Chapter 17: Database System Architectures

Database System Concepts, 6th Ed.

Compiled By: Abu Ahmed Ferdaus

Chapter 17: Database System Architectures

- Overview
- Centralized and Client-Server Systems
- Parallel Systems
- Distributed Systems
- Network Types

Overview

- The architecture of a database system is greatly influenced by the underlying computer system on which it runs, in particular by such aspects of computer architecture as :
  - Networking: Centralized or client-server system
  - Parallelism
  - Distribution
- Networking: Networking of computers allows some tasks to be executed on a server system and some tasks to be executed on client systems. This division of work has led to ***client-server database system***.
- Parallelism: Parallel processing within a computer system allows database system activities to be speeded up, allowing faster response to transactions, as well as more transactions per second. Queries can be processed in a way that exploits the parallelism offered by the underlying computer system. The need for parallel query processing has led to ***parallel database system***.

Overview

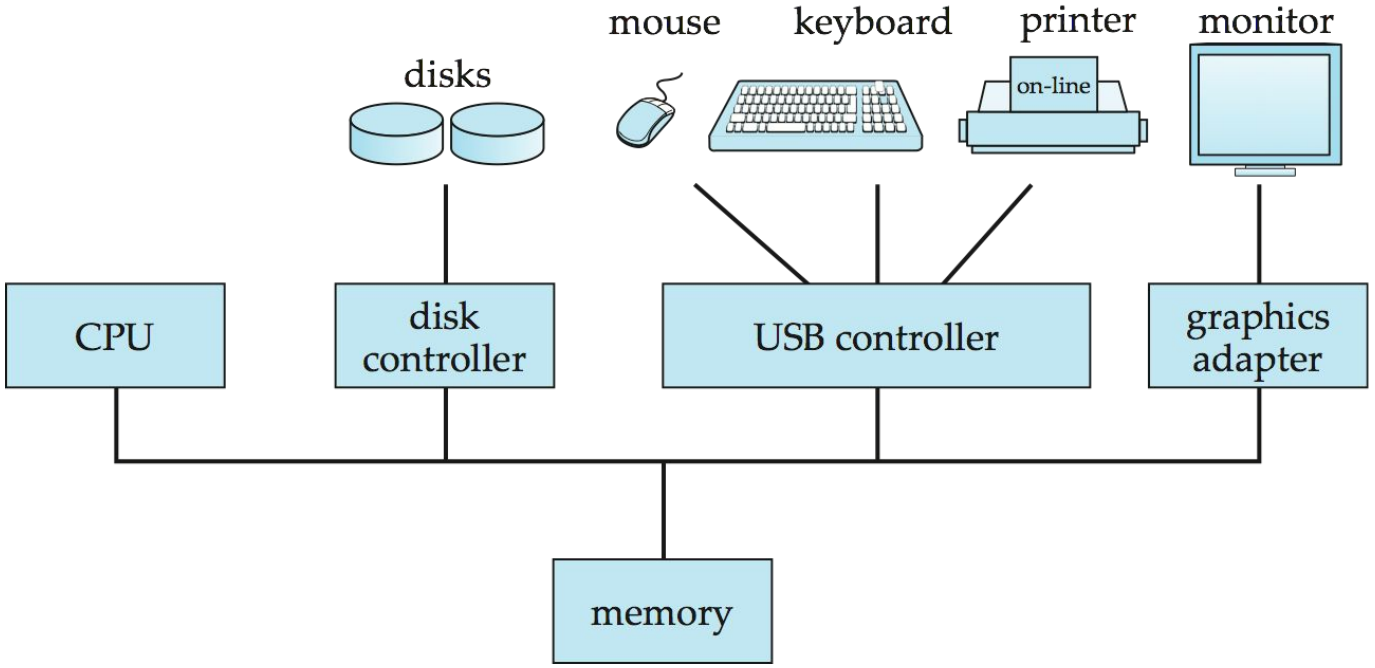
- Distribution:
- Distributing data across sites in an organization allows those data to reside where they are generated and most needed, but still to be accessible from other sites and from other departments.
- Keeping multiple copies of the database across different sites also allows large organizations to continue their database operation even when one site is affected by a natural disaster, such as flood, fire or earthquake.
- ***Distributed database systems*** handle geographically or administratively distributed data spread across multiple database system.

Centralized Systems

- Run on a single computer system and do not interact with other computer systems.
- Spans a range from single-user database system running on a personal computer to high-performance database systems running on high-end server systems.
- A modern, general-purpose computer system consists of one to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory.
- The processors have local cache memories and may have several independent **cores**, each of which can execute a separate instruction stream.
- Each device controller is in charge of a specific type of device (for example, a disk drive, an audio device, or a video display).
- The processors and the device controllers can execute concurrently, competing for memory access.
- Cache memory reduces the contention for memory access, since it reduces the number of times that the processor needs to access the shared memory.

Centralized Systems

- Computers are used in two ways:
  - Single-user system (e.g., personal computer or workstation): desk-top unit, single user, usually has only one CPU and one or two hard disks; the OS may support only one user.
  - Multi-user system: more disks, more memory, multiple CPUs, and a multi-user OS. Serves a large number of users who are connected to the system via terminals. Often called *server* systems.

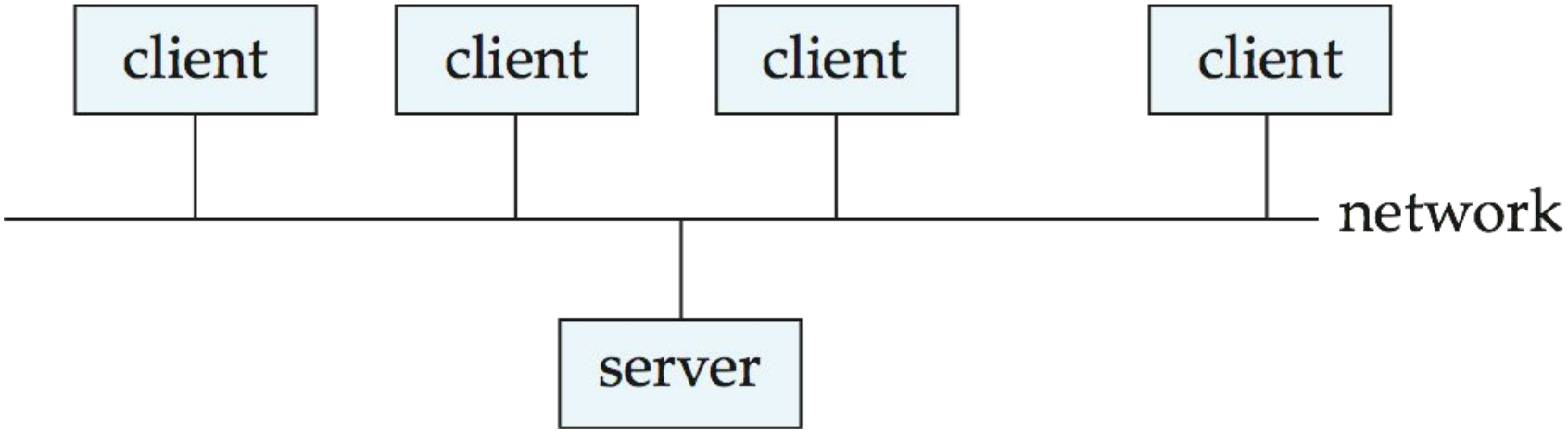


Centralized Systems (Cont.)

