

CS 347:  
Distributed Databases and  
Transaction Processing  
**Notes06: Concurrency Control**

Hector Garcia-Molina

CS 347

Notes06

1

Overview

- Concurrency Control
  - Schedules and Serializability
  - Locking
  - Timestamp Control
- Failure Recovery
  - next set of notes...

CS 347

Notes06

2

Schedule

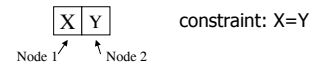
- Just like in a centralized system, a schedule represents how a set of transactions were executed
- Schedules may be "good" or "bad" (preserve constraints)

CS 347

Notes06

3

Example



	$T_1$		$T_2$
1	$(T_1) a \leftarrow X$	5	$(T_2) c \leftarrow X$
2	$(T_1) X \leftarrow a+100$	6	$(T_2) X \leftarrow 2c$
3	$(T_1) b \leftarrow Y$	7	$(T_2) d \leftarrow Y$
4	$(T_1) Y \leftarrow b+100$	8	$(T_2) Y \leftarrow 2d$

↓ Precedence relation

CS 347

Notes06

4

Schedule S1

Precedence: intra-transaction ↓  
inter-transaction ↓

	(node X)		(node Y)
1	$(T_1) a \leftarrow X$		
2	$(T_1) X \leftarrow a+100$		
5	$(T_2) c \leftarrow X$	3	$(T_1) b \leftarrow Y$
6	$(T_2) X \leftarrow 2c$	4	$(T_1) Y \leftarrow b+100$
		7	$(T_2) d \leftarrow Y$
		8	$(T_2) Y \leftarrow 2d$

If X=Y=0 initially, X=Y=200 at end (always good?)

CS 347

Notes06

5

Definition of Schedule

Let  $T = \{T_1, T_2, T_n\}$  be a set of transactions.  
A schedule  $S$  over  $T$  is a partial order with ordering relation  $<_S$  where:

- (1)  $S = \bigcup_{i=1}^N T_i$
- (2)  $<_S \supseteq \bigcup_{i=1}^N <_i$
- (3) for any two conflicting operations  $p, q \in S$ , either  $p <_S q$  or  $q <_S p$

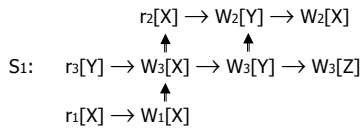
CS 347

Notes06

6

### Example

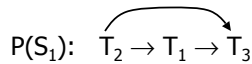
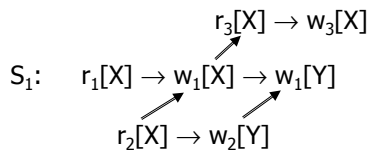
(T<sub>1</sub>) r<sub>1</sub>[X] → W<sub>1</sub>[X]  
 (T<sub>2</sub>) r<sub>2</sub>[X] → W<sub>2</sub>[Y] → W<sub>2</sub>[X]  
 (T<sub>3</sub>) r<sub>3</sub>[X] → W<sub>3</sub>[X] → W<sub>3</sub>[Y] → W<sub>3</sub>[Z]



### Definition of P(S)

- The precedence graph for schedule S, P(S), is a directed graph where
  - nodes: the transactions in S
  - edges: T<sub>i</sub> → T<sub>j</sub> is an edge IFF
    - ∃ p ∈ T<sub>i</sub>, q ∈ T<sub>j</sub> such that p, q conflict and p <<sub>S</sub> q

### Example



### Serializability Theorem

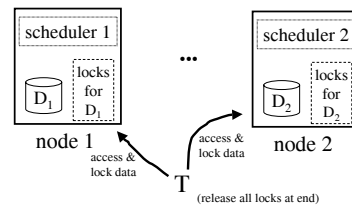
**Theorem:** A schedule S is serializable IFF P(S) is acyclic.

### Enforcing Serializability

- Locking
- Timestamps

### Locking

- Just like in a centralized system...
- But with multiple lock managers



## Locking Rules

- Well-formed transactions
- Legal schedulers
- Two-phase transactions
  
- These rules guarantee serializable schedules

CS 347

Notes06

13

## What about

- Locking in a shared-memory architecture?
- Locking in a shared-disks architecture?

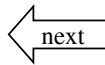
CS 347

Notes06

14

## Overview

- Concurrency Control
  - Schedules and Serializability
  - Locking
  - Timestamp Control
- Failure Recovery
  - next set of notes...



