Concurrency Control Techniques

Sanjay Kumar Gupta

Assistant Professor - PSIT

Concurrency Control

When several transaction execute simultaneously then there is a **risk of violation of data integrity**.

- **Concurrency Control** is a **procedure of managing simultaneous transactions** ensuring their **Atomicity, Isolation, Consistency & Serializability**.
- In absence of concurrency control mechanism, we may face:

Concurrency Problems in Transactions

1. **Dirty Read Problem**
2. **Lost Update Problem**

Lock Based Protocol

A Lock is a mechanism to **control concurrent access** to a data item.

1. Shared Mode (S):

-> **Only read operation** on data is applicable

Transaction T1	
LOCK S (A)	// Shared Lock on data "A"
R (A)	// Read Operation on data "A"
Unlock (A)	// Unlock data "A"

2. Exclusive Mode (X):

-> **Both read & write operations** on data are applicable.

Transaction T1	
LOCK X (A)	// Exclusive Lock on data "A"
R (A)	// Read Operation on data "A"
W (A)	// Write Operation on data "A"
Unlock (A)	// Unlock data "A"

Lock Compatibility matrix

Suppose **T1** and **T2** are **parallel transactions**, and **both** want to perform read and write operations on the same data, say "A".

A shared lock is denoted by **"S,"** and an Exclusive lock is denoted by **"X"**.

Request →

Grant ↓

	Shared (S)	Exclusive (X)
Shared (S)	✓	✗
Exclusive (X)	✗	✗

Compatibility Lock Table

Examples of a Transaction Using Locks

T1
Lock-S(A)
Read(A)
Unlock(A)
Lock-S(B)
Read(B)
Unlock(B)
Display(A+B)

Drawbacks of Lock Based Protocol

Consider the given *partial* schedule,

T3	T4
lock-X(B)	
Read(B)	
B=B-50	
write(B)	
	lock-S(A)
	Read(A)
	Lock-S(B)
lock-X(A)	
	lock-X(A)

1. Deadlock
2. Starvation

Two-Phase Locking Protocol

Time	T1	T2
Time1	Lock-S (A)	
Time2		Lock-S (A)
Time3	Lock-X (B)	
Time4		R (A)
Time5	Unlock (A)	
Time6		Lock-X (C)
Time7	Unlock (B)	
Time8		Unlock (A)
Time9		Unlock (C)

T1 Growing Phase

Lock Point Of T1

T1 Shrinking Phase

T2 Growing Phase

Lock Point Of T2

T2 Shrinking Phase

2-PL Locking Example

Two-Phase Locking Protocol

- 2-PL is an **extension of Shared/Exclusive locking**.
- It is used to reduce the problems of Shared/Exclusive locking

Any schedule that follows **2PL** will **always** be serializable, which was not in Shared/Exclusive locking

Phases of 2PL

I. **Growing Phase:** In the Growing phase, only Locks are acquired by a transaction and no locks are released by a transaction at that time.

II. **Shrinking Phase:** In the shrinking phase, only Locks are released by transaction, and no locks are acquired by a transaction at that time.

Rigorous 2-PL

It is like Strict 2-PL in its advantages and disadvantages **but** a little bit **more strict** than Strict 2-PL.

- It must satisfy the **Basic 2-PL**
- Each transaction should hold **all Exclusive(X) Locks** as well as **Shared(S) Locks** until the Transaction is Commits or Aborted.

T1	T2
Lock-S(A)	
READ (A)	
	Lock-S(A) // Wait
LOCK-X(B)	
READ (B)	
WRITE (B)	
Commit (A)	
Commit (B)	
Unlock (A)	
Unlock (B)	Lock-S(A) // Granted
	READ (A)
	Commit (A)
	Unlock (A)

9

Strict 2-PL

A schedule will be in Strict 2PL if,

- It must satisfy the **Basic 2-PL**
- Each transaction should hold all **Exclusive(X) Locks** until the Transaction is Committed or Aborted.

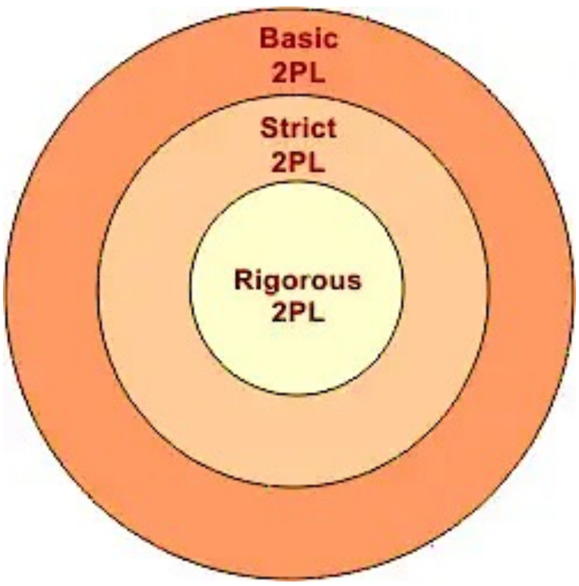
T1	T2
Lock-S(A)	
READ (A)	
	Lock-S(A) // Granted
LOCK-X(B)	
READ (B)	
WRITE (B)	
Commit (A)	
Commit (B)	
Unlock (A)	
Unlock (B)	

