# Database Management Systems

## Module 58: RDBMS Performance & Architecture

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

*ppd@cse.iitkgp.ac.in*

# Module Recap





- Understood the basic issues for optimizing queries
- For every relational expression, usually there are a number of equivalent expressions that can be created by simple transformations
- Final execution plan can be created by choose the estimated least cost expression from the alternates

# Module Objectives





- To evaluate RDBMS, especially with reference to performance and scalability, as a backbone for data-intensive application development
- To understand the role of system and database architecture in performance
- To understand options for Scaling Databases

# Module Outline





- RDBMS Performance and Scalability
- RDBMS Architecture
- Scaling Databases

# RDBMS Performance and Scalability

# What do DBMS Applications Need?



- **Throughput**, **Response Time**, & **Availability**
  - *Throughput is transactions / second (tps)*
  - *Response Time is the delay from submission of transaction to return of result*
  - *Availability is the mean time to failure*
  - At Transaction Level
    - Concurrency Control
    - Query Optimization
  - At System Level
    - System Architecture
    - Database Architecture
    - Performance Tuning
      - *Hardware*: disks to speed up I/O, memory to increase buffer hits, move to a faster processor
      - *Database System Parameters*: set buffer size to avoid paging, set checkpointing to limit log size
      - *Higher level database design*: schema, indices and transactions
- **Correctness**
  - *Any given database transaction must change affected data only in allowed ways*
  - ACID Properties
- **Scalability**
  - *Ability to scale up a database to allow it to hold increasing amounts of data without sacrificing performance*
  - Should be able to scale with volume of data, number of users, diversity of services, geographic expanse, etc.
  - Scalability can be achieved by
    - System Architecture
    - Database Architecture
    - Scale expectations with scale of the system
    - Alternate Data Models
    - Accommodate Hybrid Systems
  - ...

# RDBMS Architecture





## RDBMS Architecture

# RDBMS Architecture



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

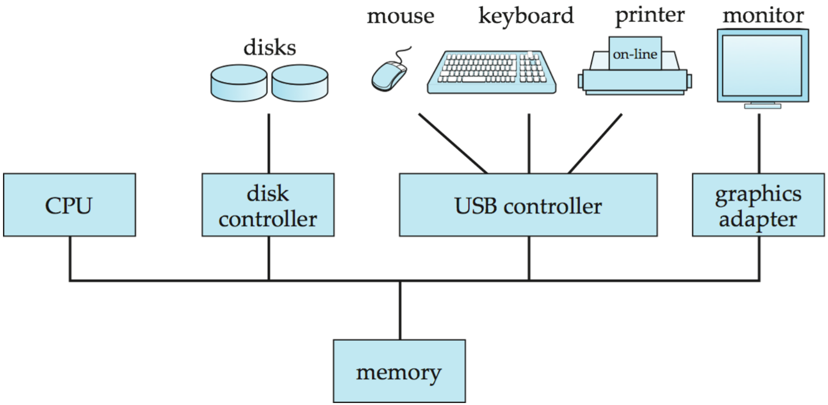
# RDBMS Architecture: Centralized & Client-Server



- **Centralized Architecture**
  - Run on a single computer system and do not interact with other computer systems
  - General-purpose computer system:
    - One to a few CPUs and a number of device controllers that are connected through a common bus that provides access to shared memory
  - Single-user system (for example, 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
    - Serve a large number of users who are connected to the system vie terminals
    - Often called server systems

