Distributed Architectures and DBMS

1 Multi User DBMS Architectures

- So far we have seen this model of interaction between end users and the database
  - One central DBMS
  - Users interact with DBMS using an application program
- Several issues still need clarification:
  - Is the DBMS on the end user's computer
  - How are the users connected to the DBMS?
  - Do we have one or more DBMS?
  - Which computations are performed where?
  - Is the database stored in one or many places
- Teleprocessing Architecture:
  - The traditional (and most basic) architecture
  - One computer with a single CPU
  - Many (end-user) terminals all cabled to the central computer
  - The terminal sends messages to the central computer
  - All data processing in the central computer
  - This puts tremendous burden on the central computer leading to decreased performance
- Nowadays, the trend is towards downsizing
  - Replace expensive mainframe computers with cost-effective networks of personal computers
  - Achieve the same/better

2 File-Server architecture

Processing is distributed around a computer network

- Typically through a LAN
- One central file-server
- Every workstation has its own DBMS and its own user application
- Workstations request files they need from the file server
- File server acts like a "shared hard disk" (it has no DBMS)

```sql
SELECT fName, IName
FROM Branch b, Staff s
WHERE b.branchNo=s.branchNo AND b.street='163 Main St.'
```

- The file-server has no knowledge of SQL - the user's DBMS has to request the whole tables Branch, Staff
- Therefore:
  - Very large amount of network traffic (the tables may be huge)
  - A full copy of the DBMS required on each workstation
  - Concurrency/recovery/integrity control is more difficult since multiple DBMSs access the same files simultaneously
- The solution to these problems is a client-server architecture

3 Client-Server architecture

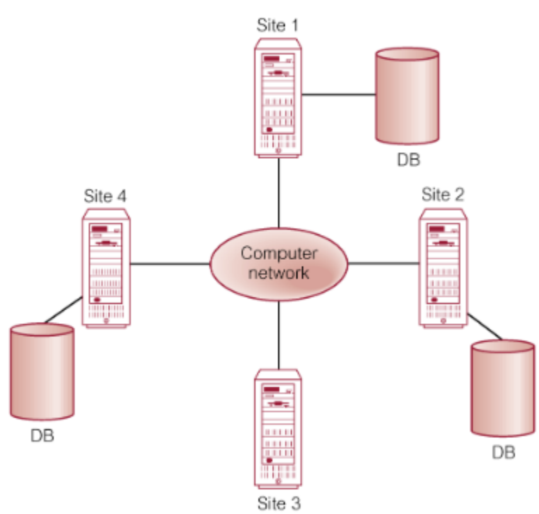
- Client - Requires some resource
- Server - provides the resource
- Client/server are not always in the same machine/place
- Two-tier architecture
  - Tier 1 (client): responsible for the presentation of data to the user
  - Tier 2 (server): responsible for supplying data services to the user
- Typical Procedure:
  - User gives a request to the client
  - Client generates SQL query and sends it to the server
  - Server accepts, processes the query and sends the result to the client
  - Client formats the result for the user
- Many advantages:
  - Increased performance: many client CPUs
  - Reduced Hardware Costs: only the server needs increased storage and computational power
  - Reduced communication costs: less data traffic (not unnecessary are transmitted)
- Database is still centralized - not a distributed database

4 Three-Tier Client-Server Arch

- In modern system: 100s/1000s of users - need for increased enterprise
- Main problem of the client that prevent scalability - a ”fat client” (many users) requires extensive resources on disk space/RAM/CPU power
- A new variant on the client server architecture
  - Three layers, potentially running on different platforms
  - First Tier: UI later (on end user's computer)
  - Second Tier: application server (connects to many users)
  - Third Tier: database server (contains DBMS, communicates with the application server)
  - ”Thin clients” - increased performance of user's computer
- Best example for a client: internet browser
- Advantages:
  - Smaller hardware cost for ”thin clients”
  - Easier application maintenance (centralized in one tier)
  - Easier to modify/replace one tier without affecting others
  - Easier load balancing between the different tiers
  - Maps naturally to web applications
- It can be extended
  - Separation of tasks into n intermediate tiers for increased flexibility and scalability