- Although most general-purpose computer systems in use today have multiple processors, they have **coarse-granularity parallelism**, with only a few processors (about two to four, typically), all sharing the main memory.
- Databases running on such machines usually do not attempt to partition a single query among the processors; instead, they run each query on a single processor, allowing multiple queries to run concurrently. Thus, such systems support a higher throughput; that is, they allow a greater number of transactions to run per second, although individual transactions do not run any faster.
- Databases designed for single-processor machines already provide multi-tasking, allowing multiple processes to run on the same processor in a time-shared manner, giving a view to the user of multiple processes running in parallel.
- Thus, coarse-granularity parallel machines logically appear to be identical to single processor machines, and database systems designed for time-shared machines can be easily adapted to run on them.
- In contrast, machines with **fine-granularity parallelism** have a large number of processors, and database systems running on such machines attempt to parallelize single tasks (queries, for example) submitted by users.

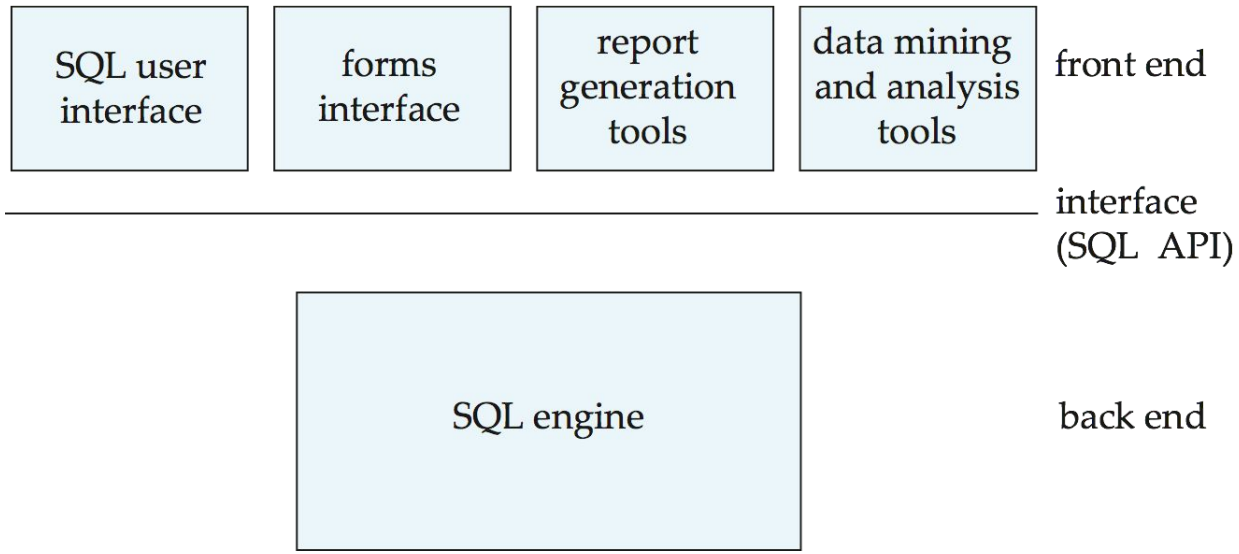
Client-Server Systems

- As personal computers became faster, more powerful, and cheaper, there was a shift away from the centralized system architecture.
- Personal computers supplanted terminals connected to centralized systems. Correspondingly, personal computers assumed the user-interface functionality that used to be handled directly by the centralized systems.
- Client-server systems have functionality split between a server system and multiple client systems.
- Server systems satisfy requests generated at *m* client systems, whose general structure is shown below:



Client-Server Systems (Cont.)

- Database functionality can be broadly divided into:
  - **Back-end**: manages access structures, query evaluation and optimization, concurrency control and recovery.
  - **Front-end**: consists of tools such as SQL user interface, forms interfaces, report generation tools, graphical user interface facilities and data mining and analysis tools .
- 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
  - flexibility in locating resources and expanding facilities
  - better user interfaces
  - easier maintenance
- Standards such as *ODBC* and *JDBC* were developed to interface clients with servers. Any client that uses the ODBC or JDBC interface can connect to any server that provides the interface.
- Systems that deal with large number of users adopt a three-tier architecture, where the front-end is a Web browser which talks to an application server. The application server, in effect, acts as a client to the database server.
- Server systems can be broadly categorized as
  - 1. transaction servers and
  - 2. data servers.

Transaction Servers

- Also called **query server** systems or SQL *server* systems
  - Clients send requests to perform an action to the server
  - Transactions are executed at the server
  - Results are shipped back to the client and displayed.
- Requests are specified in SQL, and communicated to the server through a *remote procedure call* (RPC) mechanism.
- Transactional RPC allows many RPC calls form a transaction.
- *Open Database Connectivity* (ODBC) is a C language application program interface standard from Microsoft for connecting to a server, sending SQL requests, and receiving results.
- JDBC standard is similar to ODBC, for Java
- Widely used in relational database systems.

Data Servers

- Used in high-speed LANs, in cases where
  - The clients are comparable in processing power to the server
  - The tasks to be executed are compute intensive.
- Data are shipped to clients where processing is performed, and then shipped results back to the server.
- This architecture requires full back-end functionality at the clients.
- Used in many object-oriented database systems

Cloud Based Servers

- Servers are usually owned by the enterprise providing the service, but there is an increasing trend for service providers to rely at least in part upon servers that are owned by a "third party" that is neither the client nor the service provider.
- One model for using third-party servers is to outsource the entire service to another company that hosts the service on its own computers using its own software.
- Another model for using third-party servers is **cloud computing**, in which the service provider runs its own software, but runs it on computers provided by another company.
- Under this model, the third party does not provide any of the application software; it provides only a collection of machines. These machines are not "real" machines, but rather simulated by software that allows a single real computer to simulate several independent computers. Such simulated machines are called **virtual machines.**

Cloud Based Servers

- The service provider runs its software (possibly including a database system) on these virtual machines. A major advantage of cloud computing is that the service provider can add machines as needed to meet demand and release them at times of light load.
- This can prove to be highly cost-effective in terms of both money and energy.

