Transaction Management:

Theory of Serializability

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Some slides were adopted from Prof. Dan Suciu

Learning Objectives

After this lecture, you should be able to:

- Describe the theory of serializability, including Serial, Serializable, Conflict-Serializable, and View-Serializable schedules.
- Use the Precedence Graph Algorithm to test for conflict-serializability.
- Describe how commercial database systems use two-phase locking to guarantee serializability of concurrent transactions.

Schedules

- Schedule: A sequence of interleaved actions from a set of transactions, where the actions of each individual transaction are in the original order.

- Represents an actual sequence of database actions.

- In a *complete* schedule, each transaction ends in commit or abort.

- Initial State of DB + Schedule $\rightarrow$ Final State of DB

Example

A and B are elements in the database
t and s are variables in tx source code

T1	T2
READ(A, t)	READ(A, s)
t := t+100	s := s*2
WRITE(A, t)	WRITE(A,s)
READ(B, t)	READ(B,s)
t := t+100	s := s*2
WRITE(B,t)	WRITE(B,s)

Serial Schedules

Run transactions one at a time, in a series. (Different orders might give different results.)

T1	T2
READ(A, t)	
t := t+100	
WRITE(A, t)	
READ(B, t)	
t := t+100	
WRITE(B,t)	
	READ(A, s)
	s := s*2
	WRITE(A,s)
	READ(B,s)
	s := s*2
	WRITE(B,s)

Serializable schedules

A schedule is serializable if it is equivalent to a serial schedule.

Final state must be the same as the state produced by one of the serial schedules.

T1	T2
READ(A, t)	
t := t+100	
WRITE(A, t)	
	READ(A, s)
	s := s*2
	WRITE(A,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	
	READ(B,s)
	s := s*2
	WRITE(B,s)

Is this a Serializable schedule?

T1	T2
READ(A, t)	
t := t+100	
WRITE(A, t)	
	READ(A, s)
	s := s*2
	WRITE(A,s)
	READ(B,s)
	s := s*2
	WRITE(B,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	

No, it's a non-Serializable schedule

T1	T2
READ(A, t)	
t := t+100	
WRITE(A, t)	
	READ(A, s)
	s := s*2
	WRITE(A,s)
	READ(B,s)
	s := s*2
	WRITE(B,s)
READ(B, t)	
t := t+100	
WRITE(B,t)	

Outline

- ✓ Theory of Serializability
  - ✓ Serial and serializable schedules
  - Conflict-Serializable schedules
- Two-Phased Locking Theorem
  - Two-Phased Locking (2PL)
  - Strict two-phased locking (S2PL)

Conflicts

- Write-Read – WR
- Read-Write – RW
- Write-Write – WW

Conflict-Serializability

Definition. A schedule is ***conflict-serializable*** if it can be transformed into a serial schedule by a series of swappings of adjacent non-conflicting actions

a conflict-serializable schedule is a serializable schedule

- The converse is not true in general (see textbook page 893)

Conflict-serializability is a condition that the schedulers in commercial systems generally use when they need to guarantee serializability.

Conflict Serializability

Swapping Conflicts:

- Two actions by same transaction T_i: $\boxed{R_i(X), W_i(Y)}$

- Two writes by T_i, T_j to same element: $\boxed{W_i(X), W_j(X)}$

- Read/write by T_i, T_j to same element: $\boxed{W_i(X), R_j(X)}$ $\boxed{R_i(X), W_j(X)}$

Conflict-Serializability

Example:

$$r_1(A); w_1(A); r_2(A); \boxed{w_2(A); r_1(B);}\ w_1(B); r_2(B); w_2(B)$$

$$r_1(A); w_1(A); \boxed{r_2(A); r_1(B);}\ w_2(A); w_1(B); r_2(B); w_2(B)$$

$$r_1(A); w_1(A); r_1(B); r_2(A); \boxed{w_2(A); w_1(B)}; r_2(B); w_2(B)$$

$$r_1(A); w_1(A); r_1(B); \boxed{r_2(A); w_1(B);}\ w_2(A); r_2(B); w_2(B)$$

$$r_1(A); w_1(A); r_1(B); w_1(B); r_2(A); w_2(A); r_2(B); w_2(B)$$

Testing for Conflict-Serializability

Given a schedule S, we can construct a directed graph G=(V,E) called a precedence graph