5 Distributed DBMS

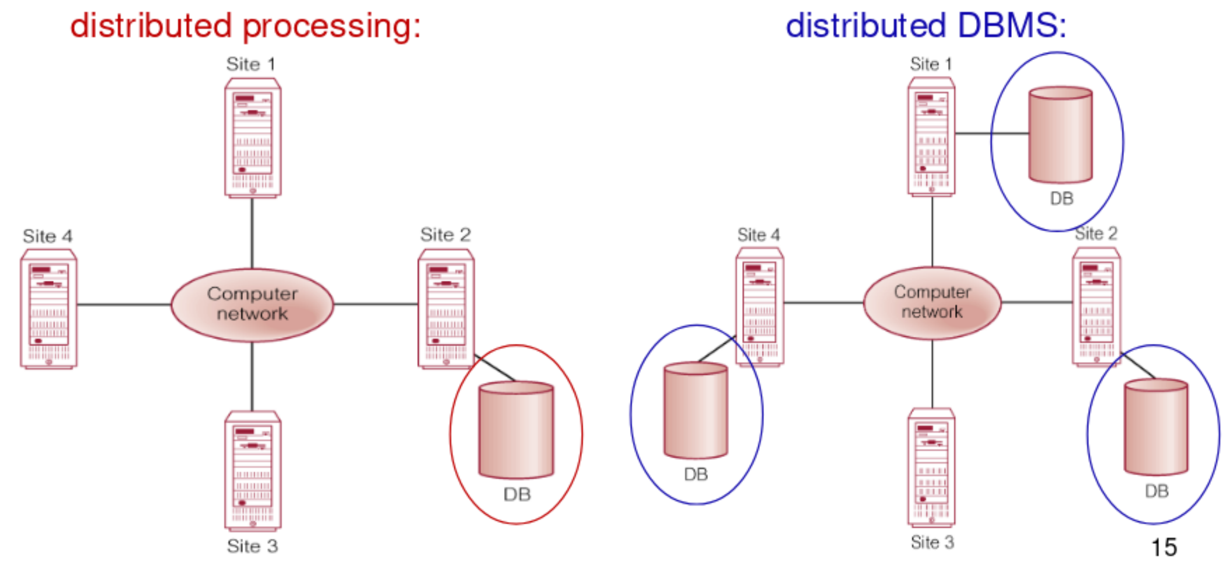
- So far we have seen centralized database systems
  - single database, located at one site
  - Controlled by one DBMS
- We can improve database performance:
  - Using networks of computers (decentralized approach)
  - It mirrors the organizational structure:
    * Logically distributed into divisions, departments, projects...
    * Physically distributed into offices, flats, units, factories...
- Main targets
  - Make all data accessible to all units
  - Store the data proximate to the location where it is most frequently used
  - Full functionality and efficiency
- Distributed Database
  - A logically interrelated collection of shared data physically distributed over a network
- Distributed DBMS (DDBMS)
  - The software system that can manage the distributed database
  - It makes the distribution transparent (invisible) to users
- In a DDBMS
  - A single logical database, which is split into fragments
  - Each fragment is stored on one (or more) computers, under the control of a separate DBMS
  - All of these computers are connected by a communications network
  - Sites have local autonomy: independent processing of local data (via local applications)
  - Sites have access to global applications (to process data fragments stored on other computers)
- Not all sites have local applications/local data
- All sites have access to global applications
- Data fragments may be replicated in more sites (data consistency must be considered)



6 Distributed processing vs Distributed DBMS

Distributed processing:

- A centralized database that is accessed over a computer network
- For example: the client server architecture
- This is not the same as a distributed DBMS



7 Design of a distributed DBMS

In addition to ER modelling, we have to consider also:

- Fragmentation:
  - How to break a relation into fragments
  - Fragments can be horizontal/vertical/mixed
- Allocation:
  - How fragments are allocated at the several sites
  - Aim is to reach an ”optimal” distribution (efficient, reliable,...)
- Replication
  - Which fragments are stored in multiple sites (and which sites)

Choices for Fragmentation and Allocation:

- Based on how the database is to be used
- Quantitative and Qualitative information is used
- Quantitative information (mainly for fragmentation)
  - The frequency with which specific transactions are run
  - The (usual) sites from which transactions are run
  - Desired performance criteria for the transactions
- Qualitative information (mainly for allocation):
  - The relations/attributes/tuples being accessed
  - The type of access (read/write)

