Advanced DB

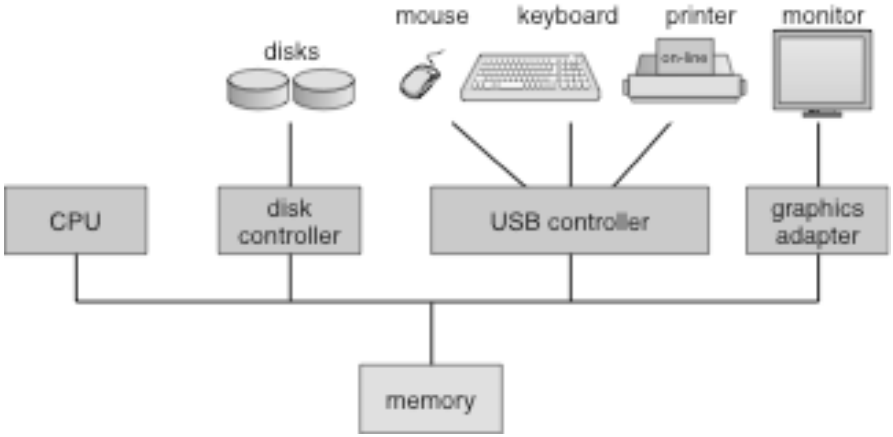
CHAPTER 20 DATABASE-SYSTEM ARCHITECTURES

Chapter 20: Database System Architectures

- Centralized Systems & Client--Server Systems
- Server System Architectures
- Parallel Systems
- Distributed Systems
- Network Types

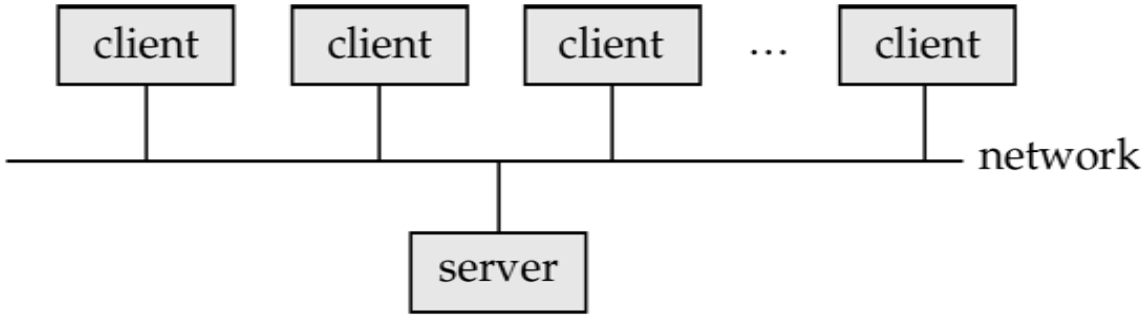
Centralized Systems

- Run on a single computer system
  - and do not interact with other computer systems.
- General-purpose computer system
  - Single-user system
    - e.g., personal computer or workstation
  - Multi-user system
    - Serve a large number of users who are connected to the system via terminals
    - Often called *server* systems.



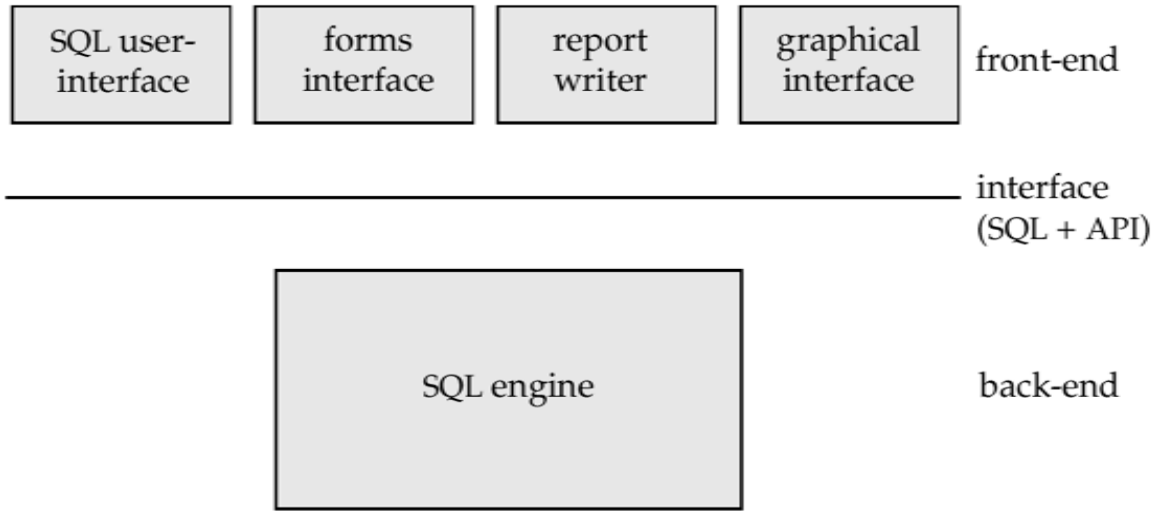
Client-Server Systems

- Server systems satisfy requests generated at multiple client systems



Client-Server Systems (Cont.)

