Distributed DBMS architecture

• Covered topics
  – Transparencies in DDBMS
  – Architecture of DDBMS
  – Fragmentation, replication, allocation
    • Types and role of fragmentation
    • Types and role of replication
    • Allocation problem
Transparency

• In general: separation of higher-level semantics from lower-level implementation issues
• ‘Hide the implementation from users’
• Forms:
  – data independence
  – network transparency
  – replication transparency
  – fragmentation transparency
• Transparency can be provided at different levels of the system
Transparency

• Data independence
  – logical data independence: user applications are not affected by changes in the logical structure of the DB (schema definition)
  – physical data independence: hiding the details of the storage structure
  – the only form that is important in centralized DBMSs as well

• Network (distribution) transparency
  – location transparency: an operation on data is independent of both the location and the system where it is executed
  – naming transparency: unique name is provided for each object in the DB.
Transparency

• Replication transparency
  – Data is replicated for reliability and performance considerations.
  – The user should not be aware of the existence of copies

• Fragmentation transparency
  – DB relations are divided into smaller fragments for performance and availability reasons
  – The global queries should be translated to fragment queries
  – A question of query processing.
The origin of transparency

• Access layer: transparency features are built into user language which is translated into the requested operations (little transparency is provided from DBMS and OS)
  – typically: language transparency

• DBMS: acts as an integrated OS and DBMS, makes all the necessary translation between OS and user interface.
  – Essential when security and fault tolerance is critical and the OS cannot provide such services.
  – Typically: data, replication and fragmentation transparency

• OS: transparent access to resources provided by OS (e.g. device drivers, etc.) can be extended to distributed environments
  – typically: network transparency
Architecture

• The architecture of a system defines its structure
• In the foregoing parts the ‘architecture’ is a reference model
  – it is an idealized view
  – real world systems may be different
  – yet it shows the essential components and it is a reasonable framework
• Goal:
  – present the issues that need to be addressed at design
  – present a framework within which the design and implementation issues can be discussed
• Analogon: the ISO/OSI 7-layered reference model for computer networks
DBMS architecture

• A reference model can be described:
  – based on components. Components and their interrelations are defined.
  – Based on functions. Functions that the system will perform are defined.
  – Based on data. Different types of data and functional units (within an architectural framework), that will realize or use data, are defined

• Usually there is an interplay among them.
ANSI/SPARC architecture

• A centralized model from the 70s but generated interest and it is the basis of Distributed DBMSs
• Based on data organization
  – external view (user) - highest level, users’ view of a portion of the DB and relationships among data
  – conceptual view (enterprise) - abstract definition of the DB, the ‘real world’ view. Represents data without considering the application or the storage requirements
  – internal view (system) - lowest level, physical definition, storage issues
ANSI/SPARC architecture

• Data organizational view
Example of schemas

- Conceptual schema: the description of the modeled world (a pseudo code is shown here)
  
  ```
  - RELATION EMPLOYEE [
    KEY = {EMPLOYEE_NUMBER}
    ATTRIBUTES = {
      EMPLOYEE_NUMBER: CHARACTER(9)
      EMPLOYEE_NAME: CHARACTER(15)
      TITLE: CHARACTER(10) }
  ]
  
  - RELATION TITLE_SALARY [
    KEY = {TITLE}
    ATTRIBUTES = {
      TITLE: CHARACTER(10)
      SALARY: NUMERIC(6) }
  ]
  ```
Example of schemas

• Internal view: description of physical realization
  - `INTERNAL_REL EMP [
    INDEX ON E# CALL EMNIX
    FIELD = {
      E#: BYTE(9)
      E_NAME: BYTE(15)
      TIT: BYTE(10) }
  ]`

• External view: a portion of the database in the form that conforms the user’s needs (SQL-like notation shown here)
  - `CREATE VIEW PAYROLL (ENO, ENAME, SAL)
    AS SELECT EMPLOYEE.EMPLOYEE_NUMBER,
    EMPLOYEE.EMPLOYEE_NAME
    TITLE_SALARY.SALARY
    FROM EMPLOYEE, TITLE_SALARY
    WHERE EMPLOYEE.TITLE=TITLE_SALARY.TITLE`
Architectural models for Distributed DBMSs

- DBMS implementation alternatives

Distribution

Heterogeneity

Autonomy

Homogeneous multiple DBMSs (composite)