Parallel Systems

- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network.
- Parallel system improve processing and I/O speeds by using multiple CPUs and disks in parallel.
- The driving force behind parallel database systems is the demands of applications that have to query extremely large databases (of the order of terabyte – 10^{12} bytes) or that have to process an extremely large number of transaction per second (of the order of thousands of transaction per second).
- Centralized and client-server database systems are not powerful enough to handle such applications.
- In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which computational steps are performed sequentially.
- A **coarse-grain parallel** machine consists of a small number of powerful processors
- A **massively parallel** or **fine grain parallel** machine utilizes thousands of smaller processors.

Parallel Systems

- Most high-end machine today offer some degree of **coarse-grain parallelism**: 2 to 4 processor machines are common, all sharing the main memory. Databases running on such machines usually do not attempt to partition a single query among the processors; instead, they run each query on a single processor, allowing multiple queries to run concurrently and thus support a higher throughput.
- Massively parallel system are available commercially with hundreds of CPUs and disks and database systems running on such machines attempt to parallelize single tasks (queries) submitted by users.
- Two main **performance measures** of a database system:
  - **throughput** --- the number of tasks that can be completed in a given time interval.
  - **response time** --- the amount of time it takes to complete a single task from the time it is submitted.
- A system that processes a large number of small transactions can improve throughput by processing many transactions in parallel. A system that processes large transactions can improve response time as well as throughput by performing subtasks of each transaction in parallel.

Speed-Up and Scale-Up

- Two important issues in studying parallelism are:
  - **Speedup**: Running a given task in less time by increasing the degree of parallelism is called speed up.
  - **Scaleup**: Handling larger tasks by increasing the degree of parallelism is called scaleup.
- **Speedup**: Consider a database application running on a parallel system with a certain number of processors and disks. Now suppose that we increase the size of the system by increasing the number of processors, disks and other components of the system. The goal is to process the task in time inversely proportional to the number of processors and disks allocated. Suppose that the execution time of a task on the larger machine is T_L, and that the execution time of the same task on the smaller machine is T_S. The speedup due to parallelism is define as:

$$speedup = \frac{\textit{small system execution time, } T_S}{\textit{large system execution time, } T_L}$$

Speed-Up and Scale-Up

- **Linear speedup**: if the speedup is N when the large system has N times the resources (CPU, disks and so on) of the smaller system.
- **Sublinear speedup:** if the speedup is less than N, the system is said to demonstrate sublinear speedup.

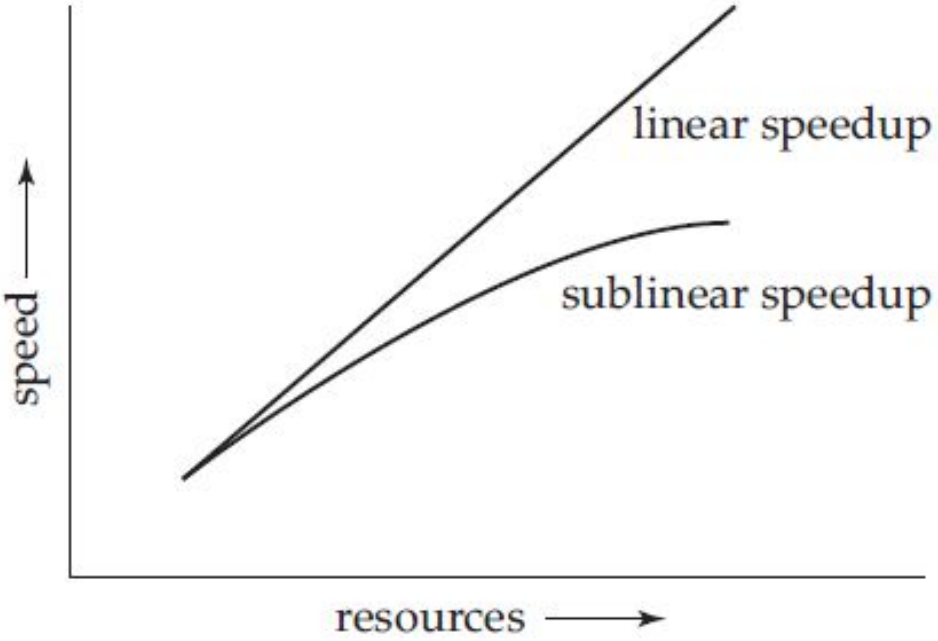


Figure 17.5 Speedup with increasing resources.

Speed-Up and Scale-Up

- **Scaleup**: Scaleup relates to the ability to process large tasks in the same amount of time by providing more resources. Let Q be a task, and let Q_N be a task that is N times bigger than Q. Suppose that the execution time of task Q on a given machine M_S is T_S, and the execution time of task Q_N on a parallel machine M_L, which is N times larger than M_S, is T_L. The scaleup is defined as:

$$scaleup = \frac{small\ system\ small\ task\ execution\ time,\ T_S}{big\ system\ big\ task\ execution\ time,\ T_L}$$

  - **Linear scaleup:** if scaleup is 1 ($T_L = T_s$) on task Q by M_L.
  - **Nonlinear scaleup:** If $T_L > T_s$

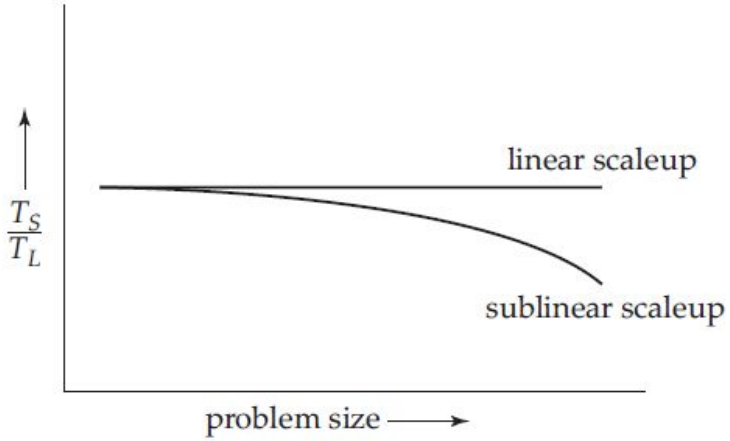


Figure 17.6 Scaleup with increasing problem size and resources.

