12.1 Distributed Database Systems
Why Distributed Databases?

- The decentralization of business functions
- Advances in computer networking
- The database is stored on several distributed systems (sites)
- Each site is able to process local transactions and participate in global transactions
- Network topology
  - long-haul vs. local-area network (WAN, LAN)
Advantages of Data Distribution

- Data sharing with local autonomy
- Reliability and availability
  - The failure of a site does not necessarily imply the shutdown of the system
  - crucial for real-time systems
- Downsizing trend
- Speedup of query processing
- Parallel query execution
Developing a Distributed Database

- **Bottom-up Integration**
  - ‘Islands of information’
  - Heterogeneous hardware and software
  - Heterogeneous distributed databases

- **Top-down Distribution**
  - Homogenous environment
  - From single-site database to distributed database
Autonomy

- The degree to which sites may operate independently
- Distribution of control
- Tight coupling
- Semi-autonomy
- Total isolation
Distribution Transparency

- The extent to which the distribution of data to sites is shielded from users and applications

- Data distribution strategies:
  - Basic distribution tables
  - Extracted tables
  - Snapshot tables
  - Replicated tables
  - Fragmented tables
    - Horizontal fragmentation
      - Primary
      - Derived
    - Vertical fragmentation
    - Mixed fragmentation
... Distribution Transparency

- **Location(Network) transparency**
  - hiding the details of the distribution of data in the network

- **Fragmentation transparency**
  - When a data item is requested, the specific fragment need not be named

- **Replication transparency**
  - A user should not know how a data item is replicated

- **Naming and local autonomy**
  - two solutions in distributed databases:
    - a central name server
    - each site prefix its own site identifier to any name it generates: site5.deposit.f3.r2
Directory Management

- Metadata (catalog) can be stored:
  - Centrally
    - Master site
  - Distributively
    - Maximize autonomy
  - Fully replicated

- Metadata caching
  - Distributed approach with caching of metadata from remote sites
Directory Management

- How to ensure that the cached metadata is up-to-date?
  - Owner responsibility
  - Versions
- Dependency tracking
  - Optimized execution plans
Distributed Concurrency Control

- **Nonreplicated Scheme**
  - Each site maintains a local lock manager to administer lock and unlock requests for local data
  - Deadlock handling is more complex

- **Single-Coordinator Approach**
  - The system maintains a single lock manager that resides in a single chosen site
  - Can be used with replicated data
  - **Advantages**
    - simple implementation
    - simple deadlock handling
  - **Disadvantages**
    - bottleneck
    - vulnerability
Distributed Concurrency Control

- **Majority Protocol**
  - A lock manager at each site
  - When a transaction wishes to lock a data item Q, which is replicated in n different sites, it must send a lock request to more than half of the n sites in which Q is stored
  - complex to implement
  - difficult to handle deadlocks
Biased Protocol

- A variation of the majority protocol
- When a transaction needs a shared lock on data item Q, it requests the lock from the lock manager at one site containing a replica of Q
- When a transaction needs an exclusive lock, it requests the lock from the lock manager at all sites containing a replica of Q

Primary Copy

- A replica is selected as the primary copy
- Locks are requested from the primary site
Deadlock Handling

- Local waits-for graph (WFG) is not enough
- Centralized site
- Distributed deadlock avoidance
  - A unique priority is assigned to each transaction (time stamp + site number)
  - Wait-die: if priority \((T_i)\) > priority \((T_j)\) then \(T_i\) waits, else \(T_j\) dies
Deadlock Handling

- Decentralized deadlock detection
  - Path pushing technique
    Periodically each site analyzes its local WFG and lists all paths. For each path $T_i \rightarrow \ldots \rightarrow T_j$, it sends the path information to every site where $T_j$ might be blocked because of a lock wait.
  
    - WFG1: $T_1 \rightarrow T_2 \rightarrow T_3$ and $T_3$ has made a RPC to site 2
    - WFG2: $T_3 \rightarrow T_4$ and $T_4$ has made a RPC to site 3
    - WFG3: $T_4 \rightarrow T_2$
Distributed Transaction Management

- More complex, since several sites may participate in executing a transaction

- Transaction manager
  - maintaining a log for recovery purposes
  - concurrency control

- Transaction coordinator
  - Starting the execution of the transaction
  - Breaking the execution into a number of subtransactions
  - Coordinating the termination of the transaction
Additional failures
- site failure
- link failure
- loss of message
- network partition

Atomicity of transactions
- All participating sites either commit or abort

Protocols to ensure atomicity
- Two-phase commit
- Three-phase commit
Two-phase Commit Protocol

- Let $T$ be a transaction initiated at site $S_i$ with coordinator $C_i$. When all the sites at which $T$ has executed inform $C_i$ that $T$ has completed, then $C_i$ starts the two-phase commit:

  - **Phase 1**
    - $C_i$ adds the record $\text{<prepare } T>\text{ to the log}$
    - sends a prepare $T$ message to all sites at which $T$ executed
    - if a sit answers no, it adds $\text{<abort } T>\text{ to its log and then responds by sending an abort } T\text{ message to } C_i$
    - if a sit answers yes, it adds $\text{<ready } T>\text{ to its log and then responds by sending a ready } T\text{ message to } C_i$

  - **Phase 2**
    - if $C_i$ receives a ready $T$ message from all the participating sites, $T$ can be committed, $C_i$ adds $\text{<commit } T>\text{ to the log and sends a commit } T\text{ message to all participating sites to be added to their logs}$
    - if $C_i$ receives an abort $T$ message or when a prespecified interval of time has elapsed, it adds $\text{<abort } T>\text{ to the log and then sends an abort } T\text{ message}$
Handling of Failures

- Failure of a participating site $S_k$
  When the site recovers it examines its log and:
  - If the log contains a `<commit T>` record, $S_k$ executes $redo(T)$
  - If the log contains a `<abort T>` record, $S_k$ executes $undo(T)$
  - If the log contains a `<ready T>` record, the site must consult $C_i$ to determine the fate of $T$. If $C_i$ is up, it notifies $S_k$ as to whether $T$ committed or aborted. If $C_i$ is down, $S_k$ must find the fate of $T$ from other sites by sending a query-status $T$ message to all the sites in the system
  - If the log has no control records concerning $T$, $S_k$ executes $undo(T)$
... Handling of Failures

- **Failure of the coordinator**
  - If an active site contains a `<commit T>` in its log, then T must be committed.
  - If an active site contains a `<abort T>` in its log, then T must be aborted.
  - If some active sites does not contain `<ready T>` in its log, then T must be aborted.
  - If non of the above cases hold, the active site must wait for C_i to recover (blocking problem).

- **Failure of a link**
  - same as before
Three-phase Commit Protocol

- The major disadvantage of the two-phase commit protocol is that coordinator failure may result in blocking
- The three-phase commit protocol is designed to avoid the possibility of blocking
Distributed Query Optimization

- Other factors of distributed query processing
  - Cost of data transmission
  - Potential gain from parallel processing
  - Relative speed of processing at each site

- Join strategies
  - Ship one operand
  - Ship both operands
  - Semijoin
  - Distributed nested loop

- Fragment access optimization
Distributed Integrity Constraints

- Fragment integrity
- Distributed integrity constraints
Truly Distributed Capabilities

- Code’s twelve rules
  - Local autonomy
  - No reliance on a central site for any particular service
  - Continues operation
  - Location independence
  - Fragmentation independence
  - Replication independence
  - Distributed query processing
  - Distributed transaction management
  - Hardware independence
  - Operating system independence
  - Network independence
  - DBMS independence
Distributed Capabilities of Oracle 8

- Transparent multisite distributed query
- Multisite update with transparent two-phase commit
- Full location transparency and site autonomy
- Global database names
- Automatic table replication
12.2 Data Replication
Data Replication

- Simpler form of a distributed database system
- Why?
- Every DBMS vendor has a replication solution
- Gaining popularity
Data Replication Concepts

- Synchronous vs. asynchronous replication
- Object replication
- Data ownership
  - Master/slave ownership
    - publish-and-subscribe metaphor
  - Workflow ownership
    - at any moment there is only one site that can update
  - Update-anywhere (symmetric replication) ownership
    - can lead to conflict
    - conflict resolution
Data Replication Concepts

- **Table snapshots**
  - Create SNAPSHOT local_staff
    REFRESH FAST (COMPLETE, FORCE)
    START WITH sysdate NEXT sysdate + 7
    AS SELECT * FROM Staff@Staff_Master_Site WHERE bno='B5'
  - Can use the transaction log
  - Refresh groups

- **Database triggers**
  - Create TRIGGER staff_after_ins_row
    BEFORE INSERT ON Staff
    FOR EACH ROW
    BEGIN
      INSERT INTO Staff_Duplicate@Staff_Duplicate_Link
      VALUES (...)
    END
  - Several drawbacks
Data Replication Concepts

- Conflict detection
  - Each site sends the before-and-after images to a replication server
  - Violation of referential integrity

- Conflict resolution
  - Earliest and latest timestamp
  - Site priority
  - Additive and average updates
  - Minimum and maximum values
  - User-defined
  - Hold for manual resolution
ORACLE8 Replication

- Read-only Replication
  - Snapshot log

- Replication Administrators
  - Setting up and implementing replication

- Data Synchronization
  - Automatic
  - Manual
  - Two-phase commit (2PC) through deferred transaction queue
  - User defined conflict resolution

- Replication Manager