Chapter 4.4 Synchronization and Lock Management

(R. Bayer, ECI 76)

Chapter 4.4.1 **Data Structures** for Lock Management

Compatibility - matrices, e.g.

	R	A	X
R	+	+	-
A	+	-	-
X	-	-	-

Lock-Request: (t, r, μ)

t : Transaction
r : general ressource, e.g. o
μ : lock type, e.g. $\mu \in \{R, A, X\}$

Granting of Locks: data structure organized according to ressources:

G (r) : set of locks granted for r
{(t, μ) ... } "granted group"

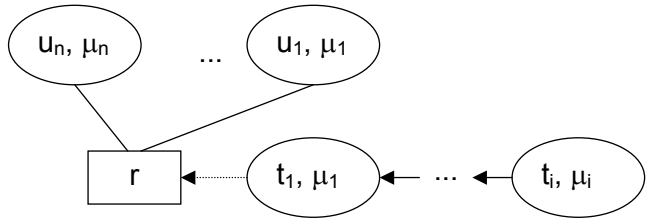
Q(r) : wait queue of requests

Operations:

G(r) := G(r) $\cup$ {(t, μ)}
:= G(r) \ {(t, μ)}

Q(r) := Q(r) **append** (t,μ)
:= Q(r) **remove** (t, μ)

G (r) :
Q(r)



At time of request (t_i, r, μ)

B_i := set of transactions, for which t_i must wait, which block t_i

B_i := $\{t_i \mid [(t_i, \nu) \in (G(r) \vee (t_j, \nu)$ vor (t_i, μ) in Q(r)] $\wedge$ ν,μ not compatible}
def

i.e. FIFO

Def: $t_i \; w \; t_k : \Leftrightarrow t_k \in B_i$

$\Leftrightarrow$

"t_i waits for t_k"
w: waits for, wait relation
wait graph $G_w = (T, w)$
T: set of transactions presently in system.

Question: relationship between $\rightarrow$ and w?

Deadlock:
cyclic wait,
cycle in G_w
let transitive, not reflexive hull be w^+
deadlock $\equiv \exists \; t : (t,t) \in w^+$

Management of w, G_w : simple

Computing of w^+

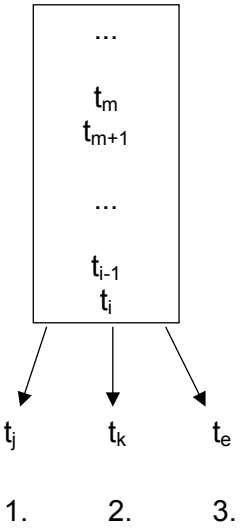
$O(n^3)$ according to Warshall, n Transactions
$O(n \cdot m)$ according to Bayer, Purdom, m edges

Note: m small, if B_i small,
goal: m = O (n)

Chapter 4.4.2 Trans. Hull in O(n • m) : Sketch

Depth-first search in G_w with nodes = trans.
let t_i be top stack element with successors in one of the following states:

1. not yet processed
2. in stack, at known position
3. old, all successors w.r. to w^+ already known



1. push t_j
2. form equivalence class t_m ... t_i t_k
   (union find algorithm.);
   remove $t_{m+1}...t_it_k$ from stack;
   Nachf (t_m) := $\cup$ Nachf(t)
   $t \in \{t_m \; t_{m+1}, \ldots, t_i, t_k\}$
3. Nachf (t_i) := Nachf (t_i) $\cup$ Nachf (t_e)
4. pop (t_i) : Nachf (t_{i-1}) := Nachf (t_{i-1}) $\cup$ Nachf (t_i)

Analysis: per edge in w at most n single operations for $\cup$ => = $\boldsymbol{O}(n \cdot m)$

Chapter 4.4.3 Transitive Hull on-line

only updating of w^+, i.e.
given w, w^+
compute w', w'^+, where
$w' := w \cup \{(t_i, t_j)\}$ or
$w' := w \setminus \{(t_i, t_j)\}$

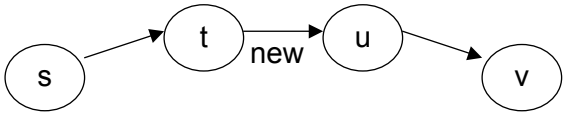
Special case: acyclic G_w, i.e.

cycles = deadlocks of transactions are broken immediately!

