CSE232: Database System Principles

Correctness Issues at the intersection of Failures and Concurrency

1

Correctness issues at the intersection of failures and concurrency

- Cascading rollback, recoverable schedule
- Deadlocks
  - Prevention
  - Detection

2

Concurrency control & recovery

Example: T_j T_i

T_j	T_i
⋮	⋮
$W_j(A)$	
	⋮
	$r_i(A)$
⋮	Commit T_i
⋮	
Abort T_j	⋮

☛ Cascading rollback (Bad!)

3

- Schedule is conflict serializable
- $T_j \longrightarrow T_i$
- But not recoverable

4

- Need to make "final' decision for each transaction:
  - **commit decision** - system guarantees transaction will or has completed, no matter what
  - **abort decision** - system guarantees transaction will or has been rolled back (has no effect)

5

To model this, two new actions:

- C_i - transaction T_i commits
- A_i - transaction T_i aborts

6

Back to example:

T_j	T_i
⋮	⋮
$W_j(A)$	
	$r_i(A)$
⋮	⋮
	C_i ← can we commit here?

7

Definition

T_i reads from T_j in S ($T_j \Rightarrow_S T_i$) if

(1) $w_j(A) <_S r_i(A)$

(2) $a_j \not<_S r_i(A)$ ($\not<$: does not precede)

(3) If $w_j(A) <_S w_k(A) <_S r_i(A)$ then $a_k <_S r_i(A)$

8

Definition

Schedule S is recoverable if whenever $T_j \Rightarrow_S T_i$ and $j \neq i$ and $C_i \in S$ then $C_j <_S C_i$

9

Note: in transactions, reads and writes precede commit or abort

⇔ If $C_i \in T_i$, then $r_i(A) < C_i$
$w_i(A) < C_i$

⇔ If $A_i \in T_i$, then $r_i(A) < A_i$
$w_i(A) < A_i$

- Also, one of C_i, A_i per transaction

10

How to achieve recoverable schedules?

11

⇔ With 2PL, hold write locks to commit (<u>strict 2PL</u>)

T_j	T_i
⋮	⋮
$W_j(A)$	⋮
⋮	⋮
C_j	⋮
$u_j(A)$	
⋮	$r_i(A)$

12

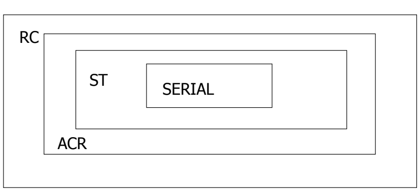
⇔ With validation, no change!

13

- S is recoverable if each transaction *commits* only after all transactions from which it read have committed.
- S avoids cascading rollback if each transaction may *read* only those values written by committed transactions.

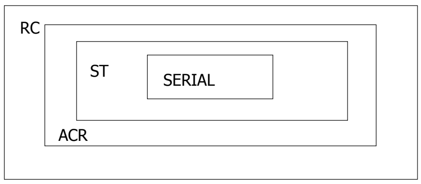
14

- S is strict if each transaction may *read and write* only items previously written by committed transactions.



15

Where are serializable schedules?



Examples

- Recoverable:
  - $w_1(A)\ w_1(B)\ \ w_2(A)\ r_2(B)\ \ \ c_1\ c_2$
- Avoids Cascading Rollback:
  - $w_1(A)\ w_1(B)\ \ w_2(A)\ \ c_1\ r_2(B)\ \ \ c_2$

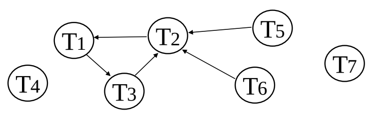
    Assumes $w_2(A)$ is done without reading
- Strict:
  - $w_1(A)\ w_1(B)\ c_1\ \ w_2(A)\ r_2(B)\ \ \ c_2$

Deadlocks

- Detection
  - Wait-for graph
- Prevention
  - Resource ordering
  - Timeout
  - Wait-die
  - Wound-wait

Deadlock Detection

- Build Wait-For graph
- Use lock table structures
- Build incrementally or periodically
- When cycle found, rollback victim



19

Resource Ordering

- Order all elements $A_1, A_2, \ldots, A_n$
- A transaction T can lock A_i after A_j only if $i > j$

Problem : Ordered lock requests not realistic in most cases

20

Timeout

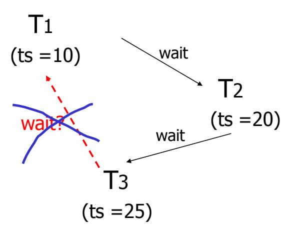
- If transaction waits more than L sec., roll it back!
- Simple scheme
- Hard to select L

21

Wait-die

- Transactions given a timestamp when they arrive …. $ts(T_i)$
- T_i can only wait for T_j if $ts(T_i) < ts(T_j)$
  ...else die

Example:



Wound-wait

- Transactions given a timestamp when they arrive … $ts(T_i)$
- T_i wounds T_j if $ts(T_i) < ts(T_j)$
  else T_i waits

"Wound": T_j rolls back and gives lock to T_i