Distributed Database Systems

Vera Goebel

Department of Informatics

University of Oslo

Outline:
- Terminology and definitions
- Architectures
- Problems, concepts and approaches

1

1

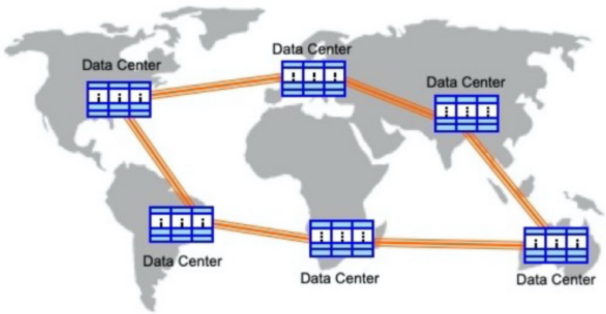
Distributed Computing

- A number of autonomous processing elements (not necessarily homogeneous) that are interconnected by a computer network and that cooperate in performing their assigned tasks.
- What is being distributed?
  - Processing logic
  - Function
  - Data
  - Control

2

2

Current Distribution – Geographically Distributed Data Centers



3

3

What is a Distributed Database System?

Distributed DB: collection of multiple, logically interrelated databases distributed over computer network.

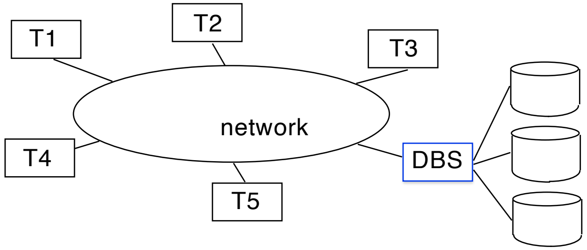
Distributed DBMS: software managing distributed DB, provides access mechanism to make distribution transparent to users.

4

4

Centralized DBS

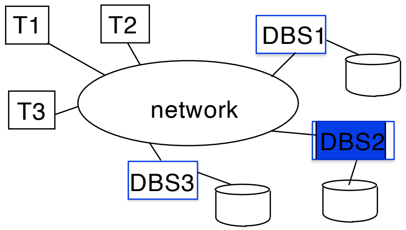
- logically integrated
- physically centralized



Traditionally: one large mainframe DBMS + n “stupid” terminals

Distributed DBS (P2P Dist. DBS)

- Data logically integrated (i.e., access based on one schema)
- Data physically distributed among multiple database nodes
- Processing is distributed among multiple database nodes



Why a Distributed DBS?

Performance via parallel execution
- more users
- quick response

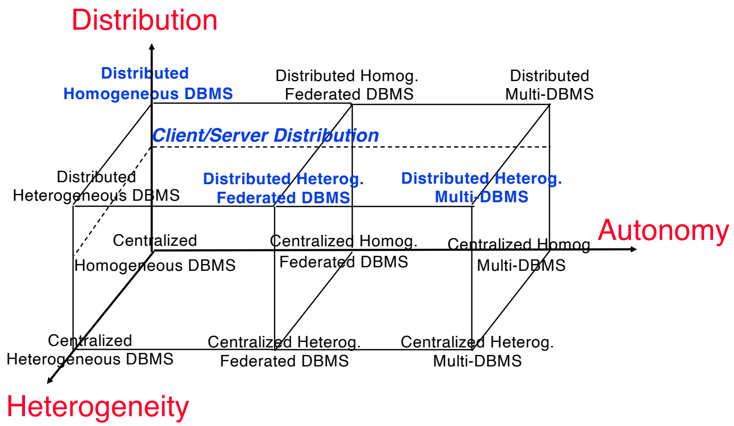
More data volume
- max disks on multiple nodes

