UNIVERSITÄT DES SAARLANDES

Fachrichtung 6.2 – Informatik

Lehrstuhl Prof. Weikum

Transaktionsorientierte Informationssysteme SS 2000

Übung 9 — Beispiellösung

Aufgabe 1:

Consider the the action history given in the following table, including checkpoints and flush actions. Assume that there is a system crash right after the last action. Determine the necessary logging actions during normal operation and the recovery actions during restart, by completing the table. First consider the case where heavy-weight checkpoints are used and flush actions are not logged, then consider light-weight checkpoints, and finally discuss the additional effect of keeping log entries for flush actions. In all cases, assume that the redo-history paradigm is employed.

Heay-weight checkpoints:

Sequence number: action	Change of cached database [PageNo: SeqNo]	Change of stable database [PageNo: SeqNo]	Log entry added to log buffer [LogSeqNo: action]	Log entries added to stable log [LogSeqNo's]
1:begin(t1)			1: begin(t1)	
2: write(p,t1)	p:2		2: write(p,t1)	
3: write(q,t1)	q:3		3: write(q,t1)	
4: commit(t1)			4: commit(t1)	1,2,3,4
5: flush(p)		p:2		
6: begin(t2)			6: begin(t2)	
7: write(p,t2)	p:7		7: write(p,t2)	
8: write(r,t2)	r:8		8: write(r,t2)	
9: checkpoint		p:7,q:3,r:8	9: CP({t2})	6,7,8,9
10: commit(t2)			10: commit(t2)	10
11: begin(t3)			11: begin(t3)	
12: flush(p)				11
13: write(p,t3)	p:13		13: write(p,t3)	
14: write(q,t3)	q:14		14: write(q,t3)	
15: flush(q)		q:14		13,14
16: write(r,t3)	r:16		16: write(r,t3)	
Crash				
Analysephase: loser{t3}				
redo(13)	p:13			
consider_redo(14)	q:14			
16: compensate(14)	q:16		17: compensate(14)	
17: compensate(13)	p:17		18: compensate(13)	
18: rollback(t3)			18:rollback(t3)	(16,17,18)

Light-weight checkpoints:

Sequence number: action	Change of cached database [PageNo: SeqNo]	Change of stable database [PageNo: SeqNo]	Log entry added to log buffer [LogSeqNo: action]	Log entries added to stable log [LogSeqNo's]
1:begin(t1)			1: begin(t1)	
2: write(p,t1)	p:2		2: write(p,t1)	
3: write(q,t1)	q:3		3: write(q,t1)	
4: commit(t1)			4: commit(t1)	1,2,3,4
5: flush(p)		p:2		
6: begin(t2)			6: begin(t2)	
7: write(p,t2)	p:7		7: write(p,t2)	
8: write(r,t2)	r:8		8: write(r,t2)	
9: checkpoint			9: CP({p:7,q:3,r:8},{t2}	6,7,8,9
10: commit(t2)			10: commit(t2)	10
11: begin(t3)			11: begin(t3)	
12: flush(p)		p:7		11
13: write(p,t3)	p:13		13: write(p,t3)	
14: write(q,t3)	q:14		14: write(q,t3)	
15: flush(q)		q:14		13,14
16: write(r,t3)	r:16		16: write(r,t3)	
Crash				
Analysephase: loser{t3}, DP={p:7,q:3,r:8}				
consider_redo(3)	q:14			
consider_redo(7)	p:7			
redo(8)	r:8			
redo(13)	p:13			
skip_redo(14)				
16: compensate(14)	q:16		17: compensate(14)	
17: compensate(13)	p:17		18: compensate(13)	
18: rollback(t3)			18:rollback(t3)	(16,17,18)

flush logentries with light-weight checkpoints:

Sequence number: action	Change of cached database [PageNo: SeqNo]	Change of stable database [PageNo: SeqNo]	Log entry added to log buffer [LogSeqNo: action]	Log entries added to stable log [LogSeqNo's]
1:begin(t1)			1: begin(t1)	
2: write(p,t1)	p:2		2: write(p,t1)	
3: write(q,t1)	q:3		3: write(q,t1)	
4: commit(t1)			4: commit(t1)	1,2,3,4
5: flush(p)		p:2	5:flush(p)	5
6: begin(t2)			6: begin(t2)	
7: write(p,t2)	p:7		7: write(p,t2)	
8: write(r,t2)	r:8		8: write(r,t2)	
9: checkpoint			9: CP({p:7,q:3,r:8},{t2}	6,7,8,9
10: commit(t2)			10: commit(t2)	10
11: begin(t3)			11: begin(t3)	
12: flush(p)		p:7	12:flush(p)	11,12
13: write(p,t3)	p:13		13: write(p,t3)	
14: write(q,t3)	q:14		14: write(q,t3)	
15: flush(q)		q:14	15: flush(q)	13,14,15
16: write(r,t3)	r:16		16: write(r,t3)	
Crash				
Analysephase: loser{t3}, DP={p:13,r:8}				
redo(8)	r:8			
redo(13)	p:13			
skip_redo(14)				
16: compensate(14)	q:16		17: compensate(14)	
17: compensate(13)	p:17		18: compensate(13)	
18: rollback(t3)			18:rollback(t3)	(16,17,18)

Now consider the extended scenario given in the table below. In contrast to the previous scenario, this action history contains two transaction rollbacks during normal operation one of which is completed before the crash whereas the second one is interrupted by the crash. Determine again the necessary logging and restart actions for this scenario by completing the table below.

