Part II
Distributed Database Systems

4 Distributed DBS Architecture

Overview

Contents
4.1 Foundations of DDBS

Architecture & Data Distribution

Dimensions
12 Rules for DDBMS by Date

1. Local Autonomy
   • Component system have maximal control over own data, local access does not require access to other components

2. No reliance on central site
   • Local components can perform independently of central component

3. Continuous operation/high availability
   • Overall system performs despite local interrupt

4. Location transparency
   • User of overall system should not be aware of physical storage location

5. Fragmentation transparency
   • If data of one relation is fragmented, user should not be aware of this

6. Replication transparency
   • User should not be aware of redundant copies of data
   • Management and redundancy is controlled by DBMS

7. Distributed query processing
   • Efficient access to data stored on different sites within one DB operation

8. Distributed Transaction Management
   • ACID properties must persist for distributed operations

9. Hardware independence
   • Component DB processing on different hardware platforms

10. Operating system independence
    • Component DB processing on different OS

11. Network independence
    • DB processing using different network protocols

12. DBMS independence (ideal)
    • Usage of different DBMS possible
**Global conceptual schema (GCS)**
- Logical structure of overall DB
- Supported by all nodes
- Ensures *transparency*

**Global distribution schema (GDS)**
- Describes fragmentation, replication, allocation
System Architecture

<table>
<thead>
<tr>
<th>Global Query Processing</th>
<th>Global Catalog Management</th>
<th>Replica Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Recovery</td>
<td>Global Transaction Management</td>
<td>Global Synchronisation</td>
</tr>
</tbody>
</table>

"normal DBMS"

local Component

global Component
4.2 Catalog Management

Catalog Management

- Catalog: collection of metadata (schema, statistics, access rights, etc.)
  - Local catalog
    * Identical to catalog of a centralized DBS
    * consists of LIS and LCS
  - Global catalog
    * Also contains GCS and GDS
    * System-wide management of users and access rights

- Storage
  - Local catalog: on each node
  - Global catalog: centralized, replicated, or partitioned

Global Catalog /1

- Centralized: one instance of global catalog managed by central node
  - Advantages: only one update operation required, little space consumption
  - Disadvantages: request for each query, potential bottleneck, critical resource

- Replicated: full copy of global catalog stored on each node
  - Advantage: low communication overhead during queries, availability
  - Disadvantage: high overhead for updates

- Mix-form: cluster-catalog with centralized catalog for certain clusters of nodes

Global Catalog /2

- Partitioned: (relevant) part of the catalog is stored on each node
  - No explicit GCS → union of LCS
  - Partitioned GDS by extend object (relations, etc.) names (see System R*).
**Coherency Control**

- Idea: buffer for non-local parts of the catalog
  - Avoids frequent remote accesses for often used parts of the catalog
- Problem: invalidation of buffered copies after updates

**Coherency Control /2**

- Approaches
  - Explicit invalidation:
    * Owner of catalog data keeps list of copy sites
    * After an update these nodes are informed of invalidation
  - Implicit invalidation:
    * Identification of invalid catalog data during processing time using version numbers or timestamps (see System R*)

**DB Object Name Management**

- Task: identification of relations, views, procedures, etc.
- Typical schema object names in RDBMS: `[<username>.]<objectname>`
- Requirement global uniqueness in DDBS
  - Name Server approach: management of names in centralized catalog
  - Hierarchic Naming: enrich object name with *node name* `[ [<nodename>.] <username>.]<objectname>`
    * Node name: birth site (or simplification via alias)

**Name Management: Node Types**

```
global Name  
|-----------|
|   Birth site
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
|        Catalog site
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
|    Store site  Store site  Store site
```
Catalog Management in System R*

- Birth site
  - Prefix of the relation name
  - Knows about storage sites

- Query processing
  - Executing node gets catalog entry of relevant relation
  - Catalog entry is buffered for later accesses

Catalog Management in System R*/2

- Query processing (continued)
  - Partial query plans include time stamp of catalog entry
  - Node processing partial query checks whether catalog time stamp is still current

- In case of failure: buffer invalidation, re-set query and new query translation according to current schema

- Summary:
  - Advantage: high degree of autinomy, user-controlled invalidation of buffered catalog data, good performance
  - Disadvantage: no uniform realization of global views
4.3 DDBS Design: Fragmentation

