Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# Chapter 11 - Data Replication Middleware



## Motivation

- Replication: controlled redundancy of data to
  - improve performance (query response time)
  - increase availability
- Replication is a common concept in
  - (homogeneous) distributed DBMS
  - centralized DBMS in the form of materialized views
  - mobile DBMS environments
  - data/information integration middleware

# Challenges

- Integration of replication with transaction processing
- Enforcing consistency of replica in the presence of updates
  - possible model: update of replicated data becomes a distributed transaction that updates all copies
- Deciding where to keep copies, and how many
- Detecting and resolving conflicts
  - reconciling "versions" of replicated data elements that have evolved independently (e.g., because of a communication failure)



# Replication Middleware

- Source table data is replicated to a target table
- Scenarios and uses
  - data distribution
    - data from one source table is replicated to more than one (read-only) target table
  - data consolidation
    - data from more than one source table is replicated to a single target table (union view)
  - bidirectional replication allows updates on target tables to be replicated back to the source table
    - master/slave replication: all updates flow back to a designated master, are then distributed to other targets
    - peer-to-peer replication: each location exchanges data with all other locations
  - combination of the above
  - multi-tier replication
    - introduction of staging area(s)
      - changes are copied to another system
      - then copied from staging are to multiple targets
    - minimizes impact on source systems

# Eager Replication (Synchronous Replication)

- Transaction synchronizes with copies of replicated elements before commit
  - guarantees globally serializable execution
    - locking
  - avoids inconsistencies
- Potential problems
  - deadlocks
  - update overhead
    - reduced update performance
    - increased transaction response time
  - lack of scalability
  - cannot be used if nodes are disconnected (e.g., mobile databases)



# Lazy Replication (Asynchronous Replication)

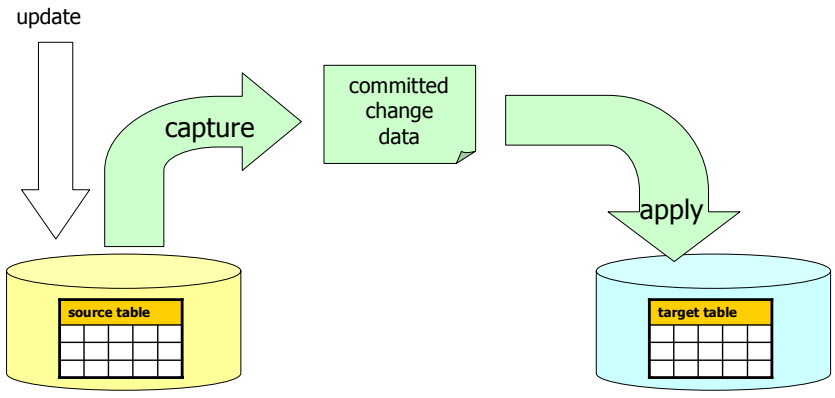
- Changes introduced at one site are propagated (as separate transactions) to other sites only after commit
  - minimal overhead
  - works also if sites are not connected (e.g., mobile environments) or temporarily unavailable
- Potential problems
  - inconsistencies between the copies
    - multi-version concept for detecting inconsistencies
    - reconcile conflicting transactions
      - rollback of (committed) transactions not possible
  - ⇨ potential for "system delusion" (Gray, Reuter)
    - inconsistent database, with no obvious way to repair it

# Replication methods

- Target table refresh
  - at intervals, target table is replaced by a fresh copy of the source table
  - no requirement to capture individual changes
  - only makes sense is target table is small or replication occurs infrequently
- Change-capture replication
  - committed changes to source table are captured and replicated to the target table
  - replication activity
    - continuous (near-real-time)
    - interval-based (e.g., during off-peak hours)
    - triggered by DB-events
    - one-time snapshot



# Change-Capture Replication



# Capturing Changes

- Source table registration
  - which parts of the table changes should be captured
  - when replication should occur
- Capture logic
  - responsible for detecting the changes to the source table
  - make committed change data available to the apply logic
  - realization approaches
    - capture program analyzes the database log files
    - use database triggers
- Committed Change Data
  - needs to (at least) include
    - new values of updated data items, plus data item identifier (keys)
    - (optional) before-image information
  - can be provided using
    - (relational) staging table at the source location
      - each change is reflected as a row in the staging table
    - message-oriented middleware
      - changes are provided as message on a queue



# Applying Changes

- Apply logic is responsible for
  - initializing target table from source table
  - applying captured changes to the target table
    - preserve order of dependent transactions
- Needs access to
  - the captured changes stored in staging tables or change message queues
  - the target table
  - (the source table)
- Enhanced capabilities
  - filtering, joins, aggregation, transformation of data for the target

# Replication Conflicts

- Two nodes may concurrently update replicas of the same object
  - "race" each other to propagate the update to all the other nodes
  - potential for lost updates
- Detecting conflicts
  - usually based on timestamps (or before-image data)
    - object carries timestamp of most recent update
    - replica update carries new value and old object timestamp
  - each node compares old object timestamp of incoming replica updates with its own object timestamp
    - if timestamps are the same, then the update is accepted
    - if not, then the incoming update transaction is rejected, submitted for reconciliation
      - rollback of transaction is not possible anymore, has been committed at the originating site
  - wait situations in eager replication <-> reconciliation in lazy replication



# Ownership of Replicas

- Group
  - "update anywhere" update model
  - any node/site with a replica can update it
    - may cause conflicts, need for reconciliation
- Master
  - "primary copy" update model
  - each object has a master node
  - only the master can update the primary copy
    - other replicas are read-only
    - other nodes request the master node to perform an update (e.g., using RPC)
  - eliminates reconciliation failures with lazy replication
  - does not work for mobile, disconnected databases

# Reconciliation

- Approaches
  - automatically, based on rules
    - site, time or value priority, merging of updates, ...
  - manually
    - conflict situations are reported in a conflicts table or queue
- Non-transactional replication schemes
  - abandon serializability for convergence
    - all nodes converge to the same replicated state, which may not correspond to a serial transaction execution
  - tolerate lost updates



# Alternatives for Conflict Detection, Avoidance

- Semantic synchronization
  - permit commutative transactions
    - requires capturing update semantics at a logical level
    - performing the transaction update may yield different results, but still be semantically correct
      - example: processing checks at a bank
  - provide acceptance criteria for detecting conflicts
    - incoming replica transaction updates are tentatively accepted and performed, but needs to pass the acceptance test
    - replaces/augments the generic conflict detection mechanisms
- Avoid conflicts by implementing update strategies in the application
  - fragmentation by key
    - a site can update only rows whose key are in a fixed range
    - no range overlaps
  - fragmentation by time
    - disjoint "time windows" for each site to perform updates

# Summary

- Replication middleware uses
  - data distribution and consolidation
  - improve performance, availability
  - information integration
- Replication middleware architecture
  - capture and apply
  - committed change data
- Change propagation and ownership strategies
  - eager vs. lazy
  - group vs. master
- Conflict detection and reconciliation approaches are required for lazy group replication!