- Database functionality can be divided into:
  - **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - **Front-end**: consists of tools such as *forms*, *report-writers*, and graphical user interface facilities.
  - The interface between the front-end and the back-end is through SQL or through an application program interface (API)



Client-Server Systems (Cont.)

- Advantages of replacing mainframes with networks of workstations or personal computers connected to back-end server machines:
  - better functionality for the cost
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance
- Server systems can be broadly categorized into two kinds:
  - **transaction servers:** widely used in relational database systems
  - **data servers:** used in object-oriented database systems

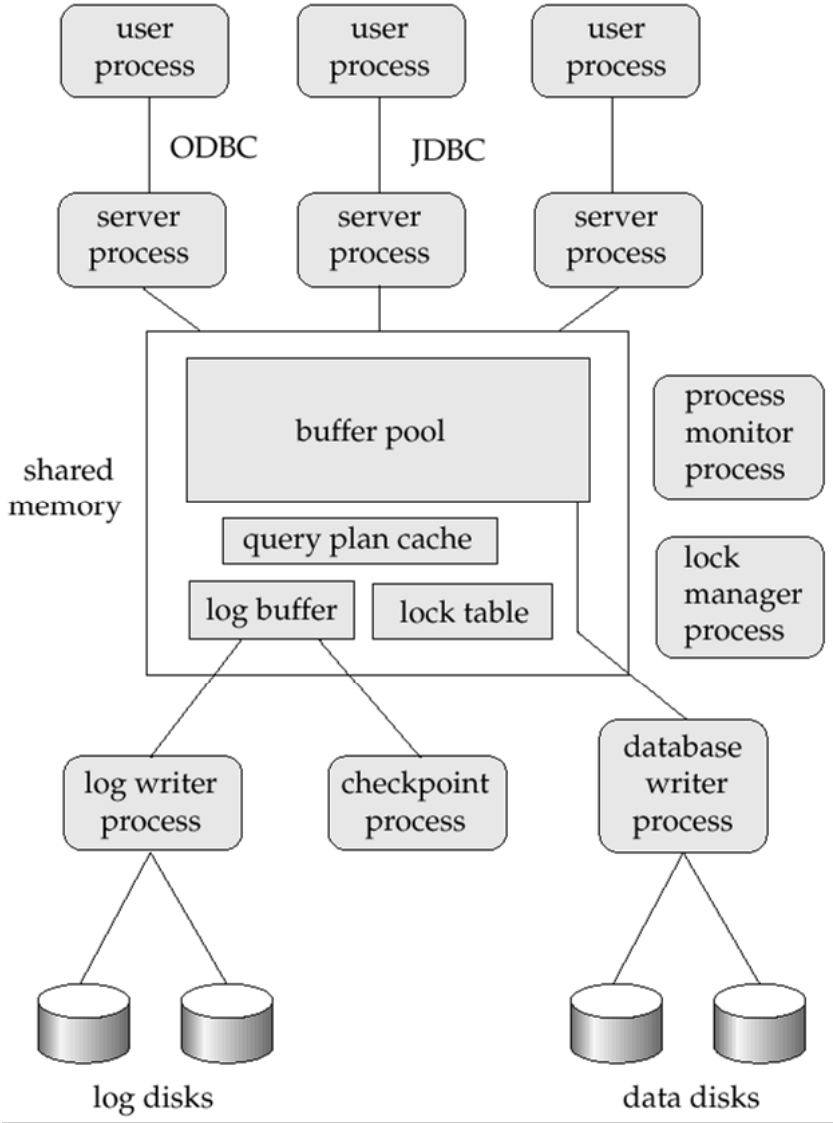
Transaction Servers

- Also called query server systems or SQL *server* systems
  - clients send requests to the server system
  - transactions are executed at server
  - results are shipped back to the client
- Requests specified in SQL, and communicated to the server through a *remote procedure call (RPC)* mechanism.
- *Open Database Connectivity (ODBC)*
  - C language API standard for connecting to a server
  - Send SQL requests and receive results.
- JDBC standard, similar to ODBC, for Java

