Last Class: Canonical Problems

- Distributed synchronization and mutual exclusion
- Distributed Transactions

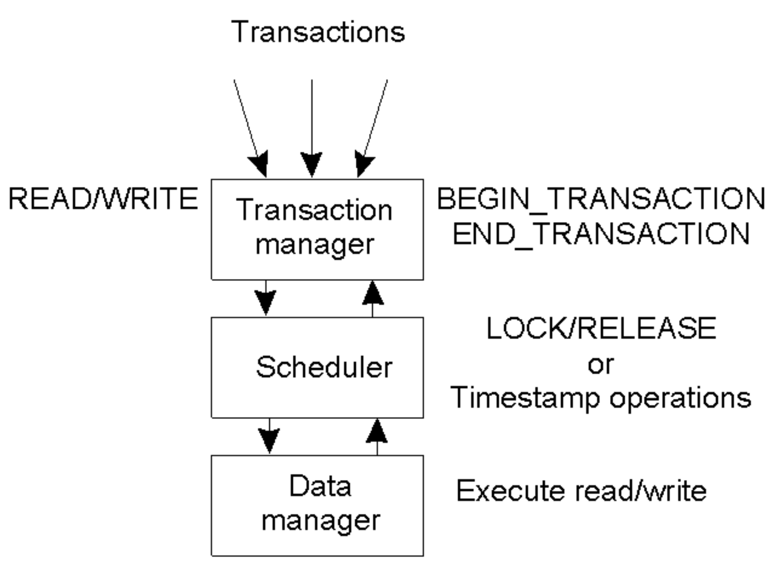
Today: Concurrency Control

- Concurrency control
  - Two phase locks
  - Time stamps
- Intro to Replication and Consistency
- Thoughts on the mid-term

Concurrency Control

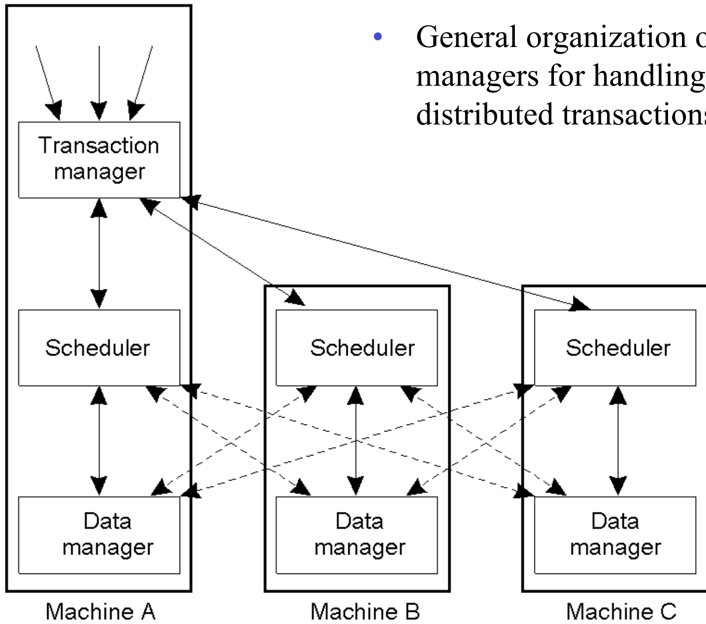
- Goal: Allow several transactions to be executing simultaneously such that
  - Collection of manipulated data item is left in a consistent state
- Achieve consistency by ensuring data items are accessed in an specific order
  - Final result should be same as if each transaction ran sequentially
- Concurrency control can implemented in a *layered* fashion

Concurrency Control Implementation



- General organization of managers for handling transactions.

Distributed Concurrency Control



- General organization of managers for handling distributed transactions.

Serializability

BEGIN_TRANSACTION
x = 0;
x = x + 1;
END_TRANSACTION

(a)

BEGIN_TRANSACTION
x = 0;
x = x + 2;
END_TRANSACTION

(b)

BEGIN_TRANSACTION
x = 0;
x = x + 3;
END_TRANSACTION

(c)

Schedule 1	x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3	Legal
Schedule 2	x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3;	Legal
Schedule 3	x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3;	Illegal

- Key idea: properly schedule conflicting operations
- Conflict possible if at least one operation is write
  - Read-write conflict
  - Write-write conflict