CS 347

Notes06

15

## Timestamp Ordering Schedulers

- Basic idea:
  - assign timestamp as transaction begins
  - if  $ts(T_1) < ts(T_2) \dots < ts(T_n)$ , then scheduler produces history equivalent to  $T_1, T_2, T_3, T_4, \dots T_n$

CS 347

Notes06

16

## TO Rule

If  $p_i[x]$  and  $q_j[x]$  are conflicting operations, then  $p_i[x]$  is executed before  $q_j[x]$

$$(p_i[x] <_S q_j[x])$$

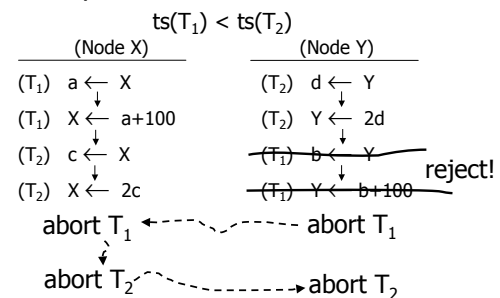
$$\text{IFF } ts(T_i) < ts(T_j)$$

CS 347

Notes06

17

## Example: a non-serializable schedule $S_2$



CS 347

Notes06

18

## Strict T.O.

- Lock written items until it is certain that writing transaction has been successful (avoid cascading rollbacks)

CS 347

Notes06

19

## Example Revisited

$ts(T_1) < ts(T_2)$

(Node X)	(Node Y)
(T <sub>1</sub> ) a ← X	(T <sub>2</sub> ) d ← Y
(T <sub>1</sub> ) X ← a+100	(T <sub>2</sub> ) Y ← 2d
(T <sub>2</sub> ) c ← X	(T <sub>1</sub> ) <del>b ← Y</del> reject!
abort T <sub>1</sub>	← abort T <sub>1</sub>
(T <sub>2</sub> ) c ← X	
(T <sub>2</sub> ) X ← 2c	

CS 347

Notes06

20

## Enforcing T.O.

- For each data item X:
  - MAX\_R[X]: maximum timestamp of a transaction that read X
  - MAX\_W[X]: maximum timestamp of a transaction that wrote X
  - rL[X]: # of transactions currently reading X (0,1,2,...)
  - wL[X]: # of transactions currently writing X (0 or 1)

CS 347

Notes06

21

## T.O. Scheduler - Part 1

ri[X] arrives  
 IF  $ts(T_i) < MAX\_W[X]$  THEN ABORT  $T_i$   
 ELSE [ IF  $ts(T_i) > MAX\_R[X]$  THEN  $MAX\_R[X] \leftarrow ts(T_i)$ ;  
 IF queue is empty AND  $wL[X] = 0$  THEN  
 $[rL[X] \leftarrow rL[X] + 1; \text{START READ OF } X]$   
 ELSE add (r,  $T_i$ ) to queue ]

CS 347

Notes06

22

## T.O. Scheduler - Part 2

wi[X] arrives  
 IF  $ts(T_i) < MAX\_W[X]$  OR  $ts(T_i) < MAX\_R[X]$   
 THEN ABORT  $T_i$   
 ELSE [  $MAX\_W[X] \leftarrow ts(T_i)$ ;  
 IF queue is empty AND  $wL[X] = 0$  AND  $rL[X] = 0$   
 THEN  $[wL[X] \leftarrow 1; \text{WRITE } X;$   
 WAIT FOR  $T_i$  TO FINISH ]  
 ELSE add (w,  $T_i$ ) to queue ]

CS 347

Notes06

23

## T.O. Scheduler - Part 3

When o finishes (o is r or w) on X  
 $oL[X] \leftarrow oL[X] - 1; \text{NDONE} \leftarrow \text{TRUE}$   
 WHILE NDONE DO  
 [ let head of queue be (q,  $T_j$ ); (*smallest timestamp*)  
 IF  $q = w$  AND  $rL[X] = 0$  AND  $wL[X] = 0$  THEN  
 $[ \text{remove } (q, T_j); wL[X] \leftarrow 1;$   
 WRITE X AND WAIT FOR  $T_j$  TO FINISH ]  
 ELSE IF  $q = r$  AND  $wL[X] = 0$  THEN  
 $[ \text{remove } (q, T_j); rL[X] \leftarrow rL[X] + 1; \text{START READ OF } X]$   
 ELSE NDONE  $\leftarrow$  FALSE ]

CS 347

Notes06

24

### Note about the code

for reads:  $[rL[X] \leftarrow rL[X]+1; \text{ START READ OF } X]$

for writes:  $[wL[X] \leftarrow 1; \text{ WRITE } X; \text{ WAIT FOR } T_i \text{ TO FINISH } ]$

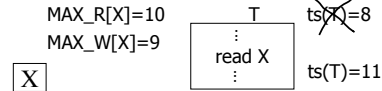
Meaning: In Part 3, the end of a write is only processed when all writes for its transaction have completed.