Transaction Server Structure

- Shared memory contains shared data
  - Buffer pool, Lock table, Log buffer
  - Cached query plans (reused if same query submitted again)
- Processes
  - *Server processes*
    - receive user queries (transactions), execute them and send results back
    - Processes may be *multithreaded*, allowing a single process to execute several user queries concurrently (typically multiple multithreaded server processes)
  - *Writer processes*
    - Database writer process, log writer process, checkpoint process
  - *Lock manager process*
    - Manages locks in shared memory
  - *Process monitor process*
    - Monitors other processes, and takes recovery actions if any of the other processes fail

Transaction Server Structure (Cont.)



Data Servers

- Ship data to client machines
  - where processing is performed,
  - and then ship results back to the server machine.
- Requires full back-end functionality at the clients.
  - the tasks to be executed are compute intensive
  - used in many object-oriented database systems
- Issues:
  - Page-Shipping vs Item-Shipping
  - Locking
  - Data Caching & Lock Caching

Data Servers (Cont.)

- Page-Shipping vs Item-Shipping
  - Smaller unit of shipping ⇒ more messages
  - Worth *prefetching* related items along with requested item
  - Page shipping can be thought of as a form of prefetching
- Locking
  - Overhead of requesting and getting locks from server is high due to message delays
  - Can grant locks on requested and prefetched items (with page shipping, transaction is granted lock on whole page)
  - Locks on a prefetched item can be *called back* by the server, and returned by client transaction if the prefetched item has not been used.

Data Servers (Cont.)

- Data Caching
  - Data can be cached at client even in between transactions
  - Must check that data is up-to-date before it is used (*cache coherency*)
- Lock Caching
  - Locks can be retained by client system even in between transactions
  - Transactions can acquire cached locks locally, without contacting server
  - Server *calls back* locks from clients when it receives conflicting lock request. Client returns lock once no local transaction is using it.

Parallel Systems

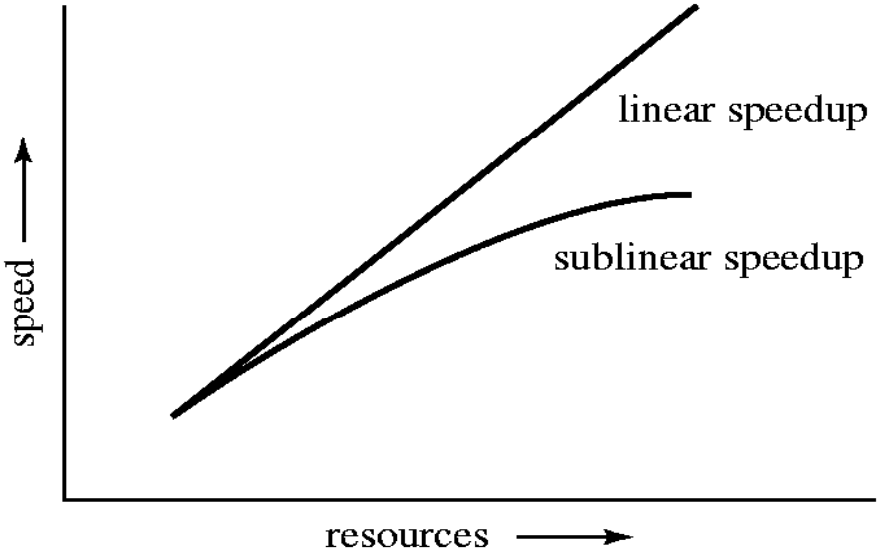
- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- *Coarse-grain* parallel machine
  - consists of a small number of powerful processors
- *Massively parallel* or *Fine grain* parallel machine
  - utilizes thousands of smaller processors
- Two main performance measures:
  - **response time** --- the amount of time it takes to complete a single task from the time it is submitted
  - **throughput** --- the number of tasks that can be completed in a given time interval

Speedup

- Measure of *response time gain* from system size increase
- Fixed-sized problem is given to a system N-times larger.

$$speedup = \frac{small\ system\ elapsed\ time}{large\ system\ elapsed\ time}$$

- Speedup is *linear* if equation equals N.