Batch and Transaction Scaleup

- There are two kinds of scaleup that are relevant in parallel database systems, depending on how the size of the task is measured:
- **Batch scaleup:**
  - In **batch scaleup**, the size of the database increases, and the tasks are large jobs whose runtime depends on the size of the database.
  - An example of such a task is a scan of a relation whose size is proportional to the size of the database. Thus, the size of the database is the measure of the size of the problem.
  - Batch scaleup also applies in scientific applications, such as executing a weather simulation at an *N*-times finer resolution, or performing the simulation for an *N*-times longer period of time.
  - A single large job; typical of most decision support queries and scientific simulation.

Batch and Transaction Scaleup

- **Transaction scaleup**:
  - In **transaction scaleup**, the rate at which transactions are submitted to the database increases, and the size of the database increases proportionally to the transaction rate.
  - Numerous small queries/updates submitted by independent users to a shared database; typical transaction processing 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 since transactions can run concurrently and independently on separate nodes, and each transaction takes roughly the same amount of time, even if the database grows.

Factors Limiting Speedup and Scaleup

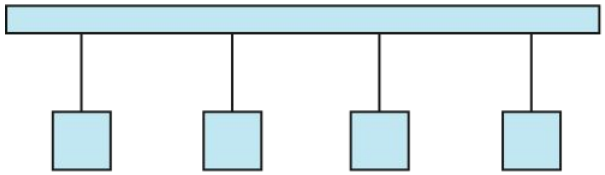
A number of factors works against efficient parallel operation and diminish both speedup and scaleup due to:

- **Start-up costs**: There is a start-up cost associated with initiating a single process. In a parallel operation consisting of thousands of processes, the start-up time may overshadow the actual processing time, affecting speedup adversely. 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 (resource contention), thus spending time waiting on other processes, rather than performing useful work. Both speedup and scaleup are affected by this phenomenon.
- **Skew**: Increasing the degree of parallelism increases the variance in service times of parallely executing tasks. Overall execution time determined by **slowest** of parallely executing tasks. Moreover, it is often difficult to divide a task into exactly equal-sized parts, and the way the sizes are distributed is therefore skewed.

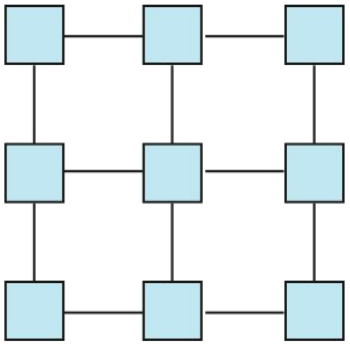
17.3.2 Interconnection Network Architectures

- Parallel systems consist of a set of components (processor, memory and disks) that can communicate with each other via an **interconnec-tion network**. Three commonly used types of interconnection networks are:
- **Bus**. All the system components can send data on and receive data from a single communication bus. The bus could be an Ethernet or a parallel interconnect.
  - Bus architecture works well for small numbers of processors.
  - However, they do not scale well with increasing parallelism, since the bus can handle communication from only one component at a time.
- **Mesh**. The components are arranged as nodes in a grid, and each component is connected to all adjacent components in the grid. In a two-dimensional mesh each nodes connects to **four** adjacent nodes, while in a three-dimensional mesh each node connects to **six** adjacent nodes. Nodes that are not directly connected can communicate with one another by routing messages via a sequence of intermediate nodes that are directly connected to one another.

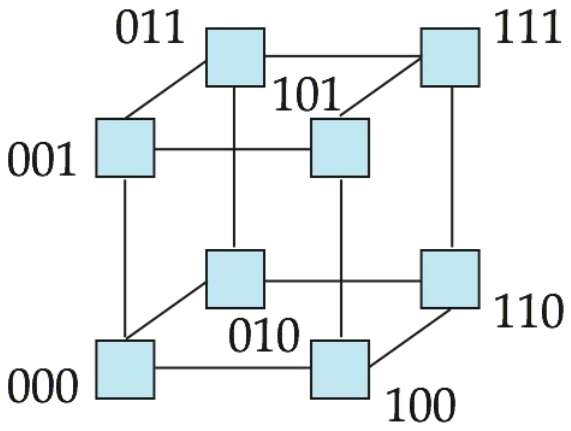
Interconnection Network Architectures



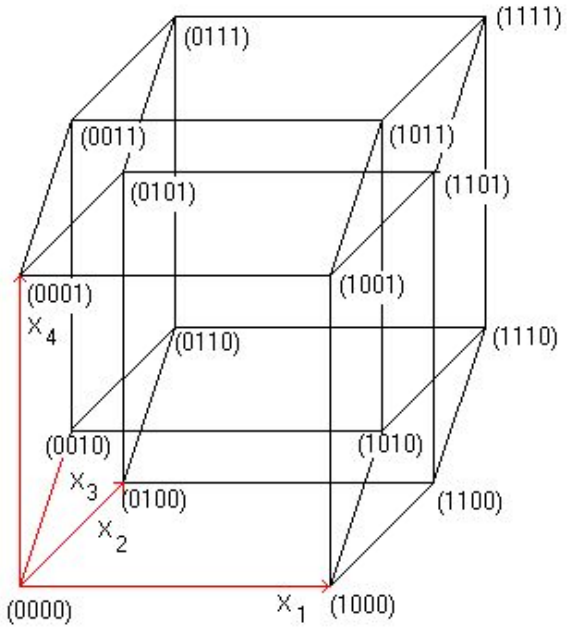
(a) bus



(b) mesh



(c) hypercube



d) Hypercube with 16 Nodes

Interconnection Network Architectures

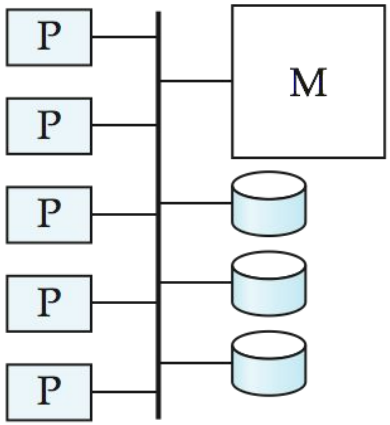
- Communication links grow with growing number of components, and the communication capacity of a mesh therefore scales better with increasing parallelism.
- But may require $2(\sqrt{n}-1)$ hops to send message to a node (or $\sqrt{n}$ with wraparound connections at edge of grid).