Conservative 2-PL

The idea is there is **no growing phase**, transaction start directly from lock point, i.e., **transaction must first acquire all the required locks then only it can start execution**.

Shrinking phase will work as usual, and **transaction can unlock any data item anytime**.

We must have a knowledge in future to understand what is data required so that we can use it.



Other Protocols

Lock based protocols **ensure Conflict Serializability**, **but** it causes two problems:

1. Deadlock &
2. Starvation

The alternative approaches to control **concurrency** are:

A. Timestamp-Based Protocols
B. Validation Based Protocols

Time Stamping Protocols

- The **Timestamp Ordering Protocol** arranges the transactions based on their respective **timestamps**.
- Basic idea of time stamping is to decide the order between the transaction before they enter in the system using a stamp (time stamp), in case of any conflict during the execution order can be decided using the time stamp.

Example: Suppose three transactions, T1, T2, and T3, execute in parallel on the same data A.

- "T1 arrived at 8:00 AM. Please assign a timestamp value of 100 and call it '**Older**.'
- "T2 arrived at 8:10 AM, so please assign a timestamp value of 200 and name it '**Younger**.'
- "T3 arrived at 8:15 AM. Therefore, assign a timestamp value of 300 and name it '**Youngest**.'

Time Stamping Protocols

❖ Let's understand how this protocol works, here we have two idea of timestamping, one for the transaction, and other for the data item.

Time stamp for transaction

- With each transaction t_i in the system, we associate a unique fixed timestamp → denoted by **TS(t_i)**.
- If a transaction has been assigned a timestamp TS(t_i) and a new transaction t_j, enters the system with a timestamp TS(t_j), then always **TS(t_i) <TS(t_j)**

Time Stamping Protocols

Two things are to be noted:

1. Time stamp of a transaction remain fixed throughout the execution
2. It is unique means no two transaction can have the same timestamp

Time stamp with data item

- **W-timestamp(Q)** is the largest time-stamp of any transaction that executed write(Q) successfully
- **R-timestamp(Q)** is the largest time-stamp of any transaction that executed read(Q) successfully

These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

Time Stamping Protocols

Suppose a transaction **Ti** request a ***read(Q),***

- If **$TS(T_i) < W\text{-}timestamp(Q)$,** then Ti needs to read a value of Q that was already overwritten.

Hence, the read operation is rejected, and Ti is rolled back

- If **$TS(T_i) \geq W\text{-}timestamp(Q)$,** then the read operation is executed, and **R-timestamp(Q)** is set to the **maximum** of **R-timestamp(*Q*)** and **TS(T_i)**.

Time Stamping Protocols

Suppose a transaction **T_i** request a ***write(Q),***

1: If **$TS(T_i) < R\text{-}timestamp(Q)$**

Write rejected; **T_i rolled back** (value needed earlier)

2: If **$TS(T_i) < W\text{-}timestamp(Q)$**

Write rejected; **T_i rolled back** (obsolete value)

3: If **$TS(T_i) \geq R\text{-}timestamp(Q)$**

Write operation is **executed**,
and **W-timestamp(Q) is set to TS(T_i)**

4: If **$TS(T_i) \geq W\text{-}timestamp(Q)$**

Write operation is **executed**,
and **W-timestamp(Q) is set to TS(T_i)**

Thomas Write Rule

Improvement over the timestamp-ordering protocol.

Increases **concurrency** and may generate **view-serializable** schedules.

Key Features:

- **Blind writes** can be ignored under specific conditions.
- **Read rules** remain unchanged.
- **Write rules** slightly modified from timestamp-ordering protocol.

When Ti attempts to write data item Q,

- **if $TS(T_i)$ < W-timestamp(Q),** then Ti is attempting to write an obsolete value of {Q}.

Rather than *rolling back T_i as the timestamp ordering protocol would have done*, this {write} operation can be ignored.

Timestamp Ordering Protocol

Advantages of Timestamp Protocol

- The Timestamp protocol ensures serializability and Deadlock removal because a transaction with a smaller timestamp (TS) executes before a transaction with a higher TS.