# RDBMS Architecture: Centralized & Client-Server

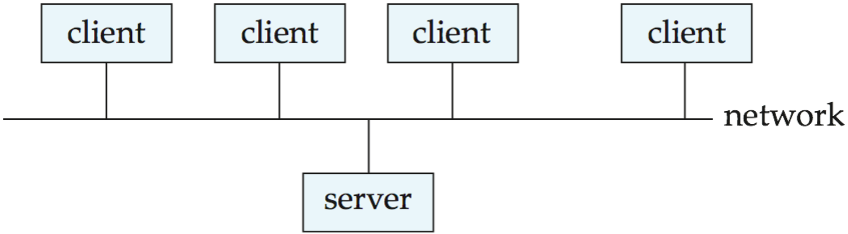




# RDBMS Architecture: Centralized & Client-Server



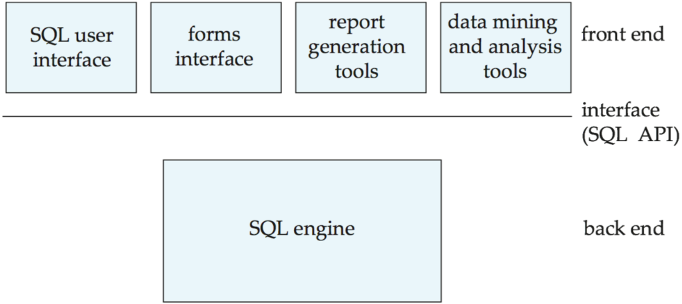
- Server systems satisfy requests generated at $m$ client systems



# RDBMS Architecture: Centralized & Client-Server



- 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



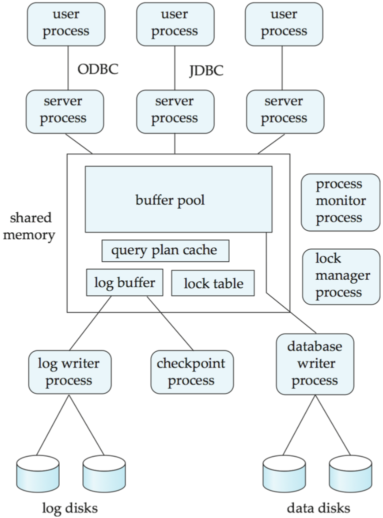
# RDBMS Architecture: Server Systems



- **Transaction or Query servers** which are widely used in relational database systems
  - A typical transaction cycle is:
    - ▷ Clients send requests to the server
    - ▷ Transactions are executed at the server
    - ▷ Results are shipped back to the client
  - Requests are specified in SQL, and communicated to the server through a remote procedure call (RPC) mechanism
  - Transactional RPC allows many RPC calls to form a transaction.
  - ODBC / JDBC used to connect
- **Data servers**, used in object-oriented database systems
  - 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
  - Issues:
    - ▷ Page-Shipping versus Item-Shipping
    - ▷ Locking
    - ▷ Data Caching
    - ▷ Lock Caching

# RDBMS Architecture: Server Systems





# RDBMS Architecture: Parallel Systems



- Parallel database systems consist of multiple processors and multiple disks connected by a fast interconnection network
- 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
- Two main performance measures:
  - **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

