**Distributed Databases**

CPS 216
Advanced Database Systems

---

**Centralized versus distributed DBMS**

- Processor
- Memory
- Disk

---

**Parallel versus distributed DBMS**

- **Parallel DBMS**
  - Fast interconnect
  - Homogeneous hardware/software
  - Total control over components

- **Distributed DBMS**
  - Geographically distributed
  - Disconnected operations possible
  - Heterogeneous hardware/software
  - Performance, data formats, data processing capabilities
  - Autonomy of individual sites

---

**Distributed DBMS issues**

- Database management with multiple sites that are possibly autonomous and heterogeneous
  - Data organization
  - Query processing and optimization
  - Concurrency control and recovery

---

**Data organization**

- **Top-down approach**
  - Have a database
  - How to partition and/or replicate it across sites

- **Bottom-up approach**
  - Have existing databases at different sites
  - How to integrate them together and deal with heterogeneity and autonomy

- **Focus for today**
  - Data partitioning using a top-down approach

---

**Partitioning schemes**

- **Horizontal**

- **Vertical**

- **Or hybrid**
Horizontal partitioning schemes

- Round-robin partitioning
- Hash partitioning
- Range partitioning
- Predicate-based partitioning
- Derived horizontal partitioning

Properties of a correct partitioning

\[ R \rightarrow \{ R_1, R_2, \ldots, R_k \} \]

- Completeness and reconstructability
  \[ R = R_1 \cup R_2 \cup \ldots \cup R_k \]
- Disjointness
  \[ R_i \cap R_j = \emptyset \text{ for any } i \neq j \]

Round-robin partitioning

<table>
<thead>
<tr>
<th>R</th>
<th>( R_0 )</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( t_1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td></td>
<td>( t_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_3 )</td>
<td></td>
<td></td>
<td>( t_3 )</td>
<td></td>
</tr>
<tr>
<td>( t_4 )</td>
<td></td>
<td></td>
<td></td>
<td>( t_4 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Evenly distributes data
- Good for full relation scans
- Not good for range queries

Hash partitioning

<table>
<thead>
<tr>
<th>R</th>
<th>( R_0 )</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_3 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_4 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Evenly distributes data (assuming a good hash function)
- Good for point queries and equijoins on the partitioning attribute
- Not good for range queries

Range partitioning

<table>
<thead>
<tr>
<th>R partitioning vector: (&lt;4, 7&gt;)</th>
<th>( R_0 )</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( k_1 = 5 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( k_2 = 8 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( k_3 = 2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( k_4 = 3 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Good for range queries on the partitioning attribute
- The choice of partitioning vector is important
  - Bad vector may result in both data skew and execution skew

Predicate-based partitioning

- Fragmentation
  - Decide how to divide a relation horizontally into fragments using a set of predicates

- Allocation
  - Decide which fragments go to which site
Predicate-based fragmentation

- Given a relation $R$ and a set of simple predicates $P = \{ p_1, p_2, \ldots, p_n \}$
- Generate minterm predicates
  - $M = \{ m \mid m = \land_{1 \leq k \leq n} p_k^* \}$, where $p_k^*$ is either $p_k$ or $\neg p_k$
  - Simplify minterms in $M$ and eliminate useless ones
- For each $m$ in $M$, generate a fragment $\sigma_m R$

Example

- Say queries use simple predicates: $A < 10$, $A > 5$, $D = 'CS'$, $D = 'EE'$
- Generate, simplify, and eliminate minterms
  - $A < 10 \land A > 5 \land D = 'CS'$, $D = 'EE'$ eliminated
  - $A < 10 \land A \leq 5 \land D = 'CS'$, $D = 'EE'$, $A \leq 5 \land D = 'CS'$
- Final set of fragments
  - $\sigma_{A < 10 \land D = 'CS'} R$
  - $\sigma_{A < 10 \land D = 'EE'} R$
  - $\sigma_{A \leq 5 \land D = 'CS'} R$
  - $\sigma_{A \leq 5 \land D = 'EE'} R$

Choice of simple predicates

- Completeness
  - There is an equal probability of access by every application to any two tuples in the same minterm fragment
    - If $p$ is used in fragmentation, then $\sigma p R$ either accesses all tuples in a fragment or none in a fragment
- Minimality
  - If a predicate causes a fragment $f$ to be further fragmented into $f_i$ and $f_j$, there should at least one application that accesses $f_i$ and $f_j$ differently
  » Use all relevant predicates in frequent queries!

Allocation of fragments

- Tough optimization problem
  - Do we replicate fragments?
  - Where we place each copy of each fragment?
- Metrics: minimize query response time; maximize throughput; minimize network traffic; …
- Constraints: available storage, bandwidth, processing power; response time requirement; …
- Issues: origin of queries; selectivity of fragments; query processing strategies; consistency enforcement; …