Concurrency II

Database Systems

Michael Pound

This Lecture

- Deadlocks
  - Deadlock detection
  - Deadlock recovery
  - Deadlock prevention
- Timestamping
- Further reading
  - The Manga Guide to Databases, Chapter 5
  - Database Systems, Chapter 22

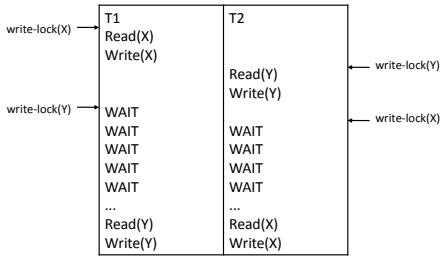
Last Lecture

- Serialisability
  - Schedules of transactions
  - Serial and serialisable schedules
  - Conflict serialisable schedules
- Locks
  - Shared (Read)
  - Exclusive (Write)
- Two-phase locking
  - Growing Phase
    - Transactions obtain locks
  - Shrinking Phase
    - Transactions release locks
- Two-phase locking guarantees conflict serialisability
  - Allows Concurrency
  - Avoids loss of Isolation

Deadlocks

- A deadlock is an impasse that may result when two or more transactions are waiting for locks to be released which are held by each other.
  - For example: T1 has a lock on X and is waiting for a lock on Y, and T2 has a lock on Y and is waiting for a lock on X.
- We can detect deadlocks that will happen in a schedule using a *wait-for graph* (WFG).

2PL Deadlock Example



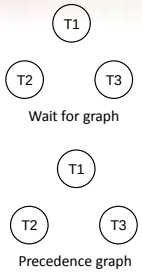
Precedence/Wait-For Graphs

- Precedence graph
  - Each transaction is a vertex
  - Edge from T1 to T2 if
    - T1 reads X before T2 writes X
    - T1 writes X before T2 reads X
    - T1 writes X before T2 writes X
- Wait-for Graph
  - Each transaction is a vertex
  - Edge from T2 to T1 if
    - T1 read-locks X then T2 tries to write-lock it
    - T1 write-locks X then T2 tries to read-lock it
    - T1 write-locks X then T2 tries to write-lock it

Example

Schedule

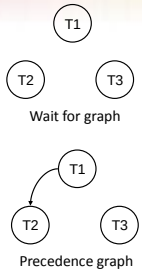
T1 Read(X)
T2 Read(Y)
T1 Write(X)
T2 Read(X)
T3 Read(Z)
T3 Write(Z)
T1 Read(Y)
T3 Read(X)
T1 Write(Y)



Example

Schedule

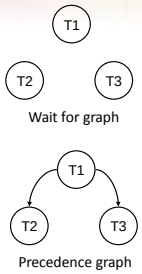
T1 Read(X)
T2 Read(Y)
T1 Write(X)
T2 Read(X)
T3 Read(Z)
T3 Write(Z)
T1 Read(Y)
T3 Read(X)
T1 Write(Y)



Example

Schedule

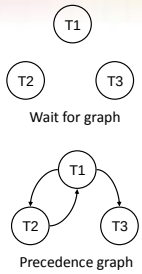
T1 Read(X)
T2 Read(Y)
T1 Write(X)
T2 Read(X)
T3 Read(Z)
T3 Write(Z)
T1 Read(Y)
T3 Read(X)
T1 Write(Y)



Example

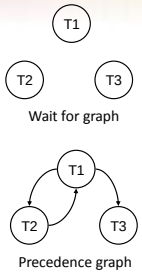
Schedule

T1 Read(X)
T2 Read(Y)
T1 Write(X)
T2 Read(X)
T3 Read(Z)
T3 Write(Z)
T1 Read(Y)
T3 Read(X)
T1 Write(Y)