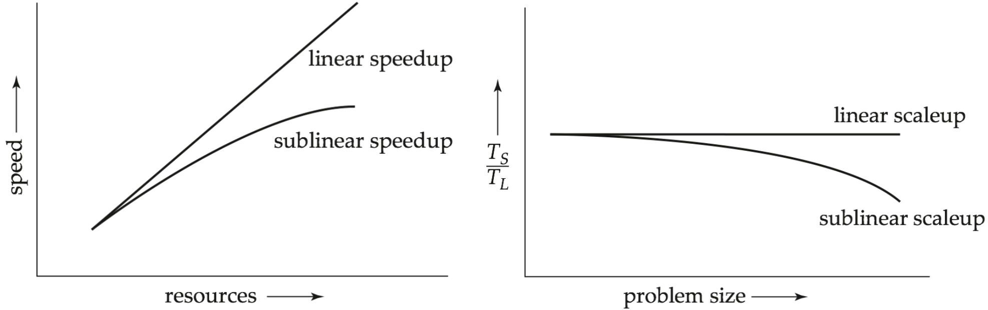
# RDBMS Architecture: Parallel Systems



- **Speedup**: a fixed-sized problem executing on a small system is given to a system which is $N$-times larger
  - Measured by:
    - Speedup $= \dfrac{small\ \ system\ \ elapsed\ \ time}{large\ \ system\ \ elapsed\ \ time}$
  - Speedup is linear if equation equals $N$
  - Speedup Percentage $= \dfrac{Speedup}{N} * 100\%$
- **Scaleup**: increase the size of both the problem and the system $N$-times larger system used to perform $N$-times larger job
  - Measured by:
    - Scaleup $= \dfrac{small\ \ system\ \ small\ \ problem\ \ elapsed\ \ time}{big\ \ system\ \ big\ \ problem\ \ elapsed\ \ time}$
  - Scale up is linear if equation equals 1

# RDBMS Architecture: Parallel Systems: Speedup and Scaleup

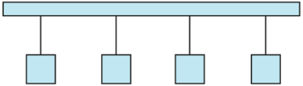


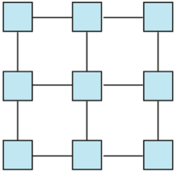


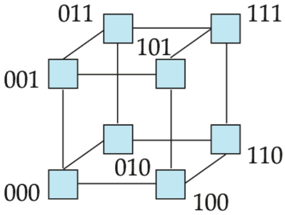

- Speedup and Scaleup are often sublinear due to:
  - **Startup costs**: Cost of starting up multiple processes may dominate computation time, if the degree of parallelism is high
  - **Interference**: Processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other, thus spending time waiting on other processes, rather than performing useful work
  - **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

# RDBMS Architecture: Parallel Systems: Interconnect

- **Bus**: Components send data on and receive data from a single communication bus
  - Does not scale well with increasing parallelism
- **Mesh**: Components are arranged as nodes in a grid, and each component is connected to all adjacent components
  - Communication links grow with growing number of components, and so scales better
  - But may require $2\sqrt{n}$ hops to send message to a node ($\sqrt{n}$ with wraparound connections at edge)
- **Hypercube**: Components are numbered in binary; components are connected to one another if their binary representations differ in exactly one bit
  - $n$ components are connected to $\log n$ other components and can reach each other via at most $\log n$ links; reduces communication delays

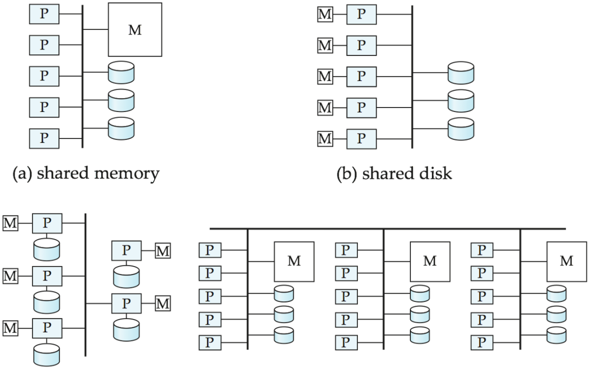






# RDBMS Architecture: Parallel Systems

- **Shared memory**: processors share a common memory
- **Shared disk**: processors share a common disk
- **Shared nothing**: processors share neither a common memory nor common disk
- **Hierarchical**: hybrid of the above architectures



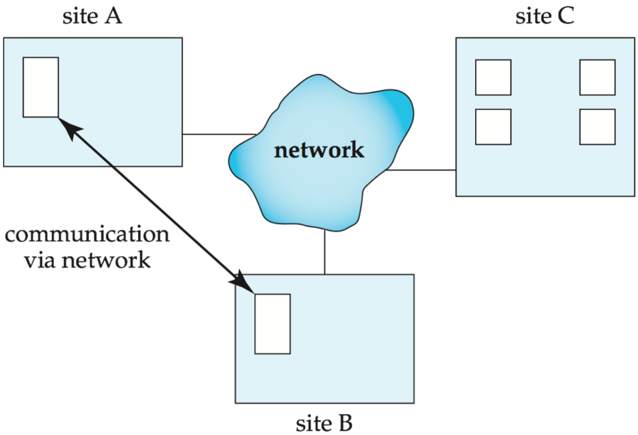
(a) shared memory (b) shared disk

(c) shared nothing (d) hierarchical

# RDBMS Architecture: Distributed Systems



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



# RDBMS Architecture: Distributed Systems



- **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
- 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