Database Distribution

- In Shared-Nothing-Systems (DDBS): definition of physical distribution of data

- Impact:
  - Communication efforts $\leadsto$ overall performance
  - Load balancing
  - Availability

Bottom Up vs. Top Down

- Bottom Up
  - Subsumption of local conceptual schemata (LCS) into global conceptual schema (GCS)
  - Integration of existing DB $\leadsto$ schema integration (Federated DBS)

- Top Down
  - GCS of local DB designed first
  - Distribution of schema to different nodes
  - *Distribution Design*

Distribution Design Tasks

![Distribution Design Diagram]

- global Relation R
- Fragments
- Allocations

Node 1
- R1, R2, R3

Node 2
- R3, R4.1

Node 3
- R4.2

Node 1
- R1

Node 2
- R3

Node 3
- R4.2
Fragmentation

- Granularity of distribution: relation
  - Operations on one relation can always be performed on one node
  - Simplifies integrity control

- Granularity of distribution: fragments of relations
  - Grants locality of access
  - Load balancing
  - Reduced processing costs for operations performed only on part of the data
  - Parallel processing

Fragmentation /2

- Approach:
  - Column- or tuple-wise decomposition (vertical/horizontal)
  - Described using relational algebra expressions (queries)
  - Important rules/requirements
    - Completeness
    - Reconstructability
    - Disjointness

Example Database

<table>
<thead>
<tr>
<th>MEMBER</th>
<th>SName</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Ian</td>
<td>SW Developer</td>
</tr>
<tr>
<td>M2</td>
<td>Levon</td>
<td>Analyst</td>
</tr>
<tr>
<td>M3</td>
<td>Tom</td>
<td>SW Developer</td>
</tr>
<tr>
<td>M4</td>
<td>Moe</td>
<td>Manager</td>
</tr>
<tr>
<td>M5</td>
<td>David</td>
<td>HW-Developer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PName</th>
<th>Budget</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>DB Development</td>
<td>200,000</td>
<td>MD</td>
</tr>
<tr>
<td>P2</td>
<td>Hardware Dev.</td>
<td>150,000</td>
<td>M</td>
</tr>
<tr>
<td>P3</td>
<td>Web-Design</td>
<td>100,000</td>
<td>MD</td>
</tr>
<tr>
<td>P4</td>
<td>Customizing</td>
<td>250,000</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSIGNMENT</th>
<th>MNr</th>
<th>PNr</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>P4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M2</td>
<td>P4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>M3</td>
<td>P4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>M4</td>
<td>P3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>P2</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALARY</th>
<th>Position</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SW Developer</td>
<td>60,000</td>
</tr>
<tr>
<td></td>
<td>HW-Developer</td>
<td>55,000</td>
</tr>
<tr>
<td></td>
<td>Analyst</td>
<td>65,000</td>
</tr>
<tr>
<td></td>
<td>Manager</td>
<td>90,000</td>
</tr>
</tbody>
</table>
Primary Horizontal Fragmentation

- "Tupel-wise" decomposition of a global relation $R$ into $n$ fragments $R_i$
- Defined by $n$ selection predicates $P_i$ on attributes from $R$
  \[ R_i := \sigma_{P_i}(R) \quad (1 \leq i \leq n) \]
- $P_i$: fragmentation predicates
- Completeness: each tuple from $R$ must be assigned to a fragment
- Disjointness: decomposition into disjoint fragments $R_i \cap R_j = \emptyset$ $(1 \leq i, j \leq n, i \neq j)$.
- Reconstructability: $R = \bigcup_{1 \leq i \leq n} R_i$

Primary Horizontal Fragmentation /2

- Example: fragmentation of PROJECT by predicate on location attribute "Loc"