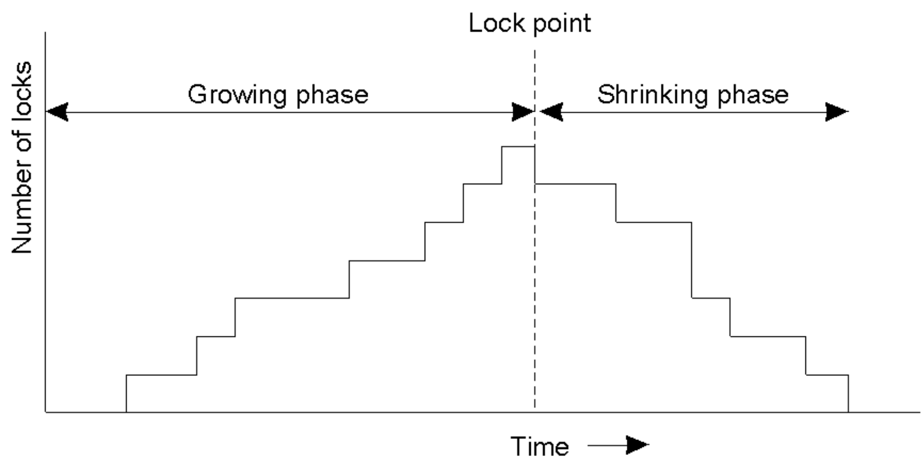
Optimistic Concurrency Control

- Transaction does what it wants and *validates* changes prior to commit
  - Check if files/objects have been changed by committed transactions since they were opened
  - Insight: conflicts are rare, so works well most of the time
- Works well with private workspaces
- Advantage:
  - Deadlock free
  - Maximum parallelism
- Disadvantage:
  - Rerun transaction if aborts
  - Probability of conflict rises substantially at high loads
- Not used widely

Two-phase Locking

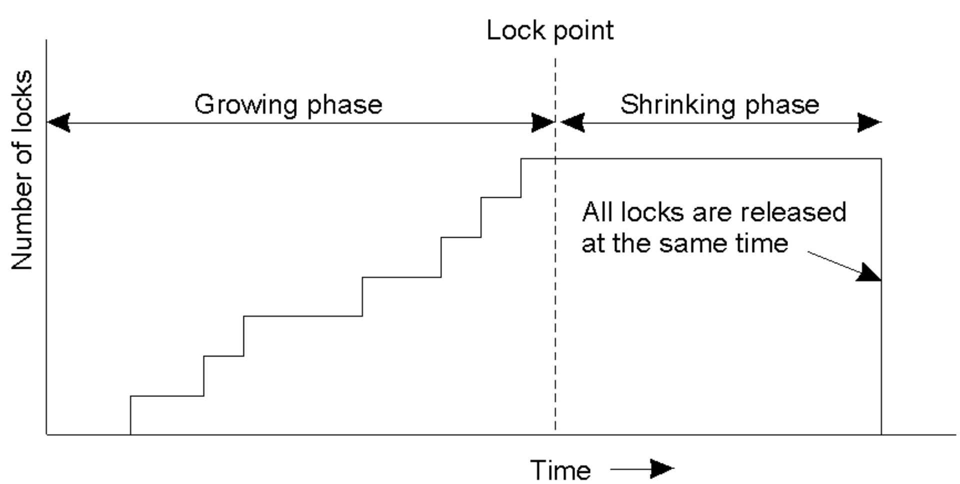
- Widely used concurrency control technique
- Scheduler acquires all necessary locks in growing phase, releases locks in shrinking phase
  - Check if operation on *data item x* conflicts with existing locks
    - If so, delay transaction. If not, grant a lock on *x*
  - Never release a lock until data manager finishes operation on *x*
  - One a lock is released, no further locks can be granted
- Problem: deadlock possible
  - Example: acquiring two locks in different order
- Distributed 2PL versus centralized 2PL

Two-Phase Locking



- Two-phase locking.

Strict Two-Phase Locking



- Strict two-phase locking.