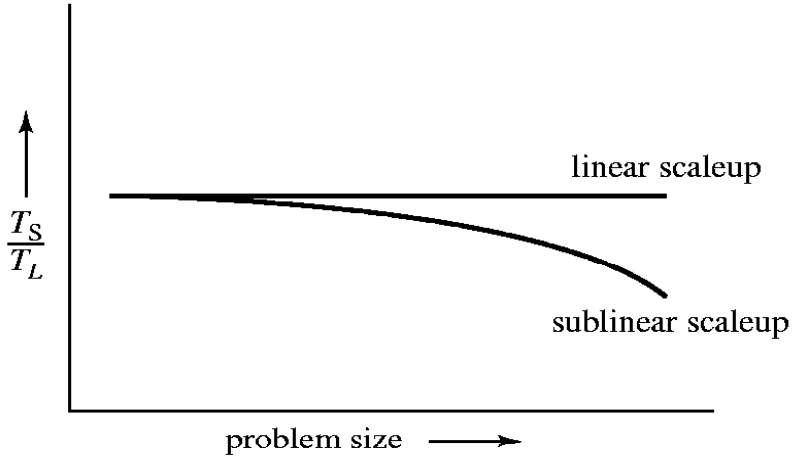


Scaleup

- Measure of *throughput gain* from system size increase
- Increase the size of both the problem and the system
  - *N*-times larger system used to perform *N*-times larger job

$$scaleup = \frac{small\ system\ small\ problem\ elapsed\ time}{big\ system\ big\ problem\ elapsed\ time}$$

- Scale up is *linear* if equation equals 1.



(resources increase proportional to problem size)

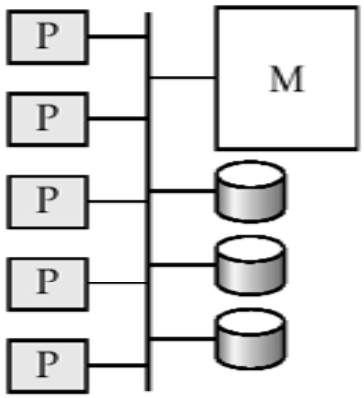
Batch and Transaction Scaleup

- Batch scaleup:
  - A single large job
  - Typical of most database queries and scientific simulation.
  - Use an *N*-times larger computer on *N*-times larger problem.
- Transaction scaleup:
  - Numerous small queries submitted by independent users to a shared database
  - Typical transaction processing and timesharing systems.
  - *N*-times as many users submitting requests (hence, *N*-times as many requests) to an *N*-times larger database, on an *N*-times larger computer.
  - Well-suited to parallel execution.

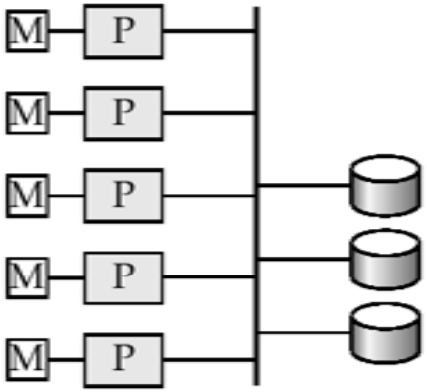
Factors Limiting Speedup and Scaleup

- Speedup and scaleup are often sublinear
- Reasons:
  - **Startup costs**: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high.
  - **Interference**: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other
  - **Skew**: Overall execution time determined by *slowest* of parallely executing tasks.

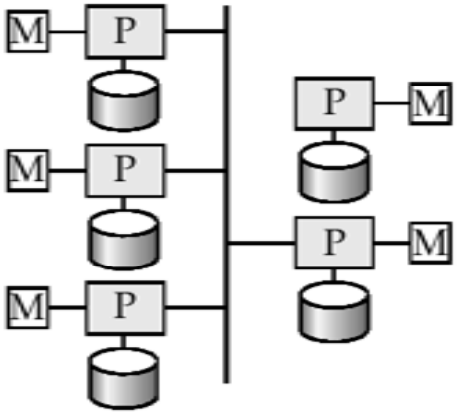
Parallel Database Architectures



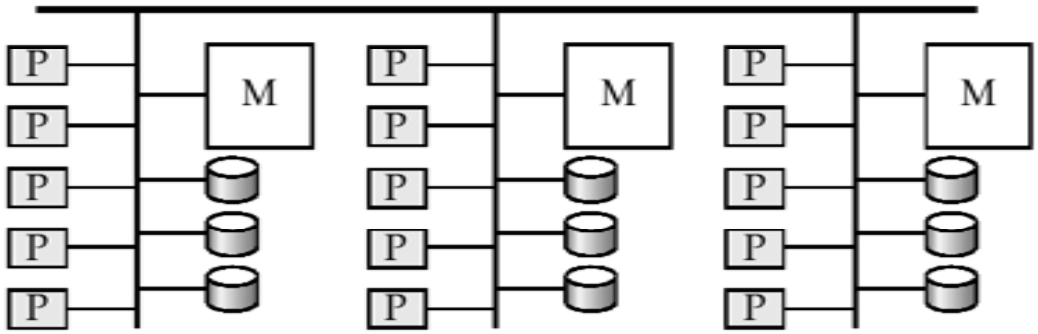
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