Traditionally: m mainframes for the DBMSs + n terminals

Distributed DBMS

- Advantages:
  - Improved performance
  - Efficiency
  - Extensibility (addition of new nodes)
  - Transparency of distribution
    - Storage of data
    - Query execution
  - Autonomy of individual nodes
- Problems:
  - Complexity of design and implementation
  - Data consistency
  - Safety
  - Failure recovery

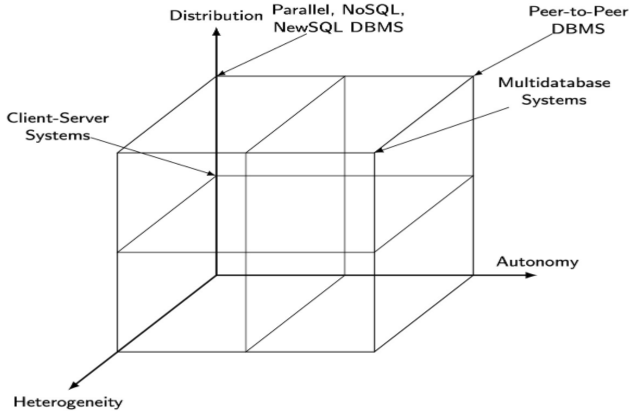
Classification of Distributed DBMS Alternatives



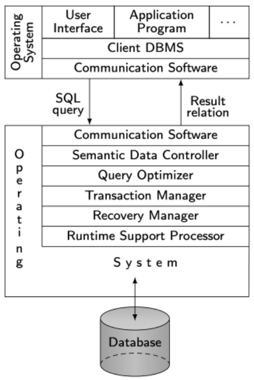
Dimensions of the Problem

- Distribution
  - Whether the components of the system are located on the same machine or not
- Heterogeneity
  - Various levels (hardware, communications, operating system)
  - DBMS important one
    - data model, query language, transaction management algorithms
- Autonomy
  - Not well understood and most troublesome
  - Various versions
    - Design autonomy: Ability of a component DBMS to decide on issues related to its own design.
    - Communication autonomy: Ability of a component DBMS to decide whether and how to communicate with other DBMSs.
    - Execution autonomy: Ability of a component DBMS to execute local operations in any manner it wants to.

Modern Distributed DBMS Alternatives



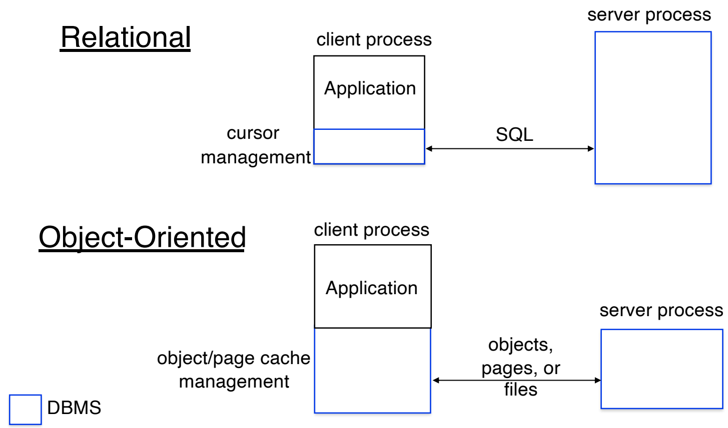
Client/Server Architecture



Advantages of Client/Server Architectures

- More efficient division of labor
- Horizontal and vertical scaling of resources
- Better price/performance on client machines
- Ability to use familiar tools on client machines
- Client access to remote data (via standards)
- Full DBMS functionality provided to client workstations
- Overall better system price/performance

Client/Server Architectures



13

Comparison of Client/Server Architectures

Page & File Server

- Simple server design
- Complex client design
- Fine grained concurrency control difficult
- Very sensitive to client buffer pool size and clustering

Object Server

- Complex server design
- "Relatively" simple client design
- Fine-grained concurrency control
- Reduces data movement, relatively insensitive to clustering
- Sensitive to client buffer pool size