Disadvantages of Timestamp Protocol

- The schedule may not be recoverable.
- The schedule may not be cascading-free.

Validation Based Protocol

It is also known as **optimistic concurrency control technique**.

- Used where transactions are **READ** only & **CONFLICTS** are low.

❖ **Every T_i must go through three phases:**

1. Read & Execution Phase

- Transaction Ti writes only to the **temporary local variable**.
- It means, it does not update actual database.

2. Validation Phase

- T_i performs **validation test** to determine if **local variables can be written without violating serializability**.

3. Write Phase

- If **T_i is validated**, the updates are **applied to the database,**
- **or Ti is rolled back**.

Validation Based Protocol

Each Transaction has **three** timestamps:

1. **start (T_i):** the time when **T_i** started its execution
2. **validation (T_i):** the time when **T_i** entered validation phase
3. **finish (T_i):** the time when **T_i** finished its write phase

To clear the **validation test** by **T_i**, must follow **anyone** conditions:

(i) Finish (T_i) < Start (T_j): It means T_i is Older transaction & It get finished before T_j starts. **(NO OVERLAP)**

(ii) Finish (T_i) < Validate (T_j): This ensures actual write by T_i & T_j will **NOT OVERLAP**.

(iii) Validate (T_i) < Validate (T_j): It ensures that T_i has completed Read phase before T_j completes Read Phase.

Validation Test

T_{14}	T_{15}
read(B)	
	read(B)
	$B := B - 50$
	read(A)
	$A := A + 50$
read(A)	
⟨ *validate* ⟩	
display($A + B$)	
	⟨ *validate* ⟩
	write(B)
	write(A)

ADVANTAGES

-> Maintains Serializability

-> Free from Cascade Rollback

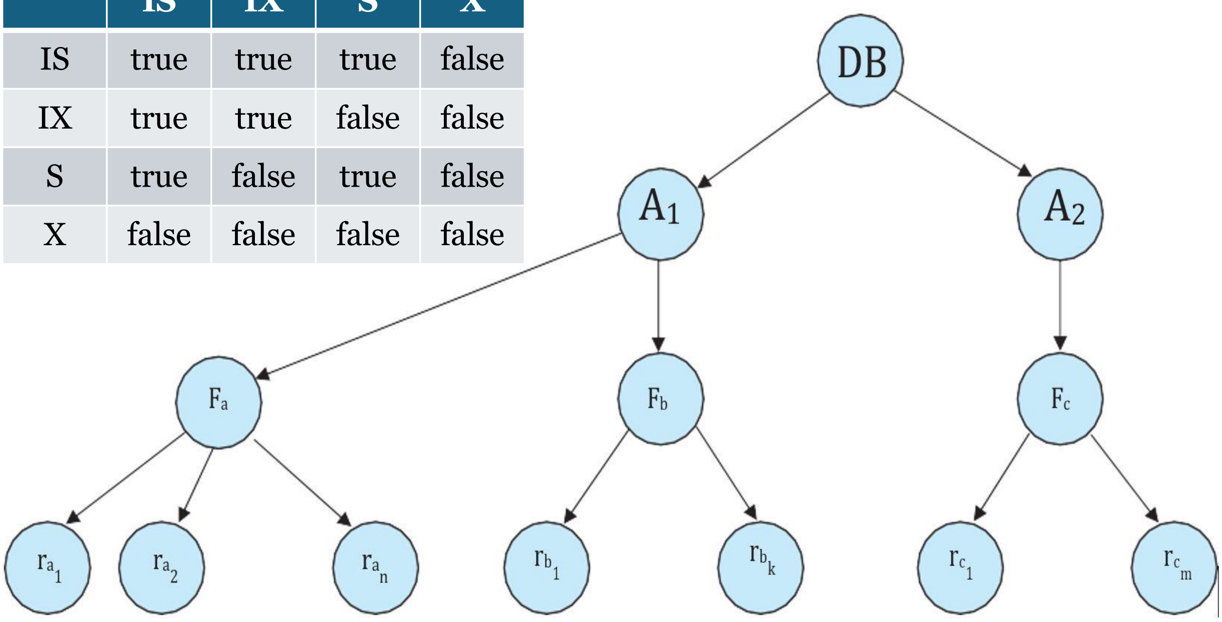
-> Less Overhead than other Protocols

DISADVANTAGES

-> Starvation of long transaction due to conflicting transaction

Multiple Granularity

	IS	IX	S	X
IS	true	true	true	false
IX	true	true	false	false
S	true	false	true	false
X	false	false	false	false



Multiple Granularity

Granularity: It is the size of data item allowed to lock.

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item.
- This type of hierarchy can be graphically represented as a tree.