# RDBMS Architecture: Distributed Systems



- **Advantages**
  - **Sharing data**: users at one site able to access the data residing at some other sites
  - **Autonomy**: each site is able to retain a degree of control over data stored locally
  - **Higher system availability through redundancy**: data can be replicated at remote sites, and system can function even if a site fails
- **Disadvantages**
  - Added complexity required to ensure proper coordination among sites
    - Software development cost
    - Greater potential for bugs
    - Increased processing overhead

# Scaling Databases

# Scaling RDBMS





- Relational databases → mainstay of business
- Web-based applications caused spikes explosion of social media sites (Facebook, Twitter) with large data needs rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Hooking RDBMS to web-based application becomes trouble
- *Issues with Scaling Up*
  - Best way to provide ACID and rich query model is to have the dataset on a *single m/c*
  - Limits to scaling up (*vertical scaling*: make a "single" machine more powerful) → dataset is just too big!
  - Scaling out (*horizontal scaling*: adding more smaller/cheaper servers) is a better
  - Different approaches for horizontal scaling (multi-node database):
    - Master/Slave
    - Sharding (partitioning)

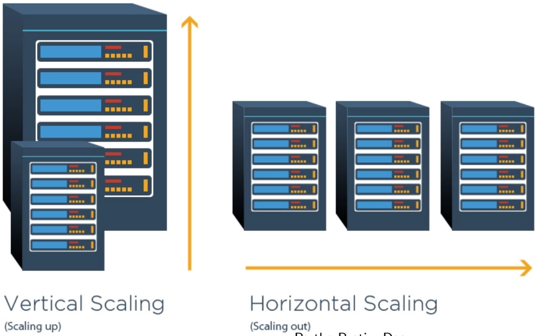
**Source**: *Introduction to NOSQL Databases, SlidePlayer*

# Horizontal Vs. Vertical Scaling





- **What Is Horizontal Scaling?**
  - Horizontal scaling (aka scaling out) is adding additional nodes to infrastructure
  - This adds complexity of your operation. You must decide which machine does what and how your new machines work with your old machines
- **What Is Vertical Scaling?**
  - Vertical scaling (aka scaling up) describes adding additional resources to a system
  - It adds more power to your current machines



# Horizontal Vs. Vertical Scaling (2)

| Horizontal Scaling | Vertical Scaling |
|---|---|
| *Advantages* | |
| • Scaling is easier from a hardware perspective<br>• Fewer periods of downtime<br>• Increased resilience and fault tolerance<br>• Increased performance | • Cost-effective<br>• Less complex process communication<br>• Less complicated maintenance<br>• Less need for software changes |
| *Disadvantages* | |
| • Increased complexity of maintenance and operation<br>• Increased Initial costs | • Higher possibility for downtime<br>• Single point of failure<br>• Upgrade limitations |

**Source**: *Horizontal Vs. Vertical Scaling: How Do They Compare?*

# Scaling out RDBMS





- **Master/Slave**
  - All writes are written to the master
  - All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large datasets can pose problems as master needs to duplicate data to slaves
- **Sharding (Partitioning)**
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards
- **Other Options**
  - Multi-Master replication
  - INSERT only, not UPDATES/DELETES
  - No JOINs, thereby reducing query time → This involves de-normalizing data
  - In-memory databases

**Source**: *Introduction to NOSQL Databases, SlidePlayer*

# Module Summary



- Evaluated RDBMS, especially with reference to performance and scalability, as a backbone for data-intensive application development
- Understood the role of system and database architecture in performance
- Understood the options for scaling databases

**Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.**
**Edited and new slides are marked with “PPD”.**