- V : all transactions in S
- E : $T_i \rightarrow T_j$ whenever an action of T_i precedes and conflicts with an action of T_j in S (RW, WR, WW)

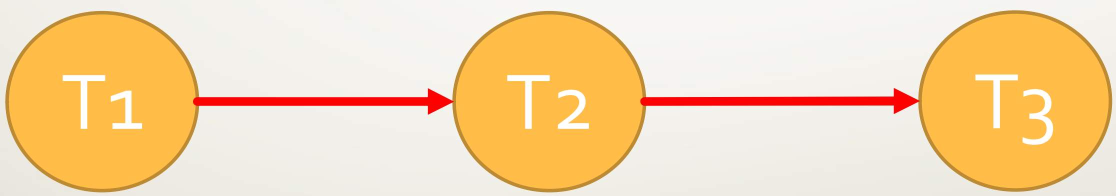
Theorem: A schedule S is **conflict serializable** iff its precedence graph <u>contains no cycles</u>

Example 1

$$\boxed{r_2(\mathrm{A});\, r_1(\mathrm{B});\, w_2(\mathrm{A});\, r_3(\mathrm{A});\, w_1(\mathrm{B});\, w_3(\mathrm{A});\, r_2(\mathrm{B});\, w_2(\mathrm{B})}$$

Example 1

$$r_2(\mathrm{A}); r_1(\mathrm{B}); w_2(\mathrm{A}); r_3(\mathrm{A}); w_1(\mathrm{B}); w_3(\mathrm{A}); r_2(\mathrm{B}); w_2(\mathrm{B})$$



This Schedule is conflict-serializable

Outline

- ✓ Theory of Serializability
  - ✓ Serial and serializable schedules
  - ✓ Conflict-Serializable schedules
- Two-Phased Locking Theorem
  - Two-Phased Locking (2PL)
  - Strict two-phased locking (S2PL)

Well-Formed, Two-Phased Transactions

A transaction is **well-formed** if it acquires at least a shared lock on Q before reading Q or an exclusive lock on Q before writing Q and doesn't release the lock until the action is performed

A transaction is **two-phased** if it never acquires a lock after unlocking one

- There are two phases:
  - a *growing phase* in which the transaction acquires locks
  - a *shrinking phase* in which locks are released

Two-Phased Locking Theorem

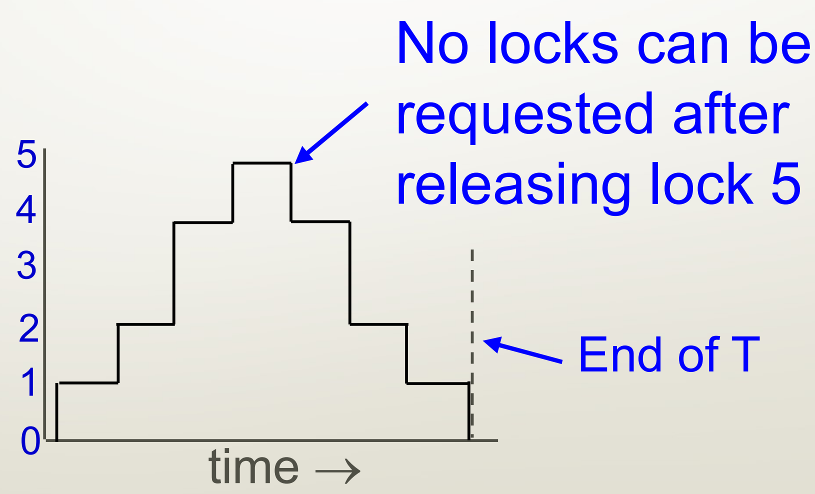
If all transactions are well-formed and two-phase, then any schedule in which conflicting locks are never granted ensures serializability

Two Phase Locking Protocol (2PL)

2PL is a way of managing locks during a transaction T

- T gets (S and X) locks gradually, as needed
- T cannot request any additional locks once it releases any locks

# of locks held by a transaction T



Can this schedule arise under 2PL?

