Distributed Database Systems
Fall 2016

Distributed Database Design
SL02

- Design problem
- Design strategies (top-down, bottom-up)
- Fragmentation (horizontal, vertical)
- Allocation and replication of fragments, optimality, heuristics
Design Problem

- **Design problem of distributed systems**: Making decisions about the placement of data and programs across the sites of a computer network as well as possibly designing the network itself.

- In DDBMS, the distribution of applications involves
  - Distribution of the DDBMS software
  - Distribution of applications that run on the database

- Distribution of applications will not be considered in the following; instead the distribution of data is studied.
## Framework of Distribution

- Dimension for the analysis of distributed systems
  - Level of sharing: no sharing, data sharing, data + program sharing
  - Behavior of access patterns: static, dynamic
  - Level of knowledge on access pattern behavior: no information, partial information, complete information

- Distributed database design should be considered within this general framework.
Design Strategies/1

- Top-down approach
  - Designing systems from scratch
  - Homogeneous systems

- Bottom-up approach
  - The databases already exist at a number of sites
  - The databases should be connected to solve common tasks
Design Strategies/2

- Top-down design strategy
Distribution design is the central part of the design in DDBMSs (the other tasks are similar to traditional databases).

Objective: Design the LCSs by distributing the data over the sites.

Two main aspects have to be designed carefully:

- **Fragmentation**: Relation may be divided into a number of sub-relations, which are distributed
- **Allocation and replication**: Each fragment is stored at site(s) with "optimal" distribution

Distribution design issues

- Why fragment at all?
- How to fragment?
- How much to fragment?
- How to test correctness?
- How to allocate?
Design Strategies/4

- Bottom-up design strategy
What is a reasonable unit of distribution? Relation or fragment of relation?

Relations as unit of distribution:
- If the relation is not replicated, we get a high volume of remote data accesses.
- If the relation is replicated, we get unnecessary replications, which cause problems in executing updates and waste disk space.
- Might be an OK solution, if queries need all the data in the relation and data stays only at the sites that use the data.

Fragments of relation as unit of distribution:
- Application views are usually subsets of relations.
- Thus, locality of accesses of applications is defined on subsets of relations.
- Permits a number of transactions to execute concurrently, since they will access different portions of a relation.
- Parallel execution of a single query (intra-query concurrency).
- However, semantic data control (especially integrity enforcement) is more difficult.

⇒ Fragments of relations are (usually) appropriate unit of distribution.
Fragmentation aims to improve:
- Reliability
- Performance
- Balanced storage capacity and costs
- Communication costs
- Security

The following information is used to decide fragmentation:
- Quantitative information: cardinality of relations, frequency of queries, site where query is run, selectivity of the queries, etc.
- Qualitative information: predicates in queries, types of access of data, read/write, etc.
Types of Fragmentation

- Horizontal: partitions a relation along its tuples
- Vertical: partitions a relation along its attributes
- Mixed/hybrid: a combination of horizontal and vertical fragmentation

(a) Horizontal Fragmentation

(b) Vertical Fragmentation

(c) Mixed Fragmentation
## Example of database instance

<table>
<thead>
<tr>
<th>EMP</th>
<th>ASG</th>
<th>PROJ</th>
<th>PAY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENO</strong></td>
<td><strong>PNO</strong></td>
<td><strong>PNAME</strong></td>
<td><strong>BUDGET</strong></td>
</tr>
<tr>
<td>E1</td>
<td>P1</td>
<td>Instrumentation</td>
<td>150000</td>
</tr>
<tr>
<td>E2</td>
<td>P1</td>
<td>Database Develop.</td>
<td>135000</td>
</tr>
<tr>
<td>E3</td>
<td>P2</td>
<td>CAD/CAM</td>
<td>250000</td>
</tr>
<tr>
<td>E4</td>
<td>P2</td>
<td>Maintenance</td>
<td>310000</td>
</tr>
</tbody>
</table>
Example (contd.): Horizontal fragmentation of PROJ relation

- PROJ1: projects with budgets less than 200’000
- PROJ2: projects with budgets greater than or equal to 200’000

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>P2</td>
<td>Database Develop.</td>
<td>135000</td>
<td>New York</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>255000</td>
<td>New York</td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>
Example (contd.): Vertical fragmentation of PROJ relation

- PROJ1: information about project budgets
- PROJ2: information about project names and locations