Conclusions:

- No clear winner
- Depends on object size and application's object access pattern
- File server ruled out

14

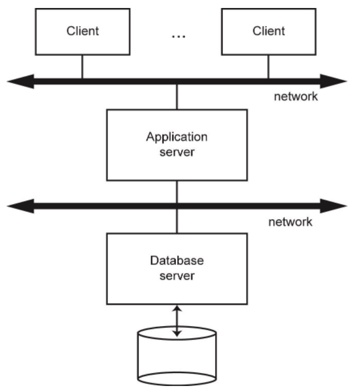
Cache Consistency in Client/Server Architectures

	Synchronous	Asynchronous	Deferred
Avoidance-Based Algorithms	Client sends 1 msg per lock to server; Client waits; Server replies with ACK or NACK.	Client sends 1 msg per lock to the server; Client continues; Server invalidates cached copies at other clients.	Client sends all write lock requests to the server at commit time; Client waits; Server replies when all cached copies are freed.
Detection-Based Algorithms	Client sends object status query to server for each access; Client waits; Server replies.	Client sends 1 msg per lock to the server; Client continues; After commit, the server sends updates to all cached copies.	Client sends all write lock requests to the server at commit time; Client waits; Server replies based on W-W conflicts only.

Best Performing Algorithms

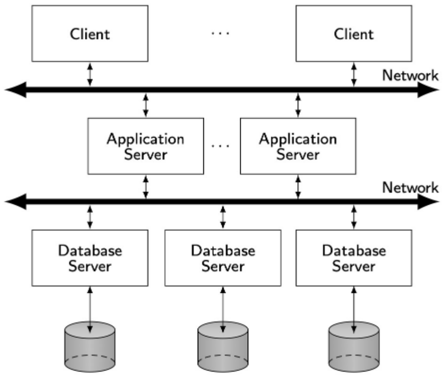
15

Database Server

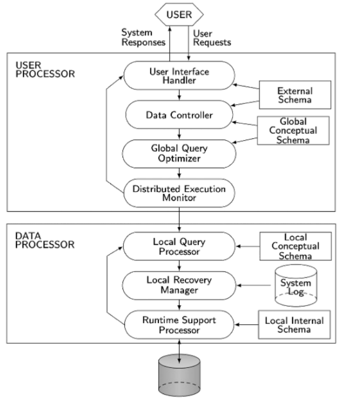


16

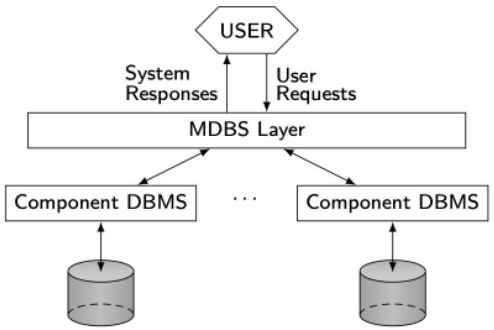
Distributed Database Servers



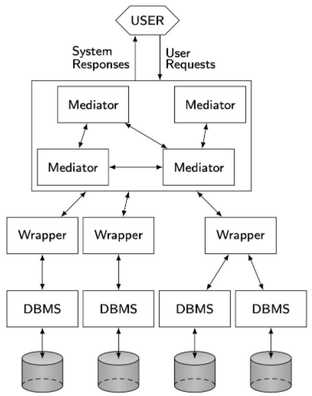
Peer-to-Peer (P2P) Component Architecture



