Transaction Processing

Anne Denton

Department of Computer Science
North Dakota State University

Outline

Table of Contents

Schedules

- Schedules specify order of execution of reads and writes of multiple transactions
  - Conflict serializability
    - Tests if schedule equivalent (“conflict equivalent") to a serial schedule
  - Recoverability
    - Tests if schedule allows recovery from failure
- Both are mathematical concepts with definitions that we will discuss

Read and write

- Both read and write are multi-step operations
  - Makes DBMS implementations more complex than combination of read and write may appear
- Steps in `read(X)`
  - Find the disk block that contains X
  - Copy disk block into a memory buffer
  - Copy X from the memory buffer into the program
- Steps in `write(X)`
  - Find the disk block that contains X
  - Copy the disk block into a memory buffer
  - Copy the variable X from a program into the buffer
  - Write the disk block back to disk

Conflict between operations

- Two operations are in conflict if they
  - Belong to different transactions
  - Access the same item X
  - At least one of them is a write statement
- Two reads are not in conflict
  - Read-only databases do not require concurrency control
  - Read-only transactions may conflict with other transactions in general but not with other read-only transactions
- If as one of the operations is a write, it is in conflict with any other (read or write), provided the other operation applies to the same item and is in a different transaction

Question 1 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(Y)
Write(Y)	

Which of the following is true?

1. Read(X) in T1 is in conflict with Read(X) in T2
2. Write(X) in T1 is in conflict with Read(X) in T2
3. Read(X) in T1 is in conflict with Write(X) in T1
4. Write(Y) in T1 is in conflict with Read(X) in T2
5. Write(Y) in T1 is in conflict with Write(Y) in T2

Conflict serializability

- Interleaved execution of transactions lead to the same result as execution of the same transactions in sequence
- Different meaning from the isolation level serializable
- A schedule is serial if no interleaving happens
  - Generally unacceptable in multiuser systems
- Definition of conflict serializability
  - A schedule is conflict serializable if it is conflict equivalent to a serial schedule
  - Schedules are conflict equivalent if the order of any two conflicting operations is the same in both schedules

Precedence (Serializability) Graph

- Conflicts can be stated in Precedence graph (serializability graph)
- Contains a node for each uncommitted transaction in S
  - An arc from T_i to T_j is placed if an action of T_i precedes and conflicts with one of T_j's actions
- A schedule S is conflict serializable if and only if its precedence graph is acyclic
  - An equivalent serial schedule is then given by topological sort

Recoverability

- A schedule is nonrecoverable if we would need a rollback for committed transactions
- Condition for Recoverability
  - A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written some item X that T reads have committed, i.e., transaction T can only commit after all those transactions T' it has read from have committed
  - A transaction T reads from transaction T' in a schedule S if some item X is first written by T' and later read by T
  - In addition, T' should not have been aborted before T reads item X, and there should be no transactions that write X after T' writes it and before T reads it (unless those transactions, if any, have aborted before T reads X)

Question 2 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
	Read(X)
	Write(X)
Write(X)	
COMMIT	
	COMMIT

Is this schedule

1. Conflict serializable
2. Recoverable

Question 3 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Read(Y)
Read(Y)	
Write(Y)	
	COMMIT
ROLLBACK	

Is this schedule

1. Conflict serializable
2. Recoverable

Cascading rollback

- A cascading rollback means that a failure of one transaction results in rollback of other not yet committed ones
  - The recovery process is much faster if cascading rollback is guaranteed not to occur
  - A schedule avoids cascading rollback if a transaction can only read values that were written by a committed transaction
- In strict schedules, transactions can neither read nor write X until the last transaction that wrote X has committed
  - The recovery process is simplest for strict schedules: Undoing a write(X) can be done by restoring X to the value before the write

Question 4 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
Write(X)	
	Read(X)
Read(Y)	
Write(Y)	
ROLLBACK	

Is this schedule

1. Conflict serializable
2. Recoverable
3. Vulnerable to cascading rollback

Table of Contents

Locks

- Locks are variables associated with (each) data item in a database
  - They describe which operations are allowed at a given time
  - Lock / unlock operations on the data item have to be indivisible, i.e. no interleaving
  - Use semaphores of operating system
- Binary locks have two states
  - Locked (1): Data item cannot be accessed
  - Unlocked (0)
- A queue of waiting processes is associated with a lock
- Usually records are only kept for those items that are locked

Multi-modal locks

- Binary locks are not flexible enough for databases
  - A read access doesn't cause problems as long as it is executed concurrently with other read operations only
  - A write access cannot not be executed concurrently with either read or write
- Multimode locks are used

  Shared lock: Allows read access

  Exclusive lock: Allows read and write access
- Five operations can be done on locks:
  - `read_lock(X)` issues a shared lock if no lock present
  - `write_lock(X)` issues an exclusive lock if no lock present
  - `unlock(X)` removes any lock
  - `write_lock(X)` upgrades a shared lock if that was set
  - `read_lock(X)` downgrades an exclusive lock if that was set

Two Phase Locking protocol (2PL)

- Locking alone doesn't guarantee conflict serializability.
- Consider the lost update problem
  - If locking is done before each read and write and ended afterwards nothing is gained
- Two phase locking protocol says that for one transaction there are two phases
  - A growing phase, where locks may be issued and upgraded
  - A shrinking phase, where locks may be released and downgraded
- Properties of 2PL
  - If every transaction follows the 2PL protocol the schedule is guaranteed to be conflict serializable, i.e., ensures that precedence graph is acyclic
  - 2PL allows cascading rollback

