Distributed architectures

- Data and computation are distributed over different machines
- Different levels of complexity
  - Depending on the independence level of nodes
- Typical advantages
  - Performance improvement
  - Increased availability
  - Stronger reliability
Distributed architectures

▷ Client/server
  • Simplest and more widespread
  • Server manages the database
  • Client manages the user interface

▷ Distributed database system
  • Different DBMS servers on different network nodes
    • autonomous
    • able to cooperate
  • Guaranteeing the ACID properties requires more complex techniques

Data replication

▷ A replica is a copy of the data stored on a different network node

▷ The replication server autonomously manages copy update

▷ Simpler architecture than distributed database
Distributed architectures

- Parallel architectures
  - Performance increase is the only objective
  - Different architectures
    - Multiprocessor machines
    - CPU clusters
      - Dedicated network connections

- Data warehouses
  - Servers specialized in *decision support*
  - Perform OLAP (On Line Analytical Processing)
    - different from OLTP (On Line Transaction Processing)

Relevant properties

- Portability
  - Capability of moving a program from a system to a different system
  - Guaranteed by the SQL standard

- Interoperability
  - Capability of different DBMS servers to cooperate on a given task
  - Interaction protocols are needed
    - ODBC
    - X-Open-DTP
Client/server Architectures

2-Tier
- **Thick** clients
  - with some application logic
- DBMS server
  - provides access to data
3-Tier

- **Thin client**
  - browser
- **Application server**
  - implements business logic
  - typically also a web server
- **DBMS Server**
  - provides access to data

### SQL execution

- **Compile & Go**
  - The query is sent to the server
  - The query is prepared
    - generation of the query plan
  - The query is executed
  - The result is shipped
    - The query plan is not stored on the server

- **Effective for one-shot query executions**
  - provides flexible execution of dynamic SQL
SQL execution

Compile & Store
- The query is sent to the server
- The query is prepared
  - generation of the query plan
  - the query plan is stored for future usage
- may continue with execution
  - the query is executed
  - the result is shipped

Efficient for repeated query executions
- parametric executions of the same query
Distributed database systems

- Client transactions access more than one DBMS server
  - Different complexity of available distributed services
- **Local autonomy**
  - Each DBMS server manages its local data in an autonomous way
    - e.g., concurrency control, recovery

Distributed database systems

- **Functional advantages**
  - Appropriate *localization* of data and applications
    - e.g., geographical distribution
- **Technological advantages**
  - Increased *data availability*
    - Total block probability is reduced
    - Local blocks may be more frequent
  - Enhanced *scalability*
    - Provided by the modularity and flexibility of the architecture
Given a relation R, a data fragment is a subset of R in terms of tuples, or schema, or both.

Different criteria to perform fragmentation:
- horizontal
  - subset of tuples
- vertical
  - subset of schema
- mixed
  - both horizontal and vertical together
The horizontal fragmentation of a relation $R$ selects a subset of tuples in $R$ with:
- same schema of $R$
- obtained by means of $\sigma_p$
  - $p$ is the partitioning predicate

Fragments are *not overlapped*.

**Example**

- The following table is given:
  - Employee ($\text{Emp}\#, \text{Ename}, \text{DeptName}, \text{Tax}$)
- Horizontal fragmentation on attribute DeptName:
  - $\text{card(DeptName)} = N$
  - $E_1 = \sigma_{\text{DeptName} = 'Production'} \text{Employee}$
  - ...
  - $E_N = \sigma_{\text{DeptName} = 'Marketing'} \text{Employee}$
- Reconstruction of the original table:
  - $\text{Employee} = E_1 \cup E_2 \cup \ldots \cup E_N$
Vertical fragmentation

The vertical fragmentation of a relation $R$ selects a subset of the schema of $R$

- Obtained by means of $\pi_X$
  - $X$ is a subset of the schema of $R$
  - The primary key should be included in $X$ to allow rebuilding $R$
- All tuples are included

Fragments are *overlapping* on the primary key

Example

The following table is given

$\text{Employee (Emp#, Ename, DeptName, Tax)}$

Vertical fragmentation

$E_1 = \pi_{\text{Emp#, Ename, DeptName}} \text{Employee}$

$E_2 = \pi_{\text{Emp#, Ename, Tax}} \text{Employee}$

Reconstruction of the original table

$\text{Employee} = E_1 \bowtie E_2$
**Fragmentation properties**

- **Completeness**
  - each information in relation R is contained in at least one fragment $R_i$

- **Correctness**
  - the information in R can be rebuild from its fragments

**Distributed database design**

- It is based on *data fragmentation*
  - Data distribution over different servers
- Each fragment of a relation R is usually stored
  - in a different file
  - possibly, on a different server
- Relation $R$ does not exist
  - it may be rebuilt from fragments
The allocation schema describes how fragments are stored on different server nodes.