Multi-DBS Components & Execution

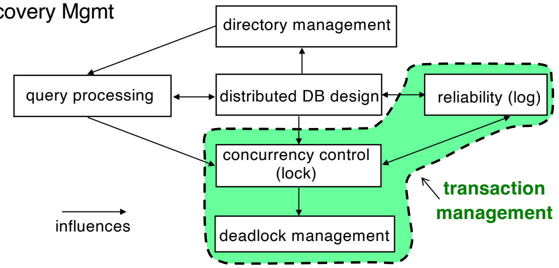


Mediator/Wrapper Architecture



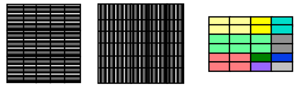
Distributed DBMS Functionality

Distributed Database Design
Distributed Directory/Cataloge Mgmt
Distributed Query Processing and Optimization
Distributed Transaction Mgmt
- Distributed Concurreny Control
- Distributed Deadlock Mgmt
- Distributed Recovery Mgmt



Distributed Database Design

- horizontal fragmentation: distribution of “rows”, selection
- vertical fragmentation: distribution of “columns”, projection
- hybrid fragmentation: ”projected columns” from ”selected rows”
- allocation: which fragment is assigned to which node?
- replication: multiple copies at different nodes, how many copies?



- Design factors:
  - Most frequent query access patterns
  - Available distributed query processing algorithms
- Evaluation Criteria
  - Cost metrics for: network traffic, query processing, transaction mgmt
  - A system-wide goal: Maximize throughput or minimize latency

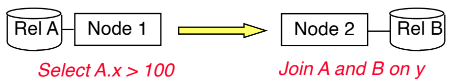
Distributed Directory and Catalogue Management

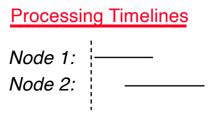
- Directory Information:
  - Description and location of records/objects
    - Size, special data properties (e.g.,executable, DB type, user-defined type, etc.)
    - Fragmentation scheme
  - Definitions for views, integrity constraints
- Options for organizing the directory:
  - Centralized ← Issues: bottleneck, unreliable
  - Fully replicated ← Issues: consistency, storage overhead
  - Partitioned ← Issues: complicated access protocol, consistency
  - Combination of partitioned and replicated ← e.g., zoned, replicated zoned

Distributed Query Processing and Optimization

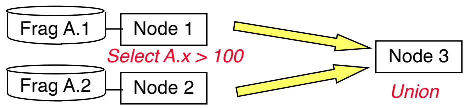
- Construction and execution of query plans, query optimization
- Goals: maximize parallelism (response time optimization)
  minimize network data transfer (throughput optimization)
- Basic Approaches to distributed query processing:

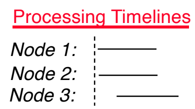
Pipelining – *functional decomposition*





Parallelism – *data decomposition*





Creating the Distributed Query Processing Plan

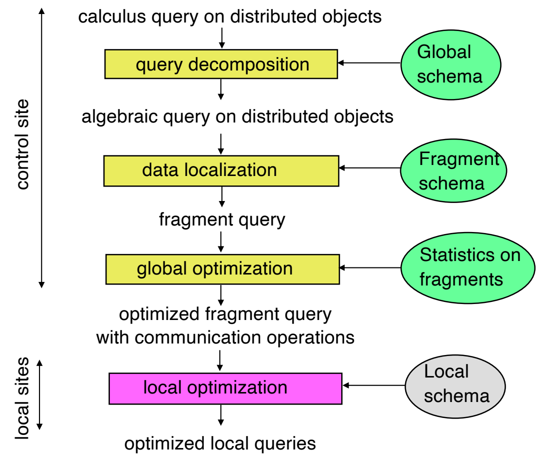
- Factors to be considered:
  - distribution of data
  - communication costs
  - lack of sufficient locally available information

- 4 processing steps:
  - (1) query decomposition
  - (2) data localization
  - (3) global optimization

  (1)–(3): control site (uses global information)

  - (4) local optimization — local sites (use local information)

Generic Layered Scheme for Planning a Dist. Query



This approach was initially designed for distributed relational DBMSs.

It also applies to distributed OODBMSs, Multi-DBMSs, and distributed Multi-DBMSs.

