Distributed Databases

CS347
Lecture 13
May 23, 2001

Expected Background

• Basic SQL
• Relational algebra
• Following aspects of centralized DB
  – Query processing: query plans, cost estimation, optimization
  – Concurrency control techniques
  – Recovery methods

Reading Material

• Primarily lecture notes
• No required textbook
• Some lecture material drawn from

Centralized DBMS

Software:

P

M

...  

Application
SQL Front End
Query Processor
Transaction Proc.
File Access

• Simplifications:
  • single front end
  • one place to keep locks
  • if processor fails, system fails, …..

Distributed DB

• Multiple processors, memories, and disks
  – Opportunity for parallelism (+)
  – Opportunity for enhanced reliability (+)
  – Synchronization issues (-)

• Heterogeneity and autonomy of “components”
  – Autonomy example: may not get statistics for query optimization from a site

Heterogeneity

Select new investments

Application

Stock ticker tape
RDBMS
Portfolio
Files

History of dividends, ratios,...
**Big Picture**

Data management with multiple processors and possible autonomy, heterogeneity. Impacts:
- Data organization
- Query processing
- Access structures
- Concurrency control
- Recovery

**Today’s topics**

- Introductory topics
  - Database architectures
  - Distributed versus Parallel DB systems
- Distributed database design
  - Fragmentation
  - Allocation

**Common DB architectures**

Shared memory

Shared disk

Common DB architectures

Shared nothing

Number of other “hybrid” architectures are possible.

**Selecting the “right” architecture**

- Reliability
- Scalability
- Geographic distribution of data
- Performance
- Cost

**Parallel vs. Distributed DB system**

- Typically, parallel DBs:
  - Fast interconnect
  - Homogeneous software
  - Goals: High performance and Transparency

- Typically, distributed DBs:
  - Geographically distributed
  - Disconnected operation possible
  - Goal: Data sharing (heterogeneity, autonomy)
Typical query processing scenarios

- **Parallel DB:**
  - Distribute/partition/sort… data to make certain DB operations (e.g., Join) fast

- **Distributed DB:**
  - Given data distribution, find query processing strategy to minimize cost (e.g. communication cost)

Distributed DB Design

**Top-down approach:**
- have a database
- how to split and allocate to individual sites

**Multi-databases (or bottom-up):**
- combine existing databases
- how to deal with heterogeneity & autonomy

Two issues in top-down design

- Fragmentation
- Allocation

Note: issues not independent, but studied separately for simplicity.

Example

Employee relation E (#,name,loc,sal,…)

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Loc</th>
<th>Sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Joe</td>
<td>Sa</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>Sally</td>
<td>Sb</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>Tom</td>
<td>Sa</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Sally</td>
<td>Sb</td>
<td>25</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

F = \{F_1, F_2\}

At Sa

At Sb

Qa: select * from E where loc=Sa and…

Qb: select * from E where loc=Sb and...

Motivation: Two sites: Sa, Sb

Qa \rightarrow Sa \leftarrow Sb

Fragmentation

- **Horizontal**
  - Primary depends on local attributes
  - Derived depends on foreign relation

- **Vertical**
Horizontal partitioning techniques

- Round robin
- Hash partitioning
- Range partitioning

Round robin

<table>
<thead>
<tr>
<th>R</th>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
</tr>
<tr>
<td>t₂</td>
<td>t₂</td>
<td>t₃</td>
<td>t₄</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Evenly distributes data
- Good for scanning full relation
- Not good for point or range queries

Hash partitioning

<table>
<thead>
<tr>
<th>R</th>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁ → h(k₁)=2</td>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
</tr>
<tr>
<td>t₂ → h(k₂)=0</td>
<td>t₂</td>
<td>t₃</td>
<td>t₄</td>
</tr>
<tr>
<td>t₃ → h(k₃)=0</td>
<td>t₃</td>
<td>t₄</td>
<td>...</td>
</tr>
<tr>
<td>t₄ → h(k₄)=1</td>
<td>t₄</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Good for point queries on key; also for joins
- Not good for range queries; point queries not on key
- Good hash function \( \Rightarrow \) even distribution

Range partitioning

<table>
<thead>
<tr>
<th>R</th>
<th>F₀</th>
<th>F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=5</td>
<td>t₁</td>
<td>t₂</td>
<td>t₃</td>
</tr>
<tr>
<td>A=8</td>
<td>4</td>
<td>7</td>
<td>t₄</td>
</tr>
<tr>
<td>A=2</td>
<td>t₀</td>
<td>t₁</td>
<td>t₄</td>
</tr>
</tbody>
</table>
| A=3 | ... | ... | ...

- Good for some range queries on A
- Need to select good vector: else create imbalance
  \( \Rightarrow \) data skew
  \( \Rightarrow \) execution skew

Which are good fragmentations?

