

Lecture 19

Note Title

3/25/2008

Transactions

ACID properties

1. Atomicity - completes instantaneously
2. Consistency - transaction takes valid states to valid states
(safety property) contrast "liveness property"
3. Isolation - transaction only "sees" results from completed transactions; serializability: result of many transactions run concurrently is the same as some serial order of same transactions together 2 & 3 Say: if ^{every} transaction preserves consistency when run alone, the system always preserves consistency

4. Durability - once committed a transaction's changes persist across: system restarts, failures, partial failures.

What do transactions give us that monitors/locks/conditions don't? Commit & Abort



- choices :
- 1) return failure indicator
 - 2) raise an exception
 - 3) abort - automatically puts state back as it was at transaction start.

Lock treatment

Simple locks - once taken cannot be taken again

R/W - reader-writer locks -

Reader lock allows other X_{act} s to get reader locks
prevents getting writer lock.

Writer prevents o.t. getting either kind of lock.

<u>X_{act2}</u>	<u>X_{act1}</u>		
L(A)	L(A)	lock A	Consistency means $A+B=C$
$A=A+1$	$A=A+1$		
$u(A) \leftarrow$	$u(A) \leftarrow$		
L(B)	L(B)		
$B=B-1$	$B=B-1$		
$u(B)$	$u(B)$		

Rule for using locks in xacts:

2-phase locking rule.

P1: Acquire locks only

P2: Release locks only

if obeyed ensures global consistency given local consistency of xacts.
in practice P2 is often delayed to commit time.

Lock manager

Helps deal w/ deadlocks

What characterizes a deadlock?

holds L_1

holds L_2

Deadly embrace

wants L_2

wants L_1

Generally: Think of wait-for graph:

nodes are transactions

edges: edge from T_1 to T_2 if T_1 is
trying to take a lock held by T_2

Deadlock: if WFG contain a cycle.

still preserves the ACID properties

to characterize we expect of system we
also need liveness properties

§ 8.5 in textbook

For deadlock can sidestep the liveness issue & formulate safety property: no cycle in WFG

How do we guarantee no deadlocks:

1) always acquire locks in same order.
— hard to do sometimes.

2) LM can maintain WFG for its own use —

- avoid acquiring locks that create a cycle.

expensive ← About the requestin transaction

- optimistically assume that giving a lock doesn't create a cycle.

— periodically check for cycles. — about some transaction in the cycle.

3) Give priority to older transaction
if older transaction wants lock held by a younger
transaction abort the younger transaction and restart it

Another approach w/o locks: Optimistic concurrency control