7. Exercise

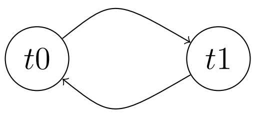
Timo Bergerbusch 344408
Thomas Näveke 311392
Shu Xing 381176

22.01.2018

Exercise 7.1(Schedules, Serializability, and Locking)

7.1.1

The schedule is not conflict serializable, because its corresponding conflict graph is cyclic.
With the $conflict(s_0)$=$\{(w_0(A), r_1(A)), (r_1(B), w_0(B))\}$:

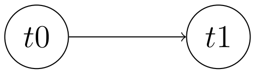


7.1.2

Using 2PL, we need to make sure that $wl_i(X) < wu_i(Y), i \in \{0,1\}, X, Y \in \{A, B\}$.
So we got the following schedule s':

t_0	t_1
$wl_0(A)$	
$r_0(A)$	
$w_0(A)$	
	$wl_1(A) \to blocks$
$wl_0(B)$	
$r_0(B)$	
$w_0(B)$	
$wu_0(A)$	
$wu_0(B)$	
c_0	
	$wl_1(A) \to granted$
	$r_1(A)$
	$wl_1(B)$
	$r_1(B)$
	$wu_1(A)$
	$wu_1(B)$
	c_1

where the $DT(s') = r_0(A)w_0(A)r_0(B)w_0(B)c_0r_1(A)r_1(B)c_1$, and its conflict graph is acyclic with $conflict(DT(s')) = \{(w_0(A), r_1(A)), (w_0(B), r_1(B))\}$, so the schedule now is conflict serializable.:



7.1.3

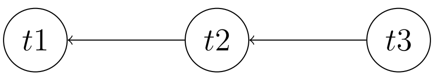
If we use locks without 2PL, we got the schedule s'':

t_0	t_1
$wl_0(A)$	
$r_0(A)$	
$w_0(A)$	
$wu_0(A)$	
	$wl_1(A)$
	$r_1(A)$
	$wu_1(A)$
	$wl_1(B)$
	$r_1(B)$
	$wu_1(B)$
	c_1
$wl_0(B)$	
$r_0(B)$	
$w_0(B)$	
$wu_0(B)$	
c_0	

where $DT(s'') = r_0(A)w_0(A)r_1(A)r_1(B)c_1r_0(B)w_0(B)c_0$, and its conflict graph is cyclic with $conflict(DT(s'')) = \{(w_0(A), r_1(A)), (r_1(B), w_0(B))\}$. So the lock leads to a not conflict serializable schedule.

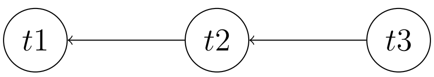
7.1.4

$s_1 = r_1(z)r_2(x)w_1(x)r_3(y)w_3(y)r_2(z)w_2(y)w_1(z)c_1c_2c_3$



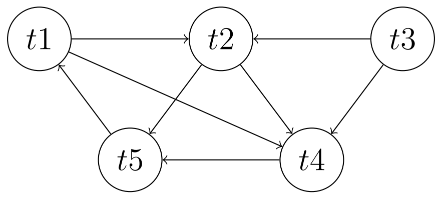
The conflict graph is acyclic, so $s_1 \in CSR$.
There is no non-overlapped transactions in s_1, so $s_1 \in OCSR$.
Commits in s_1 is $c_1c_2c_3$, not in the "conflict order" $t_3t_2t_1$, so $s_1 \notin CO$.

$s_2 = r_3(y)w_3(y)r_2(x)r_2(z)w_2(y)r_1(z)w_1(x)w_1(z)c_3c_2c_1$



The conflict graph is acyclic, so $s_2 \in CSR$.
Commits in s_2 is $c_3c_2c_1$, in the "conflict order" $t_3t_2t_1$, so $s_1 \in CO$, and also $s_1 \in OCSR$, through $OSCR \subset CO$.

$s_3 = r_1(z)r_3(z)w_3(x)w_2(z)c_3r_4(x)w_4(z)c_2r_5(z)c_4w_5(y)w_1(y)c_1c_5$
$conf(s_3) = \{(r_1(z), w_4(z)), (r_1(z), w_2(z)), (r_3(z), w_4(z)), (r_3(z), w_2(z)), (w_2(z), w_4(z)),$
$(w_2(z), r_5(z)), (w_4(z), r_5(z)), (w_3(x), r_4(x)), (w_5(y), w_1(y))\}$



The conflict graph contains cycles, for example $t1 \to t2 \to t5 \to t1$, so $s_3 \notin CSR$, as well as $s_3 \notin OCSR$, $s_3 \notin OC$.

$s_4 = r_1(z)r_3(z)w_3(x)w_2(z)r_4(x)c_2w_4(z)c_4r_5(z)c_3w_5(y)c_5w_1(y)c_1$
The order of actions except for commits in s_4 is same with this in s_3, so they have same conflict graph. Thus, $s_4 \notin CSR$, as well as $s_4 \notin OCSR$, $s_4 \notin OC$.

Exercise 7.2(Recovery)

7.2.1

1. find most recent starting point at LSN 4, since we start the checkpoint there
2. initialize the transaction table and dirty page read table as empty tables

LSN 5 : Update the tables with the operations until the checkpoint

LSN 6: update (T3,6,active) in the transaction table

LSN 7: update (T2,7,active) in the transaction table

LSN 8: update (T2,8,commit) in the transaction table

⇒:

TRANSACTION_ID	LAST_LSN	STATUS
T3	6	active
T2	8	commit

PAGE_ID	LSN
C	1
B	2

7.2.2

The REDO phase repeats all committed and active transactions from the first possible starting point (LSN 1) to the most recent one (LSN 8).

LSN 1: redo change to C

LSN 2: redo change to B

LSN 6: redo change to A

LSN 7: redo change to C

7.2.3

The UNDO phase identifies all transactions that were active (i.e. T3) at the crash and undoes the operations it has done in reverse order they were executed:

LSN 6: undo update of A from T3

Exercise 7.3(B+-tree Locking)

7.3.1

Search 52:
$rl(A)$
$r(A)$
$rl(C)$
$ru(A)$
$r(C)$
$rl(G)$
$ru(C)$
$r(G)$ $\leftarrow$read 52
$ru(G)$

7.3.2

Insert 19:
$wl(A)$
$r(A)$
$wl(B)$
$r(B)$
$wu(A)$ $\leftarrow$ because B is not full.
$wl(E)$
$r(E)$
$wu(B)$ $\leftarrow$ because E is not full.
$w(E)$ $\leftarrow$ insert 19
$wu(E)$

7.3.3

Delete 30:
$wl(A)$
$r(A)$
$wl(C)$
$r(C)$ $\leftarrow$ C is half-empty, so cannot unlock acestor.
$wl(F)$
$r(F)$ $\leftarrow$ F is half-empty, so cannot unlock acestors.
$w(F)$ $\leftarrow$delete 30, and needs to merge with its sibling
$wl(G)$

$r(G)$
$create(M_1) \leftarrow$ merge F and G
$delete(F)$
$delete(G)$
$w(C) \leftarrow$ delete the ”split key” 44, then need to merge with its sibling
$wl(B)$
$r(B)$
$create(M_2) \leftarrow$ merge C and B, incorporate the ”split key” in A
$delete(B)$
$delete(C)$
$w(A) \leftarrow$ delete the ”split key” 23, then the root is empty
$delete(A)$
$wu(M_2) \leftarrow$ the new root
$wu(M_1)$