- **Non redundant mapping if each fragment is stored on one single node**

  ![Diagram of non redundant mapping]

- **Redundant mapping if some fragments are replicated on different servers**
  - Increased data availability
  - Complex maintenance
    - Copy synchronization is needed

  ![Diagram of redundant mapping]
Transparency levels

- **Transparency levels** describe the knowledge of data distribution.
- An application should operate differently depending on the transparency level supported by the DBMS.
- Transparency levels:
  - fragmentation transparency
  - allocation transparency
  - language transparency

The following table is given:

<table>
<thead>
<tr>
<th>Horizontal fragment</th>
<th>Allocation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1 = \sigma_{\text{city} = \text{Torino}} S$</td>
<td>$<a href="mailto:S_1@xxx.torino.it">S_1@xxx.torino.it</a>$</td>
</tr>
</tbody>
</table>
| $S_2 = \sigma_{\text{city} = \text{Roma}} S$ | $S_2@xxx.roma1.it$  
|                     | $S_2@xxx.roma2.it$ |
Fragmentation transparency

Applications know the existence of tables and not of their fragments
- data distribution is not visible

Example
- The programmer only accesses table S not its fragments

```
SELECT SName
FROM S
WHERE S# = :CODE
```

Allocation transparency

Applications know the existence of fragments, but not their allocation
- not aware of replication of fragments
- must enumerate all fragments

Example

```
SELECT SName
FROM S_1
WHERE S# = :CODE
IF (NOT FOUND)
    SELECT SName
    FROM S_2
    WHERE S# = :CODE
```
Language transparency

- The programmer should select both the fragment and its allocation
  - No SQL dialects are used
- This is the format in which higher level queries are transformed by a distributed DBMS
- Example

```sql
SELECT SName
FROM S1@xxx.torino.it
WHERE S# = :CODE
IF (NOT FOUND)
  SELECT SName
  FROM S2@xxx.roma1.it
  WHERE S# = :CODE
```

Selection of a specific replica of S2

Database Management Systems

Transaction classification
**Transaction classification**

- The client requests the execution of a transaction to a given DBMS server
  - the DBMS server is in charge of redistributing it
- Classes define different complexity levels in the interaction among DBMS servers
  - They are based on the type of SQL instruction which the transaction is allowed to contain

**Transaction classification**

- Remote request
  - Read only request
    - only select statement
  - Single remote server
- Remote transaction
  - Any SQL command
  - Single remote server
Transaction classification

- Distributed transaction
  - Any SQL command
  - Each SQL statement is addressed to one single server
  - Global atomicity is needed
    - 2 phase commit protocol
- Distributed request
  - Each SQL command may refer to data on different servers
  - Distributed optimization is needed
  - Fragmentation transparency is in this class only

Example

- The following table is given
  - Account (Acc#, Name, Balance)
- Fragments and allocation schema

<table>
<thead>
<tr>
<th>Horizontal fragmentation</th>
<th>Allocation schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = \sigma_{\text{acc#} &lt; 10000} \text{Account}$</td>
<td>Node 1</td>
</tr>
<tr>
<td>$A_2 = \sigma_{\text{acc#} \geq 10000} \text{Account}$</td>
<td>Node 2</td>
</tr>
</tbody>
</table>
Money transfer transaction

BoT (Beginning of transaction)
UPDATE Account
SET Balance = Balance - 100
WHERE Acc# = 3000

UPDATE Account
SET Balance = Balance + 100
WHERE Acc# = 13000

EoT (End of transaction)

What is the class of the transaction?