<table>
<thead>
<tr>
<th>PROJECT</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PN</td>
<td>PName</td>
<td>Budget</td>
<td>Loc</td>
</tr>
<tr>
<td>PROJECT1</td>
<td>P2</td>
<td>Hardware Dev.</td>
<td>150.000</td>
<td>M</td>
</tr>
<tr>
<td>PROJECT2</td>
<td>P4</td>
<td>Customizing</td>
<td>250.000</td>
<td>B</td>
</tr>
<tr>
<td>PROJECT3</td>
<td>P5</td>
<td>Web-Design</td>
<td>100.000</td>
<td>MD</td>
</tr>
<tr>
<td></td>
<td>PN</td>
<td>PName</td>
<td>Budget</td>
<td>Loc</td>
</tr>
<tr>
<td>PROJECT</td>
<td>P1</td>
<td>DB Development</td>
<td>200.000</td>
<td>MD</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>Web-Design</td>
<td>100.000</td>
<td>MD</td>
</tr>
</tbody>
</table>
Derived Horizontal Fragmentation

- Fragmentation definition of relation $S$ derived from existing horizontal fragmentation of relation $R$
- Using foreign key relationships
- Relation $R$ with $n$ fragments $R_i$
- Decomposition of depending relation $S$
  \[
  S_i = S \bowtie R_i = S \bowtie \sigma_{P_i}(R) = \pi_{S,*}(S \bowtie \sigma_{P_i}(R))
  \]
- $P_i$ defined only on $R$
- Reconstructability: see above
- Disjointness: implied by disjointness of $R$-fragments
- Completeness: granted for lossless semi-join (no null-values for foreign key in $S$)

Derived Horizontal Fragmentation /2

- Fragmentation of relation ASSIGNMENT derived from fragmentation of PROJECT relation

\[
\begin{array}{ccc}
\text{ASSIGNMENT}_1 = & \text{ASSIGNMENT} \bowtie \text{PROJECT}_1 \\
\text{ASSIGNMENT}_2 = & \text{ASSIGNMENT} \bowtie \text{PROJECT}_2 \\
\text{ASSIGNMENT}_3 = & \text{ASSIGNMENT} \bowtie \text{PROJECT}_3 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>ASSIGNMENT</th>
<th>MN</th>
<th>PN</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT</td>
<td>M1</td>
<td>P1</td>
<td>1</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>M2</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>M3</td>
<td>P1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSIGNMENT</th>
<th>MN</th>
<th>PN</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGNMENT</td>
<td>M1</td>
<td>P1</td>
<td>2</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>M2</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>M3</td>
<td>P1</td>
<td>3</td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>M3</td>
<td>P3</td>
<td>4</td>
</tr>
</tbody>
</table>
Vertical Fragmentation

- Column-wise decomposition of a relation using relational projection
- Completeness: each attribute must be in at least one fragment
- Reconstructability: through natural join \( \leadsto \) primary key of global relation must be in each fragment

\[
R_i := \pi_{K,A_1,\ldots,A_j}(R) \\
R = R_1 \times R_2 \times \cdots \times R_n
\]

- Limited disjointness

Vertical Fragmentation /2

- Fragmentation of PROJECT-Relation regarding Budget and project name / location

\[
\text{PROJECT}_1 = \pi_{PNr,PName,Loc}(\text{PROJECT}) \\
\text{PROJECT}_2 = \pi_{PNr,Budget}(\text{PROJECT})
\]

<table>
<thead>
<tr>
<th>PROJECT1</th>
<th>PROJECT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PN</td>
<td>Name</td>
</tr>
<tr>
<td>P1</td>
<td>DB Development</td>
</tr>
<tr>
<td>P2</td>
<td>Hardware Dev.</td>
</tr>
<tr>
<td>P3</td>
<td>Web-Design</td>
</tr>
<tr>
<td>P4</td>
<td>Customizing</td>
</tr>
</tbody>
</table>
Hybrid Fragmentation

- Fragment of a relation → is relation itself
- Can be subject of further fragmentation
- Also possible: combination of horizontal and vertical fragmentation
Fragmentation transparency

- Decomposition of a relation is for user/application not visible
- Only view on global relation
- Requires mapping of DB operations to fragments by DDBMS
- Example
  - Transparent: `select * from Project where PNr=P1
                  select * from Project1 where PNr=P1
                  if not-found then
                    select * from Project2 where PNr=P1
                    if not-found then
                      select * from Project3 where PNr=P1`
  - Without transparency:
    `select * from Project2 where PNr=P1
    if not-found then
      select * from Project3 where PNr=P1`