- **Hypercube**. The components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit. Thus each of the *n* components in connected to *log(n)* other components.
  - The previous figures shows a hypercube with 8 and 16 nodes.
  - In a hypercube interconnection, a message form a component can reach any other component by going through at most *log(n)* links.
  - In contrast, in a mesh structure, a component may be $2(\sqrt{n} -1)$ links away from some of the other components
  - Thus, communication delays in a hypercube are significantly lower than in a mesh.

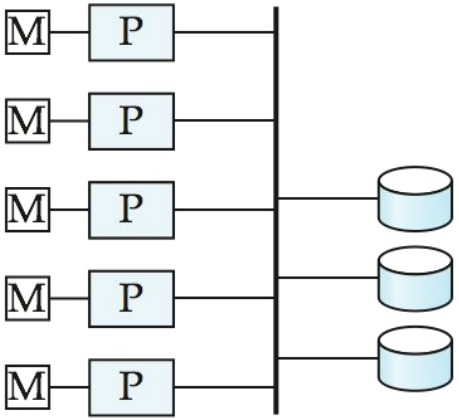
Parallel Database Architectures

- There are several architectural models for parallel machines. Among the most prominent ones are:
- **Shared memory** – All the processors share a common memory and disks.
- **Shared disk** – All the processors share a common set of disks but have independent memories. Shared-disk systems are sometimes called **clusters**.
- **Shared nothing** – The processors share neither a common memory nor common disk.
- **Hierarchical** – This model is a hybrid of the above architectures. This architecture have nodes that share neither memory nor disks with each other, but internally each node has a shared memory or shared disk architecture.

