XII. Distributed Architectures

Types of Distributed Architectures
Client-Server Architectures
Distributed Databases
Data Fragmentation and Allocation
Transparency Levels

Paradigms for Data Distribution

- **Client-server architecture**: separation of the database server from the client
- **Distributed databases**: several database servers used by the same application
- **Parallel databases**: several data storage devices and processors operate in parallel for increasing performances
- **Replicated databases**: data logically representing the same information and physically stored on different servers
- **Data warehouses**: servers specialized for the management of data dedicated to decision support.

Technologies for Distributed Systems

- Networks -- local area and wide area networks (LAN and WAN respectively); you need to install network cables.
- Internet -- use telephone lines to transfer data and programs, thereby facilitating distribution.
- Wireless communication technologies -- the way of the future
  - ✓ Packet radio technology; generally works for short distances, low transfer rates;
  - ✓ Cellular networks, support cell phones;
  - ✓ Commercial satellite systems, such as iridium
- Telephone companies world-wide are scrambling to get out of wired telephony and into wireless services.

How Much Distributed Data is there?

- It is estimated that we produce about 1 hexabyte of unique web data per year; that's 10^{18}Bytes, or ~250MB per person on earth; no disk can hold so much data.
- We produce 500 times more information through email messages than through the web.
- The great challenge (some say the greatest ever!) of technology is to search, retrieve, organize and manage all these data, and make them useful to people.
- Someone (Brewster Kahle, an archivist) is trying to archive the web…

…a lot!!!

Separation of Functionalities

- OLTP (*On-Line Transaction Processin*g) systems, aims at optimized management and reliable transactions on *database servers*, specialized for supporting hundreds or even thousands of transactions per second
- OLAP (*On-Line Analytic Processin*g) systems, aimed at data analysis, which operate on *data warehouse* servers, specialized for data management for decision support

Desirable Properties for Highly Interactive Systems

- **Portability** refers to the ease of transporting programs from one environment to another (and it is thus a typical *compile-time* property)

Facilitated by language standards (e.g., SQL-2, SQL-3)

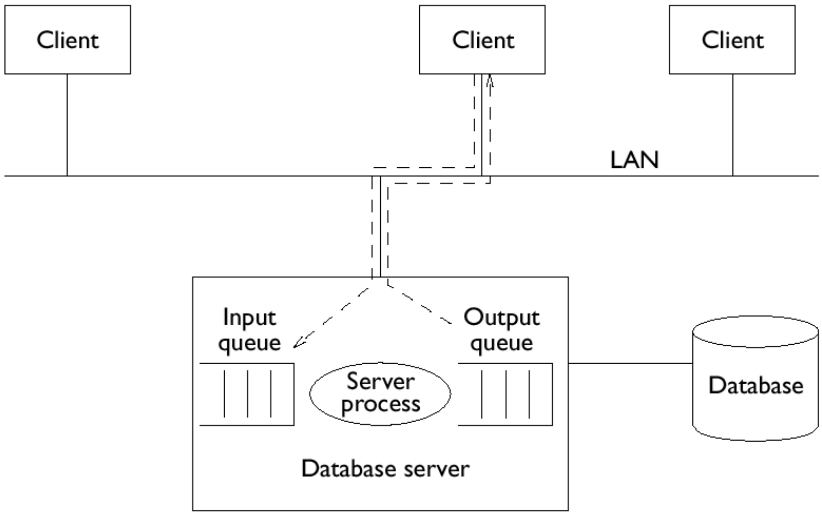
- **Interoperability** refers to a system's ability to interact with other, possibly heterogeneous, systems (and it is thus a typical *execution-time* property.)

Facilitated by standard data access protocols, including Database Connectivity (ODBC) and X-Open Distributed Transaction Processing (DTP)

Client-Server Architectures

- **Client-server** is a general model of interaction between software processes, where interacting processes are classified into ***clients*** (which require services) and ***servers*** (which offer services)
- Requires a precise definition of a ***service interfac***e, which lists the services offered by the server
- The client process performs an active role, the server process is reactive
- Normally, a client process requests few services in sequence from one or more server processes, while a process server responds to multiple requests from many process clients.