Fragmentation transparency /2

- Example (continued)
  - Transparent: `update Project set Ort='B' where PNr=P3
                 select PNr, PName, Budget
                 into :PNr, :PName, :Budget
                 from Project3 where PNr=P3`
  - Without transparency:
    `insert into Project2
     values (:PNr, :PName, :Budget, 'B')
    delete from Project3 where PNr=P3`
Computation of an optimal Fragmentation

- In huge systems with many relations/nodes: intuitive decomposition often too complex/not possible
- In this case: systematic process based on access characteristics
  - Kind of access (read/write)
  - Frequency
  - Relations / attributes
  - Predicates in queries
  - Transfer volume and times

Optimal horizontal Fragmentation

- Based on [Özsu/Valduriez 99] and [Dadam 96]
- Given: relation \( R(A_1, \ldots, A_n) \), operator \( \theta \in \{<, \leq, >, \geq, =, \neq \} \), Domain \( \text{dom}(A_i) \)
- Definition: simple predicate \( p_i \) of the form \( A_j \theta \text{const} \) with \( \text{const} \in \text{dom}(A_j) \)
  - Defines possible binary fragmentation of \( R \)
  - Example:
    \[
    \text{PROJECT}\text{1} = \sigma_{\text{Budget}>150.000}(\text{PROJECT})
    \]
    \[
    \text{PROJECT}\text{2} = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECT})
    \]
- Definition: Minterm \( m \) is conjunction of simple predicates as \( m = p_1^{*} \wedge p_2^{*} \wedge \cdots \wedge p_n^{*} \) with \( p_i^{*} = p_i \) oder \( p_i^{*} = \neg p_i \)

Optimal horizontal Fragmentation /2

- Definition: Set \( M_n(P) \) of all n-ary Minterms for the set \( P \) of simple predicates:
  \[
  M_n(P) = \{ m \mid m = \bigwedge_{i=1}^n p_i^{*}, p_i \in P \}
  \]
  - Defines complete fragmentation of \( R \) without redundancies
    * \( R = \bigcup_{m \in M_n(P)} \sigma_m(R) \)
    * \( \sigma_m \cap \sigma_m = \emptyset, \forall m_i, m_j \in M_n(P), m_i \neq m_j \)
Optimal horizontal Fragmentation

• Completeness and no redundancy not sufficient:
  - \( P = \{ \text{Budget < 100.000, Budget > 200.000, Ort = 'MD', Ort = 'B' } \} \)
  - Minterm \( p_1 \land p_2 \land p_3 \land p_4 \) not satisfiable; but \( \neg p_1 \land \neg p_2 \land \neg p_3 \land \neg p_4 \)

• Identification of \textit{practically relevant} Minterms \( M(P) \)
  1. \( M(P) := M_n(P) \)
  2. Remove irrelevant Minterms from \( M(P) \)

Elimination of irrelevant Minterms

1. Elimination of unsatisfiable Minterms If two terms \( p^*_i \) and \( p^*_j \) in one \( m \in M(P) \) contradict, \( m \) is not satisfiable and can be removed from \( M(P) \).

2. Elimination of dependent predicates If a \( p^*_i \) from \( m \in M(P) \) implies another term \( p^*_j \) (e.g. functional dependency, overlapping domains), \( p^*_j \) can be removed from \( m \).

3. Relevance of a fragmentation
   - Minterms \( m_i \) and \( m_j \), \( m_i \) contains \( p_i \), \( m_j \) contains \( \neg p_i \)
   - Access statistics: \( \text{acc}(m) \) (e.g. derived from query log)
   - Fragment size: \( \text{card}(f) \) (derived from data distribution statistics)
   - \( p_i \) is \textit{relevant}, if \( \frac{\text{acc}(m_i)}{\text{card}(f_i)} \neq \frac{\text{acc}(m_j)}{\text{card}(f_j)} \)

Algorithm \textsc{HorizFragment}

• Identification of a complete, non-redundant and minimal horizontal fragmentation of a relation \( R \) for a given set of predicates \( P \)

• Input:
  - \( P \): set of predicates over \( R \)