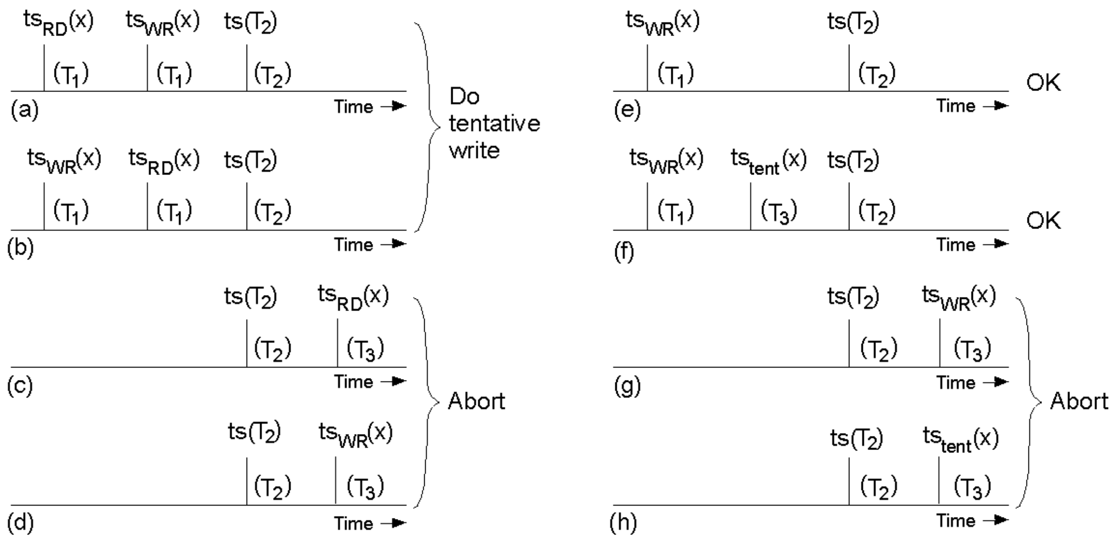
Timestamp-based Concurrency Control

- Each transaction Ti is given timestamp ts(Ti)
- If Ti wants to do an operation that conflicts with Tj
  - Abort Ti if *ts(Ti)* < *ts(Tj)*
- When a transaction aborts, it must restart with a new (larger) time stamp
- Two values for each data item *x*
  - *Max-rts(x):* max time stamp of a transaction that read *x*
  - *Max-wts(x):* max time stamp of a transaction that wrote *x*

Reads and Writes using Timestamps

- $Read_i(x)$
  - If $ts(T_i) < max\text{-}wts(x)$ then Abort T_i
  - Else
    - Perform $R_i(x)$
    - $Max\text{-}rts(x) = \max(max\text{-}rts(x), ts(T_i))$
- $Write_i(x)$
  - If $ts(T_i) < max\text{-}rts(x)$ or $ts(T_i) < max\text{-}wts(x)$ then Abort T_i
  - Else
    - Perform $W_i(x)$
    - $Max\text{-}wts(x) = ts(T_i)$

Pessimistic Timestamp Ordering

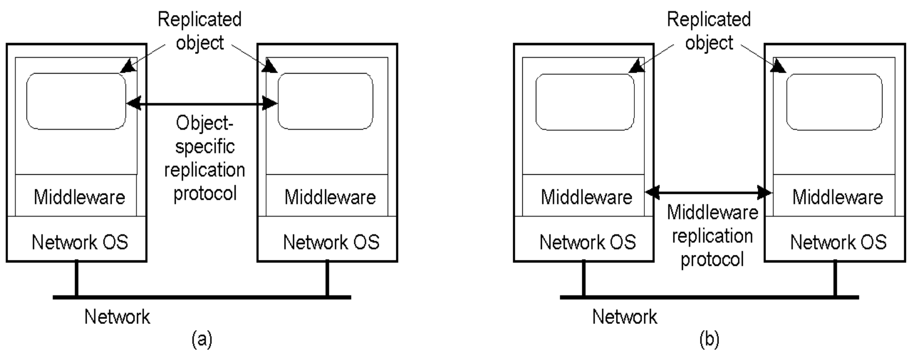


Replication

- Data replication: common technique in distributed systems
- Reliability
  - If one replica is unavailable or crashes, use another
  - Protect against corrupted data
- Performance
  - Scale with size of the distributed system (replicated web servers)
  - Scale in geographically distributed systems (web proxies)

- Key issue: need to maintain *consistency* of replicated data
  - If one copy is modified, others become inconsistent

Object Replication



- Approach 1: application is responsible for replication
  - Application needs to handle consistency issues
- Approach 2: system (middleware) handles replication
  - Consistency issues are handled by the middleware
  - Simplifies application development but makes object-specific solutions harder

Replication and Scaling

- Replication and caching used for system scalability
- Multiple copies:
  - Improves performance by reducing access latency
  - But higher network overheads of maintaining consistency
  - Example: object is replicated *N* times
    - Read frequency *R*, write frequency *W*
    - If $R<<W$, high consistency overhead and wasted messages
    - Consistency maintenance is itself an issue
      - What semantics to provide?
      - Tight consistency requires globally synchronized clocks!
- Solution: loosen consistency requirements
  - Variety of consistency semantics possible

Mid-term Exam Comments

- Closed book, closed notes, 90 min
- Lectures 1-13 included on the test
  - Focus on things taught in class (lectures, in-class discussions)
  - Start with lecture notes, read corresponding sections from text
  - Supplementary readings (key concepts) included on the test.
- Exam structure: few short answer questions, mix of subjective and “design” questions

- Good luck!