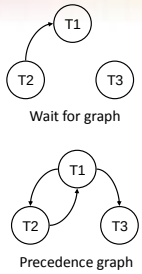
Example

Schedule	Locks
T1 Read(X)	write-lock(X)
T2 Read(Y)	read-lock(Y)
T1 Write(X)	
T2 Read(X)	
T3 Read(Z)	
T3 Write(Z)	
T1 Read(Y)	
T3 Read(X)	
T1 Write(Y)	



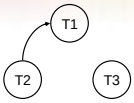
Example

Schedule	Locks
T1 Read(X)	**write-lock(X)**
T2 Read(Y)	read-lock(Y)
T1 Write(X)	
T2 Read(X)	**tries read-lock(X)**
T3 Read(Z)	
T3 Write(Z)	
T1 Read(Y)	
T3 Read(X)	
T1 Write(Y)	

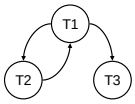


Example

Schedule	Locks
T1 Read(X)	write-lock(X)
T2 Read(Y)	read-lock(Y)
T1 Write(X)	
T2 Read(X)	tries read-lock(X)
T3 Read(Z)	write-lock(Z)
T3 Write(Z)	
T1 Read(Y)	read-lock(Y)
T3 Read(X)	
T1 Write(Y)	



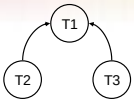
Wait for graph



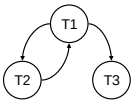
Precedence graph

Example

Schedule	Locks
T1 Read(X)	**write-lock(X)**
T2 Read(Y)	read-lock(Y)
T1 Write(X)	
T2 Read(X)	tries read-lock(X)
T3 Read(Z)	write-lock(Z)
T3 Write(Z)	
T1 Read(Y)	read-lock(Y)
T3 Read(X)	**tries read-lock(X)**
T1 Write(Y)	



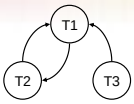
Wait for graph



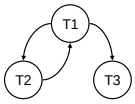
Precedence graph

Example

Schedule	Locks
T1 Read(X)	write-lock(X)
T2 Read(Y)	**read-lock(Y)**
T1 Write(X)	
T2 Read(X)	tries read-lock(X)
T3 Read(Z)	write-lock(Z)
T3 Write(Z)	
T1 Read(Y)	read-lock(Y)
T3 Read(X)	tries read-lock(X)
T1 Write(Y)	**tries write-lock(Y)**



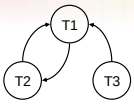
Wait for graph



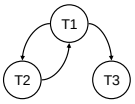
Precedence graph

Example

Schedule	Locks
T1 Read(X)	write-lock(X)
T2 Read(Y)	read-lock(Y)
T1 Write(X)	
T2 Read(X)	tries read-lock(X)
T3 Read(Z)	write-lock(Z)
T3 Write(Z)	
T1 Read(Y)	read-lock(Y)
T3 Read(X)	tries read-lock(X)
T1 Write(Y)	tries write-lock(Y)



Wait for graph



Precedence graph

A cycle in the wait-for graph means we will encounter deadlock

Deadlock Recovery

- Deadlocks can arise with 2PL
  - Deadlock is less of a problem than an inconsistent DB
  - We can detect and recover from deadlock
- Most DBMSs will detect deadlocks with a wait-for graph
  - Chose a single transaction as a 'victim' to rollback and restart
    - Which transaction has been running the longest?
    - Which transactions have made the most updates?
    - Which transactions have the most updates still to make?

