma_{p[A]}(\sigma_{p[A2]}(R)) = \sigma_{p[A \cap A2]}(R)$
    where $R[A]$ and $A' \subseteq A$, $A'' \subseteq A$ and $A' \subseteq A''$

- Commuting selection with projection
Restructuring - Transformation Rules

- Commuting selection with binary operations
  - $\sigma_{p(A)}(R \times S) \Leftrightarrow (\sigma_{p(A)}(R)) \times S$
  - $\sigma_{p(A)}(R \bowtie_{A_j B_j} S) \Leftrightarrow (\sigma_{p(A)}(R)) \bowtie_{A_j B_j} S$
  - $\sigma_{p(A)}(R \cup T) \Leftrightarrow \sigma_{p(A)}(R) \cup \sigma_{p(A)}(T)$

  where $A_j$ belongs to $R$ and $T$

- Commuting projection with binary operations
  - $\Pi_c(R \times S) \Leftrightarrow \Pi_A(R) \times \Pi_B(S)$
  - $\Pi_c(R \bowtie_{A_j B_j} S) \Leftrightarrow \Pi_{A_j}(R) \bowtie_{A_j B_j} \Pi_B(S)$
  - $\Pi_c(R \cup S) \Leftrightarrow \Pi_c(R) \cup \Pi_c(S)$

  where $R[A]$ and $S[B]$; $C = A' \cup B'$ where $A' \subseteq A$, $B' \subseteq B$

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.

```
SELECT ENAME
FROM PROJ, ASG, EMP
WHERE ASG.ENO=EMP.ENO
AND ASG.PNO=PROJ.PNO
AND ENAME<>"J. Doe"
AND PROJ.PNAME="CAD/CAM"
AND (DUR=12 OR DUR=24)
```

Project

Select

Join
Equivalent Query

\[ \pi_{ENAME} \]

\[ \sigma_{\text{PNAME} = "CAD/CAM" \land \text{DUR} > 12 \land \text{DUR} < 24} \land \text{ENAME} = "J. Doe" \]

\[ \bowtie_{\text{PNO} \land \text{ENO}} \]

ASG PROJ EMP

Restructuring

\[ \pi_{ENAME} \]

\[ \bowtie_{\text{PNO}} \]

\[ \pi_{\text{PNO}, \text{ENAME}} \]

\[ \sigma_{\text{PNAME} = "CAD/CAM"} \]

\[ \sigma_{\text{DUR} > 12 \land \text{DUR} < 24} \]

\[ \sigma_{\text{ENAME} = "J. Doe"} \]

PROJ ASG EMP
Distributed Query Processing Methodology

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 as follows:
  - EMP₁ = σ_{ENOC<12}(EMP)
  - EMP₂ = σ_{ENOC<24}(EMP)
  - EMP₃ = σ_{ENOC<24}(EMP)

- ASG fragmented into ASG₁ and ASG₂ as follows:
  - ASG₁ = σ_{ENOC<30}(ASG)
  - ASG₂ = σ_{ENOC<30}(ASG)

Replace EMP by (EMP₁ ∪ EMP₂ ∪ EMP₃) and ASG by (ASG₁ ∪ ASG₂) in any query

Provides Parallelism

- EMP₁, ASG₁, EMP₂, ASG₂, EMP₃, ASG₁, EMP₃, ASG₂
Eliminates Unnecessary Work

Reduction for PHF

- Reduction with selection
  - Relation $R$ and $F_{R} = \{R_1, R_2, ..., R_n\}$ where $R_j = \sigma_{p_j}(R)$
  - $\sigma_{p_j}(R) = \phi$ if $\forall x \in R$:
    $\neg (p(x) \land p_j(x))$
  - Example
    
    ```sql
    SELECT *
    FROM EMP
    WHERE ENO = "E5"
    ```

- Diagram showing the relationship between EMP1, EMP2, EMP3, ASG1, ASG2, and EMP2.
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 with join - Example
  - Assume EMP is fragmented as before and
  \[\sigma_{\text{ENO} \leq \text{EP}}(\text{ASG})\]
  \[\sigma_{\text{ENO} > \text{EP}}(\text{ASG})\]
  - Consider the query
    \[
    \text{SELECT}^{*} \\
    \text{FROM} \quad \text{EMP, ASG} \\
    \text{WHERE} \quad \text{EMP. ENO} = \text{ASG. ENO}
    \]

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

- Reduction with join - Example
  - 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 $R_i = \Pi_{A'}(R)$ where $A' \subseteq A$:

$\Pi_{A',\Delta}(R_i)$ is useless if the set of projection attributes $D$ is not in $A'$

Example: $EMP_1 = \Pi_{ENO,ENAME}(EMP)$; $EMP_2 = \Pi_{ENO,TITLE}(EMP)$

```
SELECT   ENAME
FROM     EMP
```

$\Pi_{ENAME}$

$\Rightarrow$

$\Pi_{ENAME}$
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

```sql
SELECT * 
FROM EMP, ASG 
WHERE ASG.ENO = EMP.ENO 
AND EMP.TITLE = "Mech. Eng."
```
Reduction for DHF

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

Reduction for HF

- 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.
Reduction for HF

Example
Consider the following hybrid fragmentation:

\[ EMP_1 = \sigma_{ENO='E5'}(\Pi_{ENAME}(EMP)) \]
\[ EMP_2 = \sigma_{ENO='E5'}(\Pi_{ENAME}(EMP)) \]
\[ EMP_3 = \Pi_{ENAME}(EMP) \]

and the query

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

Distributed Query Processing Methodology

Calculus Query on Distributed Relations

GLOBAL SCHEMA

Algebraic Query on Distributed Relations

FRAGMENT SCHEMA

Fragment Query

STATS ON FRAGMENTS

Global Optimization

LOCAL SCHEMAS

Optimized Fragment Query with Communication Operations

LOCAL OPTIMIZATION

Optimized Local Queries

LOCAL SITES

CONTROL SITE
Step 3 – Global Query Optimization

Input: Fragment query
- 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
    - Semijoin saves on communication at the expense of more local processing.
  - Join methods
    - Nested loop vs ordered joins (merge join or hash join)

Cost-Based Optimization

- Solution space
  - The set of equivalent algebra expressions (query trees).
- Cost function (in terms of time)
  - I/O cost + CPU cost + communication cost
  - These might have different weights in different distributed environments (LAN vs WAN).
  - Can also maximize throughput
- Search algorithm
  - How do we move inside the solution space?
  - Exhaustive search, heuristic algorithms (iterative improvement, simulated annealing, genetic,...)
Query Optimization Process

Input Query

Search Space Generation

Transformation Rules

Equivalent QEP

Search Strategy

Cost Model

Best QEP

Search Space

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

SELECT ENAME, RESP
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = PROJ.PNO
Search Space

- Restrict by means of heuristics
  - Perform unary operations before binary operations
  - ...
- Restrict the shape of the join tree
  - Consider only linear trees, ignore bushy ones

Linear Join Tree

Bushy Join Tree

Search Strategy

- How to “move” in the search space.
- Deterministic
  - Start from base relations and build plans by adding one relation at each step
  - Dynamic programming: breadth-first
  - Greedy: depth-first
- Randomized
  - Search for optimalities around a particular starting point
  - Trade optimization time for execution time
  - Better when > 5-6 relations
  - Simulated annealing
  - Iterative improvement
Search Strategies

- Deterministic

- Randomized

Cost Functions

- Total Time (or Total Cost)
  - Reduce each cost (in terms of time) component individually
  - Do as little of each cost component as possible
  - Optimizes the utilization of the resources
    - Increases system throughput

- Response Time
  - Do as many things as possible in parallel
  - May increase total time because of increased total activity
Total Cost (or Total Time)

Summation of all cost factors

Total cost = CPU cost + I/O cost + communication cost

CPU cost = unit instruction cost * no. of instructions

I/O cost = unit disk I/O cost * no. of disk I/Os

communication cost = message initiation + transmission

---

Total Cost: WAN vs. LAN

- Wide area network
  - message initiation and transmission costs high
  - local processing cost is low (fast mainframes or minicomputers)
  - ratio of communication to I/O costs = 20:1

- Local area networks
  - communication and local processing costs are more or less equal
  - ratio = 1:1.6
Response Time

Elapsed time between the initiation and the completion of a query

\[
\text{Response time} = \text{CPU time} + \text{I/O time} + \text{communication time}
\]

\[
\text{CPU time} = \text{unit instruction time} \times \text{no. of \textit{sequential} instructions}
\]

\[
\text{I/O time} = \text{unit I/O time} \times \text{no. of \textit{sequential} I/Os}
\]

\[
\text{communication time} = \text{unit msg initiation time} \times \\
\text{no. of \textit{sequential} msg} + \text{unit transmission time} \times \\
\text{no. of \textit{sequential} bytes}
\]