<table>
<thead>
<tr>
<th>PNO</th>
<th>BUDGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>150000</td>
</tr>
<tr>
<td>P2</td>
<td>135000</td>
</tr>
<tr>
<td>P3</td>
<td>250000</td>
</tr>
<tr>
<td>P4</td>
<td>310000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrumentation</td>
<td>Montreal</td>
</tr>
<tr>
<td>P2</td>
<td>Database Develop.</td>
<td>New York</td>
</tr>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>New York</td>
</tr>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>Paris</td>
</tr>
</tbody>
</table>
Correctness Rules of Fragmentation

▶ Completeness
  ▶ Decomposition of relation $R$ into fragments $R_1, R_2, \ldots, R_n$ is complete iff each data item in $R$ can also be found in some $R_i$.

▶ Reconstruction
  ▶ If relation $R$ is decomposed into fragments $R_1, R_2, \ldots, R_n$, then there should exist some relational operator $\nabla$ that reconstructs $R$ from its fragments, i.e., $R = R_1 \nabla \ldots \nabla R_n$
    ▶ Union to combine horizontal fragments
    ▶ Join to combine vertical fragments

▶ Disjointness
  ▶ If relation $R$ is decomposed into fragments $R_1, R_2, \ldots, R_n$ and data item $d_i$ appears in fragment $R_j$, then $d_i$ should not appear in any other fragment $R_k$, $k \neq j$ (exception: primary key attribute for vertical fragmentation)
    ▶ For horizontal fragmentation, data item is a tuple
    ▶ For vertical fragmentation, data item is an attribute
Idea of Horizontal Fragmentation

- **Intuition** behind horizontal fragmentation
  - Every site should hold all information that is used to query the site
  - The information at the site should be fragmented so the queries of the site run faster
- Horizontal fragmentation is defined as selection operation, $\sigma_p(R)$
- **Example:**
  - $\sigma_{BUDGET < 200K}(PROJ)$
  - $\sigma_{BUDGET \geq 200K}(PROJ)$
Information Requirements/1

Database information:

- Links between relations (a link models a 1:N relationship between relations that are related to each other by an equality join)

Cardinality of relations: $\text{card}(R)$
Application information:

- Simple predicates used in queries
  - Relation $R[A_1, A_2, ..., A_n]$
  - $A_i \theta v_i$ a simple predicate ($\theta \in \{=, <, \leq, >, \geq, \neq\}$, $v_i \in D_i$, $D_i$ is the domain of $A_i$)
  - For relation $R$ we define $Pr = \{p_1, p_2, ..., p_m\}$
  - Example: PNAME = 'Maintenance', BUDGET < 200000

- Minterm predicates
  - minterm predicates are conjunctions of simple predicates
  - Relation $R$, $Pr = \{p_1, p_2, ..., p_m\}$
  - $M = \{m_1, m_2, ..., m_r\}$ such that
    $M = \{m_i \mid m_i = \bigwedge_{p_j \in Pr} p_j^{*}\}, 1 \leq j \leq m, 1 \leq i \leq z$
    where $p_j^{*} = p_j$ or $p_j^{*} = \neg p_j$
Application information (cnt’d):

- **Minterm selectivities**
  - The number of tuples of the relation that would be accessed by a user query, which is specified according to a given minterm predicate $m_i$.

- **Access frequencies**
  - The frequency with which a user application $q_i$ accesses data.
We write $Pr'$ to denote the complete and minimal set of simple predicates generated from $Pr$:

- The set of predicates is **complete** if and only if any two tuples in the same fragment are referenced with the same probability by any application.

- The set of predicates is **minimal** if and only if each simple predicate is relevant for determining the fragmentation and for each pair of fragments there is at least one query that accesses the fragments differently.
A **horizontal fragment** $R_i$ of relation $R$ consists of all the tuples of $R$ that satisfy a predicate $F_j$:

$$R_j = \sigma_{F_j}(R), \ 1 \leq j \leq w$$

where $F_j$ is a selection formula, which is (preferably) a minterm predicate.