- Distributed request because Account is not explicitly partitioned
- If instead the update instructions reference explicitly A₁ and A₂
  - Distributed transaction
- If both update instructions reference only A₁
  - e.g., second update with WHERE Acc#=9000
  - Remote transaction
ACID properties

▷ Atomicity
  - It requires distributed techniques
    - 2 phase commit

▷ Consistency
  - Constraints are currently enforced only locally

▷ Isolation
  - It requires strict 2PL and 2 Phase Commit

▷ Durability
  - It requires the extension of local procedures to manage atomicity in presence of failure
Other issues

- **Distributed query optimization** is performed by the DBMS receiving the query execution request
  - It partitions the query in subqueries, each addressed to a single DBMS
  - It selects the execution strategy
    - order of operations and execution technique
    - order of operations on different nodes
      - transmission cost may become relevant
      - (optionally) selection of the appropriate replica
  - It coordinates operations on different nodes and information exchange

Atomicity

- All nodes (i.e., DBMS servers) participating to a distributed transaction must implement the *same decision* (commit or rollback)
  - Coordinated by **2 phase commit** protocol
- Failure causes
  - Node failure
  - Network failure which causes lost messages
    - Acknowledgement of messages (ack)
    - Usage of timeout
  - Network partitioning in separate subnetworks
2 Phase Commit protocol

- **Objective**
  - Coordination of the conclusion of a distributed transaction
- **Parallel with a wedding**
  - Priest celebrating the wedding
    - Coordinates the agreement
  - Couple to be married
    - Participate to the agreement

- **Distributed transaction**
  - One coordinator
    - *Transaction Manager* (TM)
  - Several DBMS servers which take part to the transaction
    - *Resource Managers* (RM)
  - Any participant may take the role of TM
    - Also the client requesting the transaction execution
New log records

- TM and RM have *separate private* logs
- Records in the TM log
  - *Prepare*
    - it contains the identity of all RMs participating to the transaction (Node ID + Process ID)
  - *Global commit/abort*
    - final decision on the transaction outcome
  - *Complete*
    - written at the end of the protocol

New log records

- New records in the RM log
  - *Ready*
    - The RM is willing to perform commit of the transaction
    - The decision *cannot be changed* afterwards
    - The node has to be in a reliable state
      - WAL and commit precedence rules are enforced
      - Resources are locked
    - After this point the RM *loses its autonomy* for the current transaction
2 Phase Commit protocol

Phase I

1. The TM
   - Writes the prepare record in the log
   - Sends the prepare message to all RM (participants)
   - Sets a timeout, defining maximum waiting time for RM answer
2. The RMs

- Wait for the prepare message
- When they receive it
  - If they are in a reliable state
    - Write the ready record in the log
    - Send the ready message to the TM
  - If they are not in a reliable state
    - Send a not ready message to the TM
    - Terminate the protocol
    - Perform local rollback
  - If the RM crashed
    - No answer is sent
2 Phase Commit protocol

Phase I

3. The TM

- Collects all incoming messages from the RMs
- If it receives ready from all RMs
  - The commit global decision record is written in the log
- If it receives one or more not ready or the timeout expires
  - The abort global decision record is written in the log
Phase II

1. The TM
   - Sends the global decision to the RMs
   - Sets a timeout for the RM answers

2 Phase Commit protocol

[Diagram showing the 2 Phase Commit protocol with messages between TM and RM, including Ready, Commit/Abort, Prepare msg, Ready/not ready msg, Global decision, and Ack msg.]
2. The RM
- Waits for the global decision
- When it receives it
  - The commit/abort record is written in the log
  - The database is updated
  - An ACK message is sent to the TM
3. The TM
   - Collects the ACK messages from the RMs
   - If all ACK messages are received
     - The complete record is written in the log
   - If the timeout expires and some ACK messages are missing
     - A new timeout is set
     - The global decision is resent to the RMs which did not answer
   until all answers are received

2 Phase Commit protocol

- RM
  - Ready
  - Commit/Abort
  - uncertainty window
- TM
  - Prepare msg
  - Global decision
  - Ack msg
  - Complete

Phase I
- Phase II

Elena Baralis, Silvia Chiusano
Politecnico di Torino
Each RM is affected by an *uncertainty window*
- Start after ready msg is sent
- End upon receipt of global decision

Local resources in the RM are locked during the uncertainty window
- It should be small

The warm restart procedure is modified with a new case
- If the last record in the log for transaction T is "ready", then T does not know the global decision of its TM

