COL362-632: Concurrency Control Schemes

Garima Gaur

April 11, 2023

Buffer Management

- Disk block: DB is logically partitioned into blocks residing on disk.
- Buffer block: During a transaction execution, relevant blocks of DB are copied to the RAM.
- Each transaction has a local variable copy.

Buffer Management

- Disk block: DB is logically partitioned into blocks residing on disk.
- Buffer block: During a transaction execution, relevant blocks of DB are copied to the RAM.
- Each transaction has a local variable copy.
- Transferring blocks between buffer and disk: input(B) and output(B).
- Read(b) includes input(B) and Write(b) copies local variable value to buffer block.

Concurrency Control Manager

- Responsibility of DBMS to control the interaction of concurrent transactions.
  - Lock based Protocol
  - Timestamp based Protocol
  - Validation Based Protocol

Locking Mechanism

- Locks to ensure sharing of data items is mutually exclusive.
- Two types of locks
  - Shared lock S(A): write operations not permitted
  - Exclusive lock X(A): both read and write operation permitted
- Lock compatibility matrix

	S	X
S	T	F
X	F	F

Example

T_1
lock-X(A);
read(A);
A := A -50;
write(A);
unlock(A);
lock-X(B);
read(B);
B := B + 50;
write(B);
unlock(B);

T_2
lock-S(B);
read(B);
unlock(B);
lock-S(A);
read(A);
unlock(A);
display(A+B);

T_1	T_2
lock-X(A)	
read(A)	
	lock-S(B)
	read(B)
	unlock(B)
write(A)	
unlock(A)	
lock-X(B)	
read(B)	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)

Example

T_1
lock-X(A);
read(A);
A := A -50;
write(A);
unlock(A);
lock-X(B);
read(B);
B := B + 50;
write(B);
unlock(B);

T_2
lock-S(B);
read(B);
unlock(B);
lock-S(A);
read(A);
unlock(A);
display(A+B);

T_1	T_2
lock-X(A)	
read(A)	
	lock-S(B)
	read(B)
	unlock(B)
write(A)	
unlock(A)	
lock-X(B)	
read(B)	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)

- Only assessing the compatibility of locks is not a good measure.

Does late unlocking help?

$\mathbf{T_1}$
lock-X(A);
read(A);
A := A -50;
write(A);
lock-X(B);
read(B);
B := B + 50;
write(B);
unlock(A);
unlock(B);

$\mathbf{T_2}$
lock-S(B);
read(B);
lock-S(A);
read(A);
display(A+B);
unlock(B);
unlock(A);

$\mathbf{T_1}$	$\mathbf{T_2}$
lock-X(A)	
read(A)	
	lock-S(B)
	read(B)
write(A)	
lock-X(B)	
	lock-S(A)

Does late unlocking help?

$\mathbf{T_1}$
lock-X(A);
read(A);
A := A -50;
write(A);
lock-X(B);
read(B);
B := B + 50;
write(B);
unlock(A);
unlock(B);

$\mathbf{T_2}$
lock-S(B);
read(B);
lock-S(A);
read(A);
display(A+B);
unlock(B);
unlock(A);

$\mathbf{T_1}$	$\mathbf{T_2}$
lock-X(A)	
read(A)	
	lock-S(B)
	read(B)
write(A)	
lock-X(B)	
	lock-S(A)

- T_1 and T_2 are in a deadlock!

2 Phase Locking Protocol

- Transactions operate in 2 phases – growing phase and shrinking phase.
- Growing phase – acquiring locks
- Shrinking phase – releasing locks
- Lock point – the point of acquiring last lock (end of growing phase).

2 Phase Locking Protocol

- Transactions operate in 2 phases – growing phase and shrinking phase.
- Growing phase – acquiring locks
- Shrinking phase – releasing locks
- Lock point – the point of acquiring last lock (end of growing phase).
- Locking point-based ordering of transactions is a serializability ordering of transactions.

Variants of 2 PL

- 2PL ensures conflict serializability but does not avoid deadlocks.
- Does not guarantee recoverability and cascadelessness.