**Computing** the horizontal fragmentation:

1. Determine Pr (80/20 rule)
2. Compute Pr’
3. Determine M
4. Minimize M (eliminating impossible minterms)
Example: Fragmentation of the \textit{PROJ} relation

Consider the following query: \textit{Find the name and budget of projects given their location.}

The query is issued at all three locations

Fragmentation based on LOC, using the set of predicates \{LOC = 'Montreal', LOC = 'NewYork', LOC = 'Paris'\}

\[ \textit{PROJ}_1 = \sigma_{\text{LOC} = 'Montreal'}(\textit{PROJ}) \]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrument.</td>
<td>150000</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

\[ \textit{PROJ}_2 = \sigma_{\text{LOC} = 'NewYork'}(\textit{PROJ}) \]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>DB Develop.</td>
<td>135000</td>
<td>New York</td>
</tr>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
</tbody>
</table>

\[ \textit{PROJ}_3 = \sigma_{\text{LOC} = 'Paris'}(\textit{PROJ}) \]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>
If access is only according to the location, the above set of predicates is complete
  ▶ i.e., in each fragment $\text{PROJ}_i$ each tuple has the same probability of being accessed

If there is a second query/application that accesses only those project tuples where the budget is less than $200K$, the set of predicates is not complete.
  ▶ $P_2$ in $\text{PROJ}_2$ has higher probability to be accessed
Example (contd.):

- Add \( BUDGET < 200K \) and \( BUDGET \geq 200K \) to the set of predicates to make it complete.

\[ \Rightarrow \{ \text{LOC} = 'Montreal', \text{LOC} = 'NewYork', \text{LOC} = 'Paris', \]
\[ BUDGET \geq 200K, BUDGET < 200K \} \]

is a complete set.

- Minterms to fragment the relation are given as follows:

\[
\begin{align*}
(\text{LOC} = 'Montreal') \land (BUDGET \leq 200K) \\
(\text{LOC} = 'Montreal') \land (BUDGET > 200K) \\
(\text{LOC} = 'NewYork') \land (BUDGET \leq 200K) \\
(\text{LOC} = 'NewYork') \land (BUDGET > 200K) \\
(\text{LOC} = 'Paris') \land (BUDGET \leq 200K) \\
(\text{LOC} = 'Paris') \land (BUDGET > 200K)
\end{align*}
\]
Example (contd.): Now, $PROJ_i, i = 1, 2, 3$ will be split in two fragments

\[
PROJ_1 = \sigma_{LOC='Montreal'}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Instrument.</td>
<td>150000</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

\[
PROJ_3 = \sigma_{LOC='NewYork'}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>DB Develop.</td>
<td>135000</td>
<td>New York</td>
</tr>
</tbody>
</table>

\[
PROJ_4 = \sigma_{LOC='NewYork'}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
</tbody>
</table>

\[
PROJ_6 = \sigma_{LOC='Paris'}(PROJ)
\]

<table>
<thead>
<tr>
<th>PNO</th>
<th>PNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>

Note that the following fragments are empty:

\[
\sigma_{LOC='Paris' \land BDGT <200K}(PROJ)
\]

\[
\sigma_{LOC='Montreal' \land BDGT \geq 200K}(PROJ)
\]
Defined on a member (target) relation of a link according to a selection operation specified on its owner (source).

- Each link is an equijoin.
- Fragments of member relation can be defined with a semijoin on fragments of owner relation.

![Diagram of derived horizontal fragmentation]

PAY is owner, EMP is member, PAY and PROJ lead to primary fragmentations, EMP and ASG lead to a derived fragmentation.
Given a link $L$ where $\text{owner}(L) = S$ and $\text{member}(L) = R$, the derived horizontal fragments of $R$ are defined as

$$R_i = R \bowtie S_i, 1 \leq i \leq w$$

where $w$ is the maximum number of fragments that will be defined on $R$ and

$$S_i = \sigma_{F_i}(S)$$

where $F_i$ is the formula according to which the primary horizontal fragment $S_i$ is defined.
Given link $L_1$ where
- \( owner(L_1) = PAY \) and
- \( member(L_1) = EMP \)

and
- \( PAY_1 = \sigma_{SAL \leq 30000}(PAY) \)
- \( PAY_2 = \sigma_{SAL > 30000}(PAY) \)

we get
- \( EMP_1 = EMP \Join PAY_1 \)
- \( EMP_2 = EMP \Join PAY_2 \)
Objective of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the applications will run on only one fragment.

Vertical fragmentation of a relation $R$ produces fragments $R_1, R_2, \ldots$, each of which contains a subset of $R$'s attributes.