Typical Client-Server Architecture



Client Server Architecture for Data Management

- It is a good idea to assign clients and servers to distinct computers because:
  - ✓ The functions of clients and servers are clearly identified
  - ✓ They give rise to a convenient separation of application and management activities
- SQL offers an ideal programming paradigm for the identification of the 'service interface'
  - ✓ SQL queries are sent from the client to the server
  - ✓ The query results are returned to the client
  - ✓ The standardization, portability and interoperability of SQL allows the construction of client applications that involve different server systems

Allocation of Clients and Servers to Different Computers

- The computer dedicated to the client must be suitable for interaction with the user and support productivity tools (electronic mail, word processing, spreadsheet, Internet access, and workflow management)
- The server computer must have a large main memory (to support buffer management) and a high capacity disk (for storing the entire database)

Multi-Threaded Architectures

- Often, database (and other) servers are ***multi-threaded***:
  - ✓ Behave like a single process that works dynamically on behalf of several transactions;
  - ✓ Each unit of execution of the server process for a given transaction is called a ***thread;***
- Servers are permanently active processes that control an ***input queue*** for client requests and an ***output queue*** for the query results
- Often, a ***dispatcher*** process distributes requests to the servers and returns the responses to the clients
- When the dispatchers can dynamically define the number of active server processes as a function of the number of requests received: we say that a ***server class*** is available

Two-Tier vs Three-Tier Architectures

- **Two-tier architectures**: the client is both the user interface and the application manager
  - ✓ The client is called ***thick-client***, as it supports the application logic
- **Three-tier architectures**: a second server is present, known as the ***application server***, responsible for the management of the application logic common to many clients
  - ✓ The client is named ***thin-client***; it is responsible only for the interface with the final user; such clients can be deployed using browser technology

Distributed Databases

- A ***distributed database*** is a system in which some clients interact with multiple servers for the execution of an application
- We discuss separately:
  - ✓ How a user can specify distributed queries
  - ✓ How the server technology is extended in a distributed database

Advantages of Distributed Databases

- Distributed databases respond to application needs. After all, enterprises are geographically distributed; distributed databases allow the distribution of data processing and control to the environment where data is generated and largely used
- Distributed databases offer greater flexibility, modularity and resistance to failures
  - ✓ Distributed systems can be configured by the progressive addition and modification of components;
  - ✓ Although they are more vulnerable to failures, they support ‘graceful degradation’ (respond to failures with a reduction in performance but without total failure)

Classification of Applications

- Based on the type of DBMS involved:
  - ✓ Homogeneous Distributed Databases: When all the servers have the same DBMS
  - ✓ Heterogeneous Distributed Databases: When the servers support different DBMSs
- Based on the network:
  - ✓ Can use a ***local area network*** (LAN)
  - ✓ Can use a ***wide area network*** (WAN)

Classification of Applications

Type of DBMS	Network type: LAN	Network type: WAN
Homogeneous	Data management and financial applications	Travel management and financial applications
Heterogeneous	Inter-divisional information systems	Integrated banking and inter-banking systems

Local Independence and Cooperation

- In a distributed database, each server has its own capacity to manage applications independently:
  - ✓ A distributed database should not maximize the interaction and the necessity of transmitting data via networks
  - ✓ On the contrary, the planning of data distribution and allocation should be done in such a way that applications should operate independently on a single server

Data Fragmentation and Allocation

- Given a relation *R*, its fragmentation consists of determining fragments R_i by applying algebraic operations to *R*.
  - ✓ In ***horizontal fragmentation***, fragments R_i are groups of tuples having the same schema as *R* (as after selection on *R*); Horizontal fragments are usually disjoint;
  - ✓ In ***vertical fragmentation***, each fragment R_i has a subset of the schema of *R* (as though projection was applied on *R*). Each vertical fragment includes the primary key of *R*
- The fragmentation is correct if it is:
  - ✓ ***Complete***: each data item of *R* must be present in one of its fragments R_i
  - ✓ ***Restorable***: the content of *R* must be restorable from its fragments

Example

- Consider EMPLOYEE (Emp#, Name, Dept#, Salary, Taxes)
- Horizontal fragmentation
  - ✓ EMPLOYEE1 = $\sigma_{Emp\#<=3}$ EMPLOYEE
  - ✓ EMPLOYEE2 = $\sigma_{Emp\#>3}$ EMPLOYEE
- Reconstruction requires a union:
  - ✓ EMPLOYEE = EMPLOYEE1 U EMPLOYEE2
- Vertical fragmentation:
  - ✓ EMPLOYEE1 = $\Pi_{Emp\#,Name}$EMPLOYEE
  - ✓ EMPLOYEE2 = $\Pi_{Emp\#,DeptName,Salary,Tax}$EMPLOYEE
- Reconstruction requires an equi-join on key values (natural join).
  - ✓ EMPLOYEE = EMPLOYEE1 ⊳⊲ EMPLOYEE2

