Overview

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

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)

Example

\[
\begin{array}{c|c}
\text{Node 1} & \text{Node 2} \\
\hline
T_1 & T_2 \\
1 & 5 \\
(T_1) a \leftarrow X & (T_2) c \leftarrow X \\
2 & 6 \\
(T_1) X \leftarrow a + 100 & (T_2) X \leftarrow 2c \\
3 & 7 \\
(T_1) b \leftarrow Y & (T_2) d \leftarrow Y \\
4 & 8 \\
(T_1) Y \leftarrow b + 100 & (T_2) Y \leftarrow 2d \\
\end{array}
\]

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

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. \( \forall_{i}^{n} \subseteq_{i} <_i \)
3. for any two conflicting operations \( p, q \in S \), either \( p <_S q \) or \( q <_S p \)

Let \( X = Y = 0 \) initially, \( X = Y = 200 \) at end (always good?)
Example

(T1) \( r_1[X] \rightarrow W_1[X] \)
(T2) \( r_2[X] \rightarrow W_2[Y] \rightarrow W_2[X] \)
(T3) \( r_3[X] \rightarrow W_3[X] \rightarrow W_3[Y] \rightarrow W_3[Z] \)

\( r_2[X] \rightarrow W_2[Y] \rightarrow W_2[X] \)

S1: \( r_1[Y] \rightarrow W_1[X] \rightarrow W_1[Y] \rightarrow W_1[Z] \)
\( r_2[X] \rightarrow W_1[X] \)

Definition of \( P(S) \)

- The precedence graph for schedule \( S \), \( P(S) \), is a directed graph where
  - nodes: the transactions in \( S \)
  - edges: \( T_i \rightarrow T_j \) is an edge IFF
    \( \exists p \in T_i, q \in T_j \) such that
    \( p, q \) conflict and \( p \prec_S q \)

Example

\( r_3[X] \rightarrow w_3[X] \)

\( r_1[X] \rightarrow w_1[X] \rightarrow w_1[Y] \)
\( r_2[X] \rightarrow w_2[Y] \)

P(\( S_1 \)): \( T_2 \rightarrow T_1 \rightarrow T_3 \)

Enforcing Serializability

- Locking
- Timestamps

Serializability Theorem

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

Locking

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

![Diagram showing lock managers on different nodes]

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

What about
• Locking in a shared-memory architecture?
• Locking in a shared-disks architecture?

Locking with Shared Memory
• Where does lock table live?
• How do we avoid race conditions?

Locking with Shared Disks
• Where does lock table live?
• How do we avoid race conditions?

Shared Disk Locking, Cases to Discuss
• Control of Data Partitioned; fixed partition function known by everyone
• Dynamic partition of control
  - For each DB object i we need
    • LT(i): lock table entry for i (transaction that has lock, waiting transactions, lock mode, etc...)
    • W(i): at what processor is LT(i) currently?
  - Replicate W(i) at all processors (why?)
  - Need replicated data management scheme!
Overview

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

Timestamp Ordering Schedulers

• Basic idea:
  - assign timestamp as transaction begins
  - if ts(T1) < ts(T2) ... < ts(Tn), then scheduler produces history equivalent to T1,T2,T3,T4, ... Tn

TO Rule

If p[x] and q[x] are conflicting operations, then p[x] is executed before q[x] (p[x] \( \leq_S \) q[x])
IFF ts(Ti) < ts(Tj)

Example: a non-serializable schedule S2

\[
\begin{align*}
(T_1) &: a \leftarrow X \\
(T_1) &: X \leftarrow a+100 \\
(T_2) &: c \leftarrow X \\
(T_2) &: X \leftarrow 2c \\
(T_1) &: a \leftarrow X \\
(T_1) &: X \leftarrow a+100 \\
(T_2) &: c \leftarrow X \\
(T_2) &: X \leftarrow 2c \\
(T_1) &: b \leftarrow Y \\
(T_1) &: Y \leftarrow b+100 \\
(T_2) &: b \leftarrow Y \\
(T_2) &: Y \leftarrow b+100 
\end{align*}
\]