Distributed Query Optimization

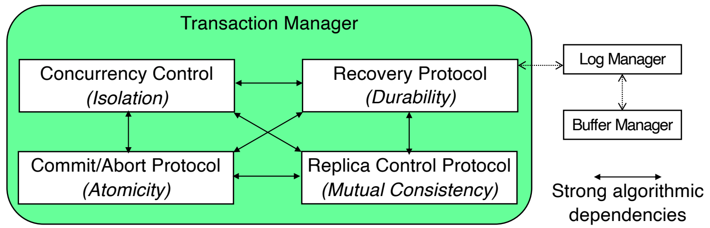
- information needed for optimization (fragment statistics):
  - size of objects, image sizes of attributes
  - transfer costs
  - workload among nodes
  - physical data layout
  - access path, indexes, clustering information
  - properties of the result (objects) - formulas for estimating the cardinalities of operation results
- execution cost is expressed as a weighted combination of I/O, CPU, and communication costs (mostly dominant).

$$\text{total-cost} = C_{CPU} * \#insts + C_{I/O} * \#I/Os + C_{MSG} * \#msgs + C_{TR} * \#bytes$$

Optimization Goals: response time of single transaction or system throughput

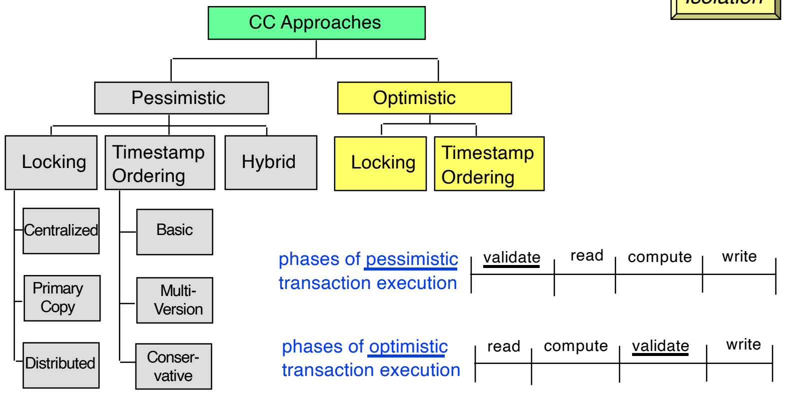
Distributed Transaction Managment

- Transaction Management (TM) in centralized DBS
  - Achieves transaction ACID properties by using:
    - concurrency control (CC)
    - recovery (logging)
- TM in DDBS
  - Achieves transaction ACID properties by using:



Classification of Concurrency Control Approaches

Isolation

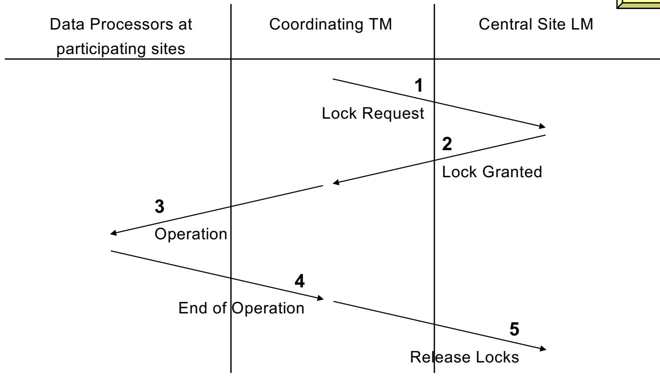


29

29

Communication Structure of Centralized 2PL

Isolation

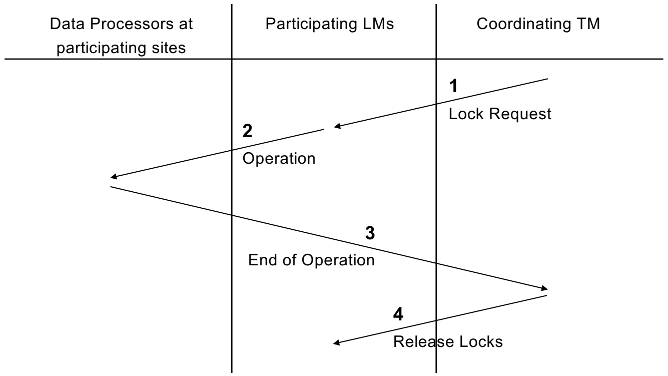


30

30

Communication Structure of Distributed 2PL

Isolation



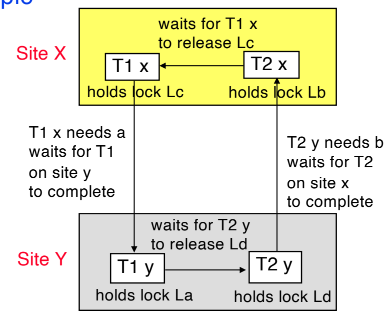
31

31

Distributed Deadlock Management

Isolation

Example



Distributed Waits-For Graph

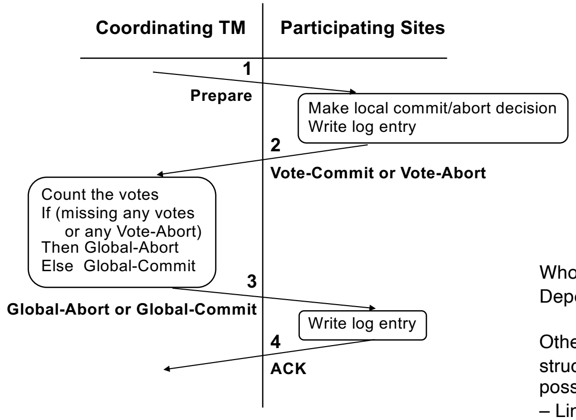
- requires many messages to update lock status
- combine the “wait-for” tracing message with lock status update messages
- still an expensive algorithm

32

32

Communication Structure of Centralized 2P Commit Protocol

Atomicity

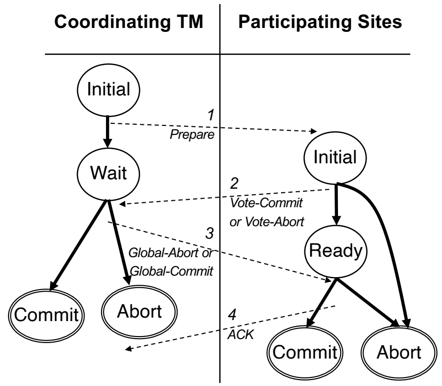


Who participates?
Depends on the CC alg.

Other communication structures are possible:
– Linear
– Distributed

State Transitions for 2P Commit Protocol

Atomicity



Advantages:
- Preserves atomicity
- All processes are "*synchronous within one state transition*"

Disadvantages:
- Many, many messages
- If failure occurs, *the 2PC protocol blocks*!

Attempted solutions for the blocking problem:
1) Termination Protocol
2) 3-Phase Commit
3) Quorum 3P Commit

Failures in a Distributed System

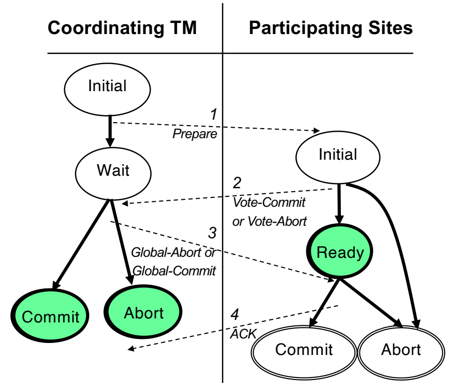
Types of Failure:
- Transaction failure
- Node failure
- Media failure
- Network failure
  - Partitions each containing 1 or more sites

Who addresses the problem?