Strict 2-Phase-Locking

- In strict 2PL issued locks are kept until the transaction is committed
  - The whole transaction happens within the expanding phase of 2PL
- Strict 2PL leads to strict schedules, i.e., transactions can neither read nor write X until the last transaction that wrote X has committed
- Precedence graph is acyclic
- Deadlock can happen
- Most DBMSs use Strict 2PL

Question 5 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Read(Y)
Read(Y)	
Write(Y)	
	COMMIT
ROLLBACK	

Is this schedule

1. Consistent with 2-Phase Locking
2. Consistent with Strict 2-Phase Locking

Question 6 (Multiple answers can be correct)

Consider the following schedule

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(Y)
ROLLBACK	

Is this schedule

1. Consistent with 2-Phase Locking
2. Consistent with Strict 2-Phase Locking

Deadlock

- Deadlock is a situation where neither of two transactions can proceed because the lock they require is held by the respective other transaction:

Read_lock(Y)	
Read(Y)	
	Read_lock(X)
	Read(X)
Write_lock(X)	
	Write_lock(Y)

Deadlock detection

- Deadlock detection strategies include timeout and detection of cycles in the wait-for-graph
- Timeout
  - System detects if a transaction has to wait longer for a lock than would be expected under normal circumstances
  - Transaction is aborted whether or not a deadlock existed
  - Advantage: simplicity
  - Much used in practice
- Wait-for graph
  - Create wait-for graph of transactions waiting for each other
  - Detect cycles, e.g., T1 waits for T2 to end, T2 waits for T3 to end, T3 waits for T1 to end
  - If cycles are detected the transactions are aborted

Question 7 (Multiple answers can be correct)

Deadlock prevention uses a

1. Wait-for graph
2. Precedence graph

Question 8 (Multiple answers can be correct)

The wait-for-graph is typically larger than the precedence graph

1. True
2. False

Deadlock prevention

- Deadlock prevention algorithms introduce priority, typically timestamp
- A transaction with higher priority never waits for one with lower priority
- Two strategies (assuming T1 requests a lock that T2 has)
  - Wait-die: If T1 has higher priority it waits, otherwise it aborts
  - Wound-wait: If T1 has higher priority, T2 is aborted, otherwise T1 waits
  - Priority (timestamp) induces order, hence no cycles

Table of Contents

Optimistic concurrency control (Validation)

- Phases in optimistic concurrency control

Read:	Writes into private work space
Validation:	Checks if there is a possible conflict
Write:	If no conflict possible write to database

- Implementation
  - Transactions receive time stamp
  - If transaction don't satisfy validation conditions, it is rolled back
- Performance
  - Works well for few conflicts
  - Otherwise: Restarting hurts performance (lost progress)
  - Problem: At most one transaction in validation/write phase

Multiversion concurrency control

- Also requires timestamps
  - A transaction T_i is only allowed to write O if $TS(T_i) > RTS(O)$
  - In that case a new version is created
  - Read and write timestamps set to the TS(Ti)
  - Otherwise T_i is aborted

  Benefits: Transactions that read-only are never blocked

  Drawbacks: Overhead of timestamps and multiple versions
- Many current systems use a limited snapshot-based type if multiversion concurrency control for read-only transactions, in which only one old version is kept

Question 9 (Multiple answers can be correct)

Which of the following is best for workloads in which many transactions involve updates

1. Lock-based concurrency control
2. Optimistic concurrency control
3. Multi-version concurrency control

Question 10 (Multiple answers can be correct)

Which of the following can be combined with Lock-based concurrency control for read-heavy workloads

1. Optimistic concurrency control
2. Multi-version concurrency control

Table of Contents

- Two characteristics of recovery algorithms
  - Steal: A buffer pool can be written back to disk if a transaction has not committed
  - Force: A committed transaction is written to disk immediately
- Simplest assumption
  - No steal: Avoids undoing changes of aborted transactions
  - Force: Avoids redoing of committed transactions
- Problems
  - No steal: Assumes modifications of ongoing transactions fit in buffer pool
  - Force: Leads to many unnecessary writes if same page is modified
- Assumption of stable storage to which all logs are written

Question 11 (Multiple answers can be correct)

Which of the following properties of crash recovery algorithms results in a simpler implementation

1. Steal is easier to implement than no-steal
2. Force is easier to implement than no-force

ARIES Recovery Algorithm

- Choice of characteristics
  - Steal
  - No-force

Analysis: Identifies dirty pages in buffer pool and active transactions

Redo: Repeats all actions from appropriate point in log, and restores database at time of crash

Undo: Undoes uncommitted transactions (aborting)

- Uses write-ahead logging: Any change to database first recorded in log, and log written to stable storage
- Log is also kept during undo: Another failure could happen during recovery

Recovery-related Structures

- Transaction table
  - Entry for each active transaction
    - Transaction id
    - Last LSN (most recent log sequence number)
    - Status (in progress, committed, aborted)
- Dirty page table
  - Contains entry for each dirty page
  - Includes first log record that caused the page to become dirty

Checkpointing

- Checkpointing is standard solution for defining starting point for recovery
  - Snapshot of DBMS state
  - Ideally checkpoint holds complete DB state
- Problem: System can't be brought to a halt
- Solution: Fuzzy checkpoint