- Strategic objectives for the choices about the fragments
  - Locality of reference
    - Data to be stored close to where it is used
    - If a fragment is used at several sites then replication is useful
  - Reliability and availability
    - Improved by replication
    - If one site fails, there are other fragment copies available

Further strategic objectives

- Acceptable performance
  - Bad allocation results in "bottleneck" effects (a site receives too many requests so has bad performance)
  - Also: Bad allocation caused underutilized resources
- Cost of storage capacities
  - Cheap mass storage to be used at sites, whenever possible
  - This must be balanced against locality of reference
- Minimal communication costs
  - Minimum retrieval costs when max locality of reference, or when each site has its own copy of data
  - But when replicated data is updated
    - All copies of this data must be updated
    - Increased network traffic/communication costs

Four alternative strategies for the placement of data

- Centralized
  - Single database and DBMS
  - Stored at one site with users distributed across the network
  - Not distributed
- Partitioned
  - Database partitioned into disjoint fragments
  - Each data item assigned to exactly one site (no replication)
- Complete replication
  - Complete copy of the database at each site
- Selective replication
  - Combination of partitioning, replication and centralization

Balance of the strategic objectives

	Locality of reference	Reliability and availability	Performance	Storage costs	Communication costs
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Fragmented	High[a]	Low for item; high for system	Satisfactory[a]	Lowest	Low[a]
Complete replication	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High[a]	Low for item; high for system	Satisfactory[a]	Average	Low[a]

[a] Indicates subject to good design.

Three correctness rules for the partitioned placement

1. Completeness
   - If relation R is decomposed into fragments $R_1, R_2, ...R_n$ each data item in R must appear in at least one fragment R_i
2. Reconstruction
   - It must be possible to define a relational algebra expression that can reconstruct R from its fragments
3. Disjointness
   - If a data item appears in fragment R_i, it should not appear in any other fragment
   - Exception for vertical fragmentation: primary key attributed must be repeated for the reconstruction

8 Fragmentation

Three main types of fragmentation

1. Horizontal
   - A subset of the tuples of the relation
2. Vertical
   - A subset of the attributes of the relation
3. Mixed
   - A vertical fragment that is then horizontally fragmented
   - Or a horizontal fragment that is then vertically fragmented

8.1 Horizontal fragmentation

- Assume there exist two property types: 'Flat' and 'House'
- We have a relation R with all properties for rent
- The horizontal fragmentation of R (by property type) is

$$P_1 = \sigma_{type='House'}(PropertyForRent)$$

$$P_2 = \sigma_{type='Flat'}(PropertyForRent)$$

- This fragmentation may be useful e.g. if we have separate applications dealing with flats/houses
- And it is correct
  - **Completeness**: Each tuple is in either P_1 or in P_2
  - **Reconstruction**: R can be constructed from the fragments P_1, P_2

$$R = P_1 \cup P_2$$

  - **Disjointness**: There is no property that is both 'flat' and 'house'

8.2 Vertical Fragmentation

- For every staff member in a company
  - The payroll department requires: `staffNo, position, sex, salary`
  - The personnel department requires: `staffNO, fName, DOB, branchNo`
- We have a relation Staff with all staff members
- For this example, the vertical fragmentation of staff is:

$$S_1 = \Pi_{\text{staffNo, position, sex, salary}}(Staff)$$

$$S_2 = \Pi_{\text{staffNo, Name, DOB, branchNo}}(Staff)$$

- Both fragments include the primary key staffNo to allow reconstruction of Staff from S_1 and S_2
- This fragmentation is useful
  - The fragments are stored at the departments that are needed
  - Performance for every department is improved (as the fragment is smaller than the original relation Staff)
- This fragmentation is correct
  - **Completeness**
    * The primary key `staffNo` belongs to both S_1 and S_2
    * Each other attribute is either in S_1 or in S_2
  - **Reconstruction**
    * R can be constructed from the fragments S_1, S_2 using the natural join operation
$$Staff = S_1 \bowtie S_2$$
  - **Disjointness**
    * The fragments are disjoint except the primary key (which is necessary for the reconstruction)

9 Advantages and Disadvantages of a distributed DBMS

Advantages

- Reflects organizational structure
- Improves share-ability and local autonomy
- Improved availability and reliability
- Improved performance
- Smaller hardware cost
- Scalability

Disadvantages

- Complexity
- Higher maintenance cost
- Security
- Integrity control more difficult
- Design more complex
- Lack of experience in the industry