Database System Architectures:

Database system architecture refers to the overall structure and design of a database system. It defines how data is stored, organized, accessed, and managed. Choosing the right architecture is crucial for efficient and effective data management.

- **What it is:** The overall design and methodology for organizing a database system. It defines how data is stored, managed, and accessed.
- **Importance:** Choosing the right architecture impacts efficiency, scalability, and maintainability of the database.

Types of Database Architectures:

1. Tier Architecture:

- **1-Tier (Single-tier):** Easiest to develop, but least scalable (think basic calculator app).
- **2-Tier:** Separates user interface from data storage (more scalable than single-tier, but server can become overloaded).
- **3-Tier:** Most scalable and flexible, with a dedicated layer for handling complex business logic (common in web applications).

2. Data Model:

- **Relational Model:** Data is stored in tables with rows and columns. Relationships between tables are established through foreign keys. (Example: MySQL, Oracle)
- **NoSQL Model:** More flexible than relational models, suitable for large unstructured datasets.

Centralized Architecture

- **Definition:** All data and processing logic reside on a single server. Clients (usually dumb terminals) connect and interact directly with the server.
- **Pros:**
  - Simple to set up and manage.
  - Cost-effective for small databases.
- **Cons:**
  - Limited scalability: Performance bottlenecks as data and user load increase.
  - Single point of failure: Server outage affects all users.
- **Example:** Early database systems on mainframes.

Client-Server Architecture

- **Definition:** Data is stored on a central server, but clients (PCs, workstations) handle user interface and some processing. Clients communicate with the server to access and manipulate data.
- **Pros:**
  - More scalable: Additional servers can be added to handle increased load.
  - Improved performance: Distributes processing tasks between clients and server.
  - More flexible: Supports diverse client applications and functionalities.
- **Cons:**
  - More complex to set up and manage compared to centralized architecture.
  - Requires robust network infrastructure.
- **Example:** Most modern database systems.

Server System Architecture

A server system architecture in a Database Management System (DBMS) refers to the organization of processes and data storage on the server side. There are two main categories:

- **Transaction Servers:** Used in relational databases, these handle queries and transactions efficiently.
- **Data Servers:** Used in object-oriented databases, these focus on data access and processing on powerful client machines.

1. Transaction Server Process Structure:

Definition: A transaction server consists of multiple processes that manage user requests (transactions), data access, and overall system functionality.

Components:

- **Server Processes:** Receive user queries (transactions), execute them, and return results. These can be multi-threaded for concurrent execution.
- **Lock Manager Process:** Ensures data consistency by granting and releasing locks on accessed data during transactions. It also detects deadlocks (conflicting transactions).
- **Database Writer Process:** Writes modified data buffers back to disk periodically.
- **Log Writer Process:** Maintains a transaction log for recovery purposes, recording all database modifications.

- **Checkpoint Process:** Periodically creates a consistent snapshot of the database for faster recovery in case of failure.
- **Process Monitor Process:** Monitors server processes for errors and restarts them if necessary.
- **Shared Memory:** Stores frequently accessed data for faster retrieval by server processes. This includes a buffer pool for temporary data storage.

Example: A typical SQL server implements a transaction server architecture.

2. Data Servers

Definition: Data servers prioritize data access and processing on client machines. They are often used in object-oriented databases.

Key Points:

- Clients have more processing power and handle complex queries locally.
- Data servers can ship data (pages or entire objects) to clients for processing.
- Data caching on clients improves performance for frequently accessed data.

Example: An object-oriented database system with client-side processing capabilities can be considered a data server architecture.

Key Differences:

- **Focus:** Transaction servers focus on processing user requests and ensuring data consistency, while data servers focus on physical storage and access.
- **Location:** Transaction servers typically reside on the same machine as the client applications or on a dedicated server machine. Data servers can be located on the same machine or on separate storage machines for scalability.

Parallel Systems

Concept: Parallel DBMS utilizes multiple processors and disks to significantly improve database performance.

Goal: Achieve faster query execution, data loading, and other database operations compared to traditional single-processor systems.

Benefits:

- **Increased Speed:** By splitting tasks across multiple processors, queries and operations run concurrently, leading to faster results.
- **Improved Scalability:** Adding more processors and disks allows the system to handle larger datasets and more users efficiently.
- **Enhanced Availability:** If one node fails, others can continue processing, minimizing downtime and improving system resilience.