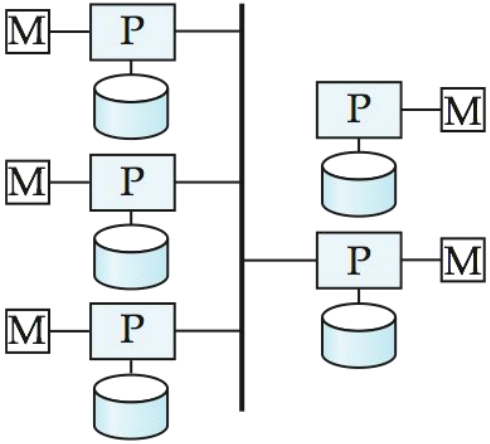
Parallel Database Architectures



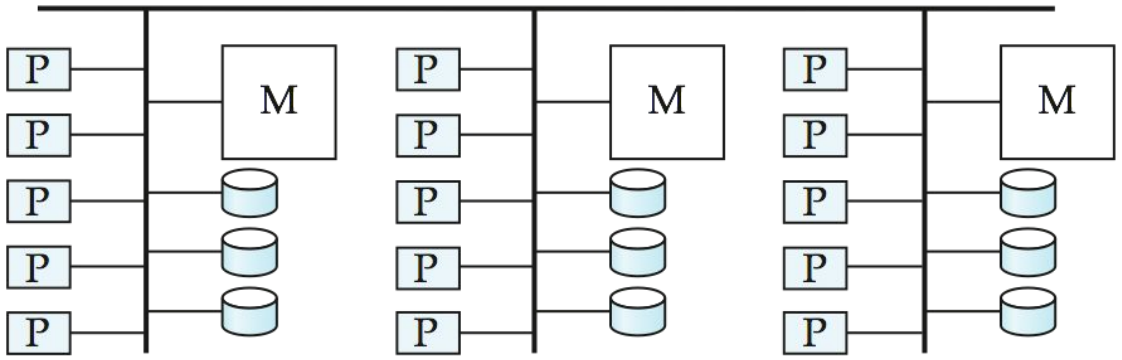
(a) shared memory



(b) shared disk



(c) shared nothing



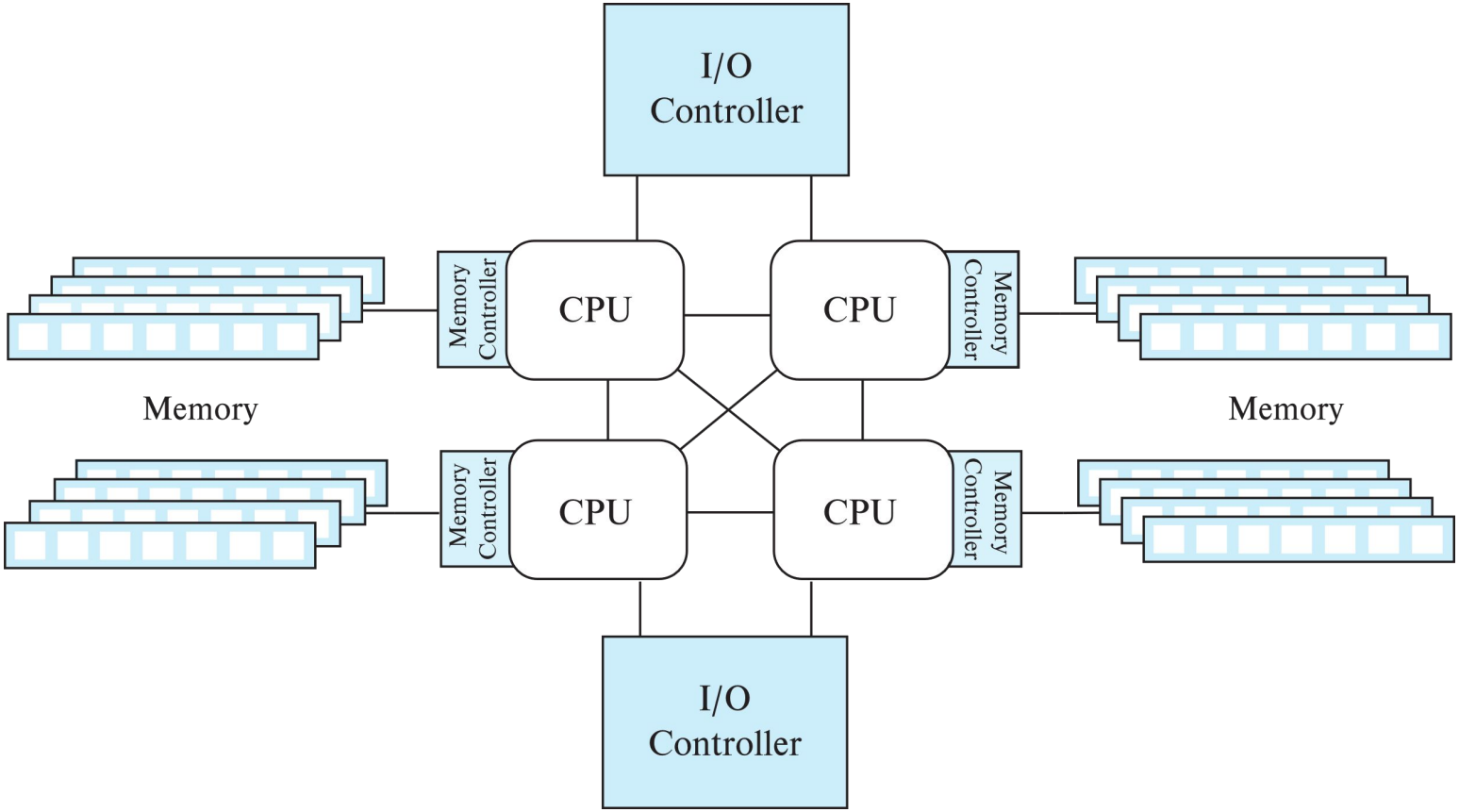
(d) hierarchical

Shared Memory

- Processors and disks have access to a common memory, typically via a bus or through an interconnection network.
- The benefit of shared memory is extremely efficient communication between processors — data in shared memory can be accessed by any processor without being moved with software.
- **Upside** - A processor can send messages to other processors much faster by using memory writes (usually takes less than a microsecond) than by sending a message through a communication mechanism.
- **Downside** – architecture is not scalable beyond 32 or 64 processors since the bus or the interconnection network becomes a bottleneck (since it is shared by all processors).
- Adding more processors does not help after a point, since the processors will spend most of their time waiting for their turn on the bus to access memory.
- Shared-memory architectures usually have large memory caches at each processor, so that referencing of the shared memory is avoided whenever possible. But cache coherency becomes an increasing overhead with increasing number of processors.
- Widely used for lower degrees of parallelism (4 to 8).

Modern Shared Memory Architecture

- Figure shows a conceptual architecture of a modern shared-memory system with multiple processors; note that each processor has a bank of memory directly connected to it, and the processors are linked by a fast interconnect system; processors are also connected to I/O controllers which interface with external storage

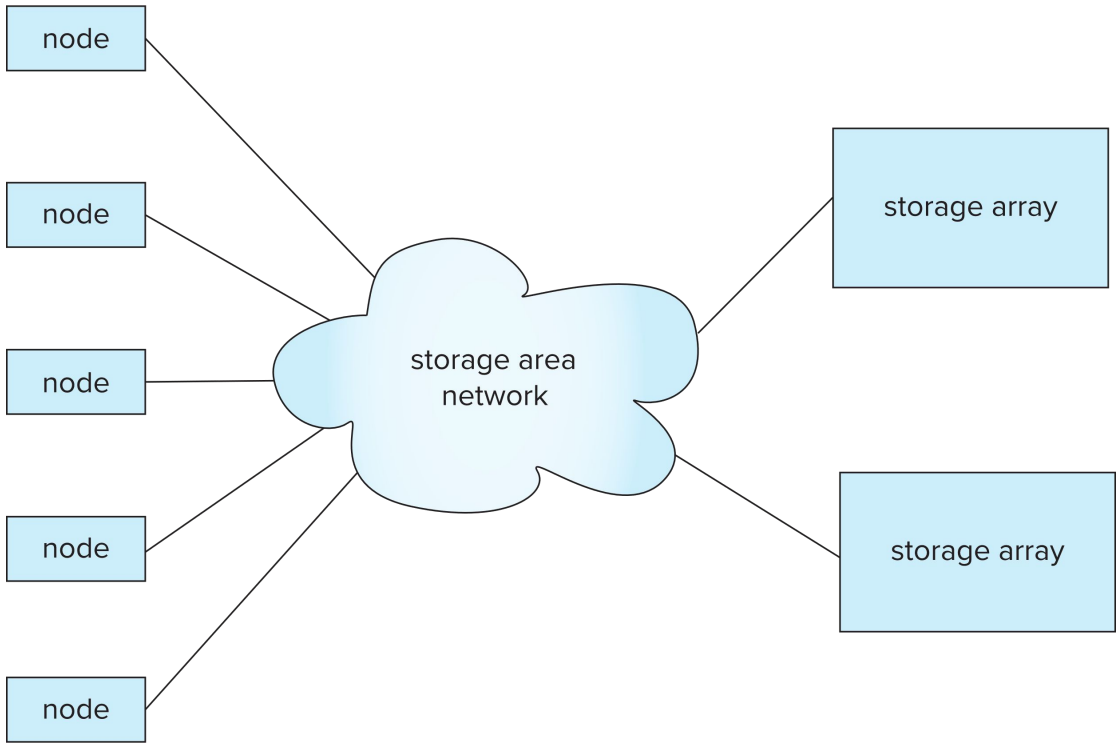


Shared Disk

- All processors can directly access all disks via an interconnection network, but the processors have private memories. There are two advantages over a shared-memory architecture:
  - The memory bus is not a bottleneck, since each processor has its own memory.
  - It offers a cheap way to provide a degree of **fault-tolerance** — if a processor (or its memory) fails, the other processors can take over its tasks since the database is resident on disks that are accessible from all processors.
- We can make the disk subsystem itself fault tolerant by using the RAID architecture. The shared-disk architecture has found acceptance in many applications.
- **Downside:** The main problem with a shared-disk problem is scalability. Although the memory bus is no longer a bottleneck, the interconnection to the disk subsystem is now a bottleneck; it is particularly so in a system where the database makes a large number of accesses to disks.
- Shared-disk systems can scale to a somewhat larger number of processors, but communication across processors is slower, since it has to go through a communication network.

Modern Shared Disk Architectures: via Storage Area Network (SAN)

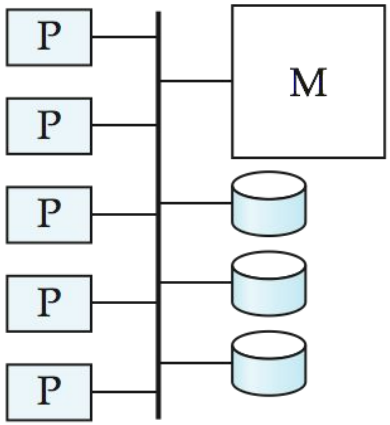
- A **storage-area network** (**SAN**) is a high-speed local-area network designed to connect large banks of storage devices (disks) to nodes that use the data.
- The storage devices physically consist of an array of multiple disks but provide a view of a logical disk, or set of disks, that hides the details of the underlying disks.