Example

Assume that only the communication cost is considered

\[
\text{Total time} = 2 \times \text{message initialization time} + \text{unit transmission time} \times (x+y)
\]

\[
\text{Response time} = \max \times \text{(time to send } x \text{ from 1 to 3, time to send } y \text{ from 2 to 3)}
\]

\[
\text{time to send } x \text{ from 1 to 3} = \text{message initialization time} + \\
\text{unit transmission time} \times x
\]

\[
\text{time to send } y \text{ from 2 to 3} = \text{message initialization time} + \\
\text{unit transmission time} \times y
\]
Optimization Statistics

- Primary cost factor: size of intermediate relations
- Make them precise more costly to maintain
  - For each relation \( R[A_1, A_2, \ldots, A_n] \) fragmented as \( R_1, \ldots, R_s \)
    - length of each attribute: \( \text{length}(A_i) \)
    - the number of distinct values for each attribute in each fragment:
      \( \text{card}(R_i) \)
    - maximum and minimum values in the domain of each attribute:
      \( \text{min}(A_i), \text{max}(A_i) \)
    - the cardinalities of each domain: \( \text{card}(\text{dom}[A_i]) \)
    - the cardinalities of each fragment: \( \text{card}(R_i) \)
  - Selectivity factor of each operation for relations
    - For joins

\[
SF_{\Join}(R,S) = \frac{\text{card}(R \bowtie S)}{\text{card}(R) \cdot \text{card}(S)}
\]

Intermediate Relation Sizes

**Selection**

\[
\text{size}(R) = \text{card}(R) \cdot \text{length}(R)
\]

\[
\text{card}(\sigma_p(R)) = SF_o(p) \cdot \text{card}(R)
\]

where

\[
SF_o(A = \text{value}) = \frac{1}{\text{card}((A = \text{value})(R))}
\]

\[
SF_o(A > \text{value}) = \frac{\text{max}(A) - \text{value}}{\text{max}(A) - \text{min}(A)}
\]

\[
SF_o(A < \text{value}) = \frac{\text{value} - \text{min}(A)}{\text{max}(A) - \text{min}(A)}
\]

\[
SF_o(p(A) \land p(A)) = SF_o(p(A)) \cdot SF_o(p(A))
\]

\[
SF_o(p(A) \lor p(A)) = SF_o(p(A)) + SF_o(p(A)) - (SF_o(p(A)) \cdot SF_o(p(A))
\]

\[
SF_o(A \in \{ \text{values} \}) = SF_o(A = \text{value}) \cdot \text{card}(\{\text{values}\})
\]
Intermediate Relation Sizes

Projection
\[ \text{card}(\Pi_a(R)) = \text{card}(R) \]

Cartesian Product
\[ \text{card}(R \times S) = \text{card}(R) \times \text{card}(S) \]

Union
- upper bound: \( \text{card}(R \cup S) = \text{card}(R) + \text{card}(S) \)
- lower bound: \( \text{card}(R \cup S) = \max(\text{card}(R), \text{card}(S)) \)

Set Difference
- upper bound: \( \text{card}(R-S) = \text{card}(R) \)
- lower bound: 0

Intermediate Relation Sizes

Join
- Special case: A is a key of R and B is a foreign key of S;
\[ \text{card}(R \bowtie_{A=B} S) = \text{card}(S) \]
- More general:
\[ \text{card}(R \bowtie S) = SF \bowtie_{A} \text{card}(R) \times \text{card}(S) \]

Semijoin
\[ \text{card}(R \bowtie A S) = SF \bowtie_{A} (S, A) \times \text{card}(R) \]
where
\[ SF \bowtie_{A} (R \bowtie A S) = SF \bowtie_{A} (S, A) = \frac{\text{card}(\Pi_A(S))}{\text{card(dom}[A])} \]
Join Ordering

- Consider two relations only

- Multiple relations more difficult because too many alternatives.
  - Compute the cost of all alternatives and select the best one.
  - Necessary to compute the size of intermediate relations which is difficult.
  - Use heuristics

Join Ordering – Example

Consider PROJ ▷◁ VNO ASG ▷◁ VNO EMP

Site 2

ASG

ENO

PNO

EMP

Site 1

PROJ

Site 3
Join Ordering – Example

Execution alternatives:

1. EMP → Site 2
   Site 2 computes EMP\rightarrow Site 2 computes EMP\rightarrow ASG

2. ASG → Site 1
   Site 1 computes EMP\rightarrow Site 1 computes EMP\rightarrow ASG

EMP' → Site 3
EMP' → Site 3

Site 3 computes EMP\rightarrow PROJ
Site 3 computes EMP\rightarrow PROJ

3. ASG → Site 3
   Site 3 computes ASG\rightarrow Site 3 computes ASG\rightarrow PROJ

ASG' → Site 1
ASG' → Site 1

Site 1 computes ASG\rightarrow EMP
Site 1 computes ASG\rightarrow EMP

4. PROJ → Site 2
   Site 2 computes PROJ\rightarrow Site 2 computes PROJ\rightarrow ASG

PROJ' → Site 1
PROJ' → Site 1

5. EMP → Site 2
   Site 2 computes EMP\rightarrow PROJ\rightarrow EMP

PROJ → Site 2
PROJ \rightarrow Site 2

Semijoin Algorithms

- Consider the join of two relations:
  - R[A] (located at site 1)
  - S[A] (located at site 2)

- Alternatives:
  1. Do the join \text{R}\bowtie_{A} S
  2. Perform one of the semijoin equivalents
     \[ R \bowtie_{A} S \Leftrightarrow (R \bowtie_{A} S) \bowtie_{A} S \]
     \[ \Leftrightarrow R \bowtie_{A} (S \bowtie_{A} R) \]
     \[ \Leftrightarrow (R \bowtie_{A} S) \bowtie_{A} (S \bowtie_{A} R) \]
Semijoin Algorithms

- Perform the join
  - send $R$ to Site 2
  - Site 2 computes $R \bowtie_A S$
- Consider semijoin $(R \bowtie_A S) \bowtie_A S$
  - $S' \leftarrow \Pi_A(S)$
  - $S' \rightarrow$ Site 1
  - Site 1 computes $R' = R \bowtie_A S'$
  - $R' \rightarrow$ Site 2
  - Site 2 computes $R' \bowtie_A S$

Semijoin is better if

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

Distributed Query Processing

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dist. INGRES</td>
<td>Dynamic Resp. time or Total time</td>
<td>Msg. Size, Proc. Cost</td>
<td>General or Broadcast</td>
<td>No</td>
<td>1</td>
<td>Horizontal</td>
<td></td>
</tr>
<tr>
<td>R*</td>
<td>Static Total time</td>
<td>No. Msg., Msg. Size, IO, CPU</td>
<td>General or Local</td>
<td>No</td>
<td>1, 2</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SDD-1</td>
<td>Static Total time</td>
<td>Msg. Size</td>
<td>General</td>
<td>Yes</td>
<td>1, 3, 4, 5</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

1: relation cardinality; 2: number of unique values per attribute; 3: join selectivity factor; 4: size of projection on each join attribute; 5: attribute size and tuple size
SDD-1 Algorithm

- Based on the **Hill Climbing Algorithm**
  - Semijoins
  - No replication
  - No fragmentation
  - Cost of transferring the result to the user site from the final result site is not considered
  - Can minimize either total time or response time

---

Hill Climbing Algorithm

Assume join is between three relations.

**Step 1:** Do initial processing

**Step 2:** Select initial feasible solution ($ES_0$)

1. Determine the candidate result sites - sites where a relation referenced in the query exist
2. Compute the cost of transferring all the other referenced relations to each candidate site
3. $ES_0 =$ candidate site with minimum cost

**Step 3:** Determine candidate splits of $ES_0$ into \{ $ES_1$, $ES_2$ \}

1. $ES_1$ consists of sending one of the relations to the other relation’s site
2. $ES_2$ consists of sending the join of the relations to the final result site
Hill Climbing Algorithm

Step 4: Replace $ES_0$ with the split schedule which gives

$$\text{cost}(ES_1) + \text{cost}(\text{local join}) + \text{cost}(ES_2) < \text{cost}(ES_0)$$

Step 5: Recursively apply steps 3–4 on $ES_1$ and $ES_2$ until no such plans can be found

Step 6: Check for redundant transmissions in the final plan and eliminate them.