• (Intermediate) Results:
  - \( M(P) \): set of relevant Minterms
  - \( F(P) \): set of Minterm-fragments from \( R \)

\[ R(m) := \sigma_m(R) \text{ with } m \in M(P) \]
Algorithm HORIZFRAGMENT

forall $p \in P$ do
  $Q' := Q \cup \{p\}$
  compute $M(Q')$ and $F(Q')$
  compare $F(Q')$ with $F(Q)$
  if $F(Q')$ significant improvement over $F(Q)$ then
    $Q := Q'$
    forall $q \in Q \setminus \{p\}$ do /* unnecessary Fragmentation? */
      $Q' := Q \setminus \{q\}$
      compute $M(Q')$ and $F(Q')$
      compare $F(Q')$ with $F(Q)$
      if $F(Q)$ no significant improvement over $F(Q')$ then
        $Q := Q'$ /* d.h., remove $q$ from $Q$ */
    end
  end
end
4.4 Allocation and Replication

Allocation and Replication

- Allocation
  - Assignment of relations or fragments to physical storage location
  - Non-redundant: fragments are stored in only one place $\rightarrow$ partitioned DB
  - Redundant: fragments can be stored more than once $\rightarrow$ replicated DB

- Replication
  - Storage of redundant copies of fragments or relations
  - Full: Each global relation stored on every node (no distribution design, no distributed query processing, high costs for storage and updates)
  - Partial: Fragments are stored on selected nodes

Allocation and Replication /2

- Aspects of allocation
  - Efficiency:
    * Minimization of costs for remote accesses
    * Avoidance of bottlenecks
  - Data security:
    * Selection of nodes depending on their "reliability"
Identification of an optimal Allocation

- Cost model for non-redundant allocation [Dadam 96]
- Goal: Minimize storage and transfer costs \( \sum_{\text{Storage}} + \sum_{\text{Transfer}} \) for \( K \) fragments and \( L \) nodes

- Storage costs:
  \[
  \sum_{\text{Storage}} = \sum_{p,i} S_p D_{pi} SC_i
  \]
  - \( S_p \): Size of fragment \( p \) in data units
  - \( SC_i \): Storage Costs per data unit on node \( i \)
  - \( D_{pi} \): Distribution of fragment with \( D_{pi} = 1 \) if \( p \) stored on node \( i \), 0 otherwise

Identification of an optimal Allocation /2

- Transfer costs:
  \[
  \sum_{\text{Transfer}} = \sum_{i,t,p,j} F_{it} O_{tp} D_{pj} TC_{ij} + \sum_{i,t,p,j} F_{it} R_{tp} D_{pj} TC_{ji}
  \]
  - \( F_{it} \): Frequency of operation of type \( t \) on node \( i \)
  - \( O_{tp} \): Size of operation \( t \) for fragment \( p \) in data units (e.g. size of query string)
  - \( TC_{ij} \): Transfer Costs from node \( i \) to \( j \) in data units
  - \( R_{tp} \): Size of the result of one operation of type \( t \) on fragment \( p \)

Identification of an optimal Allocation /3

- Additional constraints:
  \[
  \sum_i D_{pi} = 1 \quad \text{for} \quad p = 1, \ldots, K
  \]
  \[
  \sum_p S_p D_{pi} \leq M_i \quad \text{for} \quad p = i, \ldots, L
  \]
  where \( M_i \) is max. storage capacity on node \( i \)

- Integer optimization problem
- Often heuristic solution possible:
  - Identify relevant candidate distributions
  - Compute costs and compare candidates
Identification of an optimal Allocation /4

- Cost model for redundant replication
- Additional constraints slightly modified:

\[ \sum_i D_{pi} \geq 1 \text{ for } p = 1, \ldots, K \]
\[ \sum_p S_p D_{pi} \leq M_i \text{ if } p = i, \ldots, L \]

Identification of an optimal Allocation /5

- Transfer costs
  - Read operations on \( p \) send from node \( i \) to \( j \) with minimal \( TC_{ij} \) and \( D_{pj} = 1 \)
  - Update operations on \( p \) send to all nodes \( j \) with \( D_{pj} = 1 \)
  - \( \Phi_t \): of an operation \( \sum \) (in case of update) or \( \min \) (in case of read operation)

\[ \sum_{T \text{ transfer}} = \sum_{i,t,p} F_{it} \Phi_t \left( O_{tp} TC_{tj} + R_{tp} TC_{ji} \right) \]

Evaluation of Approaches

- Model considering broad spectrum of applications
- Exact computation possible
- But:
  - High computation efforts (optimization problem)
  - Exact input values are hard to obtain