Distr. homogeneous DBMS

Distr. federated DBMS

Distr. multidatabase system

Distr. heterogeneous DBMS

Single site heterogeneous federated DBMS

Heterogeneous multidatabase system
Autonomy

• Distribution of control (and not data) - the degree of independence
  – The local operations of the individual DBMSs are not affected by their participation in the multidatabase system
  – The manner in which individual DBMSs process queries and optimize them should not be affected by the execution of global queries
  – System consistency should not be compromised when individual DBMSs join or leave the multidatabase system

[ Gilgor and Popescu-Zeletin’s definition ]

• Possibilities:
  – tight integration: a single-image of the entire DB is available
  – semiautonomous (federative) systems: operate independently but participates in a federation, i.e. part of the DB is sharable
  – total isolation
Distributed DBMSs (tight integration)

• From data organizational point of view
  – different physical data organization requires different local internal schemas
  – the DB is fragmented and replicated - each site represents a part of the modeled world - the conceptual schemas at each site are different
  – but: the users access the DB by different external schemas defined above the global conceptual schema
    • it means there is a low level of autonomy - individual DBs are in tight integration
Distributed DBMS architecture

- Data organizational view

- External Schema 1
- Local Conceptual Schema 1
- Local Internal Schema 1
- Global Conceptual Schema
- ES 2
- LCS 2
- LIS 2
- ES n
- LCS n
- LIS n

- Replication, fragmentation
- Heterogeneity
Transparency in DDBMS architecture

- Data independence: it is an extension of ANSI/SPARC that supports data independence
- Location and replication transparencies: definition of local and global conceptual schemas and mapping between them
- Network transparency: definition of global conceptual schema
Another view: component based architecture model for distributed DBMS
Components of the DDBMS architecture

• **User processor**
  – User interface: interprets commands and formats results
  – Semantic data controller: checks integrity constraints and authorizations (def. in GCS) to check if the query can be processed
  – Global query optimizer and decomposer: controls execution strategy, translates global queries into local ones (GCS and LCS)
  – Distributed execution monitor (distributed transaction manager): coordinates the distributed execution of the query (via possible communication with other monitors)
Components of the DDBMS architecture

• Data processor
  – Local query optimizer (access path selector): controls the way how data is accessed (access path means data structures + algorithms to access data, e.g. indexing)
  – Local recovery manager: ensures local consistency
  – Run-time support processor: physically accesses the data storage according to the commands, interfaces the OS
Multi-DBMS architecture

• Difference between distributed multi-DBMSs and distributed tightly integrated DBMSs: autonomy

• Fundamental difference in architecture: the global conceptual schema
  – in DDBMSs it represents the conceptual view of the entire database, while in MDBMSs it represents some of the local databases

• MDBMS
  – using global conceptual schema
  – without global conceptual schema
Multi-DBMS using global conceptual schema

Global External Schema

ES 2

ES n

Global Conceptual Schema

Local External Schema

LES

LES

LES

Local Conceptual Schema 1

Local Internal Schema 1

LCS n

LIS n
Multi-DBMS using global conceptual schema (semiautonomous, federated)

- GCS is defined by either
  - by integrating local external schemas or
  - by integrating parts of the local conceptual schemas

- Main difference between the GCSs
  - integrated DDBMS: mapping from global to local conceptual schemas
  - MDBMS: mapping from local to global conceptual schemas

- In the presence of heterogeneity
  - unilingual and
  - multilingual approaches are possible
Multi-DBMS without a global conceptual schema

- **External Schema 1**
- **Local Conceptual Schema 1**
- **Local Internal Schema 1**
- **Multidatabase layer**
- **Local system layer**

- **ES 2**
- **ES n**
- **LCS 2**
- **LCS n**
- **LIS 2**
- **LIS n**
Multi-DBMS without a global conceptual schema

- Some definitions claim the essence of multidatabase management is the lack of global schema
- Federated DB architectures sometimes do not use a global schema, instead
  - export schema for sharing data with others
  - import schema for accessing the global database (global external view)
Multi-DBMS

- Federation: the content (or parts of the content) of different DBs are put together logically
  - they may use different physical implementation (LIS)
  - they may use different data representation (LCS)
- The GCS, the Import and Export Schemas serve as a bridge between the different DBs
  - there is no way to generate these schemas automatically
  - they are created and tailored to each application individually
  - the models presented here are idealized and usually do not appear in this clean form in practice
Global directory