---

Hill Climbing Algorithm – Example

What are the salaries of engineers who work on the CAD/CAM project?

$$\Pi_{\text{sal}}(\text{PAY} \bowtie_{\text{title}} (\text{EMP} \bowtie_{\text{ens}} (\text{ASG} \bowtie_{\text{pno}} (\sigma_{\text{frame}=\text{CAD/CAM}((\text{PROJ}))))))$$

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>PAY</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>PROJ</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ASG</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

Assume:
- Size of relations is defined as their cardinality
- Minimize total cost
- Transmission cost between two sites is 1
- Ignore local processing cost
Hill Climbing Algorithm – Example

Step 1:
Selection on PROJ; result has cardinality 1

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>PAY</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>PROJ</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ASG</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

Step 2: Initial feasible solution

**Alternative 1:** Resulting site is Site 1

Total cost = $\text{cost(PAY \rightarrow Site 1)} + \text{cost(ASG \rightarrow Site 1)} + \text{cost(PROJ \rightarrow Site 1)}$

$= 4 + 10 + 1 = 15$

**Alternative 2:** Resulting site is Site 2

Total cost = $8 + 10 + 1 = 19$

**Alternative 3:** Resulting site is Site 3

Total cost = $8 + 4 + 10 = 22$

**Alternative 4:** Resulting site is Site 4

Total cost = $8 + 4 + 1 = 13$

Therefore $ES_0 = \{\text{EMP \rightarrow Site 4}; S \rightarrow Site 4; \text{PROJ \rightarrow Site 4}\}$
Hill Climbing Algorithm – Example

Step 3: Determine candidate splits

Alternative 1: \{ES_1, ES_2, ES_3\}
- \(ES_1; EMP \rightarrow Site\ 2\)
- \(ES_2; (EMP \bowtie PAY) \rightarrow Site\ 4\)
- \(ES_3; PROJ \rightarrow Site\ 4\)

Alternative 2: \{ES_1, ES_2, ES_3\}
- \(ES_1; PAY \rightarrow Site\ 1\)
- \(ES_2; (PAY \bowtie EMP) \rightarrow Site\ 4\)
- \(ES_3; PROJ \rightarrow Site\ 4\)

---

Hill Climbing Algorithm – Example

Step 4: Determine costs of each split alternative

\[
\begin{align*}
\text{cost(Alternative 1)} &= \text{cost(EMP}\rightarrow\text{Site 2}) + \text{cost((EMP}\bowtie\text{PAY})\rightarrow\text{Site 4}) + \text{cost(PROJ}\rightarrow\text{Site 4}) \\
&= 8 + 8 + 1 = 17 \\
\text{cost(Alternative 2)} &= \text{cost(PAY}\rightarrow\text{Site 1}) + \text{cost((PAY}\bowtie\text{EMP})\rightarrow\text{Site 4}) + \text{cost(PROJ}\rightarrow\text{Site 4}) \\
&= 4 + 8 + 1 = 13
\end{align*}
\]

Decision: DO NOT SPLIT

Step 5: \(ES_0\) is the “best”.

Step 6: No redundant transmissions.
Hill Climbing Algorithm

Problems:

1. Greedy algorithm → determines an initial feasible solution and iteratively tries to improve it
2. If there are local minima, it may not find global minima
3. If the optimal schedule has a high initial cost, it won't find it since it won't choose it as the initial feasible solution

Example: A better schedule is

\[
\text{PROJ} \rightarrow \text{Site 4} \\
\text{ASG}' = (\text{PROJ} \bowtie \text{ASG}) \rightarrow \text{Site 1} \\
(\text{ASG}' \bowtie \text{EMP}) \rightarrow \text{Site 2} \\
\text{Total cost} = 1 + 2 + 2 = 5
\]

SDD-1 Algorithm

Initialization

Step 1: In the execution strategy (call it ES), include all the local processing

Step 2: Reflect the effects of local processing on the database profile

Step 3: Construct a set of beneficial semijoin operations \( BS \) as follows:

\[
BS = 0 \\
\text{For each semijoin } S_{J_i} \\
BS \leftarrow BS \cup S_{J_i} \text{ if } cost(S_{J_i}) < benefit(S_{J_i})
\]
SDD-1 Algorithm – Example

Consider the following query

\[
\text{SELECT } R3.C \\
\text{FROM } R1, R2, R3 \\
\text{WHERE } R1.A = R2.A \\
\text{AND } R2.B = R3.B
\]

which has the following query graph and statistics:

<table>
<thead>
<tr>
<th>Site 1</th>
<th>Site 2</th>
<th>Site 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>relation</th>
<th>card</th>
<th>tuple size</th>
<th>relation size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>30</td>
<td>50</td>
<td>1500</td>
</tr>
<tr>
<td>R2</td>
<td>100</td>
<td>50</td>
<td>3000</td>
</tr>
<tr>
<td>R3</td>
<td>50</td>
<td>30</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>attribute</th>
<th>SF</th>
<th>size(H_{attribute})</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.A</td>
<td>0.3</td>
<td>36</td>
</tr>
<tr>
<td>R2.A</td>
<td>0.8</td>
<td>320</td>
</tr>
<tr>
<td>R2.B</td>
<td>1.0</td>
<td>400</td>
</tr>
<tr>
<td>R3.B</td>
<td>0.4</td>
<td>80</td>
</tr>
</tbody>
</table>