Example 1: \( \mathbf{F} = \{ \mathbf{F}_1, \mathbf{F}_2 \} \)

\( \mathbf{F}_1 = \sigma_{\text{sal} < 10}(E) \quad \mathbf{F}_2 = \sigma_{\text{sal} > 20}(E) \)

**Problem:** Some tuples lost!

Example 2: \( \mathbf{F} = \{ \mathbf{F}_3, \mathbf{F}_4 \} \)

\( \mathbf{F}_1 = \sigma_{\text{sal} < 10}(E) \quad \mathbf{F}_2 = \sigma_{\text{sal} > 5}(E) \)

**Tuples with 5 < \text{sal} < 10 are duplicated**

Prefer to deal with replication explicitly

Example: \( \mathbf{F} = \{ \mathbf{F}_5, \mathbf{F}_6, \mathbf{F}_7 \} \)

\( \mathbf{F}_5 = \sigma_{\text{sal} < 5}(E) \)
\( \mathbf{F}_6 = \sigma_{\text{sal} < 10}(E) \)
\( \mathbf{F}_7 = \sigma_{\text{sal} > 10}(E) \)

\( \Rightarrow \) Then replicate \( \mathbf{F}_6 \) if desired as part of allocation
Horizontal Fragmentation Desiderata

\[ R \Rightarrow F = \{F_1, F_2, \ldots\} \]

1. **Completeness**
   \[ \forall t \in R, \exists F_i \in F \text{ such that } t \in F_i \]

2. **Disjointness**
   \[ F_i \cap F_j = \emptyset, \forall i, j \text{ such that } i \neq j \]

3. **Reconstruction**
   \[ \exists \nabla \text{ such that } R = \nabla F_i \]

Generating horizontal fragments

- Given simple predicates \( P_r = \{p_1, p_2, \ldots, p_n\} \) and relation \( R \).
- Generate “minterm” predicates
  \[ M = \{m \mid m = \bigwedge p_{k*}, 1 \leq k \leq m\}, \text{ where } p_{k*} \text{ is either } p_k \text{ or } \neg p_k \]
- Eliminate useless minterms and simplify \( M \) to get \( M' \).
- Generate fragments \( \sigma_m(R) \) for each \( m \in M' \).

Example

Example: say queries use predicates
\[ A < 10, A > 5, \text{Loc} = \text{SA}, \text{Loc} = \text{SB} \]

- Eliminate and simplify minterms
  \[ A < 10 \land A > 5 \land \text{Loc} = \text{SA} \land \text{Loc} = \text{SB} \]
  \[ A < 10 \land A > 5 \land \text{Loc} = \text{SA} \land \neg \text{Loc} = \text{SB} \]

- Final set of fragments
  \[ (5 < A < 10) \land (\text{Loc} = \text{SA}) \]
  \[ (5 < A < 10) \land (\text{Loc} = \text{SB}) \]
  \[ (A \leq 5) \land (\text{Loc} = \text{SA}) \]
  \[ (A \geq 10) \land (\text{Loc} = \text{SA}) \]
  \[ (A \geq 10) \land (\text{Loc} = \text{SB}) \]
  \[ (\text{Loc} = \text{SA}) \land (\text{Sal} < 10) \]
  \[ (\text{Loc} = \text{SA}) \land (\text{Sal} \geq 10) \]

More on Horizontal Fragmentation

- Elimination of useless fragments/predicates depends on application semantics:
  - e.g.: if \( \text{Loc} \neq \text{SA} \) and \( \neq \text{SB} \) is possible, must retain fragments such as \( (5 < A < 10) \land (\text{Loc} = \text{SA}) \land (\text{Loc} = \text{SB}) \)
- Minterm-based fragmentation generates complete, disjoint, and reconstructible fragments.

Choosing simple predicates

- \( E (\#, \text{name}, \text{loc}, \text{sal}, \ldots) \) with common queries
  - \( Q_a: \text{Select } \ast \text{ from } E \text{ where } \text{loc} = \text{SA} \text{ and...} \)
  - \( Q_b: \text{Select } \ast \text{ from } E \text{ where } \text{loc} = \text{SB} \text{ and...} \)
- Three choices for \( P_r \) and hence \( F[P_r] \):
  - \( P_r = \{\} \rightarrow F = \{E\} \)
  - \( P_r = \{\text{Loc} = \text{SA}, \text{Loc} = \text{SB}\} \rightarrow F = \{\sigma_{\text{loc} = \text{SA}}(E), \sigma_{\text{loc} = \text{SB}}(E)\} \)
  - \( P_r = \{\text{Loc} = \text{SA}, \text{Loc} = \text{SB}, \text{Sal} < 10\} \rightarrow F = \{\sigma_{\text{loc} = \text{SA}} \land \sigma_{\text{loc} = \text{SB}} \land \text{Sal} < 10(E)\} \)

- \( Q_a: \text{Select } \ast \text{ from } E \text{ where } \text{loc} = \text{SA} \text{ and...} \)
  - Prefer \( F_2 \) to \( F_1 \) and \( F_3 \).
Desiderata for simple predicates

- **Completeness**
  Set of predicates \( P_r \) is complete if for every \( F_i \in F[P_r] \), every \( t \in F_i \) has equal probability of access by every major application.