Vertical fragmentation is defined using the projection operation of the relational algebra:

$$\pi_{A_1, A_2, \ldots, A_n}(R)$$

Example:

$$PROJ_1 = \pi_{PNO, BUDGET}(PROJ)$$
$$PROJ_2 = \pi_{PNO, PNAME, LOC}(PROJ)$$

Vertical fragmentation has also been studied for (centralized) DBMS

- Smaller relations, and hence less page accesses
- e.g., MONET system
Vertical fragmentation is **more complicated** than horizontal fragmentation

- In horizontal partitioning: for \( n \) simple predicates, the number of possible minterms is \( 2^n \); some of them can be ruled out by existing implications/constraints.
- In vertical partitioning: for \( m \) non-primary key attributes, the number of possible fragments is equal to \( B(m) \) (\( = \) the \( m \)th Bell number), i.e., the number of partitions of a set with \( m \) members.
  - For large numbers, \( B(m) \approx m^m \) (e.g., \( B(15) = 10^9 \))
Two types of heuristics for vertical fragmentation exist:

- **Grouping**: assign each attribute to one fragment, and at each step, join some of the fragments until some criteria is satisfied.
  - Bottom-up approach

- **Splitting**: starts with a relation and decides on beneficial partitionings based on the access behaviour of applications to the attributes.
  - Top-down approach
  - Results in non-overlapping fragments

- Optimal solution is probably closer to the full relation than to a set of small relations with only one attribute
Application information: The major information required as input for vertical fragmentation is related to applications (queries).

- Since vertical fragmentation places in one fragment those attributes usually accessed together, there is a need for some measure that would define more precisely the notion of “togetherness”, i.e., how closely related the attributes are.
- This information is obtained from queries and collected in the Attribute Usage Matrix and Attribute Affinity Matrix.
Given are the user queries/applications $Q = (q_1, \ldots, q_q)$ that will run on relation $R(A_1, \ldots, A_n)$.

**Attribute Usage Matrix:** Denotes which query uses which attribute:

$$use(q_i, A_j) = \begin{cases} 1 & \text{iff } q_i \text{ uses } A_j \\ 0 & \text{otherwise} \end{cases}$$

The $use(q_i, \bullet)$ vectors for each application are easy to define if the designer knows the applications that will run on the DB (consider also the 80-20 rule).
Example: Consider relation \( PROJ(PNO, PNAME, BUDGET, LOC) \) and queries:

\[
q_1 = \text{SELECT BUDGET FROM PROJ WHERE PNO=\text{Value}}
\]
\[
q_2 = \text{SELECT PNAME,BUDGET FROM PROJ}
\]
\[
q_3 = \text{SELECT PNAME FROM PROJ WHERE LOC=\text{Value}}
\]
\[
q_4 = \text{SELECT \text{SUM(BUDGET)} FROM PROJ WHERE LOC =\text{Value}}
\]

Abbreviations:
\[A_1 = PNO, A_2 = PNAME, A_3 = BUDGET, A_4 = LOC\]

Attribute Usage Matrix

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_2)</th>
<th>(A_3)</th>
<th>(A_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(q_2)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(q_3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(q_4)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Attribute Affinity Matrix: Denotes the frequency of two attributes $A_i$ and $A_j$ with respect to a set of queries $Q = (q_1, \ldots, q_n)$:

$$
aff(A_i, A_j) = \sum_k \left( \sum_{l: \text{use}(q_k, A_i) = 1, \text{use}(q_k, A_j) = 1} \text{ref}_l(q_k) \ast \text{acc}_l(q_k) \right)
$$

where

- $\text{ref}_l(q_k)$ is the cost (= number of accesses to $(A_i, A_j)$) for each execution of query $q_k$ at site $l$
- $\text{acc}_l(q_k)$ is the frequency of query $q_k$ at site $l$
Example (contd.): Let the cost of each query be $ref_i(q_k) = 1$, and the frequency $acc_i(q_k)$ of the queries be as follows:

<table>
<thead>
<tr>
<th>Site1</th>
<th>Site2</th>
<th>Site3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$acc_1(q_1) = 15$</td>
<td>$acc_2(q_1) = 20$</td>
<td>$acc_3(q_1) = 10$</td>
</tr>
<tr>
<td>$acc_1(q_2) = 5$</td>
<td>$acc_2(q_2) = 0$</td>
<td>$acc_3(q_2) = 0$</td>
</tr>
<tr>
<td>$acc_1(q_3) = 25$</td>
<td>$acc_2(q_3) = 25$</td>
<td>$acc_3(q_3) = 25$</td>
</tr>
<tr>
<td>$acc_1(q_4) = 3$</td>
<td>$acc_2(q_4) = 0$</td>
<td>$acc_3(q_4) = 0$</td>
</tr>
</tbody>
</table>