T1:	R(A)	W(A)			R(B)	W(B)		
T2:			R(A)	W(A)			R(B)	W(B)

Enforce 2PL

T1:	R(A),				R(C)		W(B)	
T2:		R(B)		W(C)		R(B)		
T3:			R(A)					R(B)

S1(A); R1(A); S2(B) R2(B); S3(A); R3(A);
X2(C); W2(C); REL2(C); S1(C); R1(C);
R2(B); REL2(B); X1(B); W1(B); REL1(A,B,C);
S3(B); R3(B); REL3(A,B);

A problem with 2PL

T1:	R(A) W(A)			R(B) W(B)			
T2:		R(A)	W(A)		R(B)	W(B)	

Unrecoverable schedule!

T1	T2
S(A)	
R(A)	
X(A)	
W(A)	
	S(A) <Denied, wait>
S(B)	
R(B)	
X(B)	
W(B)	
Rel(A)	
	S(A) <Granted>
	R(A)
	X(A)
	W(A)
Rel(B)	
	S(B)
	R(B)
	X(B)
	W(B)
	COMMIT
ABORT	

Outline

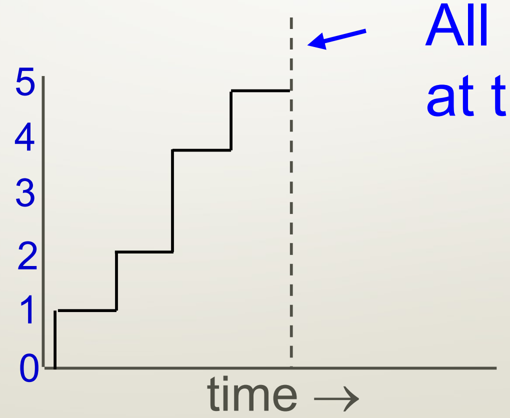
- ✓ Theory of Serializability
  - ✓ Serial and serializable schedules
  - ✓ Conflict-Serializable schedules
- Two-Phased Locking Theorem
  - ✓ Two-Phased Locking (2PL)
  - Strict two-phased locking (S2PL)

Strict Two Phase Locking Protocol (S2PL)

Strict 2PL is a way of managing locks during a transaction T

- T gets (S and X) locks gradually, as needed
- T holds all locks until end of transaction (commit/abort)

# of locks held by a transaction T



All locks are released at the end, upon commit or abort

Enforce S2PL

T1:	R(A),				R(C)		W(B)	
T2:		R(B)		W(C)		R(B)		
T3:			R(A)					R(B)

S1(A); R1(A); S2(B) R2(B); S3(A); R3(A); X2(C); W2(C); S1(C) <T1 WAIT ON C>; R2(B); COMMIT REL2(B,C); S1(C); R1(C); X1(B); W1(B); COMMIT; REL1(A,B,C); S3(B); R3(B); COMMIT; REL3(A,B);

Unrecoverable schedule solved with S2PL

T1:R(A) W(A) R(B) W(B)

T2: R(A) W(A) R(B) W(B)

Recoverable schedule!

T1	T2
S(A)	
R(A)	
X(A)	
W(A)	
	S(A) <Denied, wait>
S(B)	
R(B)	
X(B)	
W(B)	
ABORT Rel(A, B)	
	S(A) <Granted>
	R(A)
	X(A)
	W(A)
	S(B)
	R(B)
	X(B)
	W(B)
	COMMIT, Rel(A,B)

Strict 2PL guarantees serializability

- Can prove that a Strict 2PL schedule is equivalent to the serial schedule in which each transaction runs instantaneously at the time that it commits
- This is huge: A property of each transaction (2PL) implies a property of any set of transactions (serializability)
  - No need to check serializability of specific schedules
- Most DBMSs use 2PL to enforce serializability

Summary

- Transactions are all-or-nothing units of work guaranteed despite concurrency or failures in the system.
- Theoretically, the "correct" execution of transactions is serializable (i.e. equivalent to some serial execution).
- Practically, this may adversely affect throughput $\Rightarrow$ isolation levels.
- With isolation levels, users can specify the level of "incorrectness" they are willing to tolerate.