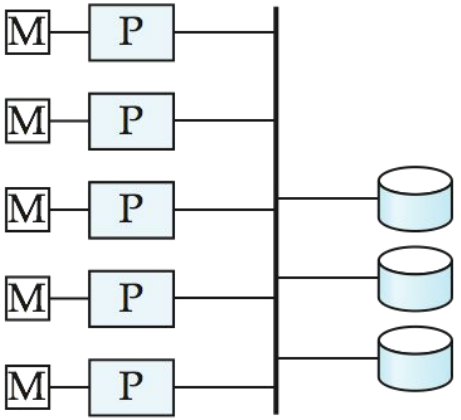
Shared Nothing

- Each node of the machine consists of a processor, memory, and one or more disks. Processors at one node communicate with another processor at another node by a high-speed interconnection network. A node functions as the server for the data on the disk or disks the node owns.
- Data accessed from local disks (and local memory accesses) do not pass through interconnection network, thereby minimizing the interference of resource sharing. Only queries, accesses to nonlocal disks, and result relations pass through the network.
- The interconnection networks for shared-nothing systems are usually designed to scalable, so that their transmission capacity increases as more nodes are added. Consequently, shared-nothing multiprocessors can be scaled up to large no. of processors without interference.
- **Main drawback:** cost of communication and non-local disk access, which is higher than in a shared-memory or shared-disk architecture since sending data involves software interaction at both ends.
- Examples: Teradata, Tandem, Oracle-n CUBE

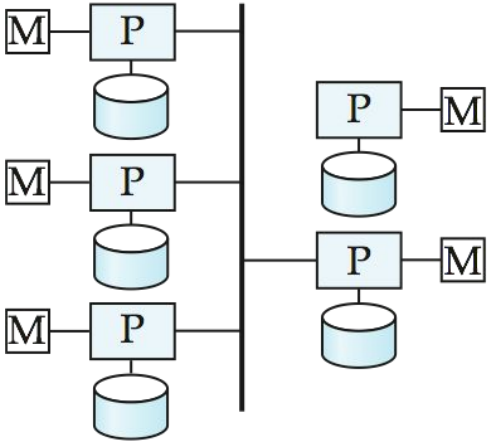
Parallel Database Architectures



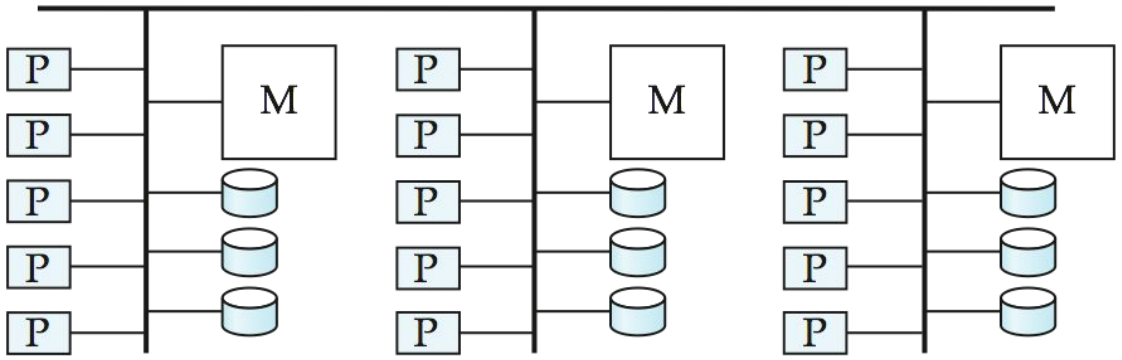
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

Hierarchical

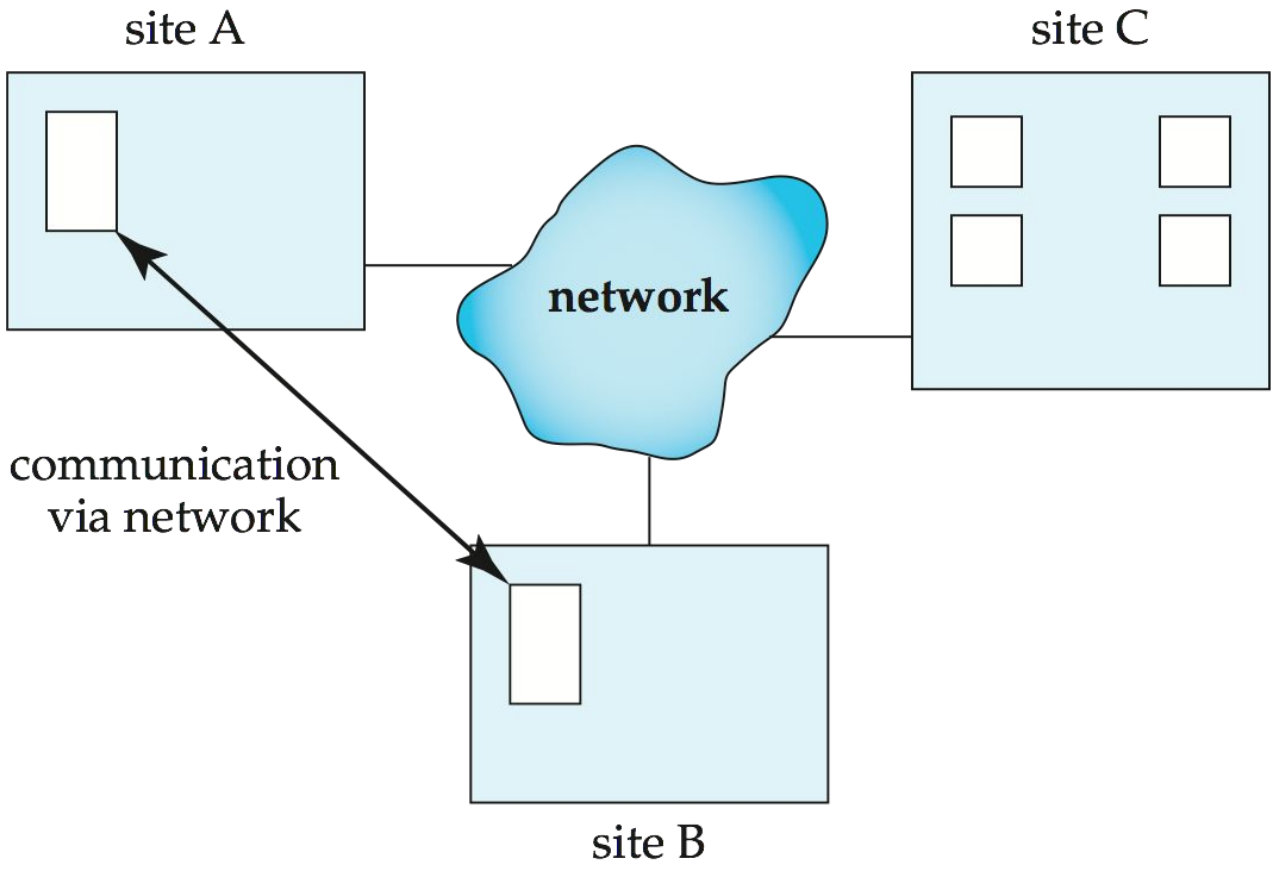
- Combines characteristics of shared-memory, shared-disk, and shared-nothing architectures.
- Top level is a shared-nothing architecture – nodes connected by an interconnection network, and do not share disks or memory with each other.
- Each node of the system could actually be a shared-memory system with a few processors.
- Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
- Thus, a system could be built as a hierarchy, with shared-memory architecture with a few processor at the base, and a shared-nothing architecture at the top, with possibly a shared-disk architecture in the middle.
- Commercial parallel database systems today run on several of these architectures.
- Reduce the complexity of programming such systems by **distributed virtual-memory** architectures
  - Also called **non-uniform memory architecture (NUMA)**