Variants of 2 PL

- 2PL ensures conflict serializability but does not avoid deadlocks.
- Does not guarantee recoverability and cascadelessness.
- Different variants of PL:
  - Strict 2PL: Avoids cascading rollbacks but may deadlock.
  - Rigorous 2PL: Another (more serial) variant to avoid cascading rollbacks.
  - Conservative 2PL: Deadlock free.

Variants of 2 PL

- 2PL ensures conflict serializability but does not avoid deadlocks.
- Does not guarantee recoverability and cascadelessness.
- Different variants of PL:
  - Strict 2PL: Avoids cascading rollbacks but may deadlock.
  - Rigorous 2PL: Another (more serial) variant to avoid cascading rollbacks.
  - Conservative 2PL: Deadlock free.
- More concurrency comes with the possibility of deadlocks.
- Deadlock handling – prevention and detection-recovery.

Timestamp based Protocol

- Deciding the ordering of transactions beforehand – assign timestamps to transactions.

Timestamp based Protocol

- Deciding the ordering of transactions beforehand – assign timestamps to transactions.
- $\mathbf{TS(T_i)}$: The timestamp assigned to a transaction T_i before it starts execution.
- $TS(T_i) < TS(T_j)$: T_i has entered the system before the T_j.
- $\mathbf{W - TS(Q)}$: Largest timestamp of any transactions that executed write(Q).
- $\mathbf{R - TS(Q)}$: Largest timestamp of any transaction that executed read(Q).

Timstamp-Ordering Protocol

- A transaction T_i issues $read(Q)$
    - $\mathbf{TS(T_i) < W - TS(Q)}$: T_i needs to read a value of Q that was already over-written. Rollback T_i
    - $\mathbf{TS(T_i) \geq W - TS(Q)}$: Read(Q) executed and set $R - TS(Q) = max\{R - TS(Q), TS(T_i)\}$

Timstamp-Ordering Protocol

- A transaction T_i issues $read(Q)$
  - $\mathbf{TS(T_i) < W - TS(Q)}$: T_i needs to read a value of Q that was already over-written. Rollback T_i
  - $\mathbf{TS(T_i) \geq W - TS(Q)}$: Read(Q) executed and set $R - TS(Q) = max\{R - TS(Q), TS(T_i)\}$
- A transaction T_i issues $write(Q)$
  - $\mathbf{TS(T_i) < R - TS(Q)}$: The value of Q that T_i is producing was needed earlier, and the system assumed that the value would never be produced. Rollback T_i.
  - $\mathbf{TS(T_i) < W - TS(Q)}$: T_i is trying to write an obsolete value of Q. Rollback T_i

Timstamp-Ordering Protocol

- Conflict serializable
- No deadlocks – processes are rolled back.

Timstamp-Ordering Protocol

- Conflict serializable
- No deadlocks – processes are rolled back.

T_4	T_5
read(Q)	
	write(Q)
write(Q)	

- With timestamp ordering protocol, will T_4 complete?

Timstamp-Ordering Protocol

- Conflict serializable
- No deadlocks – processes are rolled back.

T_4	T_5
read(Q)	
	write(Q)
write(Q)	

- With timestamp ordering protocol, will T_4 complete? Obsolete write!
- Write(Q) of T_4 can be ignored. Why?

Timstamp-Ordering Protocol

- Conflict serializable
- No deadlocks – processes are rolled back.

T_4	T_5
read(Q)	
	write(Q)
write(Q)	

- With timestamp ordering protocol, will T_4 complete? Obsolete write!
- Write(Q) of T_4 can be ignored. Why?
- Earlier transactions ($TS(T_i) < TS(T_5)$) that attempt to read Q will be rolled back.
- Obsolete writes can be just ignored and transactions need not be roll-backed.

Thomas' Write Rule

- Thomas' Write Rule: Modification in Timestamp ordering write operation protocol.
- T_i issues write(Q),
  - $\mathbf{TS(T_i) < R - TS(Q)}$: Rollback T_i
  - $\mathbf{TS(T_i) < W - TS(Q)}$: T_i is writing an obsolete value – **Ignore the write operation and proceed**.