Issues to be addressed:
- How to continue service → *Termination Protocols*
- How to maintain ACID properties while providing continued service → *Modified Concurrency Control & Commit/Abort Protocols*
- How to ensure ACID properties after recovery from the failure(s) → *Recovery Protocols, Termination Protocols, & Replica Control Protocols*

Termination Protocols

Atomicity, Consistency



Use timeouts to detect potential failures that could block protocol progress

Timeout states:
Coordinator: wait, commit, abort
Participant: initial, ready

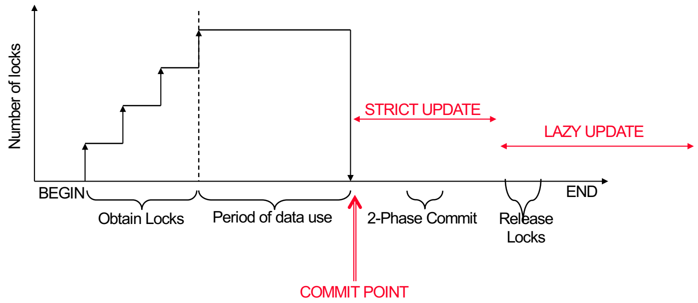
Coordinator Termination Protocol:
Wait – Send global-abort
Commit or Abort – BLOCKED!

Participant Termination Protocol:
Ready – Query the coordinator
If timeout
then query other participants;
If global-abort | global-commit
then proceed and terminate
else BLOCKED!

Replica Control Protocols

Consistency

- Update propagation of committed write operations



Strict Replica Control Protocol

Consistency

- Read-One-Write-All (ROWA)
- Part of the Concurrency Control Protocol and the 2-Phase Commit Protocol
  - CC locks all copies
  - 2PC propagates the updated values with 2PC messages (or an update propagation phase is inserted between the wait and commit states for those nodes holding an updateable value).

Lazy Replica Control Protocol

Consistency

- Propagates updates from a primary node.
- Concurrency Control algorithm locks the primary copy node (same node as the primary lock node).
- To preserve single copy semantics, must ensure that a transaction reads a current copy.
  - Changes the CC algorithm for read-locks
  - Adds an extra communication cost for reading data
- Extended transaction models may not require single copy semantics.

Recovery in Distributed Systems

Atomicity, Durability

Select COMMIT or ABORT (or blocked) for each interrupted subtransaction

Commit Approaches:

Redo – use the undo/redo log to perform all the write operations again

Retry – use the transaction log to redo the entire subtransaction (R + W)

Abort Approaches:

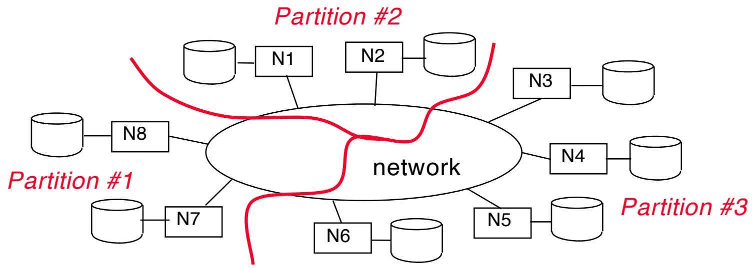
Undo – use the undo/redo log to backout all the writes that were actually performed

Compensation – use the transaction log to select and execute ”reverse” subtransactions that semantically undo the write operations.

Implementation requires knowledge of:

- Buffer manager algorithms for writing updated data from volatile storage buffers to persistent storage
- Concurrency Control Algorithm
- Commit/Abort Protocols
- Replica Control Protocol

Network Partitions in Distributed Systems



Issues:

- ✓ Termination of interrupted transactions
- ✓ Partition integration upon recovery from a network failure
- – Data availability while failure is ongoing