Sequence number: action	Change of cached db [PNo: SeqNo]	Change of stable db [PNo: SeqNo]	LE added to log buffer [LSN: action]	LEs added to stable log [LSN's]
1:begin(t1)			1: begin(t1)	
2: write(p,t1)	p:2		2: write(p,t1)	
3: write(q,t1)	q:3		3: write(q,t1)	
4: commit(t1)			4: commit(t1)	1,2,3,4
5: flush(p)		p:2		
6: begin(t2)			6: begin(t2)	
7: write(p,t2)	p:7		7: write(p,t2)	
8: write(r,t2)	r:8		8: write(r,t2)	
9: checkpoint			CP({p:7,q:3,r:8},{t2})	6,7,8,9
10: abort(t2)				
11: compensate(8:write(r,t2))	r:11		11: compensate(8), next=7	
12: compensate(7:write(p,t2))	p:12		12: compensate(7), next=nil	
13: rollback(t2) complete			13: rollback(t2)	
14: begin(t3)			14: begin(t3)	
15: flush(p)		p:12		11,12,13,14
16: write(p,t3)	p:16		16: write(p,t3)	
17: write(q,t3)	q:17		17: write(q,t3)	
18: flush(q)		q:17		16,17,18
19: write(r,t3)	r:19		write(r,t3)	
20: abort(t3)				
21: compensate(19:write(r,t3))	r:21		21: compensate(19), next=17	
22: begin(t4)			22: begin(t4)	
23: compensate(17:write(q,t3))	q:23		23: compensate(17), next=16	
24: write(s,t4)	s:24		24: write(s,t4)	
25: flush(q)				19,21,22,23,24
26: commit(t4)			26: commit(t4)	26
Crash				
Analysephase: loser={t3}, DP={p:7,r:8,q:3}				
consider_redo(3)	q:17			
consider_redo(7)	p:12			
redo(8)	r:8			
redo(11)	r:11			
skip_redo(12)				
redo(16)	p:16			
skip_redo(17)				
redo(19)	r:19			
redo(21)	r:21			
redo(23)	q:23			
redo(24)	s:24			
27: compensate(16:write(p,t3)	p:27		27: compensate(16),next=nil	
28: rollback(t3) complete			28: rollback(t3)	(27,28)

Aufgabe 2:

Consider the following specialization of the redo-winners algorithm. Assume that all updates are logged as full-writes, so that log entries have the form of page before-images or after-images. Further assume that the database cache is large enough to have the “no-steal” property, which guarantees that a page that has been modified by an active (i.e., incomplete) transaction is never

flushed. In addition, assume that the log buffer is also large enough to contain all before-images of all active transactions. Design a special recovery algorithm under this premise that (i) should be simpler than the general-purpose redo-winners algorithm and (ii) aims to shorten the restart duration as much as possible. In particular address the following issues: Are checkpoints still needed at all (under the no-steal premise)? How can you handle transaction aborts? Is there a way of gracefully degrading the special algorithm if the no-steal property cannot be guaranteed but the database cache is almost always large enough to avoid flushing pages whose last modification belongs to an active transaction?
Hint: The DB Cache algorithm of Bayer and Elhardt (1984) is such an algorithm (K. Elhardt and R. Bayer (1984): A Database Cache for High Performance and Fast Restart in Database Systems; ACM Transactions on Database Systems 9, 503–525.).

Aufgabe 3:

Reconsider the redo pass of the redo-history algorithm. Show that writes for a page that originate from loser transactions and are not followed by any winner writes on the same page do not need to be redone during the redo pass. Design a variant of the redo-history algorithm, both the redo and the undo pass, that avoids redoing writes in the above category, thus reducing the overall work of the redo pass. Note that this reduction is usually not a major gain as it affects only a small fraction of log entries; this variant is interesting in that it consolidates and deepens the understanding of the relationships between the redo and the undo pass.
Hint: The ARIES/RRH algorithm by Mohan and Pirahesh (1991) is such a variant of the redo-history algorithm (C. Mohan and H. Pirahesh (1991): ARIES-RRH: Restricted Repeating of History in the ARIES Transaction Recovery Method, Proceedings of the 7th International Conference on Data Engineering, Kobe, Japan, 1991.).