Matrix M: with

M [t, u] := number of different paths from t to u in G_w
$t\ w^+\ u \Leftrightarrow r\ M\ [t, u] > O$

Updating of M on insert or delete of (t, n) in w



M [s, v] := M [s, v] + M [s, t] • M [u, v] for $s \neq t, u \neq v$
M [s, u] := M [s, u] + M [s, t] for $s \neq t, u = v$
M [t, v] := M [t, v] + M [u, v] for $u \neq v, s = t$
M [t, u] := M [t, u] + 1 for $s = t, u = v$

=> $\boldsymbol{O}$ (n^2)

Updating of M at **endtrans** t:

t is sink in G_w
remove line t and column t from G_w

Deadlock: M [t', t'] > 0

Removal by **abort** t
or **abort** u
or cheapest cut set

at time of lock request: (t, r, μ):
t was sink in G_w

Potential measures:

1. move t forward in Q(r),
   no more deadlocks if $t \in G_r$
   i.e. disregard FIFO in case of deadlock
2. **Abort** t
3. $\forall$ t $\in$ minimal-cost cut-set
   abort t

Preventing Starvation

t carries time stamp of start AS(t)
use w^+ to define priority classes:

let $\overline{t_1}$ be oldest transaction, i.e. with smallest AS ($\overline{t_1}$)

$P_1 := \{\overline{t_1}\} \cup \{t \mid \overline{t_1}\ w^+\ t\}$

general definition:

$U_0 := \varnothing$

$\overline{t_i}$:= oldest transaction in $T \setminus U_{i-1}$

$P_i := \{\overline{t_1}\} \cup \{t \mid \overline{t_i}\, w^+\, t\} \setminus U_{i-1}$

$U_i := U_{i-1} \cup P_i$, i.e. $U_i := \bigcup_{j=1}^{i} P_j$

Strategies against Starvation

Str.1: use P_i for scheduling

Str. 2: put all t into state "waiting", except for $t \in P_1$, i.e. no new t' are added

Str. 3: let $\overline{t_1} \in Q(r)$, i. e. "waiting" for $t \in P_1$
$t \in Q(r)$: $\overline{t_1}$ may pass t
$t \in G(r)$: $\overline{t_1}$ takes away r from t, if t waits (because of r')

...

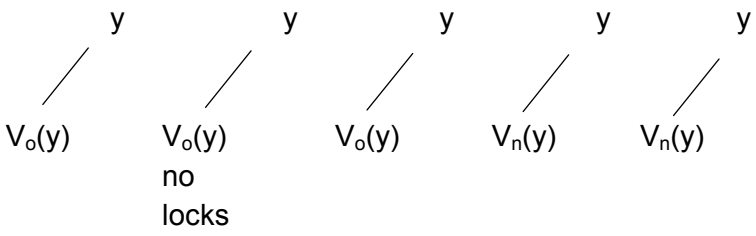
Str. x: $\overline{t_1}$ may not be passed or aborted

Chapter 4.5 Lock Modes and Synchronization

	R	X
R	+	-
X	-	-

$R \equiv S$

States of a Data Object



Wait-Situation for t:

X-lock: (t, y, X) if there is R-lock on y
R-lock: (t, y, R) if there is X-lock on y

Disadvantages of R, X:

1. UPDATEs are delayed by READS
2. READs are delayed by UPDATE
3. UPDATEs are delayed by UPDATE
4. deadlocks

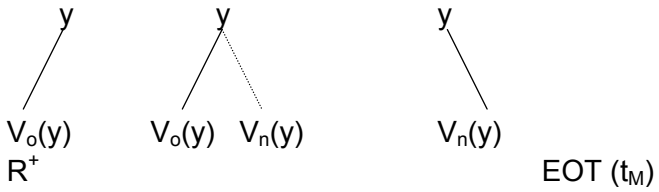
Example

t_L : R-lock on $V_o(z)$, $V_o(y)$, $V_o(w)$

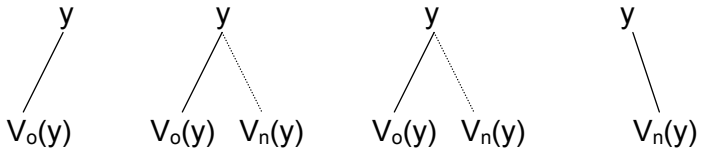
t_M : X-lock on $V_o(w)$, $V_o(z)$

$V_o(z)$ $V_o(y)$ $V_o(w)$

UPDATE with Safety Copies: t_M



UPDATE with A-locks: R, A, X

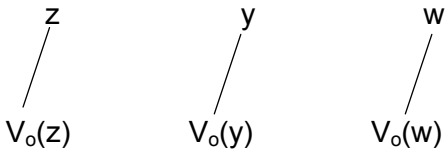


Shadow-versions and shadow technique

Compatibility:

	R	A	X
R			
A			
X			

Previous Example: t_L : R-lock on z, y, w
t_M : A-lock on w, z



Advantages:

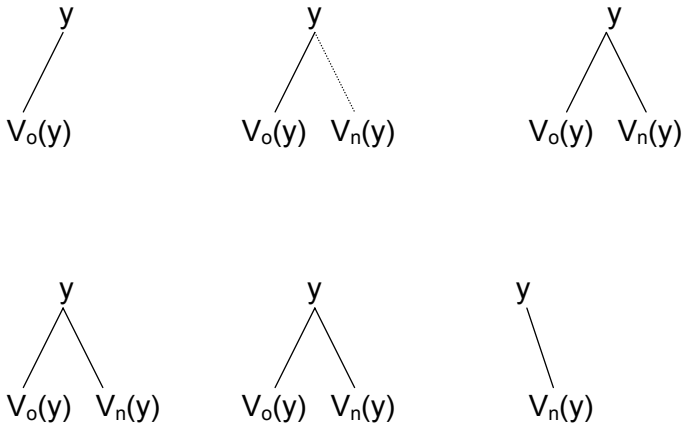
1. t_L, t_M executable in parallel
2. high data availability for readers
3. fewer deadlocks
4. in general fewer wait situations
   here: commit t_M ⟶ commit t_L

Update Protocol with R, A, X

1. A-lock on Y
2. Analyze $V_o(y)$, construct $V_n(y)$
3. convert A ⇢ X
4. update catalogue
5. remove X-lock

} atomic?

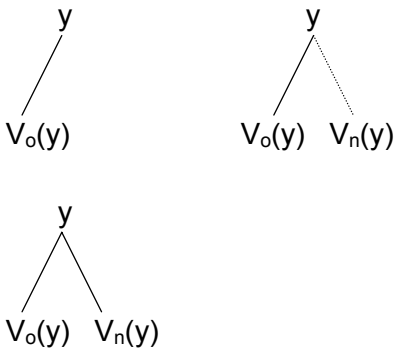
R, A, C Protocol



Compatibility

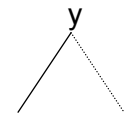
	R	A	C
R			
A			
C			

Read Requests t_L



New Semantics for G_w: with R, A, C

1. Serializability Edges: for R-lock



at time of lock request (t_M, y, A)

V_o	V_n
R*	A
t_1	t_M
...	
t_e	

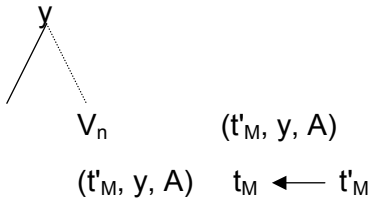
⟵ edges must be inserted in G_w,
t_M must come after t_i : i = 1, 2, ..., l
in serialization, i. e.
⟶ is not a true wait edge

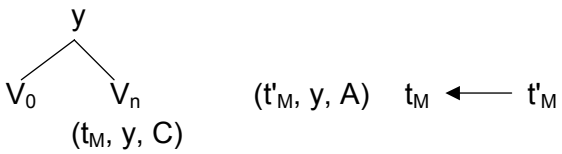
Cycle Check: now or later at A ⇢ C

2. Wait Edges: for incompatible lock requests, e.g.
A with A
C with A

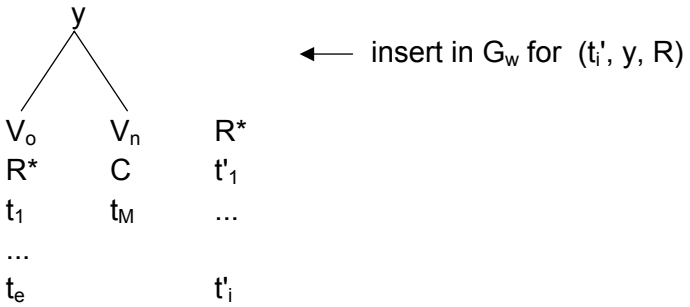
Note: A with C and C with A do not occur, if protocol is observed

	R	A	C
R	+	+	+
A	+	-	%
C	+	-	%





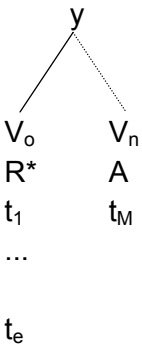
3. Serialization Edges:
when C-lock is set



Note: If cycle is caused by $t'_i \rightarrow t_M$
granting of (t_i', y, R) on $V_o(y)$ is possible

Theorem: With R, A, C - protocol read request (t, y, R) can be granted immediately, i.e. read requests do not cause deadlocks or waiting.

4. Serialization- or Wait-Edges



when A-lock is already set

for new request (t_M, y, R)

1. t'_i reads V_o, i.e. $t_M \rightarrow t'_i$
   i.e. serialization edge (cycle search)
2. when cycle is caused
   $t'_i \rightarrow t_M$,
   i.e. wait edge, but no deadlock,
   until t_M has **committed**.