Example: a non-serializable schedule S2

\[
\begin{align*}
(T_1) &: a \leftarrow X \\
&T_1 &: d \leftarrow Y \\
(T_2) &: c \leftarrow X \\
&T_2 &: X \leftarrow 2c \\
&T_2 &: b \leftarrow Y \\
&T_2 &: Y \leftarrow b+100 \\
&T_2 &: b \leftarrow Y \\
&T_2 &: Y \leftarrow b+100 
\end{align*}
\]

Example: a non-serializable schedule S2

\[
\begin{align*}
(T_1) &: a \leftarrow X \\
(T_1) &: X \leftarrow a+100 \\
&T_1 &: c \leftarrow X \\
&T_1 &: X \leftarrow 2c \\
&T_1 &: b \leftarrow Y \\
&T_1 &: Y \leftarrow b+100 \\
&T_2 &: b \leftarrow Y \\
&T_2 &: Y \leftarrow b+100 
\end{align*}
\]

Example: a non-serializable schedule S2

\[
\begin{align*}
(T_1) &: a \leftarrow X \\
&T_1 &: d \leftarrow Y \\
&T_1 &: c \leftarrow X \\
&T_1 &: X \leftarrow 2c \\
&T_1 &: b \leftarrow Y \\
&T_1 &: Y \leftarrow b+100 \\
&T_2 &: b \leftarrow Y \\
&T_2 &: Y \leftarrow b+100 
\end{align*}
\]
Strict T.O.

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

Example Revisited

\[ts(T_1) < ts(T_2)\]

(T1) \( a \leftarrow X \)
(T1) \( X \leftarrow a + 100 \)
(T2) \( c \leftarrow X \) delay
(T2) \( X \leftarrow 2c \)

(T1) \( X \leftarrow a + 100 \)
(T2) \( X \leftarrow 2d \)

(T1) \( X \leftarrow a \) delay
(T2) \( X \leftarrow b \) abort T1

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)

T.O. Scheduler - Part 1

\(r[X]\) arrives
  IF \(ts(T_i) < MAX_W[X]\) THEN ABORT Ti
  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 \((r, Ti)\) to queue]

T.O. Scheduler - Part 2

\(w[X]\) arrives
  IF \(ts(T_i) < MAX_W[X]\) OR \(ts(T_i) < MAX_R[X]\)
  THEN ABORT Ti
  ELSE [ \(MAX_W[X] \leftarrow ts(T_i)\);
    IF queue is empty AND \(wL[X]=0\) AND \(rL[X]=0\)
    THEN \([wL[X] \leftarrow 1; \text{WRITE X; WAIT FOR Tj TO FINISH}]\)
    ELSE add \((w, Ti)\) to queue ]
When \( o \) finishes (\( o \) is \( r \) or \( w \)) on \( X \)

\[
\text{oL}[X] \leftarrow \text{oL}[X] - 1; \text{NDONE} \leftarrow \text{TRUE}
\]

WHILE NDONE DO

[ let head of queue be (\( q, T_i \)); (smallest timestamp) ]

IF \( q = w \) AND \( \text{rl}[X] = 0 \) AND \( \text{wl}[X] = 0 \) THEN

\[
\text{remove (} q, T_i \text{)}; \ wL[X] \leftarrow 1; \text{WRITE X AND WAIT FOR } T_i \text{ TO FINISH }
\]
ELSE IF \( q = r \) AND \( \text{wl}[X] = 0 \) THEN

\[
\text{remove (} q, T_i \text{)}; \ rL[X] \leftarrow rL[X] + 1; \text{START READ OF } X
\]
ELSE NDONE \( \leftarrow \) FALSE

---

**Note about the code**

For reads:

\[
[\text{rl}[X] \leftarrow \text{rl}[X] + 1; \text{START READ OF } X]
\]

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.