Attribute affinity matrix $\text{aff}(A_i, A_j) =$

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>45</td>
<td>0</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>$A_2$</td>
<td>0</td>
<td>80</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>$A_3$</td>
<td>45</td>
<td>5</td>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>$A_4$</td>
<td>0</td>
<td>75</td>
<td>3</td>
<td>78</td>
</tr>
</tbody>
</table>

e.g., $\text{aff}(A_1, A_3) = \sum_{k=1}^{1} \sum_{l=1}^{3} acc_i(q_k) = acc_1(q_1) + acc_2(q_1) + acc_3(q_1) = 45$

($q_1$ is the only query to access both $A_1$ and $A_3$)
Idea: Take the attribute affinity matrix (AA) and reorganize the attribute orders to form clusters where the attributes in each cluster demonstrate high affinity to one another.

**Bond energy algorithm (BEA)** has been suggested for that purpose for several reasons:

- It is designed specifically to determine groups of similar items as opposed to a linear ordering of the items.
- The final groupings are insensitive to the order in which items are presented.
- The computation time is reasonable ($O(n^2)$, where $n$ is the number of attributes)
BEA:

- Input: AA matrix
- Output: Clustered AA matrix (CA)
- Permutation is done in such a way to maximize the following **global affinity measure** (affinity of $A_i$ and $A_j$ with their neighbors):

\[
AM = \sum_{i=1}^{n} \sum_{j=1}^{n} \text{aff}(A_i, A_j)[\text{aff}(A_i, A_{j-1}) + \text{aff}(A_i, A_{j+1}) + \text{aff}(A_{i-1}, A_j) + \text{aff}(A_{i+1}, A_j)]
\]
Example (contd.): Cluster Affinity Matrix $CA$ after running BEA

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A_3$</td>
<td>45</td>
<td>53</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>$A_2$</td>
<td>0</td>
<td>5</td>
<td>80</td>
<td>75</td>
</tr>
<tr>
<td>$A_4$</td>
<td>0</td>
<td>3</td>
<td>75</td>
<td>78</td>
</tr>
</tbody>
</table>

- Elements with similar values are grouped together, and two clusters can be identified.
- An additional partitioning algorithm is needed to identify the clusters in $CA$.
  - Usually more clusters and more than one candidate partitioning, thus additional steps are needed to select the best clustering.
- The resulting fragmentation after partitioning ($PNO$ is added in $PROJ_2$ explicitly as key):

$$PROJ_1 = \{PNO, BUDGET\}$$
$$PROJ_2 = \{PNO, PNAME, LOC\}$$
Input: The AA matrix
Output: Clustered affinity matrix CA which is a permutation of AA

Initialization: Place and fix one of the columns of AA in CA (we choose column 1 in our example)
Iteration: Place the remaining n-i columns in the remaining i+1 positions in the CA matrix. For each column choose the placement that makes the largest contribution to the global affinity measure.
Row order: Order the rows according to the column ordering.
In order to determine the best placement we define the contribution of a placement.

- **Contribution of a placement:**
  
  \[
  \text{cont}(A_i, A_k, A_j) = 2 \cdot \text{bond}(A_i, A_k) + 2 \cdot \text{bond}(A_k, A_j) - 2 \cdot \text{bond}(A_i, A_j)
  \]

- **Bond between a pair of attributes is defined as:**

  \[
  \text{bond}(A_x, A_y) = \sum_{z=1}^{n} \text{aff}(A_z, A_x) \cdot \text{aff}(A_z, A_y)
  \]
Consider the following AA matrix and the corresponding CA matrix where $A_1$ and $A_2$ have been placed.

\[
AA = \begin{pmatrix}
A_1 & A_2 & A_3 & A_4 \\
A_1 & 45 & 0 & 45 & 0 \\
A_2 & 0 & 80 & 5 & 75 \\
A_3 & 45 & 5 & 53 & 3 \\
A_4 & 0 & 75 & 3 & 78 \\
\end{pmatrix}
\]

\[
CA = \begin{pmatrix}
A_1 & A_2 \\
A_1 & 45 & 0 \\
A_2 & 0 & 80 \\
A_3 & 45 & 5 \\
A_4 & 0 & 75 \\
\end{pmatrix}
\]

\[
bond(A_3, A_1) = 45 \times 45 + 5 \times 0 + 53 \times 45 + 3 \times 0 = 4410
\]

\[
bond(A_3, A_2) = 45 \times 0 + 5 \times 80 + 53 \times 5 + 3 \times 75 = 890
\]