- The global directory is an extension of the ANSI/SPARC dictionary that describes the location and the properties of fragments.
- It is a meta-database that contains information about the database, e.g.
  - location of fragments,
  - content of fragments, etc.
Fragmentation, replication, allocation

• Difference between parallel and distributed DBs
  – A distributed DB is fragmented because data is fragmented by nature
    • geographically distributed sites of different architectures, systems, different concepts are put together logically
    • fragmentation is usually given and it is not a fundamental design issue
    • the location of DBs are also given, the allocation is addressed if replication is applied
  – A parallel DB can be fragmented to gain performance
    • fragmentation is a crucial design issue
    • allocation is also up to the designer
    • both have a fundamental impact on performance
    • a parallel DB is essentially a single DB and administrators have a greater freedom to tune the system
The role of fragmentation

• Fragmentation: decomposition of relations
  – it is not aimed at data distribution! Rather:
  – the relation is not a proper unit for distribution, a more appropriate unit can be obtained by partitioning
  – applications view only a subset of relations - locality can be optimized
  – unnecessary replication and high volume of remote memory accesses can be avoided
  – transactions can be executed concurrently: intraquery transaction

• but:
  – conflicting requirements may prevent decomposition into mutually exclusive fragments that can lead to performance degradation
  – semantic data control: checking for dependencies may be difficult
# Fragmentation alternatives

<table>
<thead>
<tr>
<th>JNO</th>
<th>JNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
</tr>
<tr>
<td>J2</td>
<td>Database</td>
<td>135000</td>
<td>New York</td>
</tr>
<tr>
<td>J3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
<tr>
<td>J4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>

## Horizontal (distr. + parallel)

<table>
<thead>
<tr>
<th>JNO</th>
<th>JNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>Instrumentation</td>
<td>150000</td>
<td>Montreal</td>
</tr>
<tr>
<td>J2</td>
<td>Database</td>
<td>135000</td>
<td>New York</td>
</tr>
</tbody>
</table>

## Vertical partitioning (parallel)

<table>
<thead>
<tr>
<th>JNO</th>
<th>JNAME</th>
<th>BUDGET</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>J3</td>
<td>CAD/CAM</td>
<td>250000</td>
<td>New York</td>
</tr>
<tr>
<td>J4</td>
<td>Maintenance</td>
<td>310000</td>
<td>Paris</td>
</tr>
</tbody>
</table>
Degree of fragmentation

• The extent to which the database is fragmented
  – not fragmented at all
  – fragmented to individual tuples (h) or attributes (v)

• How to decide?
  – The application should be characterized according to a number of parameters
  – Based on the result, the fragmentation can be designed
  – It is essential in parallel DB and less important in distributed DB
  – See chapter ‘Database Design’
Correctness rules of fragmentation

- (Vertical) fragmentation is similar to normalization process
- Rules to avoid semantic changes at fragmentation:
  - Completeness (lossless decomposition): no attributes or tuples may be eliminated at fragmentation
  - Reconstruction: there must be a relational operator ($\nabla$) that can reconstruct the original relation, i.e. $R = \nabla R_i, \forall R_i \in F_R$
  - Disjointness: horizontal fragments must be disjoint, i.e. if data item $d$ is in $R_j$, then it is not in any other $R_i$ ($i \neq j$)
Replication

• Fragments of relations are placed across the sites multiple times
  – increases reliability - if some sites fail, the data is still available
  – increases locality - the data can be retrieved from the closest or local site
  – increases performance - a certain fragment may be accessed by less users

• but the question
  – of mutual consistency
  – concurrency control
  – transparency must be addressed

• A DB can be
  – partitioned (no replication)
  – replicated
    • fully replicated - the whole DB is copied to each site
    • partially replicated
The problem of allocation

- After the DB has been partitioned, fragments must be allocated to various sites in the network or processing elements of a parallel machine - possibly multiple times when replicated.

- Given a set of fragments $F=\{F_1, F_2, \ldots, F_n\}$ and sites $S=\{S_1, S_2, \ldots, S_m\}$ on which a set of applications $Q=\{Q_1, Q_2, \ldots, Q_q\}$ is running:
  - Minimal cost. Minimize the cost of storing each $F_i$ at site $S_j$, querying $F_i$ at site $S_j$, updating $F_i$ at all sites where it is stored and the cost of communication.
  - Performance. Maintain a performance according to a metric:
    - response time
    - throughput
  - Even a very simple formulation of the optimization is proven to be NP-complete - heuristics for suboptimal solutions. For a general model see ‘Database Design’