Long Duration transactions

Long Duration Transactions

Traditional concurrency control techniques do not work well when user interaction is required:

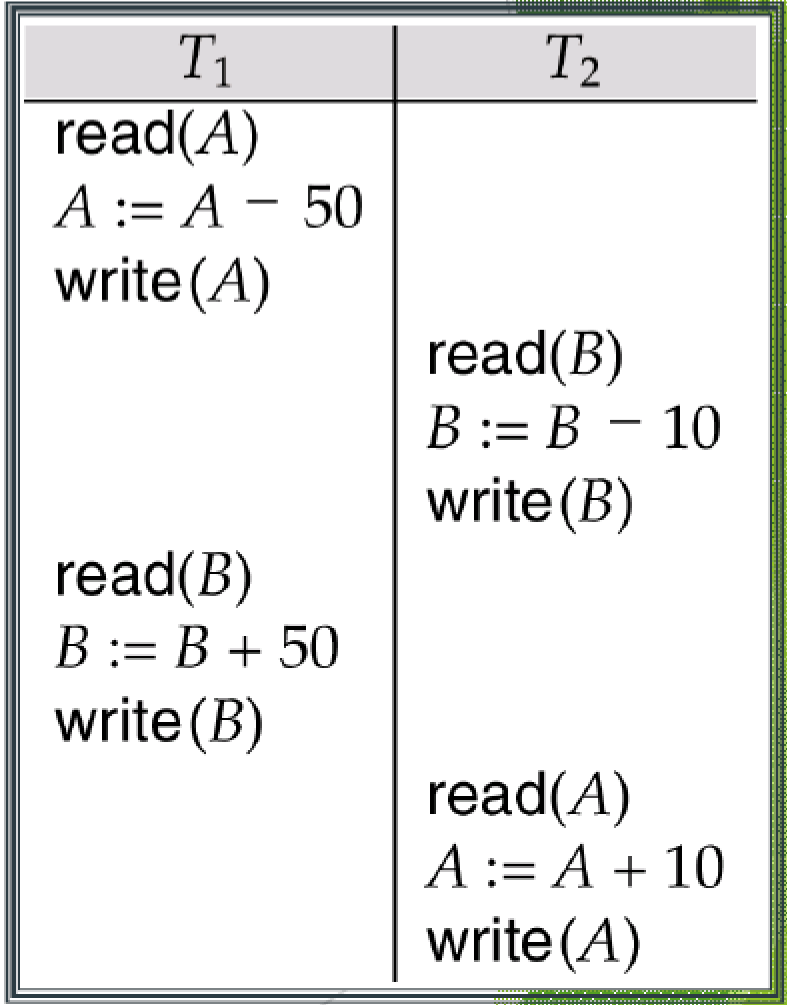
- **Long duration:** Design edit sessions are very long
- **Exposure of uncommitted data:** E.g., partial update to a design
- **Subtasks:** support partial rollback
- **Recoverability:** on crash state should be restored even for yet-to-be committed data, so user work is not lost.
- **Performance:** fast response time is essential so user time is not wasted.

Long-Duration Transactions

- Represent as a nested transaction
  - atomic database operations (read/write) at a lowest level.
- If transaction fails, only active short-duration transactions abort.
- Active long-duration transactions resume once any short duration transactions have recovered.
- The efficient management of long-duration waits, and the possibility of aborts.
- Need alternatives to waits and aborts; alternative techniques must ensure correctness without requiring serializability.

Concurrency Control (Cont.)

A non-conflict-serializable schedule that preserves the sum of A + B



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Nested and Multilevel Transactions

- A **nested or multilevel transaction** T is represented by a set $T = \{t_1, t_2, \ldots, t_n\}$ of subtransactions and a partial order P on T.
- A subtransaction t_i in T may abort without forcing T to abort.
- Instead, T may either restart t_i, or simply choose not to run t_i.
- If t_i commits, this action does not make t_i. Instead, t_i, commits to T, and may still abort (or require compensation) if T aborts.
- An execution of T must not violate the partial order P, i.e., if an edge $t_i \rightarrow t_j$ appears in the precedence graph, then $t_j \rightarrow t_i$ must not be in the transitive closure of P.

Nested and Multilevel Transactions (Cont.)

- Subtransactions can themselves be nested/multilevel transactions.
  - Lowest level of nesting: standard read and write operations.
- Nesting can create higher-level operations that may enhance concurrency.
- Types of nested/ multilevel transactions:
  - **Multilevel transaction**: subtransaction of *T* is permitted to release locks on completion.
  - **Saga**: multilevel long-duration transaction.
  - **Nested transaction**: locks held by a subtransaction t_i of *T* are automatically assign to *T* on completion of t_i.

Example of Nesting

- Rewrite transaction T_1 using subtransactions T_a and T_b that perform increment or decrement operations:
  - T_1 consists of
    - $T_{1,1}$, which subtracts 50 from A
    - $T_{1,2}$, which adds 50 to B
- Rewrite transaction T_2 using subtransactions T_c and T_d that perform increment or decrement operations:
  - T_2 consists of
    - $T_{2,1}$, which subtracts 10 from B
    - $T_{2,2}$, which adds 10 to A
- No ordering is specified on subtransactions; any execution generates a correct result.

Compensating Transactions

- Alternative to undo operation; compensating transactions deal with the problem of cascading rollbacks.
- Instead of undoing all changes made by the failed transaction, action is taken to “compensate” for the failure.
- Consider a long-duration transaction T_i representing a travel reservation, with subtransactions $T_{i,1}$, which makes airline reservations, $T_{i,2}$ which reserves rental cars, and $T_{i,3}$ which reserves a hotel room.
  - Hotel cancels the reservation.
  - Instead of undoing all of T_i, the failure of $T_{i,3}$ is compensated for by deleting the old hotel reservation and making a new one.
  - Requires use of semantics of the failed transaction.

Implementation Issues

- For long-duration transactions to survive system crashes, we must log not only changes to the database, but also changes to internal system data pertaining to these transactions.
- Logging of updates is made more complex by physically large data items (CAD design, document text); undesirable to store both old and new values.
- Two approaches to reducing the overhead of ensuring the recoverability of large data items:
  - Operation logging. Only the operation performed on the data item and the data-item name are stored in the log.
  - Logging and shadow paging. Use logging from small data items; use shadow paging for large data items. Only modified pages need to be stored in duplicate.

Presentación

- Esta presentación fue armada utilizando, además de material propio, material contenido en los manuales de Oracle y material provisto por los siguientes autores
- Silberschat, Korth, Sudarshan - Database Systems Concepts, 6th Ed., Mc Graw Hill, 2010
- García Molina/Ullman/Widom - Database Systems: The Complete Book, 2nd Ed.,Prentice Hall, 2009