Shared Memory

- Processors and disks have access to a common memory, (via bus or an interconnection network)
- Extremely efficient communication between processors
  - data in shared memory can be accessed by any processor without having to move it using software
- Downside
  - architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck
- Widely used for lower degrees of parallelism (4 to 8).

Shared Disk

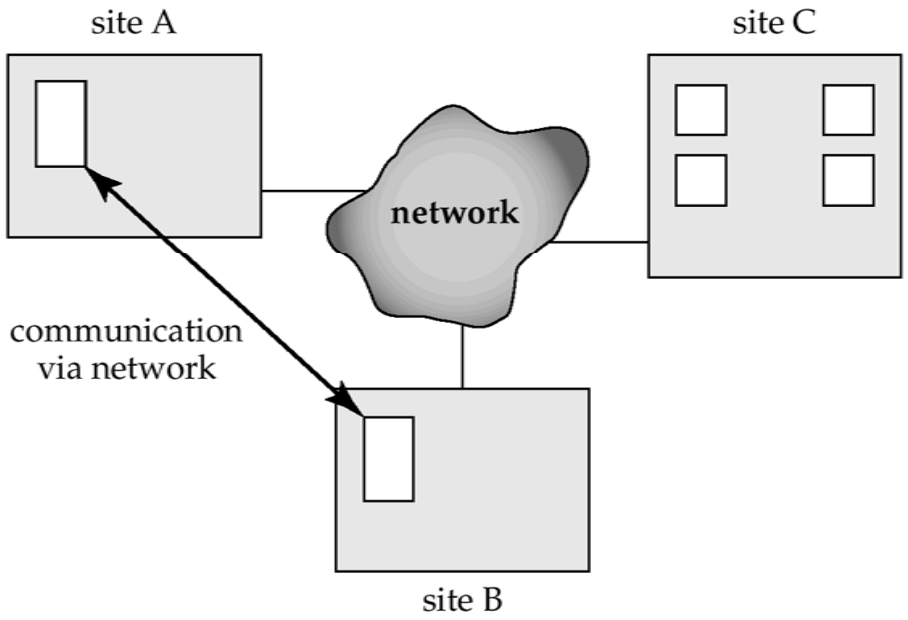
- All processors can directly access all disks via an interconnection network, but the processors have private memories.
  - The memory bus is not a bottleneck
  - Architecture provides a degree of *fault-tolerance* — if a processor fails, the other processors can take over its tasks
- Downside
  - bottleneck now occurs at interconnection to the disk subsystem
  - Communication between processor is slower
- Can scale to a somewhat larger number of processors

Shared Nothing

- A *node* consists of
  - a processor + memory + one or more disks
- Processors communicate through an interconnection network
  - A node functions as the server for the data on the disk(s) it owns.
- Can be scaled up to thousands of processors without interference
  - Data accessed from local disks (and local memory accesses) do not pass through interconnection network: minimizes interference of resource sharing
- Downside
  - cost of communication and non-local disk access; sending data involves software interaction at both ends.

Distributed Systems

- Data spread over multiple machines (*sites* or *nodes*)
- Network interconnects the machines
- Data shared by users on multiple machines



Distributed Databases

- Homogeneous distributed databases
  - Same software/schema on all sites
  - Goal: provide a view of a single database, hiding details of distribution
- Heterogeneous distributed databases
  - Different software/schema on different sites
  - Goal: integrate existing databases to provide useful functionality
- Differentiate between *local* and *global* transactions
  - A local transaction accesses data in the *single* site at which the transaction was initiated.
  - A global transaction accesses data in a site different from the one at which the transaction was initiated

Trade-offs in Distributed Systems

- Sharing data
  - users at one site able to access data residing at some other sites
- Autonomy
  - each site retains a degree of control over data stored locally
- Higher system availability through redundancy
  - data can be replicated at remote sites
- Added complexity required to ensure proper coordination
  - Software development cost
  - Greater potential for bugs
  - Increased processing overhead

END OF CHAPTER 20