---

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

\[
\text{MAX}_{R}[X] = 10 \\
\text{MAX}_{W}[X] = 9
\]

\[
\text{X}
\]

\[
\text{read } X
\]

\[
\text{ts}(T) = 8
\]

\[
\text{T}
\]

---

Starvation possible

---

**Thomas Write Rule**

\[
\text{MAX}_{R}[X] \quad \text{MAX}_{W}[X]
\]

\[
\text{ts}(T_i) \\
\text{T_i wants to write } X
\]

---

**Change in T.O. Scheduler:**

When \( Wi[X] \) arrives

\[
[\text{MAX}_{R}[X] \quad \text{MAX}_{W}[X]]
\]

\[
\text{ts}(T_i) \\
\text{T_i wants to write } X
\]

IF \( \text{ts}(T_i) < \text{MAX}_{R}[X] \) THEN ABORT \( T_i \)
ELSE IF \( \text{ts}(T_i) < \text{MAX}_{W}[X] \) THEN

\[
[\text{IGNORE THIS WRITE (tell } T_i \text{ it was } OK)]
\]
ELSE [process write as before...]

\[
\text{MAX}_{W}[X] \leftarrow \text{ts}(T_i);
\]

IF queue is empty AND \( \text{wl}[X] = 0 \) AND \( \text{rl}[X] = 0 \) THEN

\[
[wL[X] \leftarrow 1; \text{WRITE } X \text{ and WAIT FOR } T_i \text{ TO FINISH}]
\]
ELSE add (\( W, T_i \)) to queue]

\* Ignore if transaction that wrote \( \text{MAX}_{W}[X] \) did not abort
Question

MAX_W[X]

MAX_R[X]

ts(Ti)

Ti wants to write

• Why can’t we let Ti go ahead?

Question

Tj read

MAX_W[X]

MAX_R[X]

ts(Ti)

Ti wants to write

• Why can’t we let Ti go ahead?

¬ MAX_R[X] is only the latest read; there could be a Tj read as shown...

Optimizations

• Update MAX_R, MAX_W when action executed, not when action put on queue

Example:

MAX_W[X]=9 or 7?

X:

W, ts=9

W, ts=8

W, ts=7

↑ active write

Optimizations

• Use multiple versions of data

Value written with ts=9

Value written with ts=7

r[X] ts(Tj)=8

2PL ≠ TO

T1: w1[Y]
T2: r2[X] r2[Y] w2[Z]  ts(T1)<ts(T2)<ts(T3)
T3: w3[X]

S: r2[X] w1[X] w1[Y] r2[Y] w2[Z]

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

Are all 2PL schedules T.O.?

T.O. schedules

2PL schedules

previous example

any examples here??
**Theorem**

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

**Proof:**

1) say Ti → Tj in P(S)  
   ⇒ ∃ conflicting pi[x], qj[x] in S,  
   such that p[x] <S q[x]  
   Then by T.O. rule, ts(Ti)< ts(Tj)

2) Say there is a cycle  
   T1 → T2 → ...Tn → T1 in P(S)  
   then:  
   ts(T1)<ts(T2) <ts(T3)...<ts(Tn)<ts(T1)  
   A contradiction!

3) So P(S) is acyclic  
   ⇒ S is serializable

---

**Timestamp management**

<table>
<thead>
<tr>
<th>Item</th>
<th>data</th>
<th>MAX_R</th>
<th>MAX_W</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xn</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- too much space!  
- more I/O

**Timestamp cache**

- 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 = current time - d)

**Distributed T.O. Scheduler**

- Each scheduler is “independent”  
- At end of transaction, signal all schedulers involved to release all wL[X] locks
Distributed T.O. Scheduler

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

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

Others concurrency control schemes
- e.g., Certifiers, serialization graph testing
  - hard to implement in a distributed system
  - not very practical
  - need global data structure

...