# Real-Time Distributed Databases

# Definition

- Real-Time Data Base System can be defined as those computing systems that are designed to operate in a timely manner.
- It must perform certain actions within specific timing constrains (producing results while meeting predefined deadlines)
- Real-Time Distributed Data Base System can also be defined as Traditional Distributed Databases that uses an extension to give additional power to yield reliable response.

# *Real-Time* Distributed Databases

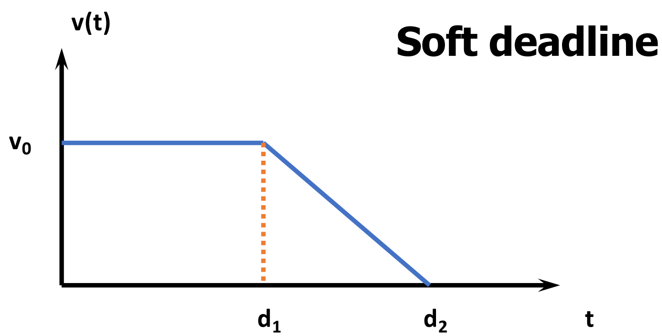
Distributed Databases with the added constraint of completing operations within a certain amount of time to accurately reflect the outside world.

# Real-Time Distributed Databases

- Strategies must consider system attributes
  - Hard or Soft
  - Concurrency Control
  - Replication

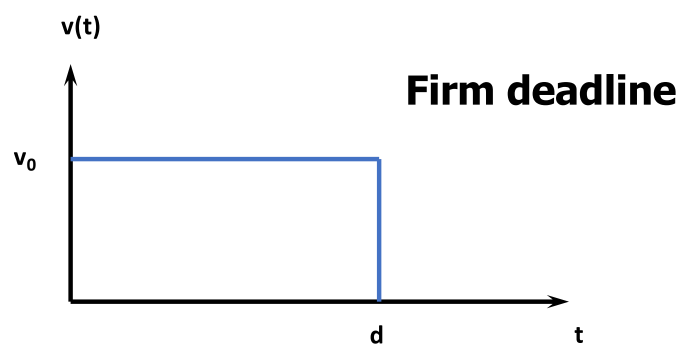
# System Models and Timing Deadlines

- Soft-Deadline:
  - desirable but not critical
  - missing a soft-deadline does not cause a system failure or compromises the system's integrity
  - Example: operator switchboard for a telephone



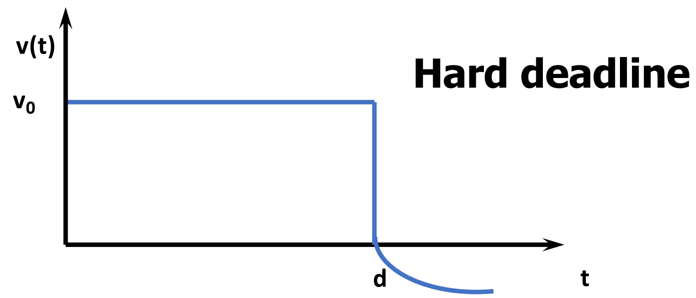
# Deadlines

- Firm-Deadline:
  - Desirable but not critical (like Soft-Deadline case)
  - It is not executed after its deadline and no value is gained by the system from the tasks that miss their deadlines
  - Example: an autopilot system



# Deadlines

- Hard-Deadline:
  - Timely and logically correct execution is considered to be critical
  - Missing a hard-deadline can result in catastrophic consequences
  - Also known as Safety-Critical
  - Example: data gathered by a sensor



# Hard vs. Soft

**Hard**

- Strict timing constraints
- Data and Service guaranteed
- Deadlines met to avoid catastrophe
- Example:
  - Control tower notifying planes where to land in inclement weather
  - Burglary System dispatch

**Soft**

- Less strict timing constraints
- Failure to meet deadlines not dangerous
- Value of data declines after deadline.
  - Example: Checkout line growing in a grocery store.

# Design Paradigms

- Time-Triggered (TT)
  - Systems are initiated as predefined instances
  - Assessments of resource requirements and resource availability is required
  - TT architecture can provide predictable behavior due to its pre-planed execution pattern.

# Design Paradigms

- Event-Triggered (ET)
  - Systems are initiated in response to the occurrence of particular events that are possibly caused by the environment
  - The resource-need assessments in ET architecture is usually probabilistic
  - ET is not as reliable as TT but provides more flexibility and ideal for more classes of applications
  - ET behavior usually is not predictable.

# Tasks Periodicity

- Prosodic Tasks
  - Executes at regular intervals of time
  - Corresponds to TT architecture
  - Have Hard-Deadlines characterized by their periods (requires worst-case analysis).