- **Minimality**
  Set of predicates \( P_r \) is minimal if no \( P_r' \subset P_r \) is complete.

To get complete and minimal \( P_r \), use predicates that are “relevant” in frequent queries.

 Derived horizontal fragmentation

- Example: Two relations Employee and Jobs
  \( E(#, \text{NAME}, \text{SAL}, \text{LOC}) \)
  \( J(#, \text{DES},...) \)

- Fragment \( E \) into \( \{E_1, E_2\} \) by \( \text{LOC} \)

- Common query:
  “Given employee name, list projects (s)he works in”

### Example

<table>
<thead>
<tr>
<th>#</th>
<th>NM</th>
<th>Loc</th>
<th>Sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Joe</td>
<td>Sa</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Tom</td>
<td>Sa</td>
<td>15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\( E_1 \) (at \( \text{Sa} \))

\( E_2 \) (at \( \text{Sb} \))

#### J

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>work on 347 hw</td>
</tr>
<tr>
<td>7</td>
<td>go to moon</td>
</tr>
<tr>
<td>8</td>
<td>build table</td>
</tr>
<tr>
<td>12</td>
<td>rest</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### J

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>work on 347 hw</td>
</tr>
<tr>
<td>7</td>
<td>go to moon</td>
</tr>
<tr>
<td>8</td>
<td>build table</td>
</tr>
<tr>
<td>12</td>
<td>rest</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

\( J_1 = J \times E_1 \)

\( J_2 = J \times E_2 \)

### Completeness of derived fragmentation

Example: Say \( J \) is

<table>
<thead>
<tr>
<th>#</th>
<th>Des</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>build chair</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- \( J_1 \cup J_2 \subset J \) (incomplete fragmentation)
- For completeness, enforce referential integrity constraint
  
  join attribute of member relation
  
  joint attribute of owner relation
Properties

\[ \mathcal{R}[\mathcal{T}] \Rightarrow \mathcal{R}_i[\mathcal{T}_i], i = 1..n \]

- Completeness: \( \bigcup \mathcal{T}_i = \mathcal{T} \)
- Reconstruction: \( \bigtriangledown \bigtriangledown \mathcal{R}_i = \mathcal{R} \) (lossless join)
  - One way to guarantee lossless join: repeat key in each fragment, i.e., key \( \subseteq \mathcal{T}_i, \forall i \)
- Disjointness: \( \mathcal{T}_i \cap \mathcal{T}_j = \{\text{key}\} \)
  - Check disjointness only on non-key attributes

Attribute affinity matrix

\[
\begin{array}{cccccc}
A_1 & A_2 & A_3 & A_4 & A_5 \\
78 & 50 & 45 & 1 & 0 \\
50 & 25 & 28 & 2 & 0 \\
45 & 28 & 34 & 0 & 4 \\
1 & 2 & 0 & 20 & 75 \\
0 & 0 & 4 & 75 & 40 \\
\end{array}
\]

- \( A_{ij} \) is a measure of how “often” \( A_i \) and \( A_j \) are accessed by the same query
- Hand constructed using knowledge of queries and their frequencies

Grouping Attributes

\begin{align*}
\mathcal{E}_1(\#,\text{NM,LOC}) & \Rightarrow \mathcal{E}_1(\#,\text{SAL}) \\
\mathcal{E}_2(\#,\text{NM}) & \Rightarrow \mathcal{E}_2(\#,\text{LOC}) \\
\mathcal{E}_3(\#,\text{SAL}) & \Rightarrow \mathcal{E}_3(\#,\text{SAL}) \\
\end{align*}

Which is the right vertical fragmentation?

Allocation

Example: \( \mathcal{E} \Rightarrow \mathcal{F}_1 = \sigma_{\text{loc=Sa}}(\mathcal{E}); \mathcal{F}_2 = \sigma_{\text{loc=Sb}}(\mathcal{E}) \)

- Do we replicate fragments?
- Where do we place each copy of each fragment?

Common way to enforce disjointness: make join attribute key of owner relation.
### Issues
- Origin of queries
- Communication cost and size of answers, relations, etc.
- Storage capacity, storage cost at sites, and size of fragments
- Processing power at the sites
- Query processing strategy
  - How are joins done? Where are answers collected?
- Fragment replication
  - Update cost, concurrency control overhead

### Optimization problem
- What is the best placement of fragments and/or best number of copies to:
  - minimize query response time
  - maximize throughput
  - minimize “some cost”
  - ...
- Subject to constraints
  - Available storage
  - Available bandwidth, processing power,…
  - Keep 90% of response time below X
  - ...

Very hard problem

### Looking Ahead
- Query processing
  - Decomposition
  - Localization
  - Distributed query operators
  - Optimization (briefly)

### Resources
- Ozsu and Valduriez. “Principles of Distributed Database Systems” – Chapter 5