The next step is to place $A_3$. 
BEA Algorithm/4

- Ordering (0-3-1):
  \( \text{cont}(A_0, A_3, A_1) \)
  \[
  = 2 \times \text{bond}(A_0, A_3) + 2 \times \text{bond}(A_3, A_1) - 2 \times \text{bond}(A_0, A_1)
  = 2 \times 0 + 2 \times 4410 - 2 \times 0 = 8820
  \]

- Ordering (1-3-2):
  \( \text{cont}(A_1, A_3, A_2) \)
  \[
  = 2 \times \text{bond}(A_1, A_3) + 2 \times \text{bond}(A_3, A_2) - 2 \times \text{bond}(A_1, A_2)
  = 2 \times 4410 + 2 \times 890 - 2 \times 225 = 10150
  \]

- Ordering (2-3-4):
  \( \text{cont}(A_2, A_3, A_4) \)
  \[
  = 2 \times \text{bond}(A_2, A_3) + 2 \times \text{bond}(A_3, A_4) - 2 \times \text{bond}(A_2, A_4)
  = 1780
  \]
After adding $A_3$ the CA matrix has the form

\[
CA = \begin{bmatrix}
A_1 & A_3 & A_2 \\
45 & 45 & 0 \\
0 & 5 & 80 \\
45 & 53 & 5 \\
0 & 3 & 75
\end{bmatrix}
\]

After adding $A_4$ and ordering rows the CA matrix has the form

\[
CA = \begin{bmatrix}
A_1 & A_3 & A_2 & A_4 \\
A_1 & 45 & 45 & 0 & 0 \\
A_3 & 45 & 53 & 5 & 3 \\
A_2 & 0 & 5 & 80 & 75 \\
A_4 & 0 & 3 & 75 & 78
\end{bmatrix}
\]
The final step is to divide a set of clustered attributes \( \{A_1, \ldots, A_n\} \) into two sets \( \{A_1, \ldots, A_i\} \) and \( \{A_{i+1}, \ldots, A_n\} \) such that the costs of queries that access both sets are minimized.

The appropriate split point on the diagonal must be determined:

![Diagram showing division of attributes]

<table>
<thead>
<tr>
<th></th>
<th>(A_1)</th>
<th>(A_3)</th>
<th>(A_2)</th>
<th>(A_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A1)</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(A3)</td>
<td>45</td>
<td>53</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>(A2)</td>
<td>0</td>
<td>5</td>
<td>80</td>
<td>75</td>
</tr>
<tr>
<td>(A4)</td>
<td>0</td>
<td>3</td>
<td>75</td>
<td>78</td>
</tr>
</tbody>
</table>

This is an optimization problem: the optimal point on the diagonal must be determined.

Note: Generalizations are needed to deal with

- partitions located in the middle of the matrix
- multiple split points
BEA Algorithm/7

- \( AQ(q_i) = \{ A_j \mid use(q_i, A_j) = 1 \} \) attributes accessed by \( q_i \)
- \( TQ = \{ q_i \mid AQ(q_i) \subseteq TA \} \) queries that access TA only
- \( BQ = \{ q_i \mid AQ(q_i) \subseteq BA \} \) queries that access BA only
- \( OQ = Q - \{ TQ \cup BQ \} \) queries that access TA and BA

\[-\]

- \( CQ = \sum_{q_i \in Q} \sum_{\forall S_j} ref_j(q_i) \ast acc_j(q_i) \) cost of all queries
- \( CTQ = \sum_{q_i \in TQ} \sum_{\forall S_j} ref_j(q_i) \ast acc_j(q_i) \) cost of TQ queries
- \( CBQ = \sum_{q_i \in BQ} \sum_{\forall S_j} ref_j(q_i) \ast acc_j(q_i) \) cost of BQ queries
- \( COQ = \sum_{q_i \in OQ} \sum_{\forall S_j} ref_j(q_i) \ast acc_j(q_i) \) cost of other queries
- \( CTQ \ast CBQ - COQ^2 \) maximize
Correctness of Vertical Fragmentation