- Aperiodic Tasks
  - Execution time cannot be priori anticipated
  - Activation of tasks is random event caused by a trigger
  - Corresponds to ET architecture
  - Have Soft-Deadlines (no worst-case analysis)

# Tasks Periodicity

- Sporadic Tasks
  - Tasks which are aperiodic in nature, but have Hard-Deadlines
  - Used to handle emergency conditions or exceptional situations
  - Worst-case calculations is done using Schedulability-Constraint
  - Schedulability-Constraint defines a minimum period between any two sporadic events from the same source.

# Scheduling

- Each task within a real-time system has
  - Deadline
  - An arrival time
  - Possibly an estimated worst-case execution
- A Scheduler can be defined as an algorithm or policy for ordering the execution of the outstanding process
- Scheduler maybe:
  - Preemptive
    - Can arbitrarily suspend and resume the execution of the task without affecting its behavior

# Scheduling (Cont)

- Non-preemptive
  - A task must be rum without interruption until completion
  - Hybrid
    - Preemptive scheduler, but preemption is only allowed at certain points within the code of each task.
  - Real-Time scheduling algorithms can be :
    - Static
      - Known as fixed-priority where priorities are computed off-line
      - Requires complete priori knowledge of the real-time environment in which is deployed
      - Inflexible: scheme is workable only if all the tasks are effectively periodic.
      - Can work only for simple systems, performs inconsistently as the load increases.

# Scheduling (Cont)

- Dynamic
  - Assumes unpredictable task-arrival times
  - Attempts to schedule tasks dynamically upon arrival
  - Dynamically computes and assigns a priority value to each task
  - Decisions are based on task characteristics and the current state of the system
  - Flexible scheduler that can deal with unpredictable events.

# Priority-Based Scheduling

- Conventional scheduling algorithms aims at balancing the number of CPU-bound and I/O bound jobs to maximize system utilization and throughput
- Real-Time tasks need to be scheduled according to their criticalness and timeliness
- Real-Time system must ensure that the progress of higher-priority tasks (ideally) is never hindered by lower-priority tasks.

# Priority-Based Scheduling Methods

- Earliest-Deadline-First (EDF):
  - the task with the current closest (earliest) deadline is assigned the highest priority in the system and executed next
- Value-Functions : highest value (benefit) first
  - the scheduler is required to assign priorities as well as defining the system values of completing each task at any instant in time

# Priority-Based Scheduling Methods

- Value-Density (VD): highest (value/computation) first
  - The scheduler tends to select the tasks that earn more value per time unit they consume
  - It is a greedy technique since it always schedules that task that has the highest expected value within the shortest possible time unit.
- Complex functions of deadline, value and slack time.

# Concurrency Control in Real Time DDB

- Ensures data is accurate in a real-time distributed system
- Two main approaches:
  - Prevent Collisions (Pessimistic)
  - Detect Collisions and Respond (Optimistic)

# Pessimistic Real-time Concurrency Control

- Two-Phase Locking:
  - Transactions acquire locks on data
  - After transaction completed, locks are removed.
  - Ensures that data integrity isn't compromised.

# Pessimistic Concurrency Control

- Disadvantages to 2PL:
  - Don’t scale well to real time distributed systems
  - Difficult to maintain locks at different locations
  - Problems inherent to locking multiplied by number of sites

# Optimistic Concurrency Control

- OCC:
  - Assumed that collisions won’t occur
  - Few or no read restrictions
  - Initial writing takes place on copy of data
  - Course of action can be decided based on collision

# Optimistic Concurrency Control

- Disadvantages to OCC:
  - The less servers, the more likely collisions
  - Collisions always cause rollbacks
  - Time wasted while restarting transactions

# Variations on CC

- Neither OCC nor PCC perfect for RTDDBS
- Variations/Augmentations frequent:
  - DHP-2PL
  - OCC Wait-50

# Replication Strategies in Real-Time Distributed Databases

## Replication

- Deadlines must be met
- Fault tolerance
- Failure Transparency
- Replication helps maintain Quality of Service and Data Freshness

# Full Replication: Eager vs. Lazy Update

Eager Update

- Replicas updated as transaction happens.

- High response times from clients.
- Locked longer.
- High Overhead

Lazy Update

- Replicas updated after transaction committed.

- Chance for inconsistency.

# Full Replication: Update Anywhere vs. Primary Copy

Update Anywhere

- Any replica can update other replicas.
- Good for fault tolerance.
- High synching and update times.

Primary Copy

- One replica designated as "server".
- Good for read-only transactions.
- Restarts at server mean long waits and missed deadlines.

# Alternative: Partial Replication

- JITRTR: Replicate as needed.
- Only parts of the database replicated to cut down on overhead.
- Works best in static systems