## Scheduler

42

- The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- How ?
  - Locks
  - Time stamps

42

## Lock Based Concurrency Control

43

- DBMS aims to allow only recoverable and serializable schedules
- Ensure committed transactions are not un-done while aborting transactions
- Use a *locking protocol*: a set of rules to be followed by each transaction to ensure serializable schedules
- *Lock*: a mechanism to control concurrent access to a data object

43

## Locking Scheduler

44

Simple idea:

- Each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
- If the lock is taken by another transaction, then wait
- The transaction must release the lock(s)

Adapted from M. Balazinska

44

## Notation

45

- Li(A) = transaction Ti acquires lock for element A
- Ui(A) = transaction Ti releases lock for element A

45

## Example 1

46

| T1 | T2 |
|---|---|
| L1(A) | |
| R1(A), W1(A) | |
| U1(A), L1(B) | |
| | L2(A) |
| | R2(A), W2(A) |
| | U2(A) |
| | L2(B) , **DENIED…** |
| R1(B), W1(B) | |
| U1(B) | |
| | **GRANTED;** |
| | R2(B), W2(B) |
| | U2(B) |

Scheduler has enforced a conflict serializable schedule

46

## Example 2

47

| T1 | T2 |
|---|---|
| L1(A) | |
| R1(A), W1(A) | |
| U1(A) | |
| | L2(A) |
| | R2(A), W2(A) |
| | U2(A) |
| | L2(B) |
| | R2(B), W2(B) |
| | U2(B) |
| L1(B) | |
| R1(B), W1(B) | |
| U1(B) | |

47

## Types of Locks

48

- Shared lock (for reading)
- Exclusive lock (for writing, and of course, also for reading)
- Notation
  - $S_T(A)$: transaction T requests shared lock on object A
  - $X_T(A)$: transaction T requests exclusive lock on object A

48

## Lock Modes

49

- S = Shared lock (for read)
- X = Exclusive lock (for write)

Lock compatibility matrix

| | None | S | X |
|---|---|---|---|
| **None** | OK | OK | OK |
| **S** | OK | OK | Conflict |
| **X** | OK | Conflict | Conflict |

49

## Strict Two Phase Locking (Strict 2PL)

50

- Most widely used locking protocol
- Two rules:
  1. Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  2. All locks held by a transaction are released when the transaction completes
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic (i.e., serializable)
- Recoverable and ACA

50

## Strict 2PL Example

51

| T1 | T2 |
|---|---|
| L(A); | |
| R(A), W(A) | |
| | L(A); **DENIED…** |
| L(B); | |
| R(B), W(B) | |
| U(A), U(B) | |
| Commit; | |
| | **…GRANTED** |
| | R(A), W(A) |
| | L(B); |
| | R(B), W(B) |
| | U(A), U(B) |
| | Commit; |

All locks held by T1 are released when T1 completes.

51

## Implications

52

- The locking protocol only allows safe interleavings of transactions
- If T1 and T2 access different data objects, then no conflict and each may proceed
- Otherwise, if same object, actions are ordered serially.
  - The Xact who gets the lock first must complete before the other can proceed

52

## Two Phase Locking Protocol (2PL)

53

- Variant of Strict 2PL
- Relaxes the 2nd rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)
- Two rules:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction cannot request additional locks once it releases any lock.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

53

## 2PL Example

54

| T1 | T2 |
|---|---|
| X(A), X(B) | |
| R(A), W(A) | |
| U(A) | |
| | X(A) |
| | R(A), W(A) |
| | X(B), **DENIED…** |
| R(B), W(B) | |
| U(B) | |
| | **..GRANTED** |
| | R(B), W(B) |
| | U(A), U(B) |

All locks are first acquired, and then released.

54

## 2PL Implications

55

- In every transaction, all lock requests must precede all unlock requests.
- This ensures conflict serializability
  - Why? (Think of order Xacts enter their shrinking phase)
  - This induces a sort ordering of the transactions that can be serialized

55

## Strict 2PL makes 2PL "strict"

56

- Recall: a strict schedule is one where a value written by T is not read/overwritten until T commits/aborts

- Strict 2PL makes T hold locks until commit/abort
- No other transaction can see or modify the data object until T is complete

56

## Remarks

57

- What if a transaction releases its locks and then aborts?
- Recall: conflict serializable definition only considers *committed* transactions

57

## Phantom Problem

58

- So far we have assumed the database to be a *static* collection of elements (=tuples)
- If tuples are inserted/deleted then the *phantom problem* appears

58

## Phantom Problem

59

| T1 | T2 |
|---|---|
| SELECT * FROM Product WHERE color='blue' | |
| | INSERT INTO Product(name, color) VALUES ('gizmo','blue') |
| SELECT * FROM Product WHERE color='blue' | |

Is this schedule serializable ?

59

## Phantom Problem

60

| T1 | T2 |
|---|---|
| SELECT * FROM Product WHERE color='blue' | |
| | INSERT INTO Product(name, color) VALUES ('gizmo','blue') |
| SELECT * FROM Product WHERE color='blue' | |

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

This is conflict serializable ! What's wrong ??

60

## Phantom Problem

61

| T1 | T2 |
|---|---|
| SELECT * FROM Product WHERE color='blue' | |
| | INSERT INTO Product(name, color) VALUES ('gizmo','blue') |
| SELECT * FROM Product WHERE color='blue' | |

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

61

## Phantom Problem

62

- A "phantom" is a tuple that is invisible during part of a transaction execution but not all of it.

- In our example:
  - T1: reads list of products
  - T2: inserts a new product
  - T1: re-reads: a new product appears !

62

## Phantom Problem

63

- In a ***static*** database:
  - Conflict serializability implies serializability

- In a ***dynamic*** database, this may fail due to phantoms

63

## Dealing With Phantoms

64

- Lock the entire table, or
- Lock the index entry for 'blue'
  - If index is available

Dealing with phantoms is expensive !

64