**Recovery**
- READY list
  - new list collecting the IDs of all transactions in ready state
  - For all transactions in the ready list, the global decision is asked to the TM at restart
  - Remote recovery request
**Failure of the coordinator (TM)**

- Messages that can be lost
  - Prepare (outgoing)
  - Ready (incoming)
  - Global decision (outgoing)

- Recovery
  - If the last record in the TM log is prepare
    - The global abort decision is written in the log and sent to all participants
    - Alternative: redo phase I (not implemented)
  - If the last record in the TM log is the global decision
    - Repeat phase II

---

**Network failures**

- Any network problem in phase I causes global abort
  - The prepare or the ready msg are not received
- Any network problem in phase II causes the repetition of phase II
  - The global decision or the ACK are not received
Protocol for the coordination of distributed transactions

- It guarantees interoperability of distributed transactions on *heterogeneous* DBMSs
  - i.e., different DBMS products

- Based on
  - One client
  - One TM
  - Several RMs
Interfaces

- X-Open-DTP defines interfaces for the communication
  - between client and TM
    - TM interface
  - between TM and RM
    - XA interface

- DBMS servers provide the XA interface
- Specialized products implement the TM and provide the TM interface
  - E.g., BEA tuxedo

Standard features

- RMs are passive and only answer to remote procedure invocations from the TM
- The control of the protocol is embedded in the TM
- The protocol implements two optimizations of 2 Phase Commit
  - Presumed abort
  - Read only
- Heuristic decision to allow controlled transaction evolution in presence of failures
**Presumed abort**

- The TM, when no information is available in the log, answers abort to a remote recovery request by a RM
  - Reduces the number of synchronous log writes
    - prepare, global abort, complete are not synchronous
  - Synchronous writes are still needed
    - global commit in TM log
    - ready, commit in RM log

**Read only**

- Exploited by a RM that did not modify its database during the transaction
- The RM
  - answers read only to the prepare request
  - does not write the log
  - locally terminates the protocol
- The TM will ignore the RM in phase II of the protocol
Heuristic decision

- Allows transaction evolution in presence of TM failures
  - During the uncertainty window, a RM may be blocked because of a TM failure
    - Locked resources are blocked until TM recovery
  - The blocked transaction evolves locally under operator control
    - Transaction end is forced by the operator
      - Typically rollback, rarely commit
        - Heuristic decision, because actual transaction outcome is not known
      - Blocked resources are released

During TM recovery, decisions are compared to the actual TM decisions
- If TM decision and RM heuristic decision are different, atomicity is lost
- The protocol guarantees that the inconsistency is notified to the client process
- Resolving inconsistencies caused by a heuristic decision is up to user applications
Parallelism

- Parallel computation increases DBMS efficiency
- Queries can be effectively parallelized
  - **Examples**
    - large table scan performed in parallel on different portions of data
    - data is fragmented on different disks
    - group by on a large dataset
    - partitioned on different processors
    - group by result merged
  - **Different technological solutions are available**
    - Multiprocessor systems
    - Computer clusters

Inter-query parallelism

- Different queries are scheduled on different processors
- Used in OLTP systems
- Appropriate for workloads characterized by
  - simple, short transactions
  - high transaction load
    - 100-1000 tps
- Load balancing on the pool of available processing units
Intra-query parallelism

- Subparts of the same query are executed on different processors
- Used in OLAP systems
- Appropriate for workloads characterized by:
  - complex queries
  - reduced query load
- Complex queries are partitioned in subqueries:
  - each subquery performs one or more operations on a subset of data
    - group by and join are easily parallelizable
    - pipelining operations is possible

DBMS benchmarks
DBMS benchmarks

- Benchmarks describe the conditions in which performance is measured for a system
- DBMS benchmarks are standardized by the TPC (Transaction Processing Council)
- Each benchmark is characterized by
  - Transaction load
    - distribution of arrival time of transactions
  - Database size and content
    - randomized data generation
  - Transaction code
  - Techniques to measure and certify performance

Types of benchmarks

- TPC C
  - Order entry transactions
  - It simulates the behavior of an OLTP system
  - New evolution is TPC E
- TPC H
  - Decision support (OLAP)
  - It is a mix of complex queries and some updates
- TPC App
  - Transactions on the web
  - Simulation of an e-commerce site