Initial Table

EMPLOYEE

EmpNum	Name	DeptName	Salary	Tax
1	Robert	Production	3.7	1.2
2	Greg	Administration	3.5	1.1
3	Anne	Production	5.3	2.1
4	Charles	Marketing	3.5	1.1
5	Alfred	Administration	3.7	1.2
6	Paolo	Planning	8.3	3.5
7	George	Marketing	4.2	1.4

Example of Horizontal Fragmentation

EMPLOYEE1

EmpNum	Name	DeptName	Salary	Tax
1	Robert	Production	3.7	1.2
2	Greg	Administration	3.5	1.1
3	Anne	Production	5.3	2.1

EMPLOYEE2

EmpNum	Name	DeptName	Salary	Tax
4	Charles	Marketing	3.5	1.1
5	Alfred	Administration	3.7	1.2
6	Paolo	Planning	8.3	3.5
7	George	Marketing	4.2	1.4

Vertical Fragmentation

EMPLOYEE1

EmpNum	Name
1	Robert
2	Greg
3	Anne
4	Charles
5	Alfred
6	Paolo
7	George

EMPLOYEE2

EmpNum	DeptName	Salary	Tax
1	Production	3.7	1.2
2	Administration	3.5	1.1
3	Production	5.3	2.1
4	Marketing	3.5	1.1
5	Administration	3.7	1.2
6	Planning	8.3	3.5
7	Marketing	4.2	1.4

Fragmentation and Allocation Schemata

- Each fragment R_i corresponds to a different physical file and is allocated to a different server.
- Thus, the relation is present in a virtual mode (like a view), while fragments are actually stored.
- The *allocation schema* describes the mapping of relations or fragments to the servers that store them. This mapping can be:
  - ✓ ***non-redundant***, when each fragment or relation is allocated to a single server
  - ✓ ***redundant***, when at least one fragment or relation is allocated to more than one server

Transparency Levels

- There are three significant levels: transparency of fragmentation, allocation and language
- In the ***absence of transparency***, each DBMS accepts its own SQL 'dialect': the system is heterogeneous and the DBMSs do not support a common interoperability standard
- Given SUPPLIER(S#,Name,City), with horizontal fragments
  - ✓ SUPPLIER1 = $\sigma_{City='London'}$ SUPPLIER
  - ✓ SUPPLIER2 = $\sigma_{City='Manchester'}$ SUPPLIER
- …and the allocation schema:
  - ✓ SUPPLIER1@company.London.uk
  - ✓ SUPPLIER2@company.Manchester1.uk
  - ✓ SUPPLIER2@company.Manchester2.uk

Fragmentation Transparency

- At this level, the programmer should not worry about whether or not the database is distributed or fragmented
- Query:

```
procedure Query1(:snum,:name);
select Name into :name
  from Supplier
  where SNum = :snum;
end procedure
```


Allocation Transparency

- The programmer should know the structure of the fragments, but does not have to indicate their allocation
- With replication, the programmer does not have to indicate which copy is chosen for access (*replication transparency*)
- Query:

```
procedure Query2(:snum,:name);
select Name into :name from Supplier1
  where SNum = :snum;
if :empty then
  select Name into :name
    from Supplier2 where SNum = :snum;
end procedure;
```

Language Transparency

- The programmer indicates both the structure of the fragments and their allocation
- Queries expressed at a higher level of transparency are transformed to this level by the distributed query optimizer.
- Query:

```
procedure Query3(:snum,:name);
select Name into :name
  from Supplier1@company.London.uk
  where SNum = :snum;
if :empty then select Name into :name
    from Supplier2@company.Manchester1.uk
    where SNum = :snum;
end procedure;
```


Optimizations

This application can be made more efficient in two ways:

- By using *parallelism*: instead of submitting the two requests in sequence, they can be processed in parallel, thus saving on the global response time
- By using *knowledge on the logical properties of fragments* (but then the programs are not flexible)

```
procedure Query4(:snum,:name,:city);
case :city of
"London": select Name into :name
   from Supplier1 where SNum = :snum;
"Manchester": select Name into :name
   from Supplier2 where SNum = :snum;
end procedure;
```

Classification of Transactions

- ***Remote requests****:* read-only transactions made up of an arbitrary number of SQL queries, addressed to a single remote DBMS (remote DBMS can only be queried)
- ***Remote transactions*** made up of any number of SQL commands (select, insert, delete, update) directed to a single remote DBMS (each transaction writes on one DBMS.)
- ***Distributed transactions*** made up of any number of SQL commands (select, insert, delete, update) directed to an arbitrary number of remote DBMSs, but each SQL command refers to a single DBMS (Transactions may update more than one DBMS, require the two-phase commit protocol)
- ***Distributed requests*** are arbitrary transactions, in which each SQL command can refer to any DBMS; assumes a distributed optimizer