- Relation $R$ is decomposed into fragments $R_1, R_2, \ldots, R_n$
  - e.g., $PROJ = \{PNO, BUDGET, PNAME, LOC\}$ into $PROJ_1 = \{PNO, BUDGET\}$ and $PROJ_2 = \{PNO, PNAME, LOC\}$

- Completeness
  - Guaranteed by the partitioning algorithm, which assigns each attribute in $A$ to one partition

- Reconstruction
  - Join to reconstruct vertical fragments
  - $R = R_1 \Join \cdots \Join R_n = PROJ_1 \Join PROJ_2$

- Disjointness
  - Attributes have to be disjoint in VF. Two cases are distinguished:
    - If tuple IDs are used, the fragments are really disjoint
    - Otherwise, key attributes are replicated automatically by the system
      - e.g., $PNO$ in the above example
Mixed Fragmentation

- In most cases simple horizontal or vertical fragmentation of a DB schema will not be sufficient to satisfy the requirements of the applications.

- **Mixed fragmentation (hybrid fragmentation):** Consists of a horizontal fragment followed by a vertical fragmentation, or a vertical fragmentation followed by a horizontal fragmentation.

- Fragmentation is defined using the selection and projection operations of relational algebra:

\[
\sigma_p(\pi_{A_1,\ldots,A_n}(R))
\]

\[
\pi_{A_1,\ldots,A_n}(\sigma_p(R))
\]
Replication and Allocation

- **Replication**: Which fragments shall be stored as multiple copies?
  - Complete Replication
    - Complete copy of the database is maintained in each site
  - Selective Replication
    - Selected fragments are replicated in some sites

- **Allocation**: On which sites to store the various fragments?
  - Centralized
    - Consists of a single DB and DBMS stored at one site with users distributed across the network
  - Partitioned
    - Database is partitioned into disjoint fragments, each fragment assigned to one site
Replication/1

- Replicated DB
  - **fully replicated**: each fragment at each site
  - **partially replicated**: each fragment at some of the sites

- Non-replicated DB (= partitioned DB)
  - **partitioned**: each fragment resides at only one site

- Rule of thumb:
  - If $\frac{\text{read only queries}}{\text{update queries}} \geq 1$, then replication is advantageous, otherwise replication may cause problems
Comparison of replication alternatives

<table>
<thead>
<tr>
<th></th>
<th>Full replication</th>
<th>Partial replication</th>
<th>Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY PROCESSING</td>
<td>Easy</td>
<td>Same difficulty</td>
<td></td>
</tr>
<tr>
<td>DIRECTORY MANAGEMENT</td>
<td>Easy or nonexistent</td>
<td>Same difficulty</td>
<td></td>
</tr>
<tr>
<td>CONCURRENCY CONTROL</td>
<td>Moderate</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
<tr>
<td>RELIABILITY</td>
<td>Very high</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>REALITY</td>
<td>Possible application</td>
<td>Realistic</td>
<td>Possible application</td>
</tr>
</tbody>
</table>
Fragment Allocation/1

- **Fragment allocation problem**
  - Given are:
    - fragments $F = \{F_1, F_2, \ldots, F_n\}$
    - network sites $S = \{S_1, S_2, \ldots, S_m\}$
    - and applications $Q = \{q_1, q_2, \ldots, q_l\}$
  - Find: the “optimal” distribution of $F$ to $S$

- **Optimality**
  - Minimal cost
    - Communication + storage + processing (read and update)
    - Cost in terms of time (usually)
  - Performance
    - Response time and/or throughput
  - Constraints
    - Per site constraints (storage and processing)
Fragment Allocation/2

- **Required information**
  - **Database Information**
    - selectivity of fragments
    - size of a fragment
  - **Application Information**
    - \( RR_{ij} \): number of read accesses of a query \( q_i \) to a fragment \( F_j \)
    - \( UR_{ij} \): number of update accesses of query \( q_i \) to a fragment \( F_j \)
    - \( u_{ij} \): a matrix indicating which queries updates which fragments,
    - \( r_{ij} \): a similar matrix for retrievals
    - originating site of each query
  - **Site Information**
    - \( USC_k \): unit cost of storing data at a site \( S_k \)
    - \( LPC_k \): cost of processing one unit of data at a site \( S_k \)
  - **Network Information**
    - communication cost/frame between two sites
    - frame size
We discuss an allocation model which attempts to minimize the total cost of processing and storage and meet certain response time restrictions.