CS 347

Notes06

25

- If a transaction is aborted, it must be retired with a new, larger timestamp



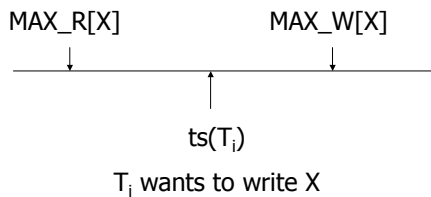
- Starvation possible

CS 347

Notes06

26

### Thomas Write Rule



CS 347

Notes06

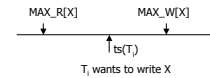
27

### Change in T.O. Scheduler:

#### When W<sub>i</sub>[X] arrives

```

IF ts(Ti) < MAX_R[X] THEN ABORT Ti
ELSE IF ts(Ti) < MAX_W[X] THEN
  [IGNORE THIS WRITE (tell Ti it was OK)]
ELSE [process write as before...
  MAX_W[X] ← ts(Ti);
  IF queue is empty AND wL[X]=0 AND rL[X]=0 THEN
    [wL[X] ← 1; WRITE X and WAIT FOR Ti TO FINISH]
  ELSE add (W, Ti) to queue]
  
```

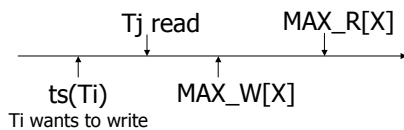


CS 347

Notes06

28

### Question



- Why can't we let T<sub>i</sub> go ahead?  
 ↳ MAX\_R[X] is only the latest read; there could be a T<sub>j</sub> read as shown...

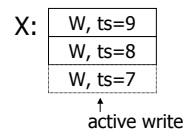
CS 347

Notes06

29

### Optimizations

- Update MAX\_R, MAX\_W when action executed, not when action put on queue
- Example: MAX\_W[X]=9 or 7?



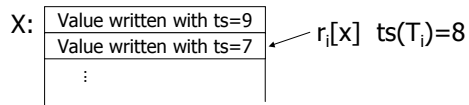
CS 347

Notes06

30

## Optimizations

- Use multiple versions of data



CS 347

Notes06

31

## 2PL $\neq$ TO

$T_1: w_1[Y]$   
 $T_2: r_2[X] r_2[Y] w_2[Z]$       $ts(T_1) < ts(T_2) < ts(T_3)$   
 $T_3: w_3[X]$

S:  $r_2[X] w_3[X] w_1[Y] r_2[Y] w_2[Z]$

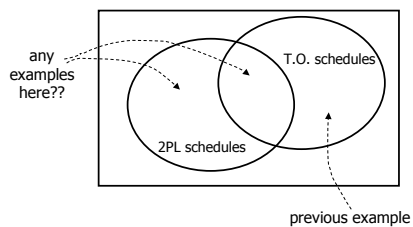
- S could be produced with T.O. but not with 2PL

CS 347

Notes06

32

## Are all 2PL schedules T.O.?



CS 347

Notes06

33

## Theorem

If S is a schedule representing an execution by a T.O. scheduler, then S is serializable

### Proof:

- say  $T_i \rightarrow T_j$  in  $P(S)$   
 $\Rightarrow \exists$  conflicting  $p_i[x], q_j[x]$  in S,  
 such that  $p_i[x] <_S q_j[x]$   
 Then by T.O. rule,  $ts(T_i) < ts(T_j)$

CS 347

Notes06

34

## Proof - Continued

- Say there is a cycle  
 $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$  in  $P(S)$   
 then:  
 $ts(T_1) < ts(T_2) < ts(T_3) \dots < ts(T_n) < ts(T_1)$   
 A contradiction!
- So  $P(S)$  is acyclic  
 $\Rightarrow S$  is serializable

CS 347

Notes06

35

## Timestamp management

Item	data	MAX_R	MAX_W
X1			
X2			
⋮	⋮	⋮	⋮
Xn			

- too much space!  
- more IO

CS 347

Notes06

36

## Timestamp cache

Item	MAX_R	MAX_W
X		
Y		
⋮		
Z		

tsMIN

- If transaction reads or writes X, make entry in cache for X (add row if not in)
- Periodically purge all items X with  $MAX\_R[X] < tsMIN$ ,  $MAX\_W[X] < tsMIN$  and remember tsMIN (choose  $tsMIN \approx \text{current time} - d$ )

CS 347

Notes06

37

## Timestamp cache

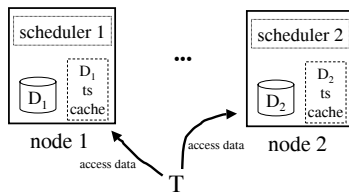
- To enforce T.O. rule for  $pi[X]$ 
  - if X entry in cache, use MAX\_R, MAX\_W values in cache
  - else assume  $MAX\_R[X]=tsMIN$ ,  $MAX\_W[X]=tsMIN$
- Use hashing (on items) for cache (same as lock table)

CS 347

Notes06

38

## Distributed T.O. Scheduler



- Each scheduler is "independent"
- At end of transaction, signal all schedulers involved to release all  $wL[X]$  locks

CS 347

Notes06

39

## Summary

- 2PL
  - the most popular
  - useful in a distributed system
  - deadlocks possible
  - several variations
- T.O.
  - good for multiple versions
  - aborts more likely
  - no deadlocks
  - useful in a distributed system

CS 347

Notes06

40

- Others concurrency control schemes e.g., Certifiers, serialization graph testing

↑  
 hard to implement in a distributed system

↑  
 not very practical need global data structure

CS 347

Notes06

41