Deadlock Prevention

- Conservative 2PL
  - All locks must be acquired before the transaction starts
  - Hard to predict what locks are needed
  - For high lock contention this works well
    - Transactions are never blocked once they start
  - For low lock contention this is not as effective
    - Locks are held longer than necessary
    - Transactions might hold on to locks for a long time, but not use them much

Timestamping

- Transactions can be run concurrently using a variety of techniques
- We looked at using locks to preserve isolation
- An alternative is timestamping
  - Requires less overhead in terms of tracking locks or detecting deadlock
  - Determines the order of transactions before they are executed
  - Most useful for a small number of transactions

Timestamping

- Each transaction has a timestamp, TS, and if T1 starts before T2 then TS(T1) < TS(T2)
  - Can use the system clock or an incrementing counter to generate timestamps
- Each resource has two timestamps
  - R(X), the largest timestamp of any transaction that has read X
  - W(X), the largest timestamp of any transaction that has written X

Timestamp Protocol

- If T tries to read X
  - If TS(T) < W(X) T is rolled back and restarted with a later timestamp
  - If TS(T) ≥ W(X) then the read succeeds and we set R(X) to be max(R(X), TS(T))
- T tries to write X
  - If TS(T) < W(X) or TS(T) < R(X) then T is rolled back and restarted with a later timestamp
  - Otherwise the write succeeds and we set W(X) to TS(T)

Timestamping Example

- Given T1 and T2 we will assume
  - The transactions make alternate operations
  - Timestamps are allocated from a counter starting at 1
  - T1 goes first

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

Timestamp Example

→ T1
Read(X)
Read(Y)
Y = Y + X
Write(Y)

→ T2
Read(X)
Read(Y)
Z = Y - X
Write(Z)

	X	Y	Z
R			
W			

	T1	T2
TS	1	2

Timestamp Example

T1
→ Read(X)
Read(Y)
Y = Y + X
Write(Y)

→ T2
Read(X)
Read(Y)
Z = Y - X
Write(Z)

	X	Y	Z
R	1		
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
→ Read(X)	→ Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	**2**		
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	→ Read(X)
→ Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	**1**	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
→ Read(Y)	→ Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	**2**	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	→ Read(Y)
→ Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
→ Y = Y + X	→ Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	→ Z = Y - X
→ Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

	T1	T2
→	Read(X)	Read(X)
	Read(Y)	Read(Y)
	Y = Y + X	→ Z = Y - X
	Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	**3**	2

Timestamp Example

	T1	T2
→	Read(X)	Read(X)
	Read(Y)	Read(Y)
	Y = Y + X	Z = Y - X
	Write(Y)	→ Write(Z)

	X	Y	Z
R	2	2	
W			**2**

	T1	T2
TS	3	2

Timestamp Example

	T1	T2
→	Read(X)	Read(X)
	Read(Y)	Read(Y)
	Y = Y + X	Z = Y - X
	Write(Y)	Write(Z)

	X	Y	Z
R	**3**	2	
W			2

	T1	T2
TS	3	2

Timestamp Example

	T1	T2
	Read(X)	Read(X)
→	Read(Y)	Read(Y)
	Y = Y + X	Z = Y - X
	Write(Y)	Write(Z)

	X	Y	Z
R	3	**3**	
W			2

	T1	T2
TS	3	2

Timestamp Example

	T1	T2
	Read(X)	Read(X)
	Read(Y)	Read(Y)
→	Y = Y + X	Z = Y - X
	Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W			2

	T1	T2
TS	3	2

Timestamp Example

	T1	T2
	Read(X)	Read(X)
	Read(Y)	Read(Y)
	Y = Y + X	Z = Y - X
→	Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W		**3**	2

	T1	T2
TS	3	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W		3	2

	T1	T2
TS	3	2

Timestamping

- The protocol means that transactions with higher times take precedence when conflict arises
  - No deadlock
  - When no conflict arises lower timestamps proceed first
- Timestamping guarantees a schedule is conflict serialisable
- Problems
  - Long transactions might keep getting restarted by new transactions - starvation
  - Rolls back old transactions, which may have done a lot of work

This Lecture in Exams

Explain, using an example, how deadlock may occur between two transactions utilising two-phase locking protocol

Describe how a DBMS might attempt to prevent deadlock from occurring, and how a DMBS might recover from deadlock that has already occured

Describe how timestamping can be used as an alternative to two-phase locking, to provide concurrent access to database resources