General Form:

\[
\text{min}(\text{Total Cost})
\]

subject to

- response time constraint
- storage constraint
- processing constraint

Decision variable \(x_{ij}\)

\[
x_{ij} = \begin{cases} 
1 & \text{if fragment } F_i \text{ is stored at site } S_j \\ 
0 & \text{otherwise} \end{cases}
\]
The total cost function has two components: storage and query processing.

\[ TOC = \sum_{S_k \in S} \sum_{F_j \in F} STC_{jk} + \sum_{q_i \in Q} QPC_i \]

- **Storage cost** of fragment \( F_j \) at site \( S_k \):
  \[ STC_{jk} = USC_k \times size(F_j) \times x_{ij} \]
  where \( USC_k \) is the unit storage cost at site \( k \)

- **Query processing cost** for a query \( q_i \) is composed of two components:
  - composed of processing cost (PC) and transmission cost (TC)
    \[ QPC_i = PC_i + TC_i \]
Processing cost is a sum of three components:

- access cost (AC), integrity constraint cost (IE), concurrency control cost (CC)

\[ PC_i = AC_i + IE_i + CC_i \]

Access cost:

\[ AC_i = \sum_{s_k \in S} \sum_{F_j \in F} (UR_{ij} + RR_{ij}) \times x_{ij} \times LPC_k \]

where \( LPC_k \) is the unit process cost at site \( k \)

Integrity and concurrency costs:

- Can be similarly computed, though depends on the specific constraints

Note: \( AC_i \) assumes that processing a query involves decomposing it into a set of subqueries, each of which works on a fragment, ..., 
- This is a very simplistic model
- Does not take into consideration different query costs depending on the operator or different algorithms that are applied
The transmission cost is composed of two components:

- Cost of processing updates (TCU) and cost of processing retrievals (TCR)

\[ TC_i = TCU_i + TCR_i \]

**Cost of updates:**
- Inform all the sites that have replicas + a short confirmation message back

\[ TCU_i = \sum_{S_k \in S} \sum_{F_j \in F} u_{ij} \times (\text{update message cost} + \text{acknowledgment cost}) \]

**Retrieval cost:**
- Send retrieval request to all sites that have a copy of fragments that are needed + sending back the results from these sites to the originating site.

\[ TCR_i = \sum_{F_j \in F} \min_{S_k \in S} (x_{jk} \times (\text{cost retrieval request} + \text{cost sending back result})) \]
Modeling the constraints

- **Response time** constraint for a query \( q_i \)
  
  execution time of \( q_i \) \( \leq \) max. allowable response time for \( q_i \)

- **Storage** constraints for a site \( S_k \)
  
  \[ \sum_{F_j \in F} \text{storage requirement of } F_j \text{ at } S_k \leq \text{storage capacity of } S_k \]

- **Processing** constraints for a site \( S_k \)
  
  \[ \sum_{q_i \in Q} \text{processing load of } q_i \text{ at site } S_k \leq \text{processing capacity of } S_k \]
Solution Methods

- The complexity of this allocation model/problem is NP-complete.
- Correspondence between the allocation problem and similar problems in other areas:
  - Plant location problem in operations research
  - Knapsack problem
  - Network flow problem
- Hence, solutions from these areas can be re-used.
- Use different heuristics to reduce the search space:
  - Assume that all candidate partitionings have been determined together with their associated costs and benefits in terms of query processing.
  - The problem is then reduced to find the optimal partitioning and placement for each relation.
  - Ignore replication at the first step and find an optimal non-replicated solution.
  - Replication is then handled in a second step on top of the previous non-replicated solution.
Distributed design decides on the placement of (parts of the) data and programs across the sites of a computer network.

There are two key technical questions: fragmentation and allocation/replication of data.

Horizontal fragmentation is defined via the selection operation $\sigma_p(R)$:

- Rewrites the queries of each site in the conjunctive normal form and finds a minimal and complete set of conjunctions to determine fragmentation.

Vertical fragmentation via the projection operation $\pi_A(R)$:

- Computes the attribute affinity matrix and groups “similar” attributes together.

Mixed fragmentation is a combination of both approaches.
Conclusion/2

- Allocation/Replication of data
  - Type of replication: no replication, partial replication, full replication
  - Optimal allocation/replication modelled as a cost function under a set of constraints
  - The complexity of the problem is NP-complete
  - Use of different heuristics to reduce the complexity