Architectural Designs:

- **Shared Memory Architecture:** Processors share a global memory space for efficient data access (e.g., Symmetric Multiprocessing - SMP systems).
    - **Example:** A multi-core server running a database management system.
- **Shared Disk Architecture:** Processors have private memories but access shared storage devices for data.
    - **Example:** A cluster of computers with shared disk storage connected through a high-speed network.
- **Shared-Nothing Architecture:** Each node has its own CPU, memory, and storage, communicating via messages (more complex to manage but highly scalable).
    - **Example:** Large database systems distributed across multiple geographically dispersed servers.

Overall, parallel systems in DBMS offer significant performance improvements and scalability for handling large and complex databases.

Speed up parallel systems

- Focuses on **reducing execution time** for a fixed workload.
- Achieved by adding more processing resources (CPUs, machines) to a system.
- Ideally, speedup is linear with the number of resources added. (e.g., 2x CPUs = 2x faster processing)
- **Example:** A complex query takes 10 seconds on a single CPU. Adding 3 more CPUs and parallelizing the query execution might reduce the time to 2.5 seconds (4x speedup).

Scale up parallel systems

- Focuses on handling an **increasing workload** while maintaining performance.
- Achieved by adding resources proportionally to the workload growth.
- Ensures the system can accommodate larger datasets and more users without performance degradation.

- **Example:** A database server struggles with 1000 users. Scaling up by adding more processors and storage allows it to handle 2000 users with similar response times.

Key Points:

- Speedup is not always linear due to communication overhead between processors.
- Not all queries can be perfectly parallelized. Some tasks may have inherent sequential steps.
- Scaleup requires careful planning to ensure balanced growth of resources and workload.

In essence:

- **Speedup** is about getting things done faster with the same amount of work.
- **Scaleup** is about handling more work without sacrificing speed.

Interconnection Networks

Interconnection networks define how processors or storage devices communicate in a Database Management System (DBMS) with parallel processing capabilities. Here's a look at three common types:

Bus:

- Simple shared channel for all devices.
- Only one device transmits at a time (bottleneck for high traffic).
    - Example: Multiple servers accessing a shared storage unit.

Mesh:

- Processors arranged in a grid, connected to nearest neighbors.
- Offers multiple paths for data transfer, improving scalability.
    - Example: Clustered DBMS where nodes communicate directly with nearby nodes for data distribution.

Hypercube:

- Highly scalable network based on n-dimensional cubes.
- Processors connected only to nodes differing in a single dimension (efficient for specific tasks).
    - Example: Parallel processing intensive database queries where data is distributed across nodes based on specific criteria.

Parallel Database Architectures

Parallel databases distribute tasks across multiple processors or computers to improve performance. Here's a breakdown of common architectures:

1. Shared Memory Architecture:

- Tightly coupled system with all processors sharing a single memory space.
- Data and code reside in the shared memory, accessible by all processors.
- **Advantages:** Fast communication, efficient for small to medium databases.
- **Disadvantages:** Scalability limited by memory capacity, complex cache coherency management.
- **Example:** A high-end server with multiple CPUs.

2. Shared Disk Architecture:

- Multiple processors connected to a shared storage device (disk).
- Data resides on the shared disk, while processors have local memory.
- **Advantages:** Simpler to implement than shared-nothing, good for read-mostly workloads.
- **Disadvantages:** Bottleneck at the shared disk, limited scalability due to disk I/O overhead.
- **Example:** A cluster of computers with a central storage array.

3. Shared-Nothing Architecture:

- Most scalable architecture, each node (computer) has its own CPU, memory, and local storage.
- Data is partitioned across nodes, requiring coordination for queries accessing data from multiple locations.
- **Advantages:** Highly scalable, good for large databases and complex workloads.
- **Disadvantages:** Increased complexity in managing data distribution and query execution.
- **Example:** A large warehouse with data spread across independent servers.

4. Hierarchical Architecture:

- Combines elements of shared memory and shared-nothing architectures.
- Upper level acts as a coordinator, distributing tasks to lower-level nodes.
- Lower levels can be shared-memory or shared-nothing themselves.
- **Advantages:** Flexibility to handle diverse workloads, good scalability.
- **Disadvantages:** Increased complexity in managing different levels.
- **Example:** A cluster of servers with a central coordinator node.