More concurrency

T_4	T_5
read(Q)	
	write(Q)
write(Q)	

- Is it conflict serializable? No.
- Is it view serializable?

More concurrency

T_4	T_5
read(Q)	
	write(Q)
write(Q)	

- Is it conflict serializable? No.
- Is it view serializable? Equivalent to a serial schedule $< T_4, T_5 >$
- Thomas' write rule allows view serializable schedules.

Validation Protocol

- Majority of transactions are read-only transactions.
- Locking or ordering read-only is inefficient.
- Each transaction operates in 3 phases (in order)

Validation Protocol

- Majority of transactions are read-only transactions.
- Locking or ordering read-only is inefficient.
- Each transaction operates in 3 phases (in order)
  - Read phase: Read all data items and update only the local copies.
  - Validation phase: Use the validation test to determine if the transaction can proceed to the write phase without causing a violation of serializability.
  - Write phase: Updated local temporary values are copied to the database.

Validation Protocol

- Each transaction maintains 3 timestamp values:
  - $\mathbf{StartTS(T_i)}$: Time when T_i started execution.
  - $\mathbf{ValidateTS(T_i)}$: Time when T_i started its validation phase.
  - $\mathbf{FinishTS(T_i)}$: Time when T_i finished its Write phase.
- For serializability checks, $validateTS(T_i)$ of a transaction is treated as its timestamp $TS(T_i)$.

Validation Test

- Validation test for T_i requires all transactions T_k with $TS(T_k) < TS(T_i)$ to satisfy one of the conditions,
    - $FinishTS(T_k) < StartTS(T_i)$: Earlier transactions have already completed.
    - Set of data items written by T_k and set of items read by T_i does not overlap and $FinishTS(T_k) < ValidationTS(T_i)$
- Validation-based protocols are called *optimisitc* concurrency control.

Snapshot Isolation

- Snapshot Isolation: Oracle, PostgreSQL, SQL Server

Snapshot Isolation

- Snapshot Isolation: Oracle, PostgreSQL, SQL Server
- Each transaction has its own snapshot of DB – reflecting changes made by the earlier committed transactions.
- Each transaction T_i has 2 timestamps:
  - $StartTS(T_i)$: Start time of T_i.
  - $CommitTS(T_i)$: Time at which validation is requested by T_i.

Snapshot Isolation

- Snapshot Isolation: Oracle, PostgreSQL, SQL Server
- Each transaction has its own snapshot of DB – reflecting changes made by the earlier committed transactions.
- Each transaction T_i has 2 timestamps:
  - $StartTS(T_i)$: Start time of T_i.
  - $CommitTS(T_i)$: Time at which validation is requested by T_i.
- Each transaction T_i is unaware of updates of concurrent transaction T_j.
- Transactions are ordered:
  - *First committer wins*: If there exist a data item A that T_i intends to write and $StartTS(T_i) < TS(A) < CommitTS(T_i)$, then abort T_i.
  - *First updater wins*: *Lock* based approach, the transaction that acquires the item lock first, updates the item.

Serializability Issue

T_1	T_2
read(A)	
read(B)	
	read(A)
	read(B)
A=B	
	B=A
write(A)	
	write(B)

Serializability Issue

T_1	T_2
read(A)	
read(B)	
	read(A)
	read(B)
A=B	
	B=A
write(A)	
	write(B)

- Both transactions can commit under isolation snapshot – writing different data items.
- Is this schedule serializable?

Serializability Issue

T_1	T_2
read(A)	
read(B)	
	read(A)
	read(B)
A=B	
	B=A
write(A)	
	write(B)

- Both transactions can commit under isolation snapshot – writing different data items.
- Is this schedule serializable? No; swapping values of A and B!

Serializability Issue

T_1	T_2
read(A)	
read(B)	
	read(A)
	read(B)
A=B	
	B=A
write(A)	
	write(B)

- Both transactions can commit under isolation snapshot – writing different data items.
- Is this schedule serializable? No; swapping values of A and B!
- Serializable snapshot isolation (SSI): Details in Section 18.8.3 of the book.