Distributed Systems

- In a distributed database system, the database is stored on different computers. The computers communicate with one another through various communication media, such as high speed networks or telephone lines. They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.
- The computers in a distributed system are referred to as **sites** or **nodes**
- The main difference between shared-nothing parallel database and distributed databases that the distributed databases are geographically separated, are separately administered and have a slower inter-connection.
- Also, we differentiate between local and global transactions in a distributed system.
- Differentiate between *local* and *global* transactions
  - A local transaction accesses data in the *single* site at which the transaction was initiated.
  - A global transaction either accesses data in a site different from the one at which the transaction was initiated or accesses data in several different sites.

Distributed Systems

- Data spread over multiple machines (also referred to as **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, data may be partitioned among 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

Trade-offs in Distributed Systems

- There are several reasons for building distributed database systems:

1. **Sharing data** – users at one site able to access the data residing at some other sites. In a distributed university system, where each campus stores data related to that campus, it is possible for a user in one campus to access data in another campus. Without this capability, the transfer of student records from one campus to another campus would have to resort to some external mechanism that would couple existing systems. Another example: Bank branch and accounts/loans.
2. **Autonomy** – each site is able to retain a degree of control over data stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending on the design of the distributed database system, each administrator may have different degree of **local autonomy**. This local autonomy is often a major advantage of distributed databases.

Trade-offs in Distributed Systems

3. **Higher system availability through redundancy** — data can be replicated at remote sites, and system can function with the help of remaining sites even if a site fails. Although recovery from failure is more complex, availability is crucial for database system used for real-time application.

- **Disadvantage:** added complexity required to ensure proper coordination among sites.
  - **Software development cost.** It is more difficult to implement a distributed system; thus, it is more costly.
  - **Greater potential for bugs.** Since the sites that constitutes the distributed system in parallel, it is harder to ensure the correctness of the algorithms, especially operation during failures of part of the system, and recovery from the failures.
  - **Increased processing overhead.** The exchange of messages and the additional computation required to achieve inter-site coordination are a form of overhead that does not arise in centralized systems.

Network Types

- Distributed databases and client–server systems are built around communication networks. Basically two types:
- **1. Local-area networks (**LANs) – composed of processors that are distributed over small geographical areas, such as a single building or a few adjacent buildings.
- **2. Wide-area networks (**WANs) – composed of processors distributed over a large geographical area (such as the United States or the entire world). WANs with continuous connection (e.g. the Internet) are needed for implementing distributed database systems
- These differences between LAN and WAN imply major variations in the speed and reliability of the communication network, and are reflected in the distributed operating-system design.
- **Storage-area networks** (SAN) – is a special type of high-speed local area network designed to connect large banks of storage devices (disks) to computers that use the data. Thus storage-area networks help build large-scale *shared-disk systems*.

Networks Types (Cont.)

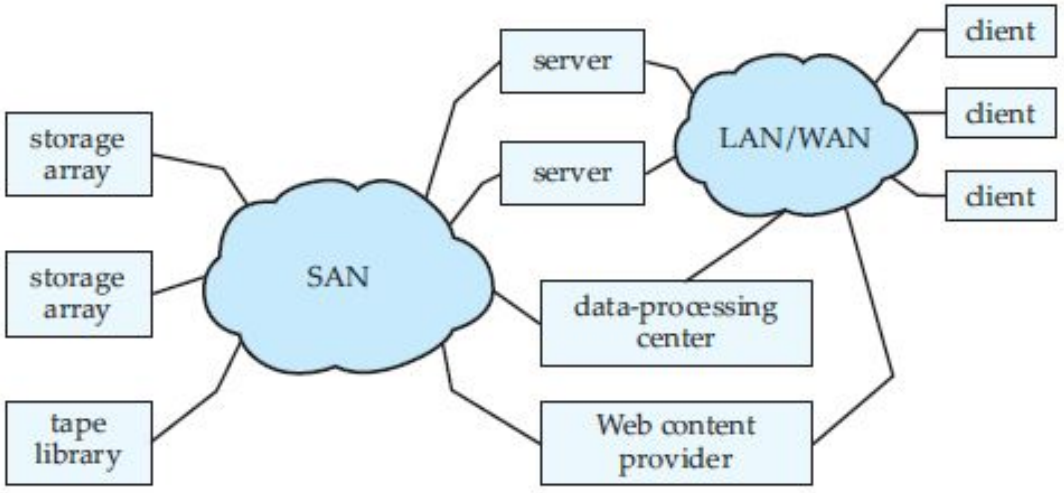


Figure 17.11 Storage-area network.

- Groupware applications such as Lotus notes can work on WANs with discontinuous connection:
  - Data is replicated.
  - Updates are propagated to replicas periodically.
  - Copies of data may be updated independently.
  - Non-serializable executions can thus result. Resolution is application dependent.

End of Chapter

Compiled By: Abu Ahmed Ferdaus