SDD-1 Algorithm – Example

- Beneficial semijoins:
  - $SJ_1 = R2 \bowtie R1$, whose benefit is $2100 = (1 - 0.3) \times 3000$ and cost is 36
  - $SJ_2 = R2 \bowtie R3$, whose benefit is $1800 = (1 - 0.4) \times 3000$ and cost is 80

- Nonbeneficial semijoins:
  - $SJ_3 = R1 \bowtie R2$, whose benefit is $300 = (1 - 0.8) \times 1500$ and cost is 320
  - $SJ_4 = R3 \bowtie R2$, whose benefit is 0 and cost is 400
SDD-1 Algorithm

Iterative Process

Step 4: Remove the most beneficial $S_{J_i}$ from $BS$ and append it to $ES$

Step 5: Modify the database profile accordingly

Step 6: Modify $BS$ appropriately
  - compute new benefit/cost values
  - check if any new semijoin need to be included in $BS$

Step 7: If $BS \neq \emptyset$, go back to Step 4.

SDD-1 Algorithm – Example

- Iteration 1:
  - Remove $S_{J_i}$ from $BS$ and add it to $ES$.
  - Update statistics
    \[
    \text{size}(R2) = 900 (= 3000+0.3) \\
    S_{K_i}(R2.A) = -0.8+0.3 = -0.24
    \]

- Iteration 2:
  - Two beneficial semijoins:
    \[
    S_{J_2} = R2 \bowtie R3, \text{ whose benefit is } 540 = (1-0.4) \times 900 \text{ and cost is } 200 \\
    S_{J_3} = R1 \bowtie R2, \text{ whose benefit is } 1140 = (1-0.24) \times 1500 \text{ and cost is } 96
    \]
  - Add $S_{J_3}$ to $ES$
  - Update statistics
    \[
    \text{size}(R1) = 360 (= 1500+0.24) \\
    S_{K_i}(R1.A) = -0.3+0.24 = 0.072
    \]
SDD-1 Algorithm – Example

- Iteration 3:
  - No new beneficial semijoins.
  - Remove remaining beneficial semijoin $SJ_2$ from $BS$ and add it to $ES$.
  - Update statistics
    \[ \text{size}(R2) = 360 (= 900 \times 0.4) \]
    Note: selectivity of $R2$ may also change, but not important in this example.

SDD-1 Algorithm

Assembly Site Selection

Step 8: Find the site where the largest amount of data resides and select it as the assembly site

Example:

Amount of data stored at sites:
- Site 1: 360
- Site 2: 360
- Site 3: 2000

Therefore, Site 3 will be chosen as the assembly site.
SDD-1 Algorithm

Postprocessing

Step 9: For each $R_i$ at the assembly site, find the semijoins of the type $R_i \bowtie R_j$

where the total cost of $ES$ without this semijoin is smaller than the cost with it and remove the semijoin from $ES$.

Note: There might be indirect benefits.

Example: No semijoins are removed.

Step 10: Permute the order of semijoins if doing so would improve the total cost of $ES$.

Example: Final strategy:

Send $(R_2 \bowtie R_1) \bowtie R_3$ to Site 3
Send $R_1 \bowtie R_2$ to Site 3

Distributed Query Processing Methodology

Calculus Query on Distributed Relations

GLOBAL SCHEMA

Algebraic Query on Distributed Relations

FRAGMENT SCHEMA

Fragment Query

STATS ON FRAGMENTS

Global Optimization

LOCAL SCHEMAS

Optimized Fragment Query with Communication Operations

Local Optimization

Optimized Local Queries
Step 4 – Local Optimization

**Input:** Best global execution schedule
- Select the best access path
- Use the centralized optimization techniques

Distributed Query Optimization Problems

- **Cost model**
  - multiple query optimization
  - heuristics to cut down on alternatives
- **Larger set of queries**
  - optimization only on select-project-join queries
  - also need to handle complex queries (e.g., unions, disjunctions, aggregations and sorting)
- **Optimization cost vs execution cost tradeoff**
  - heuristics to cut down on alternatives
  - controllable search strategies
- **Optimization/reoptimization interval